OGC Testbed-13

Cloud ER

Table of Contents

Publication Date: YYYY-MM-DD

Approval Date: YYYY-MM-DD

Posted Date: YYYY-MM-DD

Reference number of this document: OGC 17-035

Reference URL for this document: http://www.opengis.net/doc/PER/t13-NR001

Category: Public Engineering Report

Editor: Charles Chen

Title: OGC Testbed-13: Cloud ER

OGC Engineering Report

WARNING

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

LICENSE AGREEMENT

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

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

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

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third

party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

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

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

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

This OGC Engineering Report (ER) will describe the use of OGC Web Processing Service (WPS) for cloud architecture in the OGC Testbed 13 Earth Observation Cloud (EOC) Thread. This report is intended to address issues in lack of interoperability and portability of cloud computing architectures which cause difficulty in managing the efficient use of virtual infrastructure such as in cloud migration, storage transference, quantifying resource metrics, and unified billing and invoicing. This engineering report will describe the current state of affairs in cloud computing architectures and describe the participant architectures based on use case scenarios from sponsor organizations.

Cloud computing is paving the way for future scalable computing infrastructures and is being used for processing digital earth observation data. In this EOC thread effort, data is stored in various storage resources in the cloud and accessed by an OGC Web Processing Service. The methods in which these processes are deployed and managed must be made interoperable to mitigate or avoid the complexities of administrative effort for the scientific community. In other words, the intent of this effort is to develop a way to for scientist to acquire, processing, and consume earth observation data without needing to administer computing cloud resources.

1.1. Requirements

The following requirements are to be addressed in this ER:

1. Define the WPS interface and communication protocol between clients and WPS instances that work as interfaces to the cloud computing environment.
2. Document the hosted cloud environment, processing tools, and deployment and management steps.
3. Assess the status quo and the benefits of interoperability through use of OGC WPS and web services layer.
4. Record the test experiments for reproducibility and document the use of Amazon Web Services (AWS) CloudForms and OpenStack Heat, etc.

1.2. Key Findings and Prior-After Comparison

Current cloud computing architectures have advanced from virtual hypervisors and shared compute resources to include containerization using Docker. As cloud computing continues to evolve, the scientific community seeks to achieve a more efficient process for deploying, processing, and retrieving data results. It is true that today, acquiring shared computing resources is easier than ever. However significant effort is still required in order to stage computing resources, determine compute requirements, deploy software libraries, and

general administrative tasks more suited to information Technology (IT) administrators before processing of scientific data can occur. Furthermore, efficient use of compute resources when not being utilized needs to be accounted for through scalable solutions.

This engineering report seeks to document the Testbed 13 cloud implementations in which OGC web service specifications are used to in conjunction with software containers (i.e., Docker) to establish a process flow and retrieve results for data processing functions. The goal is to reduce the amount of administrative work required to stage compute resources and process data, and simplify data retrieval, specifically for earth observation data which can vary depending on data size or volume.

The results are TBD

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

The Working Group identified for the review of this engineering report is the Big Data Domain Working Group (DWG). The Big Data DWG's current purpose statement defines their scope of work to include Big Data interoperability and especially analytics. However, key members of the DWG are interested in expanding the scope to include cloud in their discussions. Particularly, at least one member of the DWG is also working on multi-cloud discovery and access to Earth Observation data within cloud compute architectures.

The work of the EOC thread aligns with the Big Data DWG interests due to the use of earth observation data storage and processing of this data using OGC WPS standards in a dynamic cloud deployment for interoperable ease of access for scientific study. The goal of this effort is to improve the way EO data is disseminated, processed, stores, and searched without scientists needing to understand how to administer cloud computing resources. This engineering report describes future architectures and deployment methods for cloud architectures utilizing OGC WPS. The use of these architectures may assist in future developments of data processing including use cases such as big data processing. Other OGC W*S services may be considered as future follow-on effort.

1.4. Document contributor contact points

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

Name	Organization
Charles Chen (Editor)	Skymantics
Tom Landry	CRIM
Ziheng Sun	GMU
Chen Zhang	GMU

Table 1. Contacts

1.5. Future Work

The work contained herein lead toward future interoperability tests across private-public cloud implementations. Future developments in cloud computing technology and container technology may lead to future work in interoperability research.

1.6. Foreword

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

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

Chapter 2. References

The following normative documents are referenced in this document. * OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] * OGC 05-007r7, Web Processing Service [http://portal.opengeospatial.org/files/?artifact_id=24151] * OGC 14-065, OGC® WPS 2.0 Interface Standard [http://docs.opengeospatial.org/is/14-065/14-065.html] * OGC 06-042, OpenGIS Web Map Service (WMS) Implementation Specification 1.3 [http://portal.opengeospatial.org/files/?artifact_id=14416] * OGC 09-110r4, OGC® WCS 2.0 Interface Standard - Core, version 2.0.1 [https://portal.opengeospatial.org/files/09-110r4]

Chapter 3. Terms and Definitions

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

3.1. Apache web server

```
Implements the HTTP protocol to serve web pages and other web content.
```

3.2. CA Siteminder

```
CA Siteminder, now called “CA Single Sign-On”, is a commercial product
providing Single Sign-On functionality.
```

3.3. CGI

```
The Common Gateway Interface (CGI) allows web requests to interact with
server executables.
```

3.4. Container

```
A method of operating system virtualization that allows packaging and running
an application and its dependencies in resource-isolated processes.
```

3.5. Cookie

```
Information stored with a user’s web browser to help a website identify the
user for subsequent communication.
```

3.6. DACS

The Distributed Access Control System is software that can limit access to any content served by an Apache web server. In other modes of operation, DACS can be used by other web servers and virtually any application, script, server software, or CGI program to provide access control or authentication functionality.

3.7. DACS Federation

A DACS federation consists of one or more jurisdictions, each of which authenticates its users, provides web services, or both.

3.8. DACS Jurisdiction

Jurisdictions coordinate information sharing through light-weight business practices implemented as a requirement of membership in a DACS federation.

3.9. DACS Cookie

A cookie set in a user's browser by DACS when the user is authenticated. Used to identify and verify the user.

3.10. Elasticity

The ability to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner. Elasticity in cloud infrastructure involves enabling the hypervisor to create virtual machines or containers with the resources to meet the real-time demand.

3.11. Hybrid Cloud

A composition of two or more clouds (private or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment models with the ability to connect collocation, managed, and/or dedicated services with cloud resources.

3.12. Hypervisor | Virtual Machine Monitor (VMM)

```
A computer software, firmware or hardware that creates and runs virtual
machines.
```

3.13. Image

```
A template for creating new instances.
```

3.14. Instance | Virtual Machine

A virtualized computing resource which provides functionality needed to execute an operating system. A hypervisor is used for native execution to share and manage hardware, allowing for multiple environments which are isolated from one another, yet exist on the same physical machine.

3.15. JavaScript

```
A programming language traditionally used by websites and interpreted on web
browsers.
```

3.16. JQuery

```
A JavaScript framework.
```

3.17. Same Origin Policy

```
A security policy that prevents scripts from accessing content other than
from where they originated.
```

3.18. Scalability

```
The ability of a system, network, or process to handle a growing amount of
work, or its potential to be enlarged to accommodate that growth.
```

3.19. Single Sign-On

```
A system which enables a user to authenticate once in order to access
multiple applications and/or services.
```

3.20. Spring Security

```
The security component of the Spring Framework.
```

3.21. Web Service

```
Provides information over web protocols (such as HTTP) with the primary
intent to be machine readable (ie: in XML or JSON format).
```

3.22. Abbreviated terms

- API Application Program Interface
- AWS Amazon Web Services
- CGI Common Gateway Interface
- CLI Command Line Interface
- CRIM Computer Research Institute of Montréal
- DWG Domain Working Group
- GPT Graph Processing Tool
- SAR Synthetic Aperture Radar
- SLC Single Look Complex
- RSTB Radarsat-2 Software Tool Box
- SNAP Sentinel Application Platform
- SQW Standard Quad Polarization

- SSH Secure Shell
- TIE Technical Interoperability Experiment
- VM Virtual Machine
- WCS Web Coverage Service
- WMS Web Map Service
- WPS Web Processing Service

Chapter 4. Overview

This engineering report describes the development effort in using a cloud computing environment for Earth Observation (EO) data integrated with OGC web services for improved interoperability. The cloud environment hosts data processing tools for deployment, management, and processing of EO data using an OGC Web Processing Service (WPS). Figure 1 describes the high level architecture for the cloud computing environment.

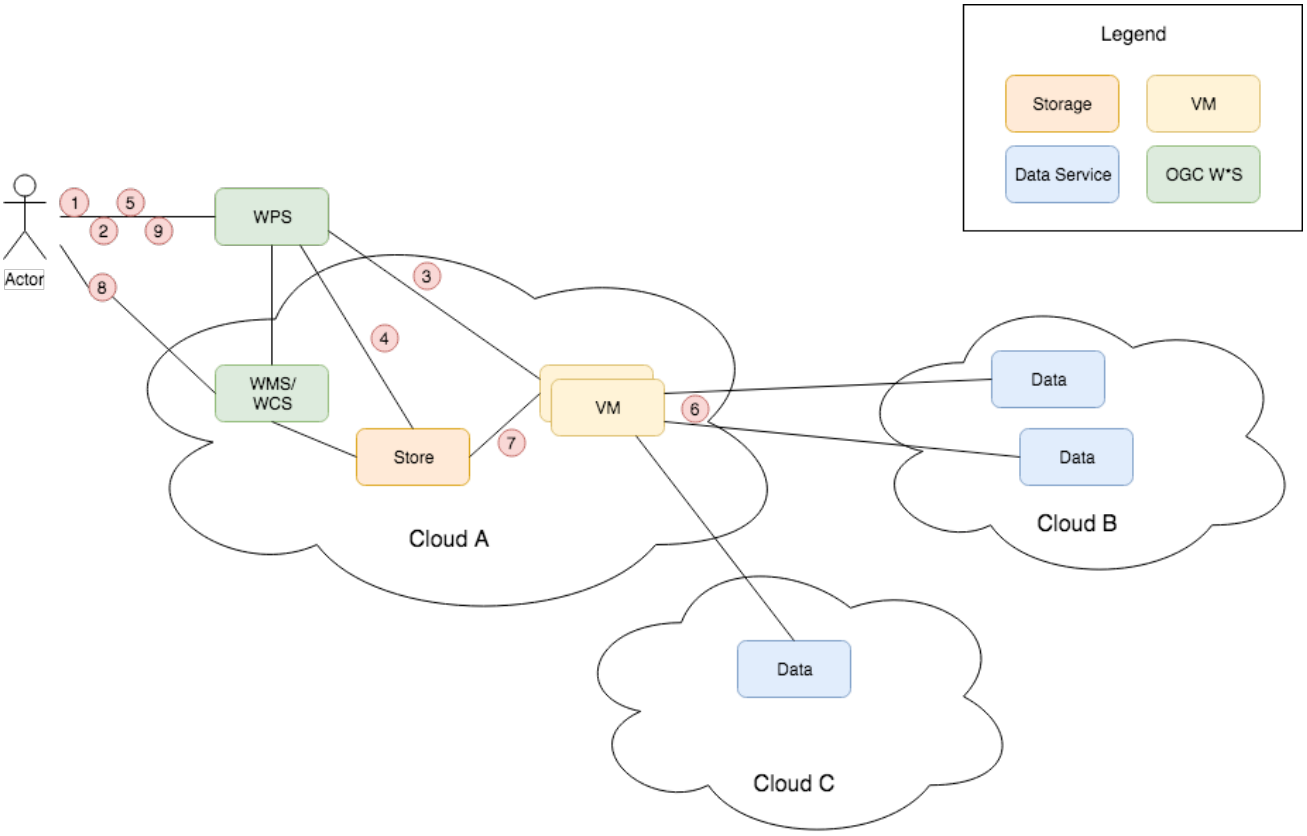


Figure 1. Cloud Environment Overview

A user accesses the cloud environment via a WPS client (i.e., user dashboard). The user inputs WPS requests containing parameters which instruct the WPS Server to allocate cloud resources by taking advantage of the flexible, dynamic, and scalable nature of a cloud computing infrastructure. In this wau, users can make use of cloud computing with minimal interaction with the IT administration of the cloud itself. The use case explored in this activity involves the processing of EO data using the RADARSAT-2 Tool Box (RSTB) deployed into a cloud environment to process and return processed images using a WMS/WCS server. The envisioned execution process flow is as follows:

1. Software toolbox deployment, configuration and maintenance
2. Receiving job request through OGC WPS. [1: It is imperative the implementations use open source software by Array Systems Computing RadarSat-2 Toolbox (RSTB) and adds a WPS layer on top. The Array's open-source RSTB can be downloaded from

http://step.esa.int/main/download/;] (WPS being part of the cloud optionally)

3. Allocating resources dynamically based on demand and performing job splitting/scheduling/processing/tracking
4. Allocating required scratch storage for intermediate and final product
5. Supporting batch processing multiple RADARSAT-2 or other SAR/optical images (a generic big /high volume data processing)
6. Capable to integrate or exchange data from different sources hosted in a cloud environment (and/or traditional computing network)
7. Gathering output elements into final products
8. Disseminating final products through OGC Web services such as WCS and WMS
9. Providing cloud usage statistics and user notification

4.1. Implementation Goals

Two separate implementations have been developed by Computer Research Institute of Montréal (CRIM) and George Mason University (GMU), respectively, to support the testbed experiments. Each implementation has been developed independently using various software tools and specifications to achieve the same implementations goals and capabilities as follows:

- The ability to leverage large pools of computing resources from the hybrid cloud and traditional dedicated servers
- The ability to easily create or expand the number of instances VMs when needed and not need to reconfigure how WPS services are advertised
- The ability to control access and authentication of users of the web services and instances VMs
- The ability to log usage and jobs being performed
- Must allow for the integration of WPS 2.0 including constructs for service discovery service capabilities job control execution and data transmission of inputs and outputs in a chain
- Will have a web-enabled dashboard of current usage and capacity of the computing resource of the cloud infrastructure, ideally this dashboard can be integrated into the WPS dashboard
- Monitors the execution, requests, responses, etc.
- Publish and consume OGC services like WMS, WCS, and WFS
- The operation cloud model must be easily reproducible and documented

- The operational cloud model should general enough to support any type of Earth Observation data processing supported by RADARSAT-2 toolbox
- Delivery of scripts that allow for the reinstallation of the cloud environment

4.2. Expanded Architecture

Based on the architectures described earlier, the overview architecture can be expanded to describe additional components within each category as shown in Figure 2 below. The WPS Client may also include a graphical user interface and security functions. The WPS Server may include synchronous or asynchronous monitoring such as a polling function to get status of a running process. The VMs may contain Docker (see Section Docker Containers) containers and the RSTB processes to process EO data and produce a result. Additionally, a Docker Registry may be used for distributing the Docker images. Data may be accessed in external clouds, and the resulting data may be shared across shared cloud storage.

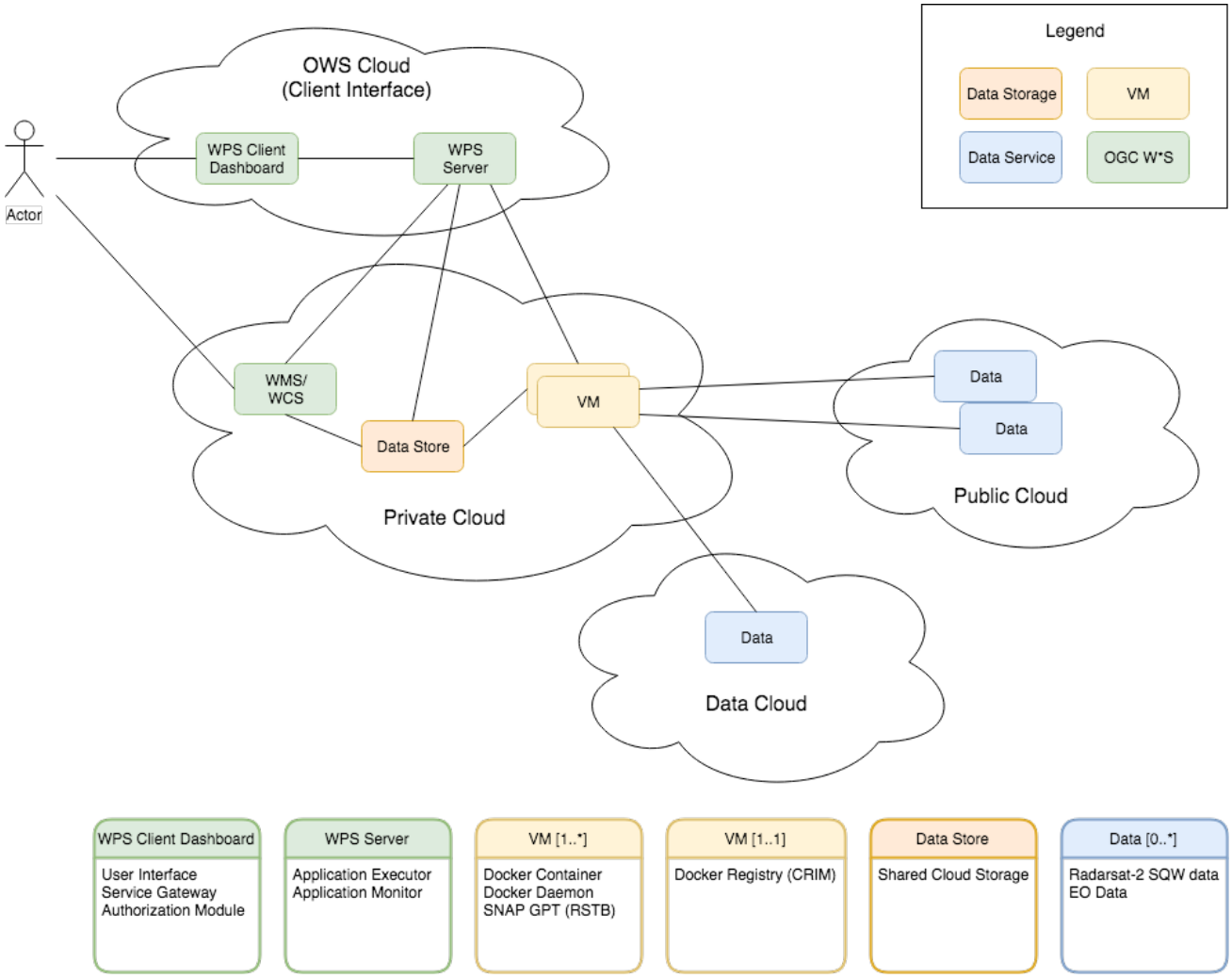


Figure 2. High Level Expanded Cloud Architecture

In general, the location of the WPS server is agnostic from the cloud environment and can be contained as a separate server or within a cloud server. While the WPS may not be represented within a cloud environment in Figure 1, the actual deployment may be within

the same or separate cloud environment. However, it is important to note that the cloud environments are able to retrieve data (i.e., RADARSAT-2 Samples) via an external cloud. In this combined expanded architecture, the WPS client contains multiple functions such as application execution and monitoring functions. The virtual machines deployed in the cloud architectures contain application containers using Docker. Both architecture implementation follow this expanded architecture with various differences in how each configures their Docker deployments and VM provisioning.

4.3. Development Approach

The development process for the cloud environments were broken out into a Work Breakdown Structure as shown in Figure 3. This structure was developed and agreed upon by all participants to ensure all implementation aspects of the cloud environment were considered.

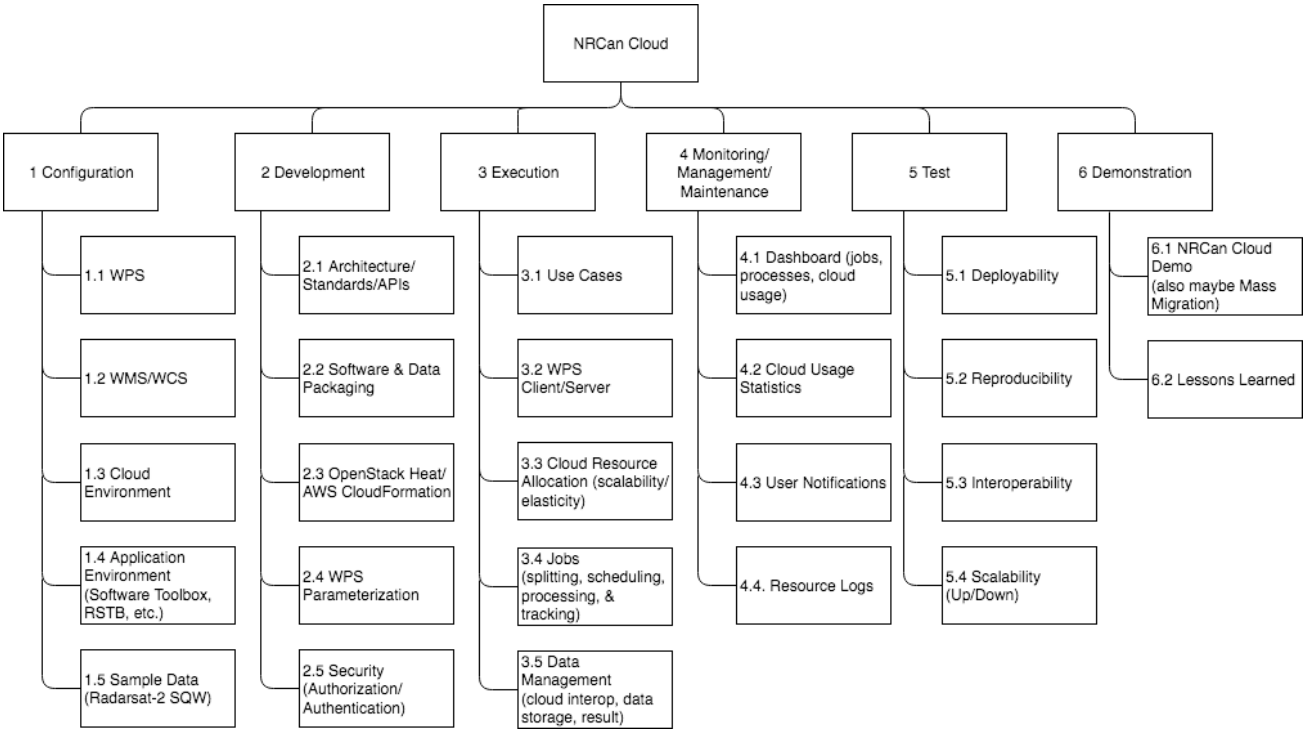


Figure 3. Testbed 13 EOC Thread - Cloud Development Work Breakdown Structure (WBS)

4.4. Outline

The engineering report describes the architecutre, configuration, execution, test, and demonstration of the component implementations as follows:

- Section 5 Architectures describes the high level architectures as implemented by the participants, GMU and CRIM.
- Section 6 Configuration describes the configuration for WPS, Cloud, Containers, Data Storage and Retrieval, and Metrics collection.

- Section 7 Execution describes the execution processes of the WPS components including operations, parameterization, and job management.
- Section 8 [Security] describes various security approaches.
- Section 9 Test Experiments describes the Technical Interoperability Experiments (TIEs).
- Section 10 Testbed 13 Demonstration describes the OGC Testbed 13 demonstration scenarios.
- Section 11 Summary discusses the final conclusions and future work.

Chapter 5. Architectures

In Testbed 13 Earth Observation Clouds (EOC) thread, two implementations of a cloud environment have been developed in order to compare and contrast different approaches while striving for interoperability. At a high level, both implementations utilize virtualized hosts in a cloud environment and Docker software containers. As seen in Figure 1 in the Overview, an Actor must be able to execute a series of operations via a WPS interface in order to interact with the cloud environment.

5.1. GMU Cloud Architecture Framework

GMU has established an architecture framework as seen in Figure 4 below. The framework is divided into four layers: Client, OGC Web Services, Cloud environment, and Internet (i.e., external network). The client consists of a browser-based application dashboard containing stored WPS requests, virtual host locations, and additional features such as priority and resource allocation functions.

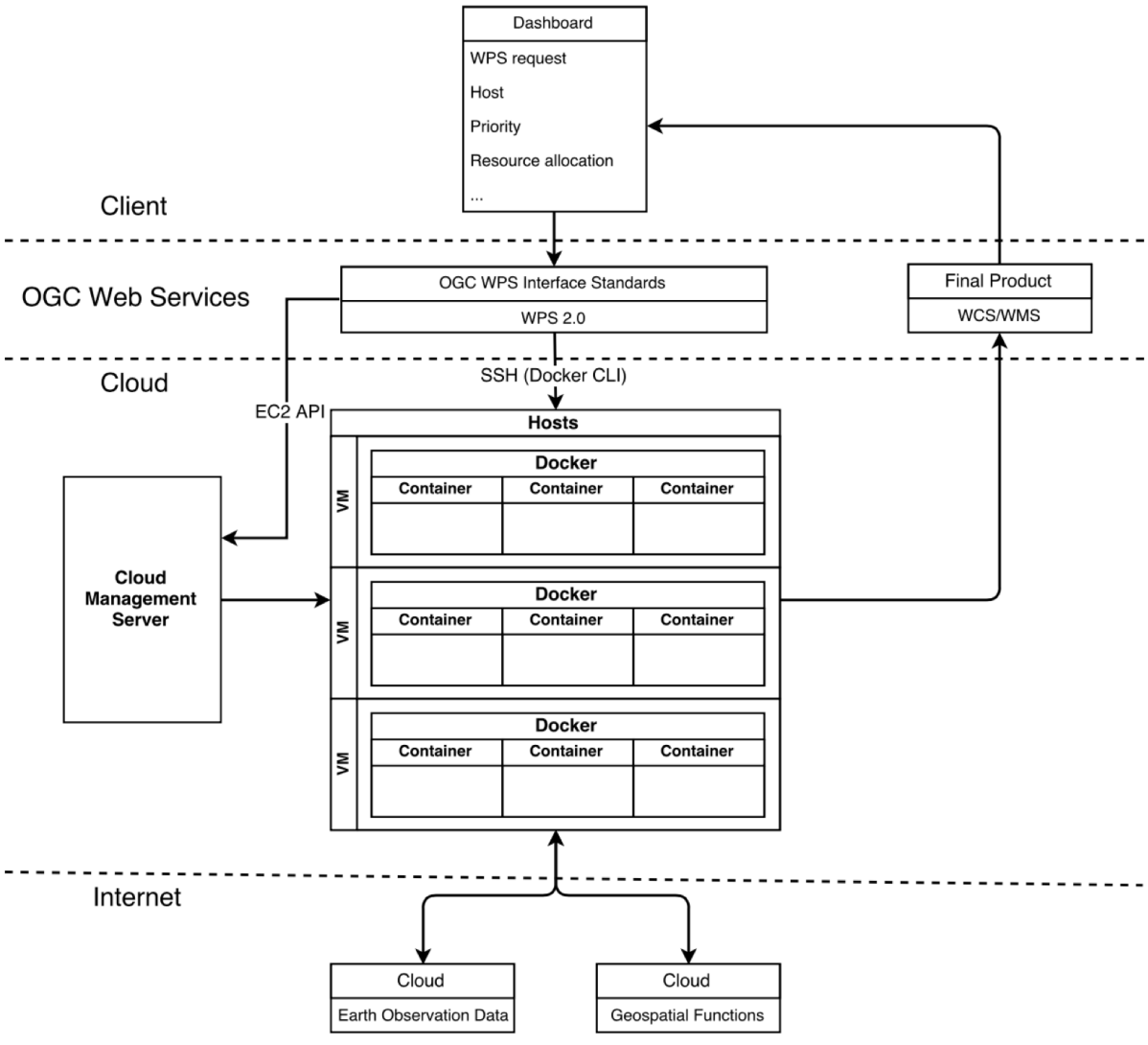


Figure 4. GMU Cloud Architecture Framework

All the capabilities of the WPS server and cloud environment are performed via the WPS interface. In other words, WPS is the only exposed interface for users to access the cloud and Docker container functionality. The user (i.e., Actor) can perform WPS request functions through the client to an OGC WPS 2.0 implementation which also resides in the cloud. Job requests/responses with the cloud are managed by Secure Shell (SSH) using the Docker command-line interface (CLI). The WPS server RSTB processes requests using the Docker CLI by deploying Docker containers in a Virtual Machine (VM) (i.e., cloud instance). The processes contained within each job request publishes the final products via an OGC Web Coverage Service (WCS) service. The final product is returned through OGC WCS. The WPS client receives reference URL to the WCS containing the results.

