Earth System Grid Federation (ESGF) Compute Challenge

Table of Contents

Publication Date: 2019-09-24

Approval Date: 2019-06-28

Submission Date: 2019-05-16

Reference number of this document: OGC 19-003

Reference URL for this document: http://www.opengis.net/doc/PER/ESGF-er

Category: OGC Public Engineering Report

Editor: Tom Landry, David Byrns

Title: Earth System Grid Federation (ESGF) Compute Challenge

OGC Public Engineering Report

COPYRIGHT

WARNING

LICENSE AGREEMENT

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

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

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

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

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

indicate compliance with any LICENSOR standards or specifications.

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

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

Chapter 1. Summary

This Open Geospatial Consortium (OGC) Engineering Report (ER) will describe the advancement of an Execution Management System (EMS) to support Web Processing Service (WPS) climate processes deployed on the Earth System Grid Federation (ESGF). The report introduces climate data, processes and applications into Common Workflow Language (CWL) workflows with the intent of advancing: application packaging, deployment and execution in clouds; interoperability of services in federated cyberinfrastructures; and geospatial workflows towards standardization. Work presented in this report is a direct continuation of the Earth Observation & Clouds (EOC) thread of Testbed-14. This report is expected to be of relevance to Testbed-15, both to the Earth Observation Process and Application Discovery (EOPAD) task and the Machine Learning task. This engineering report will describe: relevant work conducted in OGC Testbed-14; ESGF and its compute challenge; adaptations of existing climate processes into workflows; interoperability experiments with ESGF endpoints conforming to a common API.

1.1. Requirements & Research Motivation

The Thematic Exploitation Platform (TEP) open architecture advanced in previous testbeds tackled Earth observation data in the context of federated, heterogeneous infrastructures. The main requirements included support for application deployment, execution and chaining, as well as the agreed authentication and authorization mechanisms. All interfaces needed to be agreed upon and standardized so as to achieve the Sponsors' (European Space Agency, Natural Resources Canada) interoperability goals.

It was observed by certain testbed participants, observers and stakeholders that the same set of requirements applies to climate data, processes and infrastructure. As such, the U.S. Department of Energy’s (DOE) Office of Biological and Environmental Research (BER) sponsored additional engineering work to ESGF interoperability. The general objectives proposed and presented to ESGF stakeholders are as follows, in order of importance:

1. Test common ESGF climate processes and their compute nodes
2. Demonstrate Testbed-14 TEP architecture for climate processes and applications
3. Advance application packaging, deployment and execution in clouds
4. Advance interoperability of services in ESGF federated cyberinfrastructure
5. Advance geospatial workflows towards standardization
6. Explore use of Earth Observation (EO) data and processes in support of climate infrastructure
7. Facilitate use and exchange of Machine Learning (ML) systems and learned models

1.2. Prior-After Comparison

Previous work in OGC testbeds allowed definition of application packages in workflow-enabled environments, and their subsequent deployment and execution on distributed secured infrastructures. Data discoverability was superficially addressed through inclusion of OpenSearch capabilities to identify and select the appropriate data from catalogs. Applications developed for the occasion included stacking, subsetting, feature detection and index computation, both on

optical and Synthetic-Aperture Radar (SAR) data. All these concerns are of particular interest for OGC's Earth Observation Exploitation Platform (EO Ex Platform) Domain Working Group (DWG).

The work described in this ER improves interoperability of workflows with pre-deployed WPS 1.0 and WPS 2.0 processes. It also enables the inclusion of pre-deployed processes adhering to the ESGF Compute Working Team (CWT) Application Programming Interface (API). The workflows proposed in this work include climate data (CMIP6, statistically downscaled climate scenarios, variables and indexes), climate processes (index computation) and base processes (subsetting, averaging). Three different infrastructures were used for interoperability experiments.

The last two objectives of the ESGF concept, the use of EO in support of climate infrastructures and the integration of ML systems, were only partially and informally addressed in this work. It is expected that work planned and conducted in OGC Testbed-15, as well as in other collaborative innovation projects, will continue to support these efforts and drive future requirements. This report contains relevant content potentially supporting OGC request for information (RFI) [https://portal.opengeospatial.org/files/83548] on Earth Observation Big Data Architecture.

1.3. Recommendations for Future Work

This section presents recommended potential future tasks and deliverables that can support advancement of requirements and expand research motivations. The reader can also refer to Section 9 of this Engineering Report for discussion and open issues for further details.

1.3.1. Recommended Future Tasks

1. Common architecture for climate and Earth Observation (EO) platforms
2. ML systems and models interoperability in federated cyberinfrastructures

1.3.2. Recommended Future Deliverables

Recommended Future Components

The following components are suggested to be deployed for testing, demonstration and integration purposes. The resulting functionalities of these components could support both recommended future tasks.

1. Clients that can support both EO and climate data
2. ESGF Application Profile
3. Open Search enabled catalog for climate data
4. Ontology to use in CWL file formats supported in workflows
5. ML models for detection and localization of extreme climate events
6. ML systems supporting interfaces for interoperable Deep Learning (DL) models, where trained models from a ML system can be loaded into a different one to be used for inference
7. ML-enabled application packages and workflows, where a packaged ML system is used in conjunction with pre-processing steps

8. Additional application packages, with different variations of input/output mappings of Docker containers and underlying executable code
9. Advanced clients to provide user feedback on all previously mentioned future components

Recommended Future Engineering Reports (ER)

The following Engineering Reports are suggested to support and document experiments conducted inside previously mentioned recommended future tasks and components.

1. EO and climate federated infrastructures ER
2. Climate data and processes best practices ER
3. Geospatial ML systems best practices ER

1.4. Document Contributor Contact Points

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

Contacts

Name	Organization
Tom Landry	CRIM
David Byrns	CRIM
David Caron	CRIM
Francis Charette-Migneault	CRIM

1.5. Acknowledgements

In addition to participants and observers in this testbed activity, the following institutions collaborated in the ESGF Compute Challenge and coordinated their efforts:

- Centro Euro-Mediterraneo sui Cambiamenti Climatici (CMCC)
- Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS)
- Deutsches Klimarechenzentrum (DKRZ)
- Lawrence Livermore National Laboratory (LLNL)
- NASA Center for Climate Simulation (NASA NCCS)
- Ouranos
- University of Utah

1.6. Foreword

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

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

Chapter 2. References

The following normative documents are referenced in this document.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2], 2010
- OGC: OGC: OGC 13-026r8, OGC OpenSearch Extension for Earth Observation 1.0 [https://portal.opengeospatial.org/files/13-026r8], 2016
- OGC: OGC: OGC 14-065r2, OGC Web Processing Service 2.0.2 Interface Standard Corrigendum [https://portal.opengeospatial.org/files/14-065r2], 2018
- OGC: OGC: OGC 13-032r8, OGC OpenSearch Geo and Time Extensions 1.0.0 [https://portal.opengeospatial.org/files/?artifact_id=56866], 2014
- OGC: OGC: OGC 12-168r6, OGC® Catalogue Services 3.0 - General Model [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html], 2016
- W3C: A JSON-based Serialization for Linked Data [https://www.w3.org/2018/jsonld-cg-reports/json-ld/], 2018
- CWL: CWL group: Common Workflow Language Specifications, v1.0.2 [https://www.commonwl.org/v1.0/], 2018

Chapter 3. Terms and definitions

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

- AIMS (Analytics and Informatics Management Systems)

Program at LLNL enabling data discovery and knowledge integration across the scientific climate community. (AIMS [https://aims.llnl.gov/])

- CDAT (Community Data Analysis Tools)

CDAT is a powerful and complete front-end to a rich set of visual-data exploration and analysis capabilities well suited for data analysis problems. (CDAT [https://cdat.llnl.gov/])

- CDMS (Climate Data Management System)

Object-oriented data management system specialized in organizing multidimensional gridded data used in climate analyses for data observation and simulation. (CDMS [https://cdat.llnl.gov/documentation/cdms/cdms.html])

- CMIP

Sponsored by the World Climate Research Programme's Working Group on Coupled Modeling, CMIP is a community-based infrastructure for climate model diagnosis, validation, intercomparison, documentation, and data access. (CMIP [https://www.wcrp-climate.org/wgcm-cmip])

- CREATE (Collaborative REAnalysis Technical Environment)

NASA project that centralizes numerous global reanalysis data sets into a single advanced data analytics platform. (CREATE [https://cds.nccs.nasa.gov/tools-services/create/])

- Controlled vocabulary

A controlled vocabulary is an organized collection of terms used to index content in order to facilitate information retrieval (OGC 17-040). Controlled vocabularies provide a way to organize knowledge for subsequent retrieval. They are used in subject indexing schemes, subject headings, thesauri, taxonomies and other forms of knowledge organization systems.

- Dask

Dask is a flexible library for parallel computing in Python. It offers dynamic task scheduling optimized for computation and "Big Data" collections. (Dask [https://dask.org/])

- Deep learning

Deep learning is a class of machine learning algorithms that: use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation; learn in supervised and/or unsupervised manners; learn multiple levels of representations that correspond to different levels of abstraction.

- Downscaling

A method that can provide climate model outputs at a finer resolution than their original resolution. Two different approaches are prioritized: statistical downscaling and dynamical downscaling.

