OGC Testbed-13

Application Deployment and Execution Service ER

Table of Contents

Publication Date: 2018-01-11

Approval Date: 2017-12-07

Posted Date: 2017-11-14

Reference number of this document: OGC 17-024

Reference URL for this document: http://www.opengis.net/doc/PER/t13-ES002

Category: Public Engineering Report

Editor: Pedro Gonc alves

Title: OGC Testbed-13: Application Deployment and Execution Service ER

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

The Testbed-13 Earth Observation Clouds (EOC) effort supports the development of ESA's Thematic Exploitation Platforms (TEP) by exercising envisioned workflows for data integration and processing that are deployed in multiple clouds. The Application Deployment & Execution Service OGC Engineering Report (ER) identifies the Application Programming Interface (API) for delivering all functionality provided to realize the testbed scenario.

This ER will list the requirements fulfilled by Cloud APIs in order to allow an automation of the application package deployment and execution workflow and capture implementation process experiences.

1.1. Requirements

This ER will document the automation of the application package deployment taking in consideration an application package defining:

- Identify the application to which it refers
- Describe the required execution environment for the application
- Identify how to deploy and launch the application
- Describe the input data collections
- Describe additional input parameters

1.2. Key Findings and Prior-After Comparison

The serialization and deployment of applications behind an OGC Web Processing Service (WPS) service was the subject of several discussions at OGC Standards Working Group (SWG) and Domain Working Group (DWG) in the past. A diverse number of issues were identified but the main barriers were due to the diversity of runtime environment to support and the respective technology maturity hampered its feasibility.

The advance in container technology together with the concept of federation of Clouds presents new opportunities to address this issue. Previously the constraints imposed in describing the runtime environments (e.g. Linux, Java VM, Perl, Python) and the respective application installation would create a considerable amount of information needed for the application package information model outside the OGC aegis. With containers most of these requirements are addressed at their respective level with configuration and software elements packaged into isolated elements.

This ER will capture implementation process experiences for Exploitation Platform (EP) Application Packages and their deployment in multiple clouds.

1.3. What does this ER mean for the Working Group and OGC in general

This ER is relevant to the WPS SWG because it lists the requirements fulfilled by Cloud APIs in order to allow WPS-supported automation of application package deployment and execution.

1.4. Document contributor contact points

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

Table 1. Contacts

Name	Organization
Pedro Gonçalves	Terradue
Peter Vretanos	CubeWerx
Patrick Jacques	Spacebel
Christophe Noël	Spacebel
Paulo Sacramento	Solenix

1.5. Future Work

Future OGC testbeds should look to develop draft WPS profiles for Cloud application deployment and execution. It may also be necessary to develop extensions to the WPS standard, for example using interfaces based on Representational State Transfer (REST) to support Cloud application deployment and execution.

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.

NOTE: Only normative standards are referenced here, e.g. OGC, ISO or other SDO standards. All other references are listed in the bibliography. Example:

- OGC 06-121r9, OGC® Web Services Common Standard, April 2010 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC 12-084r2, OGC® Context Atom Encoding Standard, January 2014 [https://portal.opengeospatial.org/files/?artifact_id=55183]
- OGC 14-065, OGC® WPS 2.0 Interface Standard, March 2015 [http://docs.opengeospatial.org/is/14-065/14-065.html]
- IANA, Link Relations registry [https://www.iana.org/assignments/link-relations/link-relations.xhtml]

Chapter 3. Terms and definitions

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

3.1. Area Of Interest

An Area of Interest is a geographic area that is significant to a user.

3.2. Context Document

A Context Document is a document describing the set of services and their configuration, and ancillary information (area of interest etc) that defines the information representation of a common operating picture.

3.3. Dockerfile

A text document that contains all the commands a user could call on the command line to assemble an image.

3.4. Container

A lightweight and configurable virtual machine with a simulated version of an operating system and its hardware, which allow software developers to share their computational environments.

3.5. Infrastructure as a Service (IaaS)

A standardized, highly automated offering, where compute resources, complemented by storage and networking capabilities are owned and hosted by a service provider and offered to customers on-demand. Customers are typically able to self-provision this infrastructure, using a Web-based graphical user interface that serves as an IT operations management console for the overall environment. API access to the infrastructure may also be offered as an option.

3.6. Resource

A resource is a configured set of information that is uniquely identifiable to a user. Can be realized as in-line content or by one or more configured web services.

3.7. Virtual Machine Image

A virtual machine image is a template for creating new instances. An image can offer plain operating systems or can have software installed on them, such as databases, application servers, or other applications.

3.8. Thematic Exploitation Platform (TEP)

A "Thematic Exploitation Platform" refers to an environment providing a user community interested in a common Theme with a set of services such as very fast access to (i) large volume of data (EO/non-space data), (ii) computing resources (iii) processing software (e.g. toolboxes, RTMs, retrieval baselines, visualization routines), and (iv) general platform capabilities (e.g. user management and access control, accounting, information portal, collaborative tools, social networks etc.). The platforms thus provide a complete work environment for its' users, enabling them to effectively perform data-intensive research and data exploitation by running dedicated processing software close to the data.

Chapter 4. Abbreviated terms

- ADES Application Deployment and Execution Service
- AMC Application Management Client
- API Application Programming Interface
- EOC Earth Observation Clouds
- IdP Identity Provider
- JSF JavaServer Faces
- PEP Policy Enforcement Point
- SAML Security Assertion Markup Language
- STS Security Token Service
- TEP Thematic Exploitation Platform
- WPS Web Processing Service

Chapter 5. Overview

This ER reports the implementation experiments for the deployment of applications built in the context of the European Space Agency (ESA) initiative to build an ecosystem of Thematic Exploitation Platforms (TEP). A TEP refers to a computing platform that follows a given set of scenarios for users, data and ICT (Information and Communications Technology) provision aggregated around an Earth Science thematic area. The TEPs define three canonical user scenarios covering the EO data exploitation (users discover and manipulate data), new EO service development (users deploy a new processor) and new EO product development (users publish new results). The TEPs implement these canonical scenarios and move the processing to the data, rather than the data to the users, thereby enabling ultra-fast data access and processing.

Algorithms are initially developed by a third-party developer and subsequently deployed in a TEP. An application package document that contains all the information necessary to allow applications (implementing said algorithms) to be specified and deployed in federated resource providers is one of the main outcomes of this effort.

The application package encoding conveys all the information needed for encapsulating user applications and enabling their automatic management (upload, deploy, run) in diverse platforms. The consensus reached in the Testbed-13 EOC thread was to use the OGC Web Services (OWS) Context document to convey the information model. This document is used as an input parameter to a dedicated WPS service that will create and deploy it in a new WPS instance. An alternative approach is also proposed that uses the WPS description document to directly embed the OWS Context Document in an existing WPS Transactional Service.

This ER focuses on both technical approaches, with two sections that describe the different applications and their respective deployment and execution processes. Both approaches follow a common scenario described below.

5.1. Scenario

The application package will allow the developer (e.g. Markus) to register an application that will be deployed in a given ICT provider. The application is first built and registered in a third-party Application Hub (e.g. DockerHub). The developer will then directly interact with the Application Management Client (AMC) that will register the application on the Application Deployment and Execution Service (ADES).

The sequence of actions is described in the following image.

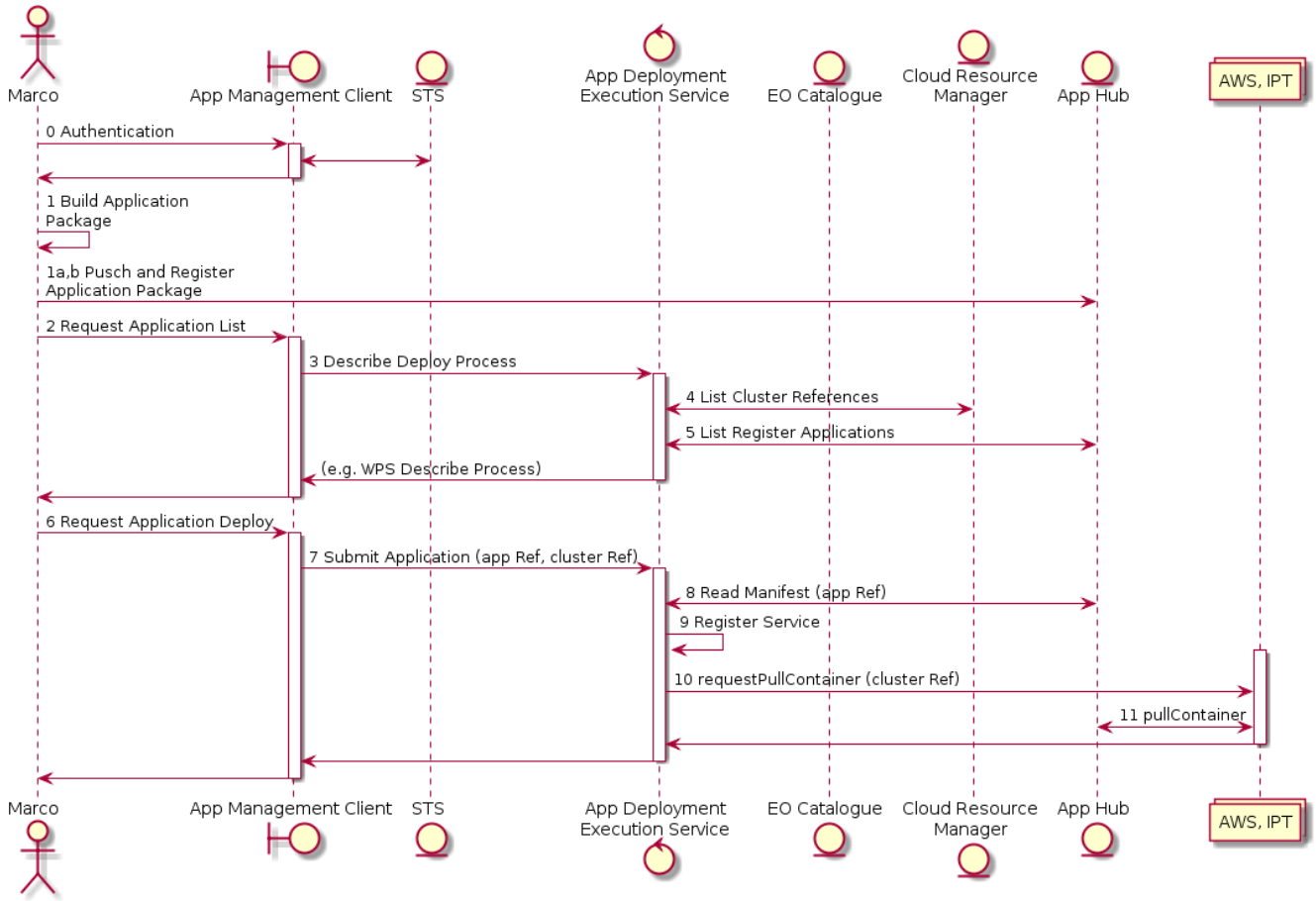

Figure 1. Sequence of Deploy and Register Steps

With the application registered, a user (e.g. Jim) can directly interact with the AMC that will allow the execution of the application on the ADES.

The sequence of actions for the execution is described in the following image.

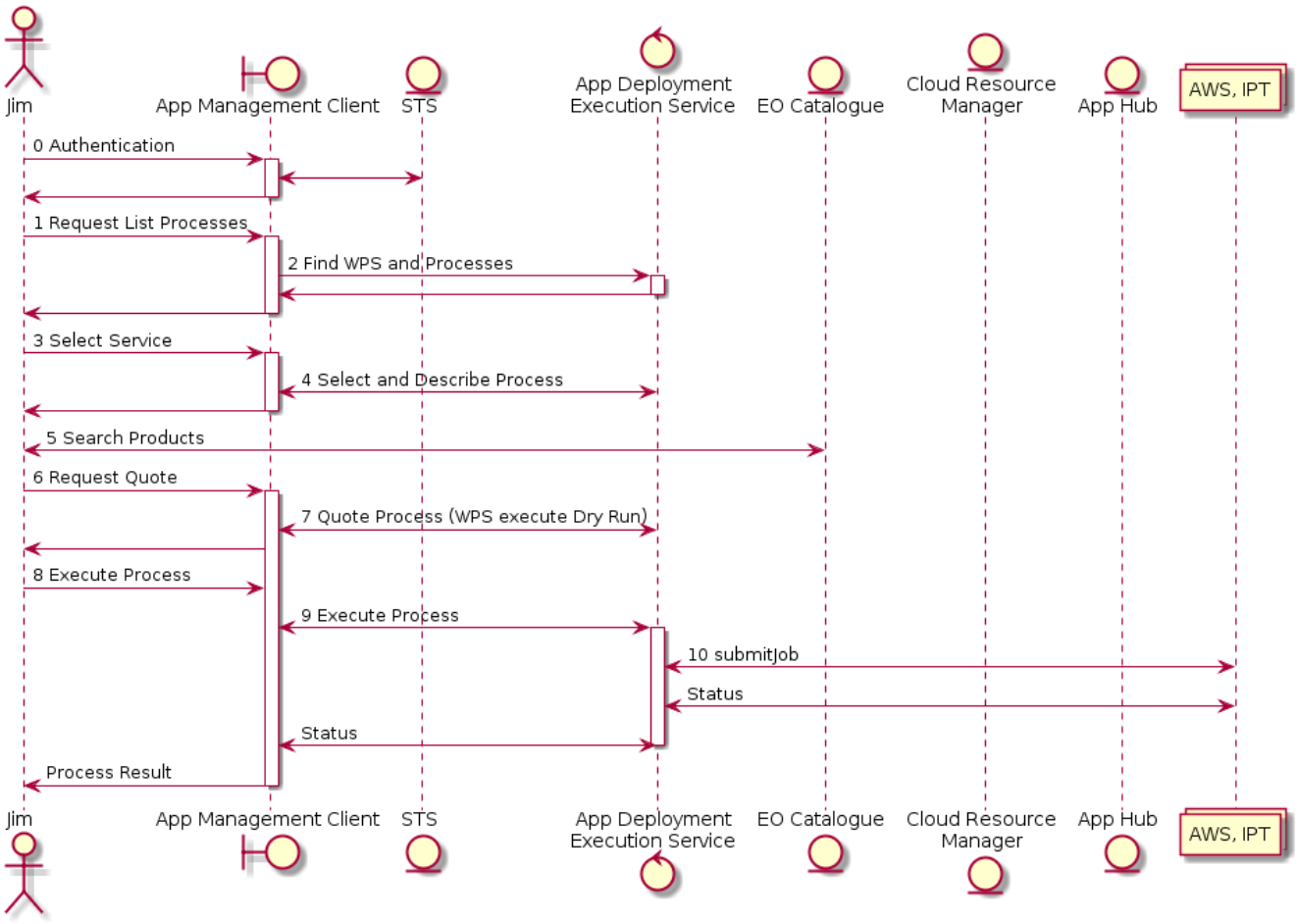

Figure 2. Sequence of Execution Steps

The overall components and respective flow is presented below.