Additionally, the WPS and cloud environment can interact over the Amazon Elastic Compute Cloud (EC2) API which can manage the deployment and scalability of VM instances using the Amazon Cloud Management Server. The cloud is capable of communicating with other cloud blocks to exchange data or call geospatial functions from different sources hosted externally to the local cloud environment. It should be noted that while there is a good compatibility guide to the OpenStack implementation of the EC2 API, many commands are not supported [EC2].

5.2. CRIM High Level Architecture

The CRIM High Level Architecture contains a WPS built on the open-source PyWPS 4.0.0, an OGC WPS 1.0 specification. Since PyWPS uses Python as its code-base, several software applications and libraries adopted for this implementation are also Python based. The current solution targets Python 2.7, in part because PyWPS 4.0 still exhibits input/output issues with Python 3.6. Python 2.7 is currently being phased out of CRIM's research infrastructure, in favor of Python 3.6. The cloud environment is developed using OpenStack Juno, a free and open source cloud operating system that controls compute, storage, and networking resources in a data center. Docker 1.10 is used as an application package containing user's process and pulled on demand from the Docker registry, but also as a way to bundle and deploy all other required components of the system. More details regarding each software package utilized can be found in Appendix C - Software Packages. The CRIM High Level Architecture is represented in Figure 5 below.

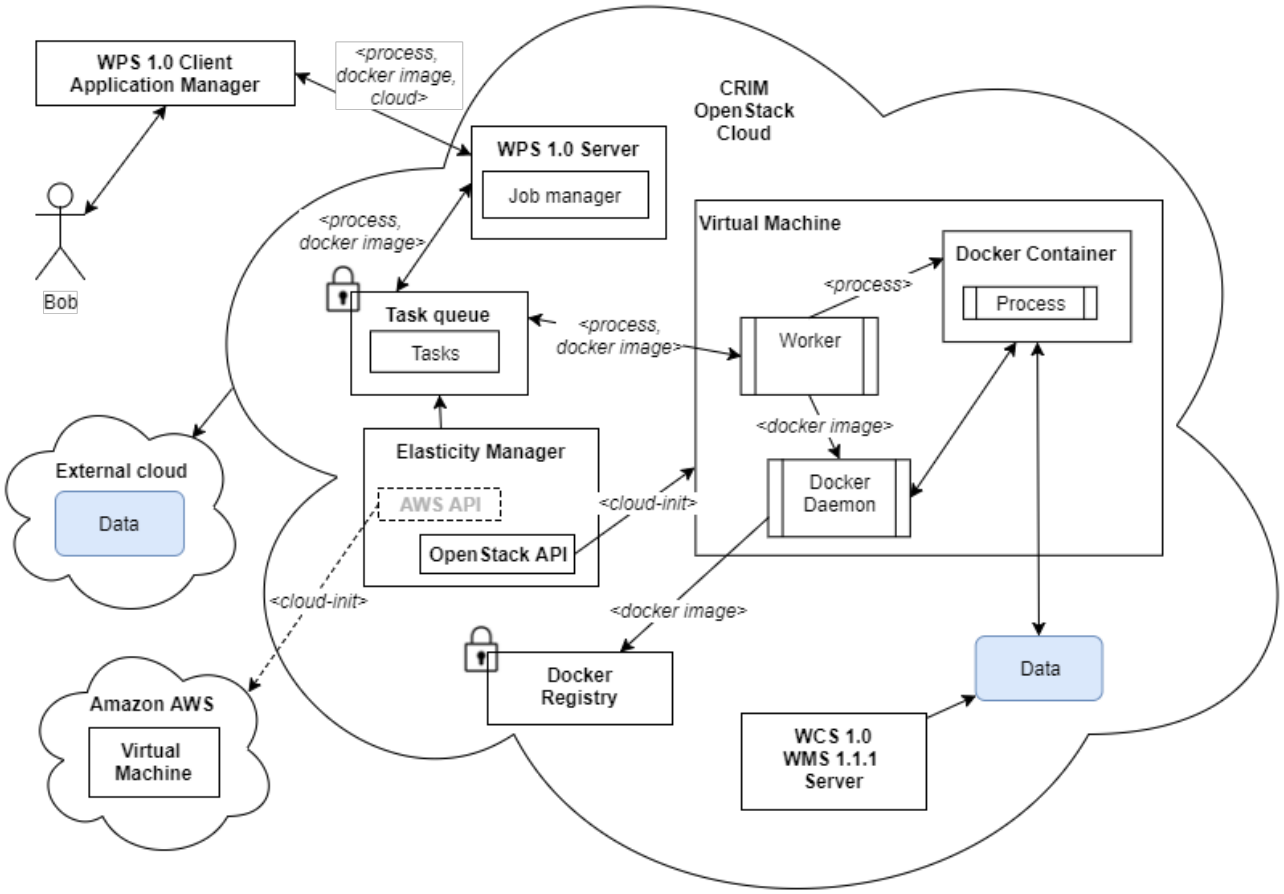


Figure 5. CRIM High Level Architecture

In this architecture view, the WPS 1.0 Client/Application Manager interacts with a WPS 1.0 Server. The server contains a job manager that splits jobs into tasks. A request containing parameters for processes, cloud, and application registry can be sent to the WPS server to manage the distribution of tasks across one to many task queues within multiple cloud environments. An elasticity manager monitors the tasks queues and automatically triggers the deployment of new VMs when predefined system load thresholds are met. The elasticity manager has been developed by CRIM and has been in operation for the last 3 years in a software research platform called VESTA (http://vesta.crim.ca/index_en.html). Additionally, the input and output parameters of the Docker application packages, as well as execution status, are passed via the task queue. The data results are stored on a shared fileserver and can be accessed either in their raw form or as WCS 1.0 or Web Map Service (WMS) 1.1.1 layers implemented using GeoServer. Once the results are computed, the WPS receives the output in the form of a reference URL to the WMS Server where the user can retrieve the image.

Chapter 6. Configuration

This section describes the key components of the architectures described in the [Architecture] section. In general, the configuration will describe each WPS implementation, cloud environment, and additional interfaces in further detail.

6.1. WPS

6.1.1. GMU WPS

GMU has developed their WPS using the OGC WPS 2.0 specification using Java in the Eclipse development environment. The GMU WPS was developed by the Center for Spatial Information Science and Systems (CSISS) including a web-based GMU WPS Dashboard tool to support the OGC Testbed 13 demonstrations. The demonstration version of GMU WPS Dashboard can be accessed at: http://cloud.csiss.gmu.edu/GMUWPS/ (Note: This is a demonstration environment and is available based on best-effort.) Figure 6 below contains a screenshot of the GMU WPS Dashboard.

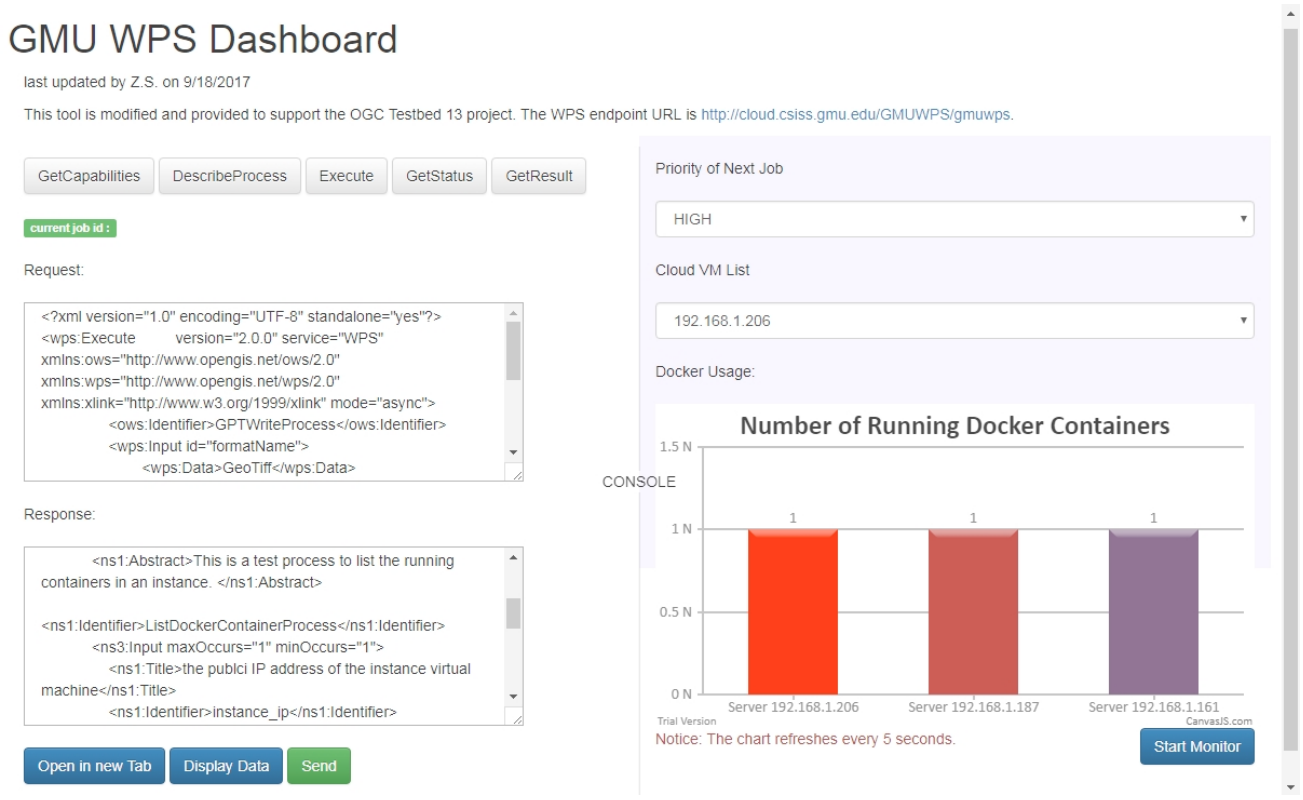


Figure 6. GMU WPS Dashboard

The GMU WPS Dashboard provides an interface to send requests and receive responses via WPS operations including GetCapabilities, DescribeProcess, Execute, GetStatus, and GetResult and supports both synchronous and asynchronous processes. The GMU WPS can interface with the cloud management service to create new VMs, list all running VMS, and shut down VMs using the Amazon Elastic Compute Cloud (EC2) API. RSTB is deployed in

software containers (i.e., Docker) on virtual machines which are accessible via the GMU WPS Dashboard. Optimized for the Testbed 13 EOC thread, the client displays the current computing statuses such as Job Splitting, Job Priority, Cloud VM Usage, and Docker Usage.

6.1.2. CRIM WPS

CRIM developed a novel job scheduler for PyWPS 4.0.0 using Celery, a distributed task queue written in Python. PyWPS 4.0 currently implements OGC WPS 1.0 specification, while OGC WPS 2.0 support is still in development. In the WPS 2.0 standard, the core concepts for job control, process execution and job monitoring are particularly well addressed by task queues. However, in order to ensure easier implementation and better interoperability with existing servers, platforms and libraries, it was decided that the WPS 1.0 standard was sufficient. It was considered that for Tested 13, CRIM's existing WPS 1.0 Application Manager Client working alongside Flower, a real-time monitor and web admin for Celery, constitute an acceptable alternative to a full-fledged WPS 2.0 client. Figure 7 depicts a screenshot of PAVICS, Open Source component RS-40 in CANARIE software registry (https://science.canarie.ca/researchsoftware/researchresource/main.html?resourceID=103). CRIM develops and uses PAVICS as an App Manager dashboard that interfaces with WPS 1.0 servers and services.

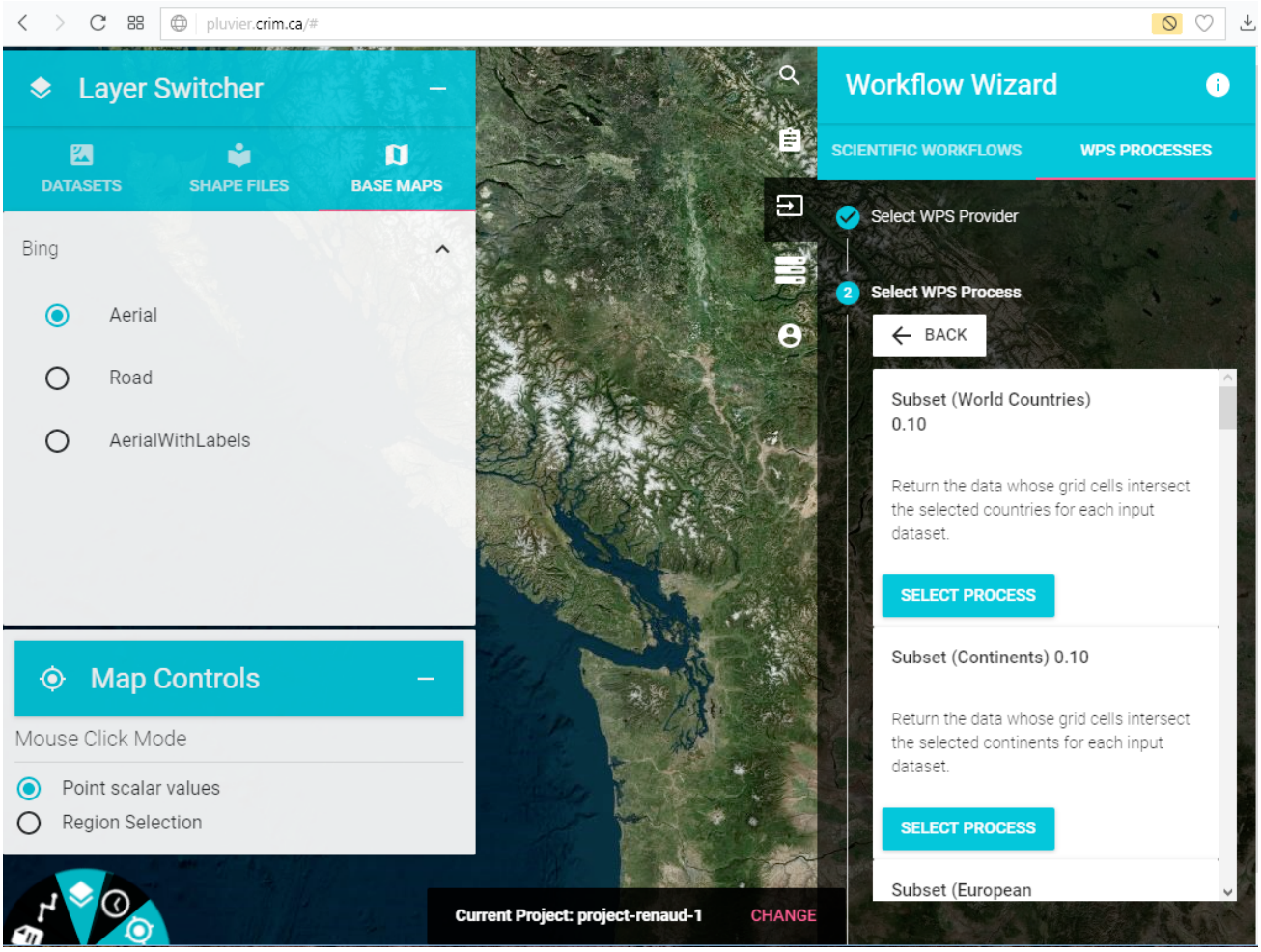


Figure 7. PAVICS, CRIM's App Manager Dashboard

If required, WPS 2.0 can be integrated in the solution in various ways: * Advance application packaging to support OWSContext files. This way, ESA-TEP that supports WPS 2.0 could transparently and dynamically register CRIM's application package. * Advance support of WPS 2.0 (GetCapabilities, DescribeProcess) in the App Manager so external WPS 2.0 servers can be discovered and processes launched, without explicit knowledge of application packages. * Advance support of WPS 2.0 in PyWPS so it can fully leverage EOC Testbed-13 findings. The enhancement request was added early 2016 to target version 4.2.0 of PyWPS (https://github.com/geopython/pywps/issues/74). * Integrate CRIM's task queue scheduler in GMU's WPS 2.0 server. This way, requests addressed to GMU WPS 2.0 server could use an interoperable interface to publish new tasks to be processed by clouds implementing CRIM's approach.

6.2. Cloud Environment

6.2.1. GMU GeoBrain

A series of state-of-the-art technologies and the latest open-source libraries/software are adopted for use in GMU's cloud platform including Apache Cloudstack, EC2 API, Docker, and Radarsat-2 Software Toolbox. The cloud platform consists of a cluster of hosts which including a primary host and a series of agent hosts. Each host contains several instances. The primary host is configured as the firewall/gateway router in the cloud environment, which makes it possible for all hosts and instances in the cloud to be accessible through the Internet. The architecture of the cloud platform is described as in Figure 8 below.

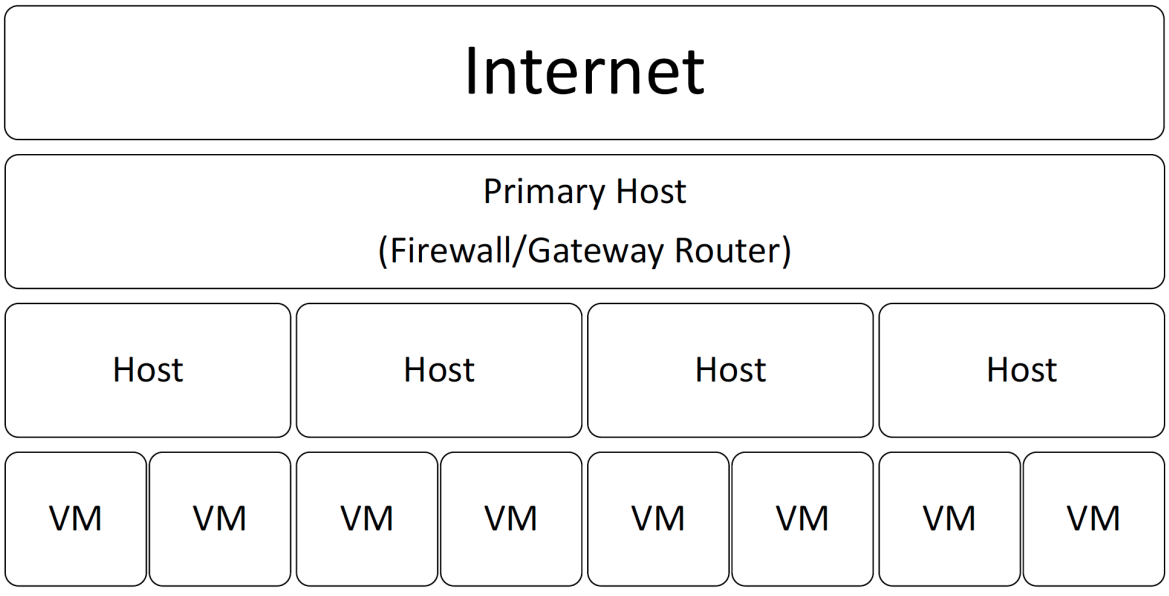


Figure 8. GeoBRAIN Cloud Hierarchy

The GMU GeoBrain Cloud (http://cloud.csiss.gmu.edu) uses the open source Apache CloudStack software to power the private cloud environment which is run by the CSISS. Apache CloudStack is open source software designed to deploy and manage large

networks of virtual machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. The Amazon EC2 API is a web service that enables you to launch and manage cloud instances in Amazon's data centers. Amazon EC2 provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Since Apache CloudStack provides the Amazon EC2 compatible API, this interoperable API stack is used to bridge Amazon EC2 API and GeoBrain Cloud which provides users with a standardized interface to access a hybrid cloud platform as shown in Figure 9.

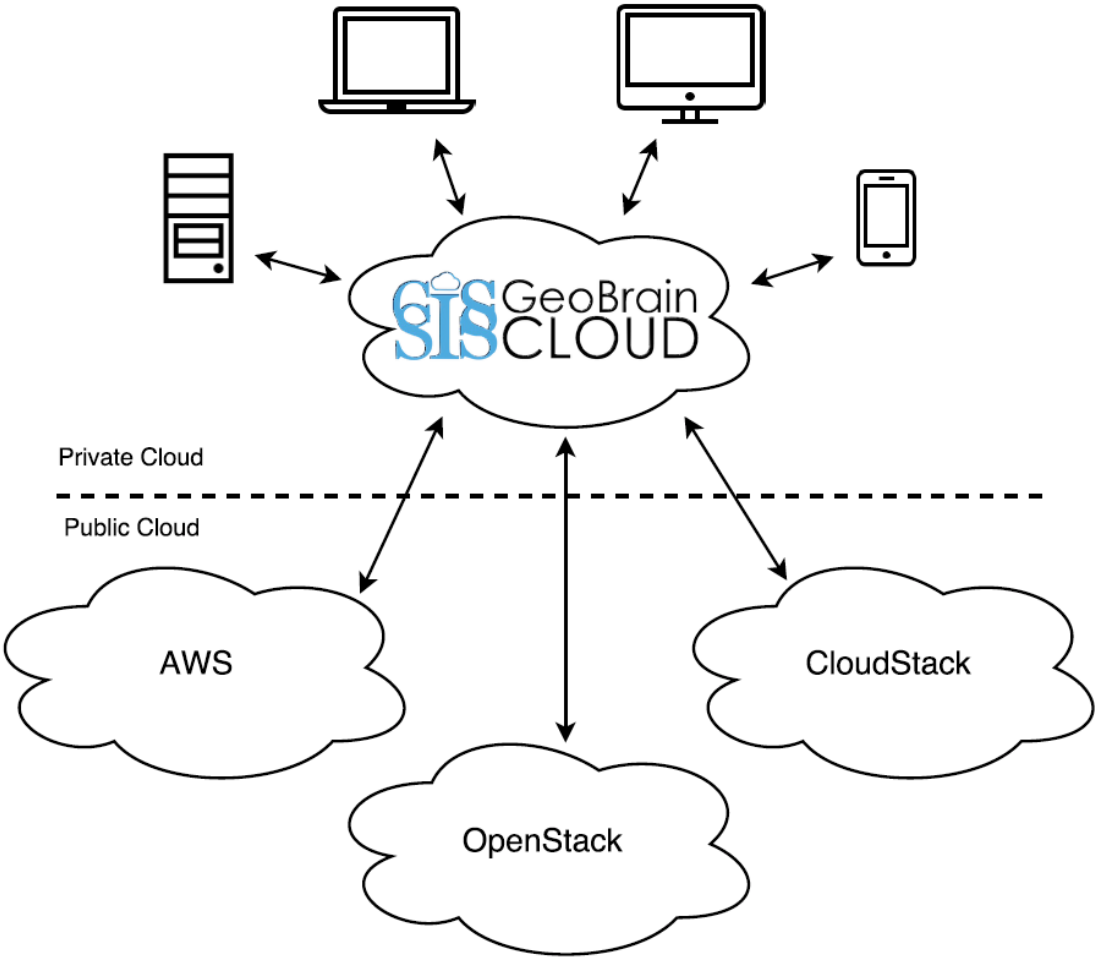


Figure 9. Architecture for GeoBrain Cloud Platform"

6.2.2. CRIM Research Infrastructure

CRIM uses OpenStack for its private and shared cloud environment. A set of resources were provisioned exclusively for Testbed 13. The WPS 1.0 server is deployed on a dedicated host in the CRIM.ca domain and is reachable via HTTPS over the web. The elasticity manager and message broker (i.e., task queues) are hosted on VMs behind a firewall. The manager controls a dynamic pool of three to twelve VMs, inaccessible externally but always available internally to process jobs from participants and affiliates. Core execution components, such as the worker and the Docker container (stored as Docker images), are templated using cloud-init files, multi-distribution packages that handles early initialization of a VM.

Figure 10 shows the nested hierarchy of the CRIM cloud down to the processes. Note that the ownership of worker to container can be reversed. A container could be started so that its entry point process launches a worker. Inversely, a new instantiated virtual machine can easily launch a daemon worker, that in turn, instantiates a Docker container. For the current implementation, CRIM selected the latter approach, mainly because the nature of the system elasticity was provided at the virtual machine level. Alternate implementations could address elasticity with distributed Docker containers. In that last case, Docker Swarm or Kubernetes can be considered as prime candidates.

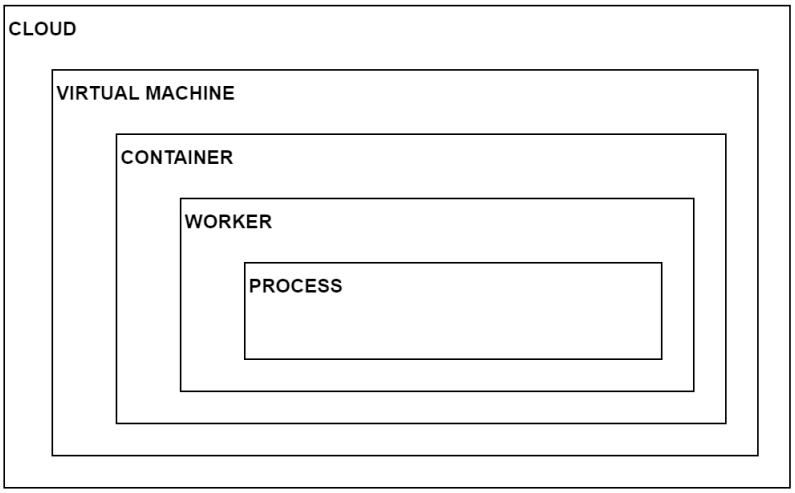


Figure 10. CRIM Cloud Hierarchy

Once created, VMs can also be stored as snapshots so that subsequent instantiations are notably faster. As shown in the following figure, the proposed solution offers several potential configurations as a hybrid cloud. The application manager can send requests to various designated WPS servers. A WPS server can publish tasks on different clouds. The elasticity manager can create VMs on its own cloud or on external clouds such as Amazon AWS. This flexibility allows future enhancements of application deployment and execution services that could minimize execution costs or time, all while taking in account user preferences and quotas. The described hybrid cloud composition is depicted in Figure 11.

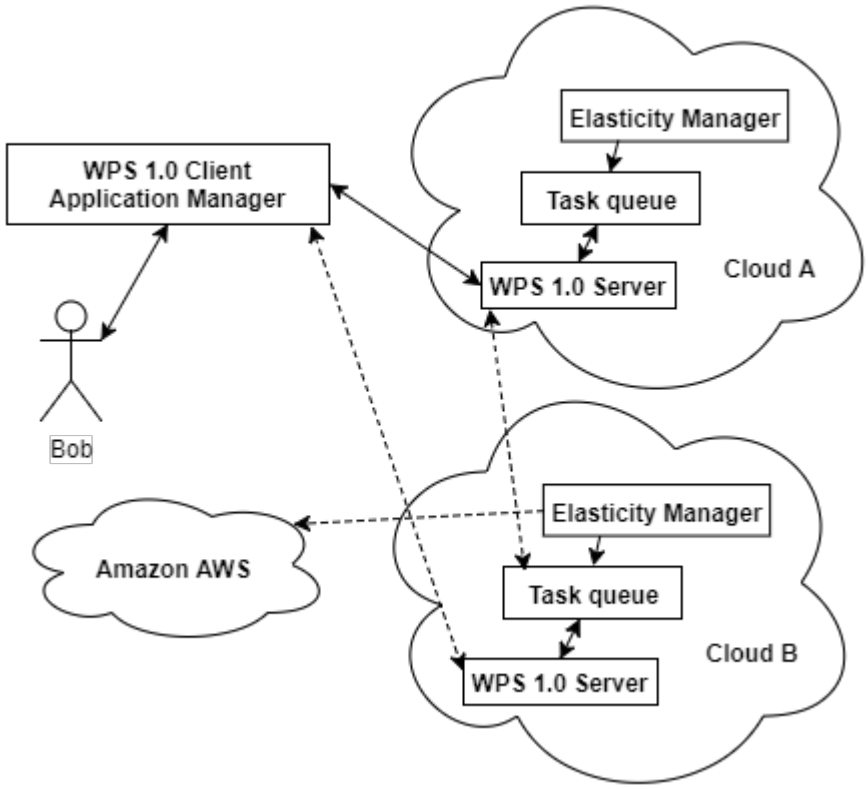


Figure 11. CRIM hybrid cloud concept

While solution presented here is stand alone and is deployable in other clouds, CRIM execution environment leverages the Platform for the Analysis and Visualization of Climate Science (PAVICS - https://pluvier.crim.ca). PAVICS offers a large selection of climate processes, token-based security proxy for WxS, workflow capabilities and load balancing. Docker components are built and deployed to registries using a continuous Integration (CI) framework. Components are assembled as solutions with Docker Compose. For demonstration purposes, the WPS 1.0 Client Application Manager will use the PAVICS web-based platform.v

6.2.3. Docker Containers

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware, containers are more portable and efficient.

