OGC® DOCUMENT: 23-044
External identifier of this OGC® document: http://www.opengis.net/doc/PER/T19-D081

Open Geospatial Consortium

OGC TESTBED 19 HIGH PERFORMANCE GEOSPATIAL COMPUTING ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2023-11-16
Approval Date: 2024-01-25
Publication Date: 2024-04-26
Editor: Eugene Yu, Liping Di

Note

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

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

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I EXECUTIVE SUMMARY

Large-scale geospatial analytical computation is critically needed for tackling a wide range of sustainability problems, such as climate change, disaster management, and food and water security. However, such computation often requires high-performance computing (HPC) resources that are not easily accessible or usable by geospatial researchers and practitioners from various domains. To address this challenge, there is a need for developing and standardizing tools and interfaces that can bridge the gap between user frontend and HPC backends and enable effective and efficient use of High-Performance Geospatial Computing (HPGC) resources for geospatial analytics.

This OGC Testbed 19 Engineering Report (ER) presents the results of a testbed task that:

- evaluated previous and current work in the application of HPC for geospatial analytics, and
- developed draft standards for HPGC resource definitions and processing interfaces.

This ER provides an overview of the Testbed 19 motivation, objectives, scope, and methodology, as well as a summary of the main findings, recommendations, and future work directions.

CyberGIS-Compute is reviewed and used as a reference to develop the HPGC API. “CyberGIS-Compute is an open-sourced geospatial middleware framework that provides integrated access to high-performance computing (HPC) resources through a Python-based SDK and core middleware services.”[3] The OGC API — Processes[14] is adopted as the base API for standardizing and developing the HPGC API. A Python client library is developed to demonstrate the process of client generation by leveraging the OpenAPI client stub/model automatic generation capability[12]. Typical use cases and scenarios are demonstrated and scripted in Jupyter Notebooks.

II KEYWORDS

The following are keywords to be used by search engines and document catalogues.

OGC, Testbed 19, high performance computing, high performance geospatial computing, application-to-the-cloud, testbed, docker, web service

III CONTRIBUTORS

All questions regarding this document should be directed to the editors or contributors.

NAME	ORGANIZATION	ROLE
Eugene Yu	George Mason University	Editor
Liping Di	George Mason University	Editor
Sina Taghavikish	OGC	Task Lead
Furqan Baig	University of Illinois Urbana-Champaign	Contributor
Gérald Fenoy	Geolabs	Contributor
Carl Reed	Carl Reed and Associates	Content Reviewer

1

INTRODUCTION

1 INTRODUCTION

The field of large-scale geospatial analytical computation has become increasingly vital in addressing a diverse range of sustainability challenges, including climate change mitigation, disaster management, and ensuring food and water security. Geospatial researchers and practitioners from various disciplines, such as geography, hydrology, public health, and social sciences, are actively engaged in utilizing geospatial analytics to derive valuable insights.

Advanced cyberinfrastructure and expertise in computer science have empowered large-scale computational problem-solving. However, expecting domain experts in geospatial-related fields to possess extensive technical knowledge to directly interact with high-performance computing (HPC) resources on advanced cyberinfrastructure is not realistic. Optimization of HPC resources for the geospatial community's specific computational challenges requires a bridge between user frontends and HPC backends in the form of middleware tools.

To address this gap, designing and implementing middleware tools that enable seamless interaction between geospatial domain experts and HPC resources is critical. Such tools should abstract the complexities of HPC systems and provide standardized interfaces for effectively accessing, utilizing, and managing High-Performance Geospatial Computing (HPGC) resources. This undertaking necessitates substantial research and development efforts, along with the generalization and standardization of various aspects related to HPGC resource definitions and processing interfaces.

The objective of this Testbed 19 task is twofold: To evaluate previous and ongoing efforts in applying HPC to geospatial analytics and to develop initial standards for HPGC resource definitions and processing interfaces. By examining existing work in the field, identifying best practices, challenges, and opportunities for enhancing the utilization of HPC in geospatial domains is possible. The goal is to establish standardized guidelines that will facilitate the seamless integration of HPGC resources into geospatial analytical workflows, ensuring their efficient and effective use across diverse application domains.

Achieving this goal will foster collaboration between geospatial domain experts and HPC specialists, enabling a more streamlined and accessible approach to large-scale geospatial analytics. This ER outlines task findings, recommendations, and initial standards thereby providing a foundation for future advancements in the field of High-Performance Geospatial Computing.

2

HIGH PERFORMANCE GEOSPATIAL COMPUTING

2 HIGH PERFORMANCE GEOSPATIAL COMPUTING

This section will review the current status of High Performance Geospatial Computing (HPGC)[4][7][10].

NOTE: Add any other clauses as needed

2.1. Definition of High-Performance Geospatial Computing

High Performance Computing (HPC) refers to the application of advanced computing technologies and techniques to solve computationally intensive problems that require a large amount of processing power, memory, and storage. HPC often involves the use of clusters or supercomputers made up of thousands of interconnected processors and storage devices.

High Performance Geospatial Computing (HPGC) refers to the use of advanced computing techniques, tools, and systems along with geospatial data to solve complex problems related to geography, environmental science, natural resources, national security, healthcare, and other areas that rely on geospatial data analysis. In short, HPGC refers to the use of high performance computing (HPC) resources to solve complex geospatial problems. HPGC utilizes parallel processing, distributed architectures, cloud computing, and high-speed networking to accelerate data processing, modeling, simulation, visualization, and analysis[4][10][13][15]. This enables users to process, analyze, and interpret geospatial data to gain insights, solve problems, and make informed decisions more efficiently and effectively. HPGC systems are typically used to process and analyze large volumes of geospatial data, such as satellite imagery, aerial photography, lidar data, or to run large simulation models in spatiotemporal domains.

Table 1 compares and summarizes major steps of HPC and HPGC.

Table 1 — Major steps in HPC and HPGC

STEP	HPC	HPGC
Problem formulation	Identify the problem that needs to be solved and formulate it in a way that can be transformed into a computationally intensive task including breaking the problem into smaller, more manageable sub-problems and defining the inputs, outputs, and constraints of the solution.	Identify the problem specifically in geospatial domains, such as mapping and charting (e.g., creation of large area maps), disaster response (e.g., monitoring and emergency response to natural disasters, such as floods, hurricanes, and wildfires), and environmental monitoring.

STEP	HPC	HPGC
Algorithm development	Develop an algorithm or set of algorithms that can efficiently solve the problem which includes designing a suitable computational model, selecting appropriate numerical methods, and optimizing the algorithm for parallel processing using techniques such as load balancing, data partitioning, and communication reduction.	Develop specialized algorithms for solving geospatial problems including designing a suitable geospatial computational model, geospatial analytics, and optimizing geo-computing with parallelism, such as geospatial data partition, spatial indexing, and spatial optimized computing.
Programming	Implement algorithms in code that can be run on an HPC system, which includes using specialized languages and libraries, such as MPI (Message Passing Interface), OpenMP (Open Multi-Processing), CUDA (Compute Unified Architecture), or OpenCL (Open Computing Language).	Implement geospatial algorithms which includes using geospatial libraries, such as Geospatial Data Abstraction Library (GDAL) and spatial projection library, and leveraging special clustering frameworks, such as Geo Spark[22][24].
Testing and debugging	Test and debug the code, which includes running the code on smaller test cases, comparing the results to analytical or experimental benchmarks, and identifying and fixing any errors or inefficiencies.	Test and debug the geospatial algorithms implemented which includes verifying the algorithms against geospatial theories and geostatistical approaches and verifying the results geospatially from ground truth data or other verified data.
Execution and monitoring	Execute the computational task, which includes monitoring the system to detect any abnormal behavior, diagnose and fix any issues that arise, and collect performance data for analysis and optimization.	Execute and monitor the geo-computing task. Includes monitoring the progress, diagnosing the geo-computing partitions and reduction, and observing the progress with geospatial visuals or maps.
Post-processing and visualization	Post-process and analyze the results to extract meaningful insights, which includes sorting, filtering, and aggregating large amounts of data, as well as visualizing the results using tools such as graphs, charts, or maps.	Post-process and analyze the geospatial results which includes scaling, spatiotemporal statistics, geostatistics, and visualization as interactive maps.

The main benefits of using HPGC are as follows.

- Speed: HPGC systems can process large datasets of geospatial data quickly and efficiently which can save time and money, and can also help organizations make better decisions faster.
- Accuracy: HPGC systems can be used to process geospatial data with greater accuracy than traditional methods, which can be important for applications such as climate modeling and earthquake prediction.

- Scalability: HPGC systems can be scaled to handle larger and more complex datasets which allows organizations to keep up with the ever-increasing volume and complexity of geospatial data.

The major challenges of using HPGC are as follows.

- Cost: HPGC systems can be expensive to purchase and maintain.
- Complexity: HPGC systems can be complex to set up and use.
- Expertise: Using HPGC systems effectively requires specialized expertise.

2.2. Key Application Drivers

The following lists some of the key applications driving the utilization of high performance geospatial computing.

- The increasing volume and complexity of geospatial data: The volume of geospatial data is exponentially growing and the data are becoming increasingly complex making the processing and analyzing of the data using traditional computing methods difficult.
- The need for real-time processing: In many cases, it is necessary to process and analyze geospatial data in real time, such as for traffic management, disaster response, and national security.
- The need for accurate and precise results: In many cases, obtaining accurate and precise results when processing and analyzing geospatial data is necessary, including applications for climate modeling, earthquake prediction, and wildfire prediction.

2.3. HPGC Frameworks

CyberGIS-Compute, a middleware framework to bridge high performance computing and end users, was reviewed. It provides the base for the development of standard HPGC middleware/services that achieve the same capabilities of CyberGIS-Compute, but in a more standards based environment, which adopts widely-accepted specifications.

CyberGIS-Compute supports HPC job management (execution and monitoring) and collaborative workflow orchestration. A more detailed review of the CyberGIS-Compute framework and its relationship to the standards development process is available in the Appendix — HPGC Frameworks.

2.4. Data-Intensive Geospatial Analytics for HPGC

HPGC can enable efficient processing of large volumes of data and data-intensive geospatial analytics. Examples of data-intensive geospatial analytics that can leverage HPGC include the following.

- **Spatial data clustering**: Clustering is a technique to group similar objects together based on their spatial properties. Spatial clustering can be used to identify patterns and trends in large geospatial datasets, such as urban planning or regional development.
- **Geospatial machine learning**: Machine learning algorithms, such as Random Forest[1], Support Vector Machine[2], and neural networks (including deep learning neural networks), can be applied to geospatial data to enable better prediction, classification, and regression analysis. For example, geospatial machine learning can be used in precision agriculture to predict crop yield, disease outbreaks, or identify soil types.
- **Geospatial image processing**: High-resolution satellite imagery can generate large volumes of data. Analyzing such volumes of data is computationally intensive. HPGC can be used to analyze large volumes of geospatial imagery data and extract meaningful information such as land use, land cover changes, urban growth, or natural resources.
- **Geospatial simulation modeling**: Simulation modeling involves the creation of mathematical models that emulate complex systems, such as traffic flow, pedestrian movement, water flow, or environmental simulation. Computational complexity often hinders simulation modeling using traditional computing techniques. HPGC can help in overcoming this challenge and help model complex systems affected by location.
- **High-Throughput Geospatial database management**: Managing geospatial data requires specialized tools and techniques. HPGC can be used to optimize geospatial database management systems, from data integration, data warehousing, data indexing, and querying. This can empower real-time geospatial analytics across various industries such as city planning, transportation planning, or environmental analysis.
- **Geospatial Optimization**: A typical geospatial optimization problem is routing, which uses spatial data such as traffic data, road network data, warehouse locations, and delivery routes to optimize the logistics and distribution of goods and services. Optimizing spatial configurations and resource allocation are computationally intensive. HPGC can be used to address computationally complex geospatial optimizations that require numerous iterations or combinatorial comparisons.

2.5. Standardization of HPGC-based Data-Intensive Geospatial Analytics

HPGC has the potential to enable advanced and efficient geospatial analytics. Standardizing and making HPGC geospatial analytic capabilities accessible to the broader geospatial community can enhance knowledge sharing and enable better decision-making across different domains. The HPGC-based geospatial analytics that can be standardized and made accessible to the broader geospatial community include the following.

- **Geospatial Data Processing Pipelines**: Standardizing the construction and execution of geospatial data processing pipelines can facilitate the integration and interoperability of HPGC workflows. Defining common data processing steps, input/output formats, and execution frameworks would enable users to easily exchange and share their geospatial processing pipelines across different HPGC platforms.
- **Large-Scale Spatial Data Analysis**: Standardizing the algorithms and methodologies for large-scale spatial data analysis tasks, such as spatial clustering, spatial interpolation, or network analysis, can promote consistency and comparability of results across different HPGC implementations, enabling researchers and practitioners to leverage shared algorithms and approaches, reducing duplication of efforts, and fostering collaboration.
- **Geospatial Machine Learning Models**: Standardizing the development and deployment of geospatial machine learning models can enhance reproducibility and interoperability. This includes standardizing the representation and serialization of trained models, input/output data formats, and evaluation metrics. Standardized geospatial machine learning models would allow users to easily share, validate, and integrate models into their HPGC workflows.
- **Geospatial Simulation and Modeling Frameworks**: Standardizing the frameworks and interfaces for geospatial simulation and modeling can promote the exchange and integration of different simulation models. By defining standard interfaces, input/output formats, and simulation control mechanisms, researchers and practitioners can more easily collaborate, validate, and reuse geospatial simulation and modeling components.
- **Geospatial Image Analysis Workflows**: Standardizing geospatial image analysis workflows can improve the accessibility and reproducibility of image processing and analysis tasks. Defining common data formats, preprocessing steps, feature extraction methods, and quality assessment metrics can simplify the adoption and sharing of HPGC-based geospatial image analysis workflows.
- **Spatial Data Fusion and Integration Techniques**: Standardizing the methods and workflows for spatial data fusion and integration can enable the seamless integration of data from multiple sources. This includes standardizing data formats, fusion algorithms, data alignment techniques, and uncertainty modeling approaches. Standardized geospatial data fusion and integration techniques would facilitate data sharing and interoperability between different HPGC systems.

- **Geospatial Optimization Models and Solvers**: Standardizing the formulation and solution approaches for geospatial optimization problems can promote the adoption and exchange of optimization models and solvers. Defining standard problem representations, optimization algorithm interfaces, and solution result formats would enable users to easily apply and integrate HPGC-based geospatial optimization techniques into their workflows.

3

HPGC API

3 HPGC API

NOTE: This section is for the implementation of high performance geospatial computing API by GeoLabs.

3.1. HPGC API

This Chapter covers the HPGC API. The initial API is developed based on the OGC API — Processes Standard[14].

3.1.1. API — Processes

In Testbed 19, OGC API — Processes was adopted as the base API for algorithm management, job scheduling, job monitoring, and workflow orchestration for HPGC.