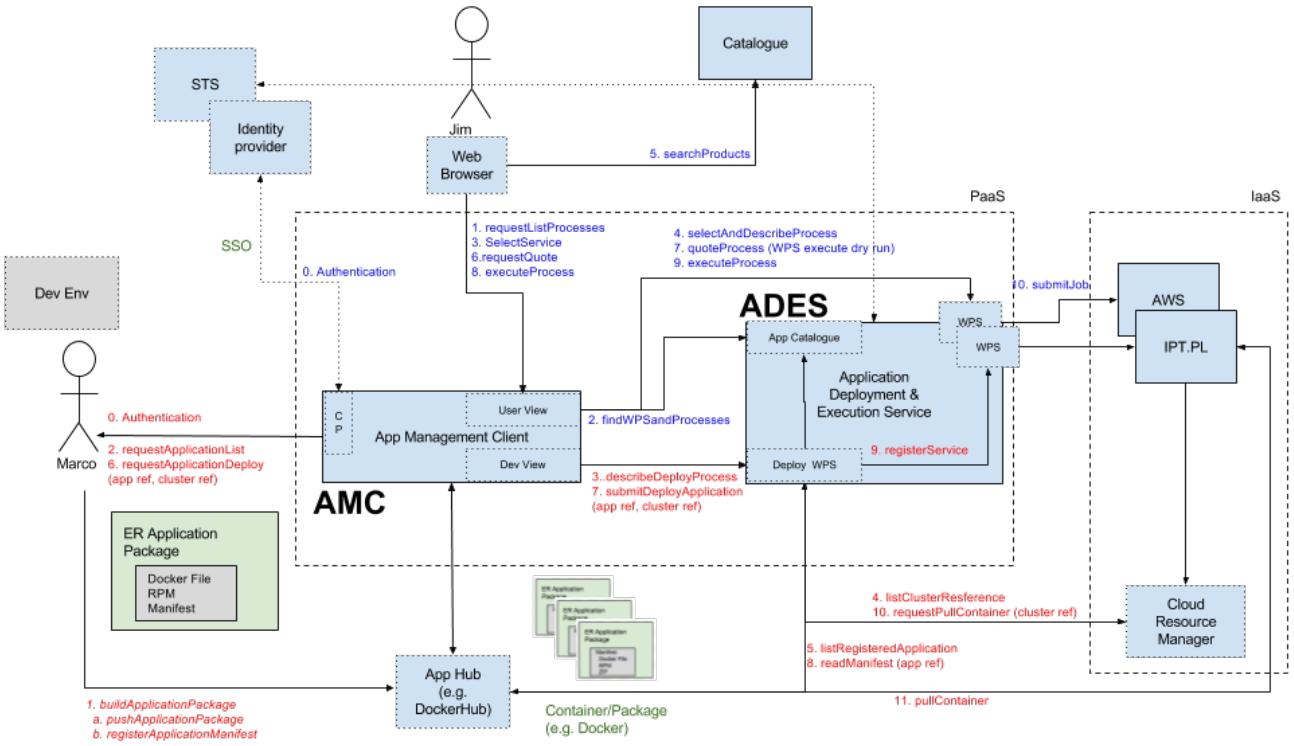

Figure 3. Overview of Deploy, Register and Execution Scenario

Chapter 6. Implementation

6.1. Overview

The Application Package targets applications built by third party developers and contains all the information necessary for their deployment in different infrastructures. It uses an ATOM OWS Context document as the document encoding and WPS as the execution interface. This chapter will address several implementation issues discussed by the EOC thread with particular focus on the approaches to application packaging, application execution and data access.

6.2. Application Package

The application package encoding conveys all the information needed for encapsulating user applications and enabling their automatic management (upload, deploy, run) in diverse platforms. It uses the OWS Context document encoding to convey the necessary information model as an input parameter to a dedicated WPS service that creates and deploys it in a new WPS instance. The requirements and respective encoding are described on the other Testbed-13 EOC ER (OGC 17-023) [2] dedicated to reporting of the Application Package

6.3. Data Fetching

The ADES provides a framework to deal with the data management tasks regarding the input file access and the registration of the output files. This is provided by two abstract functions that deal with the data staging and stage out that is executed by the application.

For data staging the convention is to map a file location from a URL (e.g. S3, FTP, Atom feed) to a local copy. The actual operation performed depends on the deployed infrastructure and data access mechanisms and it should be transparent for the application. It can consist of a simple string transformation based on previous knowledge about the data storage or to an actual data transfer mechanism that will provide the file data stream to the local processing.

```
staging https://iptpoland.pl/store/S1A_IW_RAW__0SDV_20170720···1D5B8_1722.zip
> /mounted/S1/S1A_IW_RAW__0SDV_20170720T091638_20170720T091710_017553_01D5B8_1722.zip
```

Equivalently, for the data stage out, the infrastructure will make available a specific convention. This operation will map a specific resulting file into the infrastructure publish mechanism guaranteeing the file safeguard and the subsequent access by the WPS client.

```
stageout FloodMap_20170504_max.tif
> https://store.terradue.com/production/097321986198678956-W/FloodMap_20170504_max.tif
```

6.3.1. Data Discovery

The application clients are expected to access EO catalogues using OpenSearch with Geo, Time and

EO extensions that allows standardized and harmonized access to metadata of satellite Earth observation data. This provides the necessary information for discovery (i.e. search) and retrieval (i.e. access) of EO data in the different providers. A major issue detected was the need to register each collection multiple times for each data provider. As such, each collection needed to be identified not only by the type of data but also the origin of the storage. For example, Sentinel-2 in the IPT provider was identified in the FedEO catalogue explicitly as *EOP:IPT:Sentinel2*, whereas the same data in Amazon, was identified as *EOP:SENTINEL-HUB:Sentinel2*. This implies that the same collection needs to have two different OpenSearch endpoints. When directing the processing to the resources in IPT Poland it is thus necessary to also provide a specific OpenSearch Description as:

```
http://geo.spacebel.be/opensearch/request?httpAccept=application/atom%2Bxml&parentIden
tifier=EOP:IPT:Sentinel2
```

While in processing in Amazon, the same collection is discovered using

```
http://geo.spacebel.be/opensearch/request/?httpAccept=application/atom%2Bxml&parentIde
ntifier=EOP:SENTINEL-HUB:Sentinel2
```

To overcome this issue, the thread analyzed the possibility to define an optional OpenSearch parameter that would make the intended download location a query item of the search. This parameter defined in the EO extension with the template:

```
    ...&from={eop:accessedFrom}
```

The parameter value would be a string identifying the location from which the resource will be accessed. Catalogue services that support this extension would then be able to return the download location in the enclosure atom link according to the requested processing location access service or, if possible, the real data address. For example, when processing in IPT Poland, one would add the parameter:

```
    ...&from=eocloud.eu
```

And would obtain a file directly the corresponding access URL:

```
    file:///eodata/Sentinel-
2/MSI/L1C/2017/11/03/S2B_MSIL1C_20171102T235229_N0206_R130_T57KUQ_20171103T005600.SAFE
```

If processed in AMAZON, the parameter would have the value:

```
    ...&from=amazonws
```

and would obtain the value in the s3 format such as:

```
s3://sentinel-s2-l1c/tiles/23/M/QM/2017/11/13/0
```

When the client does not provide an indication of the intended access, catalogue should return their default location:

```
    http://scihub.copernicus.eu/apihub/Products('0f1ed8d7-eb6e-51aa-a617-
7f632e8df960')/$value
```

Catalogue services are also recommended to provide the list of suggested parameters values in the OpenSearch Description Document. As such, catalogues should return a Parameter element as the example below:

```
<param:Parameter name="from" value="{eop:accessedFrom?}" minimum="0"
    title="A string identifying the location from where the download will be
initiated. The response includes the best download location in the enclosure atom link
(rel='enclosure') taking in consideration the value of the parameter." >
      <param:Option value="eocloud.eu" label="IPT Poland Cloud" />
      <param:Option value="amazonws" label="Amazon Cloud" />
      <param:Option value="PSNC" label=" EVER-EST VRE" />
</param:Parameter>
```

Terradue did an experimental implementation of this issue and the findings will be proposed as Change Request for the 13-026r8 Document to be discussed on the EO Product Metadata and OpenSearch SWG.

6.4. Application Parameters

The ADES provides a framework to deal with correct assignment of the application parameters. This is provided by one abstract function that provides the application a method for accessing the parameters values. For that the application uses a function for accessing the request parameters that from the command line would behave like:

```
$ getparam AreaOfInterest
> POLYGON(78427,321478321786, 391267841236 …. )
$ getparam inputcat
>
https://catalog.terradue.com//sentinel1/search?format=atom&uid=S1A_IW_RAW__0SDV_201707
20T091638_20170720T091710_017553_01D5B8_1722
```

An alternative solution using the environmental variable approach is also described in the next chapter.

6.5. Application Management Client

6.5.1. Solenix implementation

Introduction

This clause describes the implementation of the Solenix AMC. The section refers to fictitious users Marco and Jim.

The Solenix AMC provides two main GUIs, a Management GUI (e.g. for Marco) and a User GUI (e.g. for Jim).

The Management GUI implements three basic functionalities:

- Browse Applications: that Marco can use to get a list of applications previously registered in one of the available ADES implementations
- Register New Application: that Marco can use to upload an Application Package in OWS Context format for registration in one of the available ADES implementations
- Unregister Application: that Marco can use to unregister an application previously registered in one of the available ADES implementations

The User GUI instead implements the following functionality:

- Execution Listing: that Jim can use to list previously launched application executions, see their statuses and results
- Application Selection: that Jim can use to select one of the applications previously registered in one of the available ADES implementations. This triggers the dynamic and automated generation of user input fields
- Application Usage: highly dependent on the specific application, it allows Jim to provide all the inputs expected by the application, if necessary allowing him to perform catalogue searches in order to fill-in some of the inputs

Main GUI Screens

The Figure below shows the main screen of the Solenix AMC. From it, the user (Marco or Jim) can choose the Management (for Marco) or User GUI (for Jim):

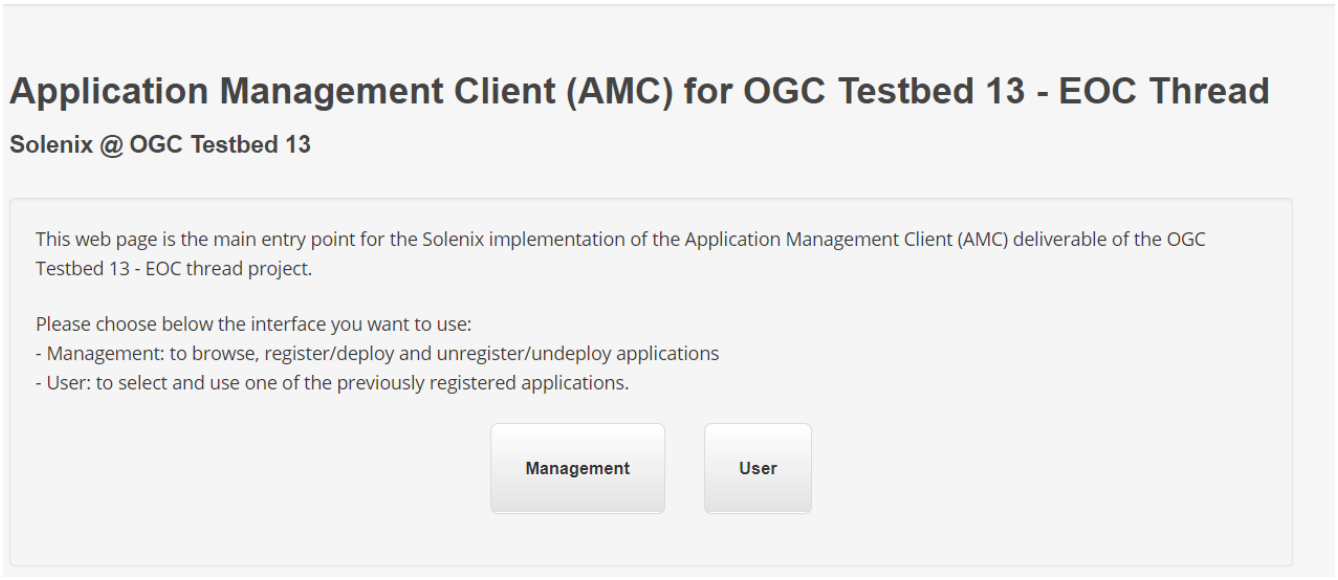

Figure 4. Main screen of Solenix AMC

Management GUI

Once on the Management GUI, Marco can choose to browse registered applications, register a new application or unregister an application:

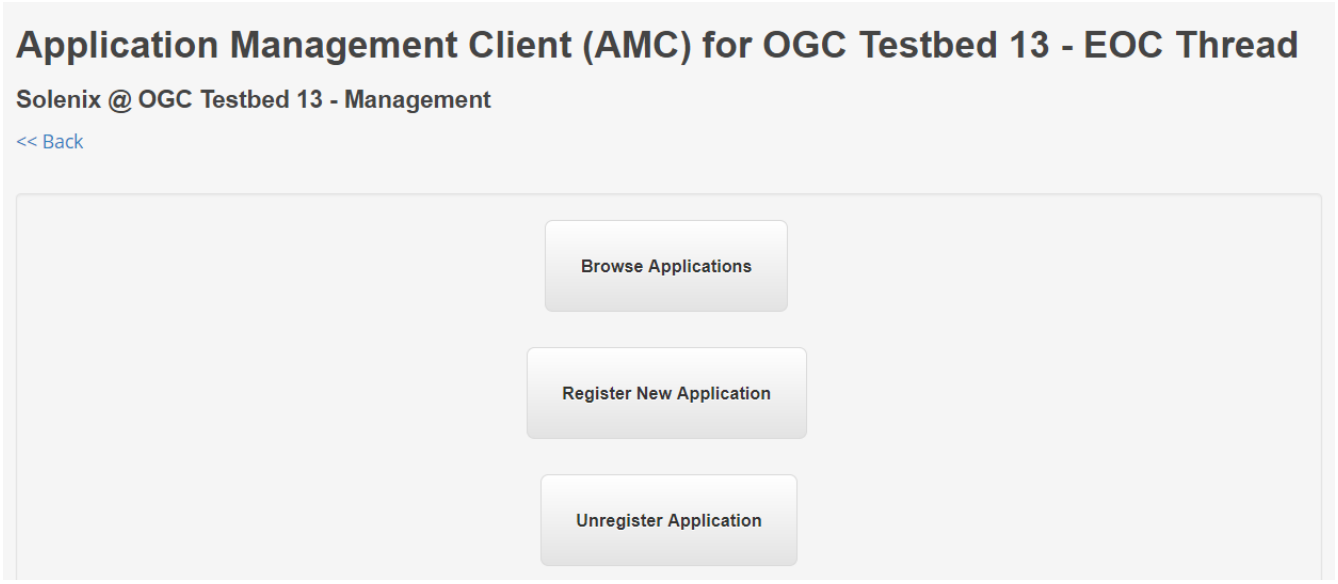

Figure 5. Solenix AMC Management GUI: Main screen

By clicking on 'Browse Applications', Marco is able to select the specific ADES implementation and list the applications registered in it. This triggers a WPS GetCapabilities on the selected ADES:

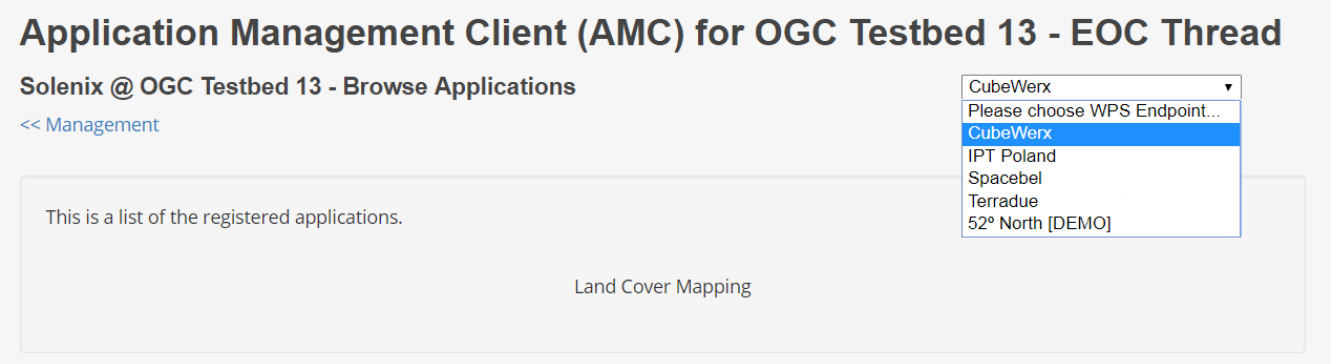

Figure 6. Solenix AMC Management GUI: Browse Applications

By clicking on 'Register New Application', and after choosing the desired server (an implementation of the ADES exposing a WPS endpoint), Marco will be able to upload an Application Package - an instance of OWS Context according to the format agreed in OGC Testbed-13 (see *Exploitation Platform Application Package Engineering Report* [2]):