- EDAS (Earth Data Analytics Service)

Developed by NASA NCCS, EDAS is a high-performance big data analytics framework built on Dask/xarray. It allows researchers to leverage computing power to analyze large datasets located at the NCCS through a web-based interface, thereby eliminating the need to download the data. EDAS [https://www.nccs.nasa.gov/services/analytics/EDAS]

- Index (climate index)

Term used to refer to climate properties that are not measured in the field or calculated by climate models but rather that are calculated or derived from climate variables such as temperature and precipitation. Examples include the number of growing degree-days, freeze-thaw cycles, and the drought code index. (see variable)

- netCDF (Network Common Data Form)

Machine-independent, self-describing binary data format (unidata.ucar.edu/software/netcdf/).

- OpenAPI

The OpenAPI Specification is a specification for describing APIs implemented according to Representational State Transfer (REST) principles.

- Profile

Set of one or more base standards and - where applicable - the identification of chosen clauses, classes, subsets, options and parameters of those base standards that are necessary for accomplishing a particular function [ISO 19101, ISO 19106].

- STRATUS (Synchronization Technology Relating Analytic Transparently Unified Services)

NASA project offering an integrative framework presenting a unified API and workflow orchestration for varied climate data analytic services.

- Variable (climate variable)

The term climate variable is used to refer to a variable that can be measured directly in the field (at meteorological stations for example) or that is calculated by climate models. (See Index)

- Xarray

Xarray is an open source project and Python package focused on multi-dimensional arrays. It includes a library of domain-agnostic functions for advanced analytics and visualization with these data structures. (Xarray [http://xarray.pydata.org/en/stable/])

3.1. Abbreviated terms

- ACL Access Control List
- ADES Application Deployment and Execution Service
- AOI Area Of Interest
- AP Application Package
- API Application Programming Interface
- BER Biological and Environment Research
- CCCS Canadian Center for Climate Services
- CERFACS Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
- CRIM Computer Research Institute of Montreal
- CWL Common Workflow Language
- CWT Compute Working Team
- DKRZ Deutsches Klimarechenzentrum
- DOE U.S. Department of Energy
- ECCC Environment and Climate Change Canada
- EMS Execution Management Service
- EO Earth Observation
- EOC Earth Observation Clouds
- EP Exploitation Platform
- ER Engineering Report
- ESA European Space Agency
- ESGF Earth System Grid Federation
- HPC High-Performance Computing
- IdP Identity Provider
- IPCC Intergovernmental Panel on Climate Change
- IT Information Technology
- JSON JavaScript Object Notation
- KNMI Koninklijk Nederlands Meteorologisch Instituut
- LLNL Lawrence Livermore National Laboratory
- MEP Mission Exploitation Platform
- ML Machine Learning
- NASA National Aeronautics and Space Administration
- NCCS NASA Center for Climate Simulation
- NRCan Natural Resources Canada

- PAVICS Power Analytics and Visualization for Climate Science
- PCIC Pacific Climate Impacts Consortium
- OWS OGC Web Services
- REST REpresentational State Transfer
- SWG Software Working Group
- TB Testbed
- TEP Thematic Exploitation Platform
- TIE Technology Integration Experiments
- TOI Time Of Interest
- URI Uniform Resource Identifier
- URL Uniform Resource Locator
- VM Virtual Machine
- WFS Web Feature Service
- WPS Web Processing Service

Chapter 4. Overview

Section 5 briefly recaps the work done in Testbed-14 regarding the execution and deployment of applications and workflows. The section also reviews the potential implications for Testbed-15.

Section 6 describes the ESGF Compute Challenge, a community effort into which this activity has been coordinated.

Section 7 presents the solution developed in this activity. It documents the execution management system and its extension to the ESGF Compute Working Team API.

Section 8 introduces the workflows and application packages created for interoperability experiments with ESGF API. It also lists the relevant climate processes and data used.

Section 9 lists issues of interest and findings that the work performed by the participants allowed to identify.

Annex A provides a client-side Python sample through ESGF CWT API.

Annex B provides a CWL file for the *ice_days* process of Finch WPS 1.0 provider. Finch is a component used to compute climate indices.

Annex C provides a CWL file for the subset process of NASA EDAS provider, exposed through ESGF CWT API.

Annex D provides a JSON file, containing all required inputs and outputs for a ESGF CWT API process.

Annex E provides a JSON file for WPS 1.0 workflow of climate processes.

Annex F provides a JSON file for the request body of a WPS 1.0 workflow.

Annex G provides a CWL file for the WPS 1.0 to CWT workflow using LLNL AIMS WPS provider.

Annex H provides a CWL file for the WPS 1.0 to CWT workflow using NASA EDAS WPS provider.

Chapter 5. Testbed 14 Recap

Section 5 presents key elements of Testbed-14 relevant to this work, such as common architecture, interfaces and implementations. In general, the reader is invited to refer to the following Engineering Reports produced in the EOC thread.

- OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report [1]
- OGC Testbed-14: Application Package Engineering Report [2]

Additionally, the reports below offer additional insights supporting challenges of the Earth System Grid Federation and helping scope specifications for next generation architecture. These topics are very briefly covered in the discussion section and are introduced as recommended future work in the summary of this report.

- OGC Testbed-14: Federated Clouds Engineering Report [3]
- OGC Testbed-14: Machine Learning Engineering Report [4]

5.1. Architecture

The figure below is taken from the Testbed-14 ADES & EMS Results and Best Practices ER [1] and shows the architecture adopted. The main building blocks included the Client, the Execution Management Service (EMS), the FEDEO OpenSearch gateway and an Application Deployment and Execution Service (ADES). While OpenSearch, ADES and the Client play an important role in the EOC thread of Testbed-14, they are not really considered in the current report. This report focuses on an extension to the EMS allowing support of a large variety of applications types. The desired outcome is a backward compatibility with existing infrastructure, all while extending compatibility with process providers using their own interfaces.

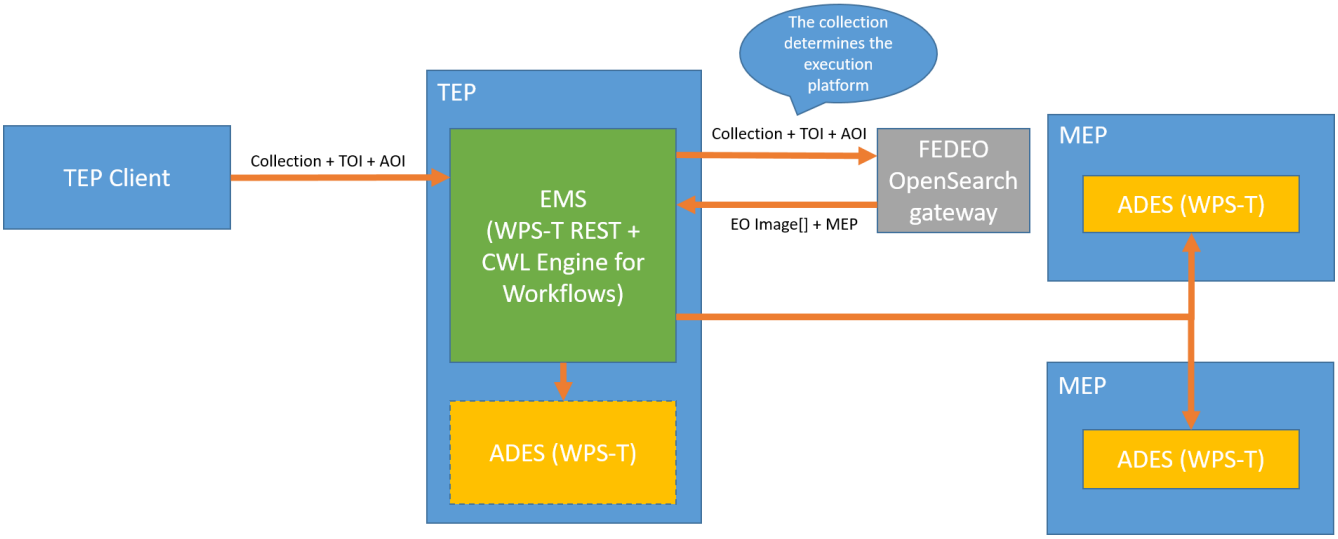

Figure 1. Final Testbed-14 EOC Thread Architecture

5.1.1. Interfaces

This subsection presents the WPS interface used in the execution components, as well as the packaging and chaining of applications.

WPS-T 2.0 REST/JSON

In Testbed 14, a RESTful protocol for the WPS interface was designed. The interface serves both the ADES and EMS components as they share the same functionality set. The complete specification defined using OpenAPI can be found on Github [https://github.com/opengeospatial/D009-ADES_and_EMS_Results_and_Best_Practices_Engineering_Report/blob/master/code/ades_wpst.json]. This interface proposes how to deploy, list, execute and monitor processes, as well as getting back the process result. While the specification covers other aspects, these are the main operations on which this report focuses.