3.1.2. HPGC Profile of API — Processes

Common geospatial processing frameworks, geospatial algorithms, clustering frameworks, and typical workflow orchestrations can be implemented as processes or resource-based composite process to be managed through an API — Processes implementation instance. The processes were designed to support similar functions of the CyberGIS-Compute, but generalized to be usable with other high performance computing frameworks.

3.2. HPGC API Client

The draft HPGC API is implemented as a profile of API — Processes Standard that is based on OpenAPI technology. The Processes API Core does not mandate any encoding or format for the formal definition of the API. The OpenAPI 3.0 specification is one option for defining the Processing API. As such a conformance class is specified for OpenAPI 3.0, which depends on the requirements class Core. While the use of OpenAPI 3.0 for the formal definition of the Processes API is not mandatory, the requests/responses of the Processes API specified are defined using OpenAPI 3.0 schemas. With OpenAPI technology, it is possible to generate clients in different languages using OpenAPI Generator, including C++, Java, Go, Python, and JavaScript. For Testbed 19, the Python client SDK was implemented with the assistance of automatic client generation against the OpenAPI specification of the draft HPGC API.

3.3. ZOO-Project with HPC support implementation

GeoLabs reused its experience in accessing and using the HPC environment and the support available in the ZOO-Project. The ZOO-Project is an Open Source Reference Implementation of the OGC API — Processes — Part 1: Core and the OGC Web Processing Service (WPS) Standard 2.0.0 released under an MIT/X11 license.

In the past few years, GeoLabs developed support for the OGC API — Processes — Part 2: Deploy, Replace, Undeploy draft specification as part of the ZOO-Project. This provides the capability to deploy CWL packaged application in a Kubernetes cluster. The implementation follows the OGC Best Practice for Earth Observation Application Package. Moving the processing to the data rather than the data to the processing engine is now possible.

By combining support for the remote HPGC job scheduling with the one for the OGC API — Processes — Part 2: Deploy, Replace, Undeploy draft specification, an end-to-end solution for ease of interactions with HPGC is proposed. This solution partly covers the CyberGIS-Compute capabilities.

The source code for the HPGC was published in the official ZOO-Project GitHub repository. The Binary docker image is published on dockerhub and the corresponding Helm Chart on artifacthub to deploy the solution on a kubernetes cluster.

3.3.1. Security consideration

When considering the client application for deploying and executing tasks on an HPC platform, which are limited resources allocated to the project, the system will require authentication to ensure that only authorized users can create and run new process resources. The security mechanism available in the ZOO-Project as a filter_in process is used. The concept of filter_in and filter_out was introduced during the implementation of the OGC API — Processes — Part 2: Deploy, Replace, Undeploy draft specification. These security filters, like any other ZOO-Kernel process, run for every received request. Filter_in processes run before the request is processed, while filter_out processes run after.

Figure 1 illustrates the implementation of authentication as a web-process. By providing such processes, the ability of a developer to change the default ZOO-Kernel behavior for any request is provided. For example, one can decide to implement a service that verifies that an authenticated user is authorized or not to access a given OGC API endpoint.

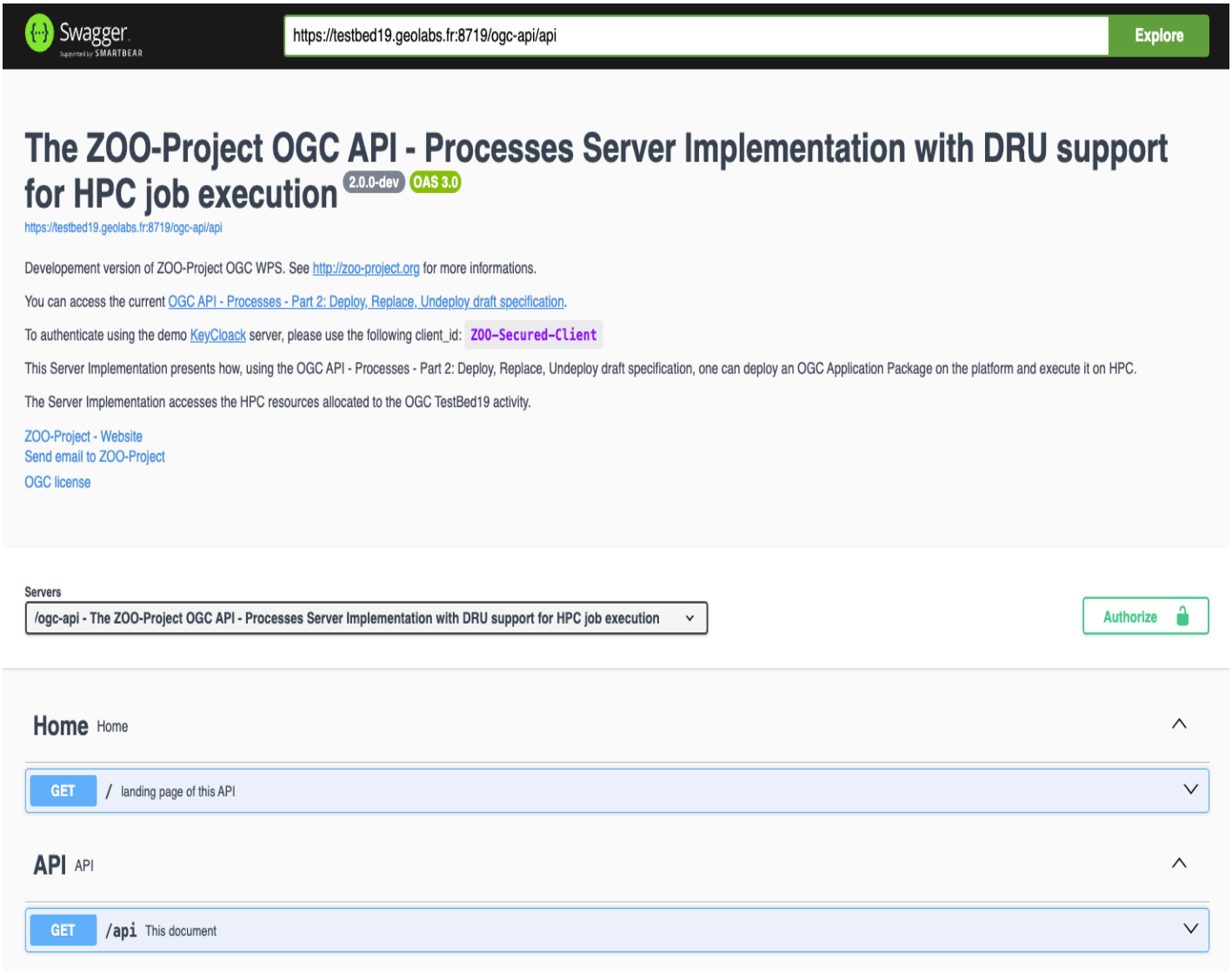


Figure 1 — Illustration of authentication implementation based on the OpenID Connect (OIDC)

The Keycloack software for authentication, which offers a state-of-the-art solution for setting up an OpenID Connect provider, was used. With it, a dedicated "OGC_TESTBED19_SECURED_AREA" realm was created for which both GitHub and GitLab (using gitlab.ogc.org) were configured as identity providers.

A dedicated ***authorized_users*** parameter in the security section to store the comma-separated list of users was added to allow access to the secured OGC API endpoints.

In addition to using Keycloack, another prototype server instance that uses the Authenix server to authenticate users was successfully implemented which is a closed-source solution providing a compliant OpenID Connect Authorization Server with federated SAML2-based login from Google, Facebook, and eduGAIN. By doing so, users can authenticate using their OGC portal credentials.

3.3.2. DeployProcess

In the OGC API — Processes — Part 2: Deploy, Replace, Undeploy draft specification, the ***Deploy*** operation allows an authorized user to deploy a new process on the processing server. The draft specification describes that the server may implement support for deploying an OGC Application Package (using the encoding ***application/ogcapppkg+json***). An OGC Application Package is a process package described using the OGC Application Package information model. The application package comprises a JSON object with two parameters: a ***processDescription*** and an ***executionUnit***. The ***processDescription*** conforms to the ***processes.yaml*** schema. The process description part corresponds to what is obtained as a response for a GET request to the ***/processes/{processId}*** path. On the other hand, the ***executionUnit*** conforms to the ***executionUnit.yaml*** schema. The execution unit part defines what should be deployed on the processing server.

In the case of HPC, only Secure Shell Protocol (SSH) access to the HPC instance is made available. Consequently, deploying a new process on the processing server will require storing the associated metadata information and the singularity container [5] to run on the HPC in case the container for the process is unavailable. A container consists of an entire runtime environment: an application, plus all its dependencies, libraries, and configuration files bundled into one package [5]. Singularity is a tool for running software containers on HPC systems, similar to Docker [5]. Singularity containers are common and well supported by HPC systems [5].

To store the metadata information associated with the new process, a dedicated ***filter_in*** process, named ***securityIn***, run by the ZOO-Kernel on every request was implemented. For deployment in an HPC environment, a dedicated service called ***DeployOnHpc*** is invoked through SSH to create a singularity container using the image provided in the ***executionUnit***.

Suppose the ***filter_in*** process (***securityIn***) detects a Deploy operation using the application/ogcapppkg+json encoding. In that case, the process then parses the metadata information from the ***processDescription*** and invokes the execution of the ***DeployOnHpc*** process asynchronously by sending an Advanced Message Queuing Protocol (AMQP) message to the ZOO-FPM.

The sequence diagram below illustrates the method of deploying a new process using the prototype server implementation.

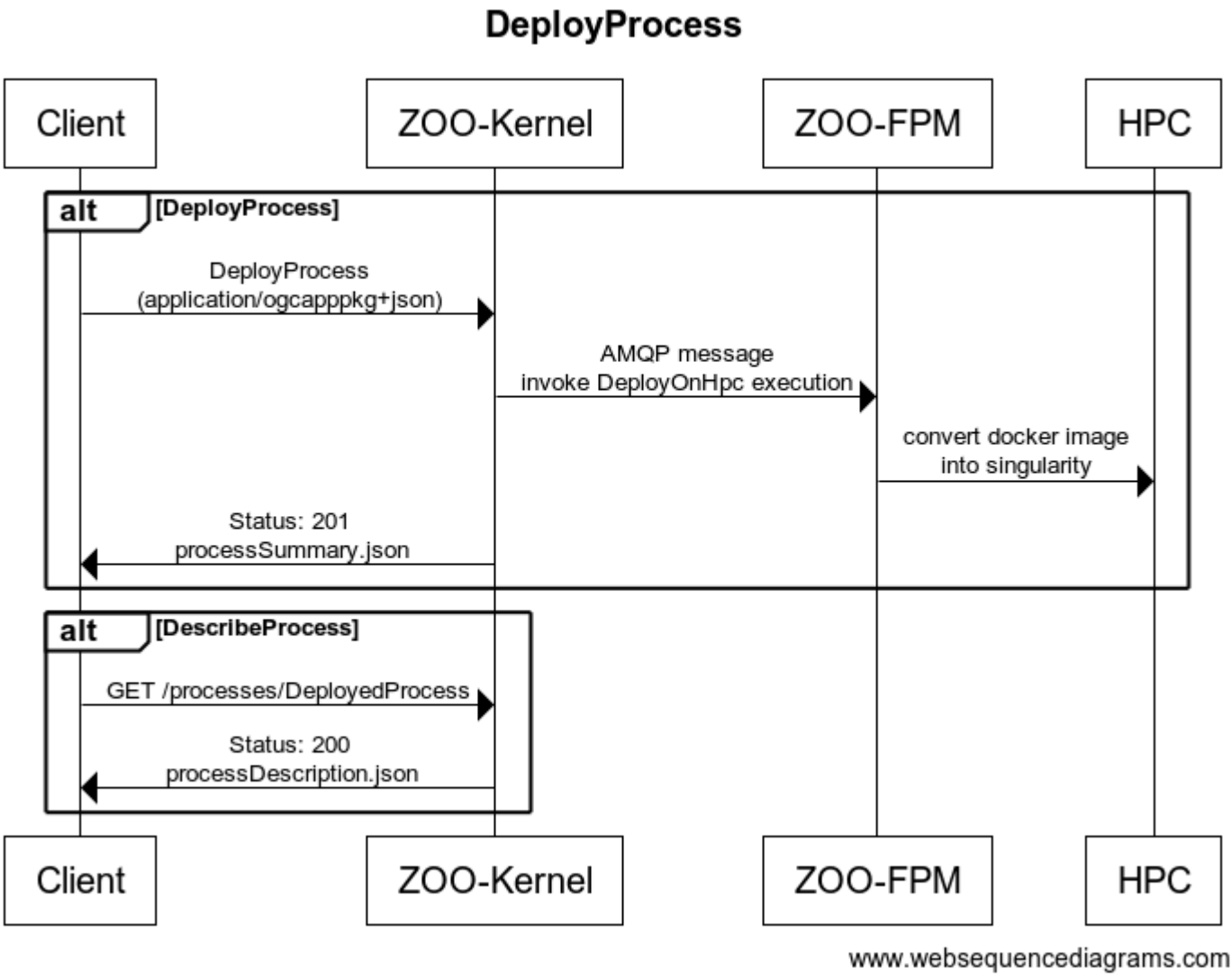


Figure 2 — Sequence diagram ZOO-Project Deploy Operation

Figure 3 illustrates the example of the API for deploying a process.