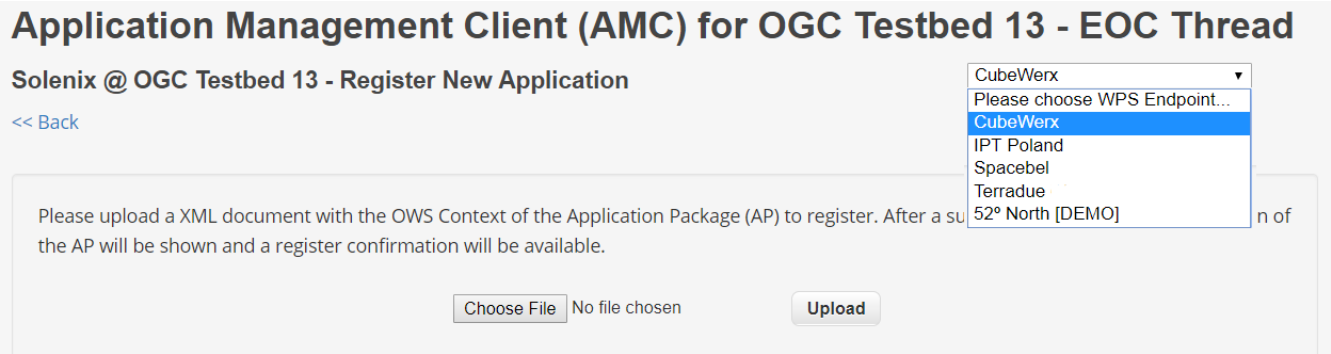

Figure 7. Solenix AMC Management GUI: Register New Application

Once this is done, some summary information about the AP will be shown and Marco will be able to confirm registration in the selected endpoint. This will trigger a WPS Execute operation on the DeployProcess process exposed by the selected ADES:

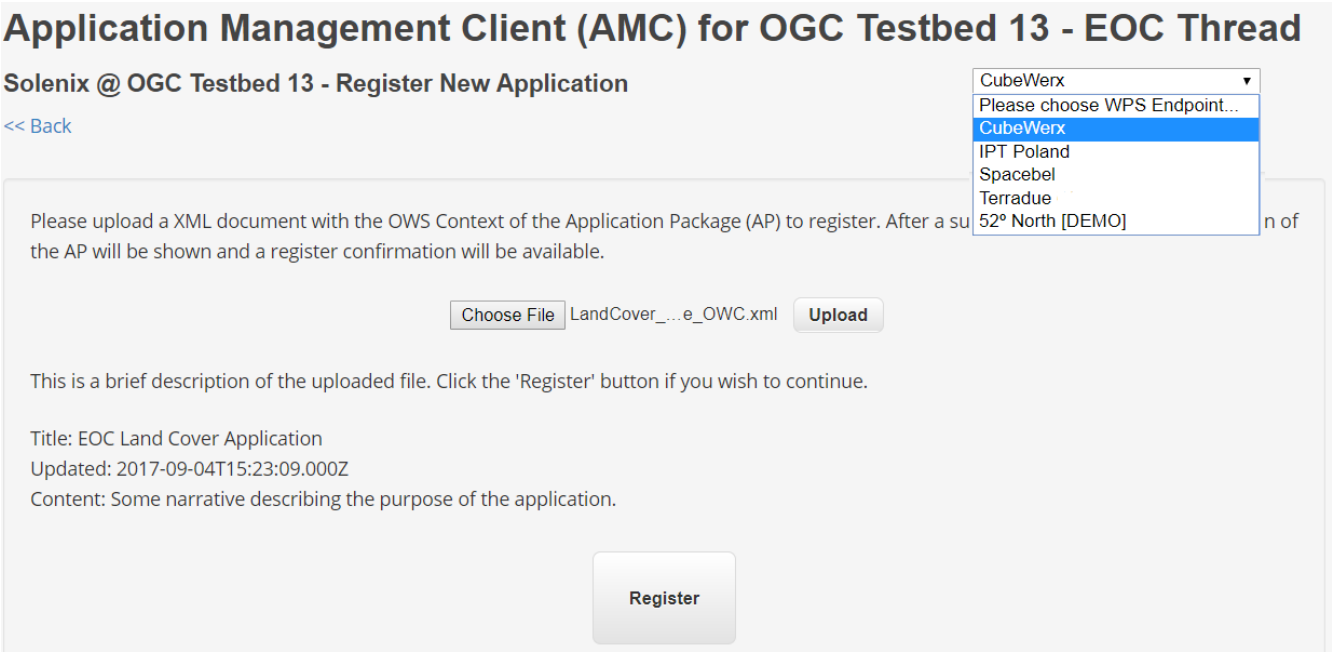

Figure 8. Solenix AMC Management GUI: Register New Application, After AP upload

User GUI

Coming back to the main screen and selecting the User GUI, Jim will see the following main screen, which allows choosing between a list of previous executions and issuing a new execution:

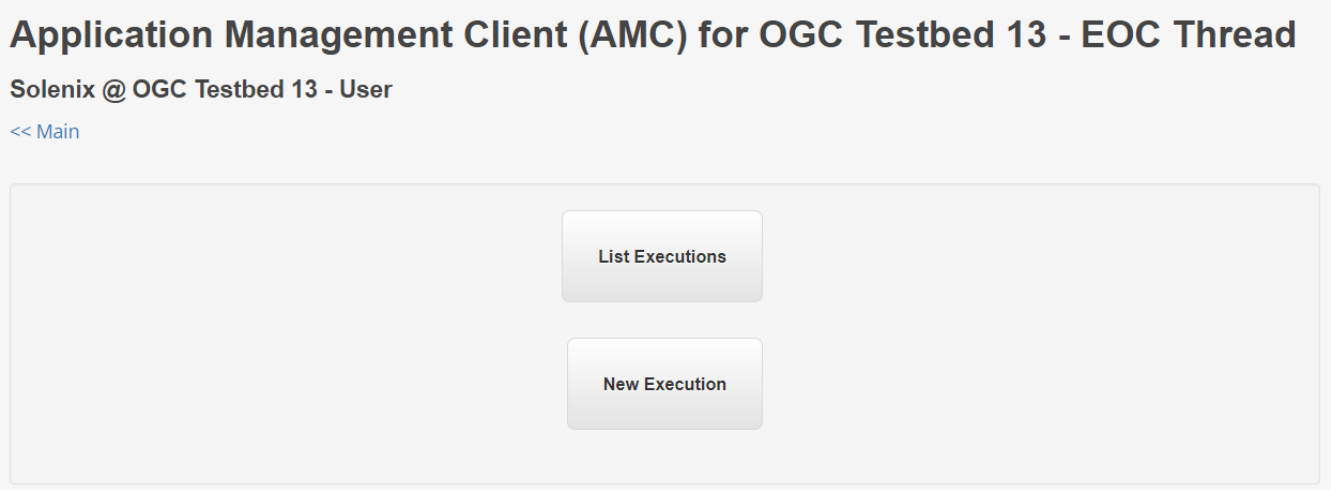

Figure 9. Solenix AMC User GUI: Main screen

The 'List Executions' page lists previously launched WPS Executions in any of the known ADES implementations:

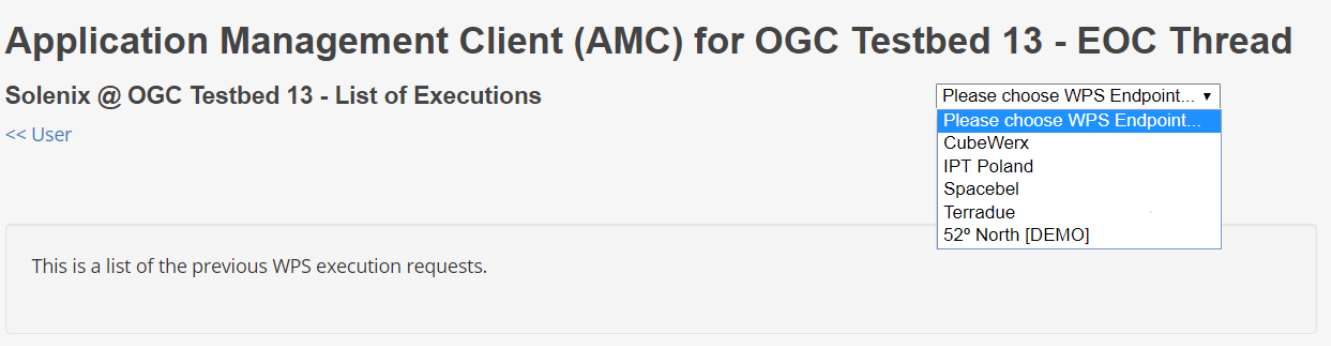

Figure 10. Solenix AMC User GUI: List of Executions

The 'New Execution' page allows launching new WPS requests on one of the known ADES implementations and previously registered applications - input fields per application are dynamically discovered and shown:

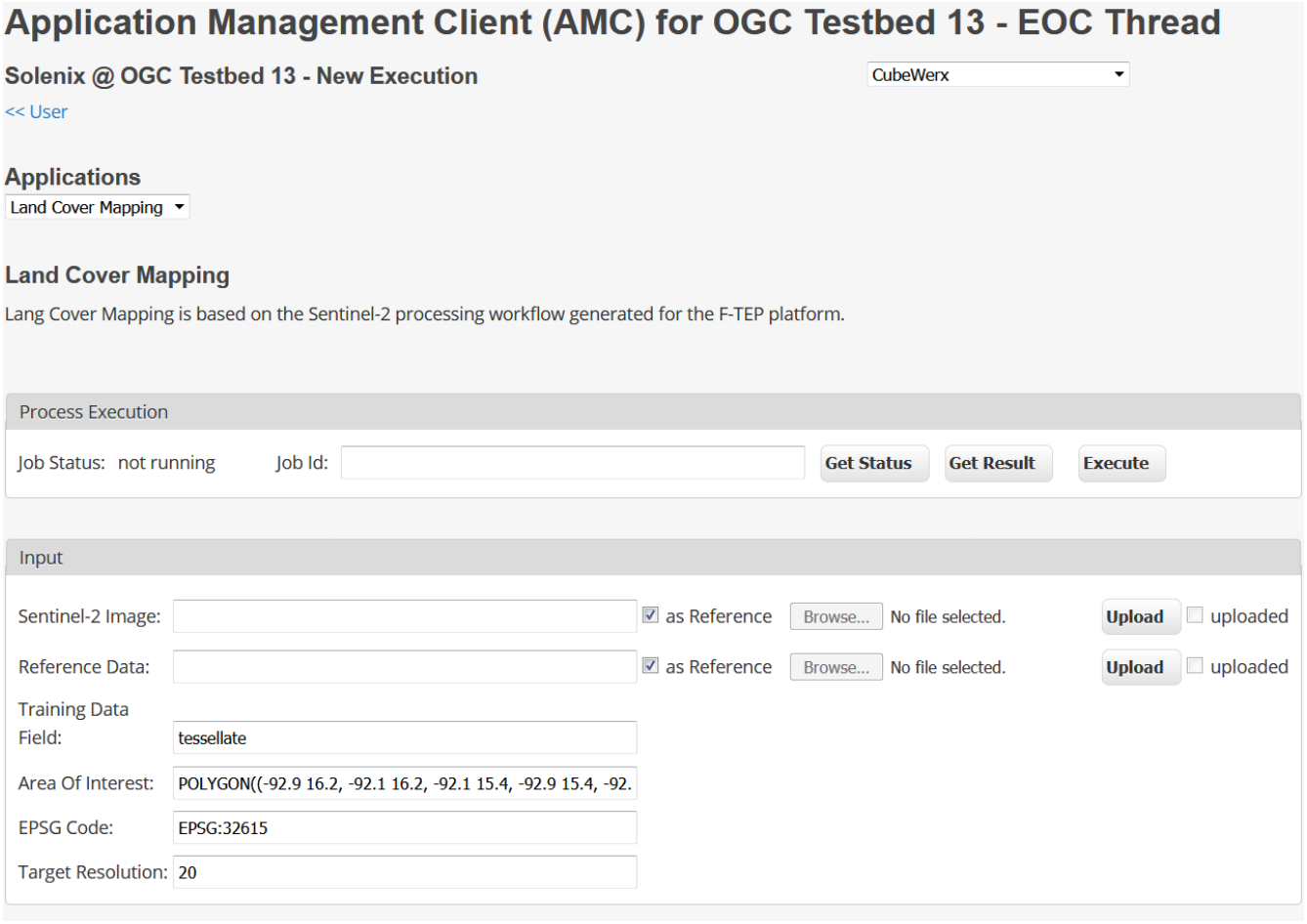

Figure 11. Solenix AMC User GUI: New Execution

6.6. Application Deployment and Execution Service

6.6.1. Introduction

This clause describes the two implementations of the ADES by CubeWerx and Terradue.

Both ADES implements the WPS standard (versions 1 & 2). The CubeWerx implementation is part of CubeSERV which is a component of the CubeWerx Suite while the Terradue implementation is part of the Terradue Cloud Platform.

6.6.2. Deploy

When initially deployed, both ADES offer two processes, DeployProcess and UndeployProcess. These processes, in turn, allow application developers to then add additional processes, each described by an application package (see OGC 17-023).

These methods are designed to emulate the DeployProcess and UndeployProcess requests defined in the Transactional WPS Extension (see OGC 13-071r1). The primary benefits of taking this approach, as opposed to implementing OGC 13-071r1, are:

1. Any version of the WPS standard can deploy these methods
2. Existing WPS clients are able to interact with these methods as they do with any other WPS processes.

The following tables, describe the parameters of the DeployProcess and UndeployProcess process offerings.

Table 2. Deploy Process Parameters

Parameter Name	O/M	In/Out	Description
applicationPackage	M	In	An ATOM-encoded OWS context document (see OGC 12-084r2) describing the application to be deployed
deployResult	M	Out	The format of the response to the operation

Table 3. Undeploy Process Parameters

Parameter Name	O/M	In/Out	Description
processIdentifier	M	In	The identifier of the process to undeploy
undeployResult	M	Out	The format of the response to the operation

NOTE Only processes added using the DeployProcess method may be undeployed.

NOTE The UndeployProcess implementation does not implement the keepExecutionUnit parameter from OGC 13-071.

NOTE At the moment, the ADES uses the application identifier specified in the application package. This means that there is the possibility of duplicate identifiers. The CubeWerx server checks for this and throws an exception if the specified identifier is a duplicate, while the Terradue implementation replaces the application. An alternative approach could be for the server to ignore the process identifier specified in the application package and simply generate a new, unique identifier.

The following XML fragment is generated from both implementations WPS server and describe the DeployProcess and UndeployProcess processes:

```
   <ProcessOffering processVersion="1.0.0">
      <Process xml:lang="en">
         <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc">
DeployProcess</ows:Identifier>
         <ows:Title>Deploy Process</ows:Title>
         <ows:Abstract>This method will deploy an application encapsulated within a
Docker container as a process accessible through the WPS interface.</ows:Abstract>
         <Input minOccurs="1">
            <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc"
>applicationPackage</ows:Identifier>
            <ows:Title>Application Package</ows:Title>
            <ows:Abstract>An application package, encoded as an ATOM-encoded OWS
context document, describing the details of the application.</ows:Abstract>
            <ComplexData>
               <Format mimeType="application/atom+xml" default="true"/>
            </ComplexData>
```

```
         </Input>
         <Output>
            <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc">
deployResult</ows:Identifier>
            <ows:Title>Deploy Result</ows:Title>
            <ows:Abstract>The server's response to deploying a process.  A successful
response will contain a summary of the deployed process.</ows:Abstract>
            <ComplexData>
               <Format mimeType="text/xml" default="true"/>
            </ComplexData>
         </Output>
      </Process>
   </ProcessOffering>
   <ProcessOffering processVersion="1.0.0">
      <Process xml:lang="en">
         <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc">
UndeployProcess</ows:Identifier>
         <ows:Title>Undeploy Process</ows:Title>
         <ows:Abstract>This method removes a previously deployed process from the
WPS.</ows:Abstract>
         <Input minOccurs="1">
            <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc"
>processIdentifier</ows:Identifier>
            <ows:Title>Process Identifier</ows:Title>
            <ows:Abstract>The identifier of the process to remove from the
WPS.</ows:Abstract>
            <LiteralData>
               <Format mimeType="text/plain" default="true"/>
               <LiteralDataDomain>
                  <AnyValue/>
                  <ows:DataType
                     ows:reference="http://www.w3.org/TR/xmlschema-2/#anyURI">
anyURI</ows:DataType>
               </LiteralDataDomain>
            </LiteralData>
         </Input>
         <Output>
            <ows:Identifier codeSpace="http://www.opengis.net/tb13/eoc">
undeployResult</ows:Identifier>
            <ows:Title>Undeploy Result</ows:Title>
            <ows:Abstract>This is the server's response when undeploying a process.  A
successful response will contain the identifier of the undeployed
process.</ows:Abstract>
            <ComplexData>
               <Format mimeType="text/xml" default="true"/>
            </ComplexData>
         </Output>
      </Process>
   </ProcessOffering>
```

The following XML schema fragments define the response elements for the DeployProcess and UndeployProcess methods:

```
   <element name="DeployResult" type="wps:DeployResultType">
      <annotation>
         <documentation>DeployProcess result.</documentation>
      </annotation>
   </element>
   <complexType name="DeployResultType">
      <sequence>
         <element name="DeploymentDone" type="boolean"/>
         <choice>
            <element name="ProcessSummary" type="wps:ProcessSummaryType"
                     minOccurs="0">
               <annotation>
                  <documentation>If DeploymentDone = true</documentation>
               </annotation>
            </element>
            <element name="FailureReason" type="xs:string">
               <annotation>
                  <documentation>If DeploymentDone = false</documentation>
               </annotation>
            </element>
         </choice>
      </sequence>
   </complexType>
   <element name="UndeployResult" type="wps:UndeployResultType">
      <annotation>
         <documentation>UndeployProcess result.</documentation>
      </annotation>
   </element>
   <complexType name="UndeployResultType">
      <sequence>
         <element name="UndeploymentDone" type="boolean"/>
         <choice>
            <sequence>
               <annotation>
                  <documentation>If UndeploymentDone = true</documentation>
               </annotation>
               <element ref="ows:Identifier" minOccurs="0">
                  <annotation>
                     <documentation>Identifier of the undeployed
process.</documentation>
                  </annotation>
               </element>
               <element ref="wps:Capabilities" minOccurs="0">
                  <annotation>
                     <documentation>Updated Capabilities document.</documentation>
                  </annotation>
               </element>
```

```
            </sequence>
            <element name="FailureReason" type="xs:string">
               <annotation>
                  <documentation>If UndeploymentDone = false</documentation>
               </annotation>
            </element>
         </choice>
      </sequence>
   </complexType>
```

These elements may be returned as bare elements or embedded within a WPS response document depending on the response form (i.e. response="raw" or response="document").

