OGC® DOCUMENT: 18-046

External identifier of this OGC® document: http://www.opengis.net/doc/PER/eoep-Hack2018

OGC®
Making location count.

OGC EARTH OBSERVATION EXPLOITATION PLATFORM HACKATHON 2018 ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2018-12-20
Approval Date: 2018-07-10
Publication Date: 2018-06-12
Editor: Ingo Simonis

Note

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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I SUMMARY

The *Earth Observation Exploitation Platform Hackathon 2018* was conducted to evaluate the standards based architecture for deploying and executing arbitrary applications close to the physical location of the data in heterogeneous cloud environments. The Hackathon was very successful in demonstrating both efficiency and sustainability of the architecture developed in Testbed-13. Efficient, because it was possible to setup the full execution workflow of 128 Sentinel-1 images within the 1.5 days of the Hackathon in a multi-vendor environment. Sustainable, because the architectural approach provides sufficient flexibility to cater for possible extensions and exchange of cloud & container middleware.

The Hackathon produced a number of suggestions for future work items. These include new tools to facilitate the process of Application Package generation to make it even simpler for scientists to bring their applications to the market; a more detailed specification to further improve the level of interoperability; and a best practice document with lots of examples that illustrate the necessary steps to make applications available.

Hackathon participants highlighted that such a level of robustness, flexibility, and maturity of the application-to-the-cloud architecture has been developed in nine months only during Testbed-13. The participants recommend to continue interlacing major OGC Innovation Program activities, such as testbeds, with short term rapid prototyping initiatives such as hackathons. Almost all participants of the Hackathon had been new to the OGC Innovation Program. These participants emphasized that the Hackathon provided an outstanding opportunity for newcomers to get quickly familiar with the latest standardization efforts and helped tremendously in understanding investments and new market opportunities for applications-in-the-cloud.

I.A. Requirements & Research Motivation

The Hackathon was conducted to stress-test results from Testbed-13 and to evaluate the benefit of having hackathons in between major OGC Innovation Program initiatives such as testbeds. Further on, the goal was to attract organizations that have not participated in the development of the applications-in-the-cloud architecture developments of Testbed-13 for a critical review.

I.B. Prior-After Comparison

Organizationally, it turned out that having hackathons in between Testbeds is extremely valuable. Hackathons attract new organizations, allow exploring alternatives very efficiently,

and help shaping upcoming initiatives with experiences and suggestions from the Hackathon discussions.

On the engineering side, Testbed-13 produced a number of alternative options without giving clear guidance on the preferred approach. This was issue was addressed successfully in the Hackathon. It turned out that the general architecture, interface design and Application Package model have been confirmed during the Hackathon. This is an important achievement, as it allows to continue the development of standardized approaches for application provision, ad-hoc deployment and dynamic execution in cloud environments in future OGC Innovation Program initiatives.

Interesting alternatives have been developed that include a WPS instance as essential part of a Docker Image. This approach (which can be considered a micro-service approach, as each instance contains a full service interface) gives more flexibility to the application developer. All mappings (e.g. the mounting of external data to internal paths) can be realized on the programmatic level. This approach requires higher programming skills if no tools are available to configure the embedded WPS service and to package the application on behalf of the developer.

I.C. Recommendations for Future Work

The results of this Engineering Report serve as direct input for the Earth Observation Clouds thread in Testbed-14. It is recommended to address the following aspects:

- More complex applications that involve data from multiple clouds,
- Fully secured environments,
- Develop tools that generate Application Packages or at least support scientists in this task,
- Develop best practices illustrating service setup and Application Package examples, and
- Provide more detailed specifications to enhance the level of interoperability.

I.D. Document contributor contact points

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

Contacts

NAME	ORGANIZATION
Ingo Simonis (editor)	OGC

NAME	ORGANIZATION
Benjamin Pross	52°North
Matthes Rieke	52°North
Christoph Stasch	52°North
Vincent Gaudissart	CS
Christophe Triquet	CS
Armin Costa	Eurac Research
Alexander Jacob	Eurac Research
Paulo Sacramento	Solenix
Daniel Robinson	Solenix
Bernard Valentin	Space Applications Services
Leslie Gale	Space Applications Services
Marian Neagul	UVT
Teodora Selea	UVT
Silviu Panica	IeAT
Jeroen Dries	VITO
David Pérez	Thales Alenia Space
Alejandro Mousist	Thales Alenia Space
Elisa Callejo	Thales Alenia Space

I.E. Cloud Providers

A special thank you to our CloudSigma, CloudFerro and Amazon for making cloud resources available to the Hackathon!

1

TERMS, DEFINITIONS AND ABBREVIATED TERMS

1 TERMS, DEFINITIONS AND ABBREVIATED TERMS

No terms and definitions are listed in this document.

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 shall apply.

1.1. Abbreviated terms

ADES	Application Deployment and Execution Service
AP	Application Package
EMS	Exploitation Platform Management Service
EO	Earth Observation
EP	Exploitation Platform
IAAS	Infrastructure as a Service
MEP	Mission Exploitation Platform
PAAS	Platform as a Service
SOA	Service Oriented Architecture
WPD	Web Processing Service ProcessDescriptor
WPS	Web Processing Service

2

INTRODUCTION

2 INTRODUCTION

The Earth Observation *Exploitation Platform Hackathon 2018* was organized by OGC at the European Space Agency Operations Center in Darmstadt, Germany, on May 3rd and 4th. Sponsored by ESA and NRCan, the goal of the Hackathon was to evaluate results from the recently concluded Testbed-13 initiative. Testbed-13 produced three Engineering Reports to describe a standards-based approach for deploying arbitrary applications on cloud platforms.

- OGC Testbed-13: Exploitation Platform Application Package (OGC 17-023) describes a container format that describes applications packaged in a Docker container in sufficient detail to allow their automated deployment and execution on cloud platforms.
- OGC Testbed-13: Application Deployment and Execution Service (OGC 17-024) describes how applications that have been described according to the Application Package specification are actually deployed and executed on cloud platforms.
- OGC Testbed-13: Cloud (OGC 17-035) provided further background on EO applications and cloud platforms in general.

In particular the first two Engineering Reports state several alternatives and provide recommendations on the preferred approach very carefully only. The full architecture is described best in [1] Thus, the Hackathon was organized to experiment with the alternatives and to give room for additional approaches suggested by Hackathon participants.

2.1. The Challenge

The challenge was rather straight forward: Take an arbitrary application, store this application in a Docker container, describe this application with the necessary detail to allow standards-based automated deployment and execution, and then register this application with a Web service that allows executing the workflow publicly. Then, pretend to be another user, execute the application in the cloud and access the results.

2.2. Use Case

The implemented use case is briefly outlined here and described in full detail in chapter Implementation Challenge, section UseCase. The use case describes a burn scare mapping scenario in Canada, where the extent of wild fires is analyzed based on radar data processing. The use case required to process a collection of Sentinel-1 radar data for the Northwest Territories (NWT) at the beginning and the end of the 2017 summer. Large wildfires occurred in 2013, 2014 and 2015 in this region, and the application should allow spatially-extensive, timely

and cost effective wildfire mapping and monitoring to better assess post-fire burns and tracking of impacts of disturbances on forestlands.

Figure 1 — The use case required to process radar data over Canada's Northwest Territories, a land area of approximately 1,144,000 km2 with a population of less than 50,000 people

anada's forests cover an area of 348 million hectares, which is 35% of Canada's land area and 9% of the world's forested area. Because vast areas are inaccessible, researchers use satellites such as Sentinel-1 to gain valuable insights into Canada's forest ecosystem. The Hackathon evaluated the extent of wild fires based on Sentinel-1 data for the summer of 2017 over the Northwest Territories, Canada. In total, 128 Sentinel-1 IW images (832 GB) for the two coverages (beginning and end of summer 2017) of the entire NWT had to be processed.

2.3. What was provided

Organizers provided the Sentinel Application Platform (SNAP) Software Toolbox together with a pre-defined workflow packaged in a Docker container. Hackathon participants could use the Docker container "as is", i.e., there was no need to modify the container or the application. Given that the container included both SNAP and an executable SNAP graph, participants could on the hand execute the application independently of the available Docker image to test alternatives to the Testbed-13 solutions.

The SNAP workflow in the Docker container required two types of data: Digital Elevation Model (DEM) data and Sentinel-1 data. Both have been made available to the participants as cloud

resources. Natural Resources Canada provided a Canadian Digital Elevation Model (CDEM) for terrain correction due to lack of SRTM over 60° North, ESA provided 128 Sentinel-1 scenes.

2.4. Requested Solution

The requested solution is defined in full detail in chapter Implementation Challenge and only briefly outlined here. The requested solution was based on the Testbed-13 results, which are illustrated in the figure below and briefly described in the following paragraphs. There was no exclusiveness, though. Participants could develop their own solution that deviated from the Testbed-13 base.

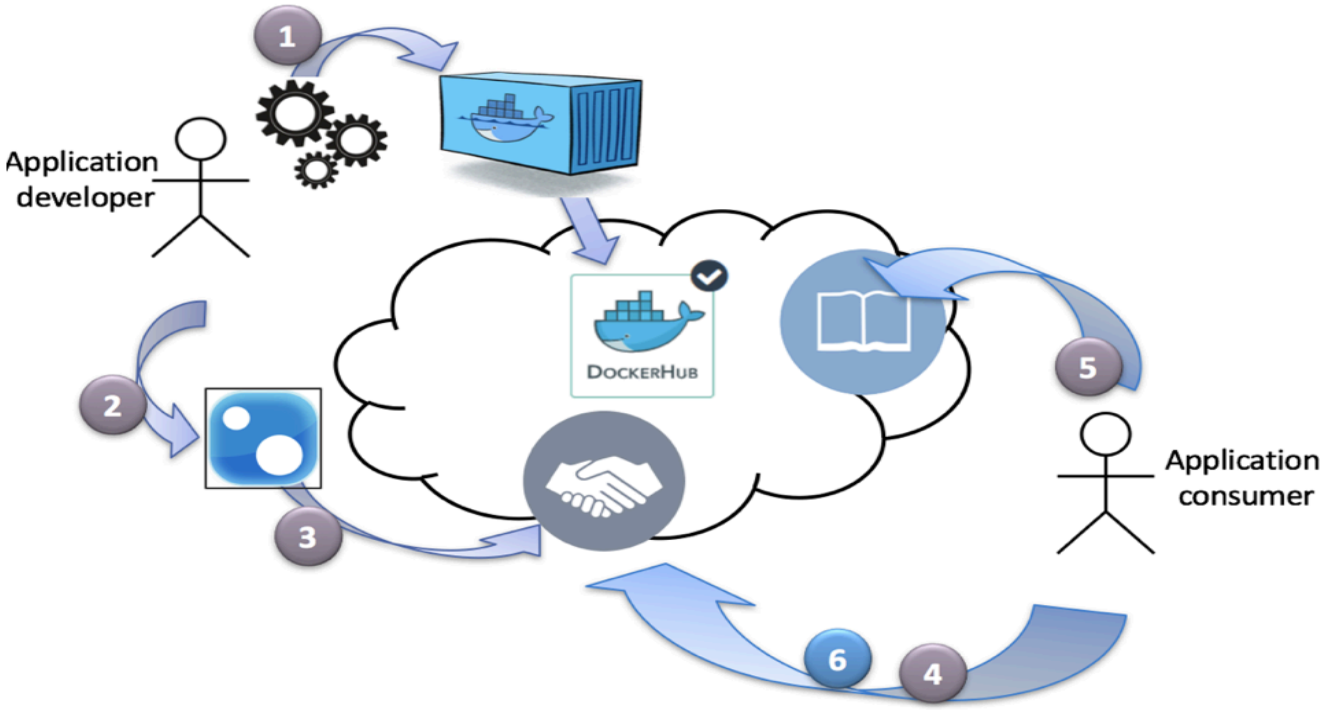


Figure 2 — Testbed-13 Architecture

The application developer in the upper left corner packages the application together with all required libraries into a Docker container (1) and describes it following the Application Package specification (2). The Application Package will then be registered with the Application Deployment and Execution Services, ADES (3). The ADES provides a WPS interface. The ADES registers the new application and makes it available as a new WPS process. On request from an application consumer (4), the ADES provides a description of all parameters required to be provided as part of an execution request. On execution (6), the ADES deploys the application container on a cloud and executes it. Once done, the application consumer is provided with instructions on how to access the results. The discovery process of the new process (5) that allows the discovery of the new process in a catalog service is ignored in the Hackathon.

2.5. Possible Deviation

Participants have been free to deviate from the architecture outlined above as long as the following requirements are met:

- the application developer can make an application available in a container to the cloud platform;
- the application can be executed with a simple WPS execute() request, i.e., mounts the input data to the Docker mount points automatically, downloads necessary data, executes all processing steps, stores the results persistently on the cloud, and informs the user upon completion of the process; and
- any consumer can discover the application and request its deployment and execution in the cloud.

2.6. Hackathon Participants

The following organizations participated in the Hackathon as sponsors, organizers, participants, cloud providers, or observers.

Table 1 — Participating organizations. Organizations marked with '*' participated in the Testbed-13 Earth Observation Cloud activities.

ESA*	52north	West University of Timisoara
NRCan*	Eurac Research	VITO
OGC*	Bind to service	Institute e-Austria
Solenix Deutschland GmbH*	CloudSigma	Space Applications Services
C-S	EUMETSAT	Thales Alenia Space

3

KEY FINDINGS

3 KEY FINDINGS

This chapter reviews architectural aspects that have been discussed at the Hackathon most intensively. Some of them identify potential alternative approaches to the solutions developed in Testbed-13, others help to answer open questions that had been identified at the end of Testbed-13.

3.1. Existing Architecture

The overall experience with the implementation based on Testbed-13 results are good.

3.1.1. WPS

It was generally agreed that it makes perfect sense to facade any type of application with a WPS. Though, mixed opinions have been stated on the actual location of the WPS, which could be a service offered by the ADES, a service that is available per MEP, or integral part of the Docker container itself (see microservice based approach). The WPS-based solutions all worked well and provided sufficient flexibility.

3.1.1.1. WPS RESTful

It was emphasized that all participants prefer an OpenAPI 3.0 RESTful interface rather than the Service Oriented Architecture Remote Procedure Call approach described in the current standard WPS 2.0. Participants said that it is much easier and better supported by available tools to interact with a WPS instance that way.

3.1.1.2. WPS Interpretation of New Process Deployment

It turned out that the WPS mechanisms to deploy or register new processes have been perceived differently by the participants. In principle, two options exist to add a new process to an WPS instance: First, to use a dedicated operation, second, to use the Execute operation with specific parameterization.

WPS operations are standard built-in functions supported at the service level (e.g., *GetCapabilities*, *DescribeProcess*, or *Execute*). In a transactional WPS, these can be extended by additional operations such as *Deploy* and *Undeploy* to allow registering new processes. WPS processes are application specific functions only available in certain WPS instances (in their process offerings). Additional processes can also be added by using the Execute operation. Then, the *Execute(New Process)* operation gets specific parameters that indicate that this operation execution shall add a new process to the WPS instance. Due to timing constraints, the latter

approach was used in the Hackathon, though the used approach has no influence on the actual interpretation problem.

As said, it turned out that the *WPS Deploy()* respectively *Execute(NewProcess)* operation was perceived differently by the participants. The intention of these calls is to add a new process to a WPS instance. The WPS advertises this new process as part of its capabilities (section *ProcessOfferings*). Confusion was caused by different interpretations what the WPS is actually doing in the background when adding a new process. In most cases, the WPS might just add this new process to the list of available processes without any further actions. Only when the new process is executed, then the WPS will run the necessary steps to deploy the process on the cloud and starts its execution. Some participants interpreted the word *deploy* in the sense of actual deployment in the cloud, which would cause a whole lot of additional overhead, such as instance control and monitoring in the cloud. It is emphasized that the behavior of the WPS is totally opaque in that sense. Any WPS instance can do with the new process whatever deemed necessary. It is only required that the new process is offered to WPS users.

3.1.2. Application Package

The Application OWS Context Document (OWC) should have a clear, minimum set of mandatory fields required for an application to be valid. This minimum set should than be validated and accepted on all clusters.

Each process should have a unique process identifier. The process identifier is provided within the Application Package (AP) in the <ows:Identifier> element, so it is not generated by the WPS instance usually. It would be possible to generate unique identifiers on the AP production side by using e.g., the SHA-256 message digest of the OWC Application context document. This would help to have a unique, traceable and interchangeable mapping between the OWC document and the process identifier.

OWS-Context vs. WPD vs. QaDJSON

Testbed-13 discussed three options to encode an implementation package:

1. Using OWS-Context with embedded Web Processing Service ProcessDescriptor elements;
2. Using Web Processing Service ProcessDescriptor (WPD) elements exclusively, i.e. avoid OWS-Context; and
3. Using a Quick and Dirty JSON (QaDJSON) representation, either based on some JSON Schema or following conventions.

Then first approach uses OWS-Context to encode the AP. WPS ProcessDescriptor elements are embedded. This approach has the advantage that:

- the developer can add any additional information to the AP that can be used by sophisticated clients to visualize e.g., background maps as part of the graphical consumer interaction interface;

- the developer can add specific mappings to the AP file which might be necessary in order to work with different catalogs (that the application consumer can use to identify the data to be processed). The mappings are necessary because many OpenSearch-based catalogs use different terminology for both request and response parameter (see section OpenSearch Based Catalogs for further details).
- We remain independent of WPS. If, for whatever reason, we need to remove WPS in the future to replace it with some new technology, this can be easily done without breaking the overall concept of APs and AP-handling services.

On the disadvantage side, one can argue that OWS-Context adds additional complexity to the AP that is not absolutely necessary. It requires the AP developer to read and understand yet another standard without gaining much. After some discussion, this aspect has been solved by agreeing on two aspects. Either tools will be developed that guide the AP developer through the process. Think of a kind of wizard that requests essential elements from the AP developer and produces the AP as a result. In this case, the application developer never sees the actual AP code. Alternatively, application developers will follow examples of existing APs and realize that the OWS-Context add some elements, but do not cause any harm during AP development. In any case, we need to have a feedback channel for AP developers to ensure that the additional flexibility gained with OWS-Context comes at minimum costs for the AP developer.

3.2. Alternative Solutions

3.2.1. Pre-Defined Product Execution

An interesting alternative that was presented by defining a single Docker image loaded with major image processing software like GDAL, OrfeoToolBox and SNAP, instead of providing a link to a Docker image including a processing command. This image can then be reused for various processes, as described here. The approach would lead to modifications in the Application Package model and may lack from version incompatibilities of the used tools and dependent libraries, but is certainly worth further experimentation.

3.2.2. Microservice Approach

The currently favored architectural approach uses Application Package for application description and deployment; and WPS for process registration, deployment and execution. It has been developed based on the target to minimize extra work for the application developer. All the application developer needs to do is to run a simple command line command to package the application in a Docker container, to upload this container to a Docker hub and to provide the application package.

A very interesting alternative approach follows a different paradigm. In the microservice approach, the developer is familiar with WPS and provides everything required to execute the application in the cloud programmatically. Using IAAS/PAAS solutions such as Kubernetes or

Amazon, the user produces a Docker container with a single process WPS embedded. This solution allows the application developer to take care of all data mountings and mappings following his own preferences. The Docker container could then be deployed on cloud platforms.

The approach was developed by several participants, e.g., by VITO as described in more detail in this chapter, or Space Applications. Potential disadvantages of that approach, including parallelization and resource scheduling issues as well as versioning issues, have been identified by Eurac.

3.3. Further Observations

3.3.1. OpenSearch Based Catalogs

It turned out that many OpenSearch based catalogs use different terminology for both request and response parameters. As an example, some catalogs require the request parameter “start”, where others require it to be “startDate”. The same applies to the responses, as illustrated in the following examples.

The following queries are requesting the same data. Note the different parameter names: VITO: http://www.vito-eodata.be/openSearch_all/findProducts?collection=urn:eop:VITO:CGS_S1_SLC_L1&start=2018-01-01

CloudFerro: http://finder.eocloud.eu/resto/api/collections/Sentinel1/search.atom?startDate=2018-01-01&_pretty=true

This issue can be mitigated by using URL templates as provided in the OpenSearch Description (OSD) documents. Given that only the name of the placeholders is standardized (such as searchTerms, geo:box, time:start, time:end), not the rest of the URL (which is completely free, including the base path and the name of the parameters), a link to an OSD document could be provided together with the list of key/values pairs for substituting the search placeholders. This link to would be provided instead of providing an OpenSearch request directly.

A major issue are inhomogeneous responses. It is currently impossible for a developer to produce a response module that could process responses from arbitrary catalogs. The response structure is always different, the response terminology is non-homogeneous, and the actual paths to the data differ from one cloud platform to the other. OWS-Context provides some mitigation options, as at least terminology mappings can be described, but standardization should address this issue in a more robust way by either developing a set of conventions, or even better to include commonly used terms in the specification directly. It cannot be expected that mediation and mapping technologies coming from the Semantic Web can solve this issue within the foreseeable future.

3.3.2. Networks, Firewalls, CORS, Security

Network connectivity (firewalls, blocked ports) and CORS (Cross-Site Scripting protection) issues took the first half day to be resolved. In particular CORS needs to be resolved on a general level, as it affects all secured interactions between clients and servers and can be considered a general issue of Spatial Data Infrastructures. It is not specific to the technology or setup used in the Hackathon.

In terms of general security (which was not in focus of the Hackathon), initial tests have shown that it requires solid best practices to setup a new secure WPS instance as part of an application-in-the-cloud-deployment scenario. Currently, security settings and requirements are very heterogeneous, thus making it hard to implement secure solutions.

3.3.3. Kubernetes, Fargate, EKS, Mesos, Docker, Marathon

There is lots of dynamic in the cloud platform market at the moment. Tools that have been state of the art just a little while ago a now superseded by tools that are even simpler to use, more user friendly, better scalable, cheaper, more robust, or better integrated with other services. At the Hackathon, several of these technologies have been discussed. The following table gives a brief overview of these technologies. Discussing and comparing these technologies is not simple. There are lots of articles or blog posts and an endless amount of social chatter available describing and comparing these technologies. You find a lot of information, often infused with marketing jargon and consultation offers, on what some call the fight-to-the death for container supremacy whereas others claim that the various technologies often solve different things and are rooted in very different contexts. If there is one thing to learn, then certainly to be very careful before building any complex system that is strictly bound to one or a specific set of technologies, tools, or platforms.