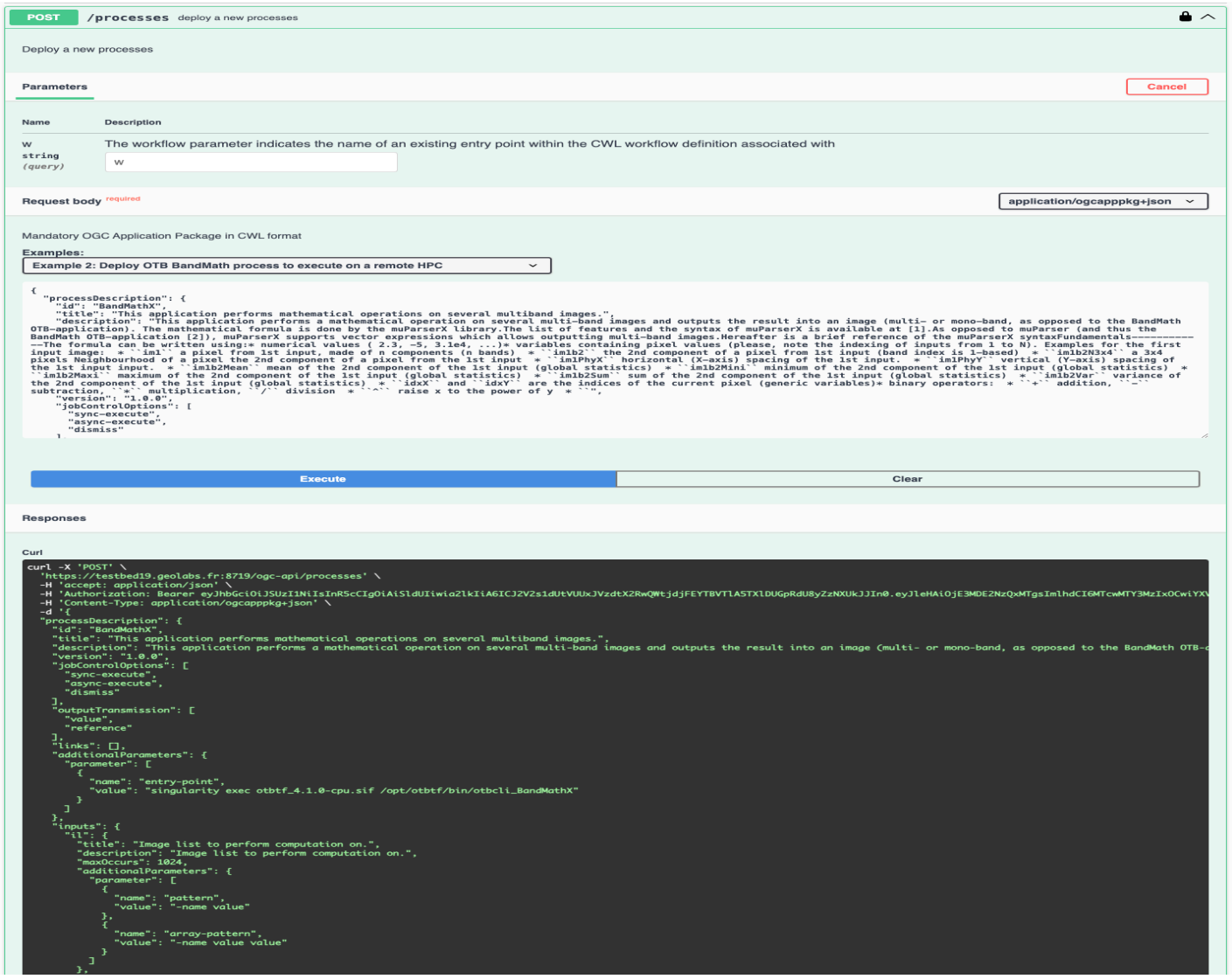


Figure 3 — Illustration of API implementation for deploying a process

The Replace operation works the same way. If version management is unavailable for the deployed process, implementing the process as another ***filter_in*** process is possible. Version management would require clearly defining the expected numbering scheme and how to handle different versions of the same processes.

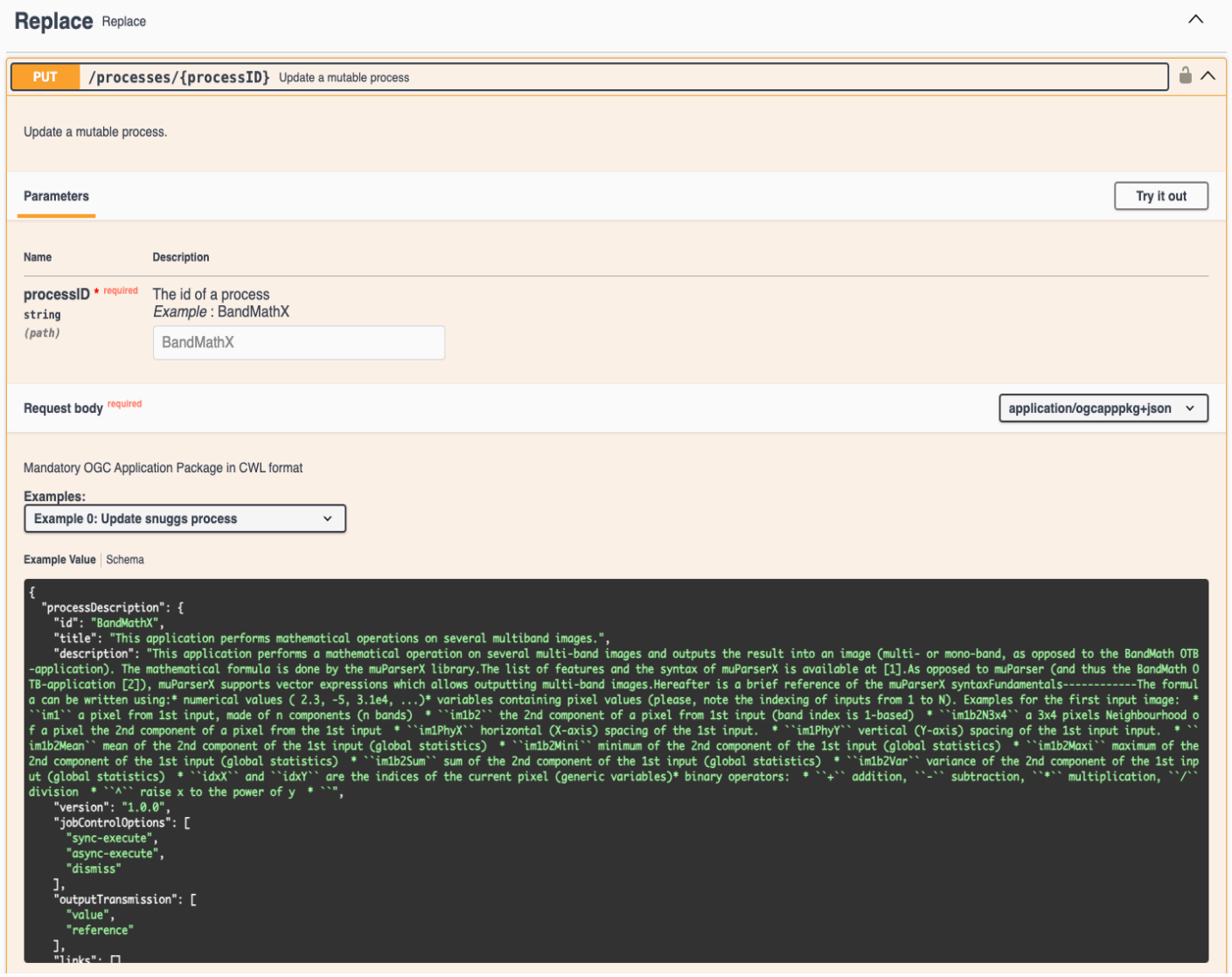


Figure 4 — Illustration of API implementation for replacing an already deployed process using ***processID***

Figure 5 illustrates the HPGC API “Undeploy” method for deleting an already deployed process. The Undeploy operation is comparatively more complicated to handle properly as multiple processes can rely on the same singularity container. This leads to no one-to-one relationship between the process and the singularity container. Consequently, the Undeploy operation is only responsible for removing the metadata information associated with a process, not the singularity container. Note that it would be possible to correctly implement the singularity container removal from the HPC server by keeping track of the one-to-many relationship between the singularity container and the processes. Once no processes are associated with a singularity container, the container can be removed.

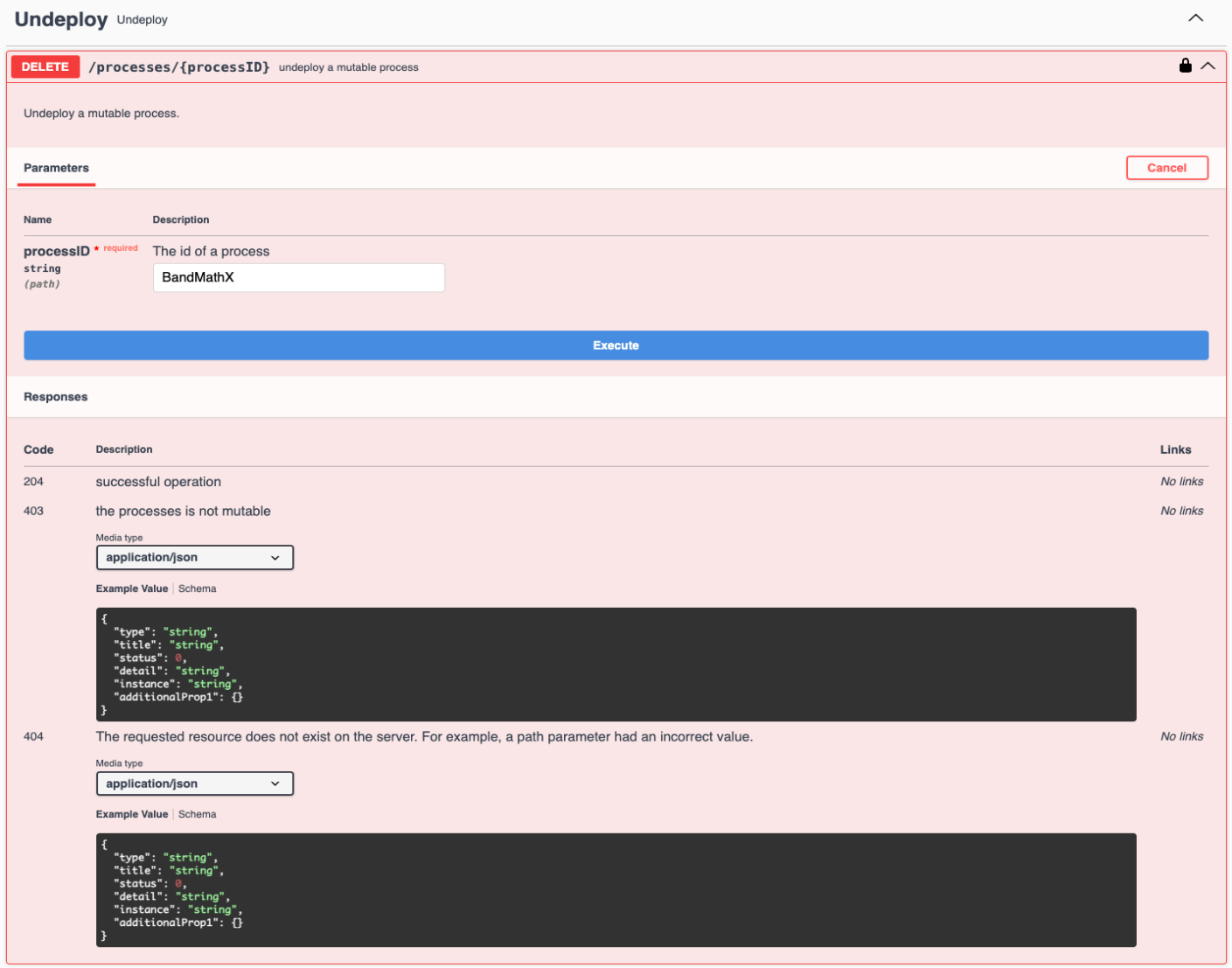


Figure 5 — Illustration of API implementation for undeploying (deleting) an already deployed process using ***processID***

Also, when considering a singularity container from the HPC, an application should pay attention to all the users with a deployed process associated with a given singularity container. Indeed, in the case where two users deploy the same process, each user will do so in their namespace. Nevertheless, on the HPC, only one singularity container is deployed.

While having a dedicated namespace on the HPC may sound good, such an approach also implies that deploying the same application by two different users will require twice deploying the same singularity container. This is why the Testbed task team decided not to use a namespace on the HPC.

3.3.2.1. Additional Parameters for process binding

The ZOO-Project implementation relies on dedicated additional parameters embedded within the ***processDescription*** provided during the Deploy operation, which can be added at every level in the process description, meaning the parameters can be at the root, input, and output levels.

At the root level, finding an additional parameter named ***entry-point*** is expected. The parameter contains the value to refer to the prefix of the command to execute on the HPC, i.e. ***singularity exec otbtf_4.1.0-cpu.sif /opt/otbtf/bin/otbcli_BandMath***.

At the input and output level, finding an additional parameter named ***pattern*** is expected. The value of this parameter defines the way to add the parameter to the command to run, i.e. ***-name value***. This parameter applies to every input and output.

For multi-valued inputs, finding another parameter named ***array-pattern*** is expected. This parameter provides the syntax to use if a client application uses multiple values for a single input.

The ZOO-Kernel can produce an sbatch file containing the desired command to run on the HPC using the additional parameters defined in this section. “sbatch” is used to submit a batch script to Slurm. Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.

Below is a process description from an HPGC API Deploy example followed by a corresponding sbatch produced at execution time. This example illustrates how Orfeo ToolBox (OTB)[8] applications can be deployed and executed.

```
{
  "processDescription": {
    "id": "BandMathX",
    "title": "This application performs mathematical operations on several
multiband images.",
    "description": "This application performs a mathematical operation on
several multi-band images and outputs the result into an image (multi- or
mono-band, as opposed to the BandMath OTB-application). The mathematical
formula is done by the muParserX library.The list of features and the syntax
of muParserX is available at [1].As opposed to muParser (and thus the BandMath
OTB-application [2]), muParserX supports vector expressions which allows
outputting multi-band images.Hereafter is a brief reference of the muParserX
syntaxFundamentals------------The formula can be written using:* numerical
values ( 2.3, -5, 3.1e4, ...)* variables containing pixel values (please,
note the indexing of inputs from 1 to N). Examples for the first input
image:  * ''im1'' a pixel from 1st input, made of n components (n bands)  *
''im1b2'' the 2nd component of a pixel from 1st input (band index is 1-based)
 * ''im1b2N3x4'' a 3x4 pixels Neighbourhood of a pixel the 2nd component
of a pixel from the 1st input  * ''im1PhyX'' horizontal (X-axis) spacing
of the 1st input.  * ''im1PhyY'' vertical (Y-axis) spacing of the 1st input
input.  * ''im1b2Mean'' mean of the 2nd component of the 1st input (global
statistics)  * ''im1b2Mini'' minimum of the 2nd component of the 1st input
(global statistics)  * ''im1b2Maxi'' maximum of the 2nd component of the 1st
input (global statistics)  * ''im1b2Sum'' sum of the 2nd component of the 1st
input (global statistics)  * ''im1b2Var'' variance of the 2nd component of the
1st input (global statistics)  * ''idxX'' and ''idxY'' are the indices of the
current pixel (generic variables)* binary operators:  * ''+'' addition, ''-
```

```
'' subtraction, ''*'' multiplication, ''/'' division  * ''^'' raise x to the
power of y  * ''",
    "version": "1.0.0",
    "jobControlOptions": [
      "sync-execute",
      "async-execute",
      "dismiss"
    ],
    "outputTransmission": [
      "value",
      "reference"
    ],
    "links": [],
    "additionalParameters": {
      "parameter": [
        {
          "name": "entry-point",
          "value": "singularity exec otbtf_4.1.0-cpu.sif /opt/otbtf/bin/otbcli_
BandMathX"
        }
      ]
    },
    "inputs": {
      "il": {
        "title": "Image list to perform computation on.",
        "description": "Image list to perform computation on.",
        "maxOccurs": 1024,
        "additionalParameters": {
          "parameter": [
            {
              "name": "pattern",
              "value": "-name value"
            },
            {
              "name": "array-pattern",
              "value": "-name value value"
            }
          ]
        },
        "schema": {
          "oneOf": [
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/tiff"
            },
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/jpeg"
            },
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/png"
            }
          ]
        }
      },
      "out": {
        "title": "Output image.",
        "description": "Output image.",
        "additionalParameters": {
```

```
        "parameter": [
          {
            "name": "pattern",
            "value": "None"
          }
        ]
      },
      "schema": {
        "type": "string",
        "default": "float",
        "enum": [
          "uint8",
          "uint16",
          "int16",
          "int32",
          "int32",
          "float",
          "double"
        ]
      }
    },
    "exp": {
      "title": "Mathematical expression to apply.",
      "description": "Mathematical expression to apply.",
      "additionalParameters": {
        "parameter": [
          {
            "name": "pattern",
            "value": "-name \"value\""
          }
        ]
      },
      "schema": {
        "type": "string"
      }
    },
    "ram": {
      "title": "Available memory for processing (in MB).",
      "description": "Available memory for processing (in MB).",
      "additionalParameters": {
        "parameter": [
          {
            "name": "pattern",
            "value": "-name value"
          }
        ]
      },
      "schema": {
        "type": "integer",
        "default": 256,
        "nullable": true
      }
    }
  },
  "outputs": {
    "out": {
      "title": "Output image.",
      "description": "Output image.",
      "additionalParameters": {
        "parameter": [
          {
            "name": "pattern",
            "value": "-name value inputs_out_value"
```

```
            }
          ]
        },
        "schema": {
          "oneOf": [
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/tiff"
            },
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/jpeg"
            },
            {
              "type": "string",
              "contentEncoding": "base64",
              "contentMediaType": "image/png"
            }
          ]
        }
      }
    }
  },
  "executionUnit": {
    "type": "SLURM",
    "image": "docker://mdl4eo/otbtf:4.1.0-cpu"
  }
}
```