The Terradue implementation provides a graphical user interface allowing deployment directly from the web browser for authorized users.

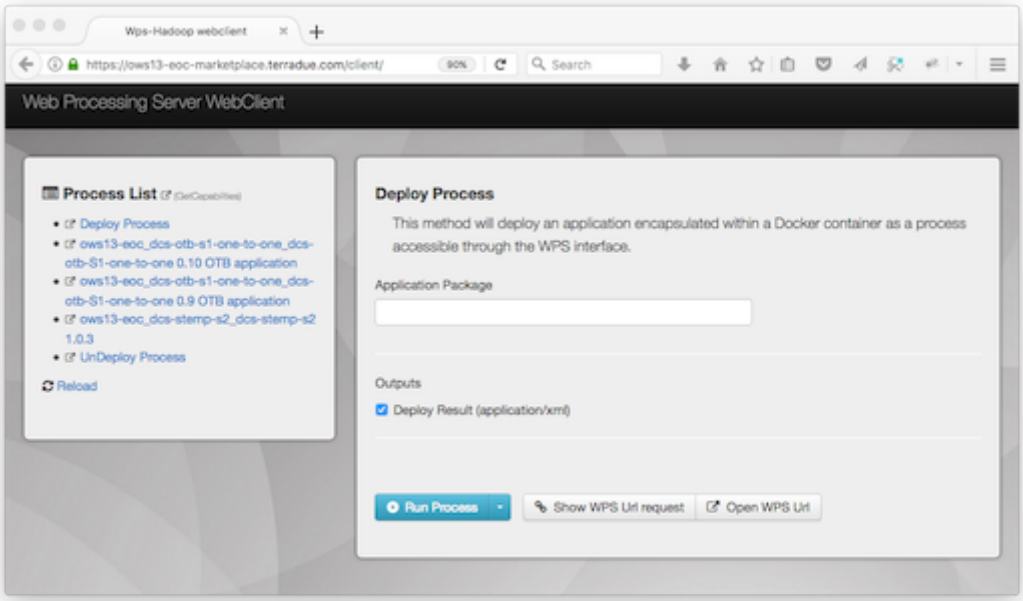

Figure 12. Terradue User GUI for Deployment

The CubeWerx implementation of the DeployResult and UndeployResult responses deviate slightly from the schemas defined on OGC 13-071r1. The CubeWerx server will never generate a "FailureReason" if process deployment or undeployment fails. Instead, the CubeWerx server will generate an ows:ExpceptionReport as defined in the OGC Web Service Common Implementation Specification (see OGC 06-121r9). It is unclear what the benefit of adding yet another communication channel (i.e. "FailureReason) for exceptions is when one is already defined in OGC 06-121r9. Furthermore, most OGC web service clients expect that a request will either succeed or generate an ows:ExceptionReport message.

It is understood that OGC 13-071r1 introduced the "FailureReason" element to allow exceptions in the back-end to be communicated to the calling process but, as the following example illustrates, this can easily be accomplished using the ows:ExceptionReport:

```
<?xml version="1.0" ?>
<ExceptionReport
   version="2.0.2"
   xmlns="http://www.opengis.net/ows/2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/ows/2.0
                       http://schemas.opengis.net/ows/2.0/owsAll.xsd">
   <Exception exceptionCode="InternalServerError"/>
   <Exception exceptionCode="VirtualMachineException">
      <ExceptionText>The OS of the Docker image is not compatible with the OS of the
VM allocated in the Cloud environment.</ExceptionText>
   </Exception>
</ExceptionReport>
```

In this example, the code "VirtualMachineException" is used to indicate that the exception occurred in the backend virtual machine.

6.6.3. Execute

Since deployed applications appear as WPS process offerings, executing a deployed application is simply a matter of using the wps:Execute operation. The following XML fragment is an example of such a request:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute
   xmlns:wps="http://www.opengis.net/wps/2.0"
   xmlns:ows="http://www.opengis.net/ows/2.0"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wps/2.0
                       http://www.pvretano.com/schemas/wps/2.0/wps.xsd"
   service="WPS"
   version="2.0.0"
   response="document"
   mode="async">
   <wps:Extension>
      <ows:ResponseHandler>mailto:pvretano_tb13@gmail.com</ows:ResponseHandler>
   </wps:Extension>
   <ows:Identifier>LandCover</ows:Identifier>
   <wps:Input id="Image">
      <wps:Data>
         <wps:LiteralValue dataType="string">/workspace/d7073c5e-825a-4257-b5e0-
62035220bc3d/stagedData</wps:LiteralValue>
      </wps:Data>
   </wps:Input>
   <wps:Input id="ReferenceData">
      <wps:Data>
         <wps:LiteralValue dataType="string"
>UEsDBBQAAAAIAGVTU0uEczQWqAIAAAsHAAAaABwARGVwbG95UHJvY2Vzc19EZXNjcmliZS54bWxV
```

```
VAkAAz226FlLTfdZdXgLAAEEHgUAAAT+AQAArVVRT9swEH7nV3h5HTQUXqaqDeoKTEigIsi0vU2u
c20Njm3ZF1L+/c5OS5PSbkxaX5L4znfffffddXixKhV7Aeel0aOk3ztNGGhhCqkXo+R7fn3yJbnI
joa19YN7ZwR4P53PwZH5iDH2ZGYTo9EZNbVIEfwo4f5VixNYgagQWCF9Kb1PgrOp0FaYO67jWUz4
...
ZENvdmVyX0FwcGxpY2F0aW9uUGFja2FnZV9XUFNULnhtbFVUBQADG2XzWXV4CwABBB4FAAAE/gEA
AFBLAQIeAxQAAAAIAIJyXkuW2aqtEQIAAN8FAAAVABgAAAAAAAEAAACwgQokAABMYW5kQ292ZXJf
RXhlY3V0ZS54bWxVVAUAA1Nt91l1eAsAAQQeBQAABP4BAABQSwUGAAAAAAgACAAjAwAAaiYAAAA
         </wps:LiteralValue>
      </wps:Data>
   </wps:Input>
   <wps:Input id="AreaOfInterest">
      <wps:Data>
         <wps:LiteralValue dataType="string">POLYGON((-92.9 16.2, -92.1 16.2, -92.1
15.4, -92.9 15.4, -92.9 16.2))</wps:LiteralValue>
      </wps:Data>
   </wps:Input>
   <wps:Input id="EPSGCode">
      <wps:Data>
         <wps:LiteralValue dataType="string">4326</wps:LiteralValue>
      </wps:Data>
   </wps:Input>
   <wps:Input id="TargetResolution">
      <wps:Data>
         <wps:LiteralValue dataType="integer">20</wps:LiteralValue>
      </wps:Data>
   </wps:Input>
   <wps:Output id="Image" mimeType="image/tiff" transmission="reference"/>
</wps:Execute>
```

The Terradue implementation provides a graphical user interface that allows execution directly from the web browser for authorized users.

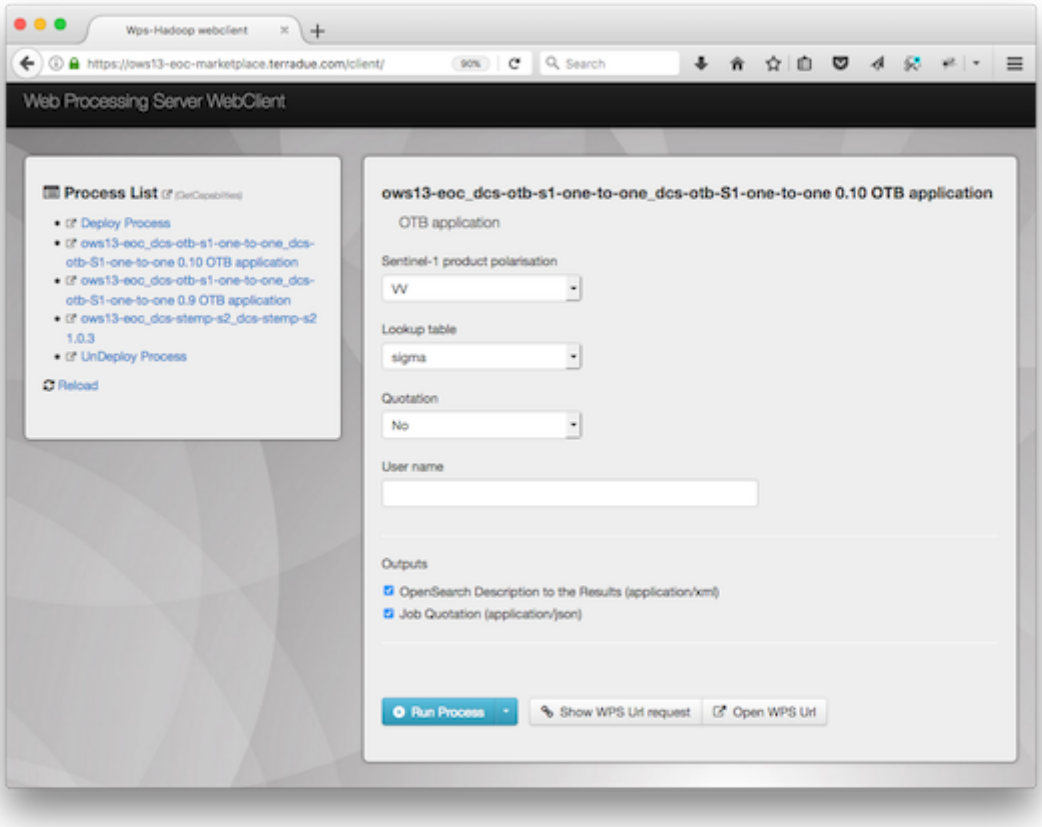

Figure 13. Terradue User GUI for Deployment

The normal WPS work flow includes using the GetStatus operation to poll the server to determine the execution status of a running job. The CubeWerx implementation offers this polling method of monitoring the process BUT it also includes a more efficient notification approach where the server notifies the client when the operation has completed processing. The notification is triggered by specifying a value for the vendor-specific responseHandler parameter on the wps:Execute operation. The value of the response handler can be something like "mailto:user@server.com" or "sms:4165555555". The details of the response handler parameter can be found in OGC 16-023r3, clause 7.2.

The example above makes use of WPS request extensibility by embedding the ows:ResponseHandler element(s) within the wps:Extension element. If one or more response handlers are specified on an execute request, then the value of the mode parameter shall be set to "async". If the mode is set to another value the server shall raise an exception (see OGC 06-121r9, 8.5).

The response to an asynchronous request shall be the standard wps:StatusInfo element (see OGC 14-065, 9.5) with the modifications shown in the following XML schema fragment:

```
<element name="StatusInfo">
   <complexType>
      <sequence>
         <element ref="wps:JobID"/>
         <element name="Status">
            <simpleType>
               <union>
                  <simpleType>
                     <restriction base="string">
                        <enumeration value="Succeeded">
                        <enumeration value="Failed">
                        <enumeration value="Accepted">
                        <enumeration value="Running">
                     </restriction>
                  </simpleType>
                  <simpleType>
                     <restriction base="string"/>
                  </simpleType>
               </union>
            </simpleType>
         </element>
         <element ref="wps:ExpirationDate" minOccurs="0"/>
         <element name="EstimatedCompletion" type="dateTime" minOccurs="0"/>
         <choice minOccurs="0">
            <element name="NextPoll" type="dateTime"/>
            <element ref="atom:link" maxOccurs="unbounded"/>
         </choice>
         <element name="PercentCompleted" minOccurs="0">
            <simpleType>
               <restriction base="integer">
                  <minInclusive value="0"/>
                  <maxInclusive value="100"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</element>
```

The modification allows zero or more hypermedia controls, in the form of ATOM links, to be included in the response. The value of the "rel" attribute informs the client what the link does and should be taken from the IANA Link Relations registry (see IANA) or defined with the OGC NA. An example of a common "rel" value is "monitor" which refers to a resource that can be used to monitor the status of a running job. The following XML fragment is an example of a wps:StatusInfo response that includes hypermedia controls:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:StatusInfo
   xmlns:wps="http://www.opengis.net/wps/2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wps/2.0 ../wps.xsd">
   <wps:JobID>1013</wps:JobID>
   <wps:Status>Accepted</wps:Status>
   <atom:link rel="monitor"
              href="http://www.someserver.com/jobs/1013"/>
   <atom:link rel="cancel"
              href="http://www.someserver.com/jobs/cancel/1013"/>
</wps:StatusInfo>
```

This hypermedia approach alleviates the client from having to understand how to form a monitor or cancel request; the client simply needs to understand the link relation and resolve the accompanying link.

6.6.4. Bindings

In addition to the OGC XML-POST and OGC KVP-GET bindings defined in the WPS 2.0 standard (see OGC 14-065, 10.1, 10.2), the CubeWerx WPS also implements a REST-XML binding. The following tables summarize each resource that the server offers and the HTTP methods that may be used on those resources.

Table 4. Rest XML Binding

Resource path	Method	Description	Parameters	Response
/	GET	Get a capabilities document	see OGC 16-121r9, Table 5[1]	body contains wps:Capabilities type[2]
	POST	*undefined*		
	PUT	*undefined*		
	DELETE	*undefined*		
/process	GET	wps:DescribeProcess app (all)	see OGC 14-065, Table 38[3]	body contains wps:ProcessOfferings type[4]
	POST	wps:DeployProcess op	body contains app package[5]	body contains wps:DeployResult response[6]
	PUT	*undefined*		
	DELETE	*undefined*		

Resource path	Method	Description	Parameters	Response
/process/{id}	GET	wps:DescribeProcess op (id)	see OGC 14-065, Table 51[7]	body contains wps:ProcessOffering type[8]
	POST	*undefined*		
	PUT	*undefined*		
	DELETE	wps:UndeployProcess op		body contains wps:UndeplyResult[9]
/job	GET	wps:GetStatus op (all)		list of jobs
	POST	wps:Execute op	body contains wps:Execute type	see OGC 14-065, Table 45
	PUT	*undefined*		
	DELETE	*undefined*		
/job/{jobId}	GET	wps:GetStatus op (jobId)		body contains wps:StatusInfo[10]
	POST	*undefined*		
	PUT	*undefined*		
	DELETE	wps:Dismiss op		body contains wps:StatusInfo[10]

NOTES:
1. exclude Service & Request parameters
2. see 14-065, 9.7.2 (wpsGetCapabilities.xsd)
3. exclude Identifier parameter
4. see wpsDescribeProcess.xsd
5. see ES001
6. see OGC 13-071, Clause 8.4
7. exclude service, version, request and identifier parameters
8. see wpsDescribeProcess.xsd
9. see OGC 13-071, Clause 9.4
10 see wpsCommon.xsd

The CubeWerx server also implemented the OPTIONS method (see RFC2616, clause 9) which allows clients to ascertain the methods and representations are support for each resource.