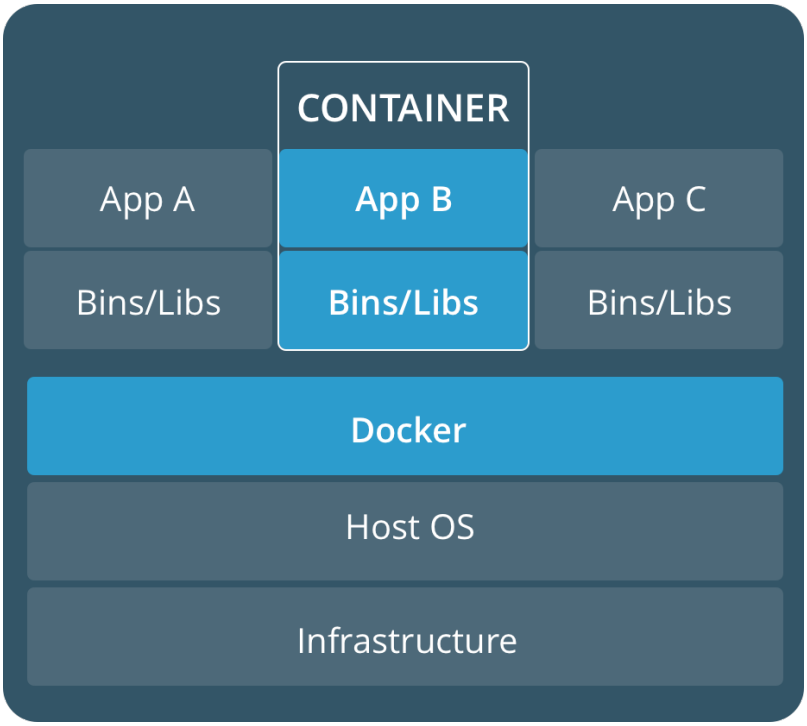


Figure 12. Containers

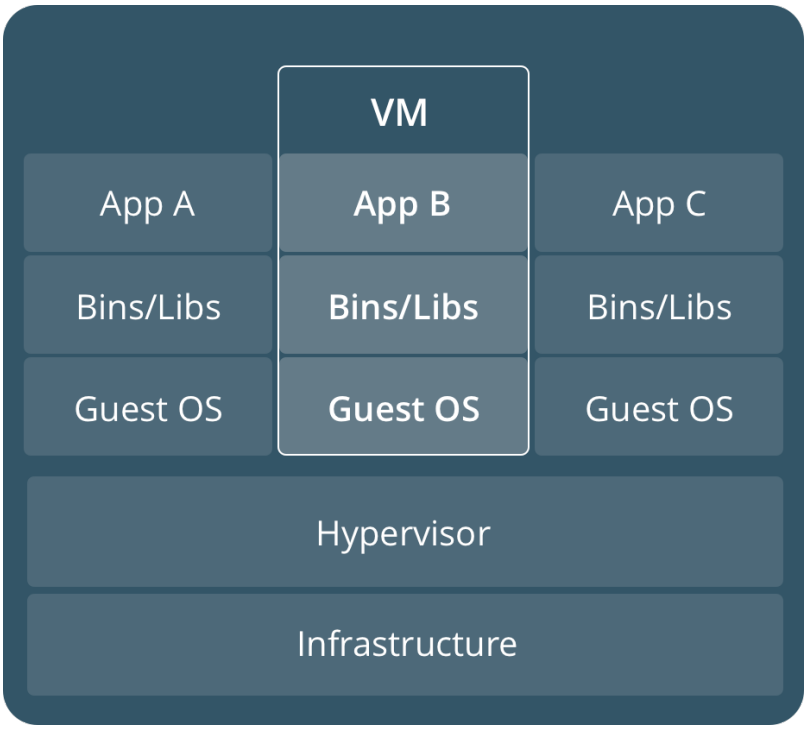


Figure 13. Virtual Machines

Containers (see Figure 12) are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

Virtual machines (see Figure 13) are an abstraction of physical hardware, turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine.

Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. However, by combining containers within a cloud environment, a natural balance of containers and virtual machines can be achieved to scale jobs and processes.

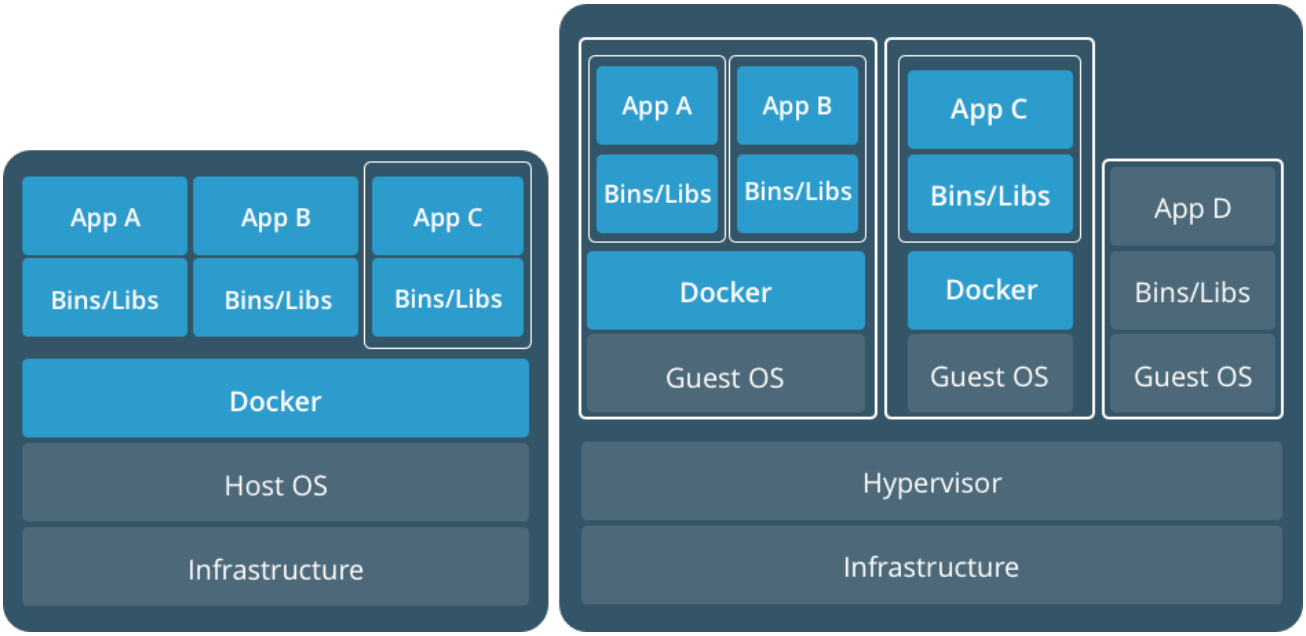


Figure 14. Containers and Virtual Machines Together

This architecture (see Figure 14) achieves a more efficient use of cloud computing infrastructure, but utilizing VMs across a cloud infrastructure while separating processes within Docker containers. There is a natural mapping of number of containers to virtual infrastructure in terms of cpu cores, ram, and storage. While this may vary across different implementations, scalability can be achieved by either deployment of additional VMs, or by deployment of additional containers within a VM.

The Application Package in this document is assimilable to a Docker Image. A Docker Image is an uninstantiated, static configuration, analogous to a VM image in cloud computing, but also a VM snapshot where a previous state was saved. A Docker Image is packaged so that all dependencies are available and configured. Through a command line interface via the Docker Daemon, a Docker image (i.e., application package) can be instantiated as a Docker container. Several containers can be instantiated in a VM from a single Docker image.

6.3. Cloud Orchestration versus Container Orchestration

The introduction of Docker containers for packaging software open the possibility of using Docker Compose to define and run multi-container Docker applications. Compose works in all environments: production, staging, development, testing, as well as continuous integration workflows. With Compose, one can use a YAML file to configure your

application's services. Then, with a single command, create and start all the services from the configuration. Docker Compose does not provision VMs, but once the cloud hosts are provisioned, Docker Compose can be used to "script" the deployment of Docker images to the cloud environment. This is the preferred solution over use of Amazon CloudForms and OpenStack Heat due to the interoperability of deployments using software containers. The scalability based on Docker containers also depends on the method of monitoring, deploying, instantiating, and destroying docker containers.

6.3.1. CRIM Cloud/Container Orchestration

The implementation provided in the testbed can either create or destroy VMs through OpenStack NovaClient version 2.0 API or directly through OpenStack API. The latter is implemented alongside AWS CloudFormation into the OpenStack Heat orchestration program. While OpenStack Heat is mentioned as a desired outcome of the EOC RFQ, CRIM did not use Heat in their solution due to the difficulty in combining VM-level orchestration comprehensively with Docker templating and Celery orchestration. Some features would have become either redundant, difficult to track and maintain or simply leading to race conditions.

Ideally, virtual machines would be created and destroyed on the fly automatically depending on system load. Such capabilities are usually carried out by an elasticity manager that either monitor the state of the task queues and the Cloud environment load. CRIM uses the open source "NEP-143-2 Load Balancing and Access to Automated Annotation Service" available at https://science.canarie.ca/researchsoftware/researchresource/main.html?resourceID=48. The component implements an elasticity manager for dynamic horizontal scaling for services. For instance, if the average time to complete a task or the number of outstanding tasks in the queue goes over predetermined thresholds, the elasticity manager would spawn new virtual machines. In the CRIM cloud implementation, those virtual machines launch workers which immediately acknowledge a number tasks in the queue, thus increasing system throughput. Inversely, when load is very light, extraneous virtual machines are taken down.

A decision was made to keep the implementation as simple as possible in order to easily demonstrate the interactions between virtual machines, Docker containers, task queues and workers. The CRIM solution allows a user to manually increase or decrease the number of virtual machines in the Cloud, up to a determined number of machines as well as keeping a minimum VM count. The user can then see the immediate effect on the task queue through Flower to better understand the impact of VM count and configuration as well as the number of workers in respect to the number of available CPU on the VM. The simplicity of incrementing and decrementing virtual machines do not warrant an orchestrator such as Heat. We would also argue that up to a point, the use of a task queue combined to Docker-packaged application constitutes a form of self-orchestration.

6.3.2. GMU Cloud/Container Orchestration

GMU provides the Docker image as an included file in the VM template so that it is automatically deployed when a VM is provisioned. A request from a WPS client will trigger a Docker Daemon to initiate the corresponding Docker image and deploys the image as a Docker container which contains the process elements which can execute the request. The prototype is multi-task supported, each time user send a new request through WPS, a new container would be created inside the VM. In each VM, multiple containers are dynamically provisioned according to the number of tasks. Once a process is complete, the Docker container is destroyed.

In Testbed 13, we wrap all Docker related operations on the server-end and we deploy the RSTB suite inside the Docker image. All Docker containers that automatically created by the WPS are Docker/RSTB pre-configured, users do not have to handle the installation/configuration of both Docker and RSTB.

6.4. Earth Observation (EO) Data

The EO data accessed by the cloud environments first begin by fetching the data from the external data source and storing it in local shared storage. In general, the deployments of each VM follows the same hierarchy: VM → Operating System → Docker → Sentinel Application Platform (SNAP) → Graph Processing Tool (GPT).

In the GMU environment (see Figure 15), the data is stored in a local shared storage which is mounted to the VM where it can be accessed by multiple Docker containers. A process in the Docker container is responsible for managing the SNAP GPT processes and transferring the resulting data to a VM containing the WMS/WCS components running in MapServer 6.4.1.

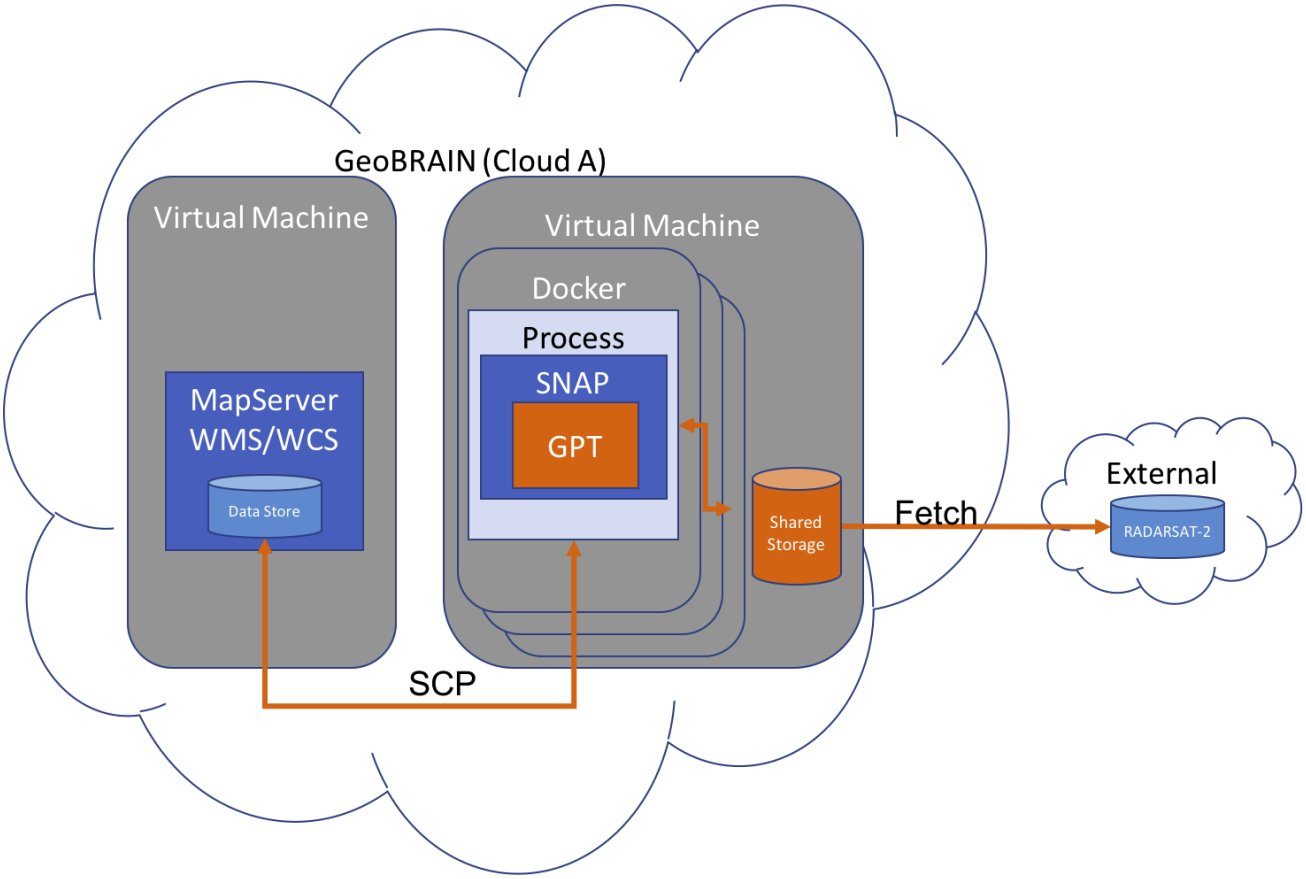


Figure 15. GMU Access to EO Data

In the CRIM environment (see Figure 16), the data is also fetched and stored in local shared storage, however a this data store is shared across all VMs using a fileserver. The shared storage is mounted to each VM using Network File System (NFS) mount, and then accessed by the Docker container. Each VM is configured with only one Docker, which manages the SNAP GPT processes and transfers the result to an external GeoServer 2.10.04 using the Geoserver API.

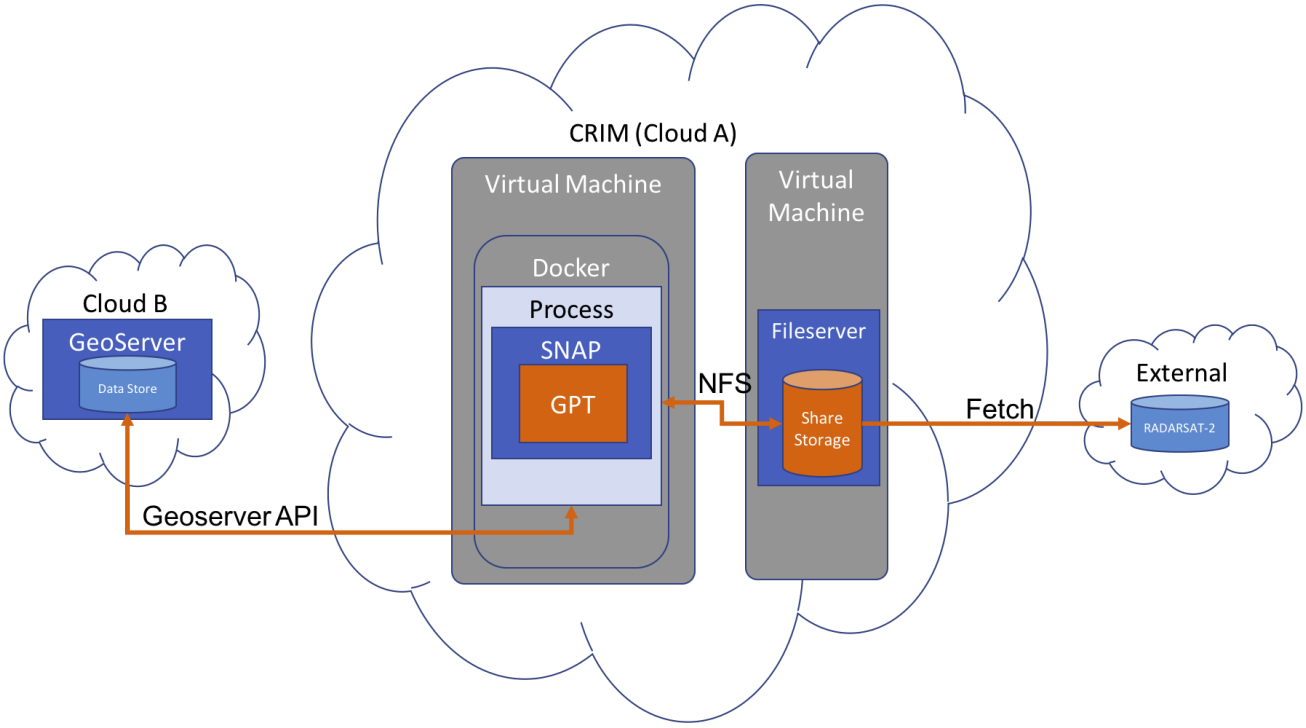


Figure 16. CRIM Access to EO Data

Two Radarsat-2 data packages, rs2_nwt.tar and rs2_vernon.tar, were provided by the sponsor for use in the OGC Testbed 13. Both data packages contained the following: * LI-11525-3 RS2 EULA_Government User_V1-6_01OCT2009_ENGLISH.pdf (license agreement for gov't user) * SNAP_CommandLine_Tutorial.pdf (Tutorial) * Graph_vanzyl.xml (RSTB GRAPH script) * SNAPtest (a folder containing the output in TIF) * Digital Elevation Model (DEM)

rs2_nwt.tar contains:

- RS2_OK77397_PK685920_DK616065_SQ13W_20160711_013038_HH_VV_HV_VH_SLC.zip
- RS2_OK77397_PK685921_DK616066_SQ13W_20160711_013042_HH_VV_HV_VH_SLC.zip
- externalDEM (an external DEM used in the later stage) **rs2_vernon.tar** contains:
- RS2_OK79000_PK698379_DK627315_FQ9W_20160907_013546_HH_VV_HV_VH_SLC.zip
- RS2_OK79000_PK698380_DK627316_FQ9W_20160907_013548_HH_VV_HV_VH_SLC.zip

Radarsat-2 Sample data is also publicly available at http://mdacorporation.com/geospatial/international/satellites/RADARSAT-2/sample-data. The Vancouver dataset offers the end-user the chance to evaluate RADARSAT-2 products from many of the modes available on RADARSAT-2 and for a variety of different applications that include urban mapping, marine surveillance, agricultural mapping, and infrastructure mapping. This

geographic location offers varied terrain from the rugged mountains to the north of Vancouver, to the flat, agricultural lands of the Fraser River Delta.

The Vancouver data set includes:

- Fine (HH/HV) SGF
- Fine Quad-Pol SLC
- Standard (HH/HV) SGF
- Extended High (HH) SGF
- Wide (HH/HV) SGF
- ScanSAR Narrow (VV/VH) SGF
- ScanSAR Wide (HH/HV) SGF

NRCan Grid and NRCan DEM are available via FTP: http://ftp.geogratis.gc.ca/pub/nrcan_rncan/elevation/cdem_mnec/index/cdem_index_250k.kml
http://ftp.geogratis.gc.ca/pub/nrcan_rncan/elevation/cdem_mnec/

6.5. Metrics

CRIM collects metrics on task execution through an Open Source component named Flower, deployed as a Docker container. Flower collects and exposes the statistics and commands of a particular Message Broker. As shown in Figure 17, the Dashboard displays all workers that are subscribed to tasks queues hosted by the broker. Flower tracks every task received and keeps count of their states (active, processed, failed, succeeded, retried) acknowledged by the workers. It also presents the load average on the worker's host, which is a standard Unix-like feature.

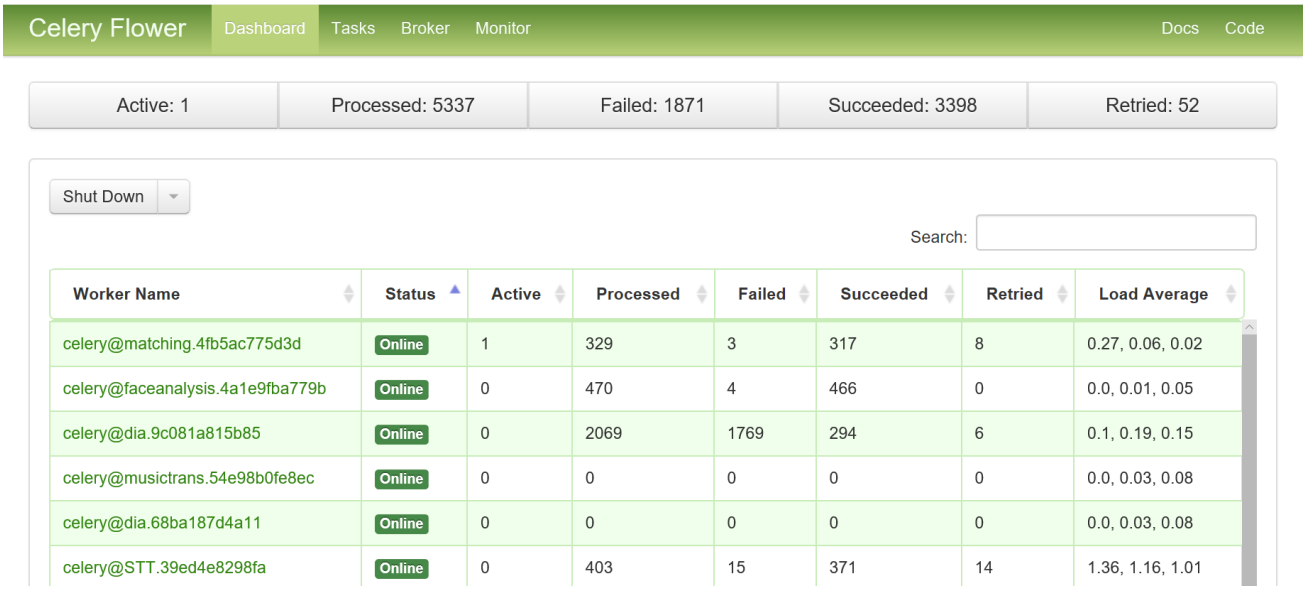


Figure 17. CRIM Dashboard Metrics using Celery Flower on its research infrastructure

Also part of CRIM monitoring tools, Portainer enables real-time control and supervision of Docker containers on a host. Just like Flower, it is deployed as a Docker container and has privileged access to the Docker Daemon running on its host. Through its Dashboard, a user can inspect and modify the execution state (stop, pause, start, restart, etc.) of all containers and connect to their standard outputs. CRIM uses Portainer mostly for the management of static system components (WMS and WPS servers, databases, security components, etc.). Portainer could also be used to debug and monitor execution of transient applications on elastic VMs.

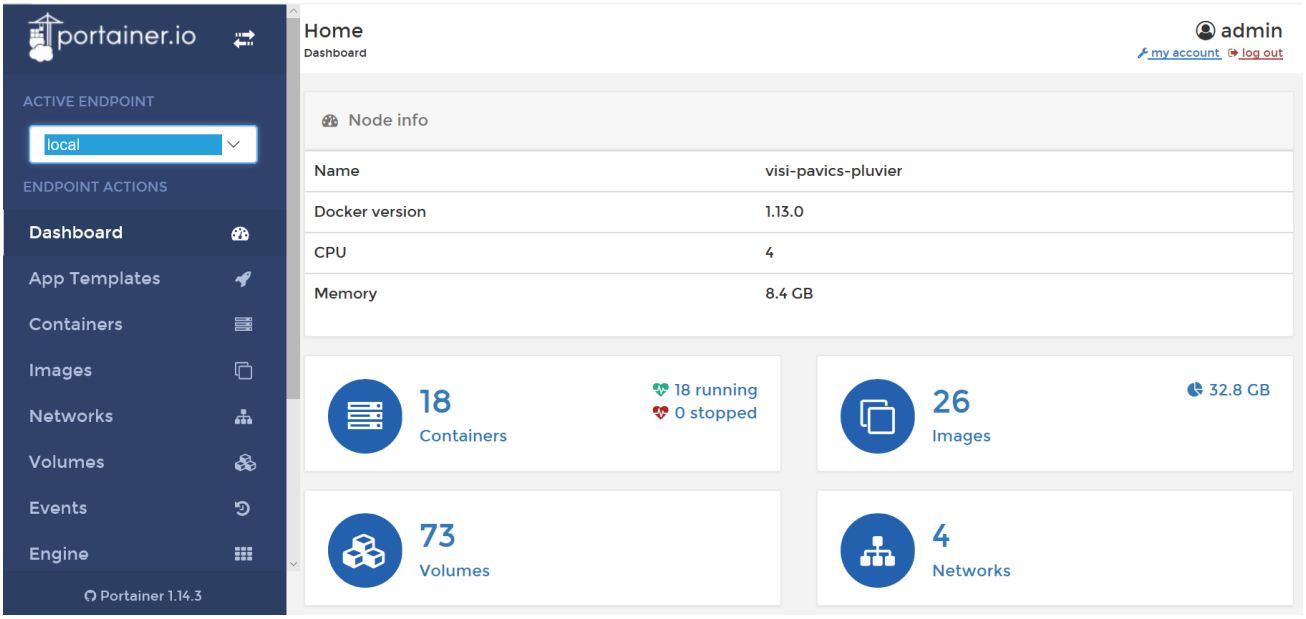


Figure 18. CRIM Dashboard Metrics using Portainer on its research infrastructure

GMU uses tools built into Apache CloudStack for monitoring and capturing cloud usage statistics. Figure 19 shows the metrics captured from the Apache CloudStack dashboard describing the resources, CPU usage, memory usage, network usage, and disk usage of each instance used by GeoBrain Cloud.