Application Package

The Application Package ER [2] defined how the applications should be packaged. It introduced a novel way to capture how the parameters could be passed to the application and how the results are retrieved. A WPS-T DeployProcess document defines the application's inputs and outputs, as well as the execution unit. The novel way to define that execution unit is to add a Common Workflow Language (CWL [5]) file that is exactly conceived to describe how to execute Docker applications and how parameters and results can be provided and retrieved. Docker is a set of software products for creating, deploying, and running applications inside containers. Each WPS input can precisely map to the Docker command line arguments. The output can be easily retrieved by mounting a volume inside the Docker container where the output is expected to be created. The *DeployProcess* document still supports defining directly the Docker image as an execution unit, but in that case *ows:metadata* must be provided to emulate what is contained in the CWL. The current report describes the modification to the CWL execution unit and shows how it can be used to easily make the EMS more versatile.

Application Chaining

Application chaining is an important part of the Testbed-14 project. Proposed interoperability experiments involve simple chaining, where in a workflow composed of application A and B, output of A is being fed into B. Both applications are packaged as Docker images and their execution is forwarded to an ADES. That forwarding is dynamic. Based on the data source, the system determines where application execution should occur, so that the data does not have to be moved around. That implies that at runtime, an ADES is targeted based on the data source, hence the application is deployed to that ADES. It is then executed before deployment of the second application, parametrized with the output of the first one.

5.1.2. Implementation

For this work, CRIM's implementation is considered. It is already partly committed into ESGF code base and is being extended to support interoperability with ESGF deployed services, as described in Section 6. The EMS implementation is based on components of the Open Source software framework Birdhouse [6]. The Twitcher [https://github.com/Ouranosinc/twitcher] component acts as a Policy Enforcement Point (PEP). It has been extended to offer a WPS 2.0 JSON proxy to WPS 1.0 endpoints, and to comply to the EMS API which involves adding the dynamic deployment of processes and CWL workflow capabilities. The Magpie [https://github.com/Ouranosinc/Magpie] component is used as an adapter to manage ACLs of deployed processes and process permissions (WPS requests) for a given user's credentials. The following component diagram shows the relation between Twitcher and Magpie.

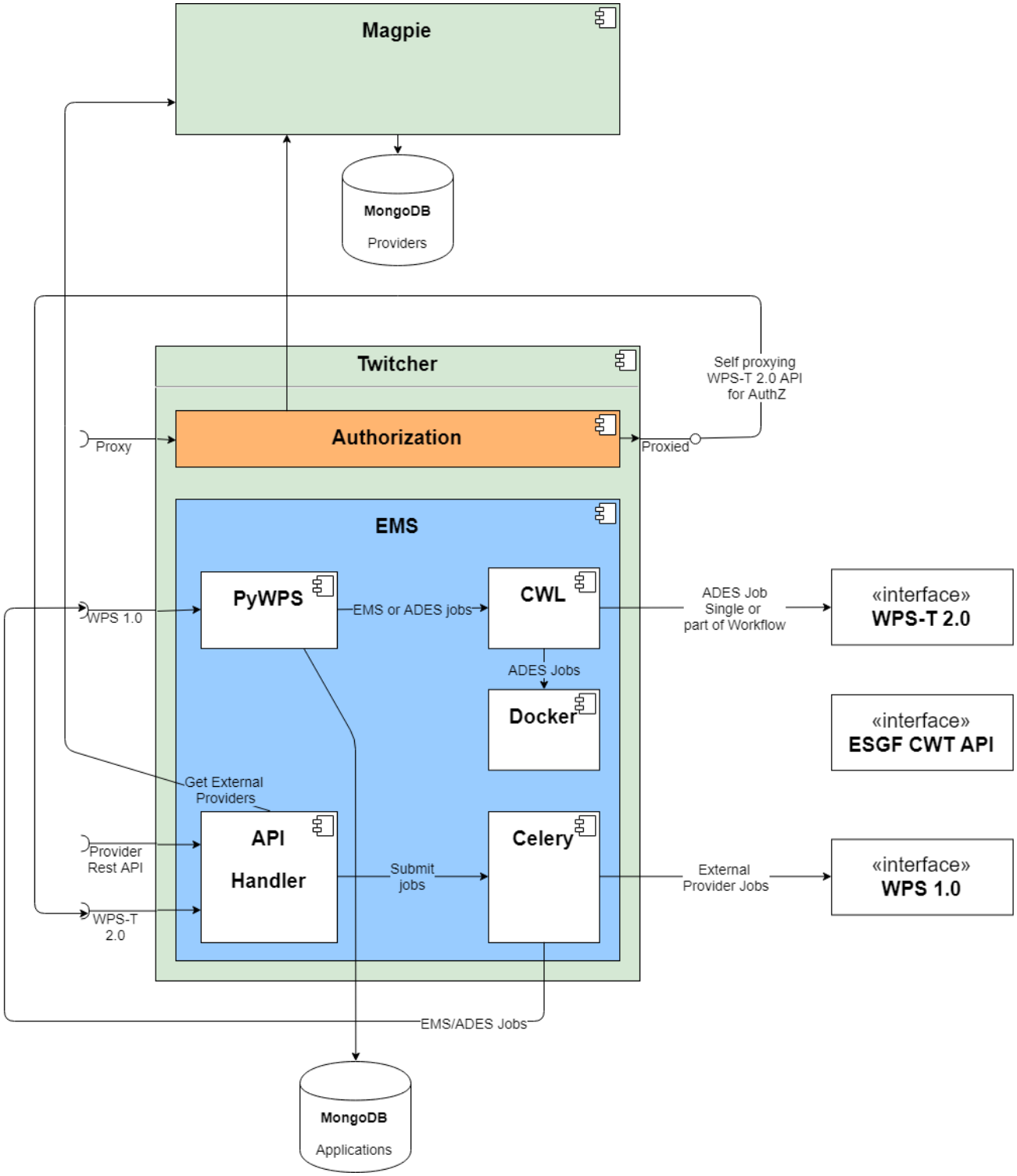

Figure 2. EMS component developed by CRIM and proposed extension to other WPS interfaces.

Incoming requests initially hit the top-left proxy connector, that first performs an authorization using the Magpie component. If the user is allowed, the request is forwarded by the proxy to the bottom left WPS-T interface, which implements the OpenAPI specification described above. Submitted jobs are managed in a queue implemented using Celery [7]. The handler forwards the job to a legacy WPS 1.0 PyWPS server which performs all the parameters validation. All jobs are then processed by the CWL engine which can execute the job in situ, if playing the ADES role, or an EMS forwarding it to the proper ADES based on the data source.

Chapter 6. ESGF Compute Challenge

Section 6 presents the Earth System Grid Federation (ESGF) collaborative effort and introduces its recent Compute challenge. The section also presents some of ESGF requirements, priorities, solutions and contributed implementations.

6.1. Background Information

Led by the Office of Biological and Environmental Research (BER) of the U.S. Department of Energy (DOE), ESGF is an international multi-agency federation that develops, deploys, and maintains software to facilitate advancements in geophysical science. ESGF's open-source, operational code base disseminates model simulation, observational, and reanalysis data for research assessments and model validation via secure storage and dissemination of petabytes of data.

Addressing "big data" challenges in Earth system research, ESGF is a collaboration of numerous computer scientists, data scientists, and climate researchers. The federation houses an enormous database of global observational and simulation data — more than five petabytes — and manages the high-performance computing (HPC) hardware and software infrastructure necessary for scientific climate research. In the nearly two decades since its launch, ESGF has grown to serve 25,000 users on six continents [8]. More details can be found in ESGF Brochure [https://esgf.llnl.gov/esgf-media/pdf/2017-ESGF-Brochure.pdf] or in Discussion for other supporting material and citations.

6.1.1. ESGF Mission

The ESGF mission is to:

- Support current CMIP6 activities, and prepare for future assessments
- Develop data and metadata facilities for inclusion of observations and reanalysis products for CMIP6 use
- Enhance and improve current climate research infrastructure capabilities through involvement of the software development community and through adherence to sound software principles
- Foster collaboration across agency and political boundaries
- Integrate and interoperate with other software designed to meet the objectives of ESGF
- Create software infrastructure and tools that facilitate scientific advancements

6.1.2. ESGF Priorities

As of early 2019, the priorities of ESGF are as follows:

- Containerized architecture
- OAuth 2.0 deployments
- CMIP6 data replication
- Next-generation search services
- Scalability on multiple fronts

- Machine learning tools
- Compute nodes challenges

On that last element, the Compute Working Team (CWT) currently conducts iterative deployments and tests to stand up multiple production-ready, officially certified ESGF compute nodes. This effort includes data selection, security scans, API development, scalability tests, documentation updates, and regular status reports. The current Engineering Report is one of the contributions to this challenge.

6.2. Compute Challenge Implementations

The ESGF Compute Challenge participants completed various experiments and integrations. The EMS implementation described in this report is based on components from both Birdhouse and PAVICS. The Solution described in this report focuses on providing access to deployed ESGF CWT API processes and providers in application packages and workflows. Future work includes exposing application packages through ESGF CWT API, either manually or dynamically. Below is a list of the various systems and frameworks involved in the Compute Challenge.