6.6.5. CubeWerx Backend

The following sequence diagram illustrates the sequence of operations performed by the WPS on the back end to execute a deployed application.

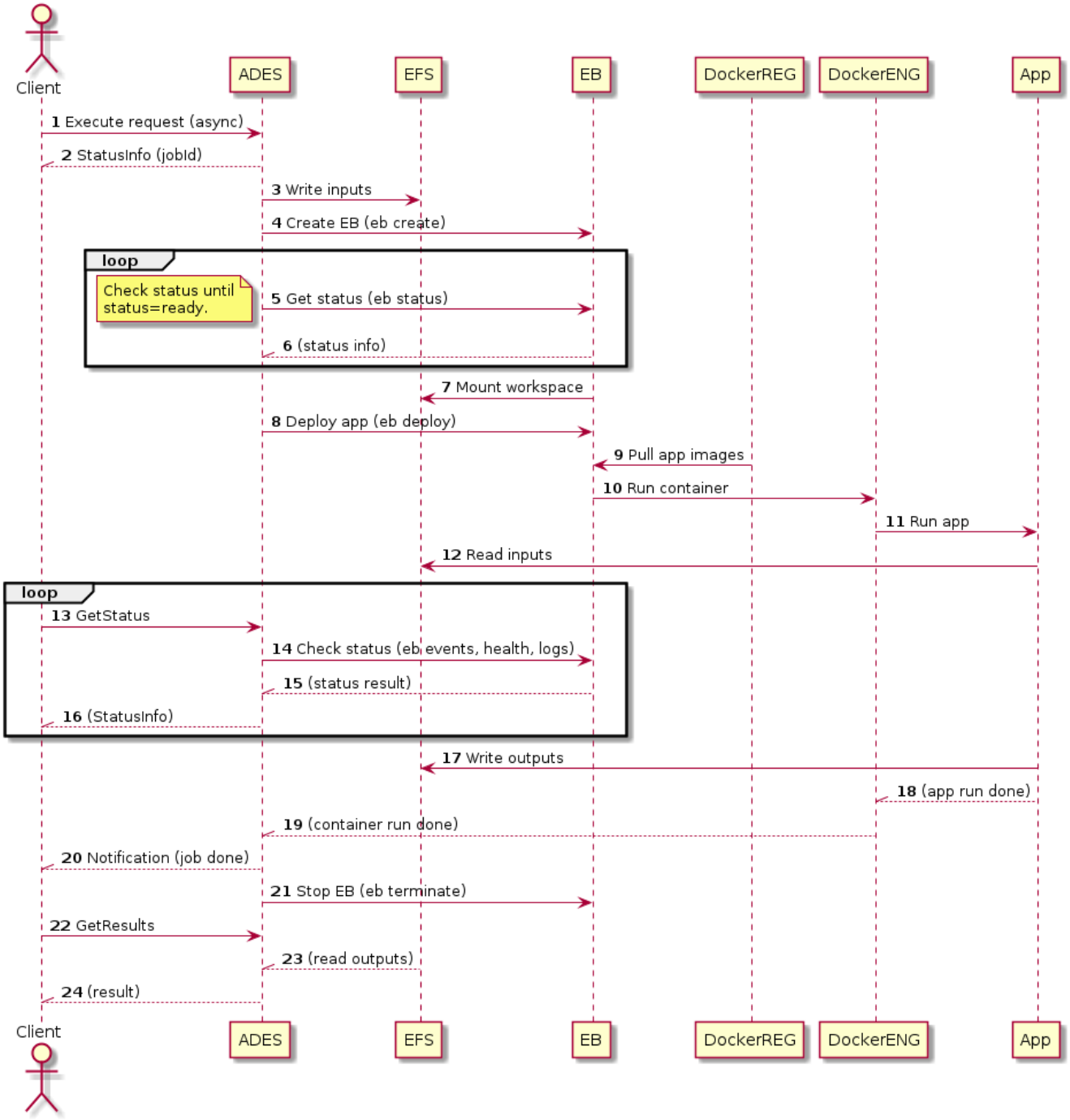

Figure 14. Sequence Diagram for Application Execution

Key:

1. ADES = Application Deployment and Execution Service
2. EFS = Elastic File System
3. EB = Elastic Beanstalk
4. DockerREG = Docker registry
5. DockerENG = Docker engine
6. App = Application bundled in Docker image

6.6.6. Notification

Introduction

The WPS standard defines a polling mechanism (see OGC 14-065, 9.4) for monitoring the status of an executing process to determine when a job is complete. A more efficient approach would be to have the ADES push a notification to the client when the execution of a job has been completed.

For Testbed-13, the CubeWerx server has implemented a push notification mechanism based on the work done during OGC Testbed 12 and described in "Testbed-12 Implementing Asynchronous Services Response Engineering Report (see OGC 15-023r3, 7.2).

NOTE	This capability is implemented in addition to the standard polling (i.e. GetStatus) approach defined in the WPS standard (see OGC 14-065,9.9).

ResponseHandler parameter

Push notification is triggered by the presence of the ResponseHandler parameter on a WPS Execute request. The following XML-schema fragment defines the ResponseHandler parameter:

```
<xsd:element name="ResponseHandler" type="xsd:anyUri"
             minOccurs="0" maxOccurs="unbounded"/>
```

The ResponseHandler parameter shall be defined in the http://www.opengis.net/ows/2.0 namespace.

One or more ResponseHandler parameters may be specified in an Execute request. The value of the ResponseHandler shall either be a URL. The form of the URL shall be a valid expression of one of the notification schemes that the server claims to support in its capabilities document.

This standard does not define a normative set of notification schemes but possible schemes include:

Email scheme as per RFC 2368

Example — mailto:tb12@pvretano.com

Sms scheme as per Apple Inc. (see RFC 5724)

Example – sms:1-555-555-5555

Webhook as per http://www.webhooks.org

Example — http://www.myserver.com/posthandler

Example:

The following example wpsExecute request shows the use of the ResponseHandler parameter:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute
   xmlns:wps="http://www.opengis.net/wps/2.0"
   xmlns:ows="http://www.opengis.net/ows/2.0"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wps/2.0
                       http://www.pvretano.com/schemas/wps/2.0/wps.xsd"
   service="WPS"
   version="2.0.0"
   response="document"
   mode="async">
   <wps:Extension>
      <ows:ResponseHandler>mailto:pvretano_tb13@gmail.com</ows:ResponseHandler>
   </wps:Extension> -->
   <ows:Identifier>...</ows:Identifier>
   .
   .
   .
</wps:Execute>
```

Response

The response to a wps:Execute request that contains ows:ResponseHandler elements is the standard asynchronous wpsStatusInfo response (see OGC 14-065,9.9.2) that has been extended to accommodate hypermedia controls. The following XML-Schema fragment redefines the standard response to include hypermedia controls in the form of ATOM links (see OGC 12-007r2, 9.1.3):

```
<element name="StatusInfo">
   <complexType>
      <sequence>
         <element ref="wps:JobID"/>
         <element name="Status">
            <simpleType>
               <union>
                  <simpleType>
                     <restriction base="string">
                        <enumeration value="Succeeded"/>
                        <enumeration value="Failed"/>
                        <enumeration value="Accepted"/>
                        <enumeration value="Running"/>
                     </restriction>
                  </simpleType>
                  <simpleType>
                     <restriction base="string"/>
                  </simpleType>
               </union>
            </simpleType>
         </element>
         <!-- reference to job expiration date -->
         <element ref="wps:ExpirationDate" minOccurs="0"/>
         <element name="EstimatedCompletion" type="dateTime" minOccurs="0"/>
         <element name="NextPoll" type="dateTime" minOccurs="0"/>
         <element name="PercentCompleted" minOccurs="0">
            <simpleType>
               <restriction base="integer">
                  <minInclusive value="0"/>
                  <maxInclusive value="100"/>
               </restriction>
            </simpleType>
         </element>
         <element ref="atom:link" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>
```

This allows zero or more hypermedia controls to be included in the response. Two "rel" values, rel="monitor" and rel="cancel" are used by the CubeWerx server to provide links that may be used to monitor the status of a job and links that may be used to cancel a job.

The following example shows the use of hypermedia control in the response to an Execute request:

```
?xml version="1.0" encoding="UTF-8"?>
<wps:StatusInfo
   xmlns:wps="http://www.opengis.net/wps/2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wps/2.0 ../wps.xsd">
   <wps:JobID>urn:cw:wps:jobId:9cf6bcc4-c8ed-11e7-a37a-1f5a82589b14</wps:JobID>
   <wps:Status>Accepted</wps:Status>
   <atom:link rel="cancel" href="http://www.acme.com/wps?..."/>
   <atom:link rel="monitor" href="http://www.acme.com/wps?service=WPS&version=2.0
&request=GetStatus&..."/>
</wps:StatusInfo>
```

In this case, the client may follow the rel="cancel" link to cancel the job and may use the rel="monitor" link to get the current status of a job. The advantage of using hypermedia controls is that the hypermedia controls are opaque thus relieving the client of having to know how to construct these URLs. All the clients need to know is how to interpret the rel value and how to follow links.

Chapter 7. Alternative approach

7.1. Overview

The approach described in this chapter is an alternative to the one described in the previous Implementation chapter. It is based on the use of a Transactional WPS (WPS-T) as described in the *WPS 2.0 Transactional Extension* [3] discussion paper. This discussion paper was itself derived from the WPS 1.0 based *Transactional WPS Extension (WPS-T) Specification* [4] and from the experience gained in several ESA projects where the Transactional WPS 1.0 was used to deploy Business Process Execution Language (BPEL) workflows, Grid applications, Oozie workflows, and Java applications.

In this approach, as illustrated in Figure 15, the Application Deployment and Execution Service (ADES) is essentially a set of WPS-T servers, each server encapsulating a particular cloud processing environment (e.g. IPT Poland or AWS) where the TEP applications are deployed and executed. The TEP applications are containerized applications running on virtual machines equipped with a container execution engine (e.g. Docker Engine) and possibly container clustering tools (e.g. Docker Swarm, Mesos, or Kubernetes). The container technology used in this testbed is Docker.

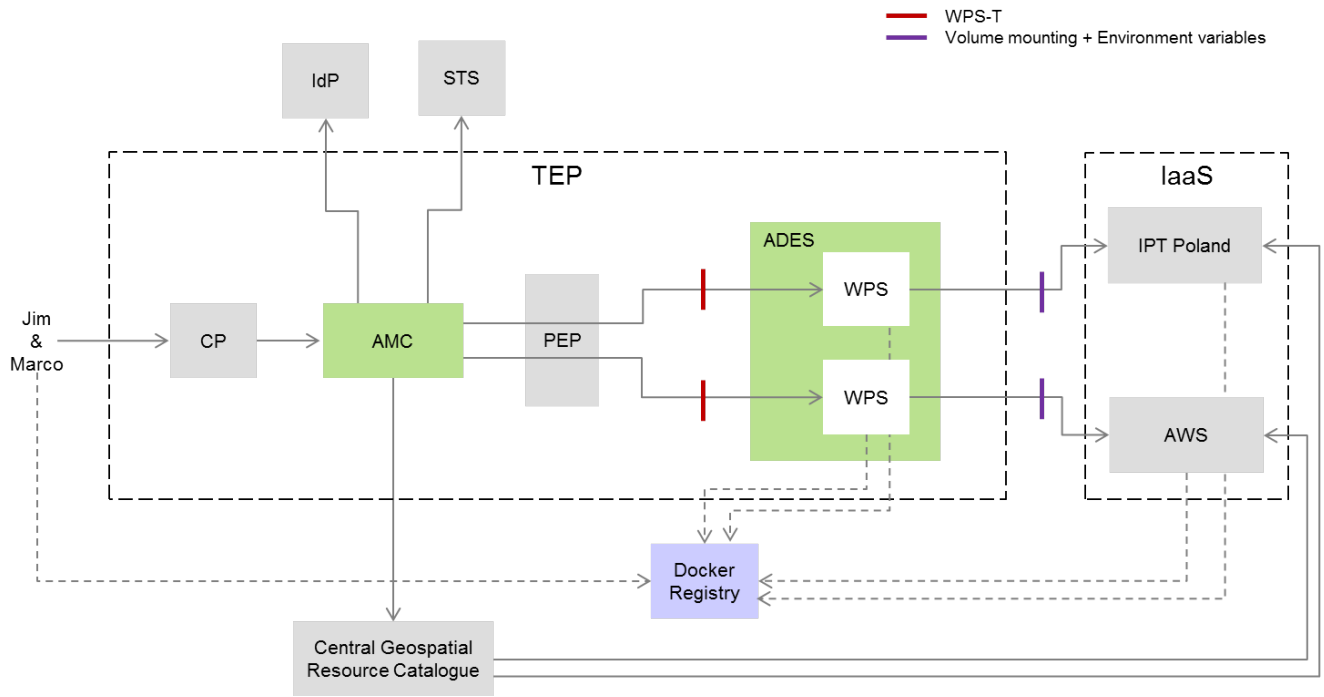

Figure 15. Alternative Approach Overview

7.2. Application Package

The Application package proposed in this testbed is an OWS Context document as defined in the *Exploitation Platform Application Package Engineering Report* [2] . This document provides all the information necessary for both the Application Management Client (AMC) and the Application Deployment and Execution Service to perform their work for the corresponding application. This information includes a WPS Process Description that matches the inputs and outputs of the application.

In this approach, instead of embedding (using an OWC Offering) the WPS Process Description inside the OWS Context document, an enhanced WPS Process Description is actually used to embed (using OWS Metadata) the remaining part of the OWS Context. This is done using the ApplicationContext element as illustrated in the XML fragment below. Therefore, no information is lost when using the enhanced WPS Process Description. The advantage of this approach is that the discovery (e.g. by the AMC) of Application Packages coincides with the discovery of WPS Processes.

The source of the Docker image used in this chapter is available on GitHub (https://github.com/spacebel/landcover) and the image is available on Docker Hub (https://hub.docker.com/r/cnlspacebel/landcover/).

WPS Process Description with OWS Context