Resources				CPU Usage			Mem Usage	Network Usage		Disk Usage			
Name	State	IP Address	Zone	Cores	Total	Used	Allocated	Read	Write	Read	Write	IOPS	Quickview
Testbed13-GMUWPS		192.168.1.155	cloud-zone-1	1	2.0 Ghz	0.02%	1.00 GB	21160 9.74 MB	6546.96 MB	3853794	170704	402498	
Testbed13-Docker-3		192.168.1.238	cloud-zone-1	2	4.0 Ghz	0.02%	2.00 GB	530.80 MB	1706.95 MB	29866	35028	64894	
Testbed13-Docker-2		192.168.1.182	cloud-zone-1	1	2.0 Ghz	0.02%	1.00 GB	492.98 MB	1722.88 MB	28744	36059	64803	
Testbed13-Docker-1		192.168.1.206	cloud-zone-1	1	1.0 Ghz	0.03%	1.00 GB	309.54 MB	3109.87 MB	21838	66347	88185	

Figure 19. GMU Metrics Collection

6.6. Configuration Comparison

Software Component	GMU	CRIM
WPS	Custom (WPS 2.0 Spec)	PyWPS (WPS 1.0 Spec)
Cloud Computing Framework	Apache CloudStack	OpenStack
Operating System	Ubuntu	Ubuntu CentOS
Docker Container Modules	Docker Daemon	Docker Daemon Docker Compose Docker Registry
Cloud VM Provisioning	Manual Provisioning	Dynamic Provisioning
Container Provisioning	Dynamic - Multiple Containers per VM	Dynamic - 1 Docker Container per VM
Message (Job) Broker	Custom GMU S/W	RabbitMQ
Task Manager	Custom GMU S/W	Celery
Metrics Monitor	Cloudstack API/Docker API	Portainer/Flower
RSTB Solution	SNAP - GPT	SNAP - GPT
WMS/WCS	MapServer	GeoServer
Data Result Transfer	SCP	Geoserver API

Chapter 7. Execution

7.1. WPS Parameterization

There are three sets of parameters that are input into the WPS:

- **Process Parameters** - GPT parameters related to Earth Observation Data
- **Cloud Parameters** - Virtual Machine configuration (e.g., OpenStack Flavor or Amazon Instance Types such as tiny, small, medium, large, etc.)
- **Docker Parameters** - Docker registry URL, Image name, and Version

7.2. CRIM WPS Process Parameters

The CRIM WPS client uses "by reference" parameters which maps Cloud resources and their ACL. Parameters are set by the worker into the Docker container environment variables. These parameters are in turn mapped to the defined executable entry point of the Docker container, in our case a script called ogc_processing.py. Below is a list of potential input parameters to describe the process.

Input:

- Radarsat-2 source file
- Polarimetric-Speckle-Filter:
- type of filter (Refined Lee, etc.)
- parameters: window size, etc.

Geocoding parameters:

- DEM, interpolation method, mapProjection, pixelSpacingInMeter, etc.

Output:

- format (GeoTiff, etc.)
- bands (6 maximum) or RGB composite

A detailed description for CRIM polarimetric SAR processing of Radarsat-2 Images can be found in [AppendixF].

Parameters can be defined dynamically in the Graph's xml file ($file, $target, etc.). For more information, see doc on GPT (http://corp.array.ca/nest-web/help/general/

commandLineReference.html). An example use for batch processing can be found in the following excerpt. In that scenario, input parameters received at application-level (Docker) can be mapped directly to GPT by command-line. Additional examples of GPT Process files can be found in Appendix E - WPS Process Descriptions.

7.3. GMU WPS Process Parameters

GMUWPS wraps the GPTGraphProcess as the process for calling GPT within the graph XML file on earth observation data in WPS 2.0 standard interface. The input parameters of the GPTGraphProcess are InputData and GraphXML. The output parameter parameters of the GPTGraphProcess is Result.

Input:

- InputData: URL of the input data
- GraphXML: URL of the graph xml file

Output:

- Result: URL of the final product in WMS/WFS format

7.3.1. Cloud Parameters

CRIM processes include a cloud parameter which directs the WPS server to publish the job process to a task queue located in a particular cloud. The resulting published job process is handled by the recipient task queue which can provision additional VMs as needed containing workers which can process the job request. As presented in Appendix E, CRIM proposes an input to its WPS identified as **IaaS_deploy_execute** where the message broker URL and the task queue name are provided. In this configuration, there is a 1:1 mapping between task queues names and supported VM types; in other words, a user can explicitely request that the application is deployed and run on medium or large VMs. Another identifier named **IaaS_datastore** specifies the base URL of an externalized fileserver where to store outputs. Both **IaaS_deploy_execute** and **IaaS_datastore**, as well as **WMS_server** in the Process params section, are excellent candidates for an OWSContext file instead of being inlined in a process description.

GMU WPS can set a VM priority which determines how much resources to apply to a particular job. This parameter can be set to High, Normal, or Low. GMU WPS provides a unique process named **SetPriorityProcess**. It allows users to specify the priority of the next placed job. The priority mechanism in GMU WPS includes three major levels: **high**, **normal**, and **low**. The three levels correspond to different instance VMs with relation to resource size specifications and availability of those resources. The jobs with higher priority will have greater chance to use the more powerful instance VMs, but it depends on the real-time

usage of resources. For example, although instance VM (A) has a very high resource specification, in circumstances of high priority job may be avoided if utilization is high. In that situation, a more idle instance will be used instead. The relationship between job priority and instance VM capability is nonlinear.

SetPriorityProcess example request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute   version="2.0.0" service="WPS"
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" mode="sync">
      <ows:Identifier>SetPriorityProcess</ows:Identifier>
      <wps:Input id="priorityLevel">
         <wps:Data>HIGH</wps:Data>
      </wps:Input>
      <wps:Output id="Result"
wps:dataTransmissionMode="reference"></wps:Output>
</wps:Execute>
```

SetPriorityProcess example response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Result xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.opengis.net/wps/2.0"
xmlns:ns3="http://www.opengis.net/ows/2.0">
   <ns4:JobID>e021a692-c10d-4ba6-9b3a-62367e6623cd</ns4:JobID>
   <ns4:ExpirationDate>2017-09-30T16:36:07.976-04:00</ns4:ExpirationDate>
   <ns4:Output id="Result">
       <ns4:Data>done</ns4:Data>
   </ns4:Output>
</ns4:Result>
```

7.3.2. Docker Parameters

The CRIM processes contain an input parameter named **docker_image** for the Docker URL. This is the location from which a worker retrieves the image and deploys as a Docker container for processing the job. The URI contains the full path to a Docker image as used by Docker Daemon, including the host, port, path, image name, and version. This input parameter does not support credentials. Credentials for private Docker registries are set as a system configuration. The credentials are injected in the environment variables of the VM instance that runs the Docker Image. the **docker_image** parameter would be better suited in an OWSContext file instead of inline in the process description.

The GMU process contains a Docker image name which is already stored inside a VM upon provisioning of the VM image. Additional Docker image snapshots can be deployed as needed per the parameter for each job.

7.4. Deployment and Management Steps (Provisioning)

7.4.1. CRIM Deployment and Execution

In order to register an application to the system, a developer needs to package all the dependencies and the entry point process in a Docker image. Once tested locally, the Docker image is pushed to a registry, either DockerHub or DockerRegistry. The developer also must provide a corresponding process description in the WPS 1.0 server with the appropriate job scheduler enabled. Once the new service is registered in the WPS Server, the developer can register the server itself, a WPS Provider hosted in a Cloud, into an Application Management Client which simplifies the browsing of providers and process descriptions, as well as the execution and monitoring of processes. In order to do so, the App Management Client requests the available cloud configurations for each provider. Figure 20 below contains a sequence diagram to describe the process flow. (Note: The names Bob and Marco are extensions from examples provided in the Testbed-13 RFQ.)

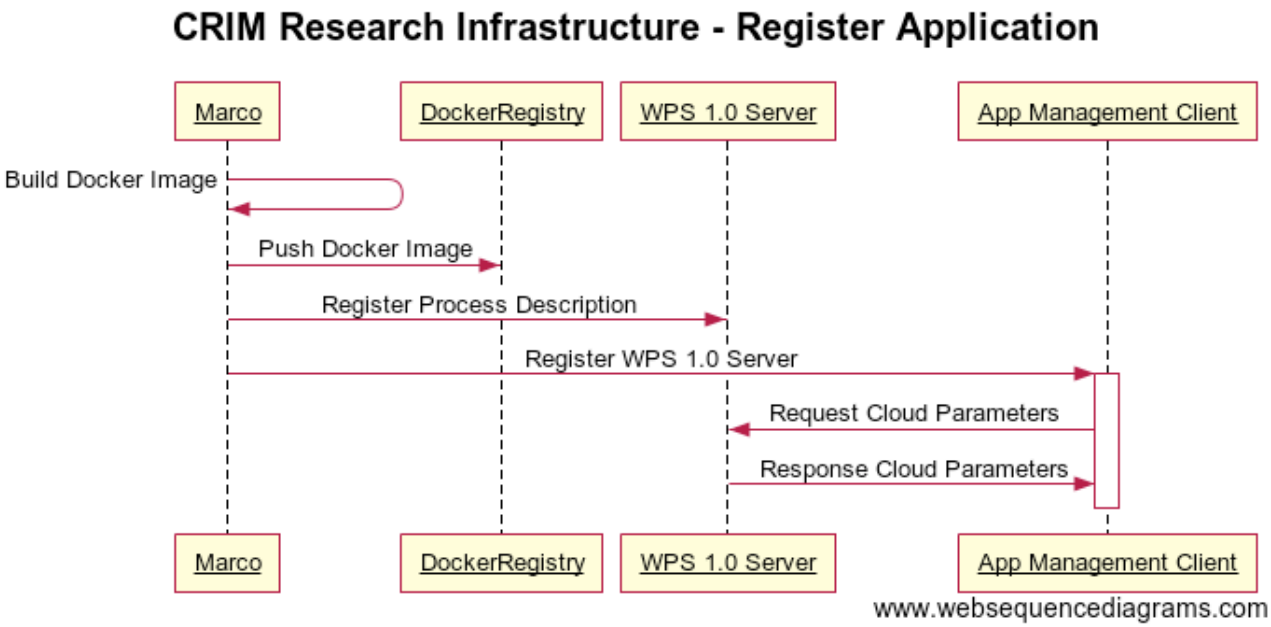


Figure 20. CRIM Sequence Flow - Register Application

The CRIM WPS interface relies on a publish-subscribe (PubSub) pattern and its underlying message queue. Upon receiving a request from a WPS client, the WPS server's job scheduler publishes a message to the queue using a PyWPS extension. The message consists of an identifier of the process and the parameters required to complete the execution. These parameters include parameters for the process, identification of the cloud for which to publish the task, and a reference to the Docker image location. A RabbitMQ broker

implemented using the Advanced Message Queuing Protocol (AMQP 1.0) open standard receives messages from publishers and routes them to subscribers. While awaiting to be acknowledged by the associated subscribers, messages are accumulated in queues. In this context, the task is the unit of work and it contains all required parameters. The subscribers, called "workers" in Celery, reside as daemon processes running in a VM. The "workers" monitor the queue on a continuous basis for tasks to process. Once a worker acknowledges a task, it pulls the Docker application package from a Docker registry and launches it as a Docker container. For simplicity, CRIM chose a 1:1 cardinality between Cloud and Docker Containers. While a container can also be suspended, it was only considered in the current work that its state is always running. It is the responsibility of the application to download the input data from the cloud, to produce the outputs and to provide status. Results are sent back to the WPS server by the worker through the task queue. Figure 21 below contains a sequence diagram to describe the process flow.

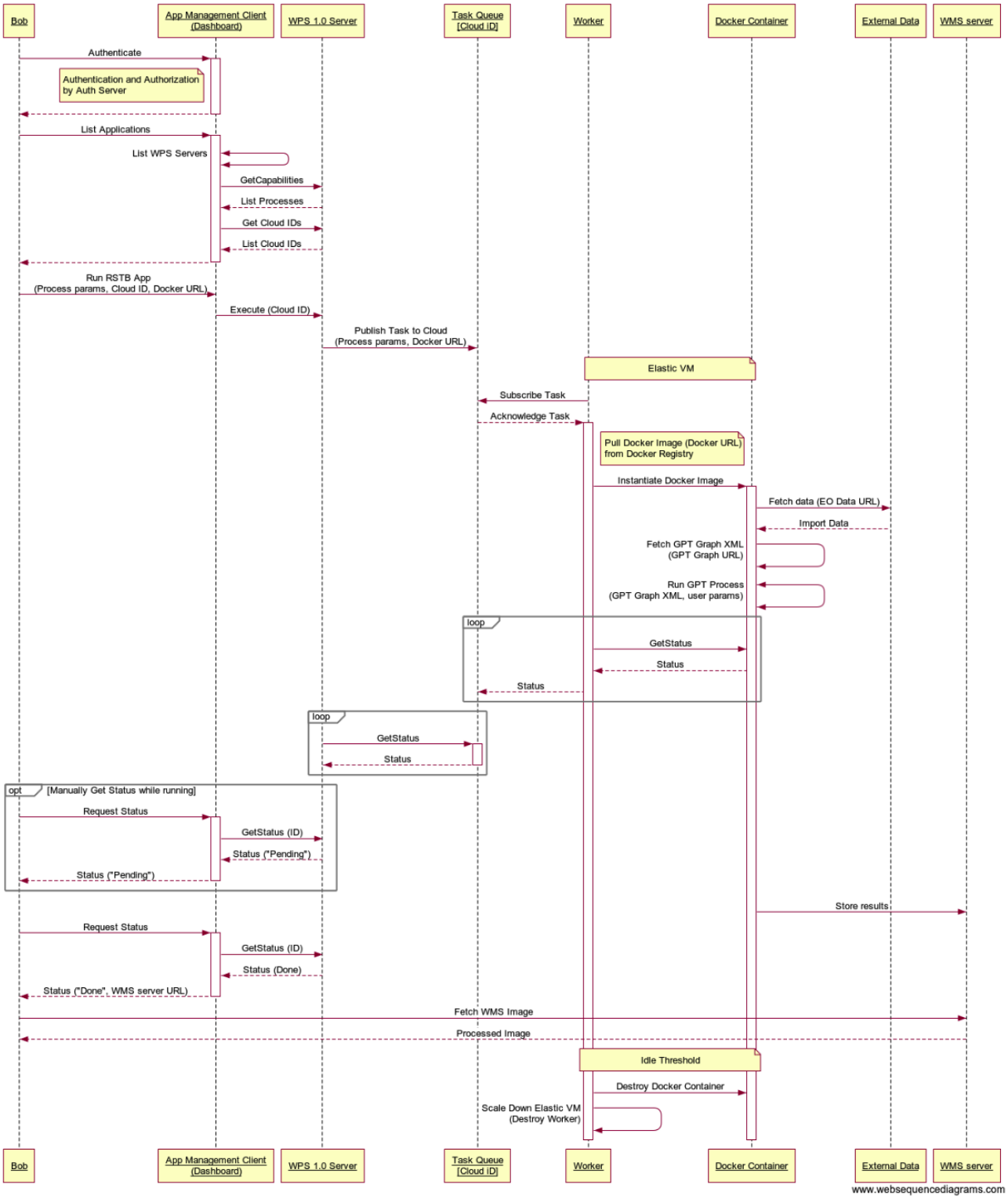


Figure 21. CRIM Deployment Sequence Diagram

7.4.2. GMU Deployment and Execution

GMU WPS can process a number of images at one request. The processing may be conducted over different instance VMs. In the specified VM, a docker container will be started and the internal RSTB command will be invoked to process the inputted data. In the process, multiple instance VMs are used. The processing is completely independent and parallel. The overall time cost will be much less than processing in sequence. Our solution shows promise to fully leverage most or all of the resources in a hybrid or private cloud

environment and let them work together efficiently on one big task.

GMUWPS provides five basic operations of WPS 2.0: GetCapabilities, DescribedProcess, Execute, GetStatus, and GetResult. The workflow of GPT graph process is described in Figure 22. In a GPT processing task, users get the list of available processes by sending GetCapabilities request. To start a GPT graph task, we have to execute the GPTGraphProcess process with required parameters (url of external EO data, url of GPT graph xml file). After the Executer request has been received, a new Docker container is created and the GPT process is performing inside the container. During the processing, the status can be checked by sending GetStatus request, either the status of "running" or "done" is returned. Once the task is done, the result is copied from the Docker container to the WCS server then the container is destroyed. Finally, user can get the WCS url of the product by sending the GetResult with process ID.

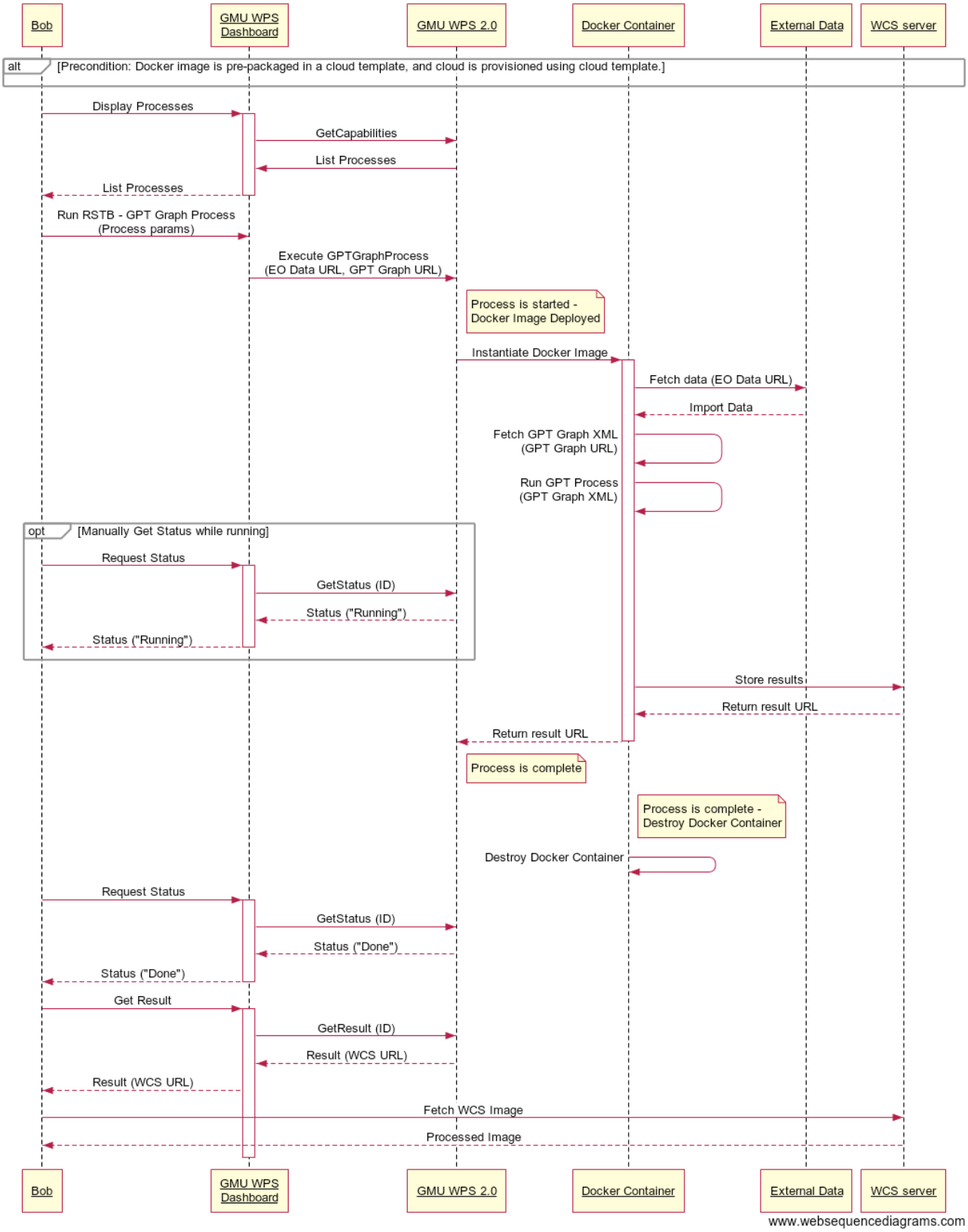


Figure 22. GMU Deployment Sequence Diagram

7.5. Result (WMS/WCS)

7.5.1. GMU WCS Result

The result of the GMU WPS is a WCS GetCoverage URL, instead of a direct file URL. WCS provides much more versatility in manipulating the results such as rendering, clipping, rotating, and re-projecting. The following is an example WCS return of GMU WPS:

Chen - Please update the link and the image to reflect an RSTB image.

```
http://cloud.csiss.gmu.edu/182/mapserv?map=/var/www/html/mapfile/sample.tif.map&SERVICE=WCS&REQUEST=GetCoverage&VERSION=1.0.0&Coverage=sample.tif&FORMAT=image/jpg&CRS=EPSG:4326&BBOX=0,0,7012,6108&WIDTH=512&HEIGHT=466
```

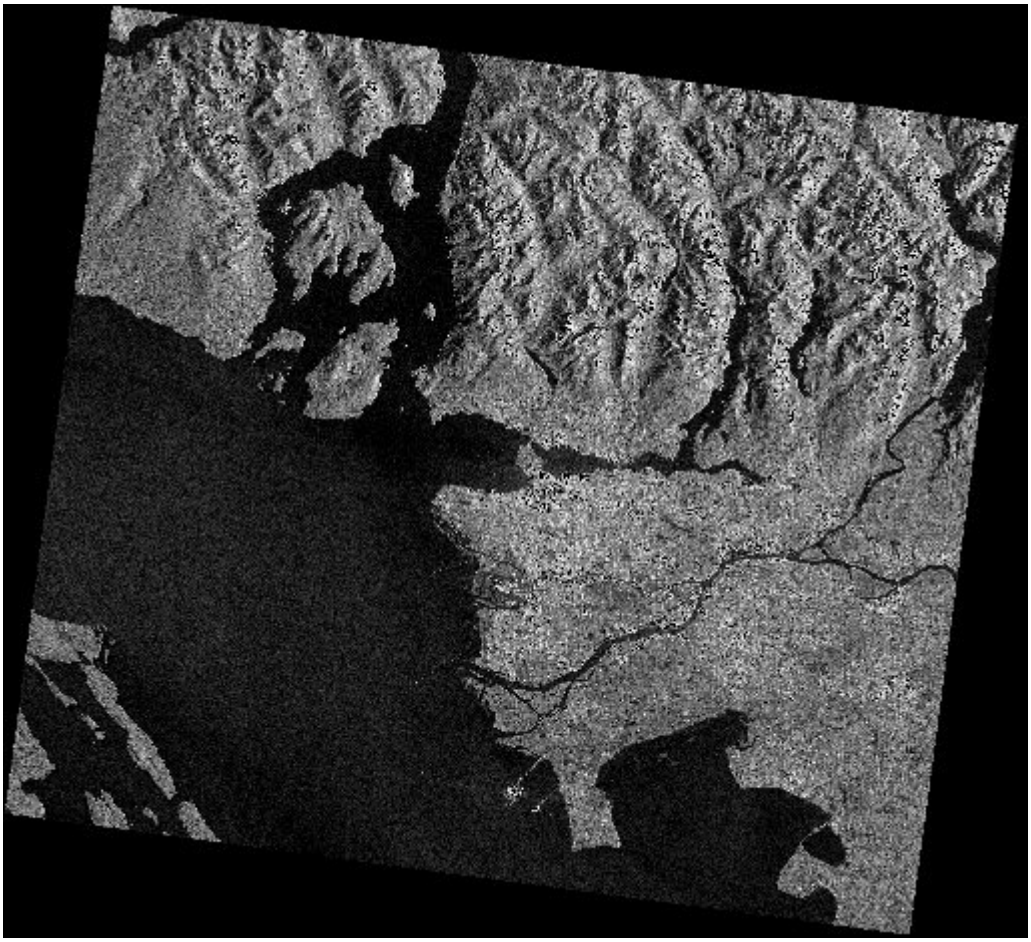


Figure 23. GMU WCS Result

7.5.2. CRIM RGB WMS Result

It is not possible to use the raw CP decompositions for WMS outputs. An RGB composite, ideally highlighting vegetation with respect to urban structures, is required. Below in Figure 24 is a WMS output (RGB composite, CP orthorectified) produced for one of the provided images, hosted on a GeoServer at CRIM (user:OGC01, passwd:2Wsx3edc)

```
http://132.217.140.40:8080/geoserver/OGC01/wms?service=WMS&version=1.1.0&request=GetMap&layers=OGC01:RS2_OK79000_PK698379_DK627315_FQ9W_20160907_013546_HH_VV_HV_VH_SLC-2017-08-02T20:30:46.818016Z&styles=&bbox=-118.93031183853228,49.834373782054,-118.15157741042583,50.228387801408914&width=768&height=388&srs=EPSG:4326&format=application/openlayers&BGCOLOR=0x000000
```

```
http://132.217.140.40:8080/geoserver/OGC01/wms?service=WMS&version=1.3.0&request=GetCapabilities
```

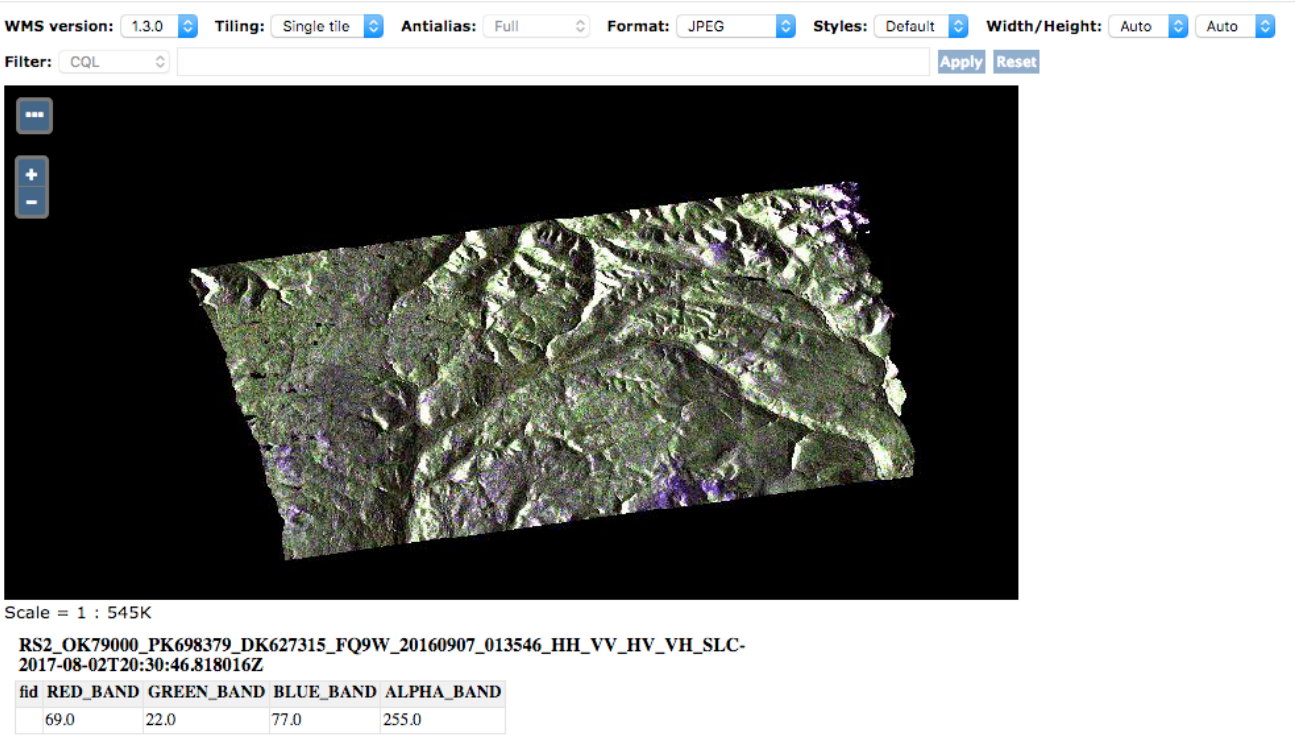


Figure 24. CRIM Terrain-corrected RGB Image as a WMS Result

7.5.3. WMS/WCS Discoverability

At the moment, discoverability of the WMS/WCS results by the user was not explicitly addressed. The result is tied to the initial query in two ways: the URL is logged in the task queue and it is logged as an output of the WPS server. In this scenario, it is the responsibility of the client to fetch the result. To improve discoverability of the product, advanced implementations could:

- Include a reference to the request/task ID at the GetCapabilities level of the WMS/WCS server (For instance, with use of Keywords)
- Include harvesting of the WMS/WCS output and appropriate metadata (at the "stage out" phase) in an OGC Catalogue Service

- Automatically mount the WMS/WCS outputs in the visualization panel of the Application Management Client

Chapter 8. Security (Authorization/Authentication)

Generally, cloud infrastructure requires tight access control for most requests a user submits. For example, access should be restricted to compute resources such as a WPS services and to data resources including geospatial data served with WCS/WMS/WFS servers, HTTPS file download, subsetting of Network Common Data Form (NetCDF) files via Open-source Project for a Network Data Access Protocol (OPeNDAP). An Access Control List (ACL) is usually attached to resources.