- AIMS [https://computation.llnl.gov/projects/aims-analytics-and-informatics-management-systems] - Analytics and Informatics Management Systems, LLNL.
- Birdhouse [http://bird-house.github.io/] Framework, DKRZ/CEDA/IPSL. See [9].
- C4I [https://climate4impact.eu/impactportal/general/index.jsp] - Climate4Impact, CERFACS/IPSL/KNMI/CMCC.
- EDAS [https://www.nccs.nasa.gov/services/analytics/EDAS] - Earth Data Analytic Services Framework, NASA Goddard SFC. See [10].
- OpenVisus [https://github.com/sci-visus/OpenVisus], University of Utah. See [11].
- Ophidia [https://github.com/OphidiaBigData/ophidia-analytics-framework] Analytics Framework, CMCC. See [12].
- PAVICS [https://ouranosinc.github.io/pavics-sdi/] - Platform for the Analysis and Visualization of Climate Science, Ouranos/CRIM. See [13].

EDAS can be used as an analytics engine for the ESGF through STRATUS, an integrative framework developed at NASA NCCS. STRATUS presents a unified API and workflow orchestration for varied climate data analytic services. See Discussion section for more information on STRATUS.

As of early April, only AIMS and EDAS offered endpoints implementing the ESGF Compute Working Team API. Interoperability experiments conducted on these providers for this work are documented in Section 7 and Section 8, while the API is briefly described below.

6.3. ESGF Compute Working Team API

The ESGF Compute Working Team (CWT) is working to improve interoperability and compute capabilities within the federation using Web services technology. The OGC WPS standard has been selected to ensure machine-to-machine interoperability. A reference server implementation of a compute node [https://github.com/ESGF/esgf-compute-wps] is offered. Certification datasets are listed in a

GoogleDocs [https://docs.google.com/document/d/1pxz1Kd3JHfFp8vR2JCVBfApbsHmbUQQstifhGNdc6U0/edit?usp=sharing] living document. The process requirements are defined on GitHub [https://github.com/ESGF/esgf-compute-api/blob/devel/docs/source/cwt.compat.rst]. These requirements are partially duplicated below for convenience.

6.3.1. Inputs

The *datainputs* parameter consists of the following three types.

- Domain
- Variable
- Operation

Domain

This WPS input should use the identifier domain. The input is passed an array of domains that are comprised of one or more dimensions.

- id [Required]
- mask [Optional]
- One or more dimensions keyed using a descriptive identifier. [Required]

Variable

This WPS input should use the identifier variable. The input is passed an array of variables that define all inputs for the process.

- id [Required] - Can be extended with a | followed by an identifier that will be used to reference the variable
- uri [Required]
- domain [Optional]

Operation

This WPS input should use the identifier operation. The input is passed an array of operations.

- name [Required]
- input [Required] - List of inputs
- result [Optional] - Name that can be referenced by other operations when creating workflows
- domain [Optional]
- axes [Optional]
- gridder [Optional]
- Zero or more additional parameters [Optional]

6.3.2. Output

The WPS process should only have a single output whose identifier is output.

- uri [Required]
- id [Optional]
- domain [Optional]
- mime-type [Optional]

Chapter 7. Solution

Section 7 presents the solution adapting Testbed-14 implementations towards interoperability with ESGF compute nodes. The section revisits the architecture, introduces two new profiles and offers code samples for integration of deployed services into workflows.

7.1. Architecture

The goal of the extension presented in this report is to improve the EMS in order to increase compatibility across existing systems. The EMS provided for Testbed-14 offers a WPS-T 2.0 interface, yet should be able to run multiple application types and use them in heterogeneous workflows. Using a single interface should allow execution of a large array of existing applications and, moreover, use them inside workflows. Two new application types are considered in this project. The first is for backward compatibility and consists of execution of existing process served by WPS 1.0 endpoint. The second is to broaden furthermore the application scope by covering the ESGF Compute Working Team (CWT) API.

Multiple approaches have been considered to achieve this goal. The first approach is to keep the EMS as is, package every type of application into Docker images, and provide alongside them a CWL describing invocation mechanisms. However, packaging existing providers would yield huge Docker images with multiple processes. This could simply prove impossible for external providers for which the code is unavailable. The second approach, much simpler, however defeats one of the principles of Testbed-14 which is to bring the application to the data. This approach consists of packaging all the information required to make a standard WPS 1.0 or ESGF CWT API request.

A DeployProcess document is still provided but two new profile names have been introduced, *wpsApplication* and *ESGFWpsApplication*, that require fewer elements as everything needed can be extracted from the existing endpoints. In the WPS 1.0, the CWL file can even be generated transparently at deployment time since the parameters mapping is trivial. The CWL file itself allows to specify execution requirements so that the engine can change the execution unit and perform a classic WPS 1.0 or ESGF CWT API execute request, without a prior deployment request.

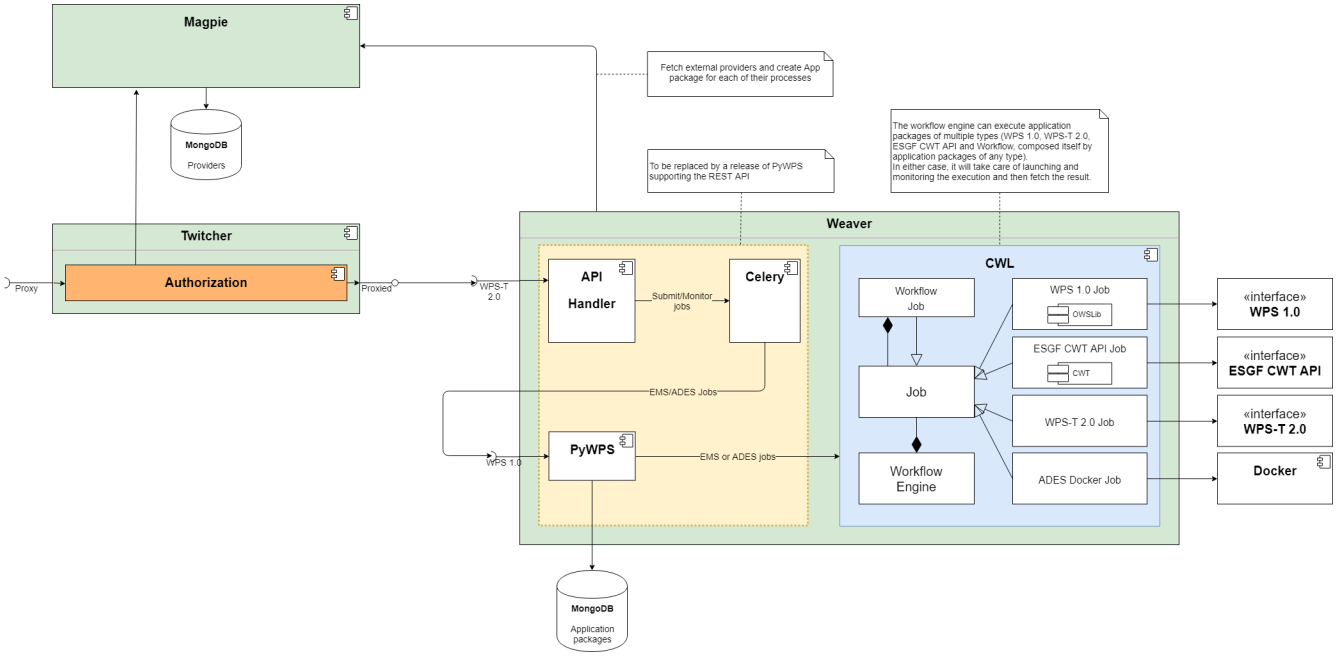

Figure 3. New EMS component called Weaver and its workflow packages.

As described in Section 5, in Testbed-14, CRIM's EMS implementation was all contained in a component named Twitcher. This component is part of the Open Source software framework Birdhouse [9]. In the current solution, Twitcher is separated into two components so that each one can focus primarily on its own role and make the architecture simpler. The previous diagram resumes the situation. Twitcher now operates as regular proxy and a Security Enforcement Point (PEP), alongside the permission provider, Magpie. The WPS-T 2.0 REST interface and the CWL engine are moved to a new component named Weaver [https://github.com/crim-ca/weaver].

7.2. Application Package

This subsection describes the various application profiles developed and introduces their integration into workflow. Subsetting and climate indices calculation are also introduced as key processes relevant to ESGF.

7.2.1. Process Deployment

The DeployProcess document conforms to the existing API but requires fewer optional elements. Using the wpsApplication deployment profile name, it only requires the process id and an execution unit referencing a WPS 1.0 endpoint. Inputs, outputs and CWL file can be implicitly deduced.

Below is a JSON file deploying a WPS 1.0 endpoint offering computation of climate indices. In this case, the WPS component named Finch [https://github.com/bird-house/finch] exposes the xclim [https://xclim.readthedocs.io/en/latest/readme.html] Python library developed by Ouranos and funded in part by Environment and Climate Change Canada (ECCC). This library is based on Xarray and benefits from the parallelization provided by Dask. XClim has for objective to make it as simple as possible for users to compute indices from large climate datasets, and for scientists to write new indices with very little boilerplate. The example below returns *Ice Days*, which takes into account the number of days in which the temperature never rises above 0 degrees Celsius and stays below freezing point.