```
<wps:ProcessOffering xmlns:wps="http://www.opengis.net/wps/2.0" ... jobControlOptions
="async-execute dismiss" outputTransmission="value reference">
    <wps:Process>
        <ows:Title>Land Cover Mapping</ows:Title>
        <ows:Abstract>Lang Cover Mapping is based on the Sentinel-2 processing
workflow generated for the F-TEP platform.</ows:Abstract>
        <ows:Identifier>LandCover</ows:Identifier>
        <ows:Metadata xlink:role="http://www.opengis.net//tb13/eoc/applicationContext
">
            <eoc:ApplicationContext>
                <feed xmlns="http://www.w3.org/2005/Atom" ... >
                    <title>EOC Land Cover Application Package</title>
                    ...
                    <entry>
                        <title>EOC Land Cover Application</title>
                        ...
                        <!-- DockerImage offering -->
                        <owc:offering code="http://www.opengis.net/tb13/eoc/docker">
                            <owc:content type="text/plain">...</owc:content>
                        </owc:offering>
                    </entry>
                    <entry>
                        <title>OpenSearch Collections</title>
                        ...
                        <!-- OpenSearch offering for IPT Poland -->
                        <owc:offering code="http://www.opengis.net/spec/owc-
atom/1.0/req/opensearch">
                            <owc:content type="application/opensearchdescription+xml"
.../>
                        </owc:offering>
                        <!-- OpenSearch offering for AWS -->
                        <owc:offering code="http://www.opengis.net/spec/owc-
atom/1.0/req/opensearch">
                            <owc:content type="application/opensearchdescription+xml"
.../>
                        </owc:offering>
                    </entry>
                </feed>
            </eoc:ApplicationContext>
        </ows:Metadata>
        <wps:Input>...</wps:Input>
        ...
        <wps:Output>...</wps:Output>
    </wps:Process>
</wps:ProcessOffering>
```

7.3. Application Management Client

The AMC is a generic component that can be used for any TEP Application. It relies entirely on the information provided in the Application Package (i.e. the enhanced WPS Process Description described in the previous section) in order to dynamically build the application specific user interface needed to:

- Discover in the Central Geospatial Resource Catalogue the satellite images (e.g. Sentinel-2) to be processed,
- Capture the inputs for the execution of the processing TEP Application,
- Monitor the execution status of the TEP application, and
- Download and visualize the processing outputs.

The AMC uses operations provided by the Central Geospatial Resource Catalogue and the ADES as illustrated in Figure 16.

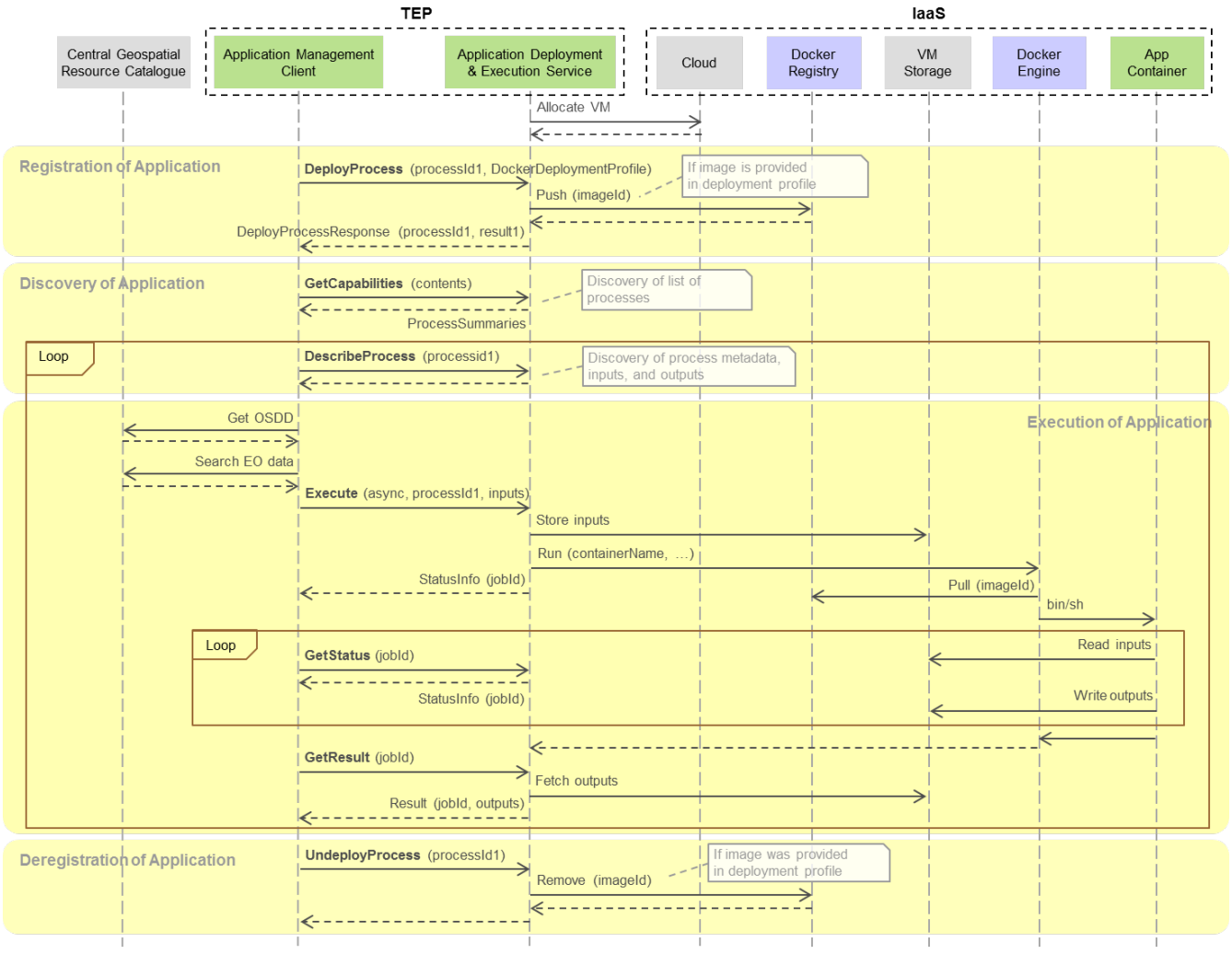

Figure 16. Sequence of WPS Operations used by Client

The TEP Application deployment is performed (at Marco's request) using the WPS-T DeployProcess operation. The typical case is when the container image is available on an external registry. However, if the image is a relatively small layer on top of another image, this image could also be passed by value in the DeployProcess request and stored in a local registry.

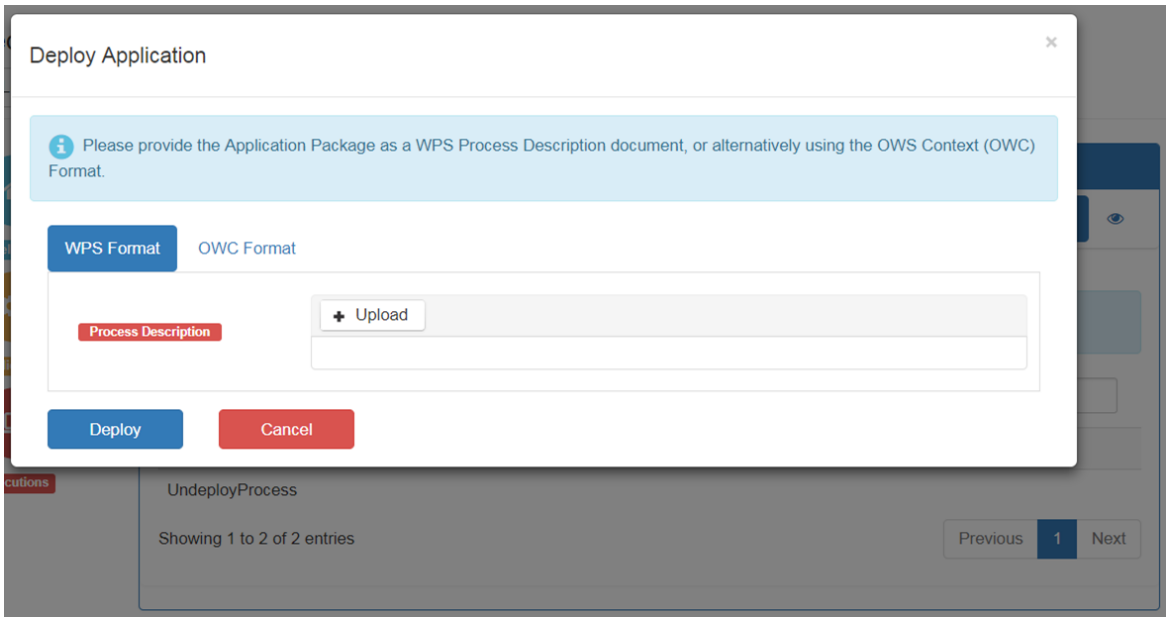

Figure 17. Deploy Application Screen

The discovery by the AMC of available (i.e. deployed) applications is performed using the WPS GetCapabilities operation followed by a WPS DescribeProcess operation. The WPS DescribeProcess operation returns a process description with all the information specific to the selected application. In particular, the OWS Context contains the entry point in the Central Geospatial Resource Catalogue i.e. the URL to get the OpenSearch Description Document (OSDD) and the reference to the container image in a Docker registry. Of course, the process description also contains all the details regarding the application inputs and outputs.

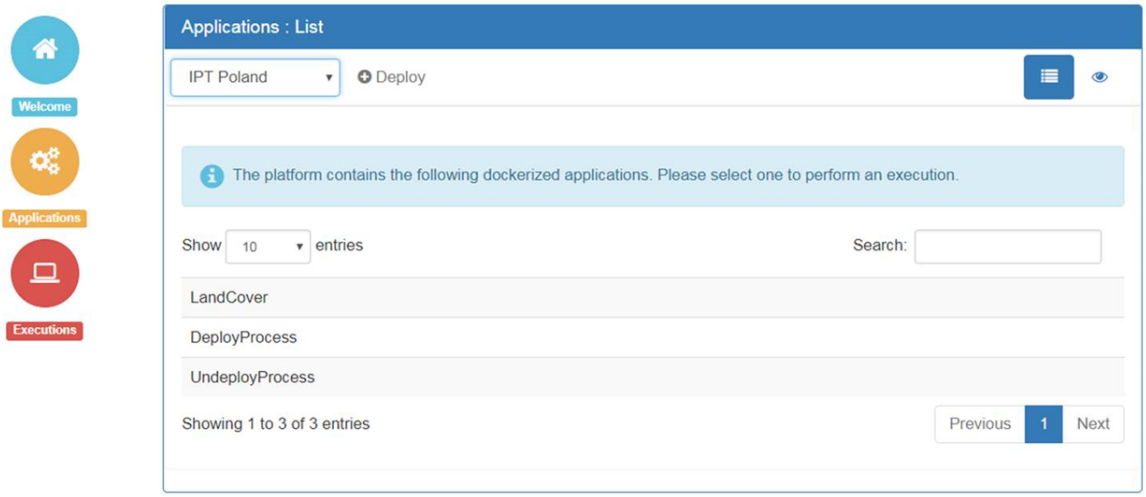

Figure 18. Applications Screen

The execution of the application is performed (at Jim's request) using WPS Execute operation. To capture the inputs, the AMC uses the process description to build the appropriate user interface. For inputs to be obtained from the Central Geospatial Resource Catalogue, the AMC uses the information (template and parameters) contained in the OSDD to build the appropriate user interface.

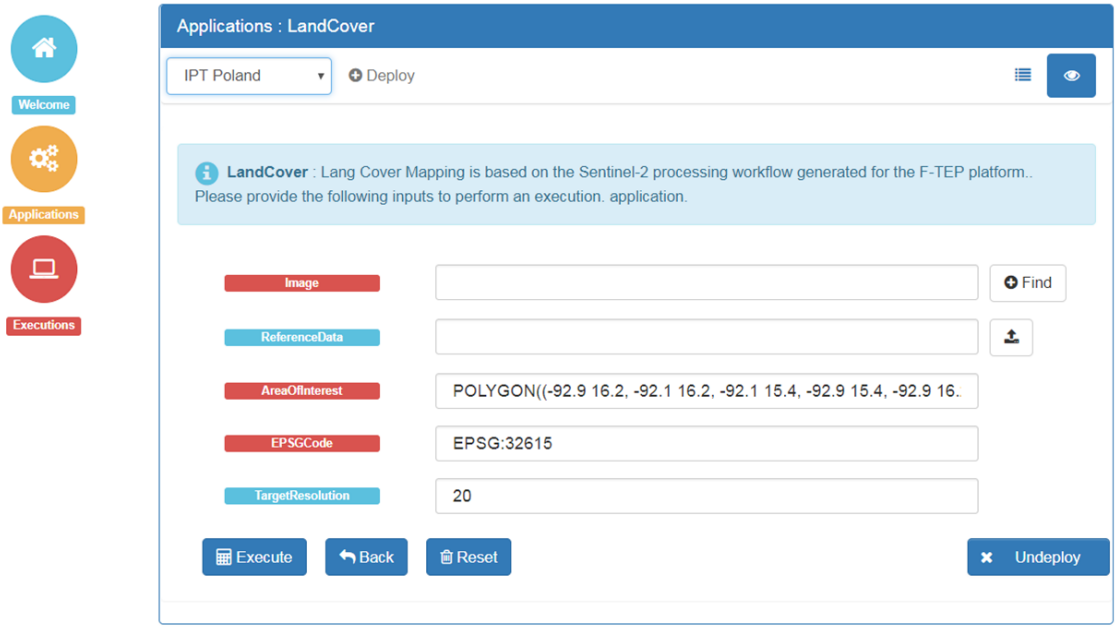

Figure 19. Execute Application Screen

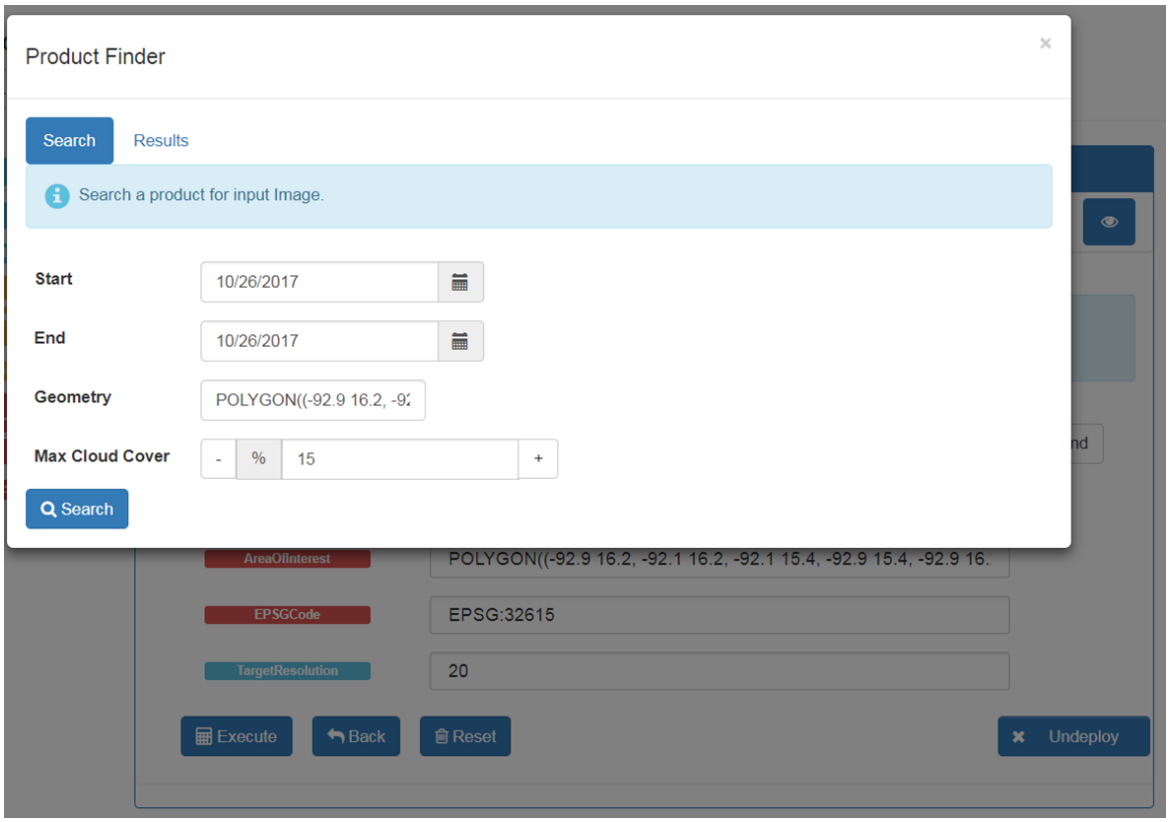

Figure 20. Product Search Screen

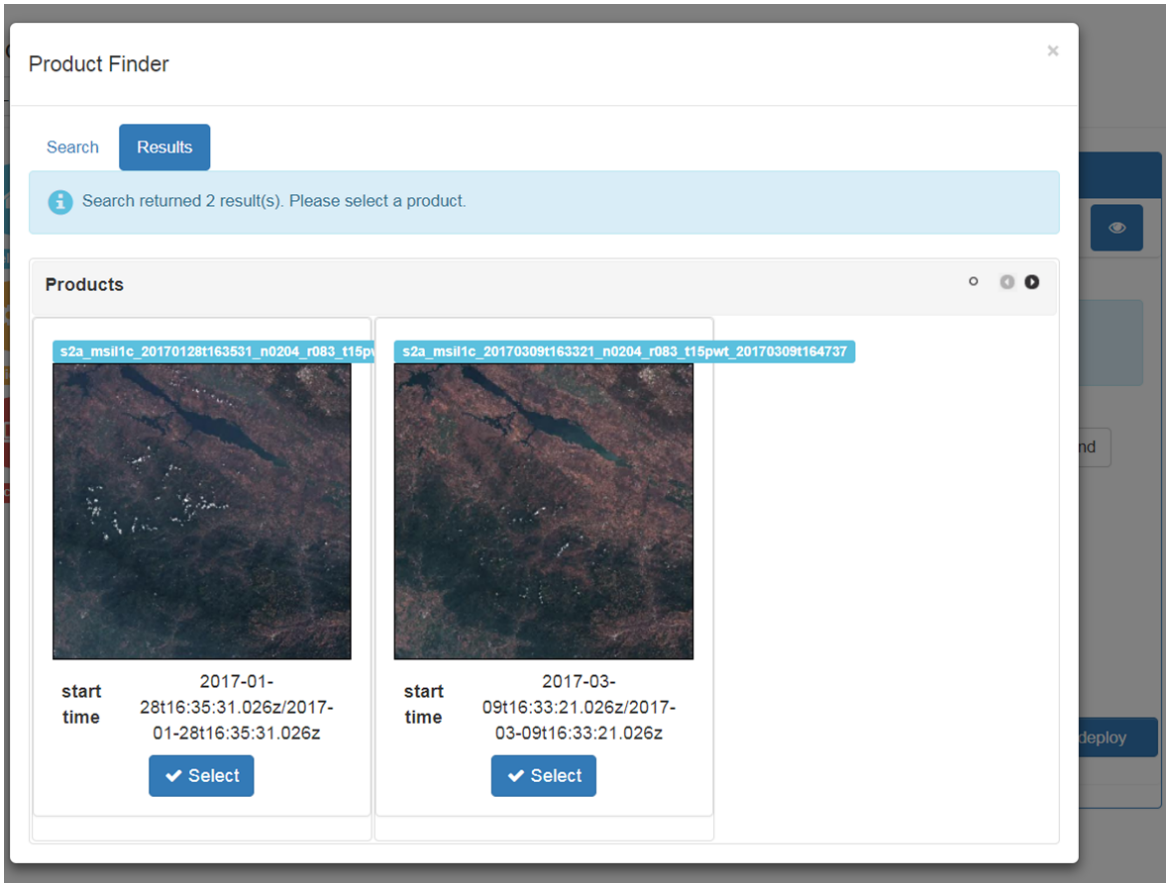

Figure 21. Product Selection Screen

To monitor the execution of the applications, the AMC uses the WPS GetStatus operation. Note that TEP Applications have typically long execution times and are, therefore, executed in asynchronous mode.

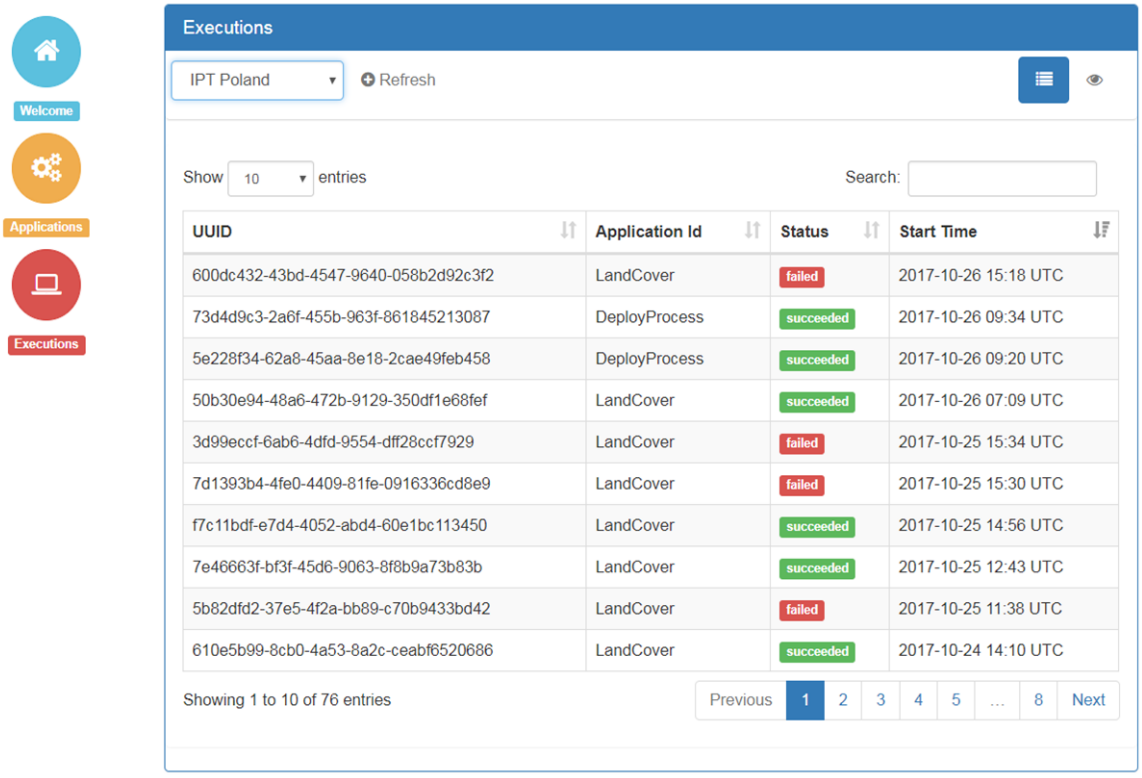

Figure 22. Executions Screen

Finally, to get the results generated by the application, the AMC uses the WPS GetResult operation. Depending on the type of results, a download and/or preview of the results is also provided.

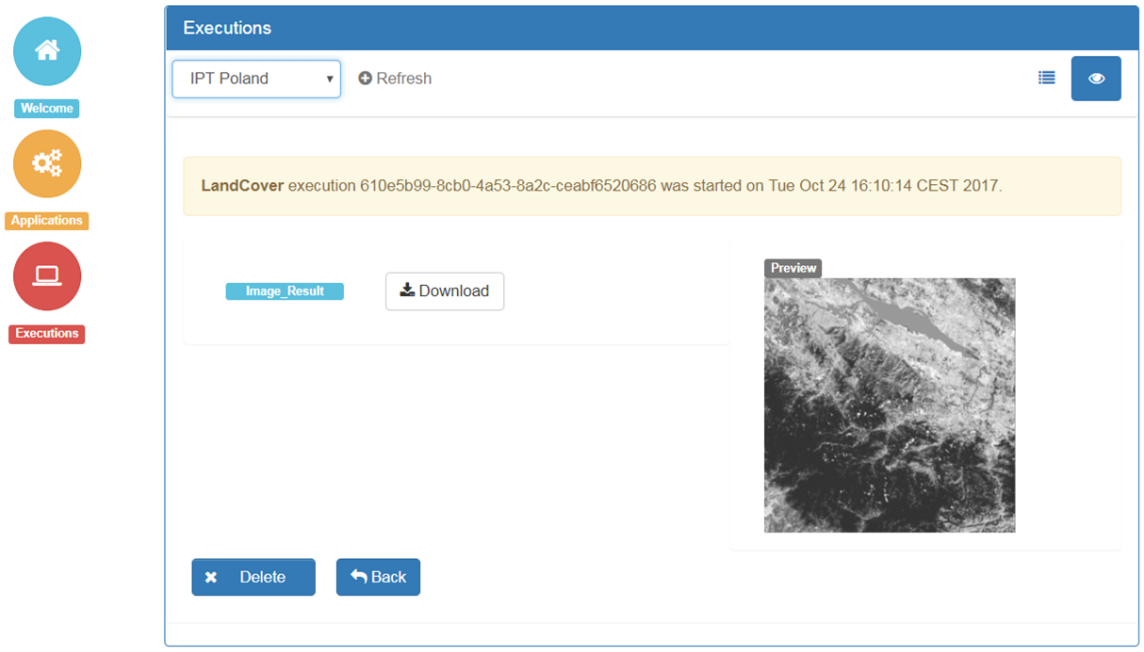

Figure 23. Execution Result Screen

The TEP Application used in Alternative approach is the Land Cover Mapping application described in the *F-TEP Application in OGC Testbed 13* [8] document.

The development of the testbed AMC is done using a Web application based on JavaServer Faces (JSF) and deployed in a GlassFish server. The AMC is available at the following location: http://ld-ogc-tb13-tep.spacebel.be/TB13-WebApp/ .

7.4. Application Deployment and Execution Service

The ADES is a component that implements the WPS-T operations shown on Figure 16. This figure also illustrates the ADES interaction with the cloud environment and with the containerized TEP Application.

The WPS-T implementation is very close to the *WPS 2.0 Transactional Extension* [3] document with some minor differences as explained below. This document covers the HTTP POST + XML and KVP encodings. REST + JSON encoding is proposed below in a separate section.

As shown on Figure 24, the Capabilities document is unchanged and contains the list of the deployment profiles supported by the server.

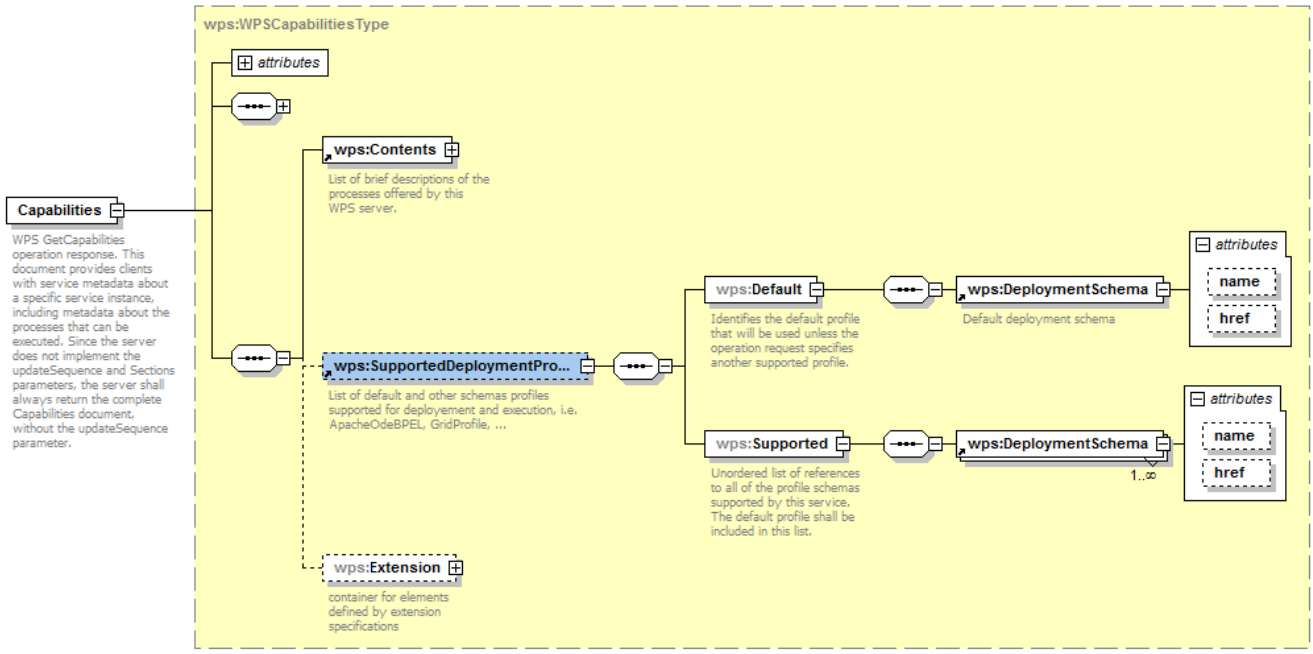

Figure 24. WPS-T Supported Deployment Profiles in Capabilities Document

The DeployProcess request is modified slightly to include a ProcessOffering (that includes a ProcessDescription) instead of a ProcessDescription and a DeploymentProfile element is added to hold the DeploymentProfileName and the ExecutionUnit. A ProfileExtension is introduced to allow for additional information specific to a particular processing environment. For example, it could possibly be used to pass the ApplicationContext described earlier instead of including it in the OWS Metadata of the Process Description. The ExecutionUnit is made optional (if specified, the corresponding information in the enhanced WPS Process Description is overwritten) and its InstructionPackage is renamed simply as Package. This is illustrated on Figure 25.

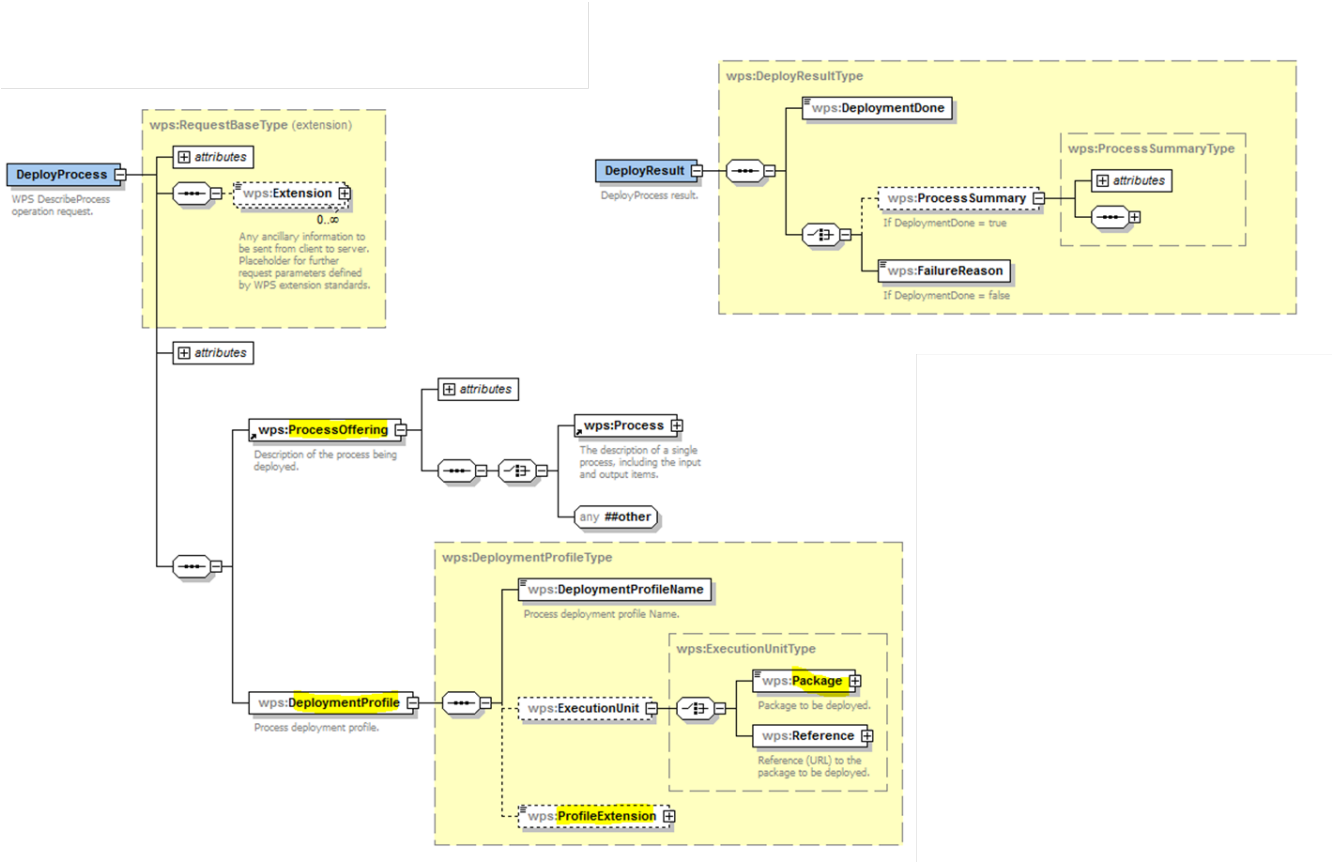

Figure 25. WPS-T DeployProcess Operation

As shown on Figure 26, the UndeployProcess operation is unchanged. The KeepExecutionUnit is not used in the testbed as this is handled by the Docker Engine.

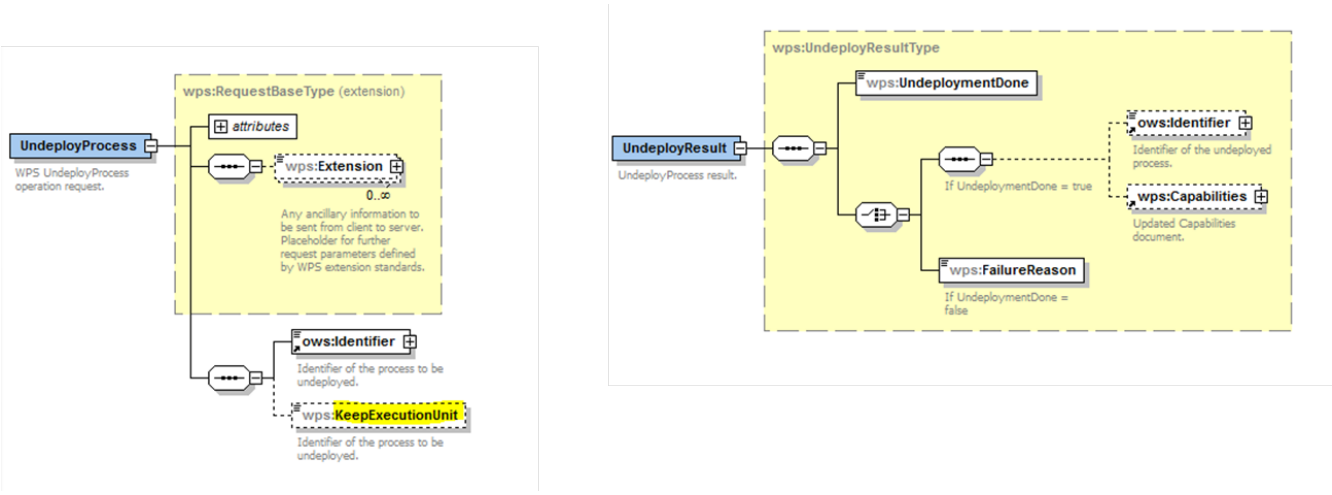

Figure 26. WPS-T UndeployProcess Operation

As illustrated in Figure 16, the ADES interacts with the Docker Engine and with the TEP Application. The ADES interacts with the Docker Engine in order to prepare the container environment (e.g. mounting shared storage in the container and creating environment variables) and to start the application in the container (resulting in the execution of the shell script defined in the Docker image). The ADES interacts with the TEP Application through environment variables to pass WPS inputs and WPS outputs.

The values of WPS inputs of LiteralData type and the file location of WPS inputs of ComplexData type are assigned (by the ADES) to the corresponding environment variables provided to the TEP Application. The target file location of WPS outputs (of both LiteralData and ComplexData types) are assigned (by the ADES) to the corresponding environment variables provided to the TEP Application. It is the responsibility of the ADES to make the input files available in the container environment by extracting them from the execute request (data transmission by value), or by fetching them from a remote location (data transmission by reference with a protocol such as http://), or by mounting them from a local store (data transmission by reference with a protocol such as file:/// or s3://). It is the responsibility of the TEP Application to create/append results to ouput files.

To ensure interoperability, a convention is used to name the environment variables i.e. by prefixing the WPS input identifier with WPS_INPUT_ (e.g. WPS_INPUT_Image) and the output identifier with WPS_OUTPUT_ (e.g. WPS_OUTPUT_Image). Although not needed by the Land Cover Mapping application used in the testbed, the convention should also cover repeated WPS inputs and WPS outputs (e.g. WPS_INPUT_Mask_x with x=1,2,3…) as well as nested WPS inputs and WPS outputs (e.g. WPS_INPUT_Band_x_IntensityMin).

The development of the testbed ADES is based on 52°North WPS 2.0 implementation that was used in OGC Testbed 12 (https://github.com/52North/WPS/tree/wps-4.0, Git revision 9aaa70b341) and is available on GitHub (https://github.com/spacebel/WPS). Its Wiki (https://github.com/spacebel/WPS/wiki) provides additional details. The ADES is available at the following location: http://ld-ogc-tb13-tep.spacebel.be:8081/wps/ .

7.5. Interoperability

The approach described in this chapter is an alternative to the approach described in the Implementation chapter. This section proposes mechanisms that can be used to allow for interoperability of the above described WPS-T based ADES with other AMC clients and Application Packages.

Some AMC clients are designed to interface with an ADES service that offers a dedicated DeployProcess WPS process that is used for the deployment of the TEP Application. The execution of this process takes the application OWS Context as an input. Therefore, a similar non-transactional DeployProcess WPS process can be added in the WPS-T based ADES. This process can extract from the OWS Context all the information needed to invoke an "internal" DeployProcess operation as illustrated in Figure 27. Once deployed, the execution of the newly deployed process (e.g. LandCover) is identical for all AMC clients. A similar approach can be used for the undeployment of the TEP Application using an UndeployProcess WPS process.

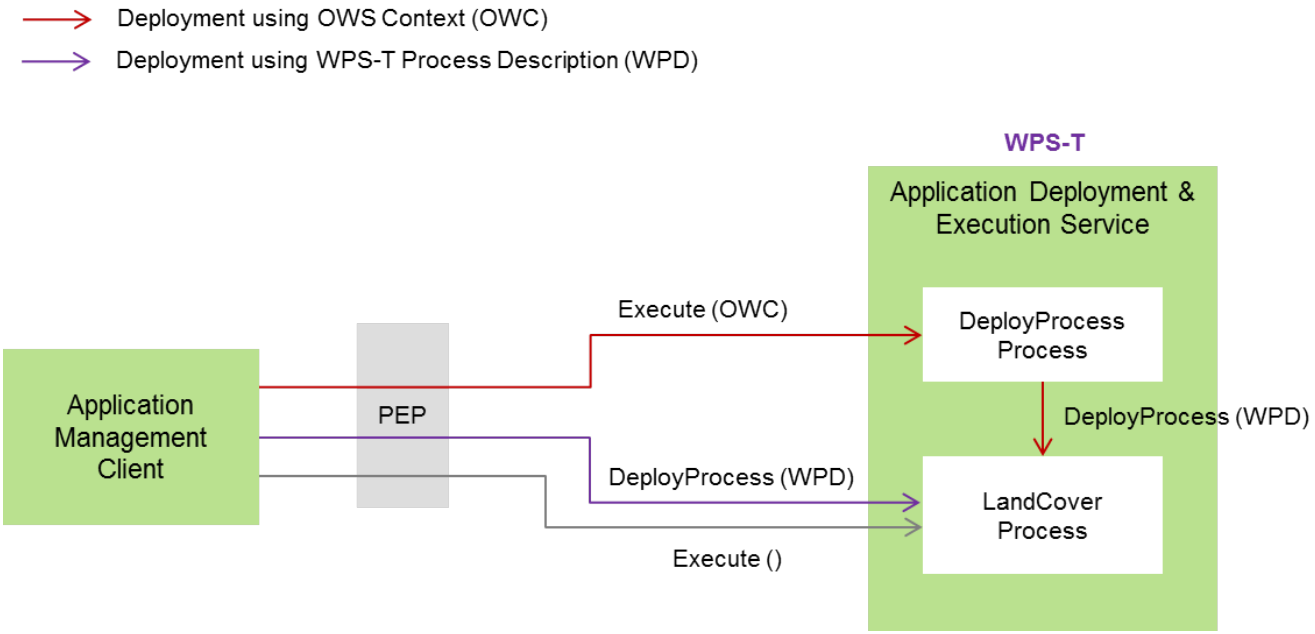

Figure 27. ADES Interoperability

Some ADES servers are designed to interface with an application that uses staging commands. Therefore, to be able to use the same application container image for all ADES servers, staging commands can be written to wrap the interface based on environment variables and volume mounting of shared storage.

7.6. REST + JSON Encoding

The REST interface to the Transactional WPS is built on top of the REST binding defined in chapter 11 of *OGC Testbed 12 REST Architecture Engineering Report* [5] . No new resources are needed but, as shown on Figure 28, two new HTTP operations must be defined to support the deployment and undeployment of WPS processes.

Note that a new HTTP DELETE operation was also added to support the Dismiss extended WPS operation defined in chapter 12 of the *OGC WPS 2.0 Interface Standard* [OGC 14-065].

HTTP Operation	Access Path	Parameters	Description
GET	{WpsRESTBaseURL}	NA	Retrieval of Capabilities resource
GET	{WpsRESTBaseURL}/processes	NA	Retrieval of ProcessCollection resource
GET	{WpsRESTBaseURL}/processes/{process-id}	NA	Retrieval of a Process resource (process description)
GET	{WpsRESTBaseURL}/processes/{process-id}/jobs	NA	Retrieval of list of jobs of a specific process (JobCollection)
POST	{WpsRESTBaseURL}/processes/{process-id}/jobs	Execute request (contained in body)	Execution of a process
GET	{WpsRESTBaseURL}/processes/{process-id}/jobs/{job-id}	NA	Retrieval of a single Job resource
GET	{WpsRESTBaseURL}/processes/{process-id}/jobs/{job-id}/outputs	NA	Retrieval of job results
DELETE	{WpsRESTBaseURL}/processes/{process-id}/jobs/{job-id}	NA	Dismiss a single Job resource
POST	{WpsRESTBaseURL}/processes	Deploy process request (contained in body)	Deployment of a process
DELETE	{WpsRESTBaseURL}/processes/{process-id}	NA	Undeployment of a process
Notations: {WpsRESTBaseURL}: The base URL of the WPS REST endpoint. {process-id}: identifier of a process. {job-id}: identifier to a job.			

Figure 28. WPS-T REST HTTP Operations

The JSON encoding of the request and responses are derived from XML encoding using the rules defined in chapter 6 of *OGC Testbed 11 Implementing JSON/GeoJSON in an OGC Standard Engineering Report* [6] .

An example of the JSON encoding of (the body of) the DeployProcess request is shown below.

WPS-T JSON DeployProcess Request