Table 2 — cloud technologies

Docker	Docker groups some of the capabilities of Linux cgroups with namespaces into a single and easy to use package that allows applications to run consistently on any infrastructure. The package is called the Docker image, which allows to package the application together with required libraries into a single container.
EKS	Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed Kubernetes service. It uses Amazon Identity and Access Management (IAM) for role based access control, PrivateLink to reach your masters, and Amazon Virtual Private Cloud (VPC) for pod networking.
Fargate	Fargate is an Amazon technology to run containers, either orchestrated by ECS or Kubernetes on EKS, without having to manage the underlying servers or

	clusters (i.e. EC2 instances.), which makes containers a first-class resource.
Kubernetes	Kubernetes is a clustered container orchestration system that automates the creation, replication, scaling and management of Docker containers.
Marathon	Marathon is a production-grade container orchestration platform for Mesosphere's Datacenter Operating System (DC/OS) and Apache Mesos.
Mesos	Apache Mesos abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to easily be built and run effectively.

When investigating favored technologies that are used in applied research projects or system developments, it seems that Docker is the common denominator. Docker containers running in production environments need to be orchestrated across multiple machines; and with the orchestration engines things divert. One of the first orchestration engines was Marathon on top of Apache Mesos. Now we do have Nomad, Kubernetes, or Docker Swarm, which is now part of Docker Engine.

The container technology Docker (i.e., the file format and runtime engine) was quickly complemented with additional technologies, such as Docker Hub, Docker Registry, Docker Cloud or Docker Datacenter. Docker Hub allows public storage of Docker images whereas the Docker Registry can be used for storing it on-premise. Docker Cloud is a managed service for building and running containers and the Docker Datacenter is a commercial offering embodying many Docker technologies. And quickly we are on the market. That's where the jargon and market speak starts and true believers espousing their faith are burning heretics who would dare to consider alternatives.

Kubernetes is a technology introduced by Google. It allows to orchestrate Docker containers without having to interact with the underlying infrastructure. It provides a standard deployment interface and primitives for a consistent app deployment experience and APIs across clouds, which can be very useful in our context of deploying and executing applications working on Earth observation data stored on mission exploitation platforms. Kubernetes modular API allows to integrate systems around the core Kubernetes technology. Experiences with Kubernetes, that is now offered as a service by many vendors, show that it could be a viable solution for EO Exploitation Platforms. It would free the application developer from the underlying infrastructure, though it is not quite clear what effort would be required on the platform side to setup and run Kubernetes deployments.

An alternative for infrastructure abstraction is Apache Mesos.

4

IMPLEMENTATION CHALLENGE

4 IMPLEMENTATION CHALLENGE

This chapter provides more background information on the implemented use case and the data and SNAP graph provided to the participants.

4.1. Use Case

Canada's forests cover an area of 348 million hectares, which is 35% of Canada's land area and 9% of the world's forested area. Because vast areas are inaccessible, researchers use satellites such as Sentinel-1 to gain valuable insights into Canada's forest ecosystem. The Hackathon evaluated the extent of wild fires based on Sentinel-1 data for the summer of 2017 over the Northwest Territories (NWT), Canada. The NWT cover an area of more than 1,300,000 km2 (500,000 sq. mi), with 28 million hectares of largely inaccessible forestlands. There is a large climate variant from north to south, with subarctic climate to polar climate, causing short and cool (mid-teens Celsius) summers and long and harsh winters. Large wildfires occurred in the area in 2013, 2014 and 2015. Given the enormous extent of that area and its inaccessibility, researchers depend on satellite data to assess post-fire burns, which are an essential element in understanding the impacts and disturbances on forestlands caused by fires.

4.2. Satellite and Terrain Data

Sentinel-1 provides an excellent opportunity to investigate C-band SAR information for mapping wildfires in NWT. ESA provided 128 Sentinel-1 Interferometric Wide Swath images (832 GB) for two coverages (beginning and end of summer 2017) of the entire NWT. The Interferometric Wide (IW) swath mode is the main acquisition mode over land and satisfies the majority of service requirements. It acquires data with a 250km swath at 5m by 20m spatial resolution (single look).

Natural Resources Canada provided a Canadian Digital Elevation Model (CDEM) for terrain correction due to lack of SRTM over 60° North.

4.3. Process

NRCan provided a SNAP graph performing that executed 5 tasks in order to generate a GeoTIFF file per satellite scene. Additional Digital Elevation Model data is used for terrain correction. Overall, the process looked like illustrated in the figure below.

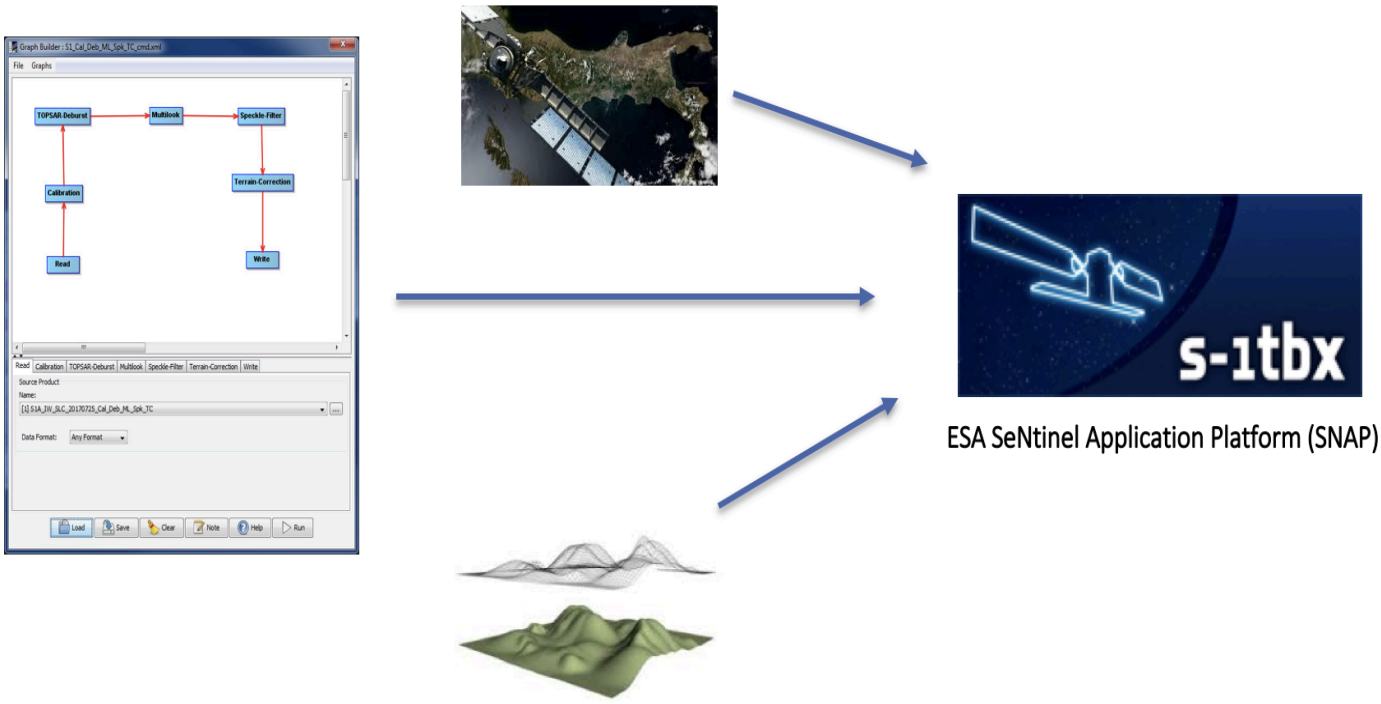


Figure 3 — SNAP Graph performing 5 tasks, using a single Sentinel-1 scene and DEM data for terrain correction

The output is a GeoTiff file that allows the identification of burn scars. The necessary mosaicking, i.e., the merging of all 128 scenes into a single image was not part of the Hackathon. A sample output file is illustrated below.

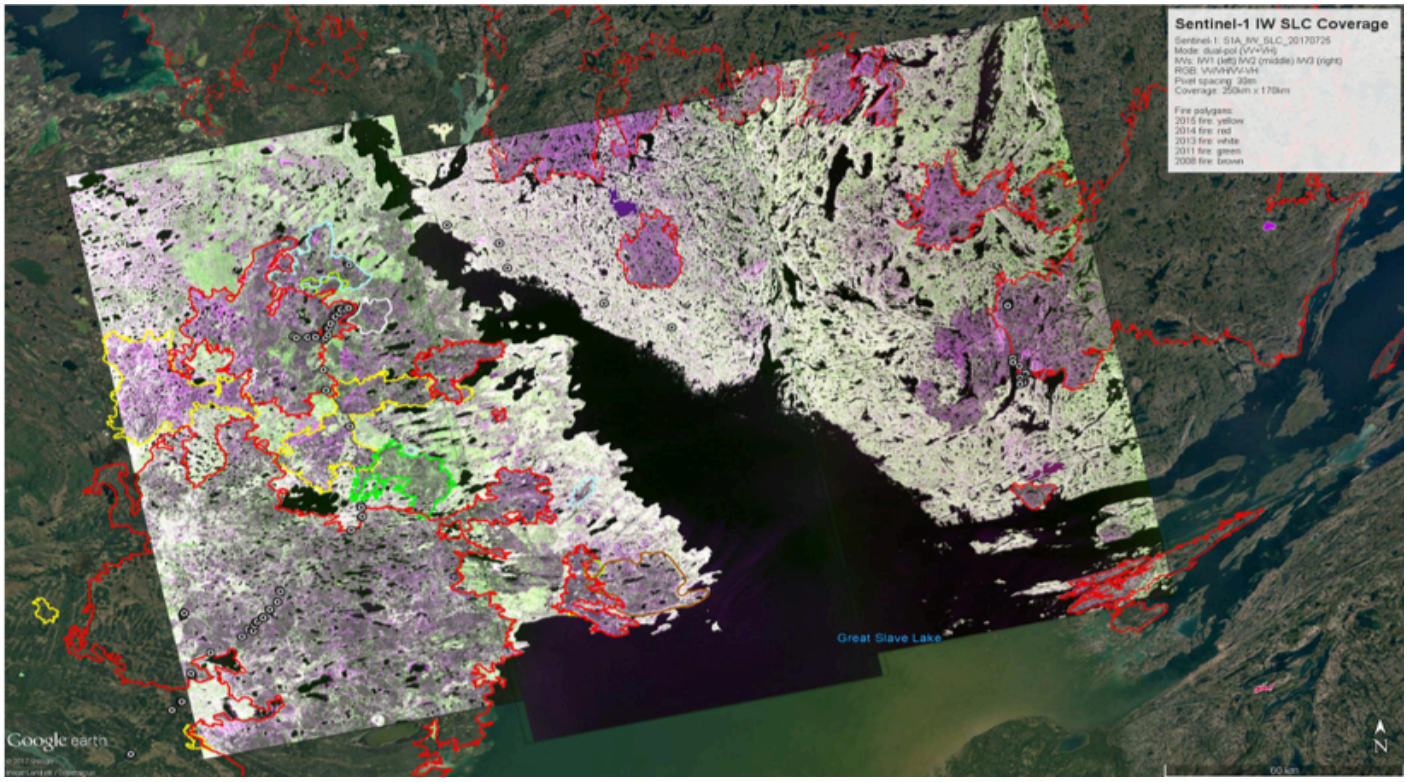


Figure 4 — SNAP Graph performing 5 tasks, using a single Sentinel-1 scene and DEM data for terrain correction

5

SOLUTIONS

5 SOLUTIONS

The following sub-chapters provide the participants' perspective and experiences.

5.1. 52°North

5.1.1. Motivation to Participate

52°North is involved in several projects dealing with processing of large amounts of Earth Observation data. For example, the research project WaCoDis aims to implement a geo-information infrastructure for river basin management monitoring tasks including water quality control, water protection and protection of access to clean water. For this purpose, remote sensing data from the Copernicus program, weather data and in-situ sensor data will be combined, merged and analyzed. Another example is the RIESGOS research project, which aims to implement multi-risk assessment from natural hazards by utilizing standardized service interfaces.

Furthermore, 52°North has long-time experience in defining and implementing Web-based Geoprocessing tools and is actively contributing to standardizing geoprocessing. Benjamin Pross is currently leading the Web Processing Service 2.0 SWG at OGC. The 52°North WPS implementation fully supports WPS 1.0.0 and 2.0 and is widely used. It has also been used in Testbed-13 Earth Observation Clouds thread.

Since 52°North provides a WPS implementation and is involved in different projects dealing with Earth Observation data, 52°North has a high interest in the activities for standardizing the processing of large amounts of Earth Observation data and wants to contribute with its experience in developing Web-based Geoprocessing systems.

5.1.2. Implemented Solution

52°North has implemented an Application Deployment and Execution Service based on the 52°North javaPS framework. The framework fully supports the WPS 2.0 interface. The means to deploy and undeploy processes were implemented as WPS process themselves. The process descriptions were aligned to the ones specified in the OGC Testbed-13: Application Deployment and Execution Service Engineering Report (OGC 17-024). Figure 5 shows the general sequence of communication between client and server. The requests are described in the following.

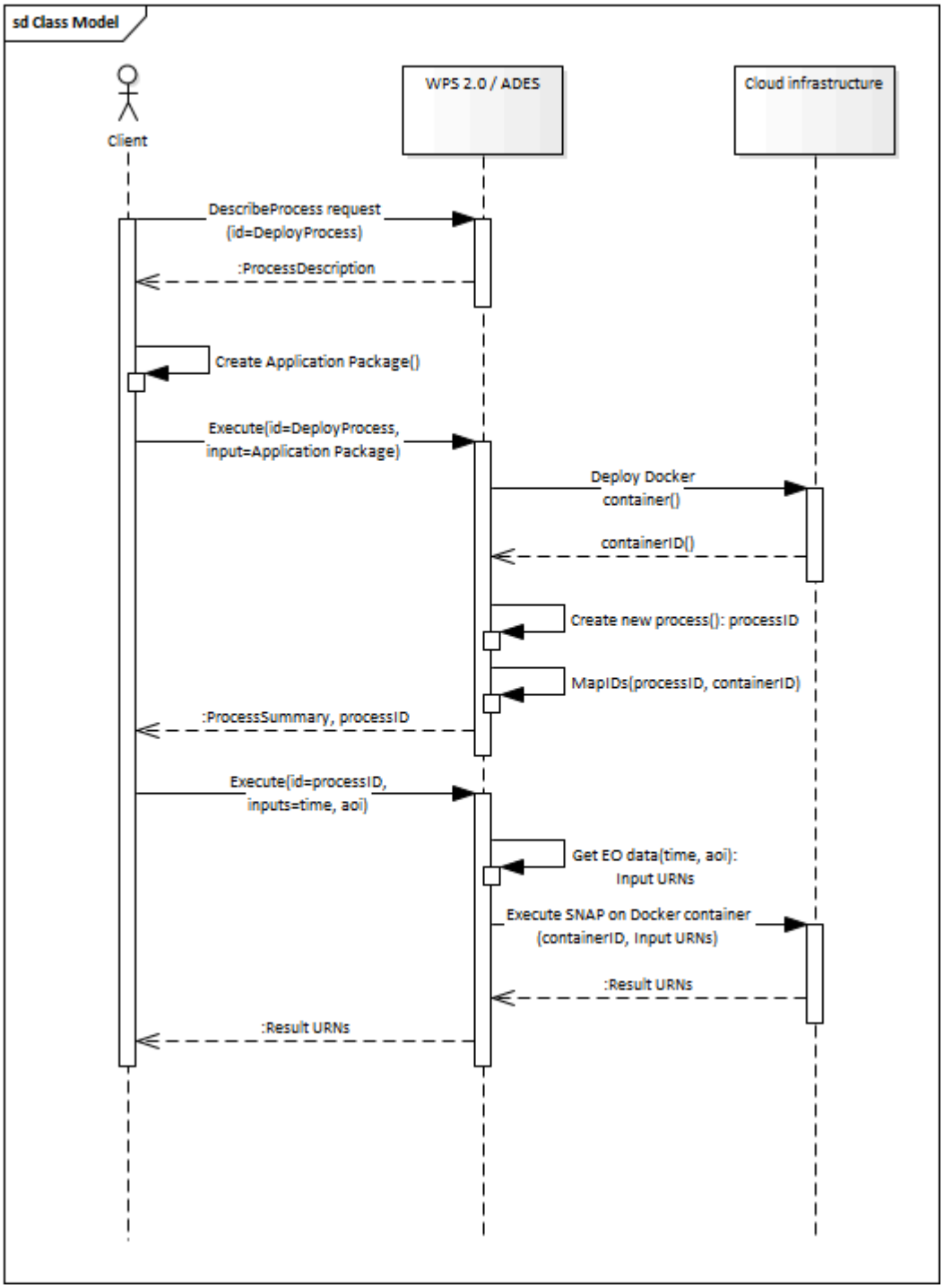


Figure 5 — Sequence of interaction between client and 52°North WPS 2.0 with ADES-specific extension

DeployProcess description

```
<?xml version="1.0" encoding="UTF-8"?><wps:ProcessOfferings 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://
schemas.opengis.net/wps/2.0/wps.xsd">
```

```
  <wps:ProcessOffering jobControlOptions="async-execute dismiss sync-execute"
 outputTransmission="value reference" processVersion="1.0.0" processModel=
"native">
    <wps:Process xsi:schemaLocation="http://www.opengis.net/wps/2.0 http://
schemas.opengis.net/wps/2.0/wps.xsd">
      <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>
      <ows:Identifier>DeployProcess</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <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>
        <ows:Identifier>applicationPackage</ows:Identifier>
        <wps:ComplexData>
          <wps:Format mimeType="application/atom+xml" default="true"/>
          <wps:Format mimeType="application/atom+xml"/>
        </wps:ComplexData>
      </wps:Input>
      <wps:Output>
        <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>
        <ows:Identifier>deployResult</ows:Identifier>
        <wps:ComplexData>
          <wps:Format mimeType="text/xml" schema="https://raw.
githubusercontent.com/bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/
main/resources/META-INF/xml/wps/2.0/wpsDeployResult.xsd" default="true"/>
          <wps:Format mimeType="text/xml" schema="https://raw.
githubusercontent.com/bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/
main/resources/META-INF/xml/wps/2.0/wpsDeployResult.xsd"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

UndeployProcess description

```
<?xml version="1.0" encoding="UTF-8"?><wps:ProcessOfferings 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://
schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering jobControlOptions="async-execute dismiss sync-execute"
 outputTransmission="value reference" processVersion="1.0.0" processModel=
"native">
    <wps:Process xsi:schemaLocation="http://www.opengis.net/wps/2.0 http://
schemas.opengis.net/wps/2.0/wps.xsd">
      <ows:Title>Undeploy Process</ows:Title>
      <ows:Abstract>This method removes a previously deployed process from the
WPS.</ows:Abstract>
      <ows:Identifier>UndeployProcess</ows:Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Process Identifier</ows:Title>
        <ows:Abstract>The identifier of the process to remove from the WPS.</
ows:Abstract>
        <ows:Identifier>processIdentifier</ows:Identifier>
        <wps:LiteralData>
          <wps:Format mimeType="application/xml" default="true"/>
          <wps:Format mimeType="text/xml"/>
```

```
          <wps:Format mimeType="text/plain"/>
          <wps:Format mimeType="text/plain" encoding="base64"/>
          <LiteralDataDomain default="true">
            <ows:AnyValue/>
            <ows:DataType ows:reference="https://www.w3.org/2001/XMLSchema-
datatypes#string">string</ows:DataType>
            <ows:DefaultValue/>
          </LiteralDataDomain>
        </wps:LiteralData>
      </wps:Input>
      <wps:Output>
        <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>
        <ows:Identifier>undeployResult</ows:Identifier>
        <wps:ComplexData>
          <wps:Format mimeType="text/xml" schema="https://raw.
githubusercontent.com/bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/
main/resources/META-INF/xml/wps/2.0/wpsUndeployResult.xsd" default="true"/>
          <wps:Format mimeType="text/xml" schema="https://raw.
githubusercontent.com/bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/
main/resources/META-INF/xml/wps/2.0/wpsUndeployResult.xsd"/>
        </wps:ComplexData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

Based on the process descriptions, execute requests could be send to the ADES.

