

OGC® DOCUMENT: 22-020R1

External identifier of this OGC® document: <http://www.opengis.net/doc/PER/T18-D041>



Open
Geospatial
Consortium

TESTBED-18: IDENTIFIERS FOR REPRODUCIBLE SCIENCE SUMMARY ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2022-11-18

Approval Date: 2022-12-27

Publication Date: 2023-03-13

Editor: Paul Churchyard, Ajay Gupta

Notice: This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is *not an official position* of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

License Agreement

Use of this document is subject to the license agreement at <https://www.ogc.org/license>

Copyright notice

Copyright © 2023 Open Geospatial Consortium

To obtain additional rights of use, visit <https://www.ogc.org/legal>

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

I. EXECUTIVE SUMMARY	vi
II. KEYWORDS	vii
III. SECURITY CONSIDERATIONS	viii
IV. SUBMITTERS	viii
V. ABSTRACT	viii
1. SCOPE	2
2. NORMATIVE REFERENCES	4
3. TERMS, DEFINITIONS AND ABBREVIATED TERMS	6
3.19. Abbreviated terms	10
4. INTRODUCTION	12
5. COMMON CONSIDERATIONS AND LIMITATIONS FOR REPRODUCIBLE WORKFLOWS	14
5.1. The Expansion of FAIR	14
5.2. Randomness in Deep Learning Applications	15
5.3. Cloud Utilization Cost Estimation	16
6. INDIVIDUAL COMPONENT DESCRIPTIONS	18
6.1. 52°North	18
6.2. Arizona State University	32
6.3. Ecere	42
6.4. GeoLabs	59
6.5. Terradue	71
7. OVERALL LESSONS LEARNED AND FUTURE WORK	89
7.1. The Expansion of Whole Tale's Capabilities	89
7.2. The Influence on Randomness in Reproducible Machine Learning Workflows	89
7.3. How Health-Related Workflows Will Benefit from FAIR data	89
ANNEX A (INFORMATIVE) REVISION HISTORY	92
BIBLIOGRAPHY	94

LIST OF TABLES

Table – Submitters	viii
--------------------------	------

LIST OF FIGURES

Figure 1 – 52°North Workflow Diagram	20
Figure 2 – Screenshot of the Jupyter Notebook "51_analyze_samples"	29
Figure 3 – Comparison of model predictions with different coordinate reference system and resolution (top: original UTM coordinates with 10 m resolution, bottom: EPSG:4326 coordinates with 0.0001° resolution.)	31
Figure 4 – Arizona State University Target Detection	33
Figure 5 – Arizona State University General Workflow	34
Figure 6 – Arizona State University Object Detection Workflow	35
Figure 7 – Coastal Erosion Workflow Diagram	44
Figure 8 – Visualizing the output of coastal erosion workflow Ecere’s GNOSIS Cartographer client (large scale view)	46
Figure 9 – Visualizing the output of coastal erosion workflow Ecere’s GNOSIS Cartographer client (closer view)	46
Figure 10 – Visualizing coastal erosion workflow definition as a flow diagram	47
Figure 11 – Crop Classification Workflow output visualized in Ecere’s GNOSIS Cartographer client (from 2020-2021 MOAW GeoConnections project)	50
Figure 12 – Crop Classification Workflow output visualized in QGIS client (from 2020-2021 MOAW GeoConnections project)	50
Figure 13 – Parcels training data for crop classification workflow (from 2020-2021 MOAW GeoConnections project)	51
Figure 14 – Enhanced Vegetation Index used as training data for crop classification workflow (from 2020-2021 MOAW GeoConnections project)	51
Figure 15 – Sentinel-2 (ESA) Datacube presented as an OGC API collection on maps.gnosis.earth, sourced from Cloud Optimized GeoTIFF managed by Element 84 hosted on Amazon Web Services	54
Figure 16 – Sentinel-2 (ESA) Datacube subset as OGC API collection on maps.gnosis.earth representing East Coast US, sourced from Cloud Optimized GeoTIFF managed by Element 84 hosted on Amazon Web Services	55
Figure 17 – GeoLabs Wholetale Workflow	60
Figure 18 – GeoLabs OTB Workshop Material integration	62
Figure 19 – GeoLabs MapServer OGC Web Map Service	63
Figure 20 – GeoLabs MapServer OGC API - Fatures	64
Figure 21 – GeoLabs MapServer OGC FAIRy tale preview	65
Figure 22 – GeoLabs Wholetale.org build run	66

Figure 23 – GeoLabs Wholetale.org build run	67
Figure 24 – GeoLabs Wholetale.org build run	69
Figure 25 – GeoLabs Wholetale.org build run	70
Figure 26 – Application Package software configuration management	73
Figure 27 – Application Package on Software Heritage	75
Figure 28 – Application Package Continuous Integration on Gitlab.com	77
Figure 29 – Application Package on Zenodo	79
Figure 30 – Input collection on Zenodo	80
Figure 31 – Output collection on Zenodo	81
Figure 32 – Software repository for seeding the Whole Tale	82
Figure 33 – Application Package on Whole Tale	83
Figure 34 – Running the Application Package on Whole Tale	84
Figure 35 – Recorded runs of the Application Package on Whole Tale	84



EXECUTIVE SUMMARY

The Testbed 18 task Identifiers for Reproducible Science explored and developed EO workflows demonstrating the best practices from FAIR data and reproducible science whilst exploring the usability of Whole Tale as a component for reproducible workflows. EO workflows involve multiple processing steps such as imagery pre-processing, training of AI / ML models, the fusion / mosaicking of imagery together, and other analytical processes. In order for EO workflows to be reproducible, each step of the process must be documented in sufficient detail and that documentation needs to be made available to scientists and end users.

The following five participating organizations contributed workflows for the Testbed 18 Reproducible Science task.

- 52 Degrees North developed a Whole Tale workflow for land cover classification.
- Arizona State University is developing a reproducible workflow for a deep learning application for target detection from earth observation imagery.
- Ecere worked on the implementation of reproducible workflows following the approach described in the OGC API Process Part 3: Workflows and Chaining for Modular OGC API Workflows.
- GeoLabs developed a reproducible workflow that runs an OGC API Process and Feature Server instance within a Whole Tale environment.
- Terradue developed a water body detection Application Package to cover the identifier assignment and reproducibility from code to several execution scenarios (local, Exploitation Platform, Whole Tale) and is the editor for the Reproducible Best Practices ER, which is another component of the Reproducible Science stream.

Over the course of the Reproducible Science task multiple considerations and limitations for reproducible workflows were discovered including the following.

- The expansion of FAIR to include replicability, repeatability, reproducibility, and reusability (reproducible-FAIR).
 - Replicability: A process with the same input yields the same output.
 - Repeatability: A process with a similar input yields the same output.
 - Reproducibility: different inputs, platforms, and outputs result in the same conclusion.
 - Reusability: The ability to use a specific workflow for different areas with the same degree of accuracy and reliability on the output.
- Addressing randomness in deep learning applications.

- Addressing the limitation of Whole Tale’s inability to assign a DOI to a binary docker image used to build a Whole Tale experiment.

Recommended future work includes the impact of FAIR workflows for healthcare use cases which makes data more available and reliable to researchers, healthcare practitioners, emergency response personnel, and decision makers.



KEYWORDS

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

testbed, docker, web service, reproducibility, earth observation, workflows, whole tale, deep learning, fair

III

SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV

SUBMITTERS

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

Table – Submitters

NAME	ORGANIZATION	ROLE
Paul Churchyard	HSR.health	Editor
Ajay K. Gupta	HSR.health	Editor
Martin Pontius	52 North	Contributor
Chia-Yu Hsu	Arizona State University	Contributor
Jerome Jacovella-St-Louis	Ecere	Contributor
Patrick Dion	Ecere	Contributor
Gérald Fenoy	GeoLabs	Contributor
Fabrice Brito	Terradue	Contributor
Pedro Goncalves	Terradue	Contributor
Josh Lieberman	OGC	Contributor

V

ABSTRACT

The OGC's Testbed 18 initiative explored the following six tasks.

- 1.) Advanced Interoperability for Building Energy
- 2.) Secure Asynchronous Catalogs
- 3.) Identifiers for Reproducible Science
- 4.) Moving Features and Sensor Integration
- 5.) 3D+ Data Standards and Streaming
- 6.) Machine Learning Training Data

Testbed 18 Task 3, Identifiers for Reproducible Science, explored and developed workflows demonstrating best practices at the intersection of Findable, Accessible, Interoperable, and Reusable (or FAIR) data and reproducible science.

The workflows developed in this Testbed included:

- the development of a Whole Tail workflow for land cover classification (52 Degrees North);
- the development of a reproducible workflow for a deep learning application for target detection (Arizona State University);
- the implementation of reproducible workflows following the approach described in the OGC API Process Part 3: Workflows and Chaining for Modular OGC API Workflows (Ecere);
- the development of a reproducible workflow that runs an OGC API – Process and Feature Server instance within a Whole Tale environment (GeoLabs); and
- the development of a water body detection Application Package to cover the identifier assignment and reproducibility from code to several execution scenarios (local, Exploitation Platform, Whole Tale) (Terradue).

Testbed 18 participants identified considerations and limitations for reproducible workflows and recommendations for future work to identify the benefits of reproducible science for healthcare use cases.

1

SCOPE

1

SCOPE

This report is a summary of activities undertaken in the execution of the Testbed 18 Identifiers for Reproducible Science Stream. This included the development of best practices to describe all steps of an Earth Observation scientific workflow, including:

- input data from various sources such as files, APIs, and data cubes;
- the workflow itself with the involved application(s) and corresponding parameterizations; and
- output data.

The participants were also tasked with producing reproducible workflows and examining the feasibility of Whole Tale as a tool for reproducible workflows.



2

NORMATIVE REFERENCES

NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Open API Initiative: **OpenAPI Specification 3.0.2**, 2018 <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md>

van den Brink, L., Portele, C., Vretanos, P.: OGC 10-100r3, **Geography Markup Language (GML) Simple Features Profile**, 2012 http://portal.opengeospatial.org/files/?artifact_id=42729

W3C: **HTML5**, W3C Recommendation, 2019 <http://www.w3.org/TR/html5/>

Schema.org: <http://schema.org/docs/schemas.html>

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: IETF RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*. RFC Publisher (1999). <https://www.rfc-editor.org/info/rfc2616>.

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. RFC Publisher (2000). <https://www.rfc-editor.org/info/rfc2818>.

G. Klyne, C. Newman: IETF RFC 3339, *Date and Time on the Internet: Timestamps*. RFC Publisher (2002). <https://www.rfc-editor.org/info/rfc3339>.

M. Nottingham: IETF RFC 8288, *Web Linking*. RFC Publisher (2017). <https://www.rfc-editor.org/info/rfc8288>.

H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. RFC Publisher (2016). <https://www.rfc-editor.org/info/rfc7946>.

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in [OGC Policy Directive 49](#), which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications ([OGC 08-131r3](#)), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

3.1. API Coverages

“A Web API for accessing coverages that are modeled according to the Coverage Implementation Schema (CIS) 1.1. Coverages are represented by some binary or ASCII serialization, specified by some data (encoding) format.” Open Geospatial Consortium

3.2. API Features

“A multi-part standard that offers the capability to create, modify, and query spatial data on the Web and specifies requirements and recommendations for APIs that want to follow a standard way of sharing feature data.” Open Geospatial Consortium

3.3. GeoTIFF

“A GeoTIFF file extension contains geographic metadata that describes the actual location in space that each pixel in an image represents.” Heavy.ai

3.4. Copernicus CORINE Land Cover dataset

A collection of Land Cover images that covers over 44 classes since 1985. Copernicus

3.5. Data Cube

“A multi-dimensional (“n-D”) array of values, with emphasis on the fact that “cube” is just a metaphor to help illustrate a data structure that can in fact be 1- dimensional, 2-dimensional, 3-dimensional, or higher-dimensional.” Open Geospatial Consortium

3.6. DVC

“Open-Source Version Control System for Machine Learning Projects” DVC

3.7. GDAL

“A translator library for raster and vector geospatial data formats that is released under an MIT style Open Source License by the Open Source Geospatial Foundation.” GDAL

3.8. Non-Deterministic Models

An algorithm that can exhibit different behaviors on different runs which are useful for finding approximate solutions when an exact solution is far too difficult or expensive to derive using a deterministic algorithm. Engati

3.9. Parameterization Tuning / Tuning Parameters / Hyperparameters

Parameters that are a component of machine learning models that cannot be directly estimated from the data, but often control the complexity and variances in the model. Kuhn M. and Johnson K.