```
{
    "DeployProcess": {
        "service": "WPS",
        "version": "2.0.0",
        "ProcessOffering": {
            "jobControlOptions": "async-execute dismiss",
            "outputTransmission": "value reference",
            "Process": {
                "Title": "Land Cover Mapping",
                "Abstract": "Lang Cover Mapping is based on the Sentinel-2 processing
workflow generated for the F-TEP platform.",
                "Identifier": "LandCover",
                "Metadata": {
                    "role": "http://www.opengis.net//tb13/eoc/applicationContext",
                    "ApplicationContext": {
                        "feed": {
                            "title": "EOC Land Cover Application Package",
                            ...,
                            "entry": [
                                {
                                    "title": "EOC Land Cover Application",
                                    ...,
                                    "offering": {
                                        "code":
"http://www.opengis.net/tb13/eoc/docker",
                                        "content": {
                                            "type": "text/plain"
                                            "content":
"registry.hub.docker.com/cnlspacebel/landcover"
                                        }
                                    }
                                },
                                {
                                    "title": "OpenSearch Collections",
                                    ...,
                                    "offering": [
                                        {
                                            "code": "http://www.opengis.net/spec/owc-
atom/1.0/req/opensearch",
                                            "content": {
                                                "type":
"application/opensearchdescription+xml",
                                                "href":
"http://geo.spacebel.be/opensearch/description.xml?parentIdentifier=EOP%3AIPT%3ASentin
el2"
                                            }
                                        },
```