JSON file for a WPS 1.0 process deploy request

```
{
    "processDescription": {
        "process": {
            "id": "Finch_IceDays"
        }
    },
    "executionUnit": [
        {
            "href":
"https://finch.crim.ca/wps?service=WPS&request=describeprocess&version=1.0.0&identifie
r=ice_days"
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/wpsApplication"
}
```

For the ESGF CWT processes, the CWL must be provided as a reference or inline, since parameter mapping is more involved than for WPS 1.0. This is discussed further in the Discussion section. Below is a JSON file deploying a WPS 1.0 endpoint enforcing the ESGF CWT API. In that case, the example is a subsetting process offered by NASA EDAS. This process is conceptually similar to setting an area and time of interest to Earth observation data, and returning the extracted data.

JSON file for an ESGF CWT process deploy request

```
{
    "processDescription": {
        "process": {
            "id": "nasa_esgf_subset"
        }
    },
    "executionUnit": [
        {
            "unit": {
                <cwl file content show below>
            }
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/ESGFWpsApplication"
}
```

7.2.2. Workflow Integration

The CWL file is modified so that the CWL engine can instantiate the appropriate job implementation. To that effect, the hints section of the CWL file are used. This replaces the traditional *DockerRequirement* value for extensions requirements, which are *WPS1Requirement* and *ESGF-CWTRequirement*. Under that key, a dictionary containing all the parameters required to make an execute request to WPS 1.0 provider is added. The only difference with the CWL provided

during Testbed-14 is the hints section declaring the WPS1Requirement and two parameters: the provider endpoint and the process which is wrapped. The file format is also now enforced in the CWL file. Below, a CWL example file describes one of the climate processes for the WPS 1.0 provider. A full example of the CWL file, containing inputs and outputs, can be found in Annex B.

Excerpt of CWL file for the ice_days process of Finch WPS 1.0 provider

```
{
  "cwlVersion": "v1.0",
  "$namespaces": {
    "edam": "http://edamontology.org/"
  },
  "class": "CommandLineTool",
  "hints": {
    "WPS1Requirement": {
      "process": "ice_days",
      "provider": "https://finch.crim.ca/wps"
    }
  },
  "inputs": {<...>},
  "outputs": {<...>}
  }
}
```

When the CWL engine encounters the file presented above, it recognizes the WPS1Requirement thus creating a WPS 1.0 Job. That job uses the same interface as the WPS-T 2.0 Job, but rather than deploying and executing an application on a remote ADES, it calls the WPS 1.0 execute request of the provider and process given in parameters. The result is then fetched similarly to the ADES implementation. In the following CWL excerpt, the *ESGF-CWTRequirement* triggers the creation of a CWT Job that will use the ESGF-compute-api Python package to run the process with a proper parameters mapping. Once again, there is no deployment involved and once the process execution completes, the result is fetched. A full example of the CWL file, containing inputs and outputs, can be found in Annex C.

Excerpt of CWL file for the NASA EDAS Subset process

```
{
    "cwlVersion": "v1.0",
    "class": "CommandLineTool",
    "hints": {
        "ESGF-CWTRequirement": {
            "provider": "https://edas.nccs.nasa.gov/wps/cwt",
            "process": "xarray.subset"
        }
    },
    "inputs": {<...>},
    "outputs": {<...>}
    }
}
```

Chapter 8. Applications and Workflows

Section 8 documents interoperability experiments conducted between CRIM's EMS, LLNL AIMS and NASA EDAS endpoints in context of ESGF Compute Challenge. The section presents execution body samples and workflows.

8.1. ESGF CWT Applications

Currently, only a small portion of the ESGF WPS processing is implemented in Weaver. This is largely due to the fact that ESGF WPS inputs are nested, and this nested structure must be translated to a flat one to correspond to standard WPS inputs or to Weaver's inputs. Some of these nested parameters are easy to implement because they are always the same across each process, but others cannot be automatically queried. A full example JSON file, containing all required inputs and outputs for the process, can be found in Annex D. Below is an excerpt of an execution body.

Excerpt of a JSON file of a CWT process execute body

```
{
  "mode": "async",
  "response": "document",
  "inputs": [
      {
         "id": "files",
         "href":
"https://boreas.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/nrcan/nrcan_nort
hamerica_monthly/tasmin/nrcan_northamerica_monthly_2015_tasmin.nc"
      },
      {
         "id": "variable",
         "data": "tasmin"
      },
      {
         "id": "api_key",
         "data": "{{ api_key }}"
      },
      {
         "id": "time_start",
         "data": "0"
      },
      {
         "id": "lat_start",
         "data": "60"
      },
      <...>
  ],
  "outputs": [
    {
      "id": "output",
      "transmissionMode": "reference"
    }
  ]
}
```

The *files* and *variable* parameters correspond to the ESGF CWT API *variable* input, as described in ESGF section. The *api_key* must be obtained by creating an account on the Lawrence Livermore National Laboratory (LLNL) website. The other inputs are a flat representation of the ESGF 'Domain' input. So far, the *time* and *lat/lon* attributes are present in every NetCDF file encountered so they are easy to implement and re-use for each process. The ESGF processes successfully tested are *CDAT.aggregate* and *CDAT.subset*, deployed on AIMS2 servers.

8.2. Application Chaining

For the application chaining, the CWL engine is now able to process all application types only by instantiating the proper job type. To demonstrate that interoperability, two workflows have been

produced and will be presented in this section. Beforehand, utility applications will be introduced.

8.2.1. Utility Applications

This concept has been added to further improve compatibility. They are small Python applications, still packaged as CWL, that can make some adaptation between related type. For example, some applications yield JSON files containing an array of NetCDF files. The JSON output is therefore incompatible with an application wanting NetCDF files as inputs. The utility application can be chained between the two. This way, the CWL engine feeds the JSON output into the utility apps that will provide an array of NetCDF files, ready to be consumed by the next application. These applications are really lightweight because the CWL file is only wrapping a Python function already inside the Weaver EMS component. Below is a sample CWL file of the JSON to NetCDF.

CWL file for the JSON to NetCDF utility application

```
#!/usr/bin/env CWL-runner
CWLVersion: v1.0
$namespaces:
  iana: "https://www.iana.org/assignments/media-types/"
  edam: "http://edamontology.org/"
class: CommandLineTool
baseCommand: python
arguments: ["-m", "weaver.processes.builtin.jsonarray2netcdf", $(runtime.outdir)]
inputs:
 input:
   type: File
   format: iana:application/JSON
   inputBinding:
     position: 1
outputs:
 output:
   format: edam:format_3650
   type:
     type: array
     items: File
   outputBinding:
     glob: "*.nc"
```

8.2.2. Workflow Chaining WPS 1.0 Processes

The first workflow consists of a subsetter and a climate indices process. The deploy body is exactly the same as in Testbed-14 as shown here. It contains the deployment profile name indicating that it is a workflow, a process id and a CWL reference containing the workflow.

JSON file for the WPS 1.0 workflow

```
{
    "processDescription": {
        "process": {
            "id": "WorkflowSubsetIceDays",
            "title": "Workflow of Subset and Ice Days",
            "abstract": "Workflow that first executes a bounding box subset of a
region and afterwards calculates days with ice within the obtained region."
        }
    },
    "executionUnit": [
        {
            "href": "tests/functional/application-
packages/workflow_subset_ice_days.CWL"
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/workflow"
}
```

The CWL is also built the same way as in Testbed-14. It contains the class indicating that it is a workflow, the workflow inputs and outputs and the steps referencing CWL files. This workflow contains two WPS 1.0 steps and one utility *json2nc* step converting the output type of the first step into an acceptable type for the third one as introduced in the previous section. Below is an excerpt of the CWL files, where some details were removed for concision. A complete listing of this CWL file can be found in Annex E. To execute that workflow, the same execute request body as in Testbed-14 is used. The complete execute request body for the workflow can be found in Annex F.

Excerpt of a CWL file of the WPS 1.0 workflow

```
{
    "cwlVersion": "v1.0",
    "class": "Workflow",
    "requirements": [
        {
            "class": "StepInputExpressionRequirement"
        }
    ],
    "inputs": {<...>},
    "outputs": {<...>},
    "steps": {
        "subset": {<...>},
        "json2nc": {<...>},
        "ice_days": {<...>}
    }
}
```

In Annex F, the *tasmax* input provides a reference to a required maximum temperature NetCDF file which is shown on the left in the image below. The *lat/lon* inputs are required as well by the

subsetter process, and finally the *freq* input is mapped to the ice days process. The subsetter performs its task using the provided bounding box, the JSON output is decapsulated by the *json2nc* step, and the NetCDF file is then fed to the last process which calculates the ice days over the provided region. The result of this workflow on Canada statistically downscaled climate scenarios is shown on the right in the image below.