8.1. NRCan Distributed Access Control System (DACS) Single Sign-On Implementation

The following section describes the preferred methodology for the sponsor, Natural Resources Canada (NRCan), for a Single Sign-On solution using the Distributed Access Control System (DACS). It was primarily written with integration of applications into Canada's National Forest Information System (NFIS) managed by Natural Resources Canada. Detailed implementations will not be discussed nor will other aspects of the Single Sign-On system (such as LDAP and the user interface).

The Single Sign-On system implemented on the NFIS relies on a key piece of software called the DACS, an open source software located at https://dacs.dss.ca. DACS is a module created for the Apache web server. It can be turned "on" for specific hosts and URLs in the web server. Once DACS is turned "on", it will deny everyone from that host or URL until configured otherwise. DACS has been tested on several OGC testbeds and is used to control access to OGC Web services on NFIS. In order for an application to make use of DACS as part of a Single Sign-On solution, the application must be protected behind one of the URLs or Hosts for which DACS is turned "on". Having done this, there are four options for further implementation of the application, as described in this document.

8.1.1. Option 1: Rely Solely on DACS filtering through DACS configuration

When DACS is turned "on" for a URL or host, it can be configured wto limit to certain users or groups based on a user's identity, IP address, user agent, etc. Additionally, it can be configured to depend on which query string parameters are passed into the URL.

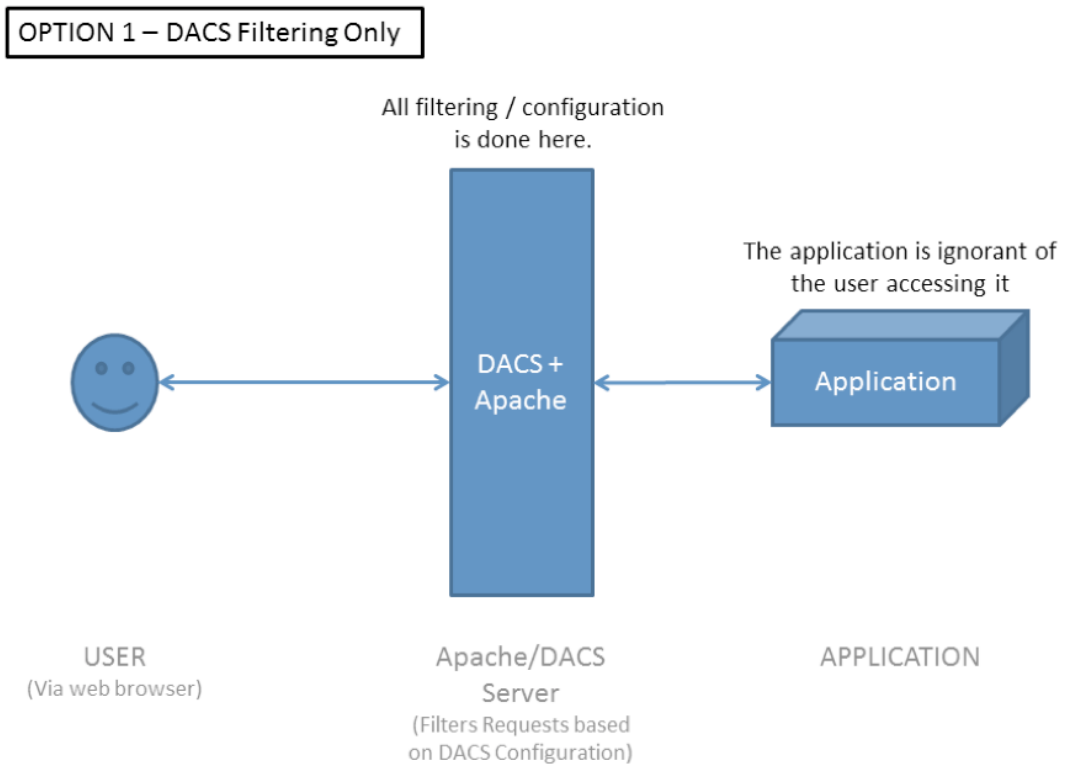


Figure 25. Option 1: DACS filtering Only

Pros	Cons
* No configuration required in the application	* The application cannot know anything about the user (username, jurisdiction, etc). To the application, the user might as well be anonymous.
* This is how the popular "CA Siteminder" SSO solution works. It may allow siteminder-compatible applications to be easily configured to use a DACS request header.	

Below is an example of DACS configuration that implements such a rule:

```
<acl_rule status="enabled">
  <services>
      <service url_pattern="/application"/>
  </services>
  <rule order="allow,deny" pass_credentials="all">
  <allow>
      (${Args::PAGE:i} eq "admin" and
       (from("192.168.0.0/24") or
        user("NRCAN:myusername")) or
 ${Args::PAGE:i} eq "user"
  </allow>
</acl_rule>
```

- It allows everyone access to url "/application?page=user"
- It denies everyone access to url "/application?page=admin" except:
    - Those with an IP address in the 192.168.0.0/24 range
    - A user in jurisdiction "NRCAN" with username "myusername"
- It denies everyone access that does not specify ?page=user or ?page=admin

While these rules can be either simple or complicated, they often need only be written once for each application. Individual users can be added instead to "groups" through DACS or LDAP. This would allow an administrator to add or remove a user from a "group" without affecting a DACS rule.

For more information, see:

- https://dacs.dss.ca/man/dacs.acls.5.html
- https://dacs.dss.ca/man/dacs.groups.5.html

8.1.2. Option 2: Use of DACS Environment Variables

This option builds on "Option 1". However, in addition to possibly configuring access through DACS configuration, an underlying application can also make use of environment variables that DACS sets when a user is authenticated. This requires that the application is on the same server as the Apache/DACS installation in order to see the environment variables. The application would only be called (and thus, only see these variables) if DACS is configured to allow the user to access the application in the first place (i.e., Option 1). Also, should DACS be configured to allow unauthenticated users to access the application, the environment variables would not be set since the user is unauthenticated.

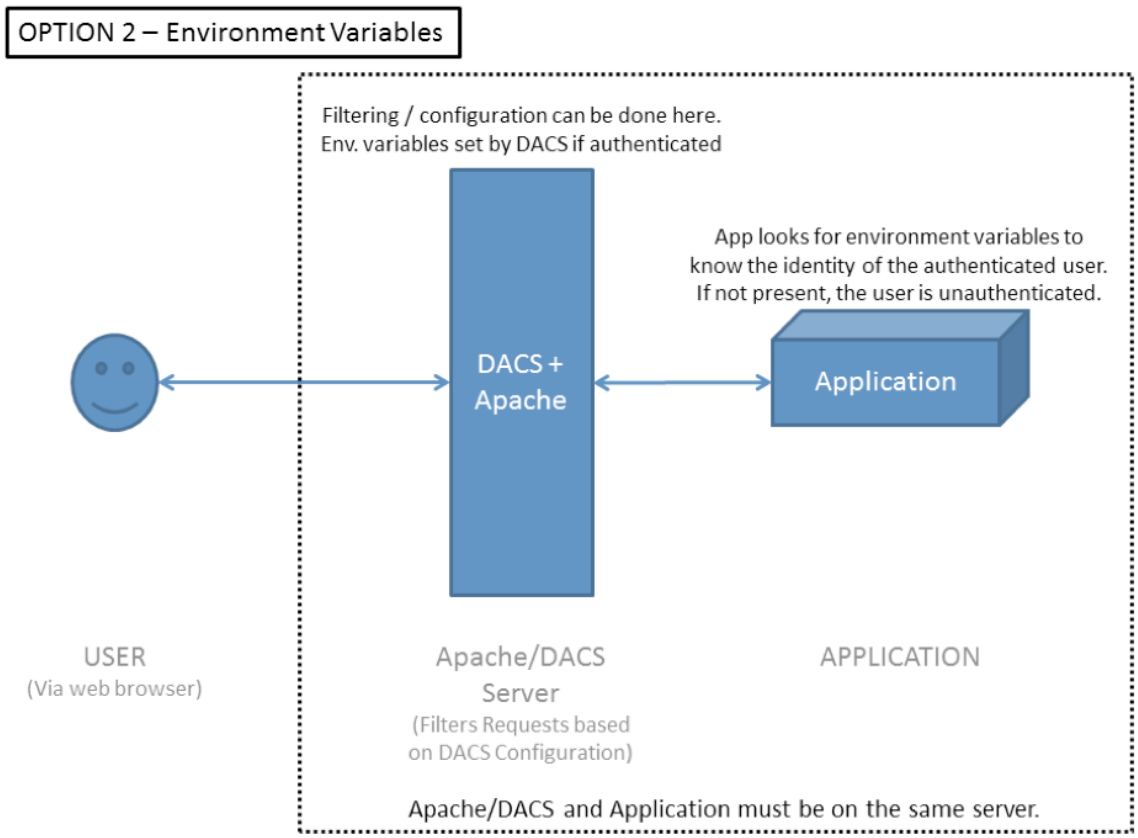


Figure 26. Option 2: Environmental Variables

Pros	Cons
* Environment variables can be accessed by practically any programming languagen	* The application must be on the same server as Apache/DACS in order to access the environment variables.

There are many environment variables exported by DACS when an authenticated user makes a request. The full list can be found at https://dacs.dss.ca/man/dacs_acs.8.html#exported_envars

These are two relevant environment variables:

- DACS_IDENTITY – the full federation, jurisdiction, and username (e.g., NFIS:NRCAN:myusername)
- DACS_USERNAME – only the "username" component (e.g., myusername)

For example, in PHP, these variables can be accessed via the "$_SERVER" global variable like in the following code snippet:

```
$username = $_SERVER['DACS_USERNAME '];
if ($username) {
  // user is authenticated, print the username
  print($username);
  // lookup application-specific permissions for the user here
} else {
  print("The user is unauthenticated");
}
```

Other programming languages would have equivalent methods of accessing such variables.

8.1.3. Option 3: Setting Request Headers

This option also builds on "Option 1". However, in addition to possibly configuring filtering through DACS configuration, downstream applications can be configured to watch for a "request header" to indicate if the user is authenticated, and if so, what their username is. This requires a little extra configuration in Apache/DACS to add the request header to the forwarded request for all downstream applications. This is configured using the "mod_headers" module in Apache (see http://httpd.apache.org/docs/current/mod/mod_headers.html).

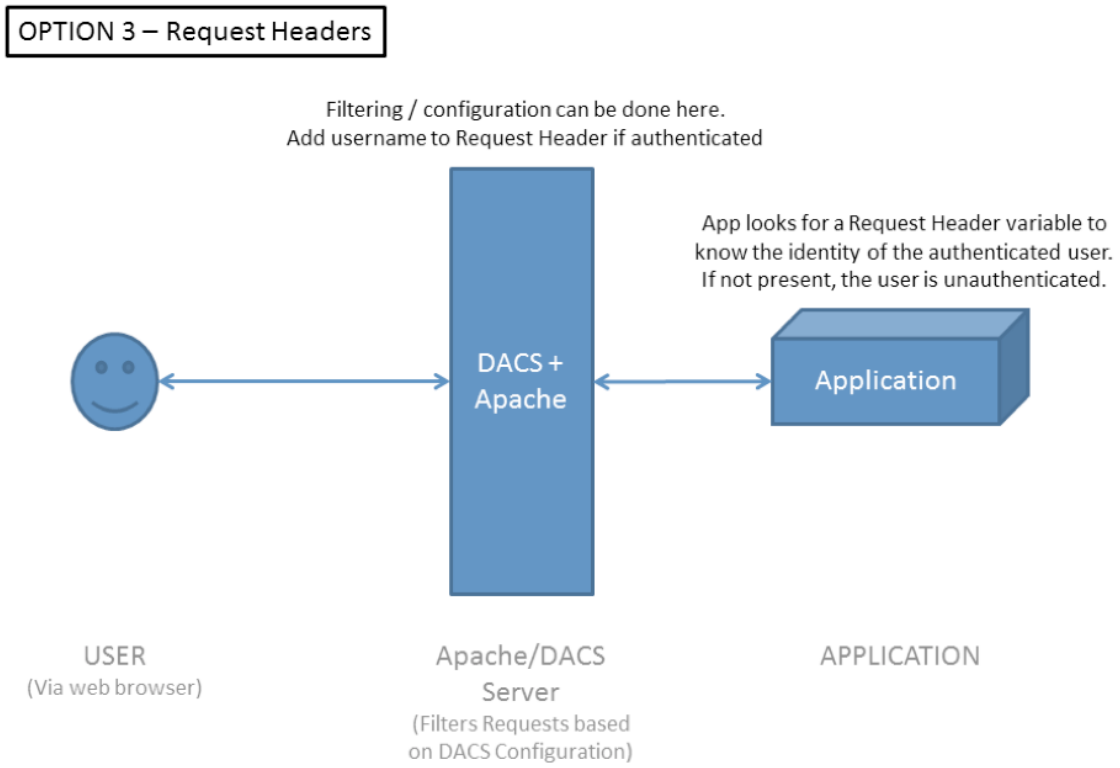


Figure 27. Option 3: Request Headers

Pros	Cons
* Request Headers can be set and processed by any application downstream of (behind) the Apache/DACS server without needing to be on the same server.	* Request Headers might be removed by firewalls, other web servers, and software if they are implemented between Apache/DACS and the application.
* Will work with any web-based application	
* Configuration is done in a single place	

This allows downstream applications to look for the request header named “DACS-USERNAME”. For example, implementing the following pseudo-code in a downstream application would be a start: This allows downstream applications to look for the request header named “DACS-USERNAME”. For example, implementing the following pseudo-code in a downstream application would be a start:

```
methodReceivingRequest( request ) {
  var userName = request.getHeaders(“DACS-USERNAME”);

  if (userName == null) {
     print( “User is unauthenticated” );
  } else {
     // user is authenticated, print the username
     print( “Username is $userName” );
     // lookup application-specific permissions for the user here
  }
}
```

However, some applications may not require any code changes. In fact, there are a number of software packages that implement handling of request headers for Single Sign-On authentication. This has been done to integrate with the popular commercial Single Sign-On solution (“CA Siteminder”) – which uses request headers. Those applications that have developed request header handling for CA Siteminder can likely be configured to observe the DACS request header.

An example is the open source Spring Security framework. This is the framework used by the GeoNetwork catalogue that FGP has implemented. Spring Security has a built-in class called **RequestHeaderAuthenticationFilter** which can be implemented in GeoNetwork using configuration like the following:

```
<bean id="httpPreAuthFilter" class=
"org.springframework.security.web.authentication.preauth.RequestHeaderAuthent
icationFilter">
 <property name="principalRequestHeader" value="DACS-USERNAME"/>
 <property name="exceptionIfHeaderMissing" value="false"/>
 <property name="authenticationManager" ref="authenticationManager"/>
 <property name="checkForPrincipalChanges" value="true" />
</bean>
```

This injects the RequestHeaderAuthenticationFilter into the "pre-auth" stage of Spring Security. This is a brief description of the configuration:

- **principalRequestHeader**: sets the name of the request header to observe (DACS-USERNAME)
- **exceptionIfHeaderMissing**: whether to raise an exception if it is missing (which would mean the user is unauthenticated. Probably "false" if having unauthenticated users view the application is not an error condition.
- **authenticationManager**: the authentication manager that is managing the full authentication lifecycle
- **checkForPrincipalChanges**: whether the request header should be observed on every request to see if the user has logged-out

Additional configuration can be implemented to perform a lookup of user permissions in a database or LDAP, for example. While full configuration of Spring Security is out of scope of this document, this snippet illustrates that, in some cases, code is not required to implement request header authentication.

8.1.4. Option 4: Verify cookies through DACS web services

This is the only option for which being behind a URL or host which has DACS turned "on" is not an absolute necessity (though it is still recommended). It is also the only option able to be directly used by client-side software (e.g., Javascript). Whenever a user is authenticated via DACS, they are provided with an encrypted cookie that can be used to verify their identity. Any time that a user contacts a URL or host for which DACS is turned "on", DACS automatically uses this cookie to verify their identity and allow or deny access.

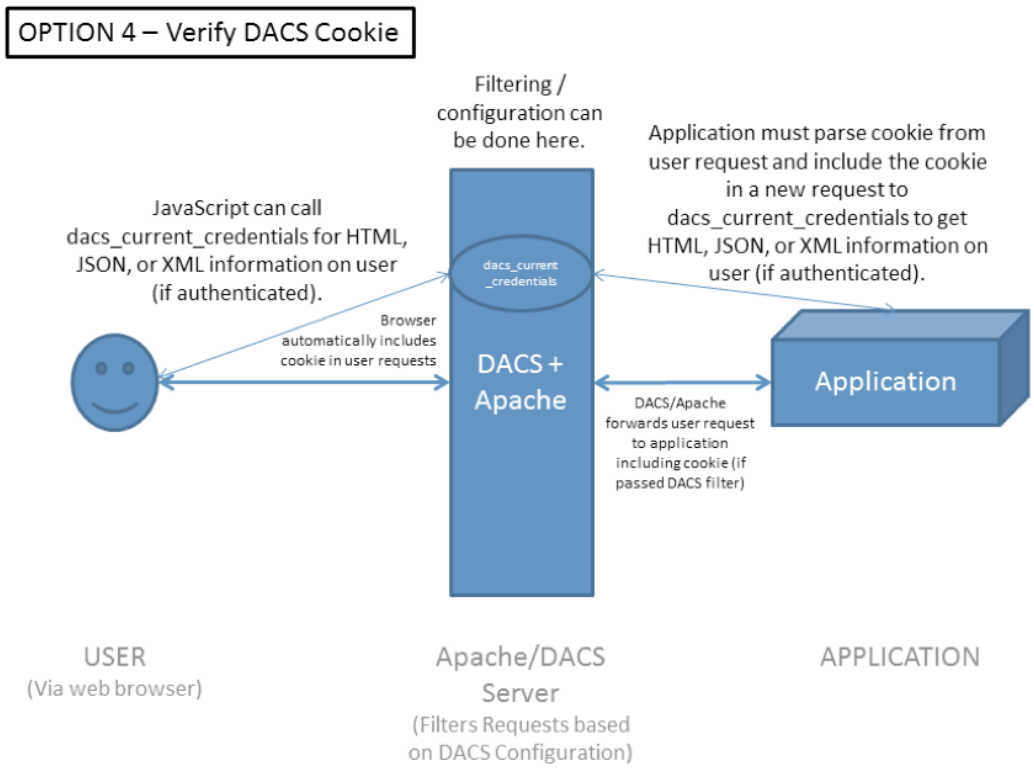


Figure 28. Option 4: Verify DACS Cookie

However, an application can also use DACS web services to fetch information on the user. The primary DACS web service to fetch information on a user is called "dacs_current_credentials". In order to invoke this web service, the procedure is different depending on if the application is a server-side application or a client-side application:

Client Side Applications

For client-side applications, the browser will generally pass the DACS cookie automatically with every request, depending on the Same Origin Policy (see the Terms and Definitions). This can be done by programming JavaScript to make an AJAX request to "dacs_current_credentials" to retrieve information on the user. For example, using the jQuery framework, if the user is authenticated, the following will pop-up an alert box with the user's username:

```
(function($) {
  $.ajax({
    url: 'https://host.example.com/cgi-
bin/dacs/dacs_current_credentials?FORMAT=JSON'
  })
  .done(function(data) {
    alert(data.dacs_current_credentials.credentials[0].name);
    // can call another web service here to get
    // application-specific user permissions
  });
})(jQuery);
```

Server Side Applications

For server-side applications, the process is a little more complex. The application first needs to parse the cookie from the incoming user request. Once it has the cookie, it needs to create a new request to "dacs_current_credentials" with the user's cookie as part of the request. For example, when making a request to "dacs_current_credentials", the request should have a "COOKIE" header like so:

```
Cookie: DACS:NFIS::NRCAN:myusername={{ ENCRYPTED COOKIE CONTENTS }}
```

There is too much variation in how a server-side application could be coded to demonstrate a specific example here. However, in very simple pseudo-code:

```
methodReceivingRequest( request ) {
  var dacsCookie = request.getCookies().getFirst();
  var newRequest = new Request( "https://host.example.com/cgi-
bin/dacs/dacs_current_credentials?FORMAT=JSON");
  newRequest.addCookie( dacsCookie );
  var response = newRequest.connect();
  var json = JsonUtil.parseJson( response.getBody() );
  print( $json['dacs_current_credentials']['credentials'][0]['name'] );
  // can obtain application-specific user permissions here
}
```

For both client-side and server-side applications, the web service "dacs_current_credentials" can return results in HTML, XML, and JSON formats as illustrated below:

HTML (the default):

```
You are authenticated within federation NFIS as:
1.  NRCAN:myusername
   at Tue Jan 17 17:28:40 2017 PST from 192.168.1.1 expires in about 12 hours
```

XML (specify FORMAT=XML in the url):

```
<dacs_current_credentials federation_name="NFIS"
federation_domain="nfis.org"><credentials federation="NFIS"
jurisdiction="NRCAN" name="myusername" roles="" auth_style="passwd"
cookie_name="DACS:NFIS::NRCAN:myusername"/></dacs_current_credentials>
```

JSON (specify FORMAT=JSON in the url):

```
{ "dacs_current_credentials": { "federation_name":"NFIS",
"federation_domain":"nfis.org", "credentials": [ { "federation":"NFIS",
"jurisdiction":"NRCAN", "name":"myusername", "roles":"",
"auth_style":"passwd", "cookie_name":"DACS:NFIS::NRCAN:myusername" } ] } }
```

This should provide the flexibility to integrate the result of dacs_current_credentials into any application.

8.1.5. Conclusion

Implementing a Single Sign-On solution into multiple applications – bridging commercial and open source solutions – is no easy task. However, this section illustrates four possible options for integrating any application with the DACS Single Sign-On solution proposed for the NFIS Platform. Deeper integration analysis for individual applications should be able to use this document as an excellent starting point.

8.2. CRIM Security Approach

CRIM reviewed the Distributed Access Control System (DACS) documentation provided by the sponsor, but did not attempt to install or test it. The approach proposed by CRIM, depicted in Figure 29 are in part overlapping with those presented previously, but substantial differences subsists. Just like DACS, an encrypted cookie is passed to the client's browser once authenticated. CRIM's HTTPS security proxy plays a similar filtering role as the Apache server in DACS, effectively rejecting requests that do not match criteria. The proxy currently does not reject requests based on a user's IP, only on the validity of the cookie and authorization checks against the ACL. In CRIM's solution, all access to data and services are made against the secured proxy. In no case can a user access the servers

directly. As the AuthZ/AuthN Service can be accessed by administrator either through a UI or REST calls, no edition to configuration files or restarting of servers is required.

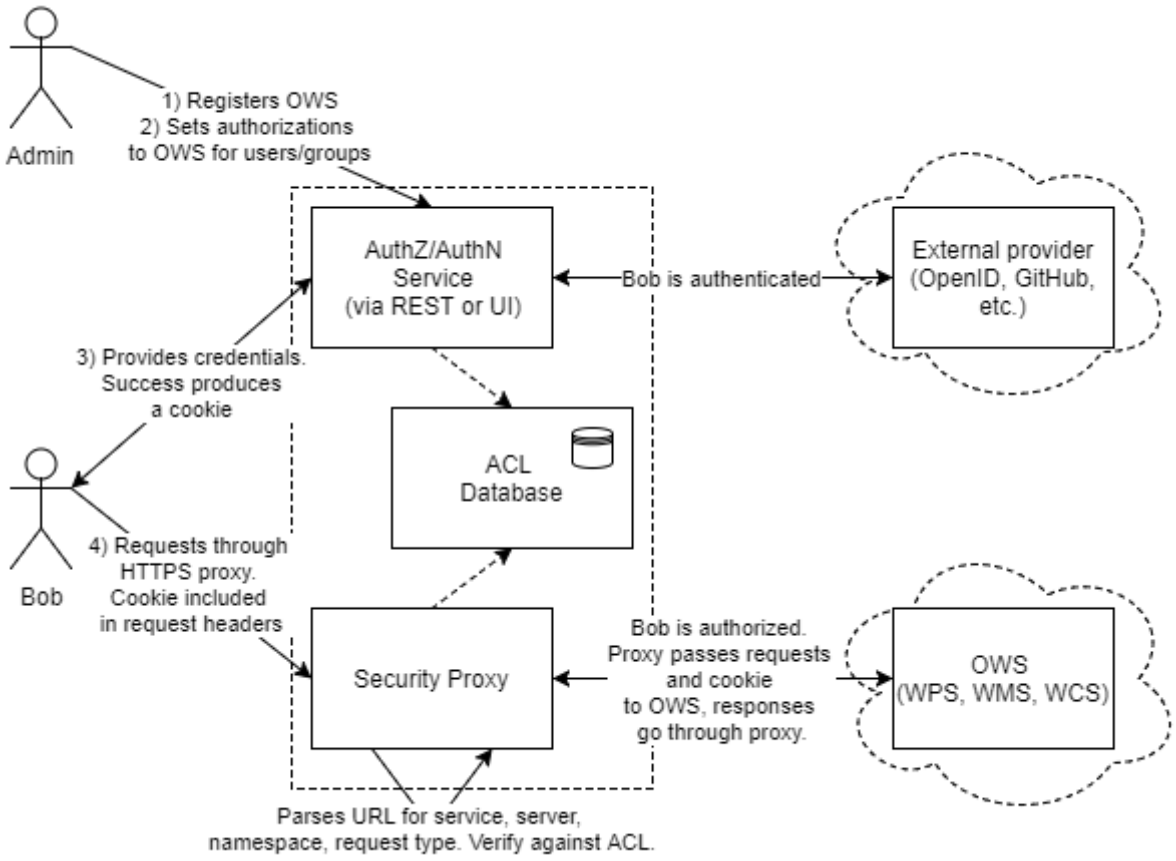


Figure 29. High-level view of CRIM research infrastructure security approach

Pyramid Web Framework was selected for implementation in CRIM cloud environment. This module constitutes a Policy Enforcement Point (PEP). A user session starts by obtaining this authentication token. Therefore, the user must first log in, either locally by providing a username/password saved in a local database or via an external provider such as OpenID or GitHub. When a user sends a request to access any type of resource, it needs to include an authentication token that will be decrypted by Pyramid to authenticate the user and the groups he belongs to. Once authenticated, Pyramid will authorize this user to access the resource by checking in a database the permissions this user has on that resource. If the login is successful, Pyramid provides an authentication token which is valid for a limited period. CRIM combined two dedicated libraries, Ziggurat-Foundations and Authomatic, to build a complete RESTful web service for AuthN/AuthZ to manage users, groups, resources, permissions and login along with the Pyramid framework.

Chapter 9. Test Experiments

Need to update the images to make more generic

Chen - add some testing and results for GMU

CRIM devised seven different experiments to offer to the EOC thread opportunity to identify interoperability scenarios and demonstration elements. The Technical Interoperability Experiments (TIE) also allows for a gradual implementation of its solution.