In this process description, the additional parameter ***pattern*** has an empty ***value*** for the ***out*** input, meaning that the result of the rewriting will be an empty string, whatever the value of the ***out*** input is. On the other hand, from the ***out*** output the ***pattern*** has the value ***-name value inputs[out]*** meaning that the value provided for the ***out*** input from the output ***pattern*** can be referenced.

SLURM is used as a type in the execution unit to distinguish HPC processes from other possible deployment supported.

```
#!/bin/bash

### *** Default ZOO-Service HEADER (no header found) *** ###

#SBATCH --job-name=ZOO-Project_9c21a7d2-6c2f-11ee-98d0-0242c0a8e008_BandMath

### *** Default ZOO-Service BODY (no body found) *** ###
echo "Job started at: $(date)"
echo "Running service: [BandMath]"
singularity exec otbtf_4.1.0-cpu.sif /opt/otbtf/bin/otbcli_BandMath \
     -il /home/x-gfenoy/wps_executions/data//6a15208156214f03ad83aa1dd991a699.
zca\
     -exp "im1b1+im1b2" \
     -ram "256" \
     -out /home/x-gfenoy/wps_executions/data/output_BandMath_out_9c21a7d2-6c2f-
11ee-98d0-0242c0a8e008.tiff float
echo "Job finished at: $(date)"
### *** Default ZOO-Service FOOTER (no footer found) *** ###
```

The ZOO-Project implementation supports three templates for producing the sbatch file: header, body, and footer. The ZOO-Project potentially uses these templates when creating the

sbatch. The generation of this sample sbatch did not use any template. The ZOO-Project also supports the definition of SBATCH parameters inserted in the generated file before the ***job-name*** definition.

3.3.3. Jobs management

OGC API — Processes — Part 1: Core Standard was used to execute processes on HPC providing a way to make the deployed processes executable remotely.

On HPC, Slurm traditionally handles job scheduling. Slurm can manage tasks provided in sbatch file format. Sbatch is similar to bash scripts but has specific options and parameters which means that a client and/or application can feed Slurm with pre-cooked recipes containing all the commands to execute.

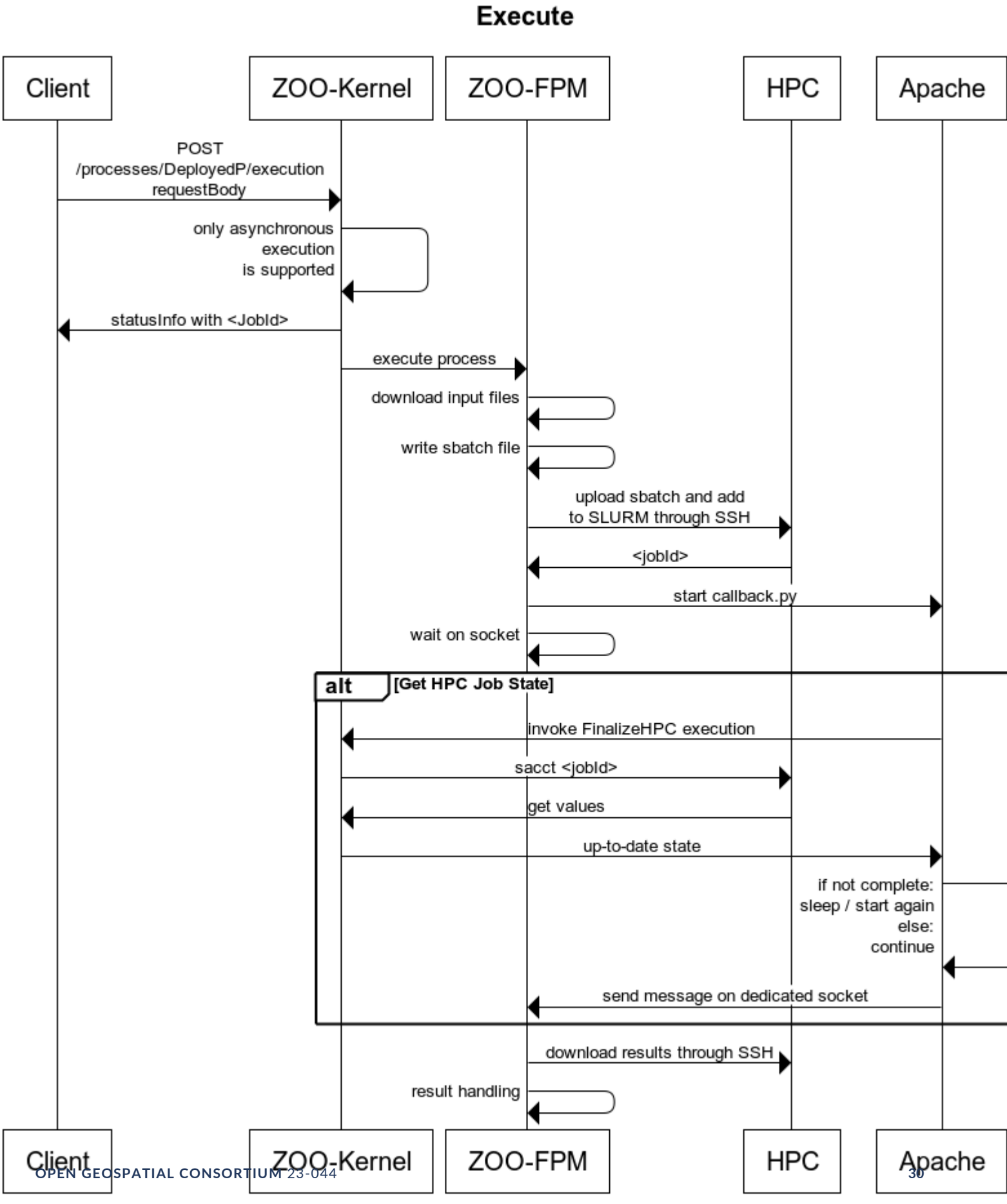
To produce this sbatch file, the ZOO-Kernel combines metadata information provided within the ***processDescription*** used to deploy the process with the payload content used to invoke its remote execution.

As explained above, specific additional parameters embedded within the ***processDescription*** metadata were defined to ease generation of the desired sbatch file. Also, the ZOO-Kernel receives internal support for downloading and setting the parameters in such sbatch files for HPC applications from a WPS request. During the testbed 19 period, the HPC support was updated to handle OGC API — Processes — Part 1: Core standard requests. Consequently, one authorized user can execute a process and create a job on the Processing Engine corresponding to the job scheduled on the HPC.

Once an HPC runs the sbatch command, the ZOO-FPM responsible for handling the job execution holds the jobid generated by Slurm, invokes the internal callback.py CGI script, opens a socket in listen mode, and waits for a message. The callback.py CGI script invokes the execution of the ***FinalizeHPC*** process, which connects to the HPC through SSH to get the state of the job. SLURM only gives basic state information such as Pending, Running, Completed, Failed, etc. The ***FinalizeHPC*** process uses this information to detect the end of an execution. In such a case, it sends a message to the ZOO-FPM waiting for the end of the job. The ZOO-FPM then follows the traditional way of handling produced results and potential failure in non-HPGC environments.

The HPC application hosts the results or error messages, which must first be downloaded by the ZOO-FPM. If the ZOO-FPM could download the result, the workflow continues with potential publication of the results using the MapServer[11] library to produce the desired mapfile for providing access to the relevant OGC Web Services, such as a Web Map Service (WMS), Web Feature Service (WFS) or Web Coverage Service(WCS). MapServer is an Open Source platform for publishing spatial data and interactive mapping applications to the web[11]. In other cases, ZOO-FPM downloads the error log files and returns the exception details.

The sequence diagram below shows how the ZOO-Project handles the execution of tasks on remote HPC.


Execute
Client
ZOO-Kernel
ZOO-FPM
HPC
Apache
POST
/processes/DeployedP/execution
requestBody
only asynchronous
execution
is supported
statusInfo with <JobId>
execute process
download input files
write sbatch file
upload sbatch and add
to SLURM through SSH
<jobId>
start callback.py
wait on socket
alt
[Get HPC Job State]
invoke FinalizeHPC execution
sacct <jobId>
get values
up-to-date state
if not complete:
sleep / start again
else:
continue
send message on dedicated socket
download results through SSH
result handling
Client
ZOO-Kernel
ZOO-FPM
HPC
Apache
www.websequencediagram

3.3.3.1. Limitation

The Dismiss requirement class defines how client applications can cancel a job execution. SLURM offers tools to cancel a job execution, which was not considered during Testbed 19.

4

HPGC API CLIENT LIBRARY

4 HPGC API CLIENT LIBRARY

This Chapter describes development of HPGC API client libraries for different programming languages. Testbed 19 focused on developing a client library for Python.

NOTE: This section describes one of the HPGC API clients for Python. This task was led by https://csiss.gmu.edu [George Mason University].

4.1. Introduction

The OGC API — Processes Standard defines requirements for describing and executing geospatial processes [16]. The OpenAPI Generator [12] is a tool that can be used to generate client libraries for interacting with APIs that are described using OpenAPI specifications.

This Section describes how to generate an API — Processes Python client library using the OpenAPI Generator. This Section also describes the major functions of the generated client library and provides example snippets of how to use the client library in Python.

4.2. Generating the Python Client Library

To generate the Python client library, the OpenAPI Generator must be installed. The OpenAPI Generator **generates code from an OpenAPI specification**. The Generator can create code for client libraries, server stubs, documentation, and configuration. It supports various languages and frameworks.

The following are several options to install and run the OpenAPI Generator.

- Install the OpenAPI generator using pip:

The following command installs the openapi-generator in Python.

```
pip install openapi-generator
openapi-generator generate -i ogcapi-processes.bundled2.json -g python
```

- Download the Java Archive (JAR): The OpenAPI Generator is a Java program. The compiled JAR can be downloaded and directly used. The following example script downloads the jar and uses the Generator in a Linux environment.

```
mkdir -p ~/bin/openapitools
curl https://raw.githubusercontent.com/OpenAPITools/openapi-generator/master/
bin/utils/openapi-generator-cli.sh > ~/bin/openapitools/openapi-generator-cli
chmod u+x ~/bin/openapitools/openapi-generator-cli
export PATH=$PATH:~/bin/openapitools/
```

```
openapi-generator-cli generate -i ogcapi-processes.bundled2.json -g python
```

- Compile from source code: The OpenAPI Generator can be compiled directly from the source code. The current version (7.0.1) requires Java 11 and Apache Maven 3.3.4 or greater. The following shows example steps for compiling the program in a Linux environment.

```
$ git clone https://github.com/OpenAPITools/openapi-generator.git
$ cd openapi-generaotr
$ ./mvnw clean install
$ java -jar modules/openapi-generator-cli/target/openapi-generator-cli.jar
generate \
    -g python -i http://geolabs.fr/dl/ogcapi-processes.bundled2.json
    -o /local/wksp/python
```

- Dokcer run: The OpenAPI Generator image can be directly used as a standalone executable with Docker. The following is an example run in a Linux environment.

```
$ docker pull openapitools/openapi-generator-cli:latest
$ docker run --rm -v "${PWD}:/local" openapitools/openapi-generator-cli
generate \
    -i http://geolabs.fr/dl/ogcapi-processes.bundled2.json \
    -g pyton \
    -o /local/wksp/python
```

In Testbed 19, two API — Process instances (OpenAPI documents in JSON) were used to generate models and stubs for the Python Client. These two documents contain definitions and components for all parts of the OGC API — Processes Standard. One is the example OpenAPI specification bundle available in the github of OGC API — Processes. Another is the OpenAPI specification bundle used by the ZOO-Project. The generated models are collectively used in developing the client library to support most extensions.

The OGC API — Processes are as follows.

- ZooWPS: http://geolabs.fr/dl/ogcapi-processes.bundled2.json
- API — Processes example bundle: https://raw.githubusercontent.com/opengeospatial/ogcapi-processes/master/openapi/ogcapi-processes.bundled.json

Generated stubs and models contained errors which were fixed manually. The specification documents contain errors in schema definitions, examples, and default values. The errors in the specification documents were required to be fixed. Iterations of re-running the Generator were done with stepwise correction of the specification documents.

Another group of errors resulted from the Generator. The OpenAPI Generator does not completely support all legit schema definitions. In the case of OGC API — Processes specifications, handling of AllOf, AnyOf, oneOf, and nested properties was problematic. The generated models ended up as incomplete models. These models were manually revised.

The Figure 7 shows the workflow to generate the Python client for HPGC API using the OpenAPI specification bundle documents of API — Processes. The Figure 8 shows the major classes generated for the HPGC Python Client library.