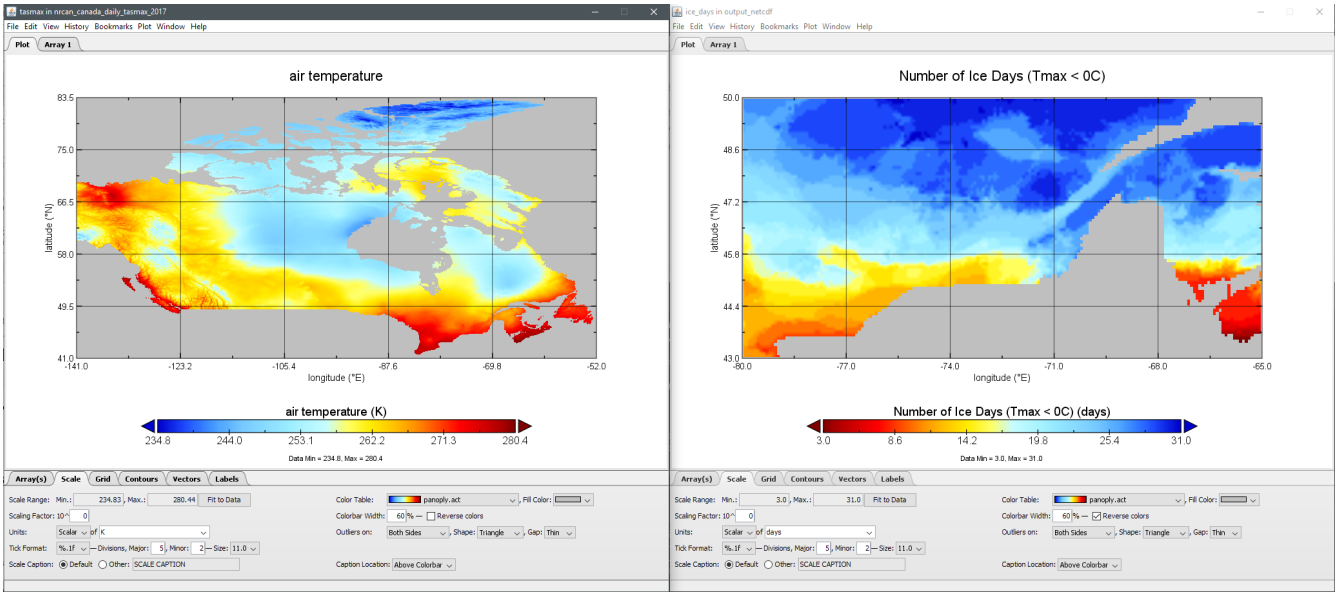

Figure 4. Image showing workflow input / output example.

8.2.3. Workflow Linking two Subsetters of CWT and WPS 1.0 Types

The second workflow has been tried both ways, first subsetting by CWT then by WPS 1.0, and using the opposite order, WPS 1.0 first then feeding the CWT interface. As for the first workflow, the deploy body is unchanged from the previous Testbed (except for the CWL file name) and is omitted here. The first CWL, detailed in Annex G, shows that the WPS 1.0, *crim_subset*, is linked to the second step, *llnl_subset*, a CWT process executed on the AIMS2 server at LLNL.

The second CWL file, detailed in Annex H, shows the opposite, this time using the CWT interface of the NASA server, *nasa_subset*, to feed the WPS 1.0 process, *crim_subset*. In this workflow, a utility application is also used to convert the file type obtained from the *nasa_subset* step to a string type required by the *crim_subset* further supporting the usefulness of these utility applications. The result of subsetting on CMIP6 data is shown on the right in the image below.

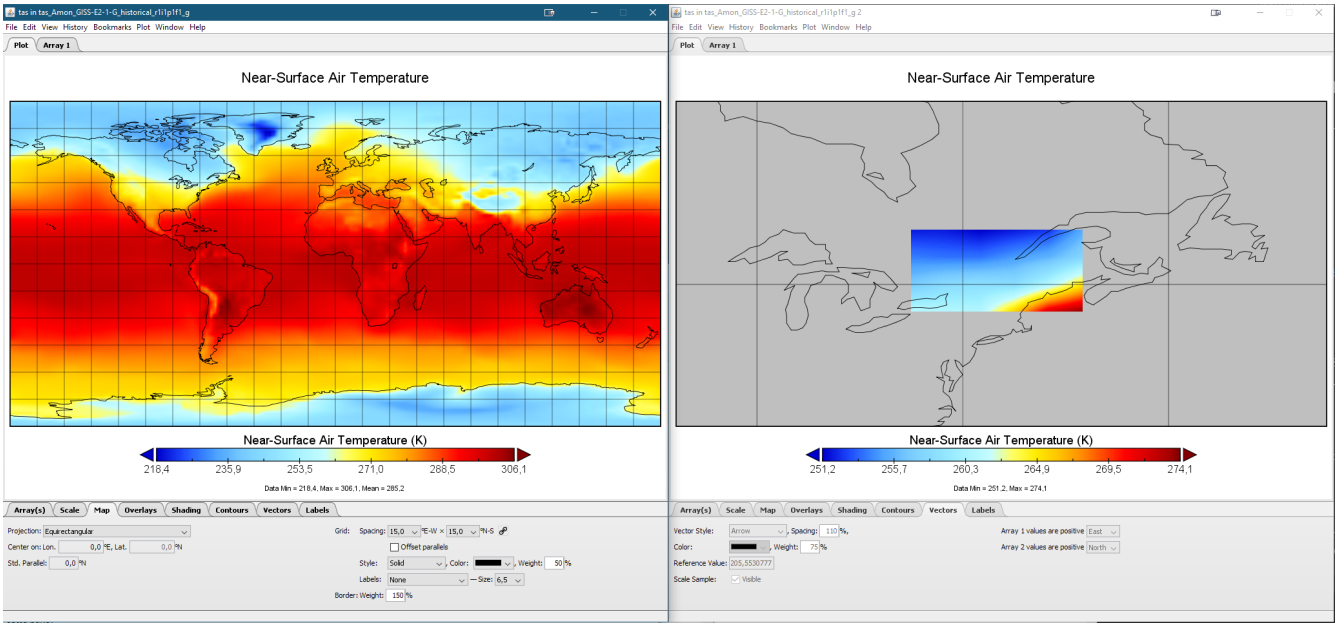

Figure 5. Image showing subsetting of CMIP6 data as processed by CRIM subsetter.

Chapter 9. Discussion

Section 9 opens up the discussion by further elaborating on findings or on raising open issues from the work.

9.1. Application and Process Terminology

There is a need to (re-)establish a common vocabulary for the software artefacts involved. For instance, the terms *Application*, *Application Package*, *Process*, *Service* and *Endpoint* lend themselves to be used interchangeably. The potential confusion is even more important in context of *Workflows* that can chain applications and execute deployed processes.

9.2. Transition from WPS 1.0 to WPS 2.0

Current adoption of WPS 2.0 standard is still very limited. For instance, 52°North implementations such as javaPS [https://github.com/52North/javaPS/releases/tag/v1.2.0] are based on OGC Web Processing Service standard version 2.0; PyWPS [https://pywps.readthedocs.io/en/master/] server is currently only planning support for this version. In the near future, one can still expect to see new WPS 1.0 endpoints appear. By enabling support of pre-deployed WPS 1.0 services into a WPS 2.0 workflow environment, and onwards to WPS 3.0, the Solution presented in this report has the potential to facilitate a transition phase for legacy deployments. This relative ease to provide WPS 2.0 RESTful interfaces comes at the expense of the increased software complexity involved with an EMS/ADES implementation such as Weaver.

It has been shown with the previous workflows that interoperability between WPS 1.0 and ESGF CWT interface can be achieved with the Weaver component. If climate processes could have been available as a Docker application during the short project time frame, a workflow composed of all WPS 1.0, ESGF CWT and WPS-T 2.0 interface could have been tested also.

9.3. From Docker Image to Application

In the process of adapting the EMS to climate applications, it became clear that having a Docker image of an application is not a sufficient condition to consider it an *Application Package*. Docker images are often packaged with ease of installation and deployment in mind, but not runtime considerations such as execution. In PAVICS [13] and Birdhouse [9], climate processes are packaged as Docker images, alongside a WPS server. These processes are intended to be run as a service, so that requests are made to it and result are fetched from it. The Docker container is deployed and then is always running. To convert this service into an *Application Package* as described in [2], the processes would need to be packaged without any web server, database or file server. The Docker image must define a *run* command that map inputs directly to the process function. The result must then be extracted from the container after it has been run. It is a completely different paradigm that require careful consideration to define clear entry points to the executable code.

9.4. CWL File Formats

In CWL, tools and workflows can take *File* types as input and produce them as output. This

specification documents the use the tool and allows simple type-checking when creating parameter files. It is possible to reference existing ontologies, like EDAM, or reference a local ontology. Additional investigation for creation of such an ontology for EO and climate is required, and recommended as future work in the summary.

9.5. Metalinks

One recommendation from OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report [1] is to use Metalinks for multiple outputs. During the ESGF Compute Challenge, Ouranos and DKRZ produced an implementation that adds *MetaFile* and *MetaLink* classes, code that has been merged in PyWPS [https://github.com/geopython/pywps/pull/466]. The intent is to improve user experience with respect to WPS outputs storing multiple files under the same identifier. For example, when computing an indicator on an ensemble of climate simulations, the output is going to be a list of files. At the moment, a usual way to send that back to the user is to either zip these files and provide the reference to the zip file, or create a txt file storing the list of references.