3.10. PyTorch

“An open source machine learning framework that accelerates the path from research prototyping to production deployment.” PyTorch

3.11. Whole Tale

“A scalable, open source, web-based, multi-user platform for reproducible research enabling the creation, publication, and execution of tales — executable research objects that capture data, code, and the complete software environment used to produce research findings.” Whole Tale

3.12. LandSat

The NASA/USGS Landsat Program provides the longest continuous space-based record of Earth’s land in existence. Landsat data give us information essential for making informed decisions about Earth’s resources and environment. National Air and Space Administration

3.13. Sentinel-2

A European wide-swath, high-resolution, multi-spectral imaging mission. European Space Agency

3.14. Software Heritage

Software Heritage is an organization that allows for the preservation, archiving, and sharing of source code for software. Software Heritage

3.15. ISEA3H DGGS

Icosahedral Snyder Equal Area Aperture 3 Hexagon Discrete Global Grid System – A specification for equal area DGGS based on the Icosahedral Snyder Equal Area (ISEA) projection
Southern Oregon University

3.16. RDF Encoding

A metadata format that “provides interoperability between applications that exchange machine-understandable information on the Web.” W3

3.17. Zenodo

A platform and repository developed and operated by CERN to enable the sharing of research data and outputs. <<zenodo>

3.18. CodeMeta

“CodeMeta contributors are creating a minimal metadata schema for science software and code, in JSON and XML. The goal of CodeMeta is to create a concept vocabulary that can be used to standardize the exchange of software metadata across repositories and organizations.”
CodeMeta

3.19. Abbreviated terms

ADES	Application Deployment and Execution Service
API	Application Programming Interface
AWS	Amazon Web Services
COG	Cloud Optimized GeoTIFF
CWL	Common Workflow Language
DGGS	Discrete Global Grid System
EMS	Exploitation Platform Management Service
ESA	European Space Agency
FAIR	Findable, Accessible, Interoperable, Reusable
IANA	Internet Assigned Numbers Authority
MIME	Multipurpose Internet Mail Extensions
MODIS	Moderate Resolution Imaging Spectroradiometer
NASA	National Air and Space Administration
NSF	National Science Foundation
ODC	Open Data Cube
SCM	Software Configuration Management
SPDX	Software Package Data Exchange
STAC	Spatio-Temporal Asset Catalog
SWG	Subject Working Group
USGS	United States Geological Survey
WG	Working Group



4

INTRODUCTION

The OGC's Testbed 18 initiative explored six tasks, including: Advanced Interoperability for Building Energy; Secure Asynchronous Catalogs; Identifiers for Reproducible Science; Moving Features and Sensor Integration; 3D+ Data Standards and Streaming; and Machine Learning Training Data.

This component of OGC's Testbed 18 focuses on the Identifiers for Reproducible Science, a part of TestBed Thread 3: FUTURE OF OPEN SCIENCE AND BUILDING ENERGY INTEROPERABILITY (FOB).

Issues around true science are the topic of numerous academic and technical journals. A common theme among these scholarly articles is that the reproducibility of studies is a key aspect of science.

OGC's mission is to make location information Findable, Accessible, Interoperable, and Reusable (FAIR). This Testbed task will explore and develop workflows demonstrating best practices at the intersection of FAIR data and reproducible science.

This task shall develop best practices to describe all steps of a scientific workflow, including:

- data curation from various authoritative sources such as files, APIs, and data cubes;
- the workflows themselves supporting multiple applications and corresponding parameterization tuning for machine learning processes; and
- workflow outputs or results that support decision-making.

The workflows included in this component represent key areas of scientific discovery leveraging location information, Earth Observation data, and geospatial processes.

The description of the models will discuss how each step of the workflow can abide by FAIR principles.

Testbed 18 builds on the OGC's past work as well as broader industry efforts. One of the tasks of the Testbed is to explore the utilization of Whole Tale as a tool for reproducible workflows and ideally work collaboratively with the Whole Tale team to identify and address limitations in the use of Whole Tale for reproducible workflows. Some of the work in the individual workflows build off of the participant's efforts in previous TestBeds.



5

COMMON CONSIDERATIONS AND LIMITATIONS FOR REPRODUCIBLE WORKFLOWS

COMMON CONSIDERATIONS AND LIMITATIONS FOR REPRODUCIBLE WORKFLOWS

Over the course of this testbed a number of considerations and limitations were discovered pertaining to the workflows of the participants. The common considerations and limitations identified across multiple workflows are discussed in this section. There are a few considerations that pertained to certain workflows which are discussed in the individual component sections that follow.

5.1. The Expansion of FAIR

1. Replicability: A process with the same input yields the same output.
 - An analysis of soil characteristics of a soil sample produces the same result each time the process is run on the same sample.
2. Repeatability: A process with a similar input yields the same output.
 - An analysis of soil characteristics performed on an identical but different soil sample produces the same result.
3. Reproducibility: Different inputs, platforms, and outputs result in the same conclusion.
 - The classification workflow for a particular city based on an analysis of Landsat data should have a similar result when performed by aerial or other satellite imagery taken at the same time for the same location. Additionally, the workflow and results should be the same whether run on different Cloud-based computing environments (e.g., Google Cloud Platform (GCP), Microsoft Azure, or Amazon Web Services).
4. Reusability: The ability to use a specific workflow for different areas with the same degree of accuracy and reliability on the output.
 - An image classification workflow created for a specific city could also be used to classify a different city within a level of accuracy or confidence interval.

5.2. Randomness in Deep Learning Applications

Randomness is important for deep learning models and applications for optimization and generalization. However, the randomness inherent in the models makes true reproducibility a challenge.

5.2.1. Where Does Randomness Appear in Deep Learning Models?

- Hardware
- Environment
- Software/framework: different version releases, individual commits, operating systems, etc,
- Function: how the models are written as code and the package dependencies based on different programming languages
- Algorithm: random initialization, data augmentation, data shuffling, and stochastic layers (dropout, noisy activations)

5.2.2. Why is Randomness Critical and Important to Deep Learning Applications?

- Different outputs from the same input: normally an expected result is the same output given the same input, e.g., classification or detection, but sometimes some “creativity” is necessary in the model. Examples are: the first move of playing GO; draw pictures given a blank paper; etc.
- Learning/optimization: a neural network loss function has many local minima and it is easy to get stuck at the local minima during the optimization process. Randomness in many algorithms allows the optimization to bounce out of the local minima, such as random sampling in a stochastic gradient descent (SGD).
- Generalization: randomness brings better generalization in the network by injecting noise to the learning process, such as Dropout

5.2.3. How to Limit the Source of Randomness and Non-Deterministic Behaviors?

Using PyTorch as an example, the sources of randomness can be limited through:

- Control sources of randomness
 - `torch.manual_seed(0)`: control the random seed of PyTorch operations; and
 - `random.seed(0)`: control the random seed of customized Python operations.
- Avoiding using non-deterministic algorithms
 - `torch.backends.cudnn.benchmark = False`: for a new set of hyperparameters, the cudnn library would run different algorithms, benchmark them, and select the fastest one. Disabling the feature could cause a reduction in performance.

5.2.4. What is the Trade-Off Between Randomness and Reproducibility?

Randomness plays an important role in deep learning and completely similar results are not guaranteed and should not be expected. As such, the same or similar results necessitated by reproducibility may be an important, but not a critical, issue in deep learning models. As such, transparency is important in the reproducibility of deep learning models so that every step of the process can be examined.

5.3. Cloud Utilization Cost Estimation

There is not always clarity in Cloud utilization costs. Any Cloud-based development efforts should make a concerted effort to track fees associated with the Cloud infrastructure. Most, if not all, Cloud hosting firms make tools available to help anticipate costs. For instance, AWS has a Cost Calculator that can help provide insights into future Cloud hosting and utilization fees.



6

INDIVIDUAL COMPONENT DESCRIPTIONS

6.1. 52°North

6.1.1. Goals of Participation

The goals of participation included the selection and development of a viable workflow on Whole Tale that provides the following features.

- End-users will be able to experience how Spatial Data Analysis can be published in a reproducible FAIR manner.
- The implementation of this task will help OGC WG derive requirements and limitations of existing standards for the enablement of reproducible FAIR workflows.
- Developers will be able to follow a proof of concept for setting up a reproducible FAIR workflow based on OGC standards and an Open Data Cube instance.

6.1.2. Contributed Workflows and Architecture

52°North brought in a selection of use-cases from their own and partner research activities. From these potential workflows the use-case “Exploring Wilderness Using Explainable Machine Learning in Satellite Imagery” was chosen. This scientific study was conducted as part of the “KI:STE – Artificial Intelligence (AI) strategy for Earth system data” project and was already published on arXiv (<https://arxiv.org/abs/2203.00379>). The goal of the study was the detection of wilderness areas using remote sensing data from Sentinel-2. Moreover, the developed machine learning models allow the interpretation of the results by applying explainable machine learning techniques. The study area is Fennoscandia (<https://en.wikipedia.org/wiki/Fennoscandia>). For this region the AnthroProtect dataset was prepared and openly released (<http://rs.ipb.uni-bonn.de/data/anthroprotect/>). This dataset consists of preprocessed Sentinel-2 data. The regions of interest were determined using data from the Copernicus CORINE Land Cover dataset and from the World Database on Protected Areas (WDPA). Additionally, land cover data from five different sources are part of the AnthroProtect dataset: Copernicus CORINE Land Cover dataset, MODIS Land Cover Type 1, Copernicus Global Land Service, ESA GlobCover, and Sentinel-2 scene classification map.

In order to make the data available inside Whole Tale and to investigate reproducibility aspects of OGC APIs in conjunction with Open Data Cube, the AnthroProtect dataset was imported and indexed in an Open Data Cube instance and published via API Coverages and STAC. It would have been beneficial to offer some sub-processes of the workflow via API Processes, but this was not possible within these Testbed activities.

The source code of the original study is available at <https://gitlab.jsc.fz-juelich.de/kiste/wilderness>. A slightly modified version is available at <https://gitlab.jsc.fz-juelich.de/kiste/asos> and was used as a starting point for Testbed 18. Based on these developments a separate github repository was created (<https://github.com/52North/testbed18-wilderness-workflow>) which includes parts of the original workflow. From this repository, a tale on Whole Tale (<https://dashboard.wholetale.org/run/633d5fb4eb89f198ef8ce83f>) was created which can be executed by interested users to reproduce parts of the study.


6.1.3. Workflow Description

Figure 1 shows an overview of the workflow steps with their inputs and outputs performed in the original study.

The first step of the workflow, the preparation of the AnthroProtect dataset, was performed with Google Earth Engine (GEE). The download and preprocessing is described in the research article in detail and can be executed using Jupyter notebooks in a sub-project of the source repository (<https://gitlab.jsc.fz-juelich.de/kiste/asos/-/tree/main/projects/anthroprotect>). Due to time constraints this step was excluded in the Testbed activities. The prepared dataset can be downloaded as a zip file and is regarded as the “source of truth” from where reproducibility is enhanced. Some thoughts on reproducibility regarding closed source APIs/services like GEE are described in 52°North’s future work section.