DeployProcess execute request

```
<wps200:Execute xmlns:atom="http://www.w3.org/2005/Atom" xmlns:dc="http://purl.
org/dc/elements/1.1/" xmlns:georss="http://www.georss.org/georss" xmlns:gml311=
"http://www.opengis.net/gml" xmlns:owc10="http://www.opengis.net/owc/1.0"
xmlns:ows200="http://www.opengis.net/ows/2.0" xmlns:wps200="http://www.opengis.
net/wps/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" mode="sync"
 response="document" service="WPS" version="2.0.0" xsi:schemaLocation="http://
www.opengis.net/wps/2.0 http://schemas.opengis.net/wps/2.0/wps.xsd">
<ows200:Identifier>DeployProcess</ows200:Identifier>
  <wps200:Input id="applicationPackage">
    <wps200:Data mimeType="application/atom+xml">
      <atom:feed xml:lang="en">
        <atom:link href="http://www.opengis.net/spec/owc-atom/1.0/req/core"
rel="profile" title="This file is compliant with version 1.0 of OGC Context" />
        <atom:link href="http://www.opengis.net/tb13/eoc" rel="profile" title=
"This file is compliant with Testbed-13 EOC Thread for Application Packing" />
        <atom:id>eoeph18-detectburnedareastestingexceptionreport-1_f4511258-
cac0-4955-b952-db6944cf7d4d</atom:id>
        <atom:title>DetectBurnedAreas.testing.ExceptionReport</atom:title>
        <atom:subtitle type="text" />
        <atom:updated />
        <atom:author>
          <atom:email />
        </atom:author>
        <atom:rights>OGC EOEP Hackathon 2018</atom:rights>
        <georss:where>
          <gml311:Polygon>
            <gml311:exterior>
              <gml311:LinearRing>
                <gml311:posList>-90 -180 90 -180 90 180 -90 180 -90 -180</
gml311:posList>
```

```
              </gml311:LinearRing>
            </gml311:exterior>
          </gml311:Polygon>
        </georss:where>
        <dc:date>2005-01-01T09:08:56.0000000Z/2020-01-23T09:14:08.0000000Z</dc:
date>
        <atom:entry>
          <atom:id>eoeph18-detectburnedareastestingexceptionreport-1_f4511258-
cac0-4955-b952-db6944cf7d4d</atom:id>
          <atom:link href="http://www.opengis.net/tb13/eoc/application" rel=
"profile" title="This entry contains an application as specified by Testbed-13
EOC Thread" />
          <atom:title>DetectBurnedAreas.testing.ExceptionReport</atom:title>
          <atom:content type="text">Process deployed through ASB platform</
atom:content>
          <owc10:offering code="http://www.opengis.net/tb13/eoc/docker">
            <owc10:content type="text/plain">www.dockerhub.com/eoeph18-
wildfires-detector:latest</owc10:content>
          </owc10:offering>
          <owc10:offering code="http://www.opengis.net/tb13/eoc/
wpsProcessOffering">
            <owc10:content type="application/xml">
             <wps200:ProcessOfferings>
              <wps200:ProcessOffering>
                <wps200:Process>
                  <ows200:Title>DetectBurnedAreas.testing.ExceptionReport</
ows200:Title>
                  <ows200:Abstract>DetectBurnedAreas.testing.ExceptionReport</
ows200:Abstract>
                  <ows200:Identifier>eoeph18-detectburnedareastestingexceptionr
eport-1_f4511258-cac0-4955-b952-db6944cf7d4d</ows200:Identifier>
                  <wps200:Input maxOccurs="1" minOccurs="1">
                    <ows200:Title>Area Of Interest</ows200:Title>
                    <ows200:Abstract />
                    <ows200:Identifier>AreaOfInterest</ows200:Identifier>
                    <wps200:LiteralData>
                      <wps200:Format default="true" encoding="" mimeType="text/
plain" schema="" />
                    </wps200:LiteralData>
                  </wps200:Input>
                  <wps200:Input maxOccurs="1" minOccurs="1">
                    <ows200:Title>Time Of Interest</ows200:Title>
                    <ows200:Abstract />
                    <ows200:Identifier>TimeOfInterest</ows200:Identifier>
                    <wps200:LiteralData>
                      <wps200:Format default="true" encoding="UTF-8" mimeType=
"text/plain" schema="" />
                    </wps200:LiteralData>
                  </wps200:Input>
                  <wps200:Output>
                    <ows200:Title>Result URI</ows200:Title>
                    <ows200:Abstract>URI pointing to a web-accessible folder
containing the processed images.</ows200:Abstract>
                    <ows200:Identifier>resultURI</ows200:Identifier>
                    <wps200:LiteralData>
                      <wps200:Format default="true" encoding="UTF-8" mimeType=
"text/plain" schema="" />
                    </wps200:LiteralData>
                  </wps200:Output>
                </wps200:Process>
              </wps200:ProcessOffering>
             </wps200:ProcessOfferings>
            </owc10:content>
```

```
          </owc10:offering>
          <atom:category label="This app runs in Linux" scheme="http://www.
opengis.net/tb13/eoc/os" term="LINUX" />
        </atom:entry>
      </atom:feed>
    </wps200:Data>
  </wps200:Input>
</wps200:Execute>
```

UndeployProcess execute 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://schemas.
opengis.net/wps/2.0/wps.xsd"
        service="WPS" version="2.0.0" response="document" mode="sync">
        <ows:Identifier>UndeployProcess</ows:Identifier>
        <wps:Input id="processIdentifier">
                <wps:Data mimeType="text/xml">
                <wps:LiteralValue>http://www.opengis.net/eoephack2018/burnscar
</wps:LiteralValue>
                </wps:Data>
        </wps:Input>
        <wps:Output id="undeployResult" transmission="value"
                mimeType="text/xml" schema="https://raw.githubusercontent.com/
bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/main/resources/META-INF/
xml/wps/2.0/wpsUndeployResult.xsd"/>
</wps:Execute>
```

Based on the Application Package, that is send along as execute-input, a new process will be created and made available in the capabilities. The process summary will be returned in the execute response.

DeployProcess execute response

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Result 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://schemas.opengis.net/wps/2.0/wps.xsd">
    <wps:JobID>fb410108-775e-4bdc-8138-f971be428362</wps:JobID>
    <wps:ExpirationDate>2018-05-15T12:33:35Z</wps:ExpirationDate>
    <wps:Output id="deployResult">
        <wps:Data mimeType="text/xml" encoding="UTF-8" schema="https://raw.
githubusercontent.com/bpross-52n/common-xml/project/eoep/52n-ogc-schema/src/
main/resources/META-INF/xml/wps/2.0/wpsDeployResult.xsd">
            <ns:DeployResult xmlns:ns="http://www.opengis.net/wps/2.0">
                <DeploymentDone>true</DeploymentDone>
                <ProcessSummary xmlns:ows="http://www.opengis.net/ows/
2.0" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xlink="http://www.
w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
processModel="native" outputTransmission="value reference" processVersion=
"1.0.0" jobControlOptions="async-execute dismiss sync-execute">
                    <ows:Title>Detect burn scars</ows:Title>
                    <ows:Abstract>This process will detect burned areas within
a specified area of interest and during a specified time.</ows:Abstract>
                    <ows:Identifier>http://www.opengis.net/eoephack2018/
burnscar</ows:Identifier>
                </ProcessSummary>
            </ns:DeployResult>
```

```
        </wps:Data>
    </wps:Output>
</wps:Result>
```

The complete process description looks like the following:

Description of the newly deployed process

```
<?xml version="1.0" encoding="UTF-8"?><wps:ProcessOfferings 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://
schemas.opengis.net/wps/2.0/wps.xsd">
  <wps:ProcessOffering jobControlOptions="async-execute dismiss sync-execute"
 outputTransmission="value reference" processVersion="1.0.0" processModel=
"native">
    <wps:Process xsi:schemaLocation="http://www.opengis.net/wps/2.0 http://
schemas.opengis.net/wps/2.0/wps.xsd">
      <ows:Title>Detect burn scars</ows:Title>
      <ows:Abstract>This process will detect burned areas within a specified
area of interest and during a specified time.</ows:Abstract>
      <ows:Identifier>http://www.opengis.net/eoephack2018/burnscar</ows:
Identifier>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>Time window</ows:Title>
        <ows:Abstract>The time window for the fire scar detection.</ows:
Abstract>
        <ows:Identifier>timeWindow</ows:Identifier>
        <wps:LiteralData>
          <wps:Format mimeType="application/xml" default="true"/>
          <wps:Format mimeType="text/xml"/>
          <wps:Format mimeType="text/plain"/>
          <wps:Format mimeType="text/plain" encoding="base64"/>
          <LiteralDataDomain default="true">
            <ows:AnyValue/>
            <ows:DataType ows:reference="https://www.w3.org/2001/XMLSchema-
datatypes#string">string</ows:DataType>
            <ows:DefaultValue/>
          </LiteralDataDomain>
        </wps:LiteralData>
      </wps:Input>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Title>aoi</ows:Title>
        <ows:Identifier>aoi</ows:Identifier>
        <wps:BoundingBoxData>
          <wps:Format mimeType="application/xml" default="true"/>
          <wps:Format mimeType="text/xml"/>
          <wps:SupportedCRS default="true">http://www.opengis.net/def/crs/EPSG/
0/4326</wps:SupportedCRS>
        </wps:BoundingBoxData>
      </wps:Input>
      <wps:Output>
        <ows:Title>Result URI</ows:Title>
        <ows:Abstract>URI pointing to a web-accessible folder containing the
processed images.</ows:Abstract>
        <ows:Identifier>resultURI</ows:Identifier>
        <wps:LiteralData>
          <wps:Format mimeType="application/xml" default="true"/>
          <wps:Format mimeType="text/xml"/>
          <wps:Format mimeType="text/plain"/>
          <wps:Format mimeType="text/plain" encoding="base64"/>
          <LiteralDataDomain default="true">
            <ows:AnyValue/>
```

```
            <ows:DataType ows:reference="https://www.w3.org/2001/XMLSchema-
datatypes#anyURI">anyURI</ows:DataType>
          </LiteralDataDomain>
        </wps:LiteralData>
      </wps:Output>
    </wps:Process>
  </wps:ProcessOffering>
</wps:ProcessOfferings>
```

The process itself is a mockup, but sending an example execute request is possible:

Example execute request for the newly deployed process

```
<?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://schemas.
opengis.net/wps/2.0/wps.xsd"
        service="WPS" version="2.0.0" response="document" mode="sync">
        <ows:Identifier>http://www.opengis.net/eoephack2018/burnscar</ows:
Identifier>
        <wps:Input id="timeWindow">
                <wps:Data mimeType="text/plain">2017-06-17T/2017-06-28TZ</wps:
Data>
        </wps:Input>
        <wps:Input id="aoi">
                <wps:Data mimeType="text/xml">
                        <ows:BoundingBox crs="http://www.opengis.net/def/crs/
EPSG/0/4326">
                                <ows:LowerCorner>59.913464 -136.448354</ows:
LowerCorner>
                                <ows:UpperCorner>78.794937 -101.931600</ows:
UpperCorner>
                        </ows:BoundingBox>
                </wps:Data>
        </wps:Input>
        <wps:Output id="resultURI" transmission="value" mimeType="text/plain" /
>
</wps:Execute>
```

A single example output TIFF is returned by reference.

Example execute response for the newly deployed process

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Result 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://schemas.opengis.net/wps/2.0/wps.xsd">
    <wps:JobID>df0d4be0-b182-4d9c-93e1-158c74bf3215</wps:JobID>
    <wps:ExpirationDate>2018-05-04T12:03:08Z</wps:ExpirationDate>
    <wps:Output id="resultURI">
        <wps:Data mimeType="text/plain" encoding="UTF-8">
            <![CDATA[http://52north.org/delivery/eoep/S1A_IW_SLC__1SDH_
20170617.tif]]>
        </wps:Data>
    </wps:Output>
</wps:Result>
```

5.1.3. Experiences with AP & ADES

The concept of the ADES works well with WPS and the 52°North WPS could be easily extended with the necessary functionality. WPS 2.0 processes are a good means for deploying Application Packages and creating new WPS processes. The input- and output-formats can be described in a standardized way and generic clients are able to execute the Deploy- and UndeployProcess. The 52°North ADES was used for testing by several client developers during the Hackathon. It supports the deployment and execution of new processes based on Application Packages. The back-end implementation, i.e. deploying the Docker container specified in the Application Package required more time than initially anticipated, so we had to create a mock-up for this functionality. In the OGC Testbed-13: Application Deployment and Execution Service Engineering Report two approaches for implementing an ADES are described. One approach is using WPS processes. This was implemented during the EOEP Hackathon. An alternative approach used a transactional extension for WPS 2.0. In the OGC Testbed-14 different approaches for enabling WPS with transactional functionality are further explored. The Testbed-14 WPS-T ER will specify how transactional behavior can be implemented for WPS using additional dedicated operations for deploying and undeploying processes. Also REST-based approaches will be explored. Findings from the EOEP Hackathon will be taken into account for discussing these different approaches and will be considered when developing novel approaches for flexible Web-based processing of EO data in 52°North's ongoing research projects.

5.1.4. Acknowledgement

The contribution of 52°North was supported by the research project WaCoDis (co-funded by the German Federal Ministry of Transport and Digital Infrastructure, program mFund, contract: 19F2038D) and by the research project RIESGOS (co-funded by the German Ministry of Research and Education, program CLIENT-II, contract: 03G0876).

5.2. CS

5.2.1. Motivation to Participate

To meet the needs for storage, processing and distribution of products to customers, CS is developing the GeoStorm platform. As part of a Pathfinder project developed for ESA for several years, CS has already integrated some image processing. GeoStorm is developed in accordance with good practices in particular regarding interoperability. We are therefore very attentive to the respect of the OGC standards into our developments.

5.2.2. Implemented Solution

During this Hackathon, we have implemented both client and server. Our client is the GeoStorm web application which provides rich interface to select processing parameters. Implemented WPS 1.0 services are *DeployProcess*, *UndeployProcess* and *ExecuteProcess*.

We didn't choose the Kubernetes approach as we did on others similar projects because of our preoccupation to deploy our solution on architectures like HPC. We therefore have chosen to implement a docker like solution that is not requiring user's privilege increase and allow the image to be used for several tasks. We were interesting in verifying that this implementation meets the requirements of the Application Package and can apply to both new cloud solutions and the traditional ones.

Here is an example of the output mosaic with some processed tiles on the study area displayed into GeoStorm platform:

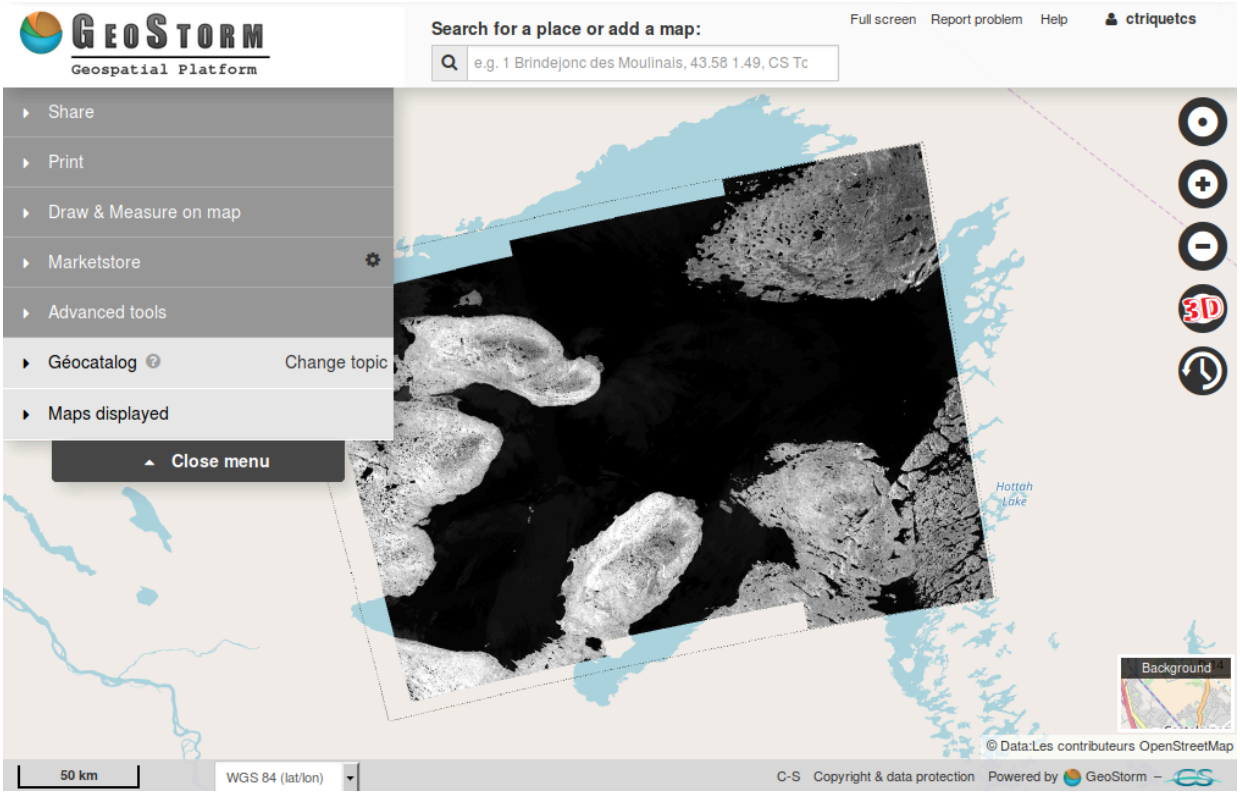


Figure 6

5.2.2.1. Application package

To meet our needs to deploy easily on HPC's like architecture using Torque distributing computing system, our implementation slightly differs from the OGC testbed architecture. Instead of providing a link to a Docker image including a processing command, we deploy an unique image including major image processing softwares like GDAL, OrfeoToolBox and SNAP.

That image can then be used for various processes. We therefore require to define in our Application Package the processing command to execute. The “wps:Command” tag provides that command. To transfer execution parameters, some placeholders are dynamically replaced at execution time. That solution seems simpler than using environment variables.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed
   xmlns="http://www.w3.org/2005/Atom"
   xmlns:wps="http://www.opengis.net/wps/2.0"
   xmlns:ows="http://www.opengis.net/ows/2.0"
   xmlns:owc="http://www.opengis.net/owc/1.0" xml:lang="en">
   <title>Forest Fire Application Package</title>
   <subtitle type="text">ForestFire APP PKG</subtitle>
   <updated>2018-05-01T12:10:00Z</updated>
   <link rel="profile" href="http://www.opengis.net/spec/owc-atom/1.0/req/core"
         title="This file is compliant with version 1.0 of OGC Context"/>
   <link rel="profile" href="http://www.opengis.net/tb13/eoc"
         title="This file is compliant with Testbed-13 EOC Thread for
Application Packing"/>
   <author>
      <name>CS</name>
   </author>
   <entry>
      <title>Forest Fire Application</title>
      <updated>2018-05-01T12:10:00Z</updated>
      <content type="text/plain">Some narrative describing the purpose of the
application.</content>
      <!-- DOCKER IMAGE -->
      <owc:offering code="http://www.opengis.net/tb13/eoc/docker">
         <owc:content type="text/plain">docker-co.terradue.com/ows13-eoc/dcs-
stemp-l8:1.0.3</owc:content>
      </owc:offering>
      <!-- THE WPS PROCESS DESCRIPTION -->
      <owc:offering code="http://www.opengis.net/tb13/eoc/wpsProcessOffering">
         <owc:content type="application/xml">
            <wps:ProcessOffering
               jobControlOptions="async-execute dismiss"
               outputTransmission="value reference"
               xmlns:wps="http://www.opengis.net/wps/2.0"
               xmlns:ows="http://www.opengis.net/ows/2.0"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.opengis.net/wps/2.0
                                   http://schemas.opengis.net/wps/2.0/wps.xsd">
               <wps:Process>
                  <ows:Title>Forest Fire</ows:Title>
                  <ows:Abstract>Forest Fire is a method to detect forest fires.
</ows:Abstract>
                  <ows:Identifier>ForestFire</ows:Identifier>
                  <wps:Command>/opt/snap_all_6_0/bin/gpt {{inputDir}}/
ForestFire.xml -Pinputdata={{inputFile}} -Poutputdata={{outputFile}} -Pfilter=
{filter} -PpixelRes={pixelRes}</wps:Command>
                  <wps:Input>
                     <ows:Title>file</ows:Title>
                     <ows:Abstract>The file from which the data product is
read.</ows:Abstract>
                     <ows:Identifier>inputdata</ows:Identifier>
                     <wps:ComplexData>
                         <wps:Format mimeType="application/octet-stream"
default="true"/>
                     </wps:ComplexData>
                  </wps:Input>
                  <wps:Input>
```

```
                        <ows:Title>Speckle Filter</ows:Title>
                        <ows:Abstract>Filter the speckles in the image.</ows:
Abstract>
                        <ows:Identifier>filter</ows:Identifier>
                        <wps:LiteralData>
                           <wps:Format mimeType="text/plain" default="true"/>
                           <LiteralDataDomain>
                              <ows:ValuesReference ows:reference="http://...."/>
                              <ows:DataType
                                 ows:reference="http://www.w3.org/2001/
XMLSchema#string">String</ows:DataType>
                           </LiteralDataDomain>
                        </wps:LiteralData>
                     </wps:Input>
                     <wps:Input>
                        <ows:Title>Pixel Resolution</ows:Title>
                        <ows:Abstract>Resolution of the pixels</ows:Abstract>
                        <ows:Identifier>pixelRes</ows:Identifier>
                        <wps:LiteralData>
                           <wps:Format mimeType="text/plain" default="true"/>
                           <LiteralDataDomain>
                              <ows:ValuesReference ows:reference="http://...."/>
                              <ows:DataType
                                 ows:reference="http://www.w3.org/2001/
XMLSchema#string">float</ows:DataType>
                           </LiteralDataDomain>
                        </wps:LiteralData>
                     </wps:Input>
                     <wps:Output>
                        <ows:Title>Output response</ows:Title>
                        <ows:Identifier>response</ows:Identifier>
                        <wps:ComplexData>
                           <wps:Format mimeType="image/tiff" encoding="raw"
                              default="true"/>
                        </wps:ComplexData>
                     </wps:Output>
                  </wps:Process>
               </wps:ProcessOffering>
            </owc:content>
         </owc:offering>
      </entry>
   </feed>
```

Further improvement

In that implementation, we assume that that the SNAP graph is already deployed on the execution platform. An improvement can consist in including the graph file into the application package constituting a package ready to deploy.

5.2.2.2. Deploy process

Here is the WPS service *DeployProcess* post content including the path to the application package:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:ows="http://www.opengis.net/
ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:wps="http://www.
opengis.net/wps/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.
net/wps/1.0.0/wpsExecute_request.xsd">
```

```
  <ows:Identifier>DeployProcess</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>applicationPackage</ows:Identifier>
      <wps:Reference xlink:href="https://host/path_to/ForestFire_atom.xml" />
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
     <wps:ResponseDocument status="true" storeExecuteResponse="true">
       <wps:Output asReference="false">
         <ows:Identifier>DeployResult</ows:Identifier>
       </wps:Output>
     </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

In our implementation, no server restart is required and the deployed WPS service is dynamically available right after the post.

5.2.2.3. Undeploy process

Here is the WPS service UndeployProcess post content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:ows="http://www.opengis.net/
ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:wps="http://www.
opengis.net/wps/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.
net/wps/1.0.0/wpsExecute_request.xsd">
  <ows:Identifier>UndeployProcess</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>ProcessIdentifier</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>ForestFire</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
     <wps:ResponseDocument status="true" storeExecuteResponse="true">
       <wps:Output asReference="false">
         <ows:Identifier>UndeployResult</ows:Identifier>
       </wps:Output>
     </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

5.3. Eurac Research

5.3.1. Motivation to Participate

We are striving after development of interoperable research independent of cloud processing facilities to foster exchange and repeatability. In particular the Institute for Earth Observation at

EURAC Research has the need to be able to run transparently applications on different clusters, where data and computing resources are available. The proposed Hackathon was perfectly in line with what we currently try to achieve in the in the **H2020 openEO project** (http://openeo.eu), where we are already developing a driver to connect the **openEO API** with the **OGC WC(P)S 2.0** and it processing extension. So participating in this Hackathon gave as the possibility to implement and test web processing services based on processing chains organized in containers and link it to the RESTful services that we are currently defining and developing in the **openEO** project.

5.3.2. Implemented Solution

The solution implemented by **Eurac Research** consists of an ADES server that exposes a set of operations as described in the **OGC Testbeds**, in form of a set of **WPS 1.0.0** compliant operation offerings. Further an interface with the **openEO API** has been started to be implemented to access those web process services in a RESTful way and render it interoperable with the client API's in **python** and **R** that are currently in development.