In this implementation, a process developer declares an output as a *FORMATS.METALINK*, instantiates a *MetaLink* object and fills it with all the output files and their metadata (name, description). The server returns an XML file matching the metalink schema. Metalink files are recognized and individual files downloaded and converted into objects. The *MetaLink* format opens up interesting possibilities, including distributing files as torrents or use of a checksum. Additionally, impacts of *Metalink* support in workflow environments is to be investigated.

9.6. NASA NCCS STRATUS

NASA Stratus Framework provides a set of standards and APIs to facilitate the integration of disparate analytic services into unified workflows with a common interface. The Stratus framework has been partially implemented at NCCS and a limited deployment is in the initial stages of testing and security review. As the framework addresses several challenges of EO platforms, it is presented to the reader of this report as a solution potentially complementing ESA Thematic Exploitation Platform (TEP) open architecture. Below are STRATUS primary requirements, as listed in its white paper [https://github.com/nasa-nccs-cds/stratus/blob/master/docs/STRATUS-WhitePaper-1.0.pdf]:

- Modular Endpoints: Requires a common endpoint API that can be used to wrap any analytic operator and expose it as an instantiation of a singular analytic module interface.
- Modular Layers: Requires a technology-agnostic architectural layer specification adhering to a common API, which can be instantiated using a wide range of applicable technologies.
- A common language: Requires a common request-response language for describing workflows that can be assimilated and understood by all of the architectural layers and endpoints.

9.7. Applicability to Machine Learning

In Testbed-15 Machine Learning tasks, there is a specification to “continue the work of cloud computing from Testbed 13, 14 with work on WPS and possibly CWL (Common Workflow Language), and continue the LiDAR best practices work from Testbed 14”. This applies for these deliverables:

- D100 Petawawa cloud mosaicking Machine Learning model
- D101 Petawawa land cover classification Machine Learning model
- D102 New Brunswick forest Machine Learning model

It is to be noted that work documented in OGC Testbed-14: Machine Learning Engineering Report [4] presents a Dockerized, self-contained application package of PyTorch, including helper libraries. The work presented in the current report is therefore seen as potentially complementary to the ML common architecture to be defined in the Machine Learning thread.

9.8. Generation of CWL Wrappers for ESGF CWT API

In this work, a CWL descriptor such as the one in Annex B is generated automatically using *DescribeProcess* on a pre-deployed WPS 1.0 service. The process description is expected to explicitly define all inputs and outputs. In the case of ESGF CWT API compliant process, *DescribeProcess* effectively lists *Domain*, *Variable* and *Operation* inputs, as specified in the API and described in Section 6. The multiple values of these parameters have to be passed separately in a JSON file. As such, they values are not automatically available for the CWL generation. A CWL file has to be manually edited to explicitly add and map the all inputs. An example can be seen in Annex C. To remediate this, a specific set of inputs, or profile, could be defined as a shared interface between deployed ESGF processes. Additionally, the EMS could implement special logic to parse the multiple values of an ESGF CWT API compliant service.

9.9. ESGF Supporting Material

The following publications offer a thorough view on various challenges and requirements of ESGF:

- Strategie Roadmap for the Earth System Grid Federation [14]
- Big Data Challenges in Climate Science: Improving the next-generation cyberinfrastructure [15]
- Enabling Reanalysis Research Using the Collaborative Reanalysis Technical Environment (CREATE) [16]
- Requirements for a global data infrastructure in support of CMIP6 [17]
- Addressing the massive CMIP6 data science challenge through the ESGF global federation [18]

Appendix A: Sample Python code for ESGF CWT API

Sample Python code for ESGF CWT API

```
import esgf

NH = esgf.Domain(dimensions=[
  esgf.Dimension(0.0, 90.0, esgf.Dimension.values, name='latitude'),
],
name = 'd0')

tas = esgf.Variable('http://.../tas.nc', 'tas', name='v0', domains=[NH])

wps = esgf.WPS('http://.../wps')

avg = wps.get_process('averager.mv')

parameters = [
  esgf.NamedParameter('axes', 'latitude'),
]

avg.execute([tas], domain=None, parameters)
```

Appendix B: CWL file for WPS 1.0 provider

CWL file for the ice_days process of Finch WPS 1.0 provider


```
{
  "cwlVersion": "v1.0",
  "$namespaces": {
    "edam": "http://edamontology.org/"
  },
  "class": "CommandLineTool",
  "hints": {
    "WPS1Requirement": {
      "process": "ice_days",
      "provider": "https://finch.crim.ca/wps"
    }
  },
  "inputs": {
    "tasmax": {
      "default": {
        "mimeType": "application/x-netcdf",
        "schema": null,
        "encoding": "base64"
      },
      "type": {
        "items": "File",
        "type": "array"
      },
      "format": "edam:format_3650"
    },
    "freq": {
      "default": "YS",
      "type": {
        "symbols": [
          "YS",
          "MS",
          "QS-DEC",
          "AS-JUL"
        ],
        "type": "enum"
      }
    }
  },
  "outputs": {
    "output_netcdf": {
      "outputBinding": {
        "glob": "output_netcdf.nc"
      },
      "type": "File",
      "format": "edam:format_3650"
    },
```

```
    "output_log": {
      "outputBinding": {
        "glob": "output_log.*"
      },
      "type": "File",
      "format": "edam:format_1964"
    }
  }
}
```

Appendix C: CWL file for NASA EDAS

CWL file for the NASA EDAS Subset process exposed through ESGF CWT API

```
{
    "cwlVersion": "v1.0",
    "class": "CommandLineTool",
    "hints": {
        "ESGF-CWTRequirement": {
            "provider": "https://edas.nccs.nasa.gov/wps/cwt",
            "process": "xarray.subset"
        }
    },
    "inputs": {
        "files": "File",
        "variable": {
            "type": "string"
        },
        "time_start": {
            "type": "float",
            "default": null
        },
        "time_end": {
            "type": "float",
            "default": null
        },
        "time_crs": {
            "type": "string",
            "default": null
        },
        "lat_start": {
            "type": "float",
            "default": null
        },
        "lat_end": {
            "type": "float",
            "default": null
        },
        "lat_crs": {
            "type": "string",
            "default": null
        },
        "lon_start": {
            "type": "float",
            "default": null
        },
        "lon_end": {
            "type": "float",
            "default": null
        },
```

```
        "lon_crs": {
            "type": "string",
            "default": null
        }
    },
    "outputs": {
        "output": {
            "outputBinding": {
                "glob": "output_netcdf.nc"
            },
            "type": "File"
        }
    }
}
```

Appendix D: JSON file of an ESGF CWT API execute body

JSON file of an ESGF CWT API execute body


```
{
  "mode": "async",
  "response": "document",
  "inputs": [
      {
         "id": "files",
         "href":
"https://boreas.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/nrcan/nrcan_nort
hamerica_monthly/tasmin/nrcan_northamerica_monthly_2015_tasmin.nc"
      },
      {
         "id": "variable",
         "data": "tasmin"
      },
      {
         "id": "api_key",
         "data": "{{ api_key }}"
      },
      {
         "id": "time_start",
         "data": "0"
      },
      {
         "id": "time_end",
         "data": "5"
      },
      {
         "id": "time_crs",
         "data": "values"
      },
      {
         "id": "lat_start",
         "data": "60"
      },
      {
         "id": "lat_end",
         "data": "40"
      },
      {
         "id": "lat_crs",
         "data": "values"
      },
      {
         "id": "lon_start",
```

```
            "data": "-80"
         },
         {
            "id": "lon_end",
            "data": "-60"
         },
         {
            "id": "lon_crs",
            "data": "values"
         }
      ],
      "outputs": [
        {
          "id": "output",
          "transmissionMode": "reference"
        }
      ]
    }
```

Appendix E: CWL file of the WPS 1.0 workflow

CWL file of the WPS 1.0 workflow


```
{
    "cwlVersion": "v1.0",
    "class": "Workflow",
    "requirements": [
        {
            "class": "StepInputExpressionRequirement"
        }
    ],
    "inputs": {
        "tasmax": {
            "type": {
                "type": "array",
                "items": "string"
            }
        },
        "lat0": "float",
        "lat1": "float",
        "lon0": "float",
        "lon1": "float",
        "freq": {
            "default": "YS",
            "type": {
                "type": "enum",
                "symbols": ["YS", "MS", "QS-DEC", "AS-JUL"]
            }
        }
    },
    "outputs": {
        "output": {
            "type": "File",
            "outputSource": "ice_days/output_netcdf"
        }
    },
    "steps": {
        "subset": {
            "run": "ColibriFlyingpigeon_SubsetBbox.cwl",
            "in": {
                "resource": "tasmax",
                "lat0": "lat0",
                "lat1": "lat1",
                "lon0": "lon0",
                "lon1": "lon1"
            },
            "out": ["output"]
```

```
        },
        "json2nc": {
            "run": "jsonarray2netcdf",
            "in": {
                "input": "subset/output"
            },
            "out": ["output"]
        },
        "ice_days": {
            "run": "Finch_IceDays.cwl",
            "in": {
                "tasmax": "json2nc/output",
                "freq": "freq"
            },
            "out": ["output_netcdf"]
        }
    }
}
```