Steps 2 and 3 include the training of the ML model and a sensitivity analysis (Activation Space Occlusion Sensitivity (ASOS)) which helps to interpret model results. For details, we refer to the research article mentioned in the previous paragraph. As these steps are processing intensive with processing times that are not well-suited for demonstration purposes, they are not part of the Whole Tale tale. However, it would be interesting to set up OGC API Processes for these. Further reproducibility aspects of machine learning could be studied, e.g., how does the choice of training data or hyperparameter tuning influence model weights or how ML models can be versioned, and it could be demonstrated how the OGC Processing API can contribute to reproducibility.

The trained model and the calculated sensitivities can be used to analyze Sentinel-2 scenes to predict activation and sensitivity maps. The sensitivity maps show the classification of a scene as wild or anthropogenic. Workflow step 5.1 allows the performance of such an analysis with available Sentinel-2 samples. It is the core of the developed Whole Tale tale in these Testbed activities. Workflow Step 4 allows for the inspection of the activation space in detail and for the investigation of how areas in the activation space relate to land cover classes.

Notes	
	Steps not included in Whole Tale
AnthroProtect dataset	Preprocessed Sentinel-2 data (wild and anthropogenic areas) and land cover data (from 5 different sources). File format: GeoTIFF http://rs.ipb.uni-bonn.de/data/anthroprotect/
WDPA	World Database on Protected Areas (WDPA)
CORINE	Copernicus CORINE Land Cover dataset by ESA
ASOS	Activation Space Occlusion Sensitivity
Image source	https://gitlab.jsc.fz-juelich.de/kiste/asos/-/tree/main/readme

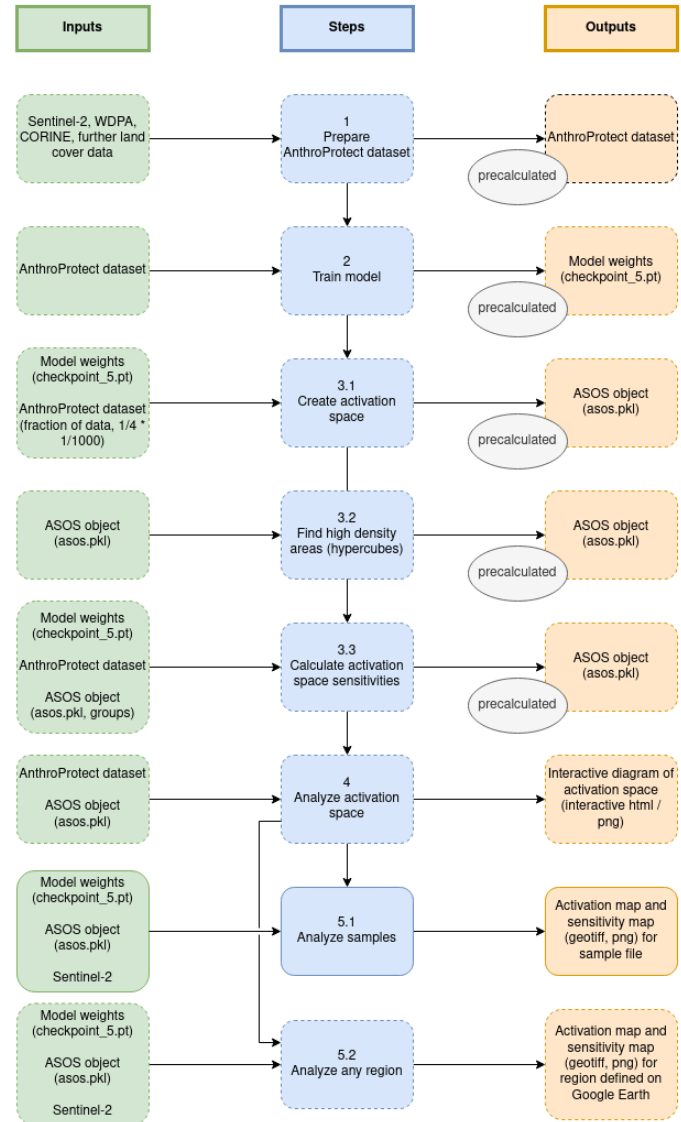
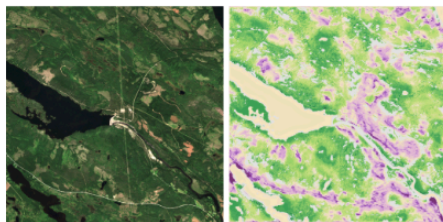
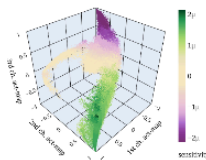
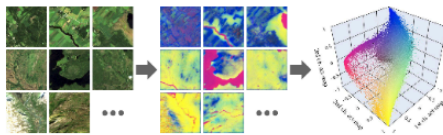
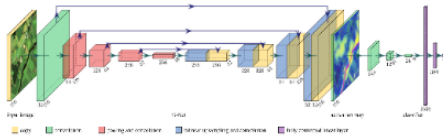


Figure 1 – 52°North Workflow Diagram

6.1.4. Local Execution

The original study is already designed and published in a way that reproducibility is enhanced. It is published on arXiv and has a DOI assigned: <https://doi.org/10.48550/arXiv.2203.00379>. The publication includes a research article as a pdf and links to associated datasets and the source code repository. Users can download these resources and repeat the analysis on their system of choice, such as their local machine.

The first experiment of the Testbed activities was to download the resources and perform the analysis on a local computer. While it was possible to do so, some challenges were observed. Mainly, a user needs to set up the runtime environment on its own. The setup of the runtime environment involves installing python and system libraries which can be challenging for non-

technical users. As the workflow needs GDAL, the system library has to be installed in addition to the Python GDAL library with a matching version. Another challenge of setting up the runtime environment is the choice of the processing device, CPU or GPU. By default GPUs are used in many ML applications and could not be configured differently. This problem was fixed during Testbed. Lastly, the chosen system needs to provide sufficient memory and disk space to perform the analysis. In the given example, the technical requirements are quite moderate (~50 GB disk space and ~10 GB RAM needed), however, in many other use cases the technical requirements on the hardware could become a practical limitation.

The challenge of setting up the runtime environment can be addressed by using Docker and running the application inside a Docker container. To address problems with hardware requirements and to eliminate the necessity of downloading or moving large amounts of data, parts of a workflow can be encapsulated in an OGC API Process which runs the workflow close to where the data resides. In this case it is important to ensure that processes and process inputs and outputs are also reproducible. Experiments with running parts of the workflow on the Whole Tale platform are described in the upcoming sections.

6.1.5. Supporting Infrastructure

As mentioned above, the AnthroProtect dataset should be made available via OGC APIs to be used inside of Whole Tale without uploading the whole dataset within Testbed 18. Moreover, reproducibility aspects of data cube infrastructures should be investigated. Therefore, an Open Data Cube instance is set up and the AnthroProtect dataset is added to the Open Data Cube index. This is done in a dedicated Docker container. The Docker image is publicly available via <https://hub.docker.com/r/52north/opendatacube-importer> as well as the source code via <https://github.com/52North/opendatacube-importer>. When the container is run it will download the AnthroProtect dataset from the public repository as ZIP file (<https://uni-bonn.sciebo.de/s/6wrgdIndjpfRJuA>), check the file hash (sha256), unpack the data, create ODC metadata files (yaml format), and add the files to the ODC index. Data that is indexed this way can be discovered and loaded using ODC's Python API. Afterwards, a pygeoapi instance is deployed which runs in a separate Docker container (<https://hub.docker.com/r/52north/pygeoapi-opendatacube>). The image extends the default pygeoapi Docker image (<https://hub.docker.com/r/geopython/pygeoapi>) with the pygeoapi-odc-provider (<https://github.com/52North/pygeoapi-odc-provider>) in order to serve data from Open Data Cube via OGC APIs (API Coverages). Both Docker images are built automatically in a pipeline using GitHub Actions when a new GitHub tag is created. After successfully being built, the Docker images are pushed to the respective image repository on Docker Hub. The images are referenced in a Kubernetes setup using Kustomize when defining which containers should be run.

```
- name: pygeoapi
  image: 52north/pygeoapi-opendatacube:0.4.0
```

As described in the considerations and limitations section of GeoLab's component, tags on Docker Hub are not completely reliable as it is possible to publish a different image with the same tag. While the steps explained above include some aspects of reproducibility in terms of setting up an infrastructure there is also a need for reproducibility when discovering and using data and associated services. Over the past few years, the STAC specification has evolved into a de facto standard which has the goal of describing spatio-temporal data for easy discovery and

access. The Open Data Cube community has already made steps to extend ODC to enable users to index datasets using STAC.

QUESTION:

Is it possible to use the ODC metadata interface (run in PostgreSQL) to index existing analysis-ready data sets? Suppose one has a lot of data already stored in Amazon AWS which is analysis-ready. How is this data ingested in ODC?

ANSWER:

Not only is this possible, but this is the strategic vision for DEA's delivery architecture using ODC. Datasets from S3 can be indexed individually – however we are currently working with Radiant Earth to both publish our datasets in AWS S3 with STAC metadata and to extend ODC to be able to index datasets using STAC. Using STAC's principles of static, lightweight representation of spatial metadata would mean large collections could be indexed quickly, with a minimum of GET requests.

– <https://www.opendatacube.org/faq>,

There are also tools and explanations to jointly work with ODC and STAC (e.g., <https://github.com/opendatacube/odc-stac> or https://docs.digitalearthafrika.org/en/latest/sandbox/notebooks/Frequently_used_code/Downloading_data_with_STAC.html).

In the following, exemplary STAC items are presented for the AnthroProtect dataset that is indexed in an Open Data Cube instance. A specific focus is put on three STAC extensions: the Datacube extension (<https://github.com/stac-extensions/datacube>), the Scientific Citation extension (<https://github.com/stac-extensions/scientific>), and the File Info extension (<https://github.com/stac-extensions/file>). It has to be noted that they have different levels of maturity, with two of them only being proposals at time of writing. STAC consists of four semi-independent specifications: STAC Item, STAC Catalog, STAC Collection, and STAC API (<https://stacspec.org/en/>). Items describing specific assets (files), catalogs, and collections are a way of organizing items where collections require some more information compared to catalogs. The STAC API defines a REST API which provides the ability to search STAC Items. As top-level STAC Catalog, a catalog for Open Data Cube is defined as follows:

```
{
  "stac_version": "1.0.0",
  "stac_extensions": [],
  "type": "Catalog",
  "id": "opendatacube",
  "title": "Open Data Cube, Testbed-18",
  "description": "Open Data Cube instance for Testbed-18 - Identifiers for Reproducible Science",
  "links": [
    {
      "rel": "root",
      "href": "./catalog.json",
      "type": "application/json"
    },
    {
      "rel": "child",
      "href": "./anthroprotect/collection.json",
      "type": "application/json"
    }
  ]
}
```

```
]
}
```

Which consists of only one child, the STAC Collection for the AnthroProtect dataset.

```
{
  "stac_version": "1.0.0",
  "stac_extensions": [
    "https://stac-extensions.github.io/scientific/v1.0.0/schema.json"
  ],
  "type": "Collection",
  "id": "anthroprotect",
  "title": "AnthroProtect dataset",
  "description": "AnthroProtect dataset consisting of Sentinel-2 and land
cover data",
  "license": "CC-BY-NC-SA-3.0",
  "sci:citation": "Stomberg, T. T., Stone, T., Leonhardt, J., Weber, I., &
Roscher, R. (2022). Exploring Wilderness Using Explainable Machine Learning in
Satellite Imagery. arXiv preprint arXiv:2203.00379.",
  "sci:doi": "10.48550/arXiv.2203.00379",
  "extent": {
    "spatial": {
      "bbox": [
        [
          5.20467414651499,
          55.353564035013825,
          30.630830204026555,
          70.36171326846416
        ]
      ]
    },
    "temporal": {
      "interval": [
        [
          "2020-07-01T00:00:00.000Z",
          "2020-08-30T23:59:59.999Z"
        ]
      ]
    }
  },
  "links": [
    {
      "rel": "root",
      "href": "../catalog.json",
      "type": "application/json"
    },
    {
      "rel": "child",
      "href": "./s2/collection.json",
      "type": "application/json"
    },
    {
      "rel": "child",
      "href": "./s2_scl/collection.json",
      "type": "application/json"
    },
    {
      "rel": "child",
      "href": "./lcs/collection.json",
      "type": "application/json"
    },
    {
      "rel": "parent",
```



```

    "href": "../catalog.json",
    "type": "application/json"
  },
  {
    "rel": "cite-as",
    "href": "https://doi.org/10.48550/arXiv.2203.00379"
  }
]
}

```

This additionally defines the spatial and temporal extent of the data. For reproducibility, the license, sci:doi, and sci:citation fields are important to mention. The “sci” prefix indicates that those fields are coming from the “Scientific Citation” extension which is added to the “stac_extensions” list. They provide a way to relate the data to a scientific publication, in this case the publication where the AnthroProtect dataset was introduced and where users find more information about how the dataset was created, e.g., which preprocessing was applied. Note that the actual DOI url is added to the “links” list.