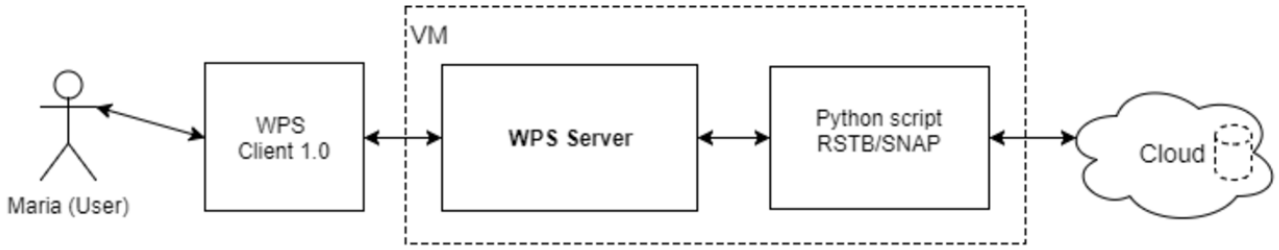


Figure 30. TIE 1: Classical execution of RSTB/SNAP as a script with WPS 1.0

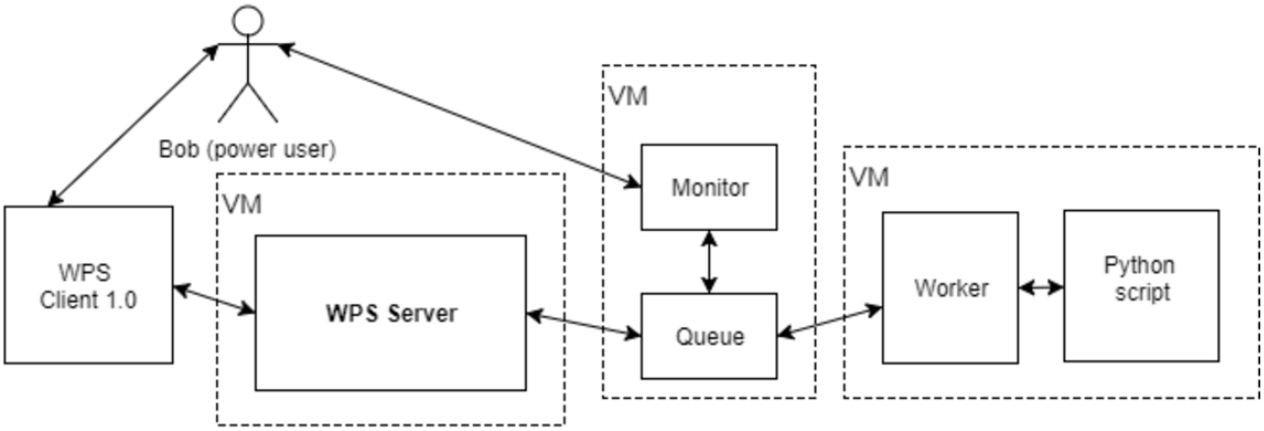


Figure 31. TIE 2: Jobs are published as tasks on a queue then a worker acknowledges the task

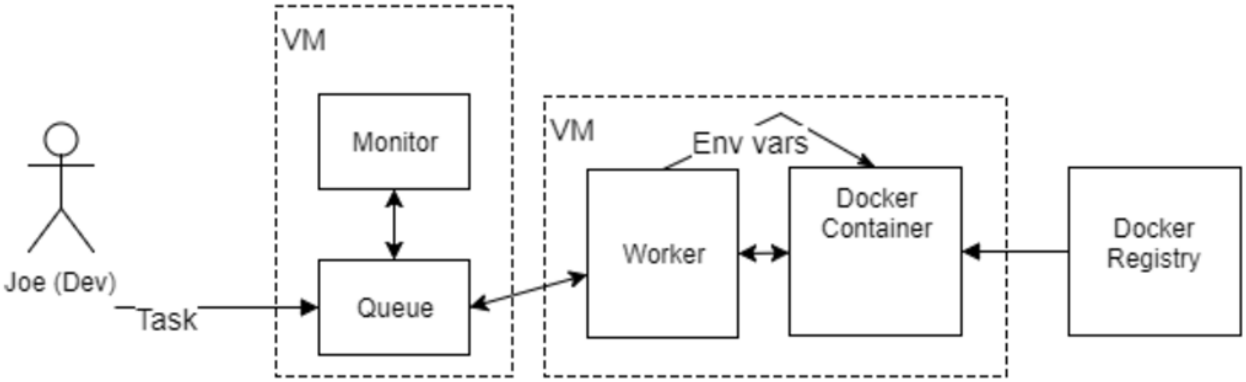


Figure 32. TIE 3: A task put on queue triggers a worker that in turns pulls a Docker image in the VM

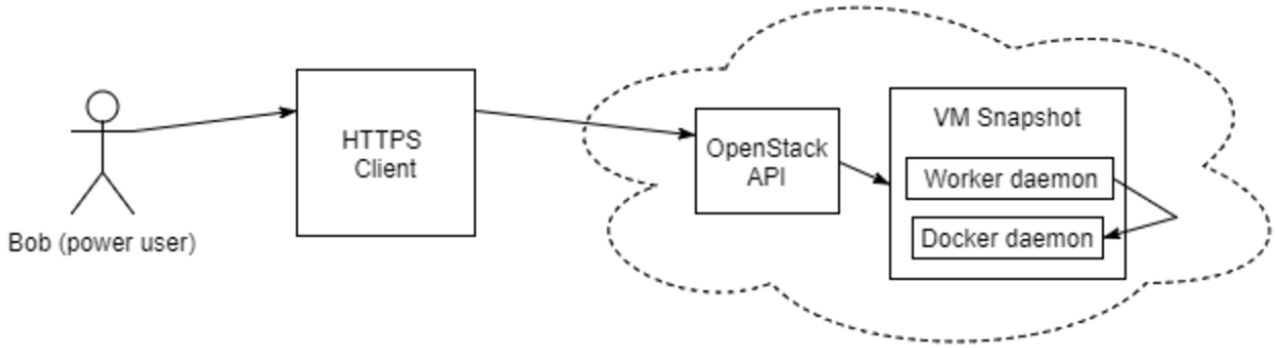


Figure 33. TIE 4: Authorized user manages VMs where VM snapshots contains both worker and Docker daemons

TIE 5 is a combination of TIE 2 and TIE 3. TIE5 insure a full round-trip between a WPS 1.0 Client, a WPS 1.0 Server, a task queue, a VM with a worker, a docker container from a Docker Registry.

TIE 6 is a data management experiment. It aims to insure adequate design for cloud, task queue and working directories. These experiments requires TIE 5 and TIE 1 and constitutes the bulk of the implementations required for our solution. Examples of requests and responses of TIE-6 can be found in Appendix D - WPS Functions.

TIE 7 is a facultative security experiment. It aims to authenticate a user and restrict execution and data access according to an Access Control List. This experiment was conducted successfully on several WPS 1.0 servers as well as on WCS/WMS data hosted in GeoServer. Please refer to [Security] section for details on the underlying implementation.

GMU publishes the GPTGraphProcess as the process to call GPT with the graph file which contains one or more pre-configured operations. The implementation is tested through GMUWPS dashboard: http://cloud.csiss.gmu.edu/GMUWPS/. In the test experiment, we test the GPTGraphProcess with the RADARSAT-2 sample dataset and GPT graph file which could be accessed at: http://cloud.csiss.gmu.edu/testbed/13/sample/.

The execution example of the GPTGraphProcess is shown as follow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute     version="2.0.0" service="WPS"
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" mode="async">
        <ows:Identifier>GPTGraphProcess</ows:Identifier>
        <wps:Input id="file">
            <wps:Reference
xlink:href="http://cloud.csiss.gmu.edu/testbed/13/sample/RS2_OK76385_PK678063
_DK606752_FQ2_20080415_143807_HH_VV_HV_VH_SLC.zip"/>
        </wps:Input>
    <wps:Input id="xml">
      <wps:Reference
xlink:href="http://cloud.csiss.gmu.edu/testbed/13/sample/mygraph.xml"/>
    </wps:Input>
        <wps:Output id="Result"
wps:dataTransmissionMode="reference"></wps:Output>
</wps:Execute>
```

After sending the Execute operation to the server, we can immediately get the response with JobID:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatusInfo xmlns="http://www.opengis.net/wps/2.0">
    <JobID>26a06e63-69a3-42d2-8893-a893eb065927</JobID>
    <Status>Accepted</Status>
</StatusInfo>
```

To check the status of this job, we can send the GetStatus request with the JobID:

```
<wps:GetStatus service="WPS" version="2.0.0"
xmlns:wps="http://www.opengis.net/wps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
<wps:JobID>26a06e63-69a3-42d2-8893-a893eb065927</wps:JobID>
</wps:GetStatus>
```

If the job is running, the status returned in the response would be "Running":

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatusInfo xmlns="http://www.opengis.net/wps/2.0">
    <JobID>26a06e63-69a3-42d2-8893-a893eb065927</JobID>
    <Status>Running</Status>
</StatusInfo>
```

Once the job is done, the status changes to "Succeeded". Then we can get the final product by sending GetResult operation:

```
<wps:GetResult service="WPS" version="2.0.0"
xmlns:wps="http://www.opengis.net/wps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<wps:JobID>26a06e63-69a3-42d2-8893-a893eb065927</wps:JobID>
</wps:GetResult>
```

The final result is shown as follow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Result xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.opengis.net/wps/2.0"
xmlns:ns3="http://www.opengis.net/ows/2.0">
    <ns4:ExpirationDate>2017-11-22T17:13:36.306-05:00</ns4:ExpirationDate>
    <ns4:Output id="Result">
        <ns4:Reference
ns2:href="http://cloud.csiss.gmu.edu/testbed/13/GMUWPS/GPTGraphProcess/output
/RS2_OK76385_PK678063_DK606752_FQ2_20080415_143807_HH_VV_HV_VH_SLC.zip.output
"/>
    </ns4:Output>
</ns4:Result>
```

9.1. Deployment Reproducibility Test

CRIM considers that TIE 3 constitutes valid deployability test. A worker listening to a queue receives a request containing: a docker image name and version number; the URL of a docker registry; a dictionary. The content of the dictionary is saved to a text file "env" with the syntax OGC_INPUT_key = value. The worker generates a docker-compose.yml file based on a template, where placeholders for docker image name and version are replaced by values from the request object. The worker launches the script docker-compose while passing "env" as an environment file and mounting volumes as needed. The selected docker image is started and has access to environment variables as defined in the "env" file. Once processing is done and the launched container is stopped, the output result from docker-

compose is retrieved, as well as data stored in the mounted volumes.

The request is emitted as follows:

```
>>> import Req
>>> from tasks import process
>>> x=Req.Req("hello1", "http://localhost", "ogc_helloworld2", "1.0",
{"HWPARAM1" : "2", "HWPARAM2" : "4"} )
>>> process(x)
```

The execution trace is shown below:

```
image name is helloworld2
Launching compose with helloworld2
WARNING: using --exit-code-from implies --abort-on-container-exit
Starting tie3_hw1_1 ...
Starting tie3_hw1_1 ... done
Attaching to tie3_hw1_1
hw1_1  | Hello world - second service
hw1_1  | Expecting OGC_INPUT_HWPARAM1, OGC_INPUT_HWPARAM2...
hw1_1  | Got OGC_INPUT_HWPARAM1: 2
hw1_1  | Got OGC_INPUT_HWPARAM2: 4
hw1_1  | OGC_INPUT_HWPARAM1 x OGC_INPUT_HWPARAM2 = 2x4 = 8
tie3_hw1_1 exited with code 0
Aborting on container exit...
retcode=0
```

CRIM considers that TIE 3 and TIE 4 constitutes valid reproductibility tests because of their support of Docker images, Docker-compose and cloud-init configuration files.

GMU provides an easy-to-use solution to deploy Cloud WPS. Cloud WPS is disseminated through VM template in qcow2 format which is a general VM format compatible with different cloud platforms including OpenStack, CloudStack, and AWS. Moreover, GMU offers the extendable solution that allow user extending the computing capability of Cloud WPS. When user deploy multiple VMs reproduced by the given template, the computing capability of Cloud WPS will be extended.

The template example could be accessed at: http://cloud.csiss.gmu.edu/testbed/13/template/testbed13-wps-template.qcow2. After downloading the template file, user can upload the template to the private or public cloud platforms. Each time the template has been deployed, a new VM would be created. At least one VM in the cloud suppose to set as the gateway with dashboard portal opening to users. This step could be set in the Apache config file located in the cloud management server. The example code to expose GMUWPS

dashboard portal in apache2.conf is shown as below:

```
ProxyPass /GMUWPS http://192.168.1.155:8080/GMUWPS
ProxyPassReverse /GMUWPS http://192.168.1.155:8080/GMUWPS
```

9.2. Interoperability Test

CRIM considers that TIE 2 constitutes a valid hybrid cloud interoperability test through the use of a shared task queue. TIE 1 is considered a valid WPS interoperability test.

In this TIE, we send a wps request to a PyWps server, which will transform that request in a task that will be a put in a Celery queue. Then, an available Celery worker will fetch the task from the queue and execute it. For example, a simple "sleep" request keeps the worker busy for 10 sec. If we send the request again immediately, another worker will fetch that new task and execute it. This experiment demonstrates that different tasks can be done in parallel thanks to a multiple workers listening to the same queue. Below is an example of request:

```
http://localhost:5000/wps?service=wps&request=Execute&version=1.0.0&identifie
r=sleep&datainputs=delay=10&storeExecuteResponse=true&status=true
```

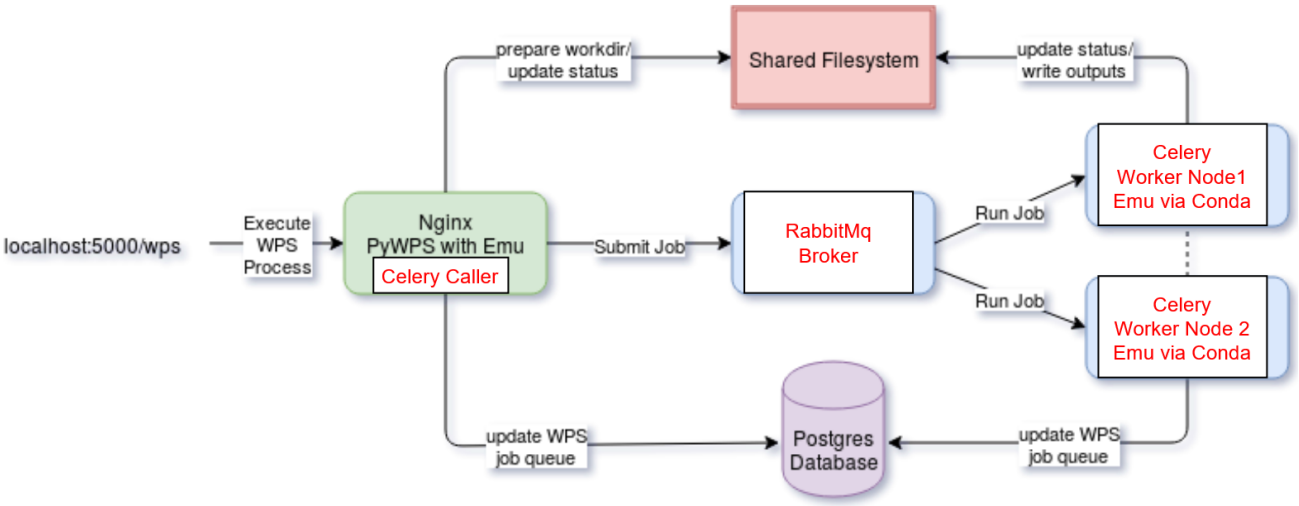


Figure 34. Diagram of a PyWPS sceduler extension developped for the testbed

9.3. Scalability Test

System scalability is conducted through TIE 4. Initial approach for TIE 4 involves the launch of a pre-configured VM snapshot on OpenStack. In order to better support configurability and other clouds support, we elected to instead use an initiation file. Using the Openstack command line client (CLI), we launch an OpenStack instance using a cloud-

init file including the required steps to: install the required packages (python-celery, docker, etc.); setup required environment variables for the worker; download the code of worker; start the worker. The instance ID returned by OpenStack is tracked. When the worker is not required anymore, we delete it using the OpenStack CLI. In this example, we have a rabbitMQ server available at 192.168.201.96. We then launch one or multiple instances that will run a simple Hello World worker.

Below is a cloud-init configuration file that we save as user_data.worker:

```
#cloud-config

write-files:
  - path: "/etc/environment"
    content: |
      export BROKER_URL=amqp://test:asdf@192.168.201.96//

package_update: true
packages:
  - python-celery
  - python-celery-common

runcmd:
  - 'export BROKER_URL=amqp://test:asdf@192.168.201.96//'
  - 'export C_FORCE_ROOT=true'
  - 'cd /home/ubuntu'
  - 'git clone https://github.com/myuser/hellow.git'
  - 'cd hellow'
  - 'celery -A tasks worker --loglevel=info'
```

We then launch a new instance using the user-data.worker file. The following example is meant to be ran at CRIM's internal R&D tenant:

```
openstack server create \
--flavor m1.tiny --security-group default \
--image 4ae0afa3-ff36-4b89-9c17-4fe7b6fa74ba \
--nic net-id=052eb4db-d04a-4364-b17e-ac85ee4a0e92 \
--key-name myuser-ptx-peld9 ogc-TIE4-worker5 --user-data user_data.worker
```

The elapsed time before the worker is ready and accepting jobs is 75 seconds. The OpenStack CLI will return a result similar to this:

Field	Value
OS-DCF:diskConfig	MANUAL

OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	NOSTATE
OS-EXT-STS:power_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	None
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	
adminPass	admpasseditedout
config_drive	
created	2017-10-12T21:20:01Z
flavor	m1.tiny (3afd523b-6ac7-490c-aadc-3828bb55361c)
hostId	b9d8f185-0d06-4c58-89f1-ed6e2e17f63e
image	Ubuntu Xenial 16.04.1 LTS (13 janvier 2017) (4ae0afa3-ff36-4b89-9c17-4fe7b6fa74ba)
key_name	myuser-ptx-peld9
name	ogc-TIE4-worker6
progress	0
project_id	39c275bd7cf647c1abe2e9ab4d10b907
properties	
security_groups	name='81ba932c-d52d-4a33-9511-a54d946c54a8'
status	BUILD
updated	2017-10-12T21:20:01Z
user_id	d5b7cf07101847068db7bcf669e3b6b4
volumes_attached	

In order to remove the newly create VM (to scale down), we need take note of the id of the VM and we emit the following command on OpenStack's CLI:

```
openstack server delete b9d8f185-0d06-4c58-89f1-ed6e2e17f63e
```

Chapter 10. Testbed 13 Demonstration

Northern boreal forests of Canada cover more than 2 million km2, distributed across vast regions that are largely inaccessible and poorly inventoried. Spatially-extensive, timely and cost effective inventory and monitoring are needed to better assess current status and to track the impacts of disturbances across a range of ecosystem services. Radarsat, Canada's primary remote sensing satellites, provides valuable information to Canada's forest ecosystem monitoring and natural resource management with better spatial and temporal coverage. In this project we use polarimetric SAR data from Canada's Radarsat-2 spaceborne radar satellite to conduct forest fire mapping over a large and far north region in NorthWest Territories (NWT).

Over the past two few years, more than 370 of Radarsat-2 standard polarimetric SAR images in the wide-beam mode (SQW) have been collected to cover a 2014 burned region in NWT, two coverages for summer of 2016 and two coverages for summer 2017. Each image covers 50km x 25km in area and has 300 Megabytes in size. This large amount of Radarsat-2 data need to be processed systematically, generating many polarimetric variants/ parameters that can be used to create SAR mosaics and assess the initial post-fire information. The mosaics can then be used for further quantitative analysis and/or a second-stage classification. Figure 1 shows a mosaic of 71 Radarsat-2 frames (yellow polygons) from summer 2016, created by using polarimetric parameters, such as entropy, alpha and lambda. The red polygons represent the burned areas occurred in 2014.

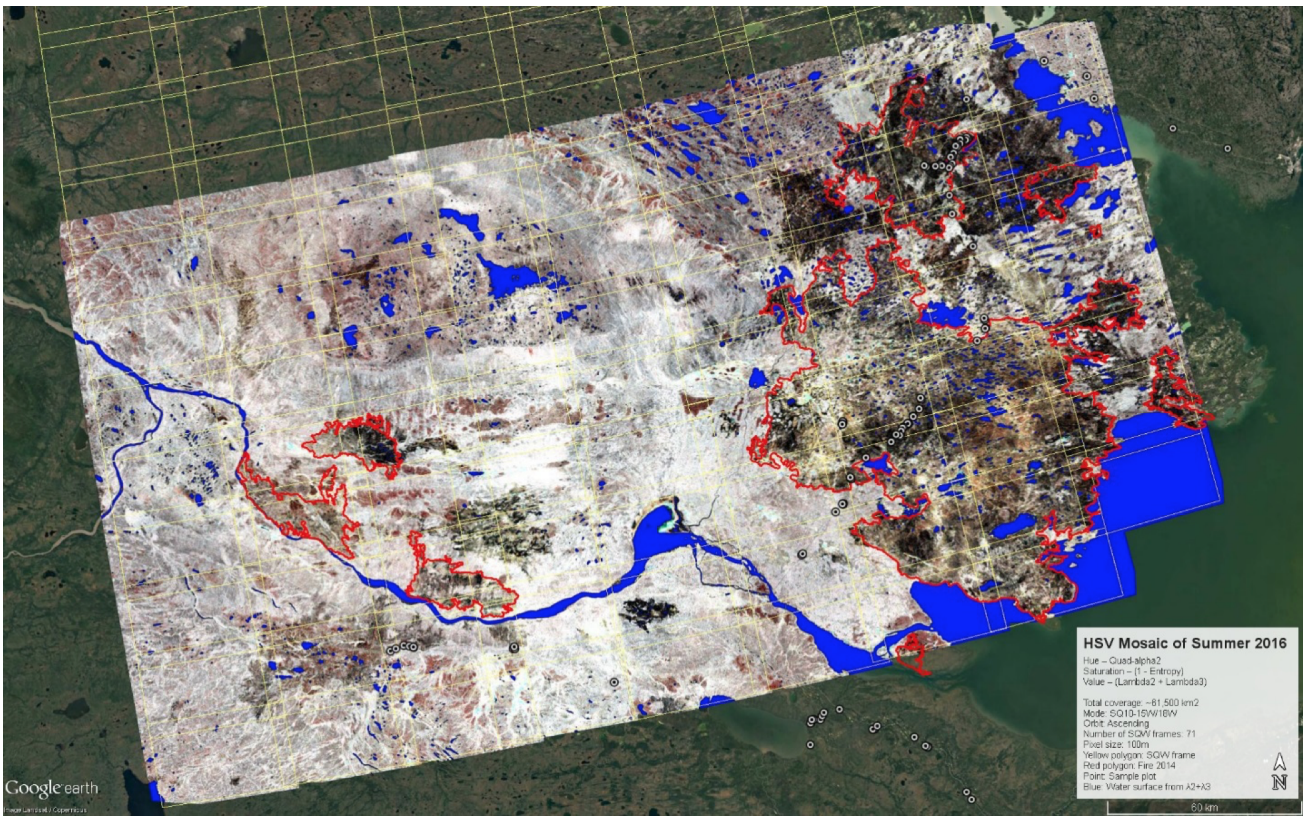


Figure 35. RADARSAT-2 Mosaic of Summer 2016

The Canadian Forest Service would like to test extracting polarimetric parameters from

Radarsat-2 SQW data using a combination of OGC Web services and Cloud environments (OGC Testbed 13). In the Cloud, processors and storage can be increased or decreased when needed. Resources are only used when necessary thus reducing overhead costs of maintaining expensive servers or computing power. In the future this implementation could be used for other large regions in Canada and possibly for the National Forest Inventory Plots across the country if our research and development are successful.

10.1. CRIM - BorealCloud Demonstration

Figure 36 depicts the hybrid cloud demonstration between BorealCloud and CRIM. BorealCloud is NRCan's high performance cloud infrastructure based on OpenStack technology at the Pacific Forestry Centre in Victoria, BC. Individual hosts VMs are identified by dashed boxes, while services/servers are plain boxes and assumes different ports. Inter-cloud access are depicted by arrows. In this scenario, three separate clouds are used. A master cloud is deployed on CRIM RD_ext tenant where all resources can be made available to the outside world. It contains the Docker Registry, a HTTPS fileserver, OWS services and Job Management components. All credentials to CRIM's resources are set once at configuration-time on slave clouds.

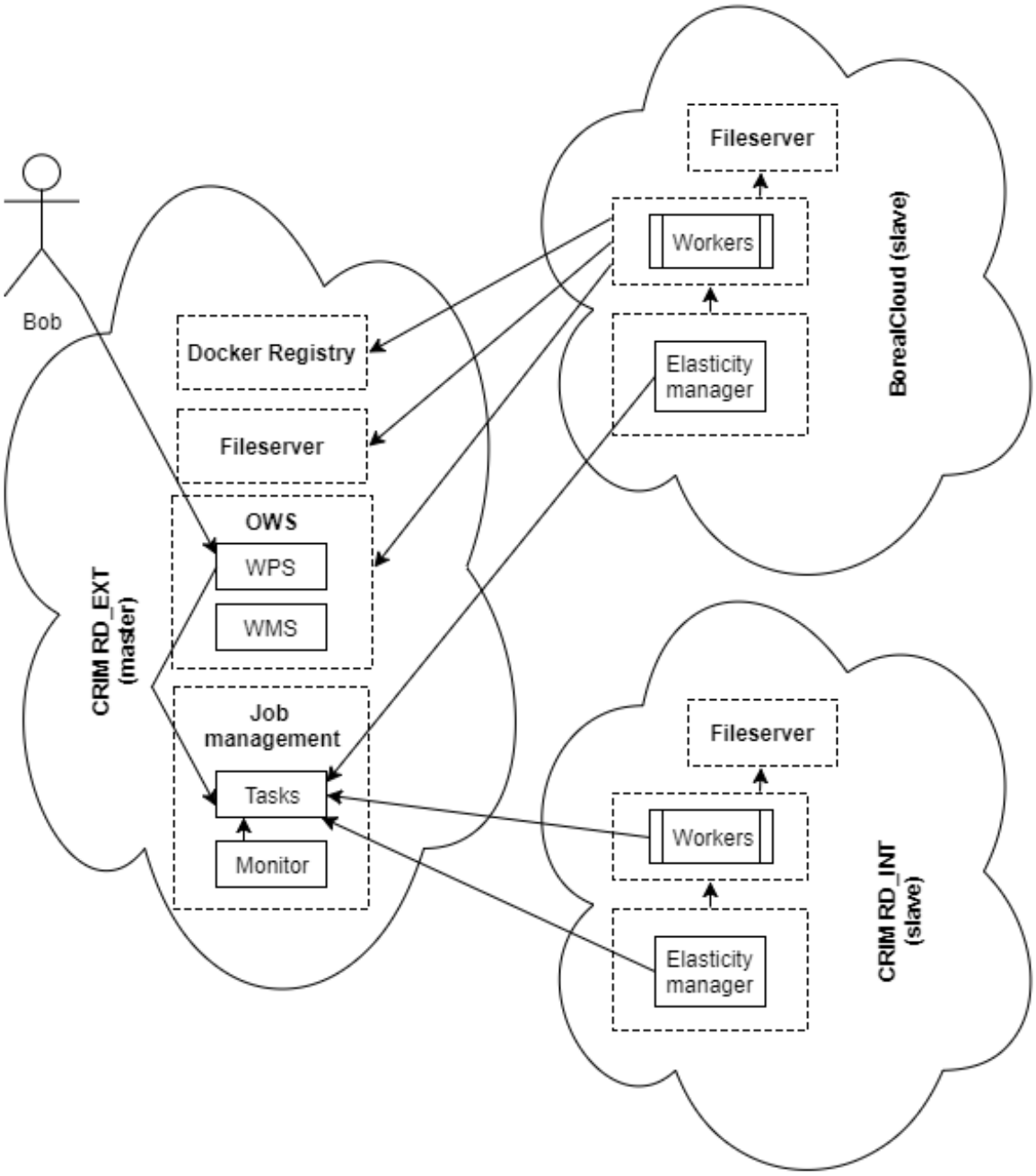


Figure 36. CRIM Research Infrastructure and Boreal Cloud Demonstration

Slave clouds are both on internal tenants; access to their resources from external hosts either difficult or impossible. Therefore, all accesses from slave clouds are outbound towards the master cloud. In this setup, Boreal's elasticity manager monitors the state of a Task Queue on CRIM's master cloud. It then creates (and tears down) processing VMs as required. Those processing VMs fetch a task, fetch the Docker App package, process the task and put the results on the fileserver and WMS server. CRIM's elasticity manager on the slave cloud will do the same, so both clouds will work together.

10.2. Results

10.3. Lessons Learned

Chapter 11. Summary

The EOC thread consists of separate sub-threads including the NRCan Cloud and ESA Cloud.

11.1. NRCan Architecture vs ESA Architecture

The following figure shows a high level comparison between ESA and NRCan architectures. It can be seen that both are identical to a large extent. The NRCan architecture identifies the App Management WPS as the key component an application consumer would work with. This component can be compared with the App Management Client in the ESA architecture. ESA has a dedicated App deployment and Execution Service, which is shown as a logical component in the NRCan part.