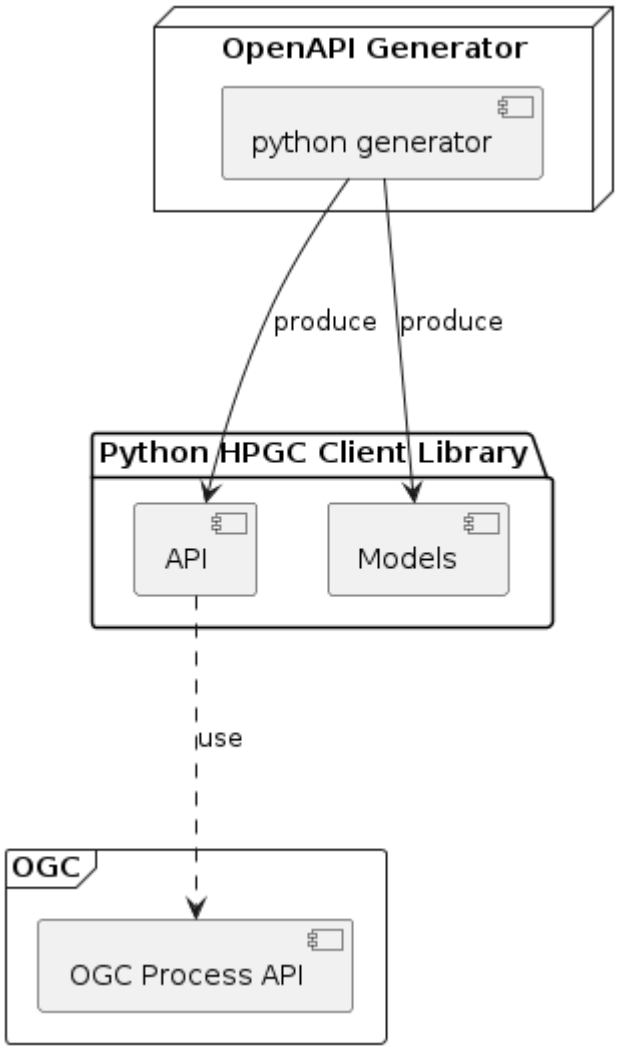


Figure 7 — The workflow to generate the HGPC API Python Client

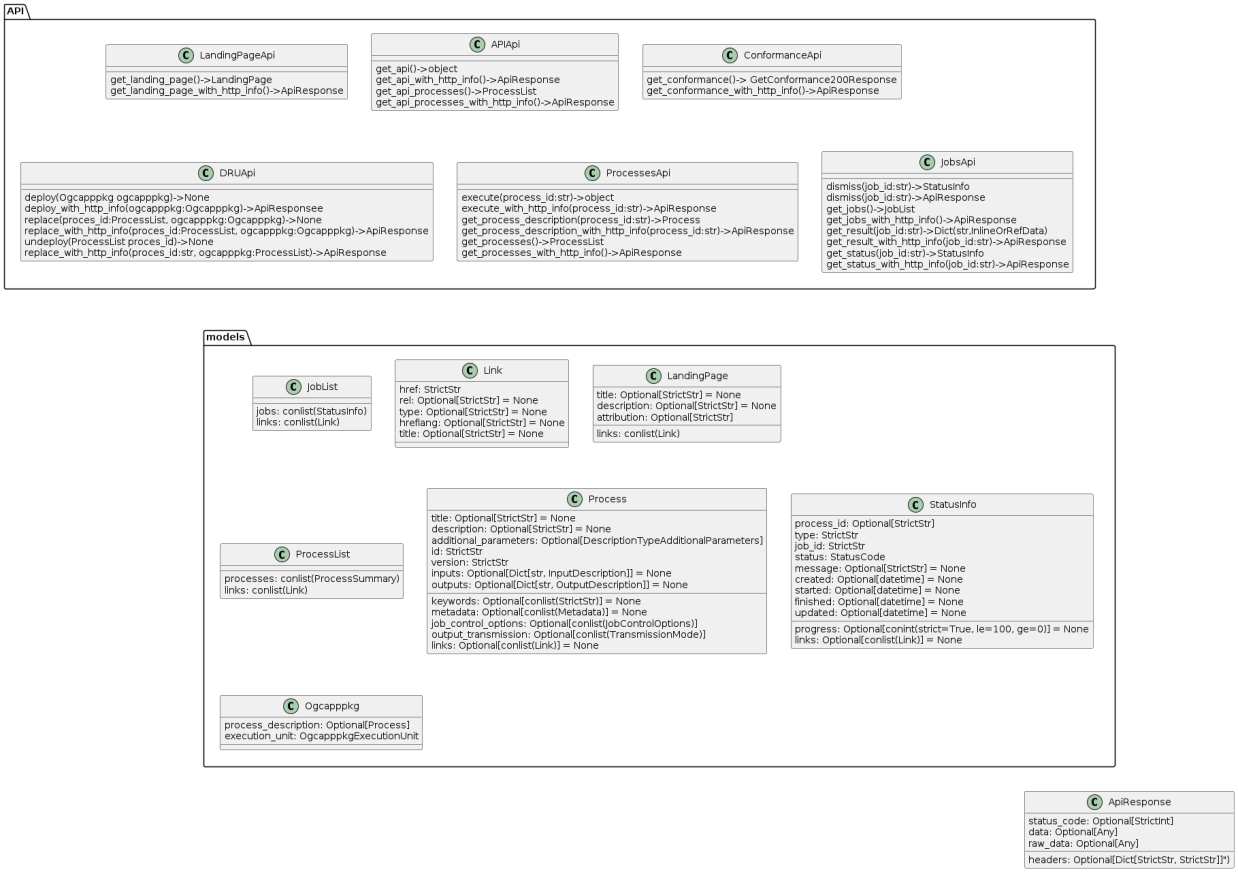


Figure 8 — Major classes for HGPC API

4.3. Major Functions of the Client Library

The Python client library provides a number of functions for interacting with the servers that implement the OGC API — Processes Standard. Some of the major functions are as follows.

- *get_landing_page()* or *get_landing_page_with_http_info()*: This function returns the landing page of the HPGC API. The landing page provides links to the API definition, the conformance declaration and the metadata about the processes offered by this service.
- *get_api()* or *get_api_with_http_info()*: This function retrieves the API definition which can be in YAML, JSON, or rendered HTML page.
- *get_conformance()* or *get_conformance_with_http_info()*: This function retrieves the set of OGC API — Processes conformance classes that are supported by this service.
- *get_processes()* or *get_processes_with_http_info()*: This function returns a list of all the processes that are available on the server. The list of processes contains a summary of each process the OGC API — Processes implementation offers, including the link to a more

detailed description of the process. For more information, see Section 7.7 of the API — Processes Standard[16].

- *get_process_description(process_id)* or *get_process_description_with_http_info(process_id)*: This function returns information about a specific process. The process description contains information about inputs and outputs and a link to the execution-endpoint for the process. The Core does not mandate the use of a specific process description to specify the interface of a process. That said, the API — Processes Core requirements class makes the following recommendation: Implementations SHOULD consider supporting the OGC process description. For more information, see Section 7.8 of the API — Processes Standard[16].

- *execute(process_id)* or *execute_with_http_info(process_id)*: This function starts a process. It executes a process (this may result in the creation of a job resource e.g., for *asynchronous execution*). The job id should be included in the response of the execution that will be used in monitoring progress status and obtaining results if the process is successfully executed. For more information, see Section 7.9 of the API — Processes Standard[16].

- *get_jobs()* or *get_jobs_with_http_info()*: This function returns the list of available jobs. For more information, see Section 12 of the API — Processes Standard[16].

- *get_status(job_id)* or *get_status_with_http_info(job_id)*: This function returns the status of an executed process job. For more information, see Section 7.10 of the API — Processes Standard[16].

- *dismiss(job_id)* or *dismiss_with_http_info(job_id)*: This function cancels a job execution and removes it from the jobs list. For more information, see Section 14 of the API — Processes Standard.

- *get_result(job_id)* or *get_result_with_http_info(job_id)*: This function returns the results of an executed process job. The response lists available results of a job. In case of a failure, the response instead lists exceptions. For more information, see Section 7.11 of the API — Processes Standard[16].

- *deploy(ogcapppkg)* or *deploy_with_http_info(ogcapppkg)*: This function deploys a process. For more information, see Section 6.3 of the API — Processes Part 2[17].

- *replace(process_id, ogcapppkg)* or *replace_with_http_info(process_id,ogcapppkg)*: This function replaces a process. For more information, see Section 6.4 of the API — Processes Part 2[17].

- *undeploy(process_id)* or *undeploy_with_http_info(process_id)*: This function undeploys a process. For more information, see Section 6.5 of the API — Processes Part 2[17].

4.4. Using the Client Library in Python

To use the Python HPGC API client library, the Python program needs to import the library. The following code will perform the import:

```
import openapi_client
```

Once the client library is imported, the functions described above can be used to interact with instance servers that implement the OGC API — Processes Standard. For example, the following code shows how to get a list of all the processes that are available on the server:

```
the_api = openapi_client.api.processes_api.ProcessesApi()
processes = the_api.get_processes()

for process in processes:
    print(process.name)
```

The following code shows how to start a process:

```
the_api = openapi_client.api.processes_api.ProcessesApi()
process_id = the_api.execute("my_process_id")

print(process_id)
```

The following code shows how to get the status of a process:

```
the_api = openapi_client.api.jobs_api.JobsApi()
process_status = the_api.get_status("my_job_id")

print(process_status)
```

The following code shows how to get the results of a process:

```
the_api = openapi_client.api.jobs_api.JobsApi()
process_results = the_api.get_result("my_job_id")

print(process_results)
```

4.5. Limitations of the HPGC API Client Library

The HPGC API Client Library was developed using the OpenAPI Generator. The Generator is a powerful tool that can be used to generate client libraries for interacting with APIs that are described using OpenAPI. This Testbed 19 ER report describes how to use the OpenAPI Generator to generate a Python client library for the OGC API — Processes Standard. This ER also describes some of the major functions of the client library and provided snippets of how to use the library in Python.

Below are some limitations of the automatic code generation for OGC API — Processes Python client, with possible solutions or resolving strategies for each limitation.

- **Incomplete or idiomatic code**: The generated code may not be complete or may be idiomatic Python. For example, the generated code may not handle all the possible error

conditions that an API can return. Further, the generated code may use Python idioms that are not in common use in the Python community. Possible solutions are as follows.

  - Use a tool such as Typeguard[19] or Pydantic[21] to validate the Python client code. These tools can help in validating and generating more idiomatic and complete Python code.

  - Review the generated code and make any necessary changes. This is especially important for handling error conditions and using Python idioms.

- **Compatibility with Python versions**: The generated code may not always be compatible with the latest versions of Python or the underlying libraries. This is because the OpenAPI Generator project or the related library is not actively maintained. Possible solutions include the following.

  - Use a tool such as Typeguard[19] or Pydantic[21] in validating and generating the Python client code. These tools are more actively maintained than the OpenAPI Generator project and are more likely to be compatible with the latest versions of Python.

  - Test the generated code with the version of Python to be used.

  - Manually extend or modify the related validation functions, such as those used in automatic validation of inputs/outputs against their schemas.

- **Misalignment of developing OpenAPI specification and code generation tool**: The OpenAPI Specification (OAS) is still under development. The OGC API standard, such as the OGC API — Processes Standard, is also under heavy development. New features of OpenAPI specification and OGC Standards will emerge. There are also imperfections existing in current specifications and the standards. The Generator may fail due to such incompleteness or imperfections. Possible solutions are as follows.

  - Use a tool like Typeguard[19] or Pydantic[21] to generate schema for OpenAPI specification during the development of standards. These tools can help in writing a more complete and accurate specification that are readily supported by existing tools.

  - Validate the OpenAPI specification using a tool like the OpenAPI Validator[23]. An OAS-specific validator can check the compliance of OpenAPI specifications and definitions. Such a rigorous specification development process with consistent validation helps in identifying any errors or omissions at an early stage.

  - Generate client/server code and validate the client/server models of multiple programming languages using a tool such as OpenAPI Generator during the development of OAS specifications for OGC Standards. Such programmatic validation assures the programming-language-specific support of automatic code generation and validation for both client and server.

    - Defer the validation during the stubs/models generation stage by using option — *skip-validate-spec*. However, this would increase the human review and handling of generated models and classes.

- **Limited support for complex workflows**: OGC API — Processes can be used to implement complex workflows, but automatic code generation may not be able to handle all of the possible cases. Possible solutions are as follows.
    - Use a tool such as ypeguard[19] or Pydantic[21] to generate the Python client code. These tools can generate code for complex workflows, but additional code to handle specific cases may need to be written.
    - Review the generated code and make any necessary changes. This is especially important for handling complex workflows.

- **Lack of customization options**: Automatic code generation tools may not offer many customization options, making it difficult to generate code that meets your specific needs. Possible solutions are as follows.
    - Use a tool such as ypeguard[19] or Pydantic[21] to generate the Python client code. These tools offer more customization options than the OpenAPI Generator project.
    - Write a specialized code generation tool. This will give complete control over the generated code.
    - Use a generic class or object as a placeholder, leaving the detailed handling of objects to the end users of the API client library.

- **Human review is still necessary**: Reviewing the generated code and making any necessary changes is important because automatic code generation tools can make mistakes. Possible solutions include the following.
    - Review the generated code carefully, which is especially important for handling error conditions and using Python idioms.
    - Test the generated code thoroughly, which will help identify any bugs in the generated code.

5

HPGC NOTEBOOK

5 HPGC NOTEBOOK

This Section describes the HPGC API with a Jupyter Notebook use case. This use case is designed to demonstrate the complete process of using an HPGC API implementation instance to plan, schedule, monitor, and receive the high demand geospatial computing task.

NOTE: This section describes the use cases documented in Jupyter Notebooks. This task was led by George Mason University.

5.1. Introduction

The HPGC API Python client library is demonstrated in several Jupyter Notebooks for interacting with an implementation of the HPGC API, which is based on the OGC API — Processes Standard. This library can be used to perform a wide variety of tasks, such as deploying and invoking HPC processes, monitoring job status, and retrieving results. The HPGC API Python client library interacts with the middleware (OGC API — Processes) to interact with HPC. The API — Processes service was implemented by GeoLabs.

The following notebooks demonstrate the use of the HPGC API Python client library in different scenarios.

- **Echo Process**: This notebook demonstrates how to use the HPGC API Python client library to deploy and invoke a simple “echo” process on the HPC cluster.
- **Image Algebra**: This notebook demonstrates how to use the HPGC API Python client library to perform image algebra (ratio of bands from Landsat images) on the HPC cluster.

5.2. Jupyter Notebooks

5.2.1. Toy echo process notebook

This notebook was designed to show a typical use case for executing a process using the HPGC Python Client Library. The toy process “echo” is used to showcase the complete steps of an asynchronous execution demonstrating how to submit a simple echo job to HPC using the HPGC API Python client library. More details about the notebook can be found in Section 1 of the Appendix — HPGC Notebooks.

5.2.2. Ratio of bands from Landsat images notebook

This notebook was designed to show one of the typical use cases involving all the steps in an HPGC process deployment, discovery, execution, and result retrieval. The process "BandMath" was used to demonstrate the complete process of interacting with the HPGC through the HPGC Python Client Library. This demonstrates:

- how to authorize the client;
- how to deploy a process to HPGC;
- how to submit (execute) an image algebra job in the HPGC;
- how to retrieve the results; and
- how to visualize the results geospatially, using the HPGC API Python client library.