The collection has three child collections: “s2_scl,” “s2,” and “lcs.” These directly correspond to ODC products in the presented example. ODC products are a central unit for structuring data inside of an ODC instance. An ODC product primarily defines which bands are available. For each product, ODC datasets are added which define the paths to specific bands which can be in separate files, spatio-temporal extent, and coordinate reference system. An ODC product can be interpreted as a data cube in a narrower sense while an Open Data Cube instance as a whole provides an index and a discovery and access mechanism to heterogeneous and distributed data.

The next example is the STAC Collection for Sentinel-2 (“s2” collection).

```

{
  "stac_version": "1.0.0",
  "stac_extensions": [
    "https://stac-extensions.github.io/datacube/v2.1.0/schema.json",
    "https://stac-extensions.github.io/scientific/v1.0.0/schema.json"
  ],
  "type": "Collection",
  "id": "anthroprotect_s2",
  "title": "Sentinel-2: MultiSpectral Instrument",
  "description": "Multi-dimensional Sentinel-2 data cube in a STAC collection.",
  "license": "CC-BY-NC-SA-3.0",
  "sci:citation": "Stomberg, T. T., Stone, T., Leonhardt, J., Weber, I., & Roscher, R. (2022). Exploring Wilderness Using Explainable Machine Learning in Satellite Imagery. arXiv preprint arXiv:2203.00379.",
  "sci:doi": "10.48550/arXiv.2203.00379",
  "extent": {
    "spatial": {
      "bbox": [
        [
          5.20467414651499,
          55.353564035013825,
          30.630830204026555,
          70.36171326846416
        ]
      ]
    },
    "temporal": {
      "interval": [
        "2020-07-01T00:00:00.000Z",
        "2020-08-30T23:59:59.999Z"
      ]
    }
  }
}

```

```

    ]
  }
},
"cube:dimensions": {
  "x": {
    "type": "spatial",
    "axis": "x",
    "extent": [
      5.20467414651499,
      30.630830204026555
    ],
    "reference_system": 4326,
    "step": 0.0001
  },
  "y": {
    "type": "spatial",
    "axis": "y",
    "extent": [
      55.353564035013825,
      70.36171326846416
    ],
    "reference_system": 4326,
    "step": 0.0001
  },
  "time": {
    "type": "temporal",
    "extent": [
      "2020-07-01T00:00:00Z",
      "2020-08-30T23:59:59Z"
    ],
    "description": "Data includes only one time slice which represents the
25th percentile of cloud-filtered scenes in the time interval described by the
temporal extent"
  },
  "spectral": {
    "type": "bands",
    "values": [
      "B2",
      "B3",
      "B4",
      "B5",
      "B6",
      "B7",
      "B8",
      "B8A",
      "B11",
      "B12"
    ]
  }
},
"links": [
  {
    "rel": "root",
    "href": "../..//catalog.json",
    "type": "application/json"
  },
  {
    "rel": "parent",
    "href": "../collection.json",
    "type": "application/json"
  }
]
}

```

```

    "rel": "item",
    "href": "./inv_hydroelectric-lets_i_2019-07-01-2019-08-30/inv_
hydroelectric-lets_i_2019-07-01-2019-08-30.json",
    "type": "application/geo+json"
  },
  {
    "rel": "item",
    "href": "./anthropo_24.81681-64.10019-5_0/anthropo_24.81681-64.10019-5_0.
json",
    "type": "application/geo+json"
  },
  {
    "rel": "cite-as",
    "href": "https://doi.org/10.48550/arXiv.2203.00379"
  }
]
}

```

Similar to the AnthroProtect collection, it uses the “Datacube” extension as well. This allows the STAC to specify the dimensions of the data cube (spatial, temporal, bands). Moreover, there are links to STAC Items. An exemplary item for the Sentinel-2 collection is shown in the following.

```

{
  "stac_version": "1.0.0",
  "stac_extensions": [
    "https://stac-extensions.github.io/file/v2.1.0/schema.json",
    "https://stac-extensions.github.io/scientific/v1.0.0/schema.json"
  ],
  "type": "Feature",
  "id": "inv_hydroelectric-lets_i_2019-07-01-2019-08-30",
  "collection": "anthroprotect_s2",
  "properties": {
    "datetime": "2020-08-01T12:00:00.000Z",
    "projection": 32634,
    "sci:citation": "Stomberg, T. T., Stone, T., Leonhardt, J., Weber, I., &
Roscher, R. (2022). Exploring Wilderness Using Explainable Machine Learning in
Satellite Imagery. arXiv preprint arXiv:2203.00379.",
    "sci:doi": "10.48550/arXiv.2203.00379"
  },
  "bbox": [
    20.26814539274694,
    66.45825882626778,
    20.496956405644095,
    66.55002311657806
  ],
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          20.26814539274694,
          66.45825882626778
        ],
        [
          20.26814539274694,
          66.55002311657806
        ],
        [
          20.496956405644095,
          66.55002311657806
        ],
        [
          20.496956405644095,

```

```

        66.45825882626778
      ],
      [
        20.26814539274694,
        66.45825882626778
      ]
    ]
  },
  "links": [
    {
      "rel": "root",
      "href": "../../../catalog.json",
      "type": "application/json"
    },
    {
      "rel": "collection",
      "href": "../collection.json",
      "type": "application/json"
    },
    {
      "rel": "parent",
      "href": "../collection.json",
      "type": "application/json"
    },
    {
      "rel": "cite-as",
      "href": "https://doi.org/10.48550/arXiv.2203.00379"
    }
  ],
  "assets": {
    "default": {
      "href": "https://18.testbed.dev.52north.org/geodatacube/stac/opendatacube
/investigative/inv_airport-lucas-U314_45664214.tif",
      "type": "image/tiff; application=geotiff",
      "title": "inv_airport-lucas-U314_45664214",
      "file:checksum": "a62448afe4db06bdae98db3e3b86e156c82582814c29f9ddeb21915
ede490849",
      "file:size": 18561018
    }
  }
}

```

In terms of reproducibility on the item level, the “File Info” extension is interesting as it allows for the specification of the file size and the file checksum. The latter helps to enhance the integrity of the data.

6.1.6. Deployment on Whole Tale

The Whole Tale initiative provides a multi-user platform where so-called tales can be created and used to reproduce scientific studies. A tale combines data, source code, the computational environment, and a narrative. By providing the computational environment some of the limitations and difficulties when repeating a study on a local machine can be overcome. Because the source code of the original scientific study depends on the whole dataset to be present in a specific folder structure, the code could not be directly used to create a tale as it is impractical to upload it manually. In principle external data can be added to a tale (https://wholetale.readthedocs.io/en/stable/users_guide/manage.html#external-data), however, only a few data

repositories are supported at the time of this report (DataONE, Dataverse, Globus, and Zenodo), whereas the AnthroProtect dataset is hosted on Sciebo.

Moreover, configuration files have to be added to enable Whole Tale to build a Docker image using repo2docker, from which the runtime environment is created. Thus a separate GitHub repository was created specifically for reproducing parts of the study on Whole Tale (<https://github.com/52North/testbed18-wilderness-workflow>). The first step is to provide configuration files for repo2docker. The original study provides a Python library (<https://gitlab.jsc.fz-juelich.de/kiste/asos/-/blob/main/setup.py>) that is used in Python scripts and Jupyter Notebooks to perform the actual analyses. The library is built on top of PyTorch and has some more dependencies like GDAL and rasterio. In order to install the GDAL Python library, the GDAL system library has to be installed first, which can be done by providing an `apt.txt` file. A limitation is that the Docker image is based on Ubuntu 18.04 which restricts the GDAL version. However, if this is a problem, a custom Dockerfile can be provided for better control over the environment. Python environments can be set up by using an `environment.yml`, a `requirements.txt`, or a `setup.py` file. Because of some dependency issues, a more customized approach was chosen by using the `postBuild` file. It is also possible to define environment variables for the runtime environment by adding in a line such as the following to the `postBuild` file.

```
echo 'export MY_ENV_VAR="MyValue"' >> ~/.profile
```

However, these environmental variables are not available inside Jupyter Notebooks and are only available inside a terminal. It also has to be noted that the approach documented for repo2docker (https://repo2docker.readthedocs.io/en/latest/config_files.html#start-run-code-before-the-user-sessions-starts) is not supported by Whole Tale. Through personal communication with the development team of Whole Tale 52 Degrees North found that the development team is working on an alternative approach. Additionally, not all relevant logging statements are exposed to the user. Even though the build process of the Docker image was successful running the tale did not work initially. As the relevant logging statements which described the problem were not provided to the user this was difficult to debug. However, with the help of Whole Tale's development team the issue could be fixed in the end.

After setting up the computational environment successfully, the tale can be run. 52°North chose Jupyter Notebook as environment but others are also possible (JupyterLab, MATLAB, RStudio, or STATA). From the original workflow, only a simplified version of Step 4 for inspecting the activation space and Step 5.1 for analyzing samples are part of the tale as a Jupyter Notebook. As a preparational step, there is an additional Jupyter Notebook for downloading samples via a Coverage API to be analyzed in step 5.1. Figure 2 shows how activation and sensitivity maps can be calculated for Sentinel-2 samples using the precalculated machine learning model.

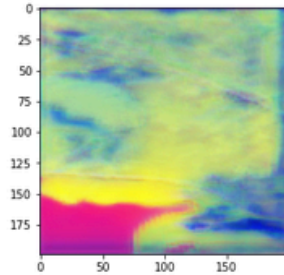
Predict activation map

```
In [12]: unet_map = utils.predict_activation_maps(input_data)
```

```
100%|██████████| 1/1 [00:01<00:00, 1.82s/it]
```

```
In [13]: plt.imshow((np.array(unet_map)[0]).transpose(1,2,0)+1)/2)
```

```
Out[13]: <matplotlib.image.AxesImage at 0x7f37c9d0e670>
```



Predict sensitivity map

```
In [14]: sensitivities = utils.predict_sensitivity_maps(input_data)
```

```
100%|██████████| 1/1 [00:01<00:00, 1.64s/it]
```

```
predicting groups: 100% ██████████ 981/981 [00:00<00:00, 13981.35it/s]
```

```
predicting sensitivities: 100% ██████████ 1/1 [00:00<00:00, 56.14it/s]
```

```
In [15]: asos = utils.load_asos()
```

```
In [16]: plt.imshow(sensitivities[0], cmap=asos.cmap, vmin=-asos.vlim, vmax=asos.vlim)
```

```
Out[16]: <matplotlib.image.AxesImage at 0x7f388d04e3a0>
```

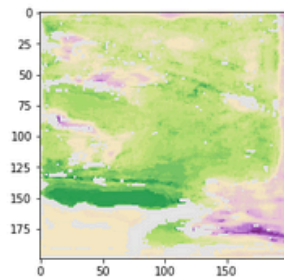


Figure 2 – Screenshot of the Jupyter Notebook "51_analyze_samples"

One practical limitation was observed regarding the available resources in the runtime environment. If the samples are large, such that a memory of ~8 GB is exceeded, the kernel of the notebook will crash. As the tale was created initially from a GitHub repository it would be beneficial to be able to synchronize them. This is possible from within a running tale. From the Jupyter Notebook User Interface a terminal can be opened. As the .git repository is available in the tale workspace, changes from the original GitHub repository can be merged into the tale using git's cli tool.