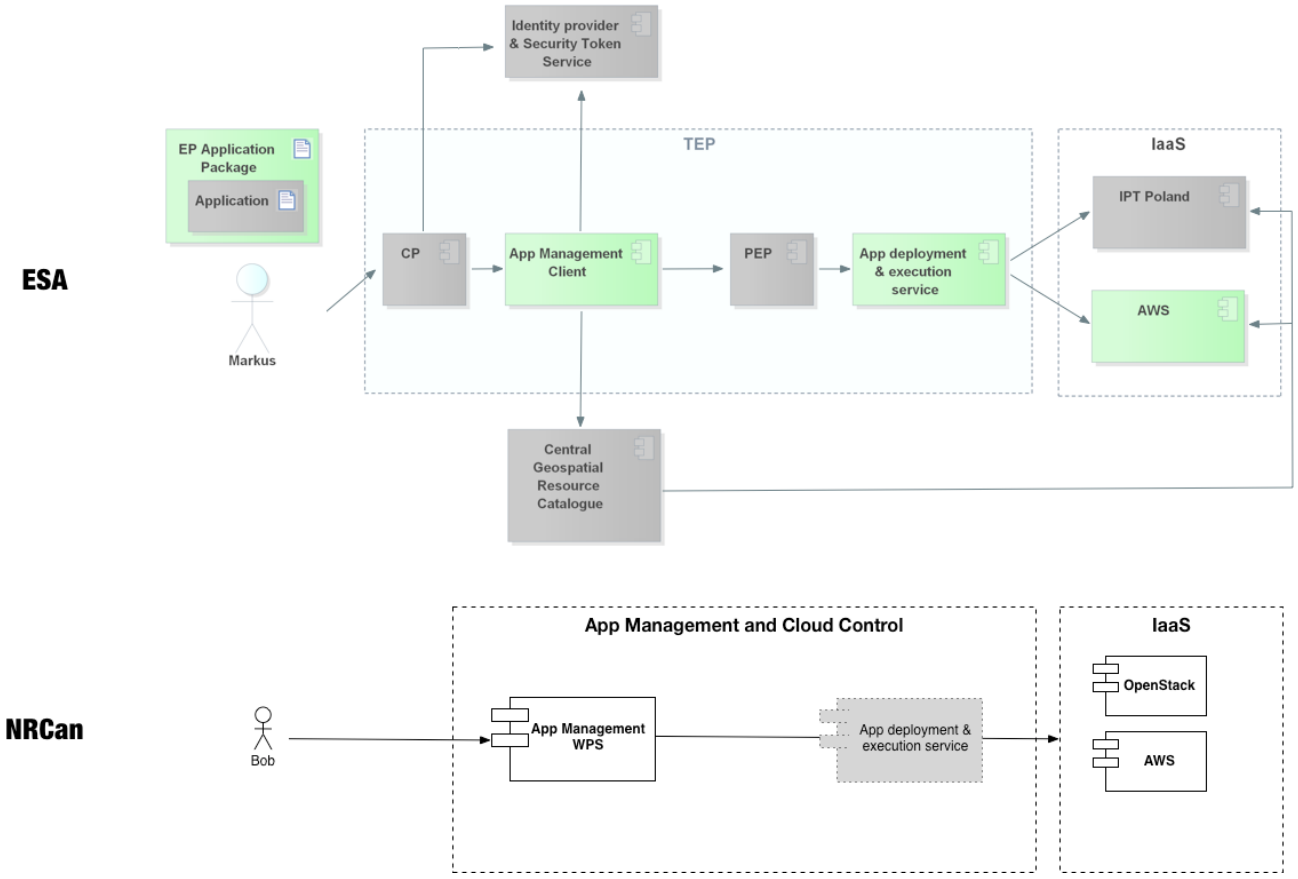


Figure 37. ESA and NRCan Cloud Deployment Architectures

11.2. Open Search

In EOC OpenSearch will be used to identify data collections which have a hosted processing services associated. ESA is participating in a parallel activity to define how this link between collections and services has to be encoded in GeoJSON and it has been decided to use the OWSContext offerings field. If for a WMS is clear what kind of information to put in offerings, that is the layers representing the data, it is not so

straightforward for a WPS. In fact there are different possibilities. One is to put the getCapability end point, which may be sufficient for the use case present in the testbed, but it could be also necessary to have a sort of filter to identify all the operations in the WPS which takes as input the products in the collection. An input from the testbed on this topic is requested in order to finalize the specification and the implementation of the GeoJSON interface which will be provided to the testbed.

11.3. Future Work

TBD

Implementation of WPS 2.0 as an application package manager similar to ESA/TEP thread.

How to handle failures (e.g. running out of memory for process How to respond to failures? Health monitoring of docker containers Health of applications Error reporting? Self-healing processes

Chapter 12. Appendix A - Data/Images

12.1. RS2-SLC-FQ9W-ASC-07-Sep-2016_01.35-PDS_05286240

(RS2_OK79000_PK698380_DK627316_FQ9W_20160907_013548_HH_VV_HV_VH_SLC)

Figure 38. Pauli, sigma0 + Improve Lee Sigma Filter (1,7x7,0.9,3x3), T22/T33/T11

12.2. RS2-SLC-FQ9W-ASC-07-Sep-2016_01.35-PDS_05286230

(RS2_OK79000_PK698380_DK627316_FQ9W_20160907_013546_HH_VV_HV_VH_SLC)


Edmonton
Calgary
3
Vancouver
Seattle
Portland
200 Km

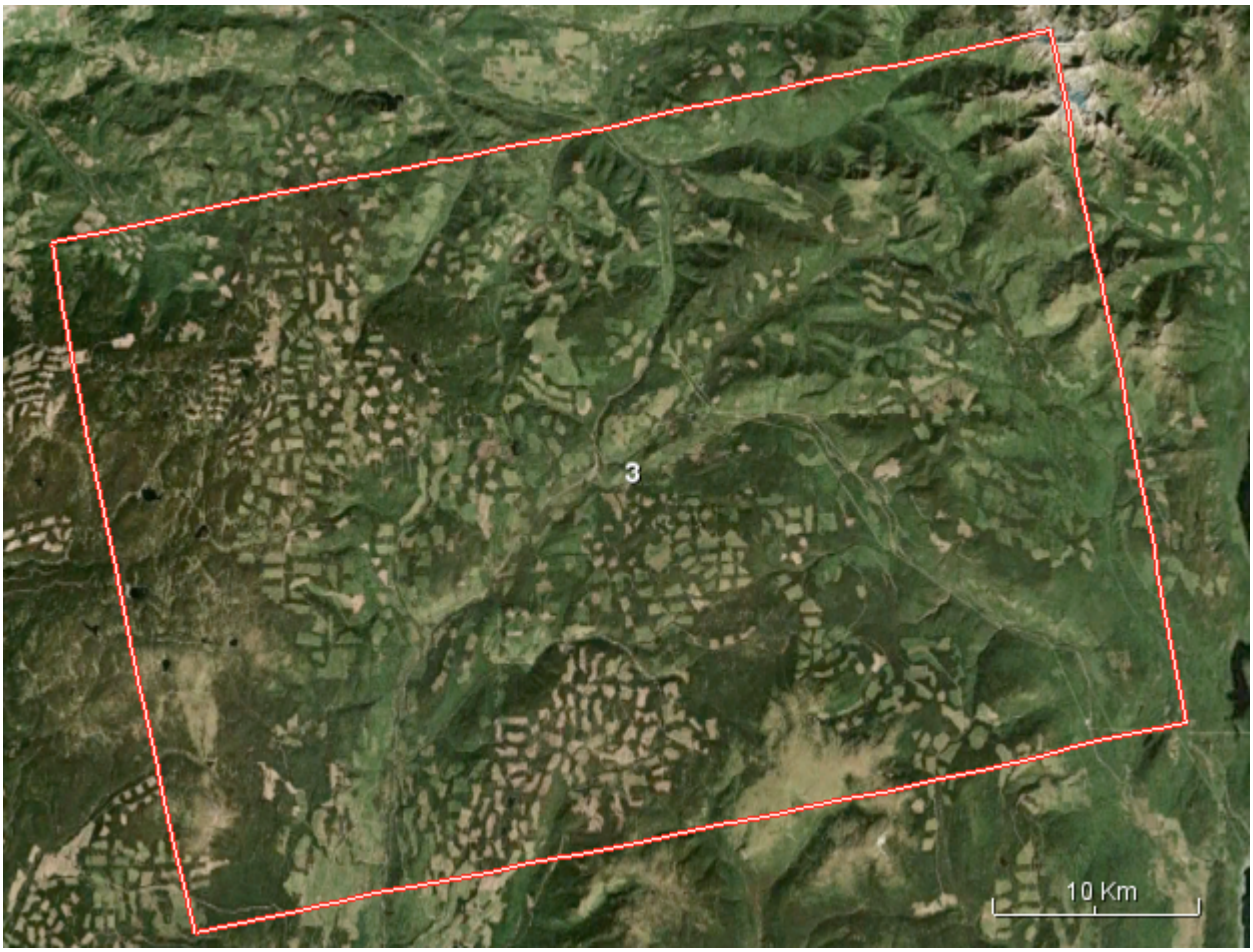
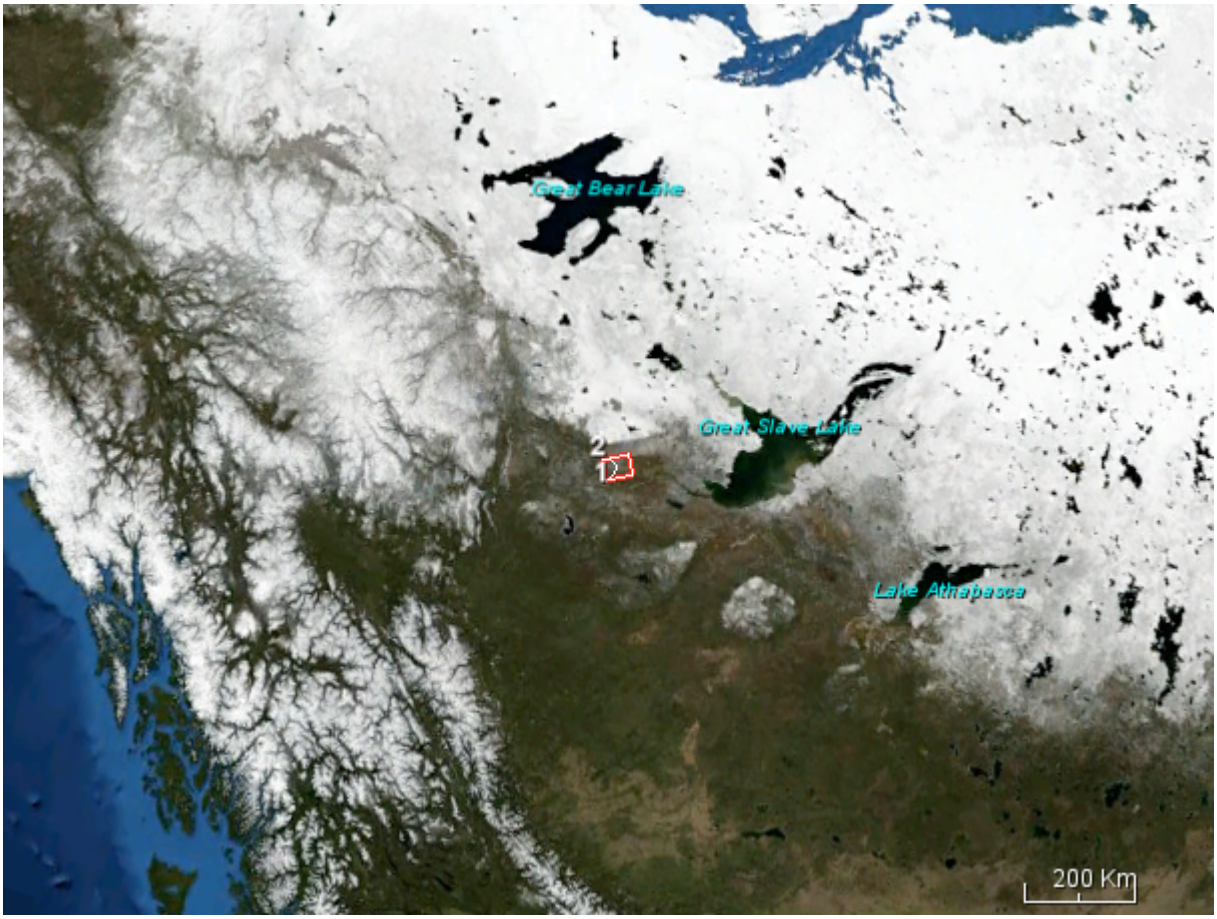
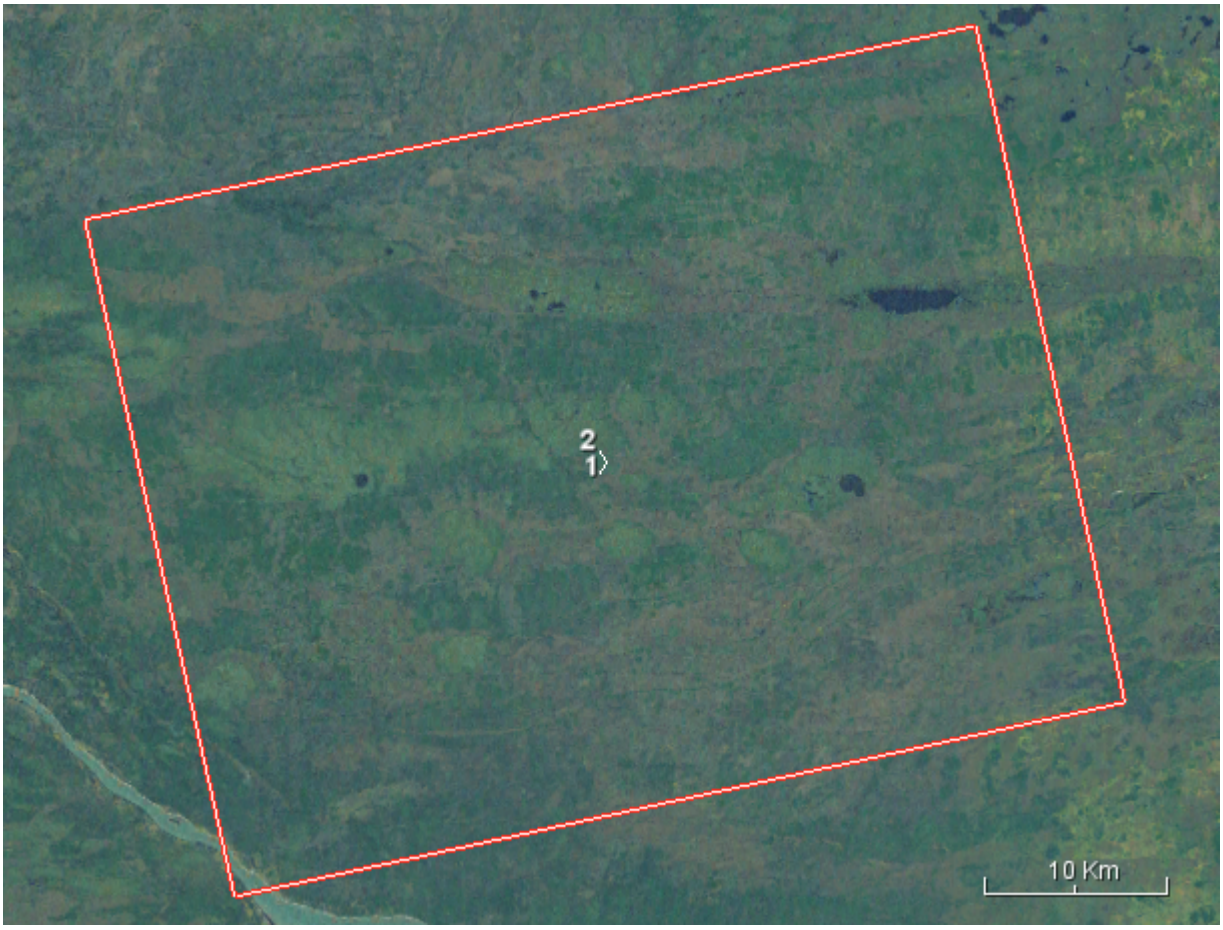

3
10 Km

Figure 39. Pauli, sigma0 + Improve Lee Sigma Filter (1,7x7,0.9,3x3), T22/T33/T11

12.3. RS2-SLC-SQ13W-ASC-11-Jul-2016_01.30-PDS_05181760

(RS2_OK77397_PK685920_DK616065_SQ13W_20160711_013038_HH_VV_HV_VH_SLC)




2
1
10 Km

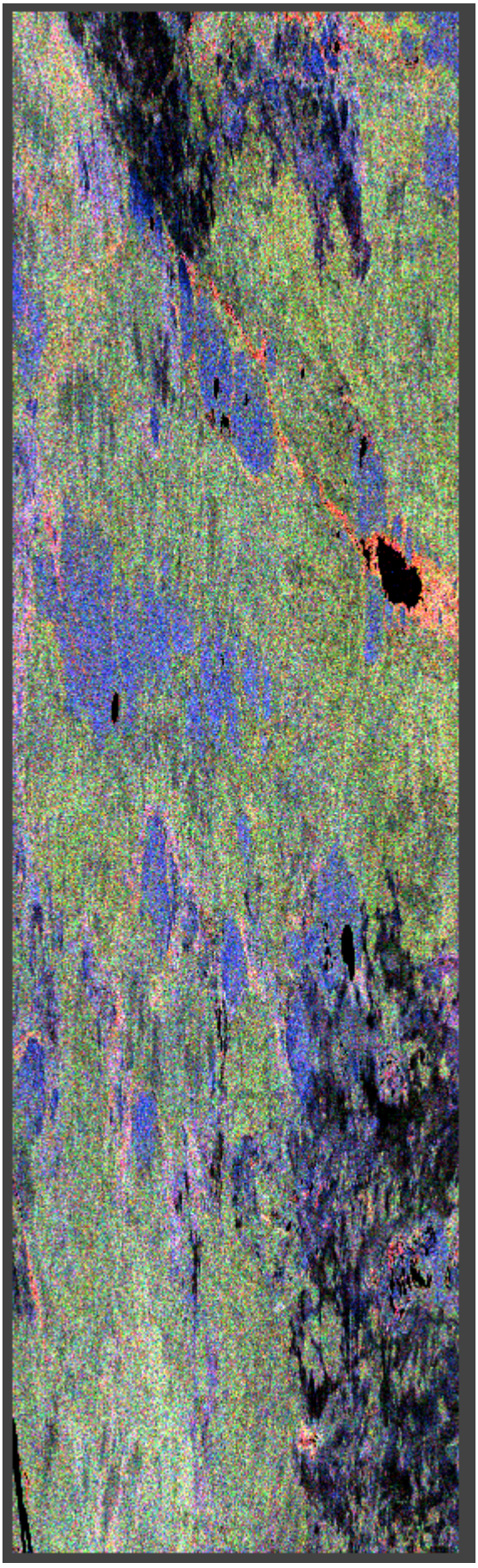
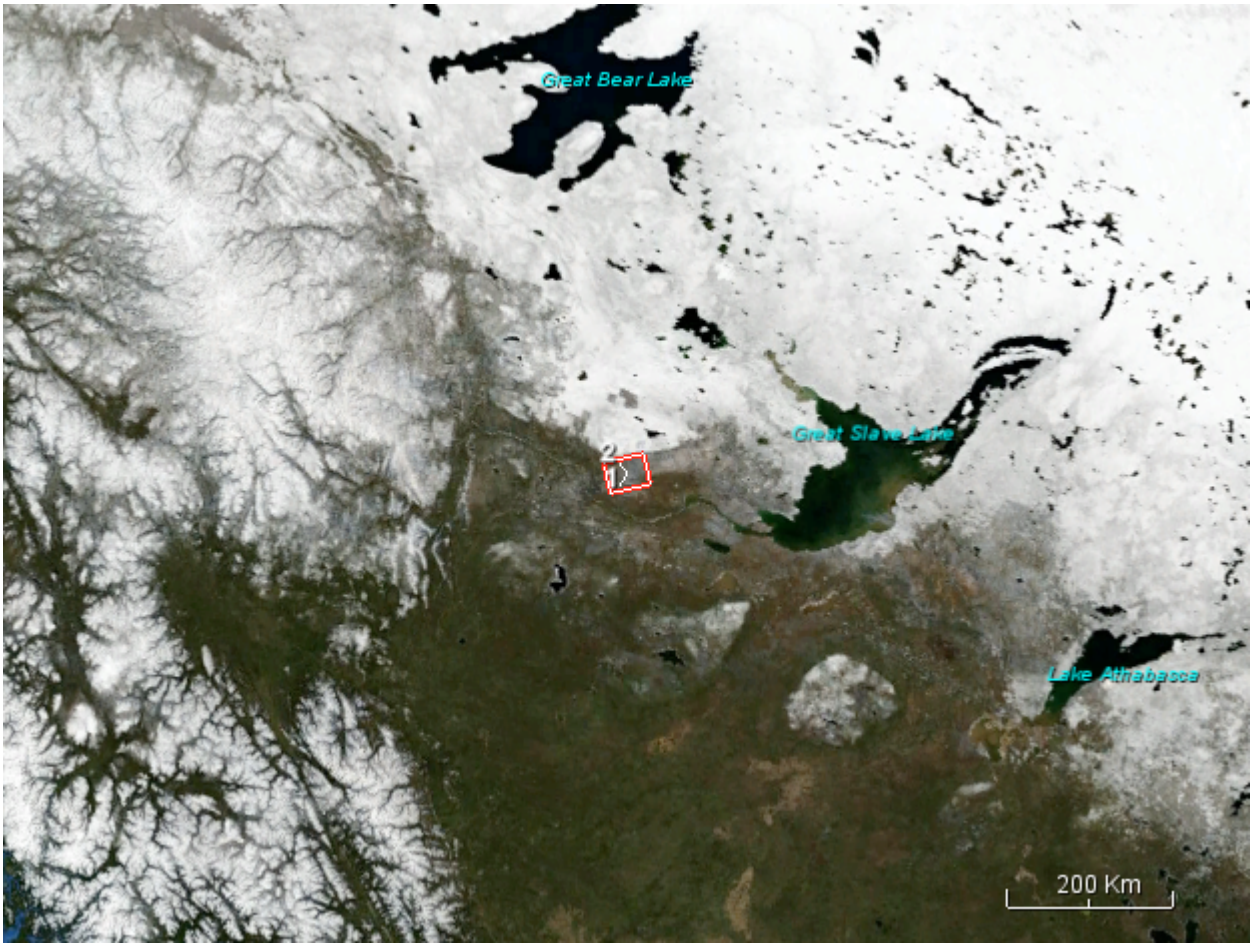


Figure 40. Pauli, sigma0 + Improve Lee Sigma Filter (1,7x7,0.9,3x3), T22/T33/T11

12.4. RS2-SLC-SQ13W-ASC-11-Jul-2016_01.30-PDS_05181760

(RS2_OK77397_PK685921_DK616066_SQ13W_20160711_013042_HH_VV_HV_VH_SLC)



Testbed

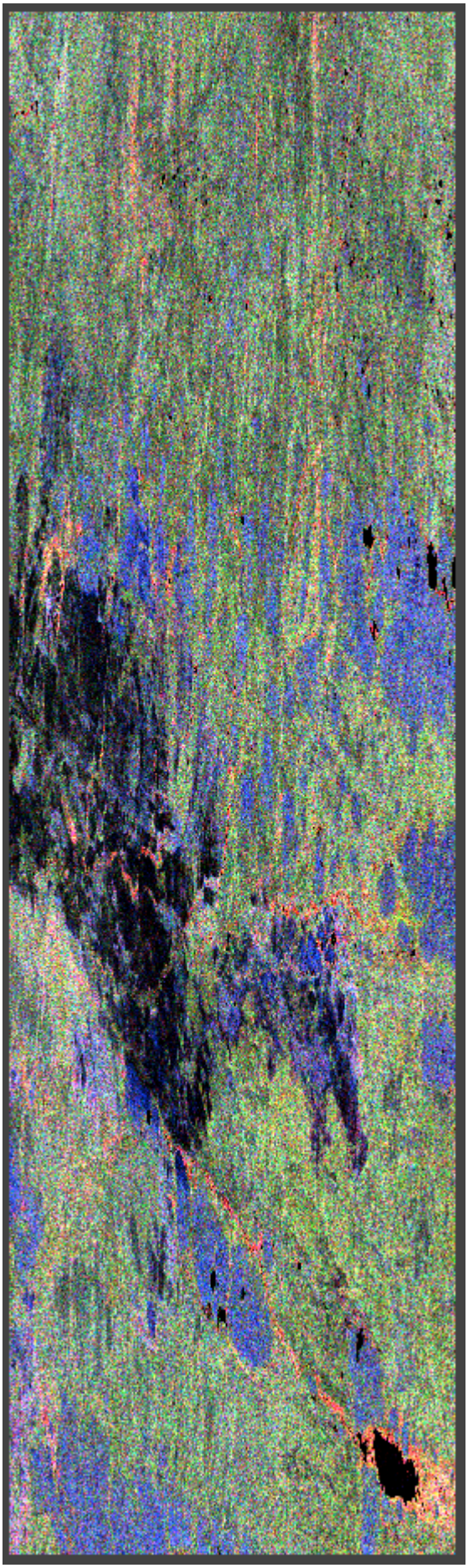


Figure 41. Pauli, sigma0 + Improve Lee Sigma Filter (1,7x7,0.9,3x3), T22/T33/T11

Chapter 13. Appendix B - Background on Compact Polarimetry

13.1. Transformation from a T3 Matrix to a Stokes vector (Compact-Pol)

$$\mathbf{T} = k_T \cdot k_T^{+} = \frac{1}{2}\begin{pmatrix} |S_{hh}|^2 + 2\Re(S_{hh}S_{vv}^*) + |S_{vv}|^2 & |S_{hh}|^2 - 2j\Im(S_{hh}S_{vv}^*) - |S_{vv}|^2 & 2S_{hv}S_{hh}^* + 2S_{hv}S_{vv}^* \\ |S_{hh}|^2 + 2j\Im(S_{hh}S_{vv}^*) - |S_{vv}|^2 & |S_{hh}|^2 - 2\Re(S_{hh}S_{vv}^*) + |S_{vv}|^2 & 2S_{hh}S_{hv}^* - 2S_{vv}S_{hv}^* \\ 2S_{hv}S_{hh}^* + 2S_{hv}S_{vv}^* & 2S_{hh}S_{hv}^* - 2S_{vv}S_{hv}^* & 4|S_{hv}|^2 \end{pmatrix}$$

T=[T_ii] (as described above)

S0= (T_11 + T_22 + T_33)/2 - Im(T_23) = Span / 2 - Im(T_23)

S1 Re(T_12) - Im(T_13)=

S2 Re(T_13) + Im(T_12)=

S4= Im(T_23) - (T_22 + T_33 - T_11) / 2

13.2. Transformation of RS2 bands (Stokes Quad-Pol) to Compact-Pol

band1: S_hh Re(S_hh) + jIm(S_hh)=

band2: S_hv

band3: S_vh (same as S_hv)

band4: S_vv

For example:

S0= (|S_hh|$^{2+2|S_hv|}$2+|S_vv|^2)/2 - Im(ShhShv $^{-SvvS\_hv}$) = (i_hh^2 + q_hh^2 +2*(i_hv^2 + q_hv^2) + i_vv^2 + q_vv^2)/2 - ((q_hh-q_vv)*i_hv-(i_hh-i_vv)*q_hv)

S1 = Re(T_12) - Im(T_13) = (i_hh^2 + q_hh^2 - (i_vv^2 + q_vv^2))/2 - ((q_hh+q_vv)*i_hv-(i_hh+i_vv)*q_hv)

S2 = Re(T_13) + Im(T_12) = ((i_hh+i_vv)*i_hv+(q_hh+q_vv)*q_hv) - (q_hh*i_vv-i_hh*q_vv)

S4 = Im(T_23) - (T_22 + T_33 - T_11) / 2 = q_hh-q_vv)*i_hv-(i_hh-i_vv)*q_hv) - (i_hv^2 + q_hv^2 - (i_hh*i_vv + q_hh*qvv

13.2.1. m-chi decomposition

$$m = \frac{\sqrt{S_1 + S_2 + S_3}}{S_0}$$

$$\chi = \frac{1}{2}\text{asin}(S_3 / S_0)$$

13.3. m-delta decomposition

% Convert a CP Stockes format to its Raney decomposition

% data [S0 S1 S2 S3 S4]=

% Ref: (Chen,2009) Unsupervised classification for Compact Pol

raney=zeros(size(data,1),size(data,2),2);

raney(:,:,1)=sqrt(data(:,:,2).^2+data(:,:,3).^2+data(:,:,4).^2)./data(:,:,1); %-- m

%raney(:,:,2)=atan(data(:,:,4)./data(:,:,3));

raney(:,:,2)=atan2(data(:,:,4),data(:,:,3)); %-- delta

raney(:,:,3) 0.5*asin(data(:,:,4)./data(:,:,1)); %-- Chi=

Chapter 14. Appendix C - Software Packages

14.1. CRIM WPS Software Configuration

Conda (4.3) - Conda is a cross-platform, language-agnostic binary package manager. It is the package manager used by Anaconda installations, but it may be used for other systems as well. Conda makes environments first-class citizens, making it easy to create independent environments even for C libraries. Conda is written entirely in Python, and is BSD licensed open source.