Appendix F: JSON file for the WPS 1.0 workflow execute request body

JSON file for the WPS 1.0 workflow execute request body


```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "tasmax",
      "href":
"https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/birdhouse/nrcan/nrcan
_canada_daily_v2/tasmax/nrcan_canada_daily_tasmax_2017.nc"
    },
    {
      "id": "lat0",
      "data": 43
    },
    {
      "id": "lat1",
      "data": 50
    },
    {
      "id": "lon0",
      "data": -80
    },
    {
      "id": "lon1",
      "data": -65
    },
    {
      "id": "freq",
      "data": "MS"
    }
  ],
  "outputs": [
    {
      "id": "output",
      "transmissionMode": "reference"
    }
  ]
}
```

Appendix G: CWL file for the WPS 1.0 to LLNL CWT workflow

CWL file for the WPS 1.0 to LLNL CWT workflow

```
{
    "CWLVersion": "v1.0",
    "class": "Workflow",
    "requirements": [
        {
            "class": "StepInputExpressionRequirement"
        }
    ],
    "inputs": {
        "files": "string[]",
        "variable": "string",
        "esgf_api_key": "string",
        "llnl_lat0": "float",
        "llnl_lat1": "float",
        "llnl_lon0": "float",
        "llnl_lon1": "float",
        "crim_lat0": "float",
        "crim_lat1": "float",
        "crim_lon0": "float",
        "crim_lon1": "float"
    },
    "outputs": {
        "output": {
            "type": "File",
            "outputSource": "llnl_subset/output"
        }
    },
    "steps": {
        "crim_subset": {
            "run": "ColibriFlyingpigeon_SubsetBbox.CWL",
            "in": {
                "resource": "files",
                "lat0": "crim_lat0",
                "lat1": "crim_lat1",
                "lon0": "crim_lon0",
                "lon1": "crim_lon1"
            },
            "out": ["output"]
        },
        "llnl_subset": {
            "run": "SubsetESGF.CWL",
            "in": {
                "files": "crim_subset/output",
                "variable": "variable",
```

```
                "api_key": "esgf_api_key",
                "lat_start": "llnl_lat0",
                "lat_end": "llnl_lat1",
                "lon_start": "llnl_lon0",
                "lon_end": "llnl_lon1"
            },
            "out": ["output"]
        }
    }
}
```

Appendix H: CWL file for the WPS 1.0 to NASA CWT workflow

CWL file for the WPS 1.0 to NASA CWT workflow

```
{
    "CWLVersion": "v1.0",
    "class": "Workflow",
    "requirements": [
        {
            "class": "StepInputExpressionRequirement"
        }
    ],
    "inputs": {
        "files": "File",
        "variable": "string",
        "nasa_lat0": "float",
        "nasa_lat1": "float",
        "nasa_lon0": "float",
        "nasa_lon1": "float",
        "crim_lat0": "float",
        "crim_lat1": "float",
        "crim_lon0": "float",
        "crim_lon1": "float"
    },
    "outputs": {
        "output": {
            "type": "File",
            "outputSource": "crim_subset/output"
        }
    },
    "steps": {
        "nasa_subset": {
            "run": "SubsetNASAESGF.CWL",
            "in": {
                "files": "files",
                "variable": "variable",
                "lat_start": "nasa_lat0",
                "lat_end": "nasa_lat1",
                "lon_start": "nasa_lon0",
                "lon_end": "nasa_lon1"
            },
            "out": ["output"]
        },
        "file2string_array": {
            "run": "file2string_array",
            "in": {
                "input": "nasa_subset/output"
            },
```

```
            "out": ["output"]
        },
        "crim_subset": {
            "run": "ColibriFlyingpigeon_SubsetBbox.CWL",
            "in": {
                "resource": "file2string_array/output",
                "lat0": "crim_lat0",
                "lat1": "crim_lat1",
                "lon0": "crim_lon0",
                "lon1": "crim_lon1"
            },
            "out": ["output"]
        }
    }
}
```

Appendix I: Revision History

Table 1. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
March 11, 2019	T. Landry	.1	all	initial version
March 15, 2019	D. Byrns	.2	all	technical background
March 28, 2019	D. Byrns	.3	all	workflows and applications
March 31, 2019	T. Landry	.4	all	annexes and various edits
April 1, 2019	D. Byrns	.5	all	references/titles fixes
April 10, 2019	T. Landry	.6	all	summary, discussion, editorial changes
April 11, 2019	T. Landry	.7	all	near-final draft
May 14, 2019	T. Landry	.8	all	internal revision of document
May 16, 2019	T. Landry	.9	solution, discussion	response to comments

Appendix J: Bibliography

1. Sacramento, P., others: OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report. OGC 18-050,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
2. Sacramento, P., others: OGC Testbed-14: Application Package Engineering Report. OGC 18-049,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
3. Lee, C.A., others: OGC Testbed-14: Federated Clouds Engineering Report. OGC 18-049,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
4. Landry, T., others: OGC Testbed-14: Machine Learning Engineering Report. OGC 18-038,Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
5. Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., Stojanovic, L.: Common Workflow Language, v1.0. Specification. (2016).
6. Ehbrecht, C., Landry, T., Hempelmann, N., Huard, D., Kindermann, S.: Projects Based on the Web Processing Service Framework Birdhouse. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-4/W8, 43–47 (2018).
7. Chen, C., others: OGC Testbed-13: Cloud ER. OGC 17-035,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-035.html (2017).
8. Auten, H.H., Ames, A.S., Williams, D.N.: 7th Annual Earth System Grid Federation Face-to-Face Conference Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States) (2018).
9. Ehbrecht, C., Kindermann, S., Stephens, A., Denvil, S.: Web Processing Services for Copernicus Climate Change Service. EGU General Assembly Conference Abstracts. 20, 6491 (2018).
10. Maxwell, T.P., Duffy, D., Carriere, L., Potter, G.L.: The Earth Data Analytic Services (EDAS) Framework. AGU Fall Meeting Abstracts. (2018).
11. Petruzza, S., Venkat, A., Gyulassy, A., Scorzelli, G., Federer, F., Angelucci, A., Pascucci, V., Bremer, P.-T.: Scaling Big Data Neuroscience: From Interactive Analytics to HPC Platforms. In: Big Data and HPC: Ecosystem and Convergence, TopHPC 2017, Tehran, Iran, 24-26 April 2017. pp. 53–68 (2017).
12. Fiore, S., Plociennik, M., Doutriaux, C., Palazzo, C., Boutte, J., Zok, T., Elia, D., Owsiak, M., D'Anca, A., Shaheen, Z., Bruno, R., Fargetta, M., Caballer, M., Molto, G., Blanquer, I., Barbera, R., David, M., Donvito, G., Williams, D., Aloisio, G.: Distributed and cloud-based multi-model analytics experiments on large volumes of climate change data in the earth system grid federation eco-system. In: IEEE International Conference on Big Data (Big Data). pp. 2911–2918 (2016).
13. Gauvin St-Denis, B., Landry, T., Huard, D.B., Byrns, D., Chaumont, D., Foucher, S.: PAVICS: A platform for the Analysis and Visualization of Climate Science - adopting a workflow-based analysis method for dealing with a multitude of climate data sources. AGU Fall Meeting Abstracts. (2017).
14. Williams, D.N., Lautenschlager, M., Balaji, V., Cinquini, L., DeLuca, C., Denvil, S., Duffy, D., Evans, B., Ferraro, R., Juckes, M., Trenham, C.: Strategie Roadmap for the Earth System Grid Federation. In: Proceedings of the 2015 IEEE International Conference on Big Data (Big Data). pp. 2182–2190. IEEE Computer Society, Washington, DC, USA (2015).

15. Schnase, J., J. Lee, T., A. Mattmann, C., Lynnes, C., Cinquini, L., M. Ramirez, P., F. Hart, A., Williams, D., Waliser, D., Rinsland, P., Phillip Webster, W., Q. Duffy, D., A. McInerney, M., S. Tamkin, G., Potter, G., Carriere, L.: Big Data Challenges in Climate Science: Improving the next-generation cyberinfrastructure. IEEE Geoscience and Remote Sensing Magazine. 4, 10–22 (2016).

16. Potter, G.L., Carriere, L., Hertz, J., Bosilovich, M., Duffy, D., Lee, T., Williams, D.N.: Enabling Reanalysis Research Using the Collaborative Reanalysis Technical Environment (CREATE). Bulletin of the American Meteorological Society. 99, 677–687 (2018).

17. Balaji, V., E. Taylor, K., Juckes, M., Lawrence, B., Durack, P., Lautenschlager, M., Blanton, C., Cinquini, L., Denvil, S., Elkington, M., Guglielmo, F., Guilyardi, E., Hassell, D., Kharin, S., Kindermann, S., Nikonov, S., Radhakrishnan, A., Stockhause, M., Weigel, T., Williams, D.: Requirements for a global data infrastructure in support of CMIP6. Geoscientific Model Development. 11, 3659–3680 (2018).

18. Evans, B.J.K., Lautenschlager, M., Cinquini, L., Denvil, S., Ames, S., Ferraro, R., Balaji, V., Kershaw, P., Landry, T., Williams, D.N.: Addressing the massive CMIP6 data science challenge through the ESGF global federation. AGU Fall Meeting Abstracts. (2018).