The example shows the whole workflow to calculate the ratio of bands from Landsat images using the HPGC. More details about the notebook can be found in Section 2 of the Appendix — HPGC Notebooks.

5.3. Limitations of the HPGC Notebooks

The HPGC API notebooks are a valuable resource for learning how to use the HPGC API to interact with an HPC system. All the notebooks are developed to demonstrate the HPGC API Python client library. The notebooks demonstrate the capabilities and advantages of using the HGPC API to interact with an HPC system.

- The HPGC API is a powerful middleware for interacting with any HPC system to perform a wide variety of tasks. The API hides the details of complexity of interacting with the HPC platform by using a unified, standard interface. An implementation of the HPGC API provides access to process management, executing a process, monitoring an executed job, retrieving the results, and displaying the results along by incorporating implementations of other OGC Web services (e.g., WMS, API — Maps).
- The HPGC API Python client library is easy to use and provides a simple interface for interacting with the HPGC API. The library is published in the public python repository — PyPI. It can be installed, imported, and used with the standard python installer — pip.
- HPGC API notebooks are a useful way to learn how to use the HPGC API and to explore the different types of jobs that can be submitted to HPC. The notebooks show the steps along with code snippets. Responses are also shown with the execution of each step. All these expose the functions of the HPGC Python Client library and the HPGC API middleware.

However, there are still limitations on the notebooks, the HPGC API Python client library, and the HPGC API middleware. Following is a list some of the limitations or challenges.

- **Limitation of HPGC API**: The HPGC API is based on the OGC API — Processes Standard. However, some Parts of the API – Processes Standard are in development and hence may change in the future. The specification of the process description is quite generic and is not specific enough for specifying the definition of HPGC processes targeting at running in an HPC system. Profiles may need to be developed to better support the HPGC process with well-defined specifications for geospatial inputs and outputs. An algorithm or workload management scheme may be included in the profile to have a more specific standard for HPGC.
- **Limitation of HPGC API Python Client library**: The HPGC API Python client library is a relatively new library and there are still some things to learn about how to use this library effectively. For example, one challenge is that the library does not currently support all the features of draft Parts to the OGC API — Processes Standard that are in development. Not all schema or components are properly modeled with the HPGC client library which forges some of the benefits of automatic validations on input and response of the OpenAPI technology. Additionally, the library can be a bit difficult to use at first as it requires some knowledge of the OGC API — Processes Standard.
- **Limitation of the notebooks**: Limited scenarios are covered in developed notebooks. These notebooks cover only the basic use of the HPGC Python Client library and the HPGC API. The notebooks may not be usable due to the availability of the HPGC API server and security restrictions for using some HPC platforms. Other things to keep in mind when using these notebooks include the following.
- The notebooks are for demonstration purposes only. They may not be suitable for production use.
- The notebooks may not be compatible with all HPC clusters.
- The notebooks may not be compatible with all versions of the HPGC API.

5.4. Future Development

The HPGC API Python client library is a powerful tool for interacting with the HPGC API, which is based on the OGC API — Processes Standard. The Jupyter notebooks described above demonstrate the use of the library in different scenarios. Additional notebooks could be developed to demonstrate other scenarios, such as machine learning, data visualization, and scientific computing.

To further demonstrate the use of the HPGC API and the HPGC Python client library, additional scenarios, such as the follwoing, may be developed and demonstrated in HPGC API notebooks.

- Machine Learning: Use the HPGC API Python client library to train and deploy a machine learning model on the HPC cluster.

- Data Visualization: Use the HPGC API Python client library to generate and visualize data on the HPC cluster.
- Scientific Computing: Use the HPGC API Python client library to perform scientific computing tasks on the HPC cluster.
- Image classification: Demonstrates the use of the HPGC API to submit an image classification job to HPC.
- Object detection: Demonstrates the use of the HPGC API to submit an object detection job to HPC.
- Natural language processing: Demonstrates the use of the HPGC API to submit a natural language processing job to HPC.
- Weather simulation and forecasting: Demonstrates the use of the HPGC API to deploy and execute a job that produces huge amounts of data in the HPC.

6

CONCLUSIONS

6 CONCLUSIONS

This Chapter will summarize lessons learned and findings.

NOTE: This section has contributions from all participants and presents major results, findings, lessons learned, open discussions, and conclusions.

6.1. Research Questions and Discussions

The HPGC task started with the following questions in mind.

- Are there any (representative) geospatial libraries that can be initially adapted in High-Performance Computing (HPC) environments to support a wide range of geospatial workflows?
- For data-intensive workloads, what is a standardized geospatial data model that can exploit heterogeneous High-Performance Computing (HPC) resources?
- How can code be contributed to an open High-Performance Geospatial Computing (HPGC) platform while avoiding potential misuse of High-Performance Computing (HPC) computational resources?

In Testbed 19, the prototype implementations and scenarios across HPGC API middleware service, access client, HPGC workflow design, and data retrieval were experimented upon. The following three sections describe the major results in responding to the research questions above, respectively.

6.1.1. The HPGC API based on OGC API — Processes

The rationale for selecting the OGC API — Processes Standard as the base to develop HPGC API: Existing and developing OGC Standards are widely accepted in the geospatial domain. Nearly all of the functional requirements for bridging HPGC can be met by using requirements as specified in the API — Processes Standard.

6.1.2. Data management and data models for HPGC

Developers and Scientists generally utilize High Performance Computing (HPC) resources to scale compute-heavy workflows. However, modern computing problems usually come coupled with large datasets and associated challenges. These challenges are further aggravated by the complexity and high dimensionality of geospatial data. Traditional geospatial workflows

accelerated on HPC infrastructure usually incur high IO overhead associated with data movement costs into and out of the HPC infrastructure.

While there are some strategies such as lazy loading, etc., to manage large input datasets, no particular methodology exists except to download output data to the local environment for further analysis. For geospatial output data, instead of downloading the data, it may be useful to visualize the data using an intermediary mapping service such as a Web Map Service (WMS), etc. The mapping service can directly render the output to the user instead of moving all the raw output data back to the local user environment.

A typical High Performance Geospatial Computing (HPGC) workflow involves the following stages on data.

- Read input data on an HPC platform.
- Process data on an HPC platform.
- The HPC platform generates output data.
- Storage strategy of output data. Possibly, either one of the following options.
    - Output data is moved to a web server hosting mapping service.
    - Output data is downloaded to the local non-HPC environment.
- Output data is visualized for further analysis.

The challenging question for data flows in HPGC is: Where should the web mapping service be hosted? If hosted outside the HPC environment, data would need to be moved to the hosting environment, defeating the purpose of removing unnecessary data transmissions. To achieve the purpose delivery of output data, possible solutions may include the following.

- *Possible Solution-1*: A possible solution is to bring the service closer to the data. HPGC resources are generally shared among multiple users and jobs usually run in singularity/Apptainer containers on an HPC platform. This allows for efficient decoupling of different environments on the HPC resources. At the same time, this also allows for the opportunity to configure custom software inside the containerized environment. Can the mapping service be embedded inside the singularity container used for HPGC tasks? **The challenge of this solution**: HPC systems are usually closed systems with high security measures. Hosting a web service will require exposing network endpoints to the outside world which theoretically goes against HPC security principles. Further exploration is required to understand security on singularity containers and requirements to configure mapping service on HPC.
- *Possible Solution-2*: Another option is to take advantage of data management services such as Globus which can integrate with HPGC backends[6][9]. In this case, the mapping service will be hosted on a non-HPC server. A shared data access point will be created between the HPGC environment and the mapping server. An HPGC job will write output data to the shared directory which will be accessible to the mapping server via a Globus-like shared endpoint. **The challenge of this solution**: While this may avoid security issues related to the HPC platform, this will eventually still require data movement between

the server and the HPGC environment. There is the assumption that Globus-like services may have incorporated methodologies on their end to mitigate costs associated with data movement between shared endpoints.

6.1.3. Standard workflows and vulnerability assurance

HPC resources need to be protected. The code to be executed on the HPC platform needs to be reviewed before deploying into the HPC environment. In CyberGIS-Compute, the workflow code needs to be pre-hosted on GitHub repositories. All submissions must be verified through a human check and review process. The workflow code provides configurations, system environment, and developer API. In CyberGIS-Compute, Git commit version locking is applied to the workflow code for enhanced security, guaranteeing consistent and tamper-proof execution.

In Testbed 19, the workflow code development process was “modernized” by adopting standardized workflow languages such as SLURM and CWL. However, to maintain the security of HPGC, a human review process is still necessary. The OGC Best Practice for Earth Observation Application Package Description was utilized to describe and register the process.

To further enhance workflow security and accessibility, consider implementing the following measures as new requirements in the OGC API — Processes Standard.

1. Automated Code Review: Employ automated code review tools to identify potential vulnerabilities or security flaws in workflow code before deployment. A specialized API operation can be integrated into the process management to facilitate human involvement in the review process.
2. Code Version Control Service: API — Processes should leverage version control to manage code at different levels of security assessment.
3. Documentation and User Guidelines: Provide comprehensive documentation and user guidelines to assist end users in effectively understanding and utilizing workflows. The Application Package Description can be expanded to incorporate this HPGC-specific information.
4. Regular Security Audits: Conduct regular security audits of the code repository and workflow deployment process to identify and address potential vulnerabilities.

6.2. Lessons Learned

Lessons learned:

- Standardization of HPGC API: Standardization of an HPGC API promotes interoperability, simplifies development, and reduces maintenance costs. The following are the fundamental challenges observed during the standardization process.

    - Identifying HPGC stakeholders is essential.
    - Specifically describing and formally defining HPGC is hard and requires input from a broad set of stakeholders.

- Use of API — Processes for HPGC API: The OGC API - Processes — Part 2: Deploy, Replace, Undeploy draft specification with the OGC Application Package offers a practical way to deploy and execute processes on a Server Instance. The OGC API — Processes — Part 1: Core Standard and its processes.yaml schema supports specifying additional parameters at every level of the process description. The prototype implementation uses them for defining process binding information embedded within the payload used for the Deploy operation. By using the CWL encoding for deploying processes and following the OGC Best Practice for Earth Observation Application Package, defining any process binding is not required, easing the creation of new processes and their management. The ZOO-Project proved to be a fast-prototyping platform.
    - Developing HPGC standards will not only help system developers but will also be useful for end users.

- Performance optimization: Performance optimization for HPGC APIs should focus on minimizing data transfer, utilizing parallel processing, and employing efficient memory management techniques. While addressing these aspects, several challenges must be considered. These may include:
    - considerable effort is required to optimize traditional geospatial algorithms to take full advantage of HPGC systems; and
    - parallelization strategies can generally improve the performance of an HPC workflow. However, the performance of modern HPC workflows, and especially HPGC workflows, are highly dependent on the data handling and management.

- Data management: Decoupling the data service from HPGC systems poses a significant challenge. Retrieving relevant results from large datasets generated during simulations or computations can be particularly difficult.

6.3. Future Work

Some future directions include the following.

- Performance optimization: The performance optimization for HPGC API implementations may further explore hardware-specific optimization (utilizing unique capabilities of specific HPGC hardware, such as GPUs or Field Programmable Gate Arrays), develop adaptive performance optimization strategies, enhance data reduction/compression, incorporate advanced machine learning-based optimization, optimize partitioning and distribution, develop performance benchmark, and optimize geospatial data models.

- Resource management for HPGC: Data resource management for HPGC may consider developing adaptive data management strategies (discovery & retrieval), integrating geospatial data visualization, develop data access control and leveled-security mechanisms, and implement data lineage and quality assurance.
- Geospatial data indexing and partition for HPGC: To advance the use of parallelism in HPGC implementations, future studies along the direction of geospatial data indexing and partitioning may investigate adaptive indexing strategies, dynamic partitioning, impact of indexing and partitioning on query performance, and best practices on geospatial data indexing and partitioning.
- More specialized notebooks for different scenarios and use cases: Potential areas for future work on developing more notebooks to demonstrate the typical scenarios and use cases for the use of HPGC through implementations of the HPGC API may include building up a rich collection of notebooks covering a wide range of HPGC use cases, developing categorized notebooks on application domains, incorporating standard interactive geospatial visualizations, supporting coding with multiple programming languages, and establishing a common repository for sharing and maintaining community-developed notebooks.
- Big data handling: Big data handling can be challenging for HPGC at both the server-side and the client-side. Future studies may explore efficient data ingestion and processing techniques, investigate data management frameworks, optimize data compression strategies, survey real-time data streaming and processing techniques, implement data quality assurance, and develop benchmark suites.

6.4. Conclusion

This report details the design and implementation of an API-based approach for high-performance geospatial computing (HPGC). A key component, the HPGC API, was developed and profiled based on the existing OGC API – Processes Standard demonstrating its ability to support and bridge the gap between HPC resources and end users. The HPGC API provides a standardized and geospatially-aware interface for accessing and manipulating HPC workflows and for simplifying the user experience compared to previous middleware solutions like CyberGIS-Compute.

To further enhance user accessibility, a client Python library based on the HPGC API was implemented. This library streamlines workflow deployment, execution, monitoring, and result visualization within the familiar environment of Jupyter notebooks. This exemplifies the ease and flexibility of utilizing the HPGC API for real-world geospatial computing tasks.

The development of the Python client library also serves as a template for creating client libraries in other programming languages. By leveraging OpenAPI Generator, a powerful tool supporting various popular languages, developers can readily create dedicated client interfaces for their preferred environment. This fosters broader adoption of the HPGC API, enabling

researchers and professionals from diverse backgrounds to readily leverage the power of HPC for geospatial analysis.

Utilizing the OGC API — Processes — Part 2: Deploy, Replace, Undeploy draft specification enabled demonstration of deploying the corresponding Singularity container in the HPC platform for a set of processes available in a Docker container. This deployment facilitated remote invocation of the targeted process execution using the OGC API — Processes — Part 1: Core. Nevertheless, to achieve this goal, an implementation-specific use of the additional parameters for defining the process binding was used. In other words, the way the command to execute the application, embedded in the deployed singularity, should be written in the sbatch file. This prototype implementation partially covers the CyberGIS-Compute capabilities, including the Job Submission and the Job monitoring & results.

If CWL had been leveraged as the encoding for deploying processes, this implementation-specific use of additional parameters would not have been required. Also, by reusing the OGC Earth Observation Application Package Best practice, there would be the advantages of using cloud-native formats and would have solved the issue of downloading and transferring the data to HPC. Also, comparing the Stage-in and Stage-out requirements classes from the OGC Earth Observation Application Package Best Practice with the Initialize and Finalize part of the CyberGIS application is possible. Also, the CWL encoding can be used to compare the manifest from CyberGIS with the Application Package. In such a new prototype implementation, the download repository operation defined in CyberGIS is comparable with the download of the CWL file.