6.1.7. Considerations and Limitations for Reproducible Workflows:

- Given the expected growth of demand for data in modern analysis and modeling applications, how well will data and processing environments be connected?
 - Will results need any level of pre-processing?
 - Will the Area of Interest need to be limited, and if so, how?
- As processing loads grow with data and complexity of the analysis, what are acceptable processing times?
 - Will results need any level of pre-processing?
 - Is there a need for processing to be “out sourced” and made available through, for example, OGC API Processes?

6.1.8. Technical Interoperability Experiments

52°North shared their Tale (<https://dashboard.wholetale.org/run/633d5fb4eb89f198ef8ce83f>) and STAC with fellow Testbed 18 participants.

6.1.9. Lessons Learned

One important lesson learned is related to the code design of the original study. It uses a file-based logic to load the needed data inside of the various analysis steps. This means that all files are required to be available and to be stored in a specific folder structure. Moving completely to an API-based approach is not trivial and was out-of-scope for the Testbed 18 activities. Thus, only some parts of the workflow could be run on Whole Tale and studied with respect to the role of Open Data Cube and OGC APIs on reproducibility. While this might be considered as a specific challenge for the chosen workflow, it is probably encountered in many other applications as well as there are practical advantages of working file-based, e.g., it is simple, often faster (depending on the network speed and internet connection), self-contained, and reliable because an API could go offline. One way of supporting all analysis steps would be to provide them as processes via the OGC Processes API working directly on the files. This is described in the section “Future work.” With this background, the role of APIs to provide data for a workflow study could be restricted to collecting data only as a preparational step, i.e., data is downloaded via an API and stored in files which are then used in the analysis.

Another lesson learned is to be careful in the choice of the coordinate reference system when training a machine learning model with geo-referenced data and using it for model inference later. In the considered study, Sentinel-2 data is used in UTM coordinates with a spatial resolution of 10 m to train the machine learning model. For model inference inside of the Jupyter Notebook, data can be downloaded from a Coverage API which provides Sentinel-2 data in EPSG:4326 with 0.0001° resolution because the collection spans an area that crosses

multiple UTM zones. Figure 3 shows a qualitative comparison of model predictions of the same Sentinel-2 sample but with a different coordinate reference system, one in UTM coordinates, the other in geographic coordinates. The latter one is stretched horizontally because the resolution is in degrees and is the same for latitude and longitude. While the general characteristics are very similar, some small deviations can be observed due to the resampling, e.g., in the south-western part of the images.

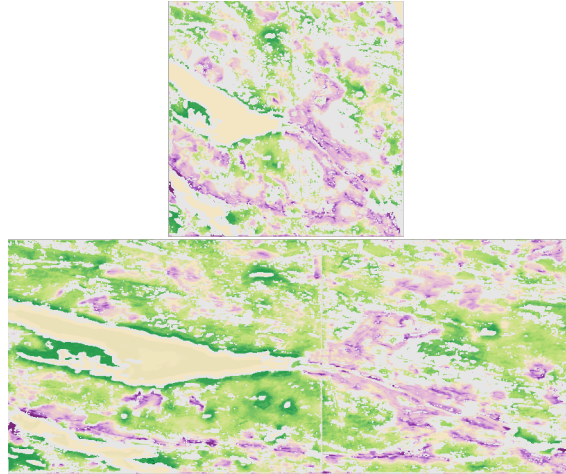


Figure 3 – Comparison of model predictions with different coordinate reference system and resolution (top: original UTM coordinates with 10 m resolution, bottom: EPSG:4326 coordinates with 0.0001° resolution.)

When working with Open Data Cube or other data cube software, STAC is a promising candidate to identify the provided data and ensure its use in a reproducible FAIR manner. STAC provides the potential to describe the data in a granular way from top-level views on complete datasets like Sentinel-2 to the level of a single file where the File Info extension offers the option to provide – amongst others – the file hash which can be checked to make sure the file has not changed. With the Scientific Citation, extension references to publications can be made so that users are able to understand where the data comes from and which preprocessing might have been applied or to acknowledge data providers and users by citation. The Datacube extension can be used to describe the dimensions of a data cube, however, when using these extensions it has to be noted that they have different levels of maturity and might still change or be disapproved. Specifically, for Open Data Cube there have been and are going to be more developments towards supporting STAC as a primary metadata language.

6.1.10. Future work

One important question that has not yet been answered is how the data and processing intensive workflow steps, i.e., the model training and the ASOS analysis, can be executed in a reproducible FAIR manner on the Whole Tale platform or similar platforms. Providing them via the OGC Processing API is an obvious approach. The processes could then be executed in a Jupyter Notebook on Whole Tale and the process outputs would be the model weights (checkpoint.pt) and the ASOS object (asos.pkl), respectively. In order to test and reproduce different model runs that use, for example, different hyperparameters or training data, a strategy is needed to version these process outputs. One possibility is to use Data Version Control (<https://dvc.org/>) which was used by Arizona State University. The question would be how

this can be integrated with Whole Tale. From a machine learning perspective, the influence of the choice of the coordinate reference system and the spatial resolution could also be further investigated.

Another aspect that could be further investigated is the reproducibility of data APIs, especially if they are not open source like the Google Earth Engine. In the considered workflow the AnthroProtect dataset was prepared using the Google Earth Engine and could be used for reproducibility experiments in the future. What metadata is needed to identify the data and how can it be tested that the same request always returns the same data? Is desirable that the data does not change or are there cases where this might not make sense?

Relating to the experiments with Open Data Cube and STAC, the existing tools to use STAC for metadata description, data discovery, and data loading could be evaluated. Of special interest is how the aforementioned STAC extensions (Datacube, Scientific Citation, File Info) can improve the reproducibility when using the tools.

6.2. Arizona State University

6.2.1. Goals of Participation

Arizona State University's goal for Testbed 18 was the development of a general workflow to demonstrate the FAIR and reproducibility principles of a scientific deep learning application, which will contain the following components.

- Input: Make the data easily accessible and re-useable. Be compliant with the OGC Standards including the new OGC API Standards.
- Application: Develop a Jupyter notebook that includes executable code and detailed comments for each step (input, model, training, evaluation, and inference) of the deep learning workflow.
- Output: Enable multiple outputs including standard object detection results and images with rendered prediction results which are compliant with OGC Standards.

The example use case developed for Testbed 18 was Target Detection: Identifying and drawing the bounding box of desired targets, as illustrated below.

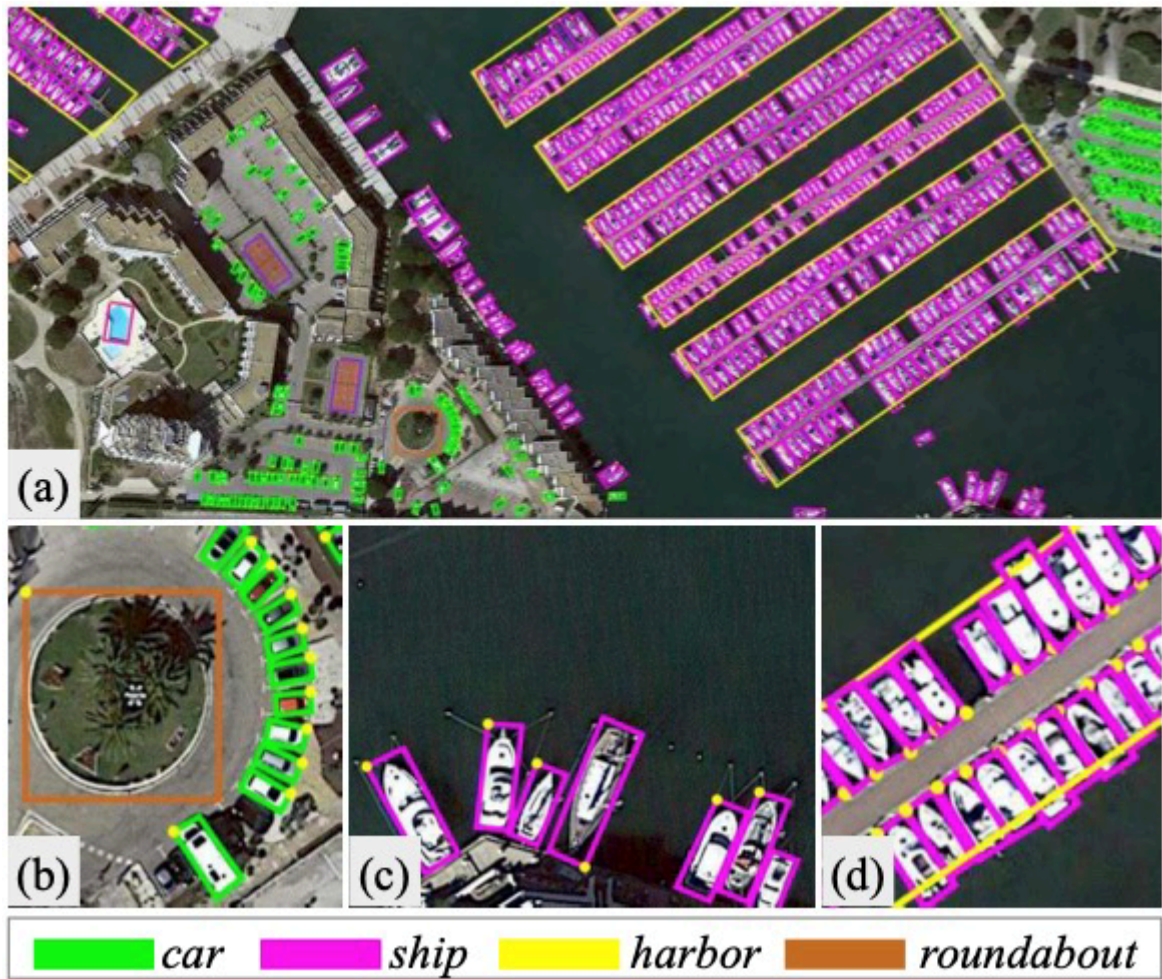


Figure 4 – Arizona State University Target Detection

This reproducible workflow for deep learning applications, especially the details to limit nondeterministic behaviors, provide the following.

- For End-users: share, deploy, and reuse date; video demonstration.
- For OGC Standards Baseline: a use case for reproducible deep learning applications.
- For Developers: a potential deep learning application reproducible workflow.