The reasons to use WPS 1.0.0 for the implementation are:

- To ensure compatibility with the tools currently employed (i.e. GeoServer);
- To ensure compatibility with a larger set of available libraries and tools available; and
- Have a general, lightweight model that can be easily transferred also to WPS 2.0.0 in near future.

The operations actually implemented as a proof of concept and working prototype are the following:

- **DeployProcess**
- **ExecuteProcess**
- **UndeployProcess**
- **GetStatus**
- Get Result (not yet fully implemented)

Those processes are also discoverable via openEO using the /processes endpoint of the RESTful API.

The GetCapabilities result for the service (wpseoproc) implemented:

```
        <wps:ProcessOfferings>
                <wps:Process wps:processVersion="1.0.0">
                        <ows:Identifier>gs:DeployProcessEO</ows:Identifier>
                        <ows:Title>DeployProcess</ows:Title>
                        <ows:Abstract>DeployProcess Operation EURAC Research EO
</ows:Abstract>
                </wps:Process>
                <wps:Process wps:processVersion="1.0.0">
```

```
                        <ows:Identifier>gs:ExecuteProcessEO</ows:Identifier>
                        <ows:Title>ExecuteProcess</ows:Title>
                        <ows:Abstract>ExecuteProcess Operation EURAC Research
EO</ows:Abstract>
                </wps:Process>
                <wps:Process wps:processVersion="1.0.0">
                        <ows:Identifier>gs:GetResultEO</ows:Identifier>
                        <ows:Title>GetResult</ows:Title>
                        <ows:Abstract>GetResult Operation of Process EURAC
Research EO</ows:Abstract>
                </wps:Process>
                <wps:Process wps:processVersion="1.0.0">
                        <ows:Identifier>gs:GetStatusEO</ows:Identifier>
                        <ows:Title>StatusInfo</ows:Title>
                        <ows:Abstract>StatusInfo Operation of Process EURAC
Research EO</ows:Abstract>
                </wps:Process>
                <wps:Process wps:processVersion="1.0.0">
                        <ows:Identifier>gs:UndeployProcessEO</ows:Identifier>
                        <ows:Title>UndeployProcess</ows:Title>
                        <ows:Abstract>UndeployProcess Operation EURAC Research
EO</ows:Abstract>
                </wps:Process>
        </wps:ProcessOfferings>
```

The implementation uses the proposed OWS Context Document (OWC) to describe and deploy an application. This seems a very good standard and convenient way to convey and validate the application capabilities on a given cluster.

The OWC document is passed as parameter to the WPS **DeployProcess** operation offering.

The ADES server, apart exposing the listed WPS services, will also perform the following tasks.

- Store and manage all application related information found in the OWS context Document in a persistent database.
- Evaluate and execute the offerings defined in the OWC document, in particular OWC offerings related to pulling a given container from the repository, executing a given OpenSearch Query on a given catalogue (optional), and executing the defined wpsProcessOffering.
- Upon the **ExecuteProcess** operation, the ADES will also perform the “Staging” operation on the data referenced, i.e., Download the data in a local process defined workspace, which in our implementation is defined as $ApplicationRunStore. The staging is performed based on the policies defined in the ProcessParameters (sections → cloud:.., data:...) and considers, if any defined, the results of the search offering defined in the OWC document. Also the configuration defined in the ‘data:” section of the ProcessParameters are evaluated and translated to data references -→ this allows actually on one side the application developer to have a defined dataset which is foreseen to work with the application, and also allows the user eventually to provide a particular or even different dataset, of course only if this is validated also by the OWC document definitions (i.e. </georss:where>, <dc:date>)

5.3.2.1. DeployProcess

The DeployProcess operation allows to deploy a process defined as an Application OWS Context document (OWC) and has the following Parameters:

Request:

- **OWC:** ATOM-encoded OWS Context Document describing the application package
- **Options:** Control Options (clusterURI=...., etc..)

Response:

- **StatusInfo:** Status of the deployment operation. If successful contains a <ows:Identifier..> tag with the job id.

```
<ows:Identifier xmlns:ows="http://www.opengis.net/ows/2.0">11d621e87b7de5bda4dd
cdb3a8bdbceb1ab90f96674fcd655ff519e9fbb5106e</ows:Identifier>
```

DescribeProcess result:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.
opengis.net/wps/1.0.0"
        xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
        xml:lang="en" service="WPS" version="1.0.0"
        xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.
opengis.net/wps/1.0.0/wpsAll.xsd">
        <ProcessDescription wps:processVersion="1.0.0"
                statusSupported="true" storeSupported="true">
                <ows:Identifier>gs:DeployProcessEO</ows:Identifier>
                <ows:Title>DeployProcess</ows:Title>
                <ows:Abstract>DeployProcess Operation EURAC Research EO</ows:
Abstract>
                <DataInputs>
                        <Input maxOccurs="1" minOccurs="1">
                                <ows:Identifier>OWC</ows:Identifier>
                                <ows:Title>OWC</ows:Title>
                                <ows:Abstract>An ATOM-encoded OWS Context
Document describing the application package</ows:Abstract>
                                <ComplexData maximumMegabytes="4">
                                        <Default>
                                                <Format>
                                                        <MimeType>application/
atom+xml</MimeType>
                                                </Format>
                                        </Default>
                                        <Supported>
                                                <Format>
                                                        <MimeType>application/
atom+xml</MimeType>
                                                </Format>
                                                <Format>
```

```
                                                        <MimeType>application/
xml</MimeType>
                                                </Format>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Supported>
                                </ComplexData>
                        </Input>
                        <Input maxOccurs="1" minOccurs="1">
                                <ows:Identifier>Options</ows:Identifier>
                                <ows:Title>Options</ows:Title>
                                <ows:Abstract>Control Options</ows:Abstract>
                                <LiteralData>
                                        <ows:AnyValue />
                                </LiteralData>
                        </Input>
                </DataInputs>
                <ProcessOutputs>
                        <Output>
                                <ows:Identifier>DeployResponse</ows:Identifier>
                                <ows:Title>DeployResponse</ows:Title>
                                <ComplexOutput>
                                        <Default>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Default>
                                        <Supported>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Supported>
                                </ComplexOutput>
                        </Output>
                </ProcessOutputs>
        </ProcessDescription>
</wps:ProcessDescriptions>
```

The ProcessIdentifier UUID returned, if valid, is usable for subsequent ExecuteProcess and UndeployProcess operations

5.3.2.2. ExecuteProcess

Once an application has been deployed and a valid UUID ProcessIdentifier is available, the ExecuteProcess operation allows to execute the application by means of additional, process specific parameters defined in a JSON format.

The parameters for the operation are the following:

Request:

- **ProcessInstanceIdentifier:** Identifier of the process instance to be executed (UUID)
- **ProcessParameters:** Parameters for the Process in JSON format

- **Options:** Control Options (mode=sync|async, etc..)

The ProcessParameters are organized in the following sections and contain a defined set of switches and configurations applicable to a process:

- **cloud:** Cloud specific parameters (ex., Amount of requested resources, Maximum amount to be spent, Timeout policy for killing a process, Scheduler used, etc.);
- **container:** Parameters which are specific to a given environment where the process is intended to be run (e.g., Docker, Kubernetes, Apache Hadoop, etc.);
- **application:** Parameters specific to the application deployed inside a container, which in the simplest general case is the path to a standard Application entry point (e.g., ../run.sh script); and
- **data:** List of data references passed to the application. If null, the data reference is adopted by the results of the OpenSearch offering in the OWC document (code=http://www.opengis.net/spec/owc-atom/1.0/opensearch). This can be a list of 1 — n Files or URLs.

This is a sample for the ProcessParameters: **ProcessParameters:**

```
{
        "cloud":"ncpu=4, environment=docker, exec_policy=parallel",
        "container":"-t --rm -v $ApplicationRunStore:/home/adesuser/data/ -e
INPUT_IMAGE=$Data",
        "application":"/home/adesuser/processing_scripts/run.sh",
        "data":"S2B_MSIL1C_20180417T102019_N0206_R065_T32TPP_20180417T140522.
zip"
}
```

Both variables **$ApplicationRunStore** and **$Data** are possible user defined placeholders and are replaced at runtime by the ADES application and set to the appropriate value.

The variable **$ApplicationRunStore** is a path placeholder and consists in this case of the dedicated working directory assigned automatically by the ADES to a given process instance.

The variable **$Data** is a placeholder for the data referenced by the user, or by a search offering operation defined in the OWC document (code=http://www.opengis.net/spec/owc-atom/1.0/opensearch)

The Parameters are validated by the ADES before executing the process

If the request is validated and executed successfully the response includes an identifier of the job executed (JobID)

Response:

- **StatusInfo** Execute Response with Status and JobID elements

e.g Response (WPS 2.0)

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:StatusInfo xmlns:wps="http://www.opengis.net/wps/2.0"
```

```
        xmlns:atom="http://www.w3.org/2005/Atom" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.opengis.net/wps/2.0 ../wps.xsd">
        <wps:JobID>809963556</wps:JobID>
        <wps:Status>Running</wps:Status>
        <atom:link rel="monitor"
                href="http://saocompute.eurac.edu/wpseoproc/jobs/809963556" />
        <atom:link rel="cancel"
                href="http://saocompute.eurac.edu/wpseoproc/jobs/cancel/
809963556" />
</wps:StatusInfo>
```

In the case where a process is executed in parallel on a larger set of files, the response includes a list of JobIDs. This is convenient so that a GetStatus($JobID) operation can be executed on each single child process.

DescribeProcess result:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.
opengis.net/wps/1.0.0"
        xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
        xml:lang="en" service="WPS" version="1.0.0"
        xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.
opengis.net/wps/1.0.0/wpsAll.xsd">
        <ProcessDescription wps:processVersion="1.0.0"
                statusSupported="true" storeSupported="true">
                <ows:Identifier>gs:ExecuteProcessEO</ows:Identifier>
                <ows:Title>ExecuteProcess</ows:Title>
                <ows:Abstract>ExecuteProcess Operation EURAC Research EO</ows:
Abstract>
                <DataInputs>
                        <Input maxOccurs="1" minOccurs="1">
                                <ows:Identifier>ProcessInstanceIdentifier</ows:
Identifier>
                                <ows:Title>ProcessInstanceIdentifier</ows:
Title>
                                <ows:Abstract>Identifier of the process
instance to be executed (UUID)</ows:Abstract>
                                <LiteralData>
                                        <ows:AnyValue />
                                </LiteralData>
                        </Input>
                        <Input maxOccurs="1" minOccurs="1">
                                <ows:Identifier>ProcessParameters</ows:
Identifier>
                                <ows:Title>ProcessParameters</ows:Title>
                                <ows:Abstract>Parameters for the Process
(application specific)</ows:Abstract>
                                <ComplexData maximumMegabytes="4">
                                        <Default>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Default>
                                        <Supported>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
```

```
                                                </Format>
                                                <Format>
                                                        <MimeType>text/json</
MimeType>
                                                </Format>
                                                <Format>
                                                        <MimeType>application/
xml</MimeType>
                                                </Format>
                                                <Format>
                                                        <MimeType>application/
json</MimeType>
                                                </Format>
                                                <Format>
                                                        <MimeType>text/plain</
MimeType>
                                                </Format>
                                        </Supported>
                                </ComplexData>
                        </Input>
                        <Input maxOccurs="1" minOccurs="1">
                                <ows:Identifier>ControlOptions</ows:Identifier>
                                <ows:Title>ControlOptions</ows:Title>
                                <ows:Abstract>Control Options</ows:Abstract>
                                <LiteralData>
                                        <ows:AnyValue />
                                </LiteralData>
                        </Input>
                </DataInputs>
                <ProcessOutputs>
                        <Output>
                                <ows:Identifier>StatusInfoResponse</ows:
Identifier>
                                <ows:Title>StatusInfoResponse</ows:Title>
                                <ComplexOutput>
                                        <Default>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Default>
                                        <Supported>
                                                <Format>
                                                        <MimeType>text/xml</
MimeType>
                                                </Format>
                                        </Supported>
                                </ComplexOutput>
                        </Output>
                </ProcessOutputs>
        </ProcessDescription>
</wps:ProcessDescriptions>
```

5.3.3. Proposed Alternatives

An alternative solution, would be to use the proposed micro-services within a container. In this way each application exposes the WPS Execute operation itself. On one side this solution would simplify the interaction with the container, but on the other side this would introduce

some pitfalls for the application development and also for the ADES applications that control the container.

Some of possible pitfalls:

- Subsequent updates in the logic or parametrization has to be applied to all already existing applications: this could introduce some latencies, difficulties and result in possible problems; and
- Different resource scheduling and parallelization technologies used to run the containers within a cluster would require ad-hoc interfaces for accessing the service on the container.

5.3.4. Experiences with AP & ADES

The overall experience with the implementation based on the Testbeds are good.

5.3.5. Other Impressions & Recommendations

The Application OWS Context Document (OWC) should have a clear, minimum set of mandatory fields required for an application to be valid. This minimum set should than be validated and accepted on all clusters.

The process UUID generated should actually be the **SHA-256** message digest of the OWC Application context document. This would help to have a unique, trackable and interchangeable mapping between the OWC document and the process identifier.

5.4. Solenix

5.4.1. Motivation to Participate

Solenix participated in the EOEP Hackathon with two main objectives.

The first objective was to demonstrate the validity and interoperability of its contribution to OGC Testbed-13 — in particular the Application Management Client (AMC) deliverable — and in general of the OGC Testbed-13 results. Two results in particular were relevant both for the Testbed activities and for the Hackathon: the Application Package encoding as a OWS Context document; and the use of WPS for the interface between the AMC and the Application Deployment and Execution Service (ADES).

The second objective was to bridge between Testbed-13 and Testbed-14, even if Solenix committed to participating in the Hackathon without knowing if it would be involved in Testbed-14. Indeed, several topics addressed in the Hackathon were also addressed in Testbed-13 and will be relevant and revisited in Testbed-14. Given that, during the Hackathon,

Solenix's participation in Testbed-14 was confirmed, the Hackathon provided a smooth flow between the two initiatives and a jump-start in Testbed-14.

5.4.2. Starting Assets

Solenix brought the two following assets to the Hackathon:

- The Application Management Client (AMC) developed for OGC Testbed-13; and
- Examples of Application Packages in OWS Context format, also taken from Testbed-13.

Solenix's AMC supports the Application Package (AP) in OWS Context format and implements generic WPS client functionality, which allows a user to browse, register and trigger execution of processes on several instances of the ADES. It also implements the basic OpenSearch catalogue flow integrated in the WebWorldWind 3D globe using FEDEO as endpoint. This implementation is very simplified, both in terms of the UI and in terms of functionality, since only the second step of the two-step search is implemented (i.e., the actual search for products. The collection discovery is bypassed and only well-known collection endpoints are used). It should be noted, in any case, that this functionality was not envisaged to be necessary for the Hackathon and in fact it was not required.

The examples of Application Package in OWS Context format are available at https://github.com/opengeospatial/EOEPHackathon2018/tree/master/AP .

5.4.3. Implemented Solution

The AMC software was deployed on a Virtual Machine hosted on CloudSigma's Frankfurt data center, taking advantage of that Cloud Provider's sponsoring of the Hackathon.

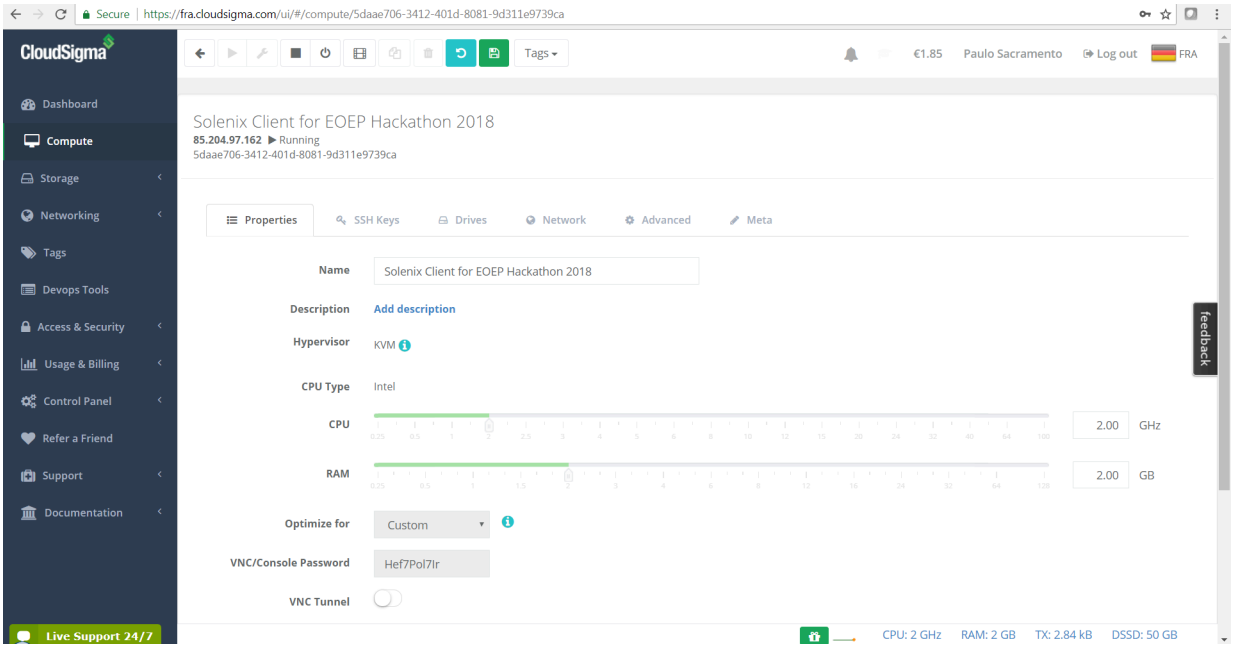
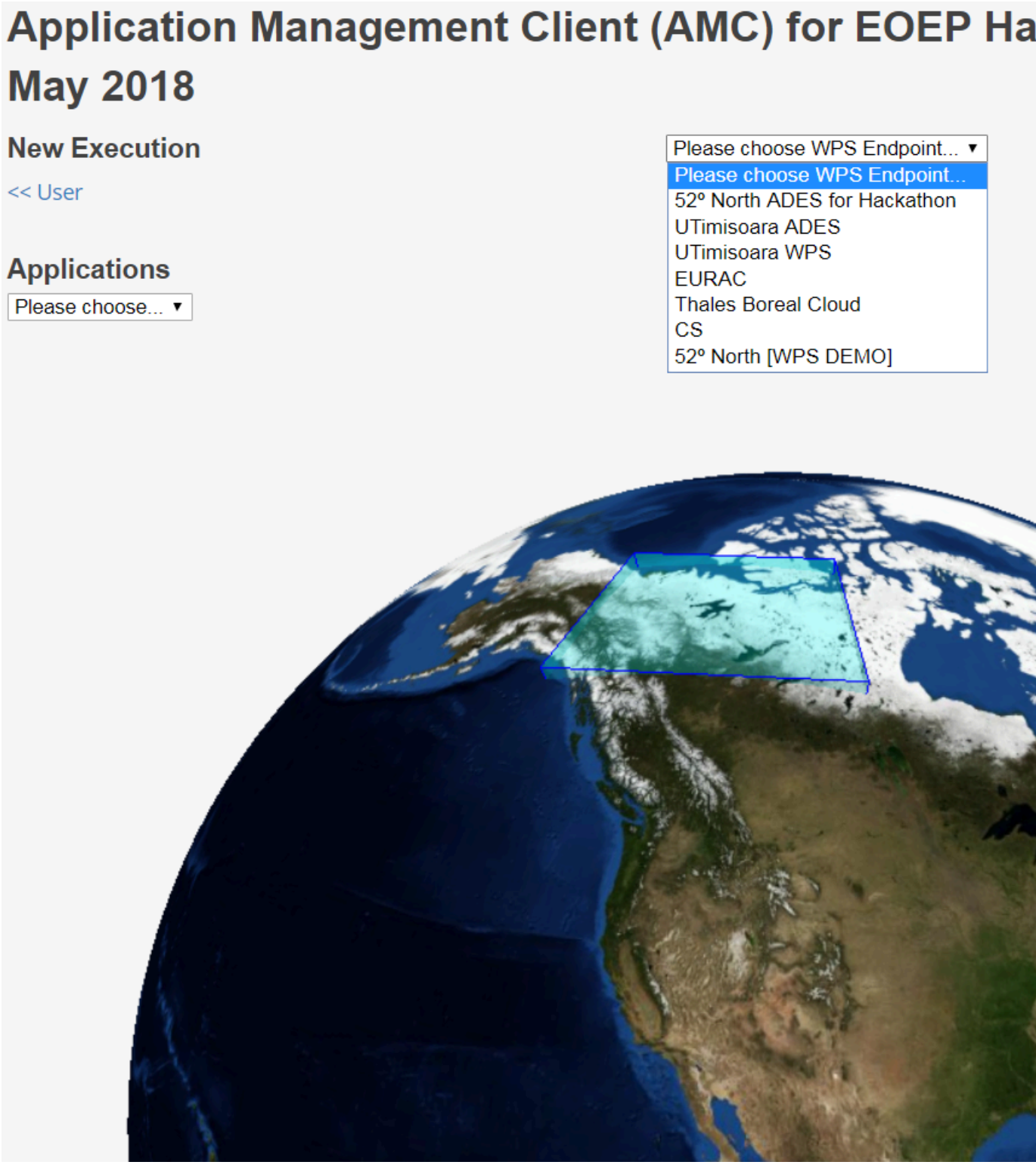


Figure 7 — CloudSigma Dashboard showing VM used during Hackathon for Solenix AMC

The AMC was then minimally tailored for the Hackathon, with no significant functional changes. The set of endpoints made available by participants providing ADES implementations was configured and a rectangle corresponding to the Canadian Northwestern Territories was determined (based on a Shape file provided by ESA/NRCAN) and drawn over the WebWorldWind 3D globe.



No other changes were made or required to be able to interoperate with available and working ADES providers, which is a demonstration of the validity of the Testbed-13 EOC results and the Solenix AMC implementation. In fact, in all attempted cases, once network connectivity (firewalls, blocked ports) and CORS (Cross-Site Scripting protection) issues were solved, the AMC managed to interact with the ADES implementations using the WPS functionality coming out of Testbed-13. This further attested to the maturity and interoperability power of WPS.

5.4.4. Experiences with AP & ADES

One of the AP examples mentioned above was modified as required for the Hackathon, several times during the final event, in agreement with other Participants. The AP used for the final demo is available at https://github.com/opengeospatial/EOEPHackathon2018/blob/master/AP/hackathon-ap.xml . It is reproduced here for completeness.

Application Package XML

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:dc="http://purl.org/dc/
elements/1.1/" xmlns:georss="http://www.georss.org/georss" xmlns:gml="http:
//www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/2.0" xmlns:owc=
"http://www.opengis.net/owc/1.0" xmlns:os="http://a9.com/-/spec/opensearch/1.1/
" xml:lang="en">
    <link rel="profile" href="http://www.opengis.net/spec/owc-atom/1.0/req/
core" title="This file is compliant with version 1.0 of OWS Context"/>
    <link rel="profile" href="http://www.opengis.net/tb13/eoc" title="This
file is compliant with Testbed-13 EOC Thread for Application Packaging"/>
    <id>http://www.opengis.net/eoephack2018/burnscar</id>
    <title>Application Package for Burn Scar Detection application</title>
    <subtitle type="text">Burn Scar Detection</subtitle>
    <updated>2018-05-03T18:25:00Z</updated>
    <author>
        <name>Paulo Sacramento</name>
    </author>
    <rights>-</rights>
    <entry>
        <title>Burn Scar Detection Application for OGC EOEP Hackathon 2018</
title>
        <id>http://www.opengis.net/eoephack2018/burnscar</id>
        <updated>2018-05-03T18:25:00Z</updated>
        <content type="html">Burn Scar Detection Application for OGC EOEP
Hackathon 2018 &#x3c;br/&#x3e; This application detects burn scars over a
Summer period on Canada's Northwestern Territories, using Sentinel data.</
content>
        <!-- DockerImage offering -->
        <owc:offering code="http://www.opengis.net/tb13/eoc/docker">
            <owc:content type="text/plain">registry.hub.docker.com/cnlspacebel/
landcover</owc:content>
            <inputs>
                    <name>image</name>
            </inputs>
        </owc:offering>
        <owc:offering code="http://www.opengis.net/tb13/eoc/
wpsProcessOffering">
            <owc:content type="application/xml">
                <wps:ProcessOffering jobControlOptions="async-execute dismiss"
 outputTransmission="value reference" xmlns:ows="http://www.opengis.net/ows/
2.0" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:xlink="http://www.w3.org/
1999/xlink">
                    <wps:Process>
                        <ows:Title>Burn Scar Detection Demo</ows:Title>
                        <ows:Abstract>This application detects burn scars over
a Summer period on Canada's Northwestern Territories, using Sentinel data.</
ows:Abstract>
                        <ows:Identifier>BurnScarDemo</ows:Identifier>
                        <wps:Input>
                            <ows:Title>Area Of Interest</ows:Title>
```

```
                            <ows:Abstract>Area of interest (AOI) as a WKT
POLYGON text string.</ows:Abstract>
                            <ows:Identifier>AreaOfInterest</ows:Identifier>
                            <wps:LiteralData>
                                <!-- Not sure if WKT has a MIME type -->
                                <wps:Format mimeType="text/plain" default=
"true"/>
                                <LiteralDataDomain default="true">
                                    <ows:AnyValue/>
                                    <ows:DataType ows:reference="http://www.w3.
org/2001/XMLSchema#string">String</ows:DataType>
                                    <ows:DefaultValue>POLYGON( (70 -140, 58 -
140, 58 -100, 70 -100, 70 -140) )</ows:DefaultValue>
                                </LiteralDataDomain>
                            </wps:LiteralData>
                        </wps:Input>
                        <wps:Input>
                            <ows:Title>Start Time Of Interest</ows:Title>
                            <ows:Abstract>Start Time of Interest for the
processing, as an RFC 3339 date-time text string. Example: 1990-12-31T23:59:60Z
</ows:Abstract>
                            <ows:Identifier>StartTimeOfInterest</ows:
Identifier>
                            <wps:LiteralData>
                                <wps:Format mimeType="text/plain" default=
"true"/>
                                <LiteralDataDomain default="true">
                                    <ows:AnyValue/>
                                    <ows:DataType ows:reference="http://www.w3.
org/2001/XMLSchema#string">String</ows:DataType>
                                    <ows:DefaultValue>2017-06-17T00:00:00Z</
ows:DefaultValue>
                                </LiteralDataDomain>
                            </wps:LiteralData>
                        </wps:Input>
                        <wps:Input>
                            <ows:Title>Stop Time Of Interest</ows:Title>
                            <ows:Abstract>Stop Time of Interest for the
processing, as an RFC 3339 date-time text string. Example: 1990-12-31T23:59:60Z
</ows:Abstract>
                            <ows:Identifier>StopTimeOfInterest</ows:Identifier>
                            <wps:LiteralData>
                                <wps:Format mimeType="text/plain" default=
"true"/>
                                <LiteralDataDomain default="true">
                                    <ows:AnyValue/>
                                    <ows:DataType ows:reference="http://www.w3.
org/2001/XMLSchema#string">String</ows:DataType>
                                    <ows:DefaultValue>2017-06-28T23:59:59Z</
ows:DefaultValue>
                                </LiteralDataDomain>
                            </wps:LiteralData>
                        </wps:Input>
                                    <wps:Output>
                            <ows:Title>Result URL (TIF file)</ows:Title>
                            <ows:Abstract>URL pointing to result (GeoTIF file)
</ows:Abstract>
                            <ows:Identifier>ResultURL</ows:Identifier>
                            <wps:ComplexData>
                                <wps:Format mimeType="text/url" default="true"/
>
                            </wps:ComplexData>
                        </wps:Output>
```

```
                    </wps:Process>
                </wps:ProcessOffering>
            </owc:content>
        </owc:offering>
    </entry>
    <entry>
        <title>OpenSearch Collections</title>
        <id>http://www.opengis.net/tb13/eoc/OS_Collections</id>
        <updated>2017-09-04T15:23:09Z</updated>
        <content type="html">EOC OpenSearch Collections</content>
        <owc:offering code="http://www.opengis.net/spec/owc-atom/1.0/
opensearch">
            <owc:content type="application/opensearchdescription+xml" href=
"https://finder.eocloud.eu/resto/api/collections/Sentinel1/describe.xml"/>
                <mapping>
                            <query>
                        <map key="AreaOfInterest" value="geometry" />
                        <map key="StartTimeOfInterest" value="startDate"/>
                        <map key="StopTimeOfInterest" value="stopDate"/>
                            </query>
                            <inputs>
                                <map key="image" value="features[*].properties.
productIdentifier" />
                            </inputs>
                     </mapping>
        </owc:offering>
    </entry>
</feed>
```