```
                                        {
                                            "code": "http://www.opengis.net/spec/owc-
atom/1.0/req/opensearch",
                                            "content": {
                                                "type":
"application/opensearchdescription+xml",
                                                "href":
"http://geo.spacebel.be/opensearch/description.xml?parentIdentifier=EOP%3ASENTINEL-
HUB%3ASentinel2"
                                            }
                                        }
                                    ]
                                }
                            ]
                        }
                    }
                },
                "Input": [
                    {
                        "Title": "Sentinel-2 Image",
                        ...
                    },
                    ...
                    }
                ],
                "Output": {
                    ...
                }
            }
        },
        "DeploymentProfile": {
            "DeploymentProfileName": "DockerDeploymentProfile",
            "ExecutionUnit": {
                "Reference": {
                    "href": "registry.hub.docker.com/cnlspacebel/landcover"
                }
            }
        }
    }
}
```

The JSON encoding of the (body of the) response to the above DeployProcess request example is shown below.

WPS-T JSON DeployProcess Response

```
{
    "DeployResult": {
        "DeploymentDone": "true",
        "ProcessSummary": {
            "jobControlOptions": "async-execute",
            "Title": "Land Cover Mapping",
            "Identifier": "LandCover"
        }
    }
}
```

The JSON encoding of (the body of) the UndeployProcess response is shown below. The UndeployProcess request has no body.

WPS-T JSON UndeployProcess Response

```
{
    "UndeployResult": {
        "UndeploymentDone": "true",
        "Identifier": "LandCover"
    }
}
```

The plan was to develop the REST + JSON encodings for the WPS-T ADES using the WPS 2.0 REST façade implementation from 52° North that was also used in OGC Testbed 12 (https://github.com/52North/wps-proxy). However, due to lack of time, this implementation was not done.

7.7. Security Aspects

As illustrated on Figure 15, the security approach to be followed in the testbed, at least for IPT Poland where an Identity Provider (IdP) and a Security Token Service (STS) are available, is compliant with the *User Management Interfaces for Earth Observation Services* [7] document. The HTTP bindings described in sections 7.1.2 (STS interface) and 7.2.2 (PEP interface) of that document are used (instead of the SOAP bindings).

Authentication (and Single Sign-On) is performed on the client side (AMC) using a checkpoint (in this case a Shibboleth Service Provider) that intercepts and redirects, if the user is not already signed on, access requests to protected AMC resources to a Web page of the IdP (in this case a Shibboleth IdP) for authentication. Once authenticated, the checkpoint includes IdP provided assertions (e.g. about the username, email, etc.) in the HTTP header of the user request and forwards this request to the AMC.

Authentication and authorization is performed on the server side (ADES) using an STS and a Policy Enforcement Point (PEP) respectively. According to the "Local STS with External IdP" approach described in section 6.4.3.3 of [7] , the AMC client sends a signed Request Security Token (RST) with the username to the STS and gets back a SAML token signed by STS and encrypted with the public key of the PEP. The AMC then inserts this SAML token in the HTTP header of all service requests it sends towards the ADES (a new SAML token must be obtained frequently to avoid token expiration). The PEP then decrypts the SAML token, verifies the token signature, and performs authorization using policies based on the SAML assertions. If authorized, the AMC request is finally forwarded to the ADES (with or without the decrypted SAML token).

The PEP software provided by ESA for the testbed does not actually perform the authorization part, but simply passes the assertions (SAML attributes) to the ADES in the HTTP header.

Although compliant with section 7.2.2 of [7] , the provided PEP is not inline with the spirit of the *User Management Interfaces for Earth Observation Services* [7] document (see section 6.1) where the PEP was meant to perform authorization in a transparent (non-intrusive) way i.e. without impacting the target service (in this case the ADES). Therefore, and also due to lack of time, the security aspects were not implemented in this testbed.

Appendix A: Revision History

Table 5. Revision History

Date	Release	Editor	Primary clauses modified	Descriptions
June 15, 2017	0.1	P. Gonc alves	all	initial version
September 30, 2017	1.0	P. Gonc alves	all	Draft ER

Appendix B: Bibliography

[1] OGC: OGC Testbed 12 Annex B: Architecture (2017).

[2] OGC: OGC 17-023, OGC Testbed-13, EP Application Package ER (2017).

[3] Cauchy A.: OGC 13-071r1, OpenGIS WPS 2.0 Transactional Extension, OGC Discussion Paper, 2014-05-28 [https://portal.opengeospatial.org/files/?artifact_id=74162]

[4] Schäffer B.: OGC 08-123, Transactional WPS Extension (WPS-T) Specification, OGC Discussion Paper, 2008-08-20 [https://portal.opengeospatial.org/files/?artifact_id=74161]

[5] OGC: OGC 16-035, Testbed-12 REST Architecture Engineering Report (2016) [https://portal.opengeospatial.org/files/?artifact_id=71494]

[6] OGC: OGC 15-053r1, Testbed 11 Implementing JSON/GeoJSON in an OGC Standard Engineering Report, 2015-08-19 [https://portal.opengeospatial.org/files/?artifact_id=64595]

[7] Denis P., Jacques P.: OGC 07-118r9, OGC User Management Interfaces for Earth Observation Services, OGC Best Practice, 2014-04-28 [https://portal.opengeospatial.org/files/?artifact_id=54929]

[8] Tergujeff R., Rauste Y., Seitsonen L.: F-TEP Applications in OGC Testbed 13, 2017-10-27 [https://portal.opengeospatial.org/files/?artifact_id=73732]