6.2.2. Contributed Workflow and Architecture with Challenges and Solutions

6.2.2.1. Workflow Tasks

1. Data preparation and web service hosting

2. Docker environment preparation
3. Application/Workflow development in Jupyter notebook
4. Reproducibility verification

6.2.3. General Workflow for a Deep Learning Application

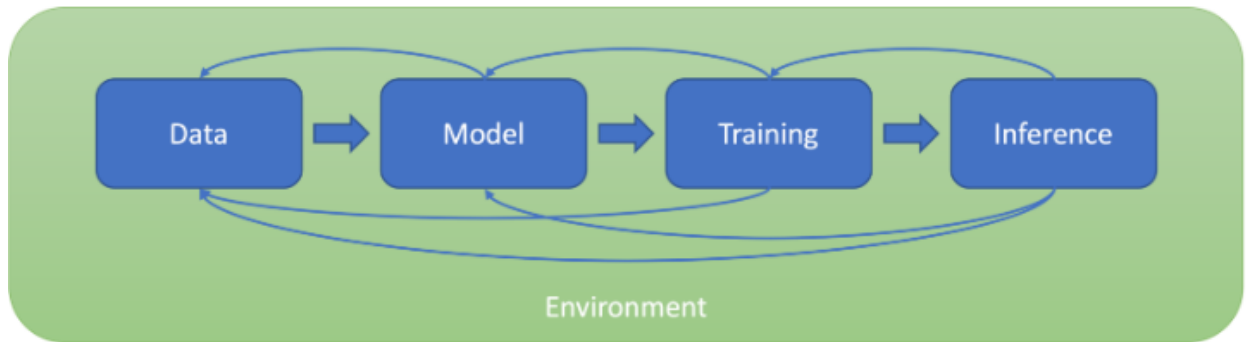


Figure 5 – Arizona State University General Workflow

The above figure shows a general workflow for a deep learning application. It contains five components: data, environment, model, training, and inference. In each component, several factors would affect the reproducibility of a deep learning application. There are also dependencies between different components (input, feedback, etc.). When implementing a reproducible workflow, consideration of both internal and external factors and interactions is important. The following sections demonstrate how to design and implement a reproducible workflow for each component.

6.2.4. Object Detection Workflow

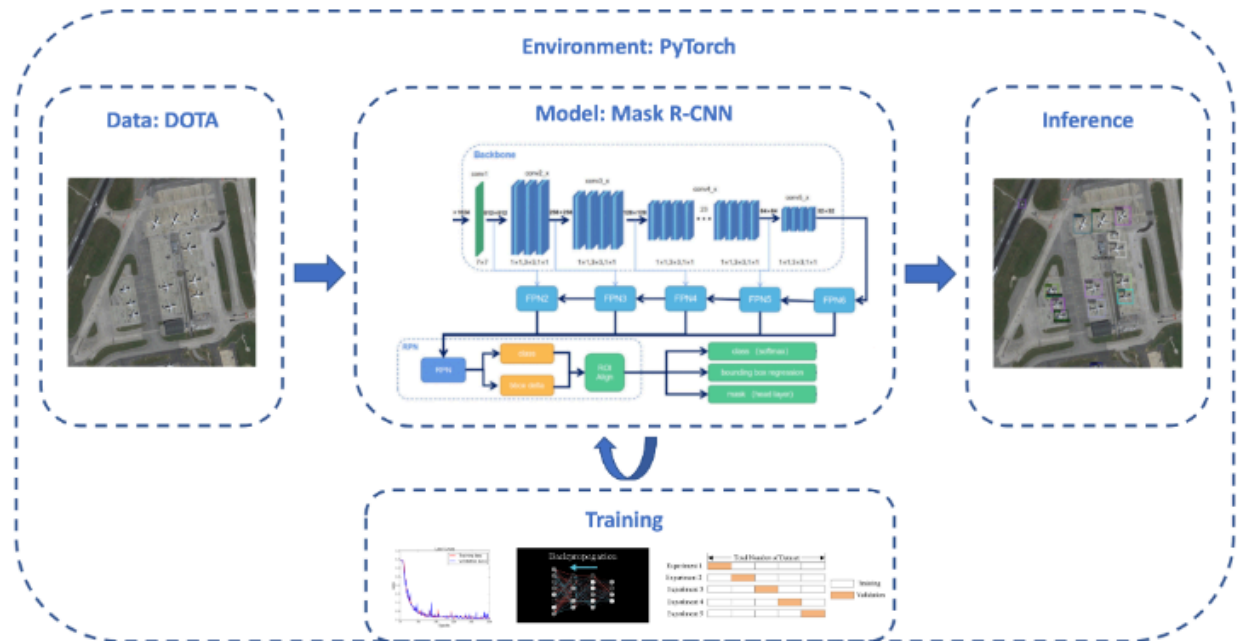


Figure 6 – Arizona State University Object Detection Workflow

To demonstrate the proposed workflow, object detection was used as an example. For each input image, the model identifies and draws bounding boxes on desired targets. The above figure shows the sample object detection workflow. There are five components.

- **Data:** the DOTA dataset was used as the core data for the workflow. This dataset was used for object detection in aerial images. There are 18 categories, 11268 images, and 1793658 instances in the dataset.
- **Environment:** PyTorch was used to build the deep learning workflow. PyTorch is an open source machine learning framework for fast research prototyping.
- **Model:** Mask R-CNN was used as the sample model. This is an object detection/segmentation model. For each input image, the model will identify and draw bounding boxes/masks on the targets.
- **Training:** the training calculates the loss between the predictions and the ground truths and uses backpropagation to update the network weights in order to minimize corresponding errors.
- **Inference:** The inference generates the bounding boxes for the targets based on the model's predictions and confidence scores.

6.2.5. Reproducible Data Workflow

<https://nbviewer.org/gist/chiayuhstu/121e5c9cc68222bdc8cbd4bd75c80fee>

This is the Jupyter notebook used to demonstrate the challenges and possible solutions of a reproducible data workflow.

6.2.6. Model Interoperability

For model interoperability, Open Neural Network eXchange (ONNX) is one of the initiatives that enables model sharing between different frameworks or ecosystems. ONNX is an open format for representing machine learning models. In ONNX, a model is represented as an acyclic dataflow graph with a list of nodes. Each node is a call to an operator with one or more inputs and one or more outputs. The graph also has metadata for documenting the authors, tasks, etc. A trained model with ONNX format can be used within different frameworks, tools, runtimes, and compilers.

6.2.6.1. Reproducibility and Replicability Discussions

- **Reproducibility:** The ability to repeat an experiment with minor differences while still achieving the same qualitative result [kastner_c]. That is, the result is stable and the conclusion and findings are consistent even with slightly different settings.
- **Replicability:** The ability to generate the same results with the same evaluation criteria under the same data, environment, model, training, and inference processes.

To achieve both reproducibility and replicability, it is important that sufficient details and specific steps are described. According to Peter and Florian [sugimura_peter_and_florian_hartl], “The root cause of most, if not all, reproducibility problems is missing information.” Following are discussions of challenges and possible solutions to the reproducibility and replicability in different components of a deep learning application.

6.2.6.2. Data

Challenges

- **Change in data:** data changes over time, including adding data, removing data, and updating data.
- **Incorrect data transformation:** for example, cleaning, augmentation, and changes in data distribution.
- **External data:** some data processing steps may depend on external data sources that do not produce stable results.

Solutions

- Data versioning and change tracking means keeping track and recording every data change at any instance. Depending on the size and structure of the data, how often the data is updated, and how often old versions are accessed, there are several different data versioning strategies.
 1. Storing copies of entire datasets. Each time a dataset is modified, a new copy of the entire dataset is stored. Pros: this approach is easy to implement in a storage system and to provide access to different versions of content. Cons: large storage space demand.
 2. Storing deltas between datasets. Pros: space efficient, Cons: restoring a specific version of a dataset may require many changes to the original dataset.
 3. Storing offsets in append-only datasets. For append-only datasets, such as log files, applications can simply use the file size to reconstruct a file at any given time. Pros: simplest dataset versioning Cons: only applicable to specific dataset types.
 4. Versioning corresponding transformations. Storing the original dataset and all steps involved to produce a specific version of dataset. Pros: may be more efficient to recreate the dataset on demand rather than to store it. Cons: the steps need to be deterministic.
- Versioning available metadata, such as schema information, license, provenance, owner, etc.
- Versioning the processing or editing steps applied to a specific version of a dataset. This applies to all processing steps including data collection/acquisition, data merging, data cleaning, or feature extraction. When using models with machine learning techniques, often input datasets are used that are quite removed from their original source. For example, data collected in the field, supplemented with data from other sources, cleaned in various manual and automated steps, and prepared with some feature extraction. At the stage where some data scientists may experiment with different network architectures, all that history and all the decisions that may have gone into the dataset, and all possible biases, may be lost. Versioning of the process can be recorded through metadata notes or drawn architectural data flow diagrams.
- Using data with documented timestamps and versions. Recording the exact training data after data processing at training time and logging all inputs to the model (including those gathered from external sources) at inference time is useful.

6.2.6.3. Environment

Challenges

- Hardware
 - Changes in GPU architectures would make reproducibility difficult unless some of the operations are enforced.
 - Floating-point discrepancy, either due to hardware settings, software settings, or compilers.
 - Non-deterministic behaviors: parallel floating point calculations in the GPU or auto-tuning features in libraries (CUDA, cuDNN)
- Software / libraries
 - ML frameworks are constantly getting upgraded. These updates can cause changes in the results. For instance, Pytorch 1.7+ supports mixed-precision natively from the apex library from NVIDIA but previous versions did not. Also, changing from one framework to another (i.e., Tensorflow to Pytorch) will generate different results.

Solutions

- Hardware: versioning the hardware, drivers, and all other parts of the environment. Besides versioning, the environment should fulfill the following criteria.
 - Having the ability to return to the previous state without destroying the setup.
 - Utilizing identical versions on multiple machines.
 - Setting randomization parameters.
- Software: versioning frameworks or libraries. It is important to version both pipeline code (see Training section) and involved frameworks or libraries to ensure reproducible executions. To version library dependencies, the common strategies are as follows.
 - Using a package manager and declare versioned dependencies on libraries from stable repository, e.g., requirements.txt and pip or package.json and npm. Specifying specific releases of the dependencies, e.g., 2.3.1 instead of floating versions, e.g., 2.* is important.
 - Committing all dependencies to the code repository.
- Packaging all dependencies into virtual execution environments. This is another way to version frameworks and libraries. Furthermore, it also tracks the environment changes, such as drivers. Docker containers are a common solution.

6.2.6.4. Model

Challenges

- The parameters of neural networks are initialized with random values, so training repeatedly on the same data will not yield the same model.
- Randomness in some operations, e.g., dropout, random augmentations, mini-batch sampling, random noise introductions, etc.

Solutions

- Nondeterminism from random numbers can be controlled by explicitly seeding the random number generator used.
- Versioning models: models are usually saved as binary data files. In deep learning applications, small changes to data or hyperparameters can lead to changes in many model parameters. Therefore, versioning the deltas between different model versions may be meaningful. Any system that tracks versions of binary objects could be useful.
- Versioning model provenance: this refers to tracking all inputs to a model, including data, hyperparameters, and pipeline code with its dependencies.

6.2.6.5. Training

Challenges

- In distributed training, timing differences in distributed systems can affect the learned parameters due to the involved merging strategies.
- The lack of proper logging of parameters such as hyperparameter values, batch sizes, learning rates, etc. during the model training results in difficulty in understanding and replicating the model.
- Machine learning is an iterative process with lots of experimentation: changing the values of the parameters; checking performance of different algorithms; and fine-tuning to get good results. Recording important details once the experimental process gets long and more complicated makes recording more difficult.
- Randomness in the software: batch ordering, data shuffling, and weight initialization
- Non-deterministic algorithms such as stochastic gradient descent, Monte Carlo methods, mini-batch sampling, etc.

Solutions

- Nondeterminism from random numbers can be controlled by explicitly seeding the random number generator used.
- Versioning feature provenance: this refers to tracing how features in the training and inference data were extracted, that is, mapping data columns to the code version that was used to create them. Also, keeping individual feature generation code independent from one another.