Besides the basic administrative information about the application (Burn Scar Detection), the following features are worth highlighting.

- On the first entry, one offering pointing to the Docker container with the application code to be run by ADES.
- Still on the first entry, one offering containing a WPS Process Description of the Burn Scar Detection process, consisting of three inputs — Area of Interest as a WKT string (the default polygon is the one mentioned previously, for the Northwestern Territories of Canada), Start and Stop times of interest — and one output, for the URL where it will be possible to obtain the results of the processing.
- On a second entry, a further offering consisting of the OpenSearch collection endpoint to be used as a catalogue and a mapping of field names which is necessary to address the fact that the OpenSearch standard does not specify/constrain this, which leads to different implementations using different names.

During the two days of the final demo event, it was possible to attempt integration with a few ADES implementations, with varying degrees of success. Using the AP above or slight variations, it is possible to register the AP after choosing one of the available ADES, using the following page:

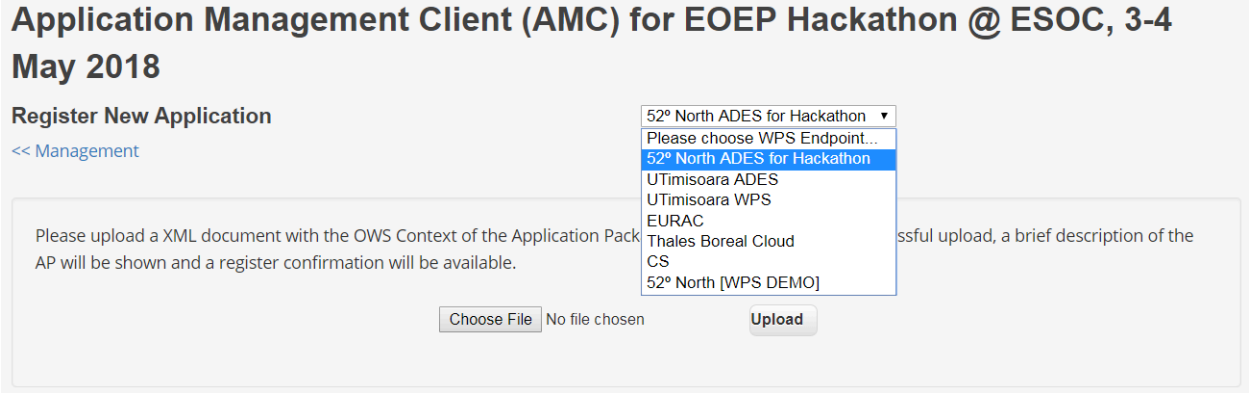


Figure 8 — Solenix AMC for Hackathon: Application registration page showing several participant ADES

On uploading the AP, the AMC prints-out some general information about the AP and asks for a confirmation. Once this is given, under the hood the AMC contacts the ADES using the WPS Execute operation on a specially prepared process called 'DeployProcess.'

After the application is registered, it can be selected for usage. Application execution can be triggered after filling-in the fields of the dynamically generated form built from the WPS Process Description returned by the ADES as a DescribeProcess response. The Figure below shows this for an earlier version of the Burn Scar Detection Application Package which considered a single time window of interest field instead of two separate fields, one for the start time and one for the stop time:

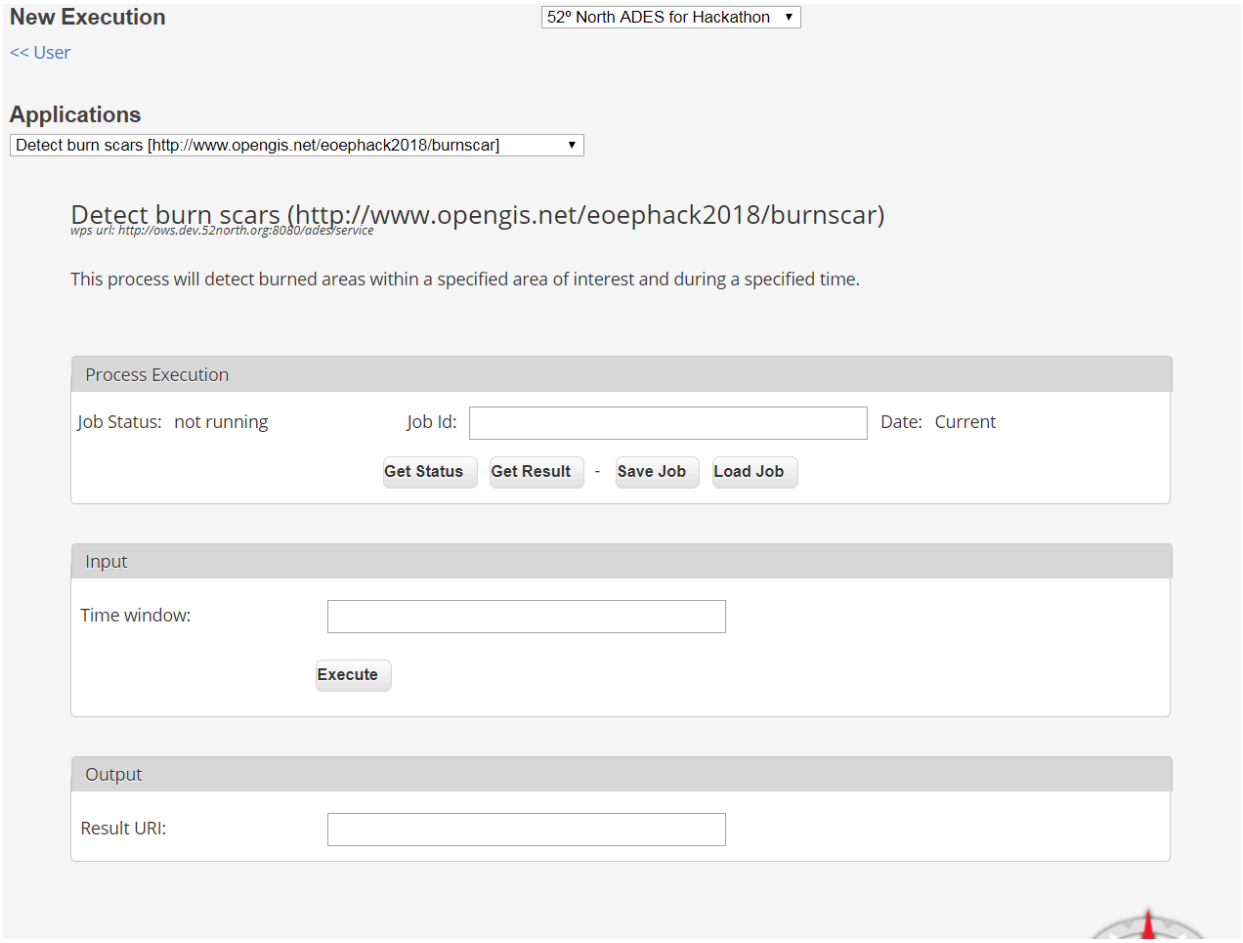


Figure 9 — Solenix AMC for Hackathon: Execution page built dynamically from 52 North ADES WPS Burn Scar Process Description

Besides the 52 North ADES, with which partial integration was successfully implemented before the final event (it was not possible to integrate the complete execution flow), it was possible to fully integrate and demonstrate the complete flow using the University of Timisoara's ADES implementation during the final event.

For what concerns the Thales implementation, by the end of the final event there were still CORS and connectivity issues due to the fact that they were using the Boreal Cloud made available by NRCan, which is not easily accessible through the Internet. All parties agreed and were confident, however, that had these issues been addressed, integration would have been possible, since both the Solenix AMC and the Thales ADES implement standard WPS.

As can be seen in the Figure above, the University of Timisoara actually exposed two endpoints, one called ADES for the registration/unregistration of Application Packages and another one called WPS for the actual execution of processes. Even if this was not the original intention and none of the Testbed-13 ADES implementations or other Hackathon implementations did this, it does not pose any negative consequences and in fact it was agreed to keep it such to highlight that it is also a valid approach. It can actually be argued that this is a desirable split between two kinds of function which are fundamentally different and have different access requirements (the

application registration/unregistration, for privileged users; and the application execution, for regular users).

It was mentioned during the final event that from the Testbed-13 ERs it was not evident that the two functions were supposed to be provided by a single endpoint, which is a point of improvement for future Tested ERs.

Finally, it should be mentioned that integration with EURAC's backend was also attempted, but this was not possible for two main reasons: the fact that EURAC's backend did not send appropriate headers and so the browser's CORS protections did not allow the requests to complete (requires adequate server configuration to work); and the fact that also EURAC, similarly to NRCan with the Boreal Cloud, has its own private infrastructure which is not easily accessible through the Internet.

To allow access, EURAC put in place a proxy and implemented simple HTTP Authentication, but all attempts to change the client to send appropriate authentication headers did not succeed. The EURAC backend responded with a valid GetCapabilities response, but the Process offering was always empty (which EURAC confirmed was the expected result when authentication fails).

5.5. Space Applications Services

5.5.1. Motivation to Participate

Space Applications Services has been developing a platform named ASB (for Automated Service Builder) for several years now. It has started before the draft Exploitation Platform Open Architecture was published and thus also before the Testbed-13 took place.

ASB has a mechanism to dynamically deploy Docker container images and run processes pre-installed in these containers in cloud environments. The actual deployment of the containers is delegated to Marathon (which relies on Mesos for the selection of the target worker nodes) and each container runs its own lightweight WPS service.

During the Hackathon, we have implemented a client component that dynamically generates an Application Package using the information we have in database (including processes, inputs/outputs, datatypes), then interacts with the WPS interface of the ADES implementations to deploy the application and trigger the executions.

The key differences between the Marathon-based and the ADES-based implementations are:

- With Marathon, deployed containers run a pre-installed WPS service that exposes ("offers") one or more arbitrary processes. After the deployment through Marathon, the client must communicate directly with the containers to execute the actual processes.
- With ADES WPS, deployed containers run arbitrary code as soon as they are started. After the deployment through the ADES, the client must still communicate with the ADES to execute the actual processes (newly added in the ADES offerings).

During the Hackathon, the new client component has been tested against the ADES implemented by Thales Alenia Space and 52°North.

5.5.2. Implemented Solution

The ASB core component in charge of requesting the deployment of the process/application images and their execution is the Tasks Manager. As most of the other ASB core components, it is implemented in the Python scripting language. It uses the WPS client package provided by the open-source library OWSLib (https://github.com/geopython/OWSLib/) to communicate with WPS servers.

5.5.2.1. Added support for WPS 2.0.0 in the client library

The WPS client package of OWSLib does not support the WPS 2.0.0 interface standard. The initial task has thus been to add basic support for WPS 2.0.0 to OWSLib, at least to be able to successfully communicate with the ADES implementations for deploying, undeploying and executing applications, both in synchronous and asynchronous modes.

5.5.2.2. Added support for ADES in the ASB Tasks Manager

5.5.2.2.1. Dynamic generation of the Application Package

The ASB Tasks Manager is generic in the sense that it is meant to be able to deploy any containerized application into any execution environment, provided it is given the appropriate adapters. At run-time, the component receives the necessary inputs for executing the application. Information necessary for deploying the application is retrieved from its database.

Because the application packages have a fixed format, the chosen lightweight solution was to use a template to automate their generation. The Python-based template engine Jinja2 uses the template document (see below) and the application properties (example on the next pages) to render the final application package. Please note that not all the possibilities have been implemented in the template. For example, it does not support process parameters of type Bounding Box nor nested inputs.

Application Package XML/Jinja2 Template

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:dc="http://purl.org/dc/
elements/1.1/"
    xmlns:georss="http://www.georss.org/georss" xmlns:gml="http://www.opengis.
net/gml"
    xmlns:owc="http://www.opengis.net/owc/1.0" xmlns:os="http://a9.com/-/spec/
opensearch/1.1/"
    xml:lang="en">
  <link rel="profile" href="http://www.opengis.net/spec/owc-atom/1.0/req/core"
      title="This file is compliant with version 1.0 of OGC Context"/>
  <link rel="profile" href="http://www.opengis.net/tb13/eoc"
      title="This file is compliant with Testbed-13 EOC Thread for Application
Packing"/>
  <id>{{ ap.id }}</id>
```

```
  <title>{{ ap.title }}</title>
  <subtitle type="text">{{ ap.subtitle }}</subtitle>
  <updated>{{ ap.updated }}</updated>
  <author>
    <email>{{ ap.author.email }}</email>
  </author>
  {% if ap.generator.uri %}<generator uri="{{ ap.generator.uri }}" version="{{
ap.generator.version }}">{{ ap.generator.title }}</generator>{% endif %}
  {% if ap.publisher.title %}<dc:publisher>{{ ap.publisher.title }}</dc:
publisher>{% endif %}
  {% if ap.rights %}<rights>{{ ap.rights }}</rights>{% endif %}

  {% if ap.aoi %}
  <!-- Geographic Area of interest for the App -->
  <georss:where>
    <gml:Polygon xmlns:gml="http://www.opengis.net/gml">
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList>{{ ap.aoi }}</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:Polygon>
  </georss:where>
  {% else %}
  <!-- No geographic area of interest available for the App -->
  {% endif %}
  {% if ap.toi %}
  <!-- A date or range of dates relevant to the App -->
  <dc:date>{{ ap.toi }}</dc:date>
  {% else %}
  <!-- No relevant date or range of dates available for the App -->
  {% endif %}
  {% for app in ap.applications %}
  <entry>
    <id>{{ app.id }}</id>
    <link rel="profile" href="http://www.opengis.net/tb13/eoc/application"
      title="This entry contains an application as specified by Testbed-13 EOC
Thread"/>
    <title>{{ app.title }}</title>
    <content type="text">{{ app.content }}</content>

    {% if app.docker_image %}
    <owc:offering code="http://www.opengis.net/tb13/eoc/docker">
      <owc:content {% if app.docker_cmd %}cmd="{{ app.docker_cmd }}"{% endif
%} type="text/plain">{{ app.docker_image }}</owc:content>
    </owc:offering>
    {% endif %}
    <!-- THE WPS PROCESS DESCRIPTION -->
    <owc:offering code="http://www.opengis.net/tb13/eoc/wpsProcessOffering">
      <owc:content type="application/xml">
        <wps:ProcessOffering jobControlOptions="async-execute dismiss"
                             outputTransmission="value reference" xmlns:ows=
"http://www.opengis.net/ows/2.0"
                             xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:
xlink="http://www.w3.org/1999/xlink">
          <wps:Process>
            <ows:Title>{{ app.process.title }}</ows:Title>
            <ows:Abstract>{% if app.process.abstract %}{{ app.process.abstract
}}{% else %}No abstract{% endif %}</ows:Abstract>
            <ows:Identifier>{{ app.process.id }}</ows:Identifier>

            {% for input in app.process.inputs %}
```

```
            <wps:Input minOccurs="{{ input.min_occurs }}" maxOccurs="{{ input.
max_occurs }}">
              <ows:Title>{{ input.title }}</ows:Title>
              <ows:Abstract>{{ input.abstract }}</ows:Abstract>
              <ows:Identifier>{{ input.id }}</ows:Identifier>
              {% for metadata in input.metadatas %}
              <ows:Metadata>
                <atom:link rel="{{ metadata.rel }}" href="{{ metadata.href }}"/
>
              </ows:Metadata>
              {% endfor %}
              {% if input.data_type|lower == "complexdata" %}
              <wps:ComplexData>
                <wps:Format mimeType="{{ input.complexdata.mimetype }}"
default="{{ input.complexdata.default }}"/>
              </wps:ComplexData>
              {% elif input.data_type|lower == "literaldata" %}
              <wps:LiteralData>
                {% for format in input.formats %}
                <wps:Format mimeType="{{ format.mimetype }}" encoding="{{
format.encoding }}" schema="{{ format.schema }}" default="{{ format.default
}}"/>
                {% endfor %}
              </wps:LiteralData>
              {% elif input.data_type|lower == "boundingbox" %}
                <!-- wps:BoundingBox TO BE IMPLEMENTED -->
              {% endif %}
            </wps:Input>
            {% endfor %}