Docker Engine (1.10) - Software container platform used to run and manage applications in parallel to achieve greater compute density. Containers package software in a format that can run isolated on a shared operating system. Unlike VMs, containers to not bundle a full operating system, and only libraries and settings required to make the software run are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it's deployed.

Docker Registry - A stateless, highly scalable server-side application that stores and allows the distribution of Docker images. The Docker Registry is open-source, under the permissive Apache license.

Docker Hub - A web-based repository of software packages for use with docker. Docker Registry is contained in this repository.

PyWPS (4.0.0) - PyWPS is an implementation of the Web Processing Service standard from the Open Geospatial Consortium written in Python. PyWPS enables integration, publishing and execution of Python processes via the WPS standard. PyWPS is Open Source and released under an MIT license.

gunicorn (19.1.0) - 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

Python (2.7) - Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use. Python's license is administered by the Python Software Foundation.

Nginx (1.10.1) - NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.

Supervisor (3.3.1) - Supervisor is a client/server system that allows its users to monitor and

control a number of processes on UNIX-like operating systems.

14.1.1. CRIM Cloud Environment Configuration

Docker 1.10 for component packaging. Solution deployment to hosts with Docker Compose Docker instances managed and monitored with Portainer Based on RDO OpenStack. https://www.rdoproject.org Dashboard accessible at: http://ops.cloud.corpo.crim.ca/dashboard (password protected) Docker Instance: http://docker-registry.crim.ca/ogc/debian8-snap5-ogc-processingt:v1 Current configuration OpenStack : Juno v2014.2.1 (Dec. 2014) CEPH : Hammer v0.94.4 (Oct. 2015) VM OS: Ubuntu 14.04.5 LTS Preferred VM Config: m1.large. Was also tested on m1.medium 6 data volumes of 1 TB each Future configuration RDO Ocata + CEPH Jewel planned for Dec 2017

14.1.2. CRIM WMS/WCS Software Configuration

GeoServer 2.10.04 No additional plugin required Deployed at http://132.217.140.40:8080, (password protected) Data volume of 500 GB Installed with a Docker image

14.1.3. CRIM Additional Software Configuration

SNAP Desktop implementation : 5.0.8 SNAP Engine implementation : 5.0.8 Has been sucessfully tested in a SNAP 6.0 developper environment JRE: 1.8.0_102-b14 JVM: Java HotSpot™ 64-bit Server VM by Oracle Sentinel-1 Toolbox (S1TBX) version 5.0.5 Sentinel-2 Toolbox (S2TBX) version 5.0.7 Radarsat-2 Toolbox (RSTB) version 7.3.5 Docker version 1.12.5, build 7392c3b Python 2.7.5

14.2. GMU Software Configuration

SNAP 5.0 In EOC thread, it is imperative to use the Array Systems Computing RadarSat-2 Toolbox (RSTB) to process Radarsat-2 or other SAR and optical images. The RSTB is integrated in Sentinel Application Platform (SNAP) which could be executed by Graph Processing Framework (GPF) that allowing the user to create processing graphs for batch processing and customized processing chains. GMUWPS wraps some of operators (e.g. GPTWrite) and provides the capability to receive the GPT configuration XML file as the input of WPS request.

Docker CE 17.07 Docker provides the capability of manage containers in VMs. In the cloud infrastructure, Docker adds an additional layer between system level and application level, which makes applications deployed in the container isolate with each other. In addition, Docker images are lightweight, comparing with the VM template, the size of a Docker image is much smaller.

JDK 1.7.21 The Java Development Kit (JDK) is an implementation of either one of the Java

Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms[1] released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, macOS or Windows.

Tomcat 6 The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed under the Java Community Process.

Client - GMU WPS Dashboard To support OGC Testbed 13 Earth Observation Cloud (EOC) thread, a web-based GMU WPS Dashboard tool is developed. The demo version of GMUWPS Dashboard could be accessed at: http://cloud.csiss.gmu.edu/GMUWPS/. The tool provides an interface to send request and get response of WPS operations including GetCapabilities, DescribeProcess, Execute, GetStatus, and GetResult. Optimized for the EOC thread, the tool displays the cloud information such as Job Splitting, Job Priority, Cloud VM Usage, and Docker Usage.

14.2.1. GMU Cloud Environment Configuration

GeoBrain Cloud (Powered by Apache CloudStack) Portal: http://cloud.csiss.gmu.edu Platform: Apache CloudStack 4.9.2 VM OS: Ubuntu 14.04.5 LTS VM Specification: Medium Instance (CPU: 1Ghz, RAM: 1GB, Volume: 100GB)

Chapter 15. Appendix D - WPS Functions

15.1. GMU WPS Function Request/Response

GetCapabilities Request Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:GetCapabilities xmlns:ows="http://www.opengis.net/ows/2.0"
xmlns:wps="http://www.opengis.net/wps/2.0"   version="2.0.0" service="WPS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></wps:GetCapabilities>
```

GetCapabilities Response Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:Capabilities service="WPS" version="2.0.0"
xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns1="http://www.opengis.net/ows/2.0"
xmlns:ns3="http://www.opengis.net/wps/2.0">
    <ns1:ServiceIdentification>
        <ns1:Title xml:lang="en">GMU Web Processing Service</ns1:Title>
        <ns1:Abstract xml:lang="en">This service is developed for OGC Testbed
13 to integrate the backend cloud resources to process Sentinel satellite
datasets using RSTB toolbox.</ns1:Abstract>
        <ns1:Keywords>
            <ns1:Keyword>Geographical Information System</ns1:Keyword>
            <ns1:Keyword>Remote Sensing</ns1:Keyword>
            <ns1:Keyword>Geospatial Web Service</ns1:Keyword>
            <ns1:Keyword>Geoprocessing</ns1:Keyword>
            <ns1:Keyword>OGC</ns1:Keyword>
        </ns1:Keywords>
        <ns1:ServiceType>WPS</ns1:ServiceType>
        <ns1:ServiceTypeVersion>2.0.0</ns1:ServiceTypeVersion>
        <ns1:Fees>NONE</ns1:Fees>
        <ns1:AccessConstraints>NONE</ns1:AccessConstraints>
    </ns1:ServiceIdentification>
    <ns1:ServiceProvider>
        <ns1:ProviderName>Center for Spatial Information Science and Systems,
George Mason University</ns1:ProviderName>
        <ns1:ProviderSite ns2:href="http://csiss.gmu.edu"/>
        <ns1:ServiceContact>
            <ns1:IndividualName>Liping Di</ns1:IndividualName>
            <ns1:PositionName>Professor, Director</ns1:PositionName>
            <ns1:ContactInfo>
```

```
                <ns1:Address>

<ns1:ElectronicMailAddress>ldi@gmu.edu</ns1:ElectronicMailAddress>
                </ns1:Address>
            </ns1:ContactInfo>
            <ns1:Role>Administrator</ns1:Role>
        </ns1:ServiceContact>
    </ns1:ServiceProvider>
    <ns1:OperationsMetadata>
        <ns1:Operation name="GetCapabilities">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Get ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                    <ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
        </ns1:Operation>
        <ns1:Operation name="DescribeProcess">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Get ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                    <ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
        </ns1:Operation>
        <ns1:Operation name="Execute">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
        </ns1:Operation>
        <ns1:Operation name="GetStatus">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Get ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                    <ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
        </ns1:Operation>
        <ns1:Operation name="GetResult">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Get ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                    <ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
```

```
        </ns1:Operation>
        <ns1:Operation name="Dismiss">
            <ns1:DCP>
                <ns1:HTTP>
                    <ns1:Get ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>

<ns1:Post ns2:href="http://cloud.csiss.gmu.edu/gmuwps?"/>
                </ns1:HTTP>
            </ns1:DCP>
        </ns1:Operation>
    </ns1:OperationsMetadata>
    <ns3:Contents>
        <ns3:ProcessSummary jobControlOptions="sync-execute async-execute
dismiss">
            <ns1:Title>GPTWriteProcess wraps the write command in the gpt
application of RSTB.</ns1:Title>
            <ns1:Identifier>GPTWriteProcess</ns1:Identifier>
        </ns3:ProcessSummary>
        <ns3:ProcessSummary jobControlOptions="sync-execute async-execute
dismiss">
            <ns1:Title>Test Process for Demonstration Purpose</ns1:Title>
            <ns1:Identifier>TestProcess</ns1:Identifier>
        </ns3:ProcessSummary>
    </ns3:Contents>
</ns3:Capabilities>
```

15.1.1. Operation

DescribeProcess Request Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:DescribeProcess version="2.0.0" service="WPS"
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">
    <ows:Identifier>GPTWriteProcess</ows:Identifier>
</wps:DescribeProcess>
```

DescribeProcess Response Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:ProcessOfferings xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.opengis.net/ows/2.0"
xmlns:ns3="http://www.opengis.net/wps/2.0">
    <ns3:ProcessOffering jobControlOptions="sync-execute async-execute">
        <ns3:Process>
            <ns4:Title>SNAP GPT write</ns4:Title>
            <ns4:Abstract>GPTWriteProcess wraps the write command in the gpt
application of RSTB. </ns4:Abstract>
            <ns4:Identifier>GPTWriteProcess</ns4:Identifier>
            <ns3:Input maxOccurs="10" minOccurs="1">
                <ns4:Title>parameter for Pfile</ns4:Title>
                <ns4:Identifier>file</ns4:Identifier>
                <ns3:ComplexData>
                    <ns3:Format schema="xsd:anyURI" mimeType="text/xml"/>
                </ns3:ComplexData>
            </ns3:Input>
            <ns3:Input maxOccurs="1" minOccurs="1">
                <ns4:Title>parameter for PformatName</ns4:Title>
                <ns4:Identifier>formatName</ns4:Identifier>
                <ns3:LiteralData>
                    <ns3:Format schema="text/plain"/>
                </ns3:LiteralData>
            </ns3:Input>
            <ns3:Output>
                <ns4:Title>parameter for output file</ns4:Title>
                <ns4:Identifier>Result</ns4:Identifier>
                <ns3:ComplexData>
                    <ns3:Format schema="xsd:anyURI" mimeType="text/xml"/>
                </ns3:ComplexData>
            </ns3:Output>
        </ns3:Process>
    </ns3:ProcessOffering>
</ns3:ProcessOfferings>
```

15.1.2. Execute Operation

Execute Request Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute     version="2.0.0" service="WPS"
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" mode="async">
        <ows:Identifier>GPTWriteProcess</ows:Identifier>
        <wps:Input id="formatName">
            <wps:Data>GeoTiff</wps:Data>
        </wps:Input>
        <wps:Input id="file">
            <wps:Reference xlink:href="http://www.adorethelife.com/wp-
content/uploads/2016/06/beauty-and-make-up.jpg"/>
        </wps:Input>
        <wps:Output id="Result"
wps:dataTransmissionMode="reference"></wps:Output>
</wps:Execute>
```

Execute Response Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatusInfo xmlns="http://www.opengis.net/wps/2.0">
    <JobID>841033d6-73f1-4d6c-8048-1864815229b7</JobID>
    <Status>Running</Status>
</StatusInfo>
```

15.1.3. GPTWriteProcess Operation

An example of multi-image processing request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute     version="2.0.0" service="WPS"
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" mode="async">
        <ows:Identifier>GPTWriteProcess</ows:Identifier>
        <wps:Input id="formatName">
            <wps:Data>GeoTiff</wps:Data>
        </wps:Input>
        <wps:Input id="file">
            <wps:Reference xlink:href="http://192.168.1.155:8080/1024px-
Wfm_vancouver_island.jpg"/>
        </wps:Input>
        <wps:Input id="file">
            <wps:Reference
xlink:href="http://192.168.1.155:8080/9555468027_b4126a646a_b.jpg"/>
        </wps:Input>
        <wps:Input id="file">
            <wps:Reference xlink:href="http://192.168.1.155:8080/nasa-metro-
vancouver-fire-smoke-satellite-954x500.jpg"/>
        </wps:Input>
        <wps:Output id="Result"
wps:dataTransmissionMode="reference"></wps:Output>
</wps:Execute>
```

An example of multi-image processing response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Result xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.opengis.net/wps/2.0"
xmlns:ns3="http://www.opengis.net/ows/2.0">
    <ns4:ExpirationDate>2017-09-30T16:42:36.732-04:00</ns4:ExpirationDate>
    <ns4:Output id="Result">
        <ns4:Reference
ns2:href="http://cloud.csiss.gmu.edu/238/mapserv?map=/var/www/html/mapfile/aE
3bLzU.jpg.map&amp;SERVICE=WCS&amp;REQUEST=GetCoverage&amp;VERSION=1.0.0&amp;C
overage=aE3bLzU.jpg&amp;FORMAT=image/jpeg&amp;CRS=EPSG:4326&amp;BBOX=0,0,7012
,6108&amp;WIDTH=600&amp;HEIGHT=400"/>
    </ns4:Output>
    <ns4:Output id="Result">
        <ns4:Reference
ns2:href="http://cloud.csiss.gmu.edu/238/mapserv?map=/var/www/html/mapfile/rn
w77YN.jpg.map&amp;SERVICE=WCS&amp;REQUEST=GetCoverage&amp;VERSION=1.0.0&amp;C
overage=rnw77YN.jpg&amp;FORMAT=image/jpeg&amp;CRS=EPSG:4326&amp;BBOX=0,0,7012
,6108&amp;WIDTH=600&amp;HEIGHT=400"/>
    </ns4:Output>
    <ns4:Output id="Result">
        <ns4:Reference
ns2:href="http://cloud.csiss.gmu.edu/182/mapserv?map=/var/www/html/mapfile/Ws
AFHDV.jpg.map&amp;SERVICE=WCS&amp;REQUEST=GetCoverage&amp;VERSION=1.0.0&amp;C
overage=WsAFHDV.jpg&amp;FORMAT=image/jpeg&amp;CRS=EPSG:4326&amp;BBOX=0,0,7012
,6108&amp;WIDTH=600&amp;HEIGHT=400"/>
    </ns4:Output>
</ns4:Result>
```

15.1.4. GetStatus Operation

GetStatus Request Example:

```
<wps:GetStatus service="WPS" version="2.0.0"
xmlns:wps="http://www.opengis.net/wps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
<wps:JobID>841033d6-73f1-4d6c-8048-1864815229b7</wps:JobID>
</wps:GetStatus>
```

GetStatus Response Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatusInfo xmlns="http://www.opengis.net/wps/2.0">
    <JobID>841033d6-73f1-4d6c-8048-1864815229b7</JobID>
    <Status>Running</Status>
</StatusInfo>
```

15.1.5. GetResult Operation

GetResult Operation:

```
GetResult Request Example
<wps:GetResult service="WPS" version="2.0.0"
xmlns:wps="http://www.opengis.net/wps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">   <wps:JobID>841033d6-
73f1-4d6c-8048-1864815229b7</wps:JobID>
</wps:GetResult>
```

GetResult Response Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Result xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns4="http://www.opengis.net/wps/2.0"
xmlns:ns3="http://www.opengis.net/ows/2.0">
    <ns4:ExpirationDate>2017-09-16T11:30:20.401-04:00</ns4:ExpirationDate>
    <ns4:Output id="Result">
        <ns4:Reference
ns2:href="http://cloud.csiss.gmu.edu/mapserv?map=/var/www/html/mapfile/beauty
-and-make
-up.jpg.map&amp;SERVICE=WCS&amp;REQUEST=GetCoverage&amp;VERSION=1.0.0&amp;Cov
erage=beauty-and-make-
up.jpg&amp;FORMAT=image/jpeg&amp;CRS=EPSG:4326&amp;BBOX=0,0,7012,6108&amp;WID
TH=600&amp;HEIGHT=400"/>
    </ns4:Output>
</ns4:Result>
```

15.2. CRIM WPS Function Request/Response

15.2.1. Execute operation

Execute Request Example 1

http://10.1.35.59:48095/wps?service=wps&version=1.0.0&request=execute&identifier=hellodocker&datainputs=dockerim_name=docker-registry.crim.ca/ogc/inout_app;dockerim_version=tie6;registry_url=nimportequoi;queue_name=celery_tiny;url_to_download=https://www.unidata.ucar.edu/software/netcdf/examples/test_hgroups.nc;new_file_name=test_hgroups_from_unidata&storeExecuteResponse=true&status=true

Execute Response Example 1

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml"
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"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
version="1.0.0" xml:lang="en-US"
serviceInstance="http://localhost:8095/wps?service=WPS&request=GetCapabilitie
s"
statusLocation="http://localhost:8095/wps?service=wps&request=status&task_id=
82389ab2-307f-461c-989b-b2081cac5468">
    <wps:Process wps:processVersion="0.1">
        <ows:Identifier>hellodocker</ows:Identifier>
        <ows:Title>Hello Docker</ows:Title>
        <ows:Abstract>This wps exposes every parameters to the client of a
specific app, including the docker params (ows:context)</ows:Abstract>
    </wps:Process>
    <wps:Status creationTime="2017-11-10T20:17:36Z">
        <wps:ProcessAccepted>PyWPS Process hellodocker
accepted</wps:ProcessAccepted>
    </wps:Status>
</wps:ExecuteResponse>
```

As we can see, the response to the Execute operations contains an URL where the latest status of the task can be fetched. The task_id was provided by the Job Manager. Status responses for this request can be seen in the GetStatus operation examples 1 and 2 of the current appendix.

```
statusLocation="http://localhost:8095/wps?service=wps&request=status&task_id=
82389ab2-307f-461c-989b-b2081cac5468"
```

Execute Request Example 2

http://10.1.35.59:48095/wps?service=wps&version=1.0.0&request=execute&identifier=generate_dem_processing&datainputs=rsat2_product_xml_path=/data/RS2_OK79000_PK698379_DK627315_FQ9W_20160907_013546_HH_VV_HV_VH_SLC/product.xml;output_directory=/outputs/generate_dem_test;output_dem_filename=DEM_RS2_OK79000_PK698379_DK627315_FQ9W_20160907_013546_HH_VV_HV_VH_SLC_worker2.tif;download_directory=/outputs/download;queue_name=celery_medium&storeExecuteResponse=true&status=true

Execute Response Example 2

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml"
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"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
version="1.0.0" xml:lang="en-US"
serviceInstance="http://localhost:8095/wps?service=WPS&request=GetCapabilitie
s"
statusLocation="http://localhost:8095/wps?service=wps&request=status&task_id=
82389ab2-307f-461c-989b-b2081cac5468">
  <wps:Process wps:processVersion="1">
    <ows:Identifier>generate_dem_processing</ows:Identifier>
    <ows:Title>Generate Dem Processing</ows:Title>
    <ows:Abstract>This wps process only shows the process params, while the
docker params are in the description process itself</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2017-11-10T20:25:41Z">
    <wps:ProcessAccepted>PyWPS Process generate_dem_processing
accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```

The generate_dem_processing WPS is an EO process that generates images as outputs. When the process is a success, resulting images can be feteched at an exteranl URL. A sample URL for EO outputs can be seen in the GetStatus Response Example 3.

15.2.2. GetStatus operation

GetStatus Response Example 1

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml"
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"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
version="1.0.0" xml:lang="en-US"
serviceInstance="http://localhost:8095/wps?service=WPS&request=GetCapabilitie
s"
statusLocation="https://outarde.crim.ca:443/wpsoutputs/flyingpigeon/b5ddb6dc-
c881-11e7-8229-0242ac150009.xml">
  <wps:Process wps:processVersion="0.1">
    <ows:Identifier>hellodocker</ows:Identifier>
    <ows:Title>Hello Docker</ows:Title>
    <ows:Abstract>This wps exposes every parameters to the client of a
specific app, including the docker params (ows:context) </ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2017-11-13T14:48:54Z">
    <wps:ProcessSucceeded>Status: {'status': u'PROGRESS', 'metadata':
{u'current': 50, u'start_time': u'2017-11-10T20:19:32', u'total': 100,
u'host': u'celery-tin-e0794192-a47c-41b5-8b72-3e198603ac14',
u'worker_id_version': None}, 'uuid': u'82389ab2-307f-461c-989b-b2081cac5468',
'result': {u'current': 50, u'start_time': u'2017-11-10T20:19:32', u'total':
100, u'host': u'celery-tin-e0794192-a47c-41b5-8b72-3e198603ac14',
u'worker_id_version': None}}</wps:ProcessSucceeded>
  </wps:Status>
</wps:ExecuteResponse>
```

GetStatus Response Example 2

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml"
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"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
version="1.0.0" xml:lang="en-US"
serviceInstance="http://localhost:8095/wps?service=WPS&request=GetCapabilitie
s"
statusLocation="https://outarde.crim.ca:443/wpsoutputs/flyingpigeon/b5ddb6dc-
c881-11e7-8229-0242ac150009.xml">
  <wps:Process wps:processVersion="0.1">
    <ows:Identifier>hellodocker</ows:Identifier>
    <ows:Title>Hello Docker</ows:Title>
    <ows:Abstract>This wps process only shows the process params, while the
docker params are in the description process itself</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2017-11-13T14:48:54Z">
    <wps:ProcessSucceeded>Status: {'status': u'SUCCESS', 'metadata':
{u'result': {u'type': u'Useless', u'value': {u'volume_mapping': {},
u'dockerim_name': u'docker-registry.crim.ca/ogc/inout_app',
u'param_as_envar': True, u'queue_name': u'celery', u'input_data':
{u'dockerim_name': u'docker-registry.crim.ca/ogc/inout_app',
u'new_file_name': u'test_hgroups_from_unidata', u'queue_name':
u'celery_tiny', u'url_to_download':
u'https://www.unidata.ucar.edu/software/netcdf/examples/test_hgroups.nc',
u'registry_url': u'nimportequoi', u'dockerim_version': u'tie6'},
u'dockerim_version': u'tie6', u'registry_url': u'nimportequoi'}}}, 'uuid':
u'82389ab2-307f-461c-989b-b2081cac5468', 'result': {u'result': {u'type':
u'Useless', u'value': {u'volume_mapping': {}, u'dockerim_name': u'docker-
registry.crim.ca/ogc/inout_app', u'param_as_envar': True, u'queue_name':
u'celery', u'input_data': {u'dockerim_name': u'docker-
registry.crim.ca/ogc/inout_app', u'new_file_name':
u'test_hgroups_from_unidata', u'queue_name': u'celery_tiny',
u'url_to_download':
u'https://www.unidata.ucar.edu/software/netcdf/examples/test_hgroups.nc',
u'registry_url': u'nimportequoi', u'dockerim_version': u'tie6'},
u'dockerim_version': u'tie6', u'registry_url':
u'nimportequoi'}}}}</wps:ProcessSucceeded>
  </wps:Status>
</wps:ExecuteResponse>
```

GetStatus Response Example 3

http://localhost:3000/fs/tasks/82389ab2-307f-461c-989b-b2081cac5468/outputs/generate_dem_test/DEM_RS2_OK79000_PK698379_DK627315_FQ9W_20160907_013546_HH_VV_HV_VH_SLC_worker2.tif

Chapter 16. Appendix E - WPS Process Descriptions

16.1. CRIM Process Description

```
<!-- PyWPS 4.0.0 -->
<wps:ProcessDescriptions xmlns:gml="http://www.opengis.net/gml"
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"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
service="WPS" version="1.0.0" xml:lang="en-US">
  <ProcessDescription wps:processVersion="1" storeSupported="true"
statusSupported="true">
    <ows:Identifier>NR102</ows:Identifier>
    <ows:Title>Cloud WPS Biomass with WCS/WMS support 2</ows:Title>
    <DataInputs>
<!-- Docker image params -->
    <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>docker_image</ows:Identifier>
    <ows:Title>URI of the Docker Image to be deployed and
executed</ows:Title>
    <ows:Abstract>The URI contains the full path to a Docker image as used by
Docker Daemon, including the host, port, path, image name and version. This
input parameter does not support credentials. Credentials for private Docker
registries are set as a system configuration. The credentials are injected in
the environment variables of the VM instance that runs the Docker
Image.</ows:Abstract>
    <LiteralData>
      <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
      <ows:AnyValue/>
    </LiteralData>
    </Input>
<!-- Cloud params -->
    <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>IaaS_deploy_execute</ows:Identifier>
        <ows:Title>URI of the IaaS resource where the job will be deployed
and executed</ows:Title>
    <ows:Abstract>If the WPS Server contains a Task Queue scheduler, the URI
contains two part. The first part is the URI of the Message Broker in the
form of amqp://broker_ip:broker_port//. The second part is the Task Queue
```

```
name. For simplicity, both part are appended in a single string. This input
parameter does not support credentials. Credentials for Message brokers are
set as a system configuration. The credentials are injected in the
environment variables of the VM instance that will host the WPS
Server.</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
    <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>IaaS_datastore</ows:Identifier>
        <ows:Title>URI of an IaaS data store where the outputs will
stored</ows:Title>
    <ows:Abstract>This parameter sets the target for all outputs of the
process (HTTPS fileserver, AWS S3, SWIFT, Globus, etc.). Outputs will be
staged out in the datastore by the process. The current implementation only
supports HTTP fileservers. This input parameter does not support credentials.
Credentials for datastores are set as a system configuration. The credentials
are injected in the environment variables of the VM instance that runs the
Docker Image.</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
<!-- Process params -->
        <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>Radarsat2_data</ows:Identifier>
        <ows:Title>URI from where to download the Radarsat-2 data
ZIP</ows:Title>
    <ows:Abstract>The Radarsat-2 file is unzipped in the local drive. The
product.xml file is located then provided to RSTB/SNAP. The long string
identifying the product is used to create temporary directories and to format
the output file names.</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
    <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>WMS_server</ows:Identifier>
        <ows:Title>URI where to register a WMS-compatible output</ows:Title>
    <ows:Abstract>The process produces an RGB image of the data output. Its
```

```
smaller footprint is better managed by WMS/WCS servers. The RGB output will
be staged out in the specified WMS Server by the process. The output
parameter named output_data_WMS_url will contain an URL pointing to this WMS
server. The credentials are injected in the environment variables of the VM
instance that runs the Docker Image.</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
    <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>input_graph_url</ows:Identifier>
        <ows:Title>URL where to download the GPT graph used to process the
Radarsat-2 data</ows:Title>
    <ows:Abstract>Allows a user to provide a different processing graph to
the process. In case none is specified, a default graph stored in the
application package is used. A graph provided here should present the same
Inputs (reads) and Ouputs (writes) as the default graph.</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
    <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>input_graph_parameters</ows:Identifier>
        <ows:Title>KVP used to parametrize the graph itself</ows:Title>
    <ows:Abstract>Allows a user to provide customized parameters to the graph
in the form of a JSON file. In case none are specified, default values will
be used. Currently, the default graph supports Polarimetric-Speckle-
Filter.filter, Polarimetric-Speckle-Filter.windowSize and Polarimetric-
Speckle-Filter.numLooksStr</ows:Abstract>
        <LiteralData>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
    </DataInputs>
    <ProcessOutputs>
      <Output>
        <ows:Identifier>output_data_url</ows:Identifier>
        <ows:Title>URL to data produced by the process</ows:Title>
    <ows:Abstract>The URL provided here is dependent on the IaaS_datastore
selected. It allows an user to access and download from the Cloud the image
data produced by the process. By default in the current implementation, the
```

```
output data is accessible through HTTP fileservers.</ows:Abstract>
        <LiteralOutput>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
        </LiteralOutput>
      </Output>
      <Output>
        <ows:Identifier>output_data_WMS_url</ows:Identifier>
        <ows:Title>URL to WMS layer for the data produced</ows:Title>
    <ows:Abstract>The URL provided here is dependent on the WMS_server
provided. If no WMS server was specified, this field is left blank. The URL
allows an user to access the image data produced by the process in WMS
client.</ows:Abstract>
        <LiteralOutput>
          <ows:DataType
ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
        </LiteralOutput>
      </Output>
    </ProcessOutputs>
  </ProcessDescription>
</wps:ProcessDescriptions>
```