- Versioning code: track and record changes in code and algorithms during experimentation.
- Versioning pipelines: a training pipeline can contain many stages for extracting features, calculating losses, updating parameters, and optimizing hyperparameters. Versioning all steps for tracking which versions of the individual parts went into creating a specific model is necessary. Pipeline code and hyperparameters can be expressed in normal code and configuration files and can be versioned like traditional code.
- Versioning experiments: data scientists routinely experiment with different versions of extracting features, different modeling techniques, and different hyperparameters. Tracking information about specific experiments and their results is useful. The approach focuses on keeping and comparing results, often visualized in some dashboard, e.g., MLflow.

6.2.6.6. Inference

Challenges

- Model versioning: are the models still available? Which model performs the best?
- Model provenance: what training data was used? What is the feature extraction code? What version of the ML library/framework? What hyperparameters was used?
- Model inference: will the same result be achieved using different inference approaches?

Solutions

- Versioning model provenance: these metadata provide the connection of all the processing elements. For example, a model version v12 was built from data version v17, pipeline version v4.1, etc. Such metadata can also ensure the same code (data cleaning, feature extraction, etc.) is used at the inference time that was used at training.
- Versioning deployed models: knowing which model version handles specific inputs during inference, as part of log files or audit traces, is useful. This application or user can track the model version responsible for every output over time and in the presence of A/B tests.

6.2.6.7. Tools

Many tools have been developed to help track versions of data, pipelines, and models. These tools also often help record further metadata, such as training time or evaluation results (e.g., prediction accuracy or robustness). An example follows.

- In **DVC**, all processing steps are modeled as separate stages of a pipeline. Each stage is implemented by an executable that is called with some input data and produces some output data. The stages and data dependencies are described as pipelines in some internal format, such as the following example from the DVC documentation.

```

stages:
  features:
    cmd: jupyter nbconvert --execute featurize.ipynb
    deps:
      - data/clean
    params:
      - levels.no
    outs:
      - features
    metrics:
      - performance.json
  training:
    desc: Train model with Python
    cmd:
      - pip install -r requirements.txt
      - python train.py --out ${model_file}
    deps:
      - requirements.txt
      - train.py
      - features
    outs:
      - ${model_file}:
          desc: My model description
    plots:
      - logs.csv:
          x: epoch
          x_label: Epoch
    meta: 'For deployment'
    # User metadata and comments are supported

```

Both implementations of the stages and the pipeline description are versioned in a Git repository. DVC then provides command line tools to execute individual pipeline steps or the entire pipeline at once. DVC runs the executable of each step according to the dependencies for each step and tracks the versions of all inputs and outputs as part of metadata stored in the Git repository.

- [MLflow](#) is a popular framework for tracking experiment runs. It is easy to log and track experiments and show results in dashboards. MLFlow stores results with corresponding hyperparameters for a run, source version of the pipeline, and recording training time. The dashboard is useful for comparing results of training with different hyperparameters, but it could also be used to compare different versions of the pipeline or training with different datasets.
- [Verdant](#) Researchers often use Jupyter notebooks for exploratory work that does not commonly use Git for versioning because the code might not be in cohesive incremental pieces. In addition, the file format of notebooks that include output data and images together with the cells' code in a JSON format does not lend itself easily to textual differences and many traditional version control and code review processes. Recently, there are some history recording tools designed for notebooks like Verdant. Verdant is a JupyterLab extension that automatically records the history of all experiments run in a Jupyter notebook and stores them in a tidy .ipyhistory JSON file designed to work alongside with and compliment any other version control you use, such as SVN or Git.

6.2.7. Additional Considerations and Limitations for Reproducible Deep Learning Workflows

- What is reproducible training?
 - Under the same training code, same environment, and the same training dataset, the resulting trained model yields the same results under the same evaluation criteria.
- What are the challenges of reproducible training?
 - Randomness in the software: batch ordering, data shuffling, and weight initialization.
 - Non-determinism in the hardware: parallel floating-point calculation in GPU and auto-tuning features in libraries such as CUDA, cuDNN.
 - Lack of systematic guidelines: system information to support reproducibility, e.g., resources (dataset environment), software (source code), metadata (dependencies), execution data (execution results, log), and how to manage them (Git, DVC, MLflow).

6.2.8. Lessons Learned

The reproducibility of deep learning applications and experiments has strong impacts and implications for science. Reproducibility brings credibility and trust into the system which are the foundations of scientific research and advancement. However, based on work in Testbed 18, the difficulties related to generating the same experimental results in deep learning applications due to its the intrinsic (models, optimization methods, etc.) and extrinsic (hardware, software, etc.) factors were identified. To address the reproducibility crisis, more focus and analysis on reproducibility are required in deep learning research. For example, what details regarding the application are needed to be provided to guarantee the reproducibility? How to analyze, estimate, and control the output differences from the details? It is important to develop a standard and systematic way to evaluate the reproducibility in deep learning as it has become a part of our everyday life?

6.3. Ecere

6.3.1. Goals of Participation

Ecere's participation in Testbed 18 was to produce one or more reproducible FAIR workflow(s) following the approach to define workflows described in [OGC API – Processes – Part 3: Workflows and Chaining](#).

Task objectives were as follows.

- The reproducible workflow implementation will help establish best practices for reproducible science.
- The task will provide use cases for Coverages and Processes SWG, EOXP and Workflows DWG.
- The task use cases will provide developers with a potential approach to implement, define, and share reproducible workflows.

6.3.2. Contributed Workflow and Architecture

The Reproducible Workflows implementation allows users to define, execute, and retrieve outputs from workflows. Ecere has provided workflow execution endpoints providing deterministic results. The workflow accepts inputs from external OGC APIs with the ability to retrieve results and trigger processing for a region and/or time and/or resolution of interest. Several encodings are supported such as GeoTIFF, GeoJSON, and GNOSIS Map Tiles, among others.

The workflow scenarios explored for this task were crop classification using machine learning as well as coastal erosion in the Arctic. As a result of significant challenges encountered for each of these scenarios, the ability to demonstrate these workflows at the end of Testbed 18 was still limited, and efforts were ongoing to complete the development of the planned capabilities.

Additional processing capabilities supported as part of the capability to submit ad-hoc workflows to the instance deployed by Ecere include:

- performing basic analytics e.g., derived fields like index calculations from multi-band imagery, filtering data (for instance to merge multiple scenes or eliminate values outside of a certain threshold);
- gridding point clouds;
- tracing elevation contours;
- routing calculations; and
- server-side map rendering.

Ecere's instance uses the GNOSIS Map Server which provides support for several OGC API Standards (draft and approved), including the following.

- OGC API – Processes (including *Part 3: Workflows and chaining*)
- OGC API – Features (including *Part 2: CRS by reference* and *Part 3: Filtering*)
- OGC API – Coverages
- OGC API – Tiles

- OGC API – Discrete Global Grid System
- OGC API – 3D GeoVolumes
- OGC API – Maps

In conjunction with this task, significant progress was made on improving the *OGC API – Processes – Part 3* draft candidate Standard itself, writing the requirements and organizing them into requirements classes with enough clarity so as to facilitate the development of additional conforming implementations.

6.3.2.1. OGC API Processes – Part 3: Workflows and Chaining Diagram

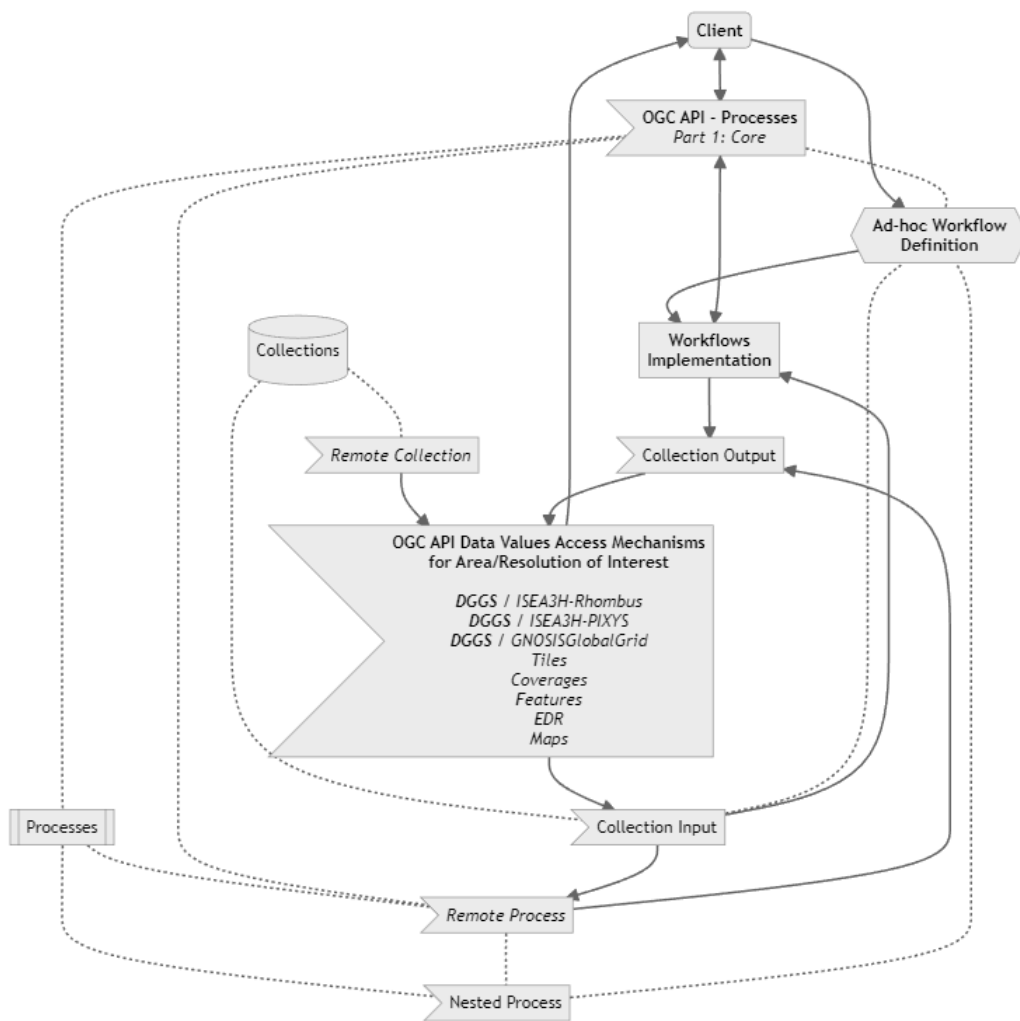


Figure 7 – Coastal Erosion Workflow Diagram

6.3.3. Coastal Erosion Workflow

For Testbed 18, in collaboration with the University of Calgary, Ecere expressed the Coastal Erosion model developed by Perry Peterson in the context of the Federated Marine SDI (FMSDI) Pilot Phase 3 scenario as an *OGC API – Processes* execution request leveraging the *Part 3: Workflows and Chaining* extension. Ecere also mapped this workflow to a flow diagram, providing a clear visual illustration of how the different data sources are integrated. For the FMSDI pilot, Ecere provided a DGGs client accessing and visualizing the output from the coastal erosion susceptibility workflow executed on the University of Calgary's DGGs Data Integration Server, triggering on-demand processing. This DGGs server at the University of Calgary quantizes and samples the data to an Icosahedral Equal Area aperture 3 Hexagonal (ISEA3H) DGGs, parameterizes the data for the erosion model, and finally serves the resulting collection through a DGGs API.