            {% for output in app.process.outputs %}
            <wps:Output>
              <ows:Title>{{ output.title }}</ows:Title>
              <ows:Abstract>{{ output.abstract }}</ows:Abstract>
              <ows:Identifier>{{ output.id }}</ows:Identifier>
              {% for metadata in output.metadatas %}
              <ows:Metadata>
                <atom:link rel="{{ metadata.rel }}" href="{{ metadata.href }}"/
>
              </ows:Metadata>
              {% endfor %}
              {% if output.data_type|lower == "complexdata" %}
              <wps:ComplexData>
                <wps:Format mimeType="{{ output.complexdata.mimetype }}"
default="{{ output.complexdata.default }}"/>
              </wps:ComplexData>
              {% elif output.data_type|lower == "literaldata" %}
              <wps:LiteralData>
                {% for format in output.formats %}
                <wps:Format mimeType="{{ format.mimetype }}" encoding="{{
format.encoding }}" schema="{{ format.schema }}" default="{{ format.default
}}"/>
                {% endfor %}
              </wps:LiteralData>
              {% elif output.data_type|lower == "boundingbox" %}
                <!-- wps:BoundingBox TO BE IMPLEMENTED -->
              {% endif %}
            </wps:Output>
            {% endfor %}

          </wps:Process>
        </wps:ProcessOffering>
      </owc:content>
```

```
    </owc:offering>

    <category scheme="http://www.opengis.net/tb13/eoc/os" term="LINUX" label=
"This app runs in Linux"/>

  </entry>
  {% endfor %}

  {% for cat in ap.catalogues %}
  <entry>
    <id>{{ cat.id }}</id>
    <link rel="profile" href="http://www.opengis.net/tb13/eoc/catalogue"
        title="This entry contains an catalogue as specified by Testbed-13 EOC
Thread"/>
  </entry>
  {% endfor %}

</feed>
```

The following JSON document is an example data structure that contains the necessary information for rendering the above template and generate an Application Package document.

Example Application Properties

```
{
    "ap": {
        "publisher": {},
        "subtitle": "",
        "generator": {},
        "author": {},
        "catalogues": [],
        "rights": "OGC EOEP Hackathon 2018",
        "applications": [
            {
                "title": "SnapProcess.SpaceApps",
                "process": {
                    "inputs": [
                        {
                            "data_type": "LiteralData",
                            "title": "Input Image",
                            "abstract": "The path of the image to process",
                            "formats": [
                                {
                                    "mimetype": "text/plain",
                                    "default": "true",
                                    "schema": "",
                                    "encoding": "UTF-8"
                                }
                            ],
                            "max_occurs": "1",
                            "id": "inputdata",
                            "min_occurs": "1",
                            "metadatas": []
                        },
                        {
                            "data_type": "LiteralData",
                            "title": "Output Path",
                            "abstract": "The output path must be placed into /
target directory",
                            "formats": [
                                {
                                    "mimetype": "text/plain",
                                    "default": "true",
```

```
                                "schema": "",
                                "encoding": "UTF-8"
                            }
                        ],
                        "max_occurs": "1",
                        "id": "outputdata",
                        "min_occurs": "1",
                        "metadatas": []
                    }
                ],
                "abstract": "",
                "id": "SnapProcess.SpaceApps",
                "outputs": [
                    {
                        "data_type": "LiteralData",
                        "title": "Product URL",
                        "abstract": "product url",
                        "formats": [
                            {
                                "mimetype": "text/plain",
                                "default": "true",
                                "schema": "",
                                "encoding": "UTF-8"
                            }
                        ],
                        "id": "ProductURL",
                        "metadatas": []
                    }
                ],
                "title": "SnapProcess.SpaceApps"
            },
            "docker_cmd": "gpt -c 8G -q 8 /S1_Cal_Deb_ML_Spk_TC_cmd.xml -
Poutputdata=${outputdata} -Pinputdata=${inputdata}",
            "content": "Process deployed through ASB platform",
            "id": "eoeph18-snapprocessspaceapps-1_5f2c9acf-82d3-455c-807a-
3a9e70b7e4e1",
            "docker_image": "thalesaleniaspace/snap:latest"
        }
    ],
    "title": "SnapProcess.SpaceApps",
    "id": "eoeph18-snapprocessspaceapps-1_5f2c9acf-82d3-455c-807a-
3a9e70b7e4e1"
  }
}
```


5.5.2.2.2. Interactions with the ADES WPS Interface

When the ASB Task Manager receives an execution request from the workflow engine, it fetches the definition of the process, its input and output parameters, and the data types from its database and generates the Application Package document.

The component then verifies that the application is not already deployed in the ADES (using the configured process title string). If so, it is not deployed again. If not, the DeployProcess process is executed (synchronously) and provided with the Application Package. When the deployment is complete, the client verifies that the application is now listed in the WPS offerings.

The component then generates a process Execute request and sends it to the ADES. This execution is asynchronous. A GetStatus request is issued at regular interval. When the execution is complete, a GetResult request is issued to obtain the outputs.

The ASB Task Manager executes the UndeployProcess process on the ADES to remove the application from its offerings.

5.5.3. Experiences with AP & ADES

The generation of the Application Packages did not pose specific problems. The EP Application Package E.R. is detailed enough to understand how the applications metadata must be encoded as Atom feeds.

The integration of the client component with the ADES of Thales Alenia Space has required some adaptations on both sides. The main incompatibilities are listed hereafter.

- The *DeployProcess* and *UndeployProcess* WPS processes did not have the expected title.
- The *UndeployProcess* input parameter did not have the expected identifier.
- The *DeployProcess* implementation was expecting a non-empty process abstract string (even though it is optional in WPS 2.0.0).
- The application was expecting two inputs: “Input Image” and “Output Path.” Because the first parameter can only receive the path to a single image, the application had to be executed several times, once for each image in the AOI. The selection of these images had thus to be done on the client side.
- The Application Package was expecting an extra attribute for specifying the actual command to be executed within the Docker container. This attribute has been hardcoded for testing purpose.
- It is a challenge to track issues when the response received from the WPS is a “NoApplicableCode” exception report with no additional explanatory text. Checking logs on the server side has been necessary to identify the problems.

5.5.4. Other Impressions & Recommendations

- The extra attribute required by the Thales implementation of the ADES (see the “cmd” attribute in the docker offering in the AP template, above) is an easy solution that allows preparing generic Docker images (only including the SNAP toolbox in this case) and dynamically providing the command to be run at execution time. This is similar to preparing a generic Docker image using a Dockerfile without “CMD” instruction and providing the command when the container is instantiated (using docker-compose or on the command line).
- The possibility to include Catalogue offerings in the Application Packages and let the ADES query these catalogues for identifying the images to process should not be offered without taking some precautions. In particular, if the client does not provide query

parameters (e.g. geo and time constraints) that make sense, the risk is that a single application execution will trigger the processing of a huge amount of images, without even informing the client. The client should keep the control over the exact set of images to be processed by the ADES.

- The Thales application accepts a single image path per execution. This forces the client to implement a mechanism that executes the application once for each image and collects the results when they become available. An application that accepts a list of paths and executes the process on each entry in the list could be a solution in this particular case. However this is not considered generic enough. What is missing in the Application Package specification is a means to indicate which of the process inputs receives a list of entries that may be processed independently.
- The WPS 2.0.0 service that implements the ADES collects Job Identifiers that are then listed automatically in the *GetCapabilities* responses (as possible values for the *JobId* input parameter of the *GetStatus* and *GetResult* operations). By default, nothing prevents a client to get the status and then the results of any of the listed jobs.
- As a multi-purpose generic workflow engine, ASB generates, deploys and executes individual processes dynamically, when comes the time to execute them within processing chains. This has resulted in the implementation of the template of Application Package provided above, which is thus rendered at run-time. Our work together with views raised by others at the Hackathon causes us to have some concerns on the usability and maintainability of the resulting implementations. This will need to be addressed in future work. In particular, libraries should be created or extended to facilitate the implementation of new clients and servers. This includes adding support for WPS 2.0.0 and WPS-T in more programming languages, and implementing libraries for generating, parsing and validating Application Packages.

5.6. West University of Timisoara and Institute eAustria Timisoara

In the following, West University of Timisoara is abbreviated *UVT* and Institute eAustria Timisoara as *IeAT*.

5.6.1. Motivation to Participate

Our participation to the EOEP Hackathon was motivated by our aim of further extending the WPS 2.0 server developed as part of the ESA funded EO4SEE Project.

Particularly we are interested in insuring interoperability with third-party WPS implementations and adherence to the recommendations originating from the Testbed activities.

5.6.2. Implemented Solution

Our solution involved introducing a series of new conventions inside the Application Package, conventions agreed upon on-site during the Hackathon. Particularly the conventions have been endorsed by Solenix, 52North and VITO. The details of the modified Application Package are described in section Experiences with AP & ADES.

The implemented solution demonstrated complete interaction between the AMC developed by Solenix and the EWPS Server provided by UVT and supported by IeAT. The solution demonstrated that interaction between the various components envisioned in Testbed-13 is feasible.

The demonstration presented at the Hackathon involved the execution of the complete application lifecycle:

- Application Deployment
- Application Execution
- Application UnDeployment

It is worth mentioning that the triggered application execution successfully completed the proposed use-case scenario. Runtime information regarding the execution can be seen in Figure 10.

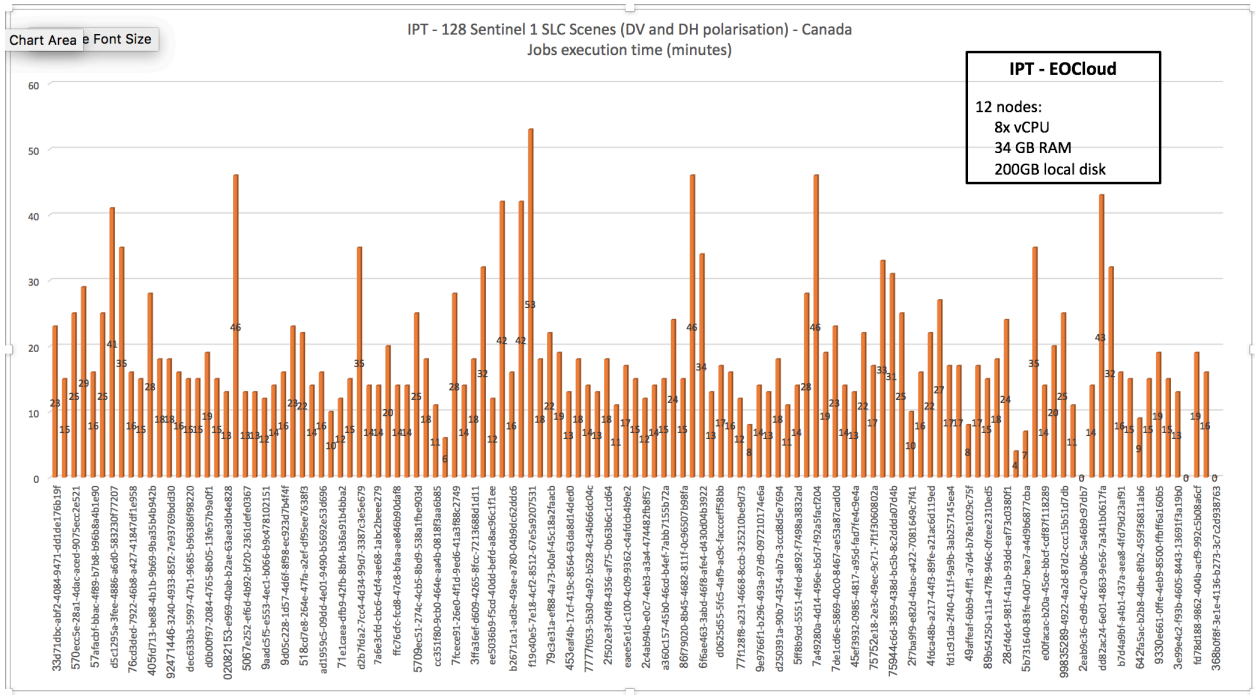


Figure 10 — Application Execution Statistics. x-axis in minutes computing time, y-axis identifies the 128 Sentinel-1 scenes

5.6.2.1. Implementation

The solution proposed by UVT and IeAT involved the deployment of EWPS and the required backend components on the IPT Poland cloud.

From a high level perspective, our solution was composed out of:

- EWPS WPS 2.0 Server running as managed Marathon container;
- Cook Scheduler managed by Marathon providing job scheduling capabilities;
- The Marathon container orchestration framework; and
- Mesos Mesos Cluster running on top of 12 nodes providing compute resources.

5.6.3. Experiences with AP & ADES

As part of the Hackathon we agreed together with the other participants (particularly Solenix, 52North, and VITO) an modified version of the application package

The application package assumed the introduction of an mapping section inside the owc:offering node, as depicted below.

```
<entry>
  <title>OpenSearch Collections</title>
  <id>http://www.opengis.net/tb13/eoc/OS_Collections</id>
  <updated>2017-09-04T15:23:09Z</updated>
  <content type="html">EOC OpenSearch Collections</content>
  <!-- OpenSearch offering for Spacebel -->
  <owc:offering code="http://www.opengis.net/spec/owc-atom/1.0/opensearch">
    <owc:content type="application/opensearchdescription+xml" href="https://
finder.eocloud.eu/resto/api/collections/Sentinel1/describe.xml"/>
    <mapping>
      <query>
        <map key="AreaOfInterest" value="geometry"/>
        <map key="StartTimeOfInterest" value="startDate"/>
        <map key="StopTimeOfInterest" value="completionDate"/>
      </query>
      <inputs>
        <map key="image" value="features[*].properties.productIdentifier"/>
      </inputs>
    </mapping>
  </owc:offering>
</entry>
```

The mapping node was aiming two different purposes:

- Mapping attributed to OpenSearch queries (the query node); and
- Mapping WPS processing inputs to jobs handled by the underlying execution system, and merged together from the OpenSearch query using an jsonpath expression.

5.6.4. Other Impressions & Recommendations

The Hackathon was of great value for UVT/IeAT as we have not been involved in the Testbed activities. It allowed us to test interoperation with different clients (Solenix) and ADES implementation (Solenix).

Our main recommendation would be that the next Hackathon activities should have a dedicated session for decision making and agreeing upon common rules.

Also complete (end-to-end) practical demonstrations would be great.

5.7. VITO

The full documentation of VITO's work is available on Github and mostly repeated here for persistency. It contains:

1. Deployment of simplified application package (https://github.com/VitoTAP/ADES/blob/master/resources/executeDeploy.xml)
2. Direct execution of arbitrary command in arbitrary docker image, no deploy needed (https://github.com/VitoTAP/ADES/blob/master/resources/executeDocker.xml)
3. Deployment of a full WPS packaged as Docker container. Kubernetes was used as a scalable/multi-cloud container orchestrator. https://github.com/VitoTAP/ADES/tree/master/microservice

5.7.1. Motivation to Participate

- VITO operates the PROBA-V Exploitation Platform
- VITO is a potential provider of application packages
- VITO is a potential user of application packages
- VITO participates in the OpenEO project which has similar/complementary goals.

5.7.2. Implemented Solution

VITO implemented a backend (ADES), exploring both the WPS and OpenEO approach. The backend is based on pyWPS as an additional interface to the OpenEO backend.

First implementation is on the PROBA-V MEP processing cluster where we have:

- Apache Spark for parallelization
- Catalog containing SLC S1 data
- Support to distribute docker images
- SNAP preinstalled, so we can also run gpt flows directly
- OpenSearch endpoint with some S1 SLC and GRD files

Second implementation on CloudFerro: — Kubernetes for deployment and parallelization — CloudFerro catalog and archive

5.7.2.1. Use case implementation options

While implementing the use case, we noted that there are in fact different ways to support it. The right option also depends on which more generic problem we want to solve:

1. Deployment of an arbitrary WPS process using containers
2. Applying an operation (algorithm, workflow,...) to a list of EO products, independently of each other

5.7.2.1.1. Arbitrary WPS deployment approach

For arbitrary WPS deployment, I believe we should take into consideration the functionality offered by more generic IAAS/PAAS solutions such as Kubernetes and Amazon Lambda. This type of technology allows anyone to easily deploy a scalable web service, packaged as a docker container.

So what if a user wishes to add an operation to a WPS, simply packages a one-process WPS into a docker container? This would allow him to deploy his WPS operation on a whole range of cloud platforms.

If we want to go a step further, a DIAS or Exploitation Platform can wrap this process into a 'DeployWPS' call, and provide some additional services such as an 'aggregating' WPS that manages the lifecycle of these user-defined WPS's, and exposes the operations offered by these WPS's into a single WPS interface, proxying any incoming requests to the correct end point.

The standardization needed for this micro-services based approach is limited. A proof of concept for a micro-service can be found here: https://github.com/VitoTAP/ADES/tree/master/microservice The python script for the WPS is currently limited to 51 lines. The dockerfile is 21 lines, mostly because of having to set up Python 3.5 on CentOS.

5.7.2.1.2. Applying operations to EO products

To simply apply an operation to a list of EO products, we may not really require the deployment of a new WPS process. These are the basic elements needed.

- An execution environment (the docker image).
- A command line template to invoke the operation, which works inside the execution environment. This should be a template that is evaluated against the properties of an EO product.
- Other docker options, such as mounts/volumes that need to be present to stage data offered by the DIAS/EP into the container.
- An OpenSearch query, that references the list of products to process.

A very important detail to make this work, is to properly describe how an OpenSearch query resulting in a list of products, and a command template, is transformed into a list of commands. In my current implementation, I decided to avoid a 'generic' approach, where this would be done through nested WPS processes, and an OpenSearch endpoint provided in an OWS Context. Instead, I assume that the DIAS/EP should implement this in a proper way, hereby using whatever catalogue they have available. This allows them to also take care of ensuring that the requested product is made available to the container.

The 'get_commands' function in this python file: https://github.com/VitoTAP/ADES/blob/master/ades/docker_generic.py already shows a sample implementation.

5.7.3. ADES implementation

The proof-of-concept implementation is based on the latest version of PyWPS (4.0.0). This version implements WPS 1.0.0, thus support for 2.0.0 is not available.

5.7.3.1. Processing OpenSearch results

One of the fundamental cases in this Hackathon, is how to apply an arbitrary command to the results of an OpenSearch query.

The problem boils down to the fact that an OpenSearch result does not contain a local filename, while most commands do expect some kind of file name as input. The solution we implemented was as follows.

1. Preprocess OpenSearch results to add a 'local_filename' property.
2. Make sure this path exists in the execution environment. Depending on whether the EO data archive is on NFS or object storage, different solutions may be needed. It is the responsibility of the ADES implementor to make this transparent.

3. Now the WPS execute/deploy call can specify the command as a template. For instance: /opt/snap/bin/gpt -e “/S1_Cal_Deb_ML_Spk_TC_cmd.xml” “-Pinputdata=$local_filename” “-Poutputdata=/out/S1result”

4. In the case of docker, the execute/deploy call should properly mount the required archive directory inside the container.

5.7.3.2. Deploy process

Adding the DeployProcess operation proved difficult, as I was not able to dynamically add a new operation from within an execute call. However, I did achieve this by adding a /deploy REST call, implemented outside of PyWPS.

5.7.3.3. Direct Docker execution

To test basic integration with Docker, I added a DockerProcessFiles process. A sample execute call is provided below:

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service=
"WPS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.
opengis.net/wps/1.0.0" xmlns:wfs="http://www.opengis.net/wfs" xmlns:wps="http:
//www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:
gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:
wcs="http://www.opengis.net/wcs/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.
net/wps/1.0.0/wpsAll.xsd">
    <ows:Identifier>DockerProcessFiles</ows:Identifier>
    <wps:DataInputs>
        <wps:Input>
            <ows:Identifier>local_filename</ows:Identifier>
            <wps:Reference mimeType="application/json" xlink:href="http://
localhost:5000/wps" method="POST">
                <wps:Body>
                    <wps:Execute version="1.0.0" service="WPS">
                        <ows:Identifier>find_local_files</ows:Identifier>
                        <wps:DataInputs>
                            <wps:Input>
                                <ows:Identifier>OpenSearch Query</ows:
Identifier>
                                <wps:Data>
                                    <wps:LiteralData><![CDATA[https:
//finder.eocloud.eu/resto/api/collections/Sentinel1/search.json?
maxRecords=2000&startDate=2017-06-17T00:00:00Z&completionDate=2017-06-
28T23:59:59Z&productType=SLC&processingLevel=LEVEL1&sensorMode=IW&sortParam=
startDate&sortOrder=descending&geometry=POLYGON( (-122.87109374999997+78.595
29919212491,-74.00390624999997+78.52557254138316,-99.31640624999996+58.72259
88280434,-140.80078124999997+58.90464570302001,-150.99609374999994+71.9108878
7611528,-122.87109374999997+78.59529919212491) )&dataset=ESA-DATASET]]></wps:
LiteralData>
                                </wps:Data>
                            </wps:Input>
                        </wps:DataInputs>
                        <wps:ResponseForm>
                            <wps:RawDataOutput mimeType="application/json">
                                <ows:Identifier>local_filename</ows:Identifier>
```

```
                            </wps:RawDataOutput>
                        </wps:ResponseForm>
                    </wps:Execute>
                </wps:Body>
            </wps:Reference>
        </wps:Input>
        <wps:Input>
            <ows:Identifier>DockerImage</ows:Identifier>
            <wps:Data><wps:LiteralData>ogc/eoephackaton:latest</wps:
LiteralData></wps:Data>
        </wps:Input>
        <wps:Input>
            <ows:Identifier>DockerRunCommand</ows:Identifier>
            <wps:Data><wps:LiteralData>/opt/snap/bin/gpt -e "/S1_Cal_Deb_ML_
Spk_TC_cmd.xml" "-Pinputdata=${local_filename}" "-Poutputdata=/out/S1result"</
wps:LiteralData></wps:Data>
        </wps:Input>
        <wps:Input>
            <ows:Identifier>DockerMount</ows:Identifier>
            <wps:Data><wps:ComplexData mimeType="application/json"><!
[CDATA[{"type": "bind", "target": "/eodata", "source":"/eodata","read_only":
true}]]></wps:ComplexData></wps:Data>
        </wps:Input>
        <wps:Input>
            <ows:Identifier>DockerMount</ows:Identifier>
            <wps:Data><wps:ComplexData mimeType="application/json"><!
[CDATA[{"type": "bind", "target": "/home/ogc/.snap/auxdata/dem", "source":"/
DEM","read_only": true}]]></wps:ComplexData></wps:Data>
        </wps:Input>
        <wps:Input>
            <ows:Identifier>DockerMount</ows:Identifier>
            <wps:Data><wps:ComplexData mimeType="application/json"><!
[CDATA[{"type": "bind", "target": "/out", "source":"/tmp","read_only":
false}]]></wps:ComplexData></wps:Data>
        </wps:Input>
    </wps:DataInputs>
    <wps:ResponseForm>
        <wps:RawDataOutput mimeType="application/json">
            <ows:Identifier>file_list</ows:Identifier>
        </wps:RawDataOutput>
    </wps:ResponseForm>
</wps:Execute>
```

This shows that it might even be possible to run a process packaged in a Docker container, without even requiring explicit 'DeployProcess' functionality.