In conclusion, the HPGC API and its client SDK represent a significant step forward in bridging the gap between HPC infrastructure and end users. By providing a standardized, geospatially-aware interface, coupled with readily deployable client libraries, this approach empowers users of all skill levels to harness the immense potential of HPC for groundbreaking geospatial discoveries.

While the HPGC API and client SDK offer a powerful foundation, several exciting avenues beckon for further exploration.

- Human-in-the-Loop HPC Workflows: Integrating human expertise into the workflow deployment and security process, particularly through an “audition” step, could ensure code quality and address potential security vulnerabilities before full-scale HPC execution. This could involve automated code reviews with human oversight, fostering a more robust and secure environment for HPC geospatial analysis.

- Data-Intensive Processes: The HPGC API should be further optimized for data-intensive workflows, where massive datasets are either fed into HPC for analysis or produced as outputs from simulations. This entails efficient data transfer mechanisms, real-time progress monitoring tailored to data volumes, and optimized data retrieval and visualization solutions that can handle large geospatial datasets effectively.

- Deep Dive into HPC Workflow Coding: Moving beyond the API layer, future efforts could delve into the actual workflow coding for HPC, incorporating geospatial considerations from the ground up. This might involve developing common frameworks specifically designed for geospatial HPC workflows, enabling programmers to leverage optimized geospatial algorithms and data structures within their code.

- Advanced Use Cases for HPGC: Advanced Use Cases for HPGC: By pushing the boundaries of HPGC, its potential for powering cutting-edge geospatial applications can be further explored. This could involve leveraging deep learning models for automated feature extraction and classification, employing AI techniques for knowledge extraction and prediction from geospatial data, and developing specialized frameworks for tackling complex geospatial challenges like climate modeling and urban planning.

A

ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

A ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Architecture
CWL	Common Workflow Language
CyberGIS	Cyberinfrastructure-based GIS
DRU	deploy, replace, and undeploy a process
FGPA	field-programmable gate array
GIS	Geographic Information System
GPU	Graphic Processing Unit
HPC	High Performance Computing
HPGC	High Performance Geospatial Computing
HTML	HyperText Markup Language
JAR	Java Archive
JSON	JavaScript Object Notation
MPI	Message Passing Interface
OAS	OpenAPI Specification
OIDC	OpenID Connect
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing

OTB	Orfeo Toobox
SLURM	Simple Linux Utility for Resource Management
SSH	Secure Shell Protocol
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
YAML	YAML Ain't Markup Language

B

ANNEX B (INFORMATIVE) HPGC FRAMEWORKS

B

ANNEX B (INFORMATIVE) HPGC FRAMEWORKS

NOTE: This annex provides a review of existing HPGC frameworks.

B.1. HPGC Frameworks

The Testbed 19 participants reviewed the existing CyberGIS-Compute, the middleware developed and used at UIUC for bridging HPC systems and end users.

B.1.1. CyberGIS-Compute

B.1.1.1. Architecture

CyberGIS is defined as a "fundamentally new GIS modality based on holistic integration of high-performance and distributed computing, data-driven knowledge discovery, visualization and visual analytics, and collaborative problem-solving and decision-making capabilities."[18]

CyberGIS-Compute is an open-source framework for HPGC that provides a high-level API for interacting with HPC resources[20]. It is designed to be scalable and flexible, making it suitable for handling large datasets and complex workflows.

CyberGIS-Compute is based on a three-tier architecture.

- The CyberGIS-Compute SDK is a Python-based SDK that provides a high-level API for interacting with the CyberGIS-Compute framework. It is used to submit workflows to HPC resources, manage jobs, and access data and results.
- The CyberGIS-Compute Core is the core middleware service that provides the underlying infrastructure for the CyberGIS-Compute framework. It includes a scheduler, a file system, and a communication layer. The scheduler is responsible for managing the execution of workflows on HPC resources. The file system is used to store data and intermediate results. The communication layer is used to exchange data between the different components of the framework.

- The CyberGIS-Compute Plugins are a collection of third-party plugins that extend the functionality of the CyberGIS-Compute framework. Plugins are available for a variety of tasks, such as data processing, visualization, and analysis.

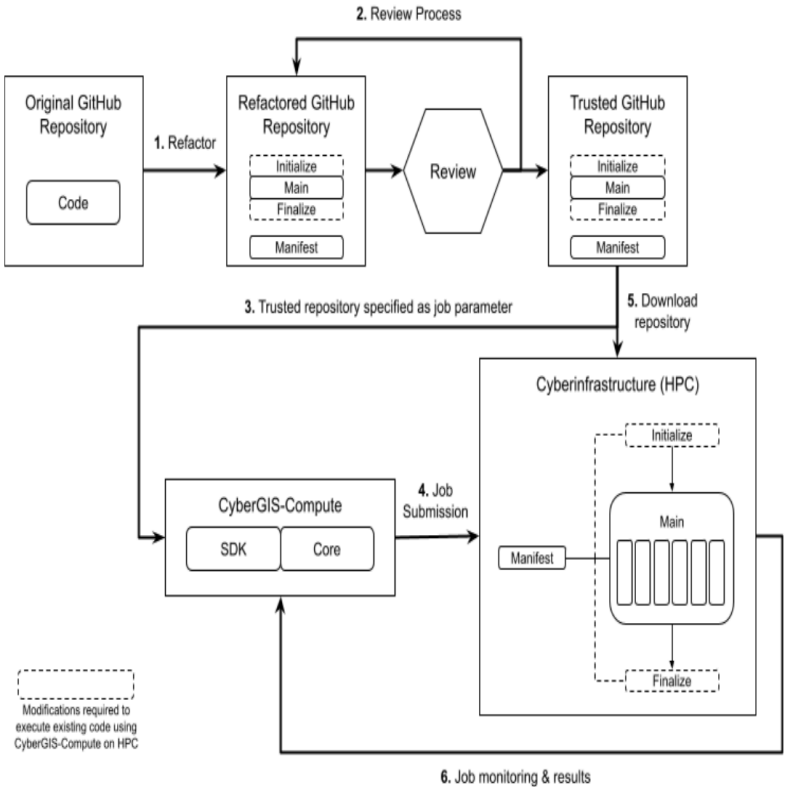


Figure B.1 — Architecture of CyberGIS-Compute

B.1.1.2. Internal Working

The internal working of CyberGIS-Compute's architectural components is designed to support High-Performance Geospatial Computing (HPGC) workflows. The following is a review and commentary on these components.

1. High Performance Computing Infrastructure: CyberGIS-Compute leverages an HPC platform that is built as a parallel computing architecture across multiple architectures (across CPU, GPU, and FPGA), with cluster computing, or a grid and distributed computing. The HPC platform provides mechanisms for parallel computation, allowing tasks to be executed across multiple computing nodes or clusters. The HPC platform can handle the computational demands of HPGC workflows.

2. Geospatial Abstractions: The framework incorporates geospatial abstractions that simplify the representation and manipulation of geospatial data. These abstractions provide higher-level programming interfaces and tools that are specifically tailored for geospatial analysis. By abstracting the complexity of

working with geospatial data, CyberGIS-Compute enables users to focus on the analysis tasks rather than low-level data handling.

3. Data Storage and Management: Efficient data storage and management are critical components of HPGC workflows. CyberGIS-Compute utilizes geospatial databases, distributed file systems, or cloud storage to store and manage geospatial data. This allows for easy access, retrieval, and processing of data during geospatial analysis tasks. The choice of storage systems depends on the specific requirements and scale of the geospatial data being processed.

4. Workflow Orchestration: CyberGIS-Compute supports workflow orchestration, enabling users to define and manage complex geospatial computing workflows. This component supports the integration of multiple geospatial analysis tasks into a cohesive and automated workflow. Users can define dependencies, task execution order, and data flow between different stages of the workflow. Workflow orchestration enhances the efficiency and reproducibility of HPGC workflows.

5. Integration with CyberGIS Gateway: CyberGIS-Compute integrates with the CyberGIS Gateway, a web-based platform that provides access to geospatial computing resources and tools. This integration enhances the usability and accessibility of the framework by providing a user-friendly interface for interacting with CyberGIS-Compute's capabilities. Users can access and utilize the framework's features through the gateway, making it easier to deploy and manage HPGC workflows.

6. Performance Optimization: CyberGIS-Compute emphasizes performance optimization to achieve efficient geospatial computing. This includes load balancing techniques, task scheduling algorithms, and resource allocation strategies to effectively distribute computation across the available resources. These optimizations aim to maximize the utilization of computing resources, reduce processing time, and enhance the overall performance of HPGC workflows.

The internal working of CyberGIS-Compute's architectural components aligns with the requirements of High-Performance Geospatial Computing. The integration of HPC infrastructures, geospatial abstractions, workflow orchestration, data storage and management, and performance optimization enables efficient and scalable processing of geospatial data. The integration with the CyberGIS Gateway further enhances the usability and accessibility of the framework. However, the specific implementation details and configuration options may vary, depending on the chosen technologies and customization requirements for specific HPGC workflows.

B.1.1.3. Potential Improvements of CyberGIS-Compute

Possible improvements to the framework of CyberGIS-Compute to make it be a general High-Performance Geospatial Computing framework are as follows.

1. Standardization of Geospatial Abstractions: Establish and promote standard geospatial abstractions within the framework. This includes adopting widely recognized geospatial data models, formats, and metadata standards to ensure interoperability with other geospatial tools and systems. Standardized geospatial abstractions facilitate data exchange, collaboration, and reusability across different HPGC workflows.

2. Support for multiple programming languages: CyberGIS-Compute currently provides a Python-based interface for accessing HPC resources. However, to be a more general HPGC framework, it could support other programming languages such as R, Java, and C++ to cater to researchers with different programming backgrounds.

3. Interoperability with External Systems: Enhance interoperability with external geospatial systems and tools commonly used in the geospatial community. This can be achieved by implementing standard-compliant connectors or APIs that allow seamless integration with popular geospatial software packages, spatial databases, or cloud-based geospatial services. This interoperability enables users to leverage existing geospatial resources and tools while utilizing the capabilities of CyberGIS-Compute.

4. Enhanced Workflow Composition and Management: Improve the capabilities for workflow composition and management within CyberGIS-Compute. This involves providing intuitive graphical interfaces or domain-specific languages (DSLs) for defining and visually composing complex geospatial workflows. Additionally, incorporating features like versioning, error handling, and provenance tracking can improve the overall robustness and manageability of HPGC workflows.

5. Adaptive Resource Management: Develop adaptive resource management techniques within the framework to dynamically allocate and manage computing resources based on the varying demands of geospatial analysis tasks. This includes dynamic scaling of resources to handle workload fluctuations, intelligent scheduling algorithms for task allocation, and optimized resource utilization to improve performance and cost efficiency.

6. Advanced Performance Optimization: Implement advanced performance optimization techniques tailored specifically for HPGC workflows. This may involve exploring data partitioning strategies, algorithmic optimizations, and leveraging advanced computing architectures like GPUs or FPGAs for specific geospatial analysis tasks. Optimizing the execution of geospatial algorithms can significantly improve the performance and scalability of the framework.

7. Improved integration with other geospatial tools and frameworks: CyberGIS-Compute is designed to be integrated with other CyberGIS components, such as CyberGIS-Jupyter. However, to be a more general HPGC framework, it could also be integrated with other popular geospatial tools and frameworks such as GDAL, QGIS, and GeoSpark.

B.1.1.4. Standard Development from CyberGIS-Compute

The CyberGIS-Compute framework can serve as a valuable starting point for the standardization of High-Performance Geospatial Computing (HPGC). The following is a preliminary assessment of its applicability in this context.

1. Addressing Common Challenges: CyberGIS-Compute tackles common challenges in HPGC, such as scalability, performance, and parallel computing. By providing a framework that addresses these challenges, it establishes a foundation for standardizing the fundamental aspects of HPGC workflows. This can help establish common practices and guidelines across different implementations and platforms.

2. Geospatial Abstractions: The framework's geospatial abstractions provide a standardized way of handling and processing geospatial data. This can contribute to standardization efforts by promoting consistency and interoperability in geospatial analysis tasks. By adopting common geospatial abstractions, developers and researchers can ensure compatibility and portability of their HPGC workflows.

3. Workflow Orchestration: The support for workflow orchestration in CyberGIS-Compute is crucial for standardizing HPGC. By defining standardized workflow structures and interfaces, it becomes easier to exchange and integrate geospatial workflows across different systems and platforms. This can facilitate collaboration and interoperability between HPGC implementations.

4. Open-Source and Community Collaboration: The open-source nature of CyberGIS-Compute encourages community collaboration and contributions. This can foster a collective effort towards standardization by allowing researchers, developers, and users to actively participate in shaping the framework's development. Open discussions, feedback, and contributions can drive the identification and establishment of best practices and standards in HPGC.

5. Integration Challenges: While CyberGIS-Compute provides a strong foundation, integrating it into existing geospatial computing environments and workflows may require careful consideration. Standardization efforts should consider the compatibility and integration challenges associated with adopting CyberGIS-Compute or similar frameworks. Ensuring interoperability with existing tools and technologies is crucial for broader adoption and acceptance within the geospatial community.

6. Broad Stakeholder Involvement: To establish effective standards in HPGC, it is important to involve a broad range of stakeholders, including researchers, practitioners, industry experts, and standards organizations. CyberGIS-Compute can serve as a catalyst for bringing these stakeholders together, providing a starting point for discussions, collaborations, and consensus-building around standardization.

CyberGIS-Compute has the potential to serve as a starting point for the standardization of High-Performance Geospatial Computing. Its focus on addressing common challenges, providing geospatial abstractions, supporting workflow orchestration, and fostering community collaboration are essential elements for establishing standards in HPGC. However, integration challenges and the involvement of a diverse set of stakeholders should be considered to ensure the practical applicability and acceptance of such standards.

C

ANNEX C (INFORMATIVE) HPGC NOTEBOOKS

C

ANNEX C (INFORMATIVE) HPGC NOTEBOOKS

NOTE: This annex contains the detailed descriptions of the HPGC notebooks.

C.1. Toy echo process notebook

The Figure C.1 is a simple diagram showing the flow of data between the user, the HPGC API Python client library, and the HPC cluster for the echo process scenario.