5.7.4. Experiments

The following experiments have been conducted.

1. Local execution

```
$ /usr/local/snap/bin/gpt -e "S1_flow.xml" "-Pinputdata2=/data/MTDA/CGS_S1/CGS_
S1_SLC_L1/IW/DV/2018/04/07/S1A_IW_SLC__1SDV_20180407T055900_20180407T055927_
021357_024C25_82F4/S1A_IW_SLC__1SDV_20180407T055900_20180407T055927_021357_
024C25_82F4.zip" "-Poutputdata=product"
```

2. Docker app run

```
docker run  --mount type=bind,source=/home/driesj/alldata/CGS_S1,target=/data,
readonly --mount type=bind,source=/tmp,target=/out ogc/eoephackatong:latest
```

```
/opt/snap/bin/gpt -e "/S1_Cal_Deb_ML_Spk_TC_cmd.xml" "-Pinputdata=/data/CGS_
S1_SLC_L1/IW/DV/2018/02/17/S1B_IW_SLC__1SDV_20180217T060533_20180217T060600_
009659_0116BD_A58F/S1B_IW_SLC__1SDV_20180217T060533_20180217T060600_009659_
0116BD_A58F.zip" "-Poutputdata=/out/S1result"
```

Remarks

- Docker image does not specify a user, so all processes executed will run as root, user can be set like this: RUN useradd -ms /bin/bash ogc USER ogc
- Docker image was installing gcc and gcc-c++, but this seems unneeded.
- Docker image did a yum update, might be better to leave it out, to get more predictable image.

5.7.4.1. OpenSearch

This query returns a number of S1 SLC products, available in the platform:

```
http://www.vito-eodata.be/openSearch_all/findProducts?collection=urn:eop:VITO:
CGS_S1_SLC_L1&dateOfAssociation=(2018-01-01)
```

These are the details for the first product in the list:

```
https://www.vito-eodata.be/openSearch_all/findProducts?collection=urn:eop:VITO:
CGS_S1_SLC_L1&uid=urn:eop:VITO:CGS_S1_SLC_L1:S1B_IW_SLC__1SDV_20171231T060534_
20171231T060601_008959_00FFCD_F1D9&mdDetail=full
```

Unfortunately, the OpenSearch result does not yet return the file location on the platform, which is:

```
/data/MTDA/CGS_S1/CGS_S1_SLC_L1/IW/DV/2017/12/31/S1B_IW_SLC__1SDV_
20171231T060534_20171231T060601_008959_00FFCD_F1D9/S1B_IW_SLC__1SDV_
20171231T060534_20171231T060601_008959_00FFCD_F1D9.zip
```

Luckily, there is a fixed pattern:

```
/data/MTDA/CGS_S1/CGS_S1_SLC_L1/IW/DV/{year}/{month}/{day}/{product_id}/
{product_id}.zip
```

So we can translate the result of an OpenSearch query into a list of files to process.

5.7.4.2. SciHub OpenSearch query example

The following query returns a number of products that are to be used in the Hackathon. Only problem seems to be to have a decent bounding polygon that is not too long.

```
beginPosition:[2017-06-17T00:00:00Z TO 2017-06-28T00:00:00Z] geometry:
MULTIPOLYGON ( ( (-124.5616980590070000 69.7597329383173985, -
120.8884747912599948 69.6227946704093057, -120.6641845703124858
68.0046997070312500, -112.5052261352539062 65.4994659423828125,
-101.9999999999999858 64.2405604248810391, -101.9999999999999858
59.9999999999999929, -123.8654327392577983 60.0004577636718750,
-124.5959930419921875 60.9513816833496023, -126.8194122314453125
60.7560386657714844, -129.2740631103515625 62.1518974304199148,
-130.1314849853515625 63.8682479858398366, -132.6178741455078125
64.8146667480468750, -132.3343505859375000 65.9790191650390625,
```

```
-136.4512023925781250 67.6393966674804688, -136.4307284766541102
68.8755994928747981, -134.3083673030710088 68.4917448882628861,
-134.8237124547030135 68.8966088831299999, -130.6645470088639627
70.1698270626655045, -129.5095813211379721 70.0601366636771843,
-132.5646735357719876 69.1877721075583025, -129.0187433979369871
69.6915834897223050, -127.7071199605330065 70.1875195944390953,
-128.1326847613649988 70.5379774698684940, -126.0062665758830036
69.3847449052028935, -124.5616980590070000 69.7597329383173985) ),
( (-109.9983668621582211 77.9759995442161653, -110.8591498996379983
77.4023791996747974, -113.4681886292650006 77.6945919972263823, -
109.9983668621582211 77.9759995442161653) ), ( (-109.9997816018178725
74.8822736447125550, -113.5332400552899941 74.4418064633312042,
-114.5448389104920039 74.6774789020902006, -111.2039409509199999
75.2391718029844014, -117.6377255677609810 75.3148700312898001,
-115.2479251951070012 75.7145220021956931, -117.2705855820070013
75.6768381378722950, -115.0933121976689790 75.9141480049089949,
-116.9215272085669994 75.9540951481941846, -114.9193462053499957
76.5584615036720066, -109.9991405123388404 75.5487163795551737, -
109.9997816018178725 74.8822736447125550) ), ( (-119.6310074024499812
74.0050620988056806, -115.5659439123839860 73.5355747734022884,
-118.5053305985285164 72.5527911859559680, -114.6742138861499996
73.3828481898944034, -114.0481792908909995 73.0843834539102062,
-114.6889136530499940 72.6100967431794970, -110.0003451507280232
72.5171386154111985, -109.9998550415038920 69.9996109008789062,
-116.7560806274414062 70.0005035400390625, -116.4440612792968750
69.5389175415039062, -117.4235819004919819 70.0333776124058005,
-111.4053370170420010 70.2529903079055060, -117.6759862481169847
70.7030885373091991, -118.4600936720009940 71.0423339487693966,
-115.2384654435920055 71.5296347004300799, -118.2823495867020114
71.4532951159156937, -117.7320417690929872 71.6111558249129985,
-119.0458562469499952 71.7222252601340813, -118.6497684410870761
72.5044984105474697, -122.8378231256180015 71.1042228269755014,
-125.9959464892440053 71.9849549325246869, -123.8614612339069936
73.7625852114308032, -124.8762465749349815 74.3488695598643972, -
119.6310074024499812 74.0050620988056806) ), ( (-119.8137628055140169
77.0844884604478011, -115.7360034420759973 77.3173424823808944,
-119.8561035042509957 75.8373730846247867, -123.0635936882130039
76.1754883731667007, -119.8137628055140169 77.0844884604478011) ),
( (-118.5139306387499971 75.5284167223964999, -119.3458409045809816
75.6584810669376964, -117.6433493457100070 76.1109042978966954, -
118.5139306387499971 75.5284167223964999) ), ( (-135.1827105776989981
69.4641202526612034, -133.8366374803470080 69.5144298659631943,
-136.1770482693100064 69.2024494710097002, -135.1827105776989981
69.4641202526612034) ) ) orbitdirection:DESCENDING platformname:Sentinel-1
polarisationmode:HH HV producttype:SLC'
```

5.7.4.3. CloudFerro OpenSearch

CloudFerro has something that looks like an OpenSearch endpoint:

- http://finder.eocloud.eu/resto/collections.xml
- http://finder.eocloud.eu/resto/api/collections/Sentinel1/describe.xml
- http://finder.eocloud.eu/resto/api/collections/Sentinel1/search.atom?startDate=2017-06-17&_pretty=true

But it is currently unclear if these endpoints can handle queries that are similar to other OpenSearch endpoints.

5.7.5. Proposed Alternatives

Solution 2 is my preferred solution, as the deployment step can be replaced by uploading the image to a Docker registry (which can still be a customized marketplace for remote sensing related images). Standardization will be limited to specifying the generic docker run process itself. Most existing WPS's already support this type of custom process, without modifying the WPS itself. The spec of the generic process can also be made fairly foolproof, as there is no need to have a command line template, clients should just specify the command line directly.

5.7.6. Experiences with AP & ADES

- OWS Context based AP had a relatively steep learning curve, partially because the OWS Context standard is not written with the AP in mind.
- A lot of implementations made the assumption that the deployed process should be invoked with a filename. In this case, parallelization can be achieved by the client sending multiple requests in parallel.
- Parallelization on the WPS side, by sending a list of files for instance, is not supported by the standard. However, processes packaged inside docker containers also can not really parallelize themselves, as it depends on the distributed processing system that is available to the ADES. Not sure if it is desirable to have this in the standard, as sending multiple requests on the client side is also relatively simple.

5.7.7. Other Impressions & Recommendations

- Refactor the AP to a simplified XML document that has an optional OWS Context document rather than the being an OWS Context. (Basically use composition over inheritance.) This ensure that simple client/server implementations can be written more easily, while more advanced implementations can still show a COP if desirable.

5.8. Thales Alenia Space

5.8.1. Motivation to Participate

Our participation to the EOEP Hackathon was aimed for positioning inside the earth observation with different solutions using real products from Sentinel 1, working with common tools as SNAP or WPS server development and providing our background knowledge in different tools

and services. Particularly, we are interested in developing a server back-end fully integrated with a client and make a portable solution, which can be installed in a cloud totally accessible.

5.8.2. Implemented Solution

At this chapter, it will be described the final Thales Alenia Space implemented solution, after testing different solution (can be seen in the next chapter).

During the Hackathon, several developments and configurations were done. First of all, the cloud required had to be configured in order to upload the developments there. At the beginning, it was chosen CloudFerro, but after uploading our developments there and due a lack of resources in the cloud, during the Hackathon meeting we migrate all the developments to Boreal Cloud, where we can have the same size machines.

CloudFerro configuration:

- 5 hosts: 16vCPU and 128GB RAM
- 1 host: 8vCPU and 64GB RAM
- 4 hosts: 8vCPU and 32GB RAM

As can be seen the machines were different, therefore we had a lack of performance in those machines, having as maximum of 15 workers (SNAP executions) executed at the same time. The total performance for the whole cluster was 120vCPU and 832GB RAM.

Boreal cloud configuration:

- 7 hosts: 40vCPU and 163GB RAM

Using this last configuration, we got an improved configuration using 3 nodes less and the total performance for the whole cluster was empowered 160vCPU and 120GB RAM compared with the previous option. After migrate to this configuration, the maximum number of workers were 35, therefore having 140 products to process, the whole process last around 45 minutes, using less than four executions.

The products from sentinel and the DEM were mounted on the machines as mount point, therefore were easily accessible. In the first cloud (CloudFerro), those mounting points were unmount automatically after some time, therefore this connection was not reliable enough to automate the process.

In that cloud environment were needed just to install docker-ce and include all the hosts in the same cluster using docker swarm. Once the docker swarm was initiated and working, the docker images can be downloaded in order to execute all the processes, which makes the solution fully portable. Thales Alenia Space has also written a docker compose file in order to execute the process automatically. This docker compose also limited the number of memory and CPU for the execution, therefore if the SNAP process needs 8vCPU and 24GB RAM and we have a machine with 16vCPU and 52 GB RAM, when up the docker compose, will limit the SNAP executions to two. On other machines, it will be executed more if fit in the machine requirements.

This made the migration really easy, having to install just docker and introduce the hosts in the same cluster if we want to migrate the solution.

At high scale, the implemented solution had to be based on the WPS server (2.0) and SNAP docker image. First of all an application package was designed in order to *Deploy*, *Execute* and *UnDeploy* commands. This xml file called the ADES, which connected with the WPS launched the SNAP application inside the cluster.

To make this possible, we used a queue message service (ActiveMQ) reused and modified from different Thales Alenia Space products in order to queue messages from the WPS and dequeue in the nodes (listener). After dequeue the messages the process was executed in the nodes calling the SNAP application, which requires two inputs (eodata and DEM) and after processing an output was stored in a public accessible bucket in .tif format. The processing time for a single product using SNAP is around 13 minutes.

The outline of the Thales solution can be seen in Figure 11.

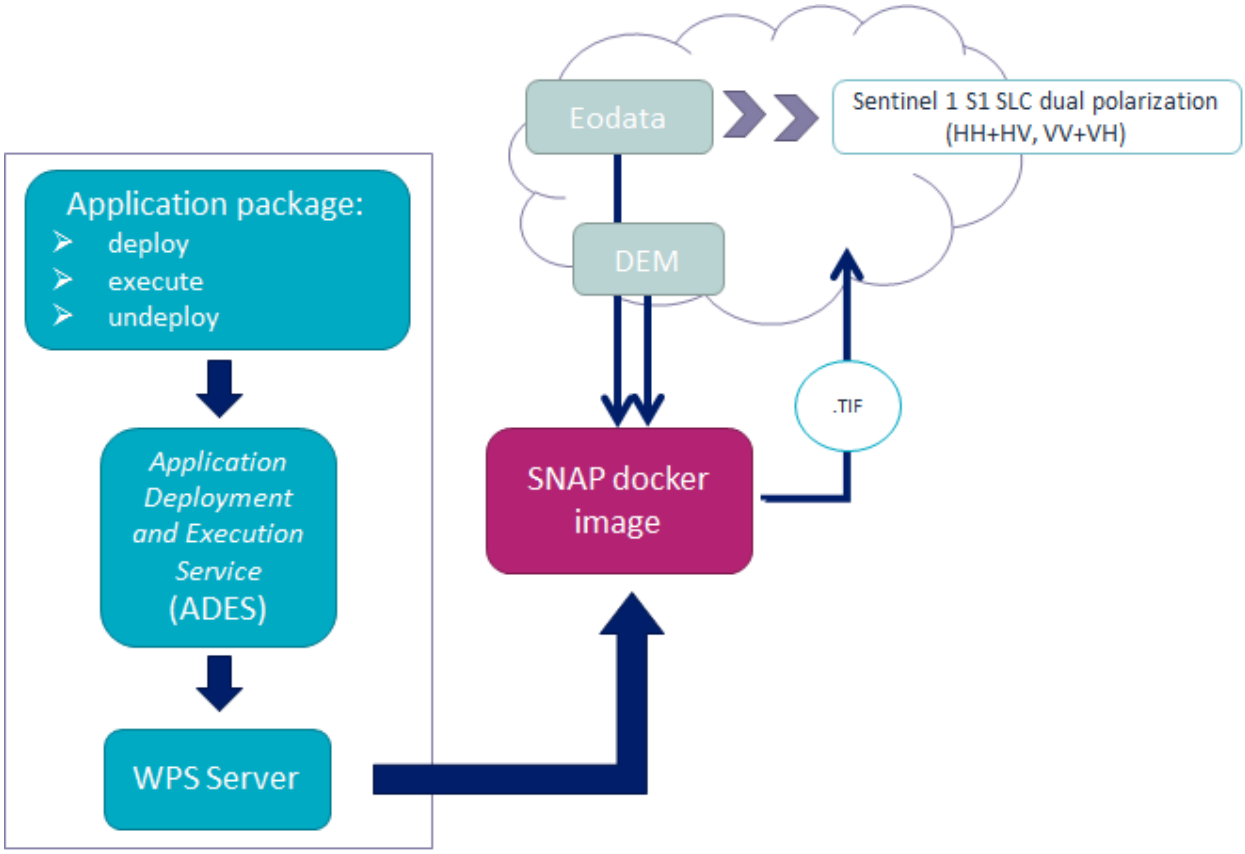


Figure 11 — Hackathon outline

The final solution was composed of a SOAP API to connect the WPS client with the North52 WPS server and this connects with the queue message service, which sends the execution to the workers (nodes). Also, in order to test the solution launching all the products to process, we used Jmeter to automate the process in case we could not integrate with a real client. The final implemented solution can be seen in Figure 12.

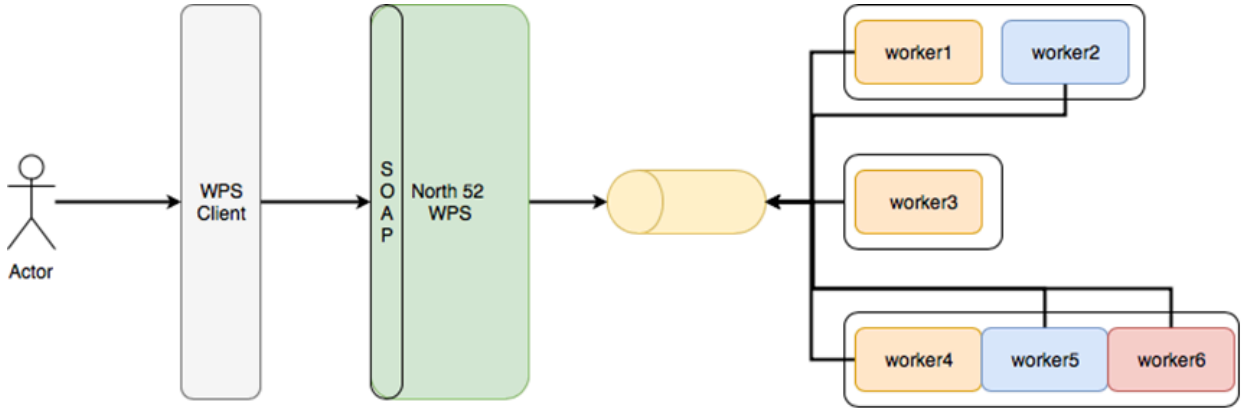


Figure 12 — Final TASE implemented solution

5.8.3. Proposed Alternatives

During the period of the Hackathon we tried several options to make the backend works. First of all our ideas was to implement FaaS (Function as a Service) integrated with docker swarm, in order to manage the balancing between all the nodes when running SNAP application to process the products. This implemented solution can be seen in Figure 13.

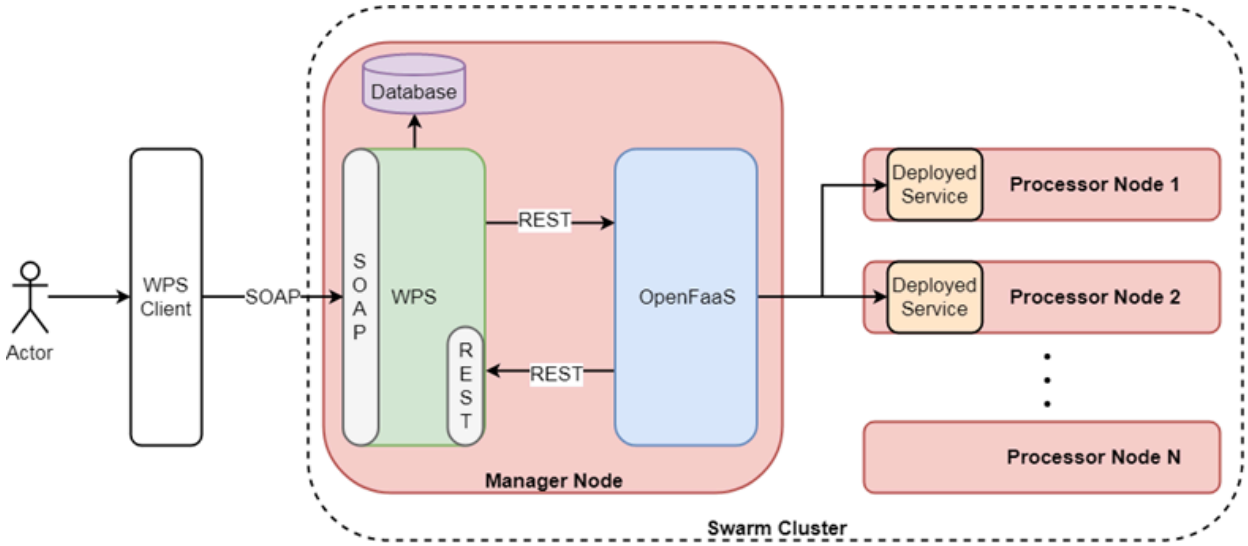


Figure 13 — Faas implemented solution of TASE

After the trial, the balancing in the FaaS solution was not working properly, so the solution had to be changed and the alternative approach was using JaaS (Jobs as a Service), which is able to run docker images balancing the load between the whole docker swarm cluster. Using this solution, in an automated environment where the tasks were run automatically, in some of the nodes those executions were not properly run, and therefore this solution was not valid to test the performance.This implemented solution can be seen in Figure 14.

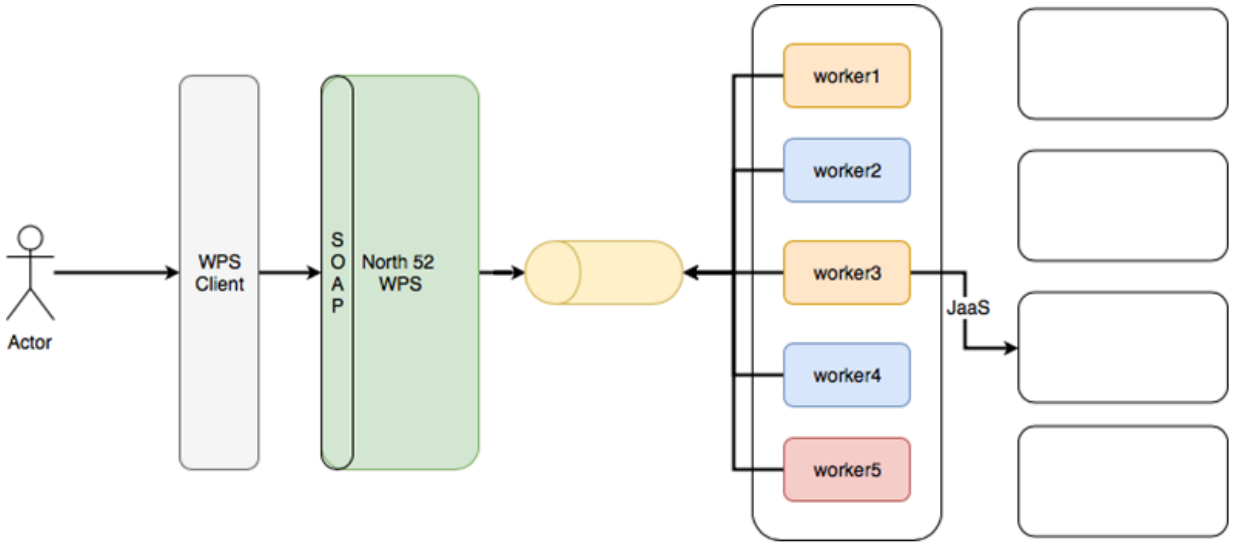


Figure 14 — Jaas implemented solution of TASE

Discarding the previous two solutions, we go on for the solution done in implemented solution section, using a Thales Alenia Space designed solution reusing as WPS server, the server is done by North52.