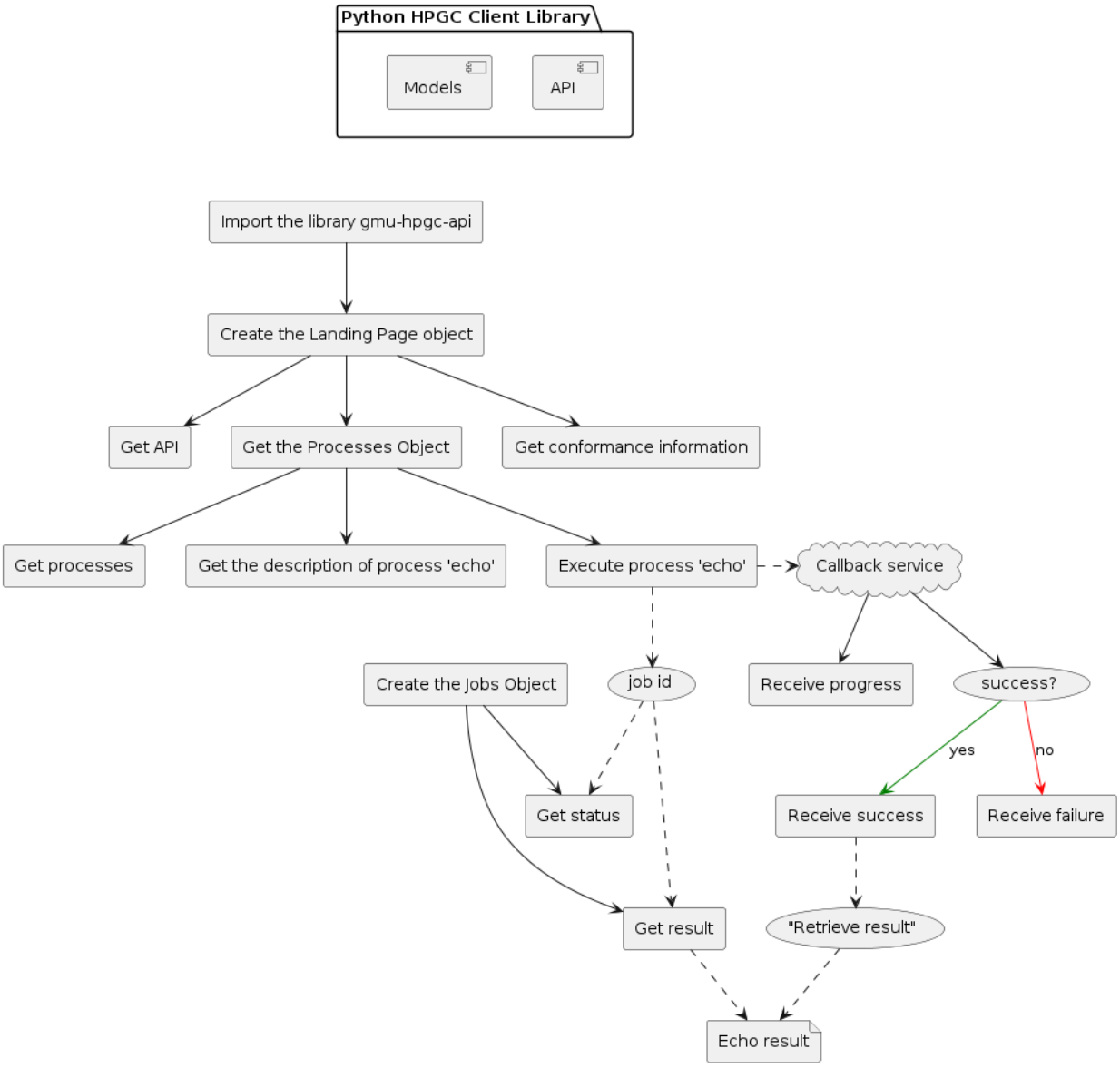


Figure C.1 — The notebook to execute a toy echo process

The notebook includes the following steps:

1. *Preparation*: Import the necessary libraries.

2. *Create an HPGC LandingPage object*: The object allows the user to check the basic information of the service, including metadata information, API information, API testing page, conformance information, and list of processes.

3. *Create an HPGC Processes object*: This object allows the user to list all the processes and retrieve the description of a specific deployed service.

4. *Execution of the echo process*: In this case, the Processes object is used to retrieve the description of the deployed "echo" process and use the description to form an execution request. A request is created with definitions of necessary inputs, execution method, outputs, and callback services (if applicable). The

asynchronous execution approach is selected in the request. The request also gives the callback service for receiving progress status, success result, and failure exception. Then, the formed execution request is submitted to the service.

5. *Create an HPGC Jobs object*: This object allows the user to list all the jobs at their service, get the status of a job, and get the result of a job.

6. *Monitor the job status*: There are generally two approaches to monitor the submitted job, depending on the implementation of the API — Processes service. One approach is using the job ID returned when submitting the execution request. The client may keep on pulling the status until the status is changed to the final step — either a success response or an exception response. Another approach relies on the service to push back progress status during the execution of the job request. The service will receive three service callbacks: a progress callback, a success callback, and a failure callback. The callback services will receive the status update from the service. Results may be streamed down to the callback service if the server implements so.

7. *Retrieve the job results*: The results may be given as links when the success status is reached. The result for a specific job can be retrieved. In the test echo process, a result link will be shown in the success response. The result can be used to retrieve the actual results.

8. *Print the job results*: Use the link in the success response to retrieve and download the results. In the case of "echo" process, a simple text results will be returned along with input parameters. The results can be printed out in this case.

C.2. Ratio of bands from Landsat images notebook

The Figure C.2 is the more complex diagram showing the flow of data between the user, the HPGC API Python client library, and the HPC cluster for the image algebra scenario. This diagram includes details such as the different steps involved in the image algebra process and the different types of data that are exchanged between the different components of the system. The process package management is also demonstrated which may include operations of deploy, undeploy, and replace.

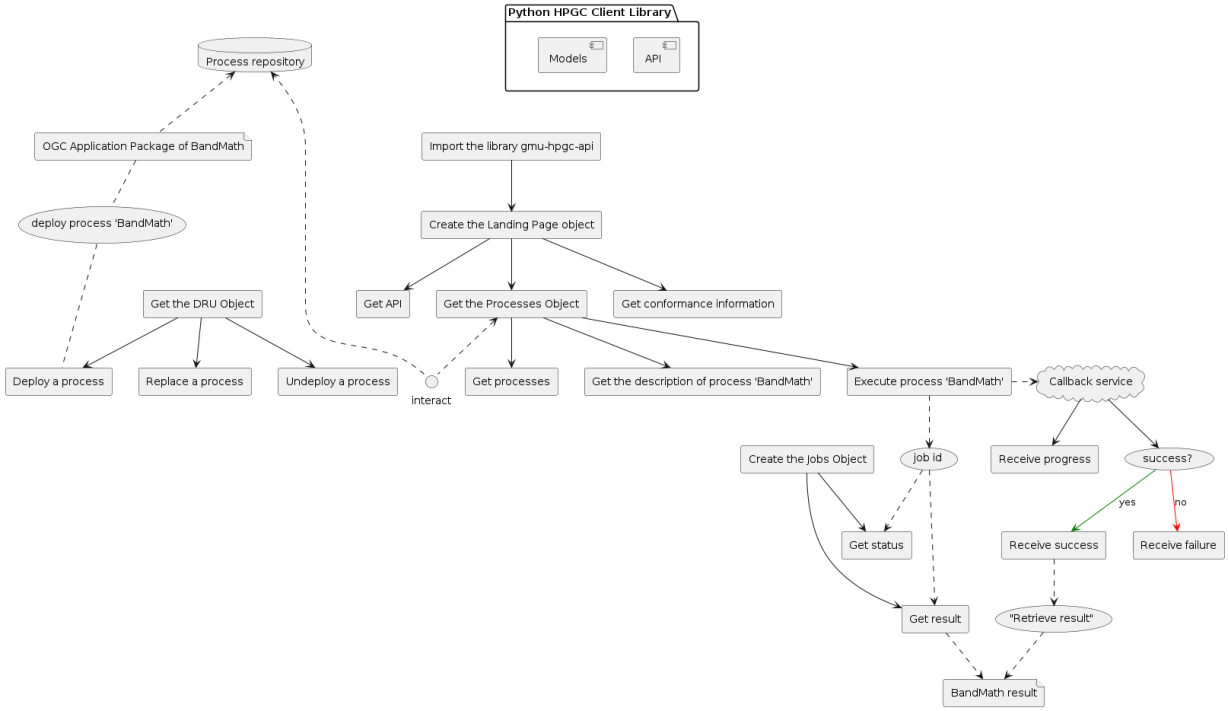


Figure C.2 — The notebook to execute a image algebra process

The notebook includes the following steps.

1. *Preparation*: Import the necessary libraries.

2. *Create an HPGC LandingPage object*: Use this entrypoint object to review the basic information about the service.

3. *Authentication to the server*: This scenario uses a secure service to interact with the HPC. Advanced transaction operations (such as Deploy, Replace, and Undeploy) are also required to be protected. The authorization process is an Open Identity authorization process.

4. *Create a DRU object*: DRU stands for operations of deploy, replace, and undeploy a process. This object interacts with the process's transaction operations. This functionality is not in the core of API – Processes Standard. This capability supports the object allowing the user to list all the processes and retrieve the description of a specific deployed service.

5. *Deploy "BandMath" process*: Deploy the process "BandMath" into the service using the DRU object. The BandMath process is described following the OGC Application Package schema which must have an execution unit that describes the process or package. The process description may be included in the Package definition to provide additional metadata information for the process. In this scenario, the process is pre-packaged as a singularity based SLURM workload manager script. Once the process is deployed, it can be used as a normal process.

6. *Create an HPGC Processes object*: Use this object to search for a deployed service. This can be used to verify and find the properly deployed “BandMath” process.

7. *Execution of the BandMath process*: In this case, the Processes object is used to retrieve the description of the deployed “BandMath” process and use the description to form an execution request. A request will be created following the definitions of necessary inputs, execution method, outputs, and callback services. The process supports only an asynchronous execution approach. The request will also give the callback service for receiving progress status, success result, and failure exception. Then, the complete execution request will be submitted to the service.

8. *Create an HPGC Jobs object*: This object allows the user to get information about executed jobs at the server.

9. *Monitor the job status*: The status monitoring uses the pull approach against the returned job ID. The client may keep on pulling the status until the status is changed to the final step — either a success response or an exception response.

10. *Retrieve the job results*: The results may be given as links when the success status is reached. The result for a specific job can be retrieved. In the test BandMath process, a result link will be shown in the success response to retrieve the final resulted image. With the current implementation of the service, it can request the result image to be returned as a static image data or a dynamic geospatial Web Service such as an OGC Web Map Service (WMS) endpoint.

11. *Show the job results*: If the data output is requested, the user may use the link in the success response to retrieve and download the results. The downloaded image can be imported into a geospatial package for displaying and further analysis. If the rendered image is requested, the WMS link will be included in the result response. The WMS service can be embedded as an interactive map in the notebook using Folium. Folium makes it easy to visualize data that has been manipulated in Python on an interactive leaflet map.

BIBLIOGRAPHY

BIBLIOGRAPHY

NOTE: Place annex material in sequential order and set `obligation` attribute as "normative" (default) or "informative" according to the case.

[1] L. Breiman, "Random forests", Machine learning, 45, 5-32, 2001.

[2] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines", *IEEE Intelligent Systems and their applications*, 13(4), 18-28, 1998.

[3] R. Vandewalle, "What is CyberGIS-Compute?", Noveber 9, 2021, URL: https://cybergisxhub.cigi.illinois.edu/knowledge-base/components/cybergis-compute/what-is-cybergis-compute/ (last accessed 12/08/2023).

[4] M. P. Armstrong, "High Performance Computing for Geospatial Applications: A Retrospective View," in *High Performance Computing for Geospatial Applications*, vol. 23, W. Tang and S. Wang, Eds., in Geotechnologies and the Environment, vol. 23. , Cham: Springer International Publishing, 2020, pp. 9–25. doi: 10.1007/978-3-030-47998-5_2.

[5] "Introduction to Singularity Containers on HPC", URL: https://centers.hpc.mil/users/singularity.html (last accessed on 12/8/2023)

[6] *Data Sharing With Globus*. URL: https://www.globus.org/data-sharing (last accessed: 10/27/2023).

[7] F. Huang, H. Yang, J. Tao, and Q. Zhu, "Universal workflow-based high performance geo-computation service chain platform," Big Earth Data, vol. 4, no. 4, pp. 409–434, Oct. 2020, doi: 10.1080/20964471.2020.1776201.

[8] *Orfeo ToolBox* URL: https://www.orfeo-toolbox.org/

[9] *How To Share Data Using Globus*. URL: https://docs.globus.org/how-to/share-files/ (last accessed: 10/27/2023)

[10] Z. Li, "Geospatial Big Data Handling with High Performance Computing: Current Approaches and Future Directions," in High Performance Computing for Geospatial Applications, vol. 23, W. Tang and S. Wang, Eds., in Geotechnologies and the Environment, vol. 23. , Cham: Springer International Publishing, 2020, pp. 53–76. doi: 10.1007/978-3-030-47998-5_4.

[11] *MapServer* URL: https://mapserver.org/

[12] *OpenAPI Generator*. URL: https://github.com/OpenAPITools/openapi-generator

[13] W. Tang and S. Wang, Eds., *High performance computing for geospatial applications*. in Geotechnologies and the environment, no. volume 23. Cham, Switzerland: Springer, 2020.

[14] *API-Processes*. URL: https://ogcapi.ogc.org/processes/

[15] E. Borin, L. M. A. Drummond, J.-L. Gaudiot, A. Melo, M. M. Alves, and P. O. A. Navaux, High performance computing in clouds: moving applications to a scalable and cost-effective environment. Cham, Switzerland: Springer, 2023.

[16] Benjamin Pross and Panagiotis (Peter) A. Vretanos (eds) *OGC API — Processes — Part 1: Core*. OGC 18-062r2, version 1.0.0, 2021-12-20. URL: https://docs.ogc.org/is/18-062r2/18-062r2.html

[17] Panagiotis (Peter) A. Vretanos and Gérald Fenoy (eds) *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*. OGC 20-044, version 1.0. URL: https://docs.ogc.org/DRAFTS/20-044.html

[18] S. Wang, "Cyberinfrastructure," *Geogr. Inf. Sci. Technol. Body Knowl.*, vol. 2019, no. Q2, Apr. 2019, doi: 10.22224/gistbok/2019.2.4.

[19] *Typeguard* URL: https://typeguard.readthedocs.io/

[20] A. Michels, A. Padmanabhan, Z. Xiao, M. Kotak, F. Baig, and S. Wang, "CyberGIS-Compute: Middleware for Democratizing Scalable Geocomputation," *SSRN*, preprint, 2023. doi: 10.2139/ssrn.4625703.

[21] *Pydantic* URL: https://github.com/pydantic

[22] S. Abuayeid, and L. Alarabi, "Comparative Analysis of Spark and Ignite for Big Spatial Data Processing", *International Journal of Advanced Computer Science and Applications*, 12(9), 2021.

[23] *OpenAPI Validator* URL: https://github.com/IBM/openapi-validator

[24] J. Yu, amd M. Sarwat, "Big geospatial data processing made easy: A working guide to geospark". In *Handbook of Big Geospatial Data* (pp. 35-53). Cham: Springer International Publishing, 2012.