5.8.4. Experiences with AP & ADES

The demonstration developed by Thales Alenia Space involved the execution of the complete lifecycle:

- Application Deployment
- Application Execution
- Application UnDeployment

The WPS modified server, returns the public url where the product processed was placed when the process finish, which will be returned to the client in order to display the product.

5.8.5. Other Impressions & Recommendations

Different constrains have been found during the Hackathon implementation.

The SNAP application has two major improvements which will give more value to the tool. In the first place, the application size should be reduced in order to work properly in a docker clustered environment, which easily can be done in the docker file. The other improvement is the based on the kind of application, which right now is monolithic, which means that it cannot be split among different workers along a cluster. With this approach, the processing time of the execution cannot be reduced, so it will be needed more computers or larger computers in order to run more than one SNAP application at the same time. Using an environment based on big

data (spark), and adapting the tool, this time can be reduced in order to make calls in near real time and decreasing the processing time exponentially.

Based on the size and the computing requirements of SNAP, the cloud where the process is run should be big enough to be able to run the process. Most of the issues regarding the cloud were in terms of performance and accessing through the different open ports. Also the mount point in the cloud has to be reliable enough to allow the execution of SNAP as stated in implemented solution.

The Hackathon was valuable for Thales Alenia Space as we have not been involved in previous Testbed activities. It allowed us to develop and test the communication with clients and ADES and develop in a low time frame a fully functional solution based on the WPS standard and SNAP application built on cluster.

A

ANNEX A (INFORMATIVE) CALL FOR PARTICIPATION

A

ANNEX A (INFORMATIVE) CALL FOR PARTICIPATION

This annex includes the Call for Participation in its original form together with the clarifications and corrections applied.

A.1. Corrigenda

The following table identifies all corrections that have been applied to this call compared to the original release. Minor editorial changes (spelling, grammar, etc.) are not included.

SECTION	DESCRIPTION
Main Body	Travel cost compensation clarification added
Cloud Infrastructure	Details on OpenStack added
Technical Architecture, Section Introduction, last paragraph	Additional role added. Now, participants can provide applications together with application packages, clients instances, or server instances.
Technical Architecture, Section Cloud Infrastructure	Details on available cloud infrastructure added

A.2. Introduction

The Open Geospatial Consortium (OGC®) is releasing this ***Call for Participation*** ("CFP") to solicit proposals for the *Earth Observation Exploitation Platform Hackathon 2018* ("the Hackathon") initiative. The Hackathon builds on results from the recently concluded Testbed-13 initiative and paves the way for Tested-14 and subsequent initiatives in the context of deployment and execution of applications in cloud environments. The goal is to demonstrate that the Testbed-13 results, described in the Engineering Reports OGC Testbed-13: Exploitation Platform Application Package (OGC 17-023) and OGC Testbed-13: Application Deployment and Execution Service (OGC 17-024) are fit for purpose. Further background on Testbed-13 Cloud work is provided

by OGC Testbed-13: Cloud (OGC 17-035). Given that these Engineering Reports state several options, this Hackathon shall identify the best solution and identify any missing elements as basis for Testbed-14. More details are provided in section Technical Architecture.

Under this CFP, the OGC will provide reimbursements of travel expenses on behalf of sponsoring organizations ("Sponsors"). This CFP requests proposals from organizations ("Bidders") wishing to participate. Bidders do not need to be OGC member organizations. Any Bidder interested in participation should respond by submitting a proposal per the instructions provided herein. All proposals will be reviewed by the organization team, consisting of OGC, the European Space Agency (ESA), and Natural Resources Canada (NRCan). The organization team applies the following selection criteria: Clear statement of the bidder's motivation to participate in the Hackathon, sufficient prior knowledge and software code to implement the Hackathon architecture within the timeframe of the Hackathon, description of intended future use of the technology, and reasonable travel costs compensation request. OGC reserves the right to cap the number of participants to ensure a reasonable size of the Hackathon. Travel cost compensation is only available to entities from ESA Member States and Canada, except for Hungary.

The Hackathon is executed under the OGC Innovation Program that provides global, hands-on, collaborative prototyping for rapid development and delivery of candidate specifications to other Innovation Program activities and to the OGC Standards Program.

A.2.1. Benefits of Participation

This Hackathon provides a business opportunity for stakeholders to experiment with the latest results from Testbed-13 in the context of ESA's Earth Observation (EO) Exploitation Platforms. Participants will establish design decisions that influence the upcoming Testbed-14 and future work in the context of application handling in cloud environments operated by the sponsoring organizations. Participants will have the opportunity to network and establish direct relationships with other participants and with sponsors. They will also be part of the essential initial stages of the standardization process. During the Demonstration Meeting, participants will have the opportunity to visit the *ESA Operations Centre* in Darmstadt (Germany), including the *Main Control Room*.

A.2.2. Hackathon Policies and Procedures

O<GC Principles of Conduct> will govern all personal and public interactions in this initiative. The OGC Innovation Program Policies and Procedures apply together with the OGC Principles of Conduct and the OGC Intellectual Property Rights Policy.

One Hackathon objective is to support the OGC Standards Program in the development and publication of open standards. Each Participant will be required to allow OGC to copyright and publish documents based in whole or in part upon intellectual property contributed by the Participant during Hackathon performance. Specific requirements are described under the "Copyrights" clauses of the OGC Intellectual Property Rights Policy.

Each Participant shall contribute to the Hackathon Results Engineering Report.

This Hackathon seeks to deliver technical demonstrations in the form of software demonstrations at the Demonstration Workshop. Recording and publication of these demonstrations will be discussed at the Kick-off workshop. Software implementations remain with the participants and do not need to be made available after the Hackathon.

Experiences and lessons learned from the Hackathon will be captured in the Hackathon Results Engineering Report. This report will be edited by OGC staff with contributions by all participants.

There are no reporting duties during the Hackathon. Communication is reduced to the kick-off workshop, a Final Decision web conference, a Demonstration Preparation web conference, and the physical demonstration meeting. Please find further details here.

A.2.3. Initiative Roles

The roles generally played in any OCG Innovation Program initiative are defined in the OGC Innovation Program Policies and Procedures, including Sponsors, Bidders, Participants, Observers, and the Innovation Program Team ("IP Team").

Compared to other Innovation Program initiatives, the Hackathon provides maximal flexibility and minimal organizational and communication overhead on the participants.

A.3. Proposal Submission

In order to participate in this Hackathon, it is sufficient to fill out this Application Form. By submitting your application, you express your:

- interest to participate in the Hackathon and willingness to travel to the demonstration meeting,
- willingness to participate in the kick-off meeting either in person or remotely, and
- acceptance of the Hackathon Policies and Procedures.

Proposals must be submitted before the appropriate response due date indicated in the Master Schedule. Organizations are invited to team up with other organizations or to submit proposals individually. Teaming up can even be discussed during the setup phase of kick-off.

This CFP is open to OGC members and non-members. Information submitted in response to this CFP will be accessible to OGC and Sponsor staff members. This information will remain in the control of these stakeholders and will not be used for other purposes without prior written consent of the Bidder. Each Participant selected to participate in this Hackathon will be required to enter into a Participation Agreement contract ("PA") with the OGC.

A.4. Questions and Clarifications

Once the original CFP has been published, OGC is organizing a *CFP Clarifications phone conference*. Bidders may submit questions via timely submission of email(s) to the OGC Technology Desk (techdesk@opengeospatial.org) prior to the *CFP Clarifications phone conference* or pose questions during the conference. Question submitters will remain anonymous, and answers will be regularly compiled and published on the CFP clarifications page.

A.5. Master Schedule

The following table details the major Testbed milestones and events:

Table A.1 — Master schedule

MILESTONE	DATE	EVENT
M01	13 February 2018	CFP Release
M02	19 February 2018	Clarification Webinar (17:00 CET, login instructions)
M03	**06 March 2018**	CFP Proposal Submission Deadline (11:59pm U.S. Eastern time)
M04	10 March 2018	Bidder Notifications
M05	22 March 2018	All CFP Participation Agreements Signed
M06	**22 March 2018**	**Kickoff Workshop** (Orleans, France, 13:30-15:00CET)
M07	?? April 2018	Decision Conference: Final Decision Making for Hackathon Details
M08	19 April 2018	Preparation Conference for demo meeting
M09	**03-04 May 2018**	**Demonstration Workshop** (@ ESOC Darmstadt) Start: Thursday 14:00, ends Friday 16:00
M10	31 May 2018	Publication of the Hackathon results Engineering Report

A.6. Sequence of Events, Phases, and Milestones

Kickoff Workshop: A Kickoff Workshop ("Kickoff") is a face-to-face meeting with the option to attend via Web conference (GoToMeeting). It is guided by OGC staff and sponsors and allows participants to exchange ideas. The main purpose is to refine the Hackathon architecture and settle upon specific interface models to be used as a baseline for prototype component interoperability. Participants will be ***required*** to attend the Kickoff either in person or by Web conference.

Decision Conference: Conducted one week after the kick-off meeting, the Decision Conference web conference will decide any outstanding design decisions from the kick-off meeting. It is the conference with all participants prior to the *Preparation Conference*.

After the Decision Conference, all Hackathon activities will be conducted remotely. Communication overhead is reduced to email-list discussions.

Preparation Conference: This webinar discusses details for the *Demonstration Meeting*.

Demonstration Workshop: Physical meeting, where participants demonstrate their results.

After the demonstration meeting, all participants, sponsors, and OGC staff develop the *Hackathon Results Engineering Report*.

A.7. Technical Architecture

A.7.1. Introduction

This Annex provides background information on the OGC baseline, describes the Hackathon architecture, and identifies all requirements and corresponding work items.

The Hackathon builds on results from the recently concluded Testbed-13 initiative and paves the way for Tested-14 and subsequent initiatives in the context of deployment and execution of applications in cloud environments. The goal is to demonstrate that the Testbed-13 results, described in the Engineering Reports OGC Testbed-13: Exploitation Platform Application Package, OGC Testbed-13: Application Deployment and Execution Service, and OGC Testbed-13: Cloud are fit for purpose. Given that these Engineering Reports state several options, this Hackathon shall identify the best solution and identify any missing elements as basis for Testbed-14 and future initiatives.

ESA has started in 2014 the Earth Observation Exploitation Platforms initiative that created an ecosystem of interconnected Thematic Exploitation Platforms (TEP) for Earth Observation data. Testbed-13 focused on two key aspects:

1. To allow TEP users to develop applications on the their local machines, then to upload these to the TEP in form of Docker containers with complementing metadadata to allow for automated deployment and execution.

2. The automated deployment and execution of these containerized applications with subsequent standards-based result access using cloud platforms.

The metadata describing an application in its Docker container is bundled in a so called ***Application Package***. Details on this Application Package are described in the OGC Testbed-13: Exploitation Platform Application Package Engineering Report. An Application Package encapsulates the description of the application itself, i.e. the application metadata, a reference to the application software container, metadata about the container itself and its resource types, deployment, execution, and mapping instructions of external data to container-specific locations for input and result data, and auxiliary information such as Web-based catalogues for data discovery and selection.

The Application Package developer uploads the Docker container that includes the application to a Docker Hub and provides the Application Package to the TEP (Thematic Exploitation Platform). The TEP uses an *Application Deployment and Execution Service* (ADES) as described in the OGC Testbed-13: Application Deployment and Execution Service Engineering Report. This service, implemented as a WPS v2.0 profile, allows the dynamic deployment and execution of the Docker container on cloud infrastructure.

The Hackathon shall verify the specifications provided in the Engineering Reports, shall develop recommendations where the Engineering Reports describe several options, and shall detect any missing elements or defects that need to be corrected in future initiatives.

Participants can provide either a client application or a server application or both to the Hackathon. Alternatively, participants can provide alternative applications if these applications work on Sentinel-1 input data (or the input data can be made available by the participant). In the case of application provision, the participant needs to provide the application together with the corresponding application package. In case a participant provides both the client and the server application, then the participant agrees to pro-actively engage with others participants to ensure proper interoperability testing. At the demonstration meeting, we will test all server instances with all clients. Given that we will agree on the interface between client and server during the initial phase of the Hackathon, clients and servers can be developed independently of each other. A simple CURL client will be used as reference. Each server will receive the same two calls as described in the Hackathon Scenario below. Clients will be evaluated based on functionality and ease of use. Servers will be evaluated based on performance.

A.7.2. Hackathon Implementation

Scenario

The Hackathon shall implement the following scenario:

- Canada's forest cover an area of 348 million hectares, which is 35% of Canada's land area and 9% of the world's forested area. Because vast areas are inaccessible, researchers use satellites such as Sentinel-1 to gain valuable insights into Canada's forest ecosystem.

- The Hackathon shall evaluate the extent of wild fires based on Sentinel-1 data for the summer of 2017 over the Northwest Territories, Canada. Organizers will provide the Sentinel Application Platform (SNAP) Software Toolbox together with a pre-defined workflow packaged in a Docker container. Thus, Hackathon participants can use the Docker container "as is" and do not need to modify the container or the application. The SNAP workflow in that Docker container requires two types of data: Digital Elevation Model (DEM) data and Sentinel-1 data. Both will be made available to the participants as cloud resources.
- Participants need to develop the *Application Package* for the application in the Docker container. This is a joint effort to ensure that all Application Packages look the same.
- An *Application Deployment and Execution Service* (ADES) needs to be set up that supports two requests:
    - The registration of the *Application Package* as a new process. Here, the client will issue a WPS request that includes the *Application Package* either inline or by reference (to be decided during the Hackathon).
    - The execution of that new process, which includes the deployment and execution of the Docker container on cloud platforms provided by sponsors.
- The ADES can be setup on server operated by the participant.
- The ADES shall provide a WPS interface that allows a client to execute the processing of all Sentinel-1 scenes over the Northwest Territories. Roughly 300 Sentinel files need to be processed.
- The client application will issue the same deployment and execution requests to all ADES implementations. Two requests will be tested:
    - The first request calls the ADES to deploy the Docker Container only once and to process a single Sentinel-1 scene.
    - The second request calls the ADES to deploy the Docker Container n-times to process all Sentinel-1 scenes. It is up to the ADES to either execute the same Container sequentially or to optimize performance and to deploy the Container n-times for max parallel processing. The results do not need to be processed any further (in particular no mosaicing required).
        - The resulting scenes will be accessed by the client. In both cases, the ADES shall return a URL that contains all results in a single folder.
- Results are accessed and downloaded.

Design Decisions

The Hackathon participants need to agree on the following design decisions:

- Implementation scenario details

- Application Package details to ensure that a single Docker image works for all teams
- ADES profile details
- Result access details

Implementations

The Hackathon participants shall demonstrate the following elements:

- Overall, the capacity to process a large amount of data in an interoperable way across an heterogeneous environment
- The Application Package describing the wild fire application
- The ADES that registers the Application Package and allows deployment and execution for a single or all scenes. The ADES shall be implemented as a WPS v2.0.
- The result access mechanism
- The client (for client developers only)

A.7.3. Hackathon Trophy

The winning team will receive the Hackathon Trophy and prizes. The winner will be selected by all teams being present at the Demonstration Workshop.

A.7.4. Hackathon Baseline

A.7.4.1. Deliverables

This Hackathon seeks to deliver technical demonstrations in the form of software demonstrations at the Demonstration Workshop. Recording and publication of these demonstrations will be discussed at the Kick-off workshop. Software implementations remain with the participants and do not need to be made available after the Hackathon.

Experiences and lessons learned from the Hackathon will be captured in the Hackathon Results Engineering Report. This report will be edited by OGC staff with contributions by all participants.

A.7.4.2. Data

The Hackathon will use Sentinel-1 data provided by the sponsoring organizations. The Digital Elevation Model (DEM) data required by the SNAP workflow will be provided either online or as part of the Docker Image. In any case, this process will be transparent to the participants,

because the Docker Image will retrieve the required files automatically. This step only requires correct configuration in the *Application Package*.

All data will be made available to the participants free of charge. The data is stored on cloud infrastructure and can be accessed via Web APIs (based on HTTP REST).

A.7.4.3. Cloud Infrastructure

Cloud infrastructure will be made available by the sponsoring organizations. The cloud will support VMs running Linux. The following clouds will be provided by the sponsors:

- OpenStack version Pike September 2017 (private cloud hosted by NRCan)
- Cloudferro (ESA)

The following commercial cloud providers offered to support the activity:

- Amazon Web Services
- Cloudsigma

A.8. Glossary

CFP	Call for Participation
Application Package	Atom or XML based file that contains all information about the application that is packaged in a Docker container
TEP	ESA Thematic Exploitation Platform. In short, an EO exploitation platform is a collaborative, virtual work environment providing access to EO data and the tools, processors, and Information and Communication Technology resources required to work with them, through one coherent interface. As such the EP may be seen as a new ground segments operations approach, complementary to the traditional operations concept.
ADES	Application Deployment and Execution Service: WPS v2.0 interface that supports the registration of an Application Package as a new service and its deployment and execution.

SNAP	A common architecture for all Sentinel Toolboxes is called the Sentinel Application Platform (SNAP). The SNAP architecture is ideal for Earth Observation processing and analysis due to the following technological innovations: Extensibility, Portability, Modular Rich Client Platform, Generic EO Data Abstraction, Tiled Memory Management, and a Graph Processing Framework.
Sentinel-1	Sentinel-1 is a space mission funded by the European Union and carried out by the ESA within the Copernicus Programme, consisting of a constellation of two satellites. The payload of Sentinel-1 is a Synthetic Aperture Radar in C band that provides continuous imagery (day, night and all weather).
ESA	European Space Agency
NRCan	Natural Resources Canada
OGC	Open Geospatial Consortium
Testbed-13/14	OGC's leading annual Innovation Program initiative with a volume of ~5M USD per year.

A.9. Clarifications

The clarifications Webinar took place on Mon Feb 19th, 11:00 AM EST. The Webinar was recorded. The recording is available online (23MB file!). The following questions have been raised:

1. Can I participate as a cloud provider?

YES, you can participate as a cloud provider by making computing resources, Virtual Machines, or storage capacity available. In that case, please contact Ingo Simonis directly at isimonis@opengeospatial.org.

2. Can I provide a different application as well?

YES, if you do provide the corresponding application package in addition. In principle, we can run any type of application as long as the application is properly described, follows the data location mapping approaches agreed upon at the kick-off meeting, and loads the required data from a Web accessible archive.

3. What level of flexibility do you have regarding the architecture defined in the annex Technical Architecture?

In principle, this Hackathon has the goal to identify the best solution for the given problem, i.e. the provisioning of arbitrary applications in heterogeneous clouds to allow their execution close to the actual data. Testbed-13 has addressed this problem and documented a set of solutions and recommendations in the Engineering Reports listed above. This does not mean that we are 100% bound to what was developed in Testbed-13. We want to use it as a baseline, but allow room for discussions that may lead to modifications. Therefore we welcome additional ideas. In any case, the goal is to develop a sustainable solution for an Application Package that allows executing arbitrary applications in different environments. This solution shall lead to an open OGC standard.

4. A parallel set of studies has been run by us that led to a solution that allows running processes in cloud environments easily. We continue this work in other research projects. Would that solution be of interest to the Hackathon?

YES, as said before, the goal is to develop an open standard. Currently, we are still trying to understand the best solution for the given problem. We are aware that many research projects address this challenge. The goal here is to develop the best solution in an open, collaborative approach, following the well-established OGC consensus process that eventually leads to an open standard. We invite all research projects to join us in this activity. We invite all interested parties to join our ad hoc meeting during the next OGC Technical Committee meeting in Orleans, Tuesday, March 20th, 15:15-16:45h. At that meeting, we will discuss the topic in general with the goal to coordinate between different activities, standardization efforts, and R&D work executed in different organizations or consortia.

5. We are surprised that we need to submit a proposal in order to participate. Is it possible to join the Hackathon just to learn?

The proposal serves the purpose to allow us understanding your motivation to join the Hackathon and to learn about your background. The Application Form is very simple and should not take more than 5min to provide brief answers to the few questions. If you like, you can participate as an observer, though we appreciate your more active participation.

6. We do have a fully functional implementation of what you are trying to develop here. Does it make sense for us to demonstrate our solution?

We are interested in developing the best solution that shall be released as an open standard eventually. If you are willing to through your solution as an example into the consensus process, then we appreciate your participation. Key is the willingness to help us, not the interest to sell a given product.

7. Is the Hackathon only open to Testbed-13 participants?

No, the Hackathon is open to everyone, even non-OGC members. The results of Testbed-13 are publicly available, and there is absolutely no requirement for previous participation in the Testbed. Being familiar with the OGC process, in particular the consensus principle, certainly helps, though.

8. What is the link with Testbed-14?

Testbed-14 will build on results from Testbed-13 and this Hackathon. Testbed-14 focuses on three aspects: First, complex workflows where a workflow includes several processing steps that feed into each other and that might be distributed across several clouds. Second, security aspects, and third billing and quoting, as application consumers are not running any software locally, costs may occur on the clouds that need to be covered.

9. What profile are participants expected to have?

We expect that any organisation with EO Exploitation and/or Cloud Processing or similar experience will be able join. In essence anyone with the skills and the willingness to work in consensus towards interoperability can participate.

10. What Cloud Infrastructure will be available?

The following commercial cloud providers offered to support the activity:

- Amazon Web Services
- Cloudsigma

The conditions will be disclosed at the kick-off meeting.

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B

ANNEX B (INFORMATIVE) REVISION HISTORY

Table B.1 — Revision History

DATE	EDITOR	RELEASE	PRIMARY	DESCRIPTIONS
May 08, 2018	I. Simonis	0.1	all	initial version
May 18, 2018	I. Simonis	0.2	all	all contributions added
May 22, 2018	I. Simonis	0.3	all	first full draft
May 23, 2018	I. Simonis	0.4	summary	revised
June 11, 2018	I. Simonis	1.0	all	revised

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] : Standardized Big Data Processing in Hybrid Clouds. In: Proceedings of the 4th International Conference on Geographical Information Systems Theory, Applications and Management — Volume 1: GISTAM, pp. 205–210. SciTePress (2018).