The workflow integrated vector features and coverage data from four different sources:

- ArcticDEM (<https://www.pgc.umn.edu/data/arcticdem/> – NGA)
- Global Land Cover (<https://www.usgs.gov/centers/eros/science/usgs-eros-archive-land-cover-products-global-land-cover-characterization-glcc> – USGS)
- Geologic map of Alaska (Surficial Geology) (<https://www.usgs.gov/centers/alaska-science-center/science/geologic-map-alaska> – USGS)
- Circum-Arctic permafrost and ground ice (<https://nsidc.org/data/ggd318/versions/2> – NSIDC)

The client accessed the data values using a dual DGGs of ISEA3H that was named *ISEA9R* in the FMSDI project, using rhombic zones and an aperture 9. Other DGGs and reference systems, such as the ISEA3H PIXYS indexing or the GNOSIS Global Grid, or other access mechanisms such as *OGC API – Tiles, Coverages* or *EDR*, could also be used by clients to trigger the same workflow if the server would provide them.

As well as adding support for using the output of a process as an input to another, the *Processes – Part 3* extension allows an *OGC API* collection to be an input of a process, or the output of a process or of the overall workflow.

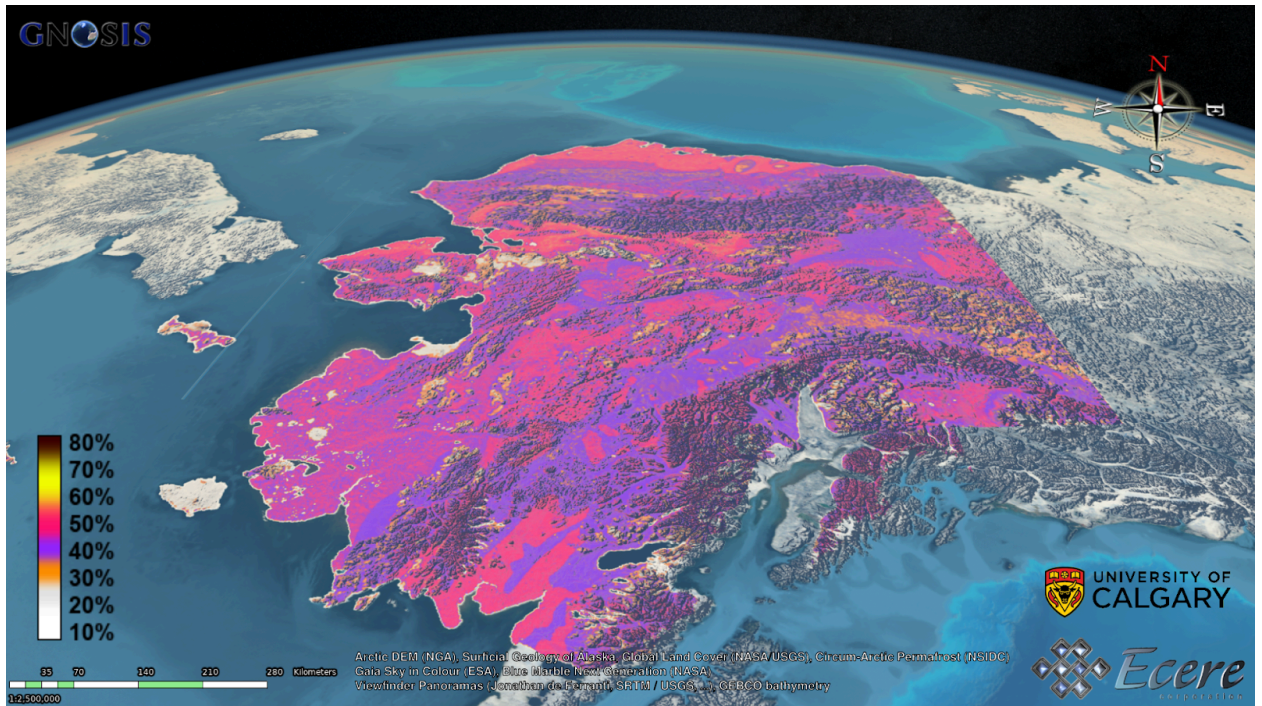


Figure 8 – Visualizing the output of coastal erosion workflow Ecere’s GNOSIS Cartographer client (large scale view)



Figure 9 – Visualizing the output of coastal erosion workflow Ecere’s GNOSIS Cartographer client (closer view)

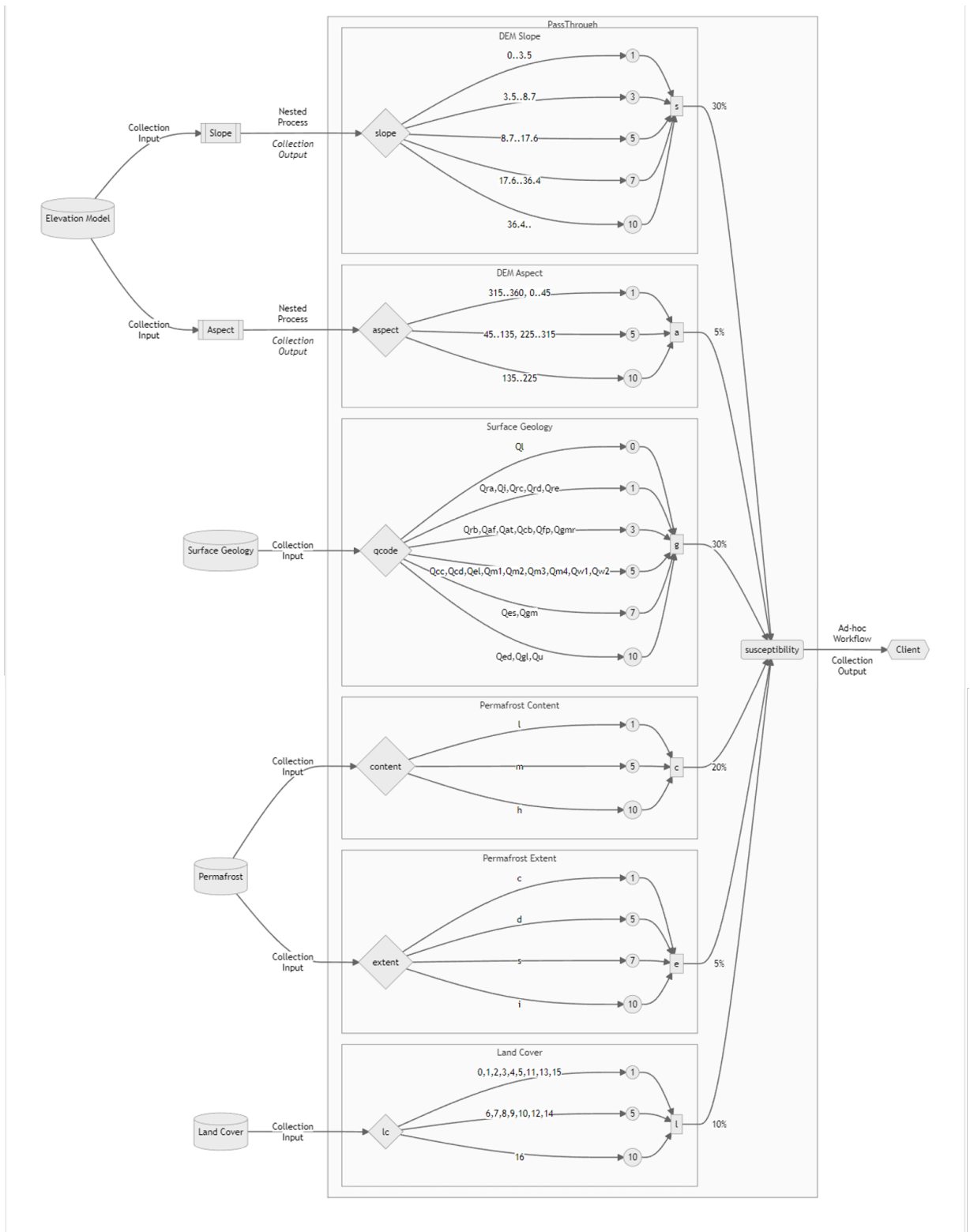


Figure 10 – Visualizing coastal erosion workflow definition as a flow diagram

The visualization also integrated additional datasets:

- Blue Marble Next Generation (<https://earthobservatory.nasa.gov/features/BlueMarble> – NASA)
- Black Marble (<https://blackmarble.gsfc.nasa.gov/> – NASA)
- GEBCO (https://www.gebco.net/news_and_media/gebco_2014_grid.html)
- Viewfinder Panoramas (<https://viewfinderpanoramas.org> – Jonathan de Ferranti (SRTM / USGS and other sources)
- Gaia Sky in Colour (<https://sci.esa.int/web/gaia/-/60196-gaia-s-sky-in-colour-equirectangular-projection> – ESA)

Ecere initiated the development of the necessary components to execute this workflow in its GNOSIS Map Server, such as the *PassThrough* process and the ability to specify field modifiers in execution requests. However, at the end of the initiative some additional work still needed to be completed, in particular regarding the ability to perform spatial join operations on both coverage and vector features. The efforts to complete this functionality and allow to execute the sample workflow shown below were ongoing at the time of publishing this report.

Once Ecere's implementation of the coastal erosion workflow is completed, it will provide an opportunity to compare the output side-by-side with the one produced by the University of Calgary's DGGs server, and validate the reproducibility of the workflow on two different implementations, each using a different Discrete Global Grid System (ISEA3H and GNOSIS Global Grid) to perform the data integration and calculations.

6.3.3.1. JSON Execution Request with CQL2

In this example scenario, expressions written using CQL2, the OGC Common Query Language, are used to map properties from the datasets to a numerical estimation of how much they could be contributing to erosion, and to specify final weights in order to output a single susceptibility percentage value.

```
{
  "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
  "inputs": {
    "data": [
      {
        "process": "https://maps.gnosis.earth/ogcapi/processes/Slope",
        "inputs": {
          "dem": { "collection": "https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama" }
        },
        "properties": { "s" : "slope >= 36.4 ? 10 : slope >= 17.6 : 7 : slope >= 8.7 ? 5 : slope >= 3.5 ? 3 : 1" }
      },
      {
        "process": "https://maps.gnosis.earth/ogcapi/processes/Aspect",
        "inputs": {
          "dem": { "collection": "https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama" }
        },
        "properties": { "a" : "aspect >= 315 or aspect < 45 ? 1 : aspect >= 225 or aspect < 135 : 5 : 10" }
      }
    ]
  }
}
```

```

    },
    {
      "collection": "https://maps.gnosis.earth/ogcapi/collections/ArcticP
ermafrost",
      "properties": {
        "e": "extent = 'c' ? 1 : extent = 'd' ? 5 : extent = 's' ? 7 :
extent = 'i' ? 10 : null"
        "c": "content = 'l' ? 1 : content = 'm' ? 5 : content = 'h' ? 10 :
0"
      }
    },
    {
      "collection": "https://maps.gnosis.earth/ogcapi/collections/Landsat
7LandCover",
      "properties": { "l" : "lc in(0,1,2,3,4,5,11,13,15) ? 1 : lc
in(6,7,8,9,10,12,14) ? 5 : lc = 16 ? 10 : 0" }
    },
    {
      "collection": "https://maps.gnosis.earth/ogcapi/collections/AlaskaS
urficialGeology",
      "properties": {
        "g":
          "qcode = 'Ql' ? 0 : qcode in ('Qra','Qi','Qrc', 'Qrd', 'Qre') ? 1 :
qcode in ('Qrb','Qaf', 'Qat', 'Qcb','Qfp','Qgmr') ? 3 : qcode in
('Qcc','Qcd','Qel','Qm1', 'Qm2','Qm3','Qm4','Qw1','Qw2') ? 5 :
qcode in ('Qes','Qgm') ? 7 : qcode in ('Qed','Qgl','Qu') ? 10 : 0"
      }
    }
  ],
  "properties": { "susceptibility" : "0.30 * s + 0.05 * a + 0.05 * e + 0.20
* c + 0.10 * l + 0.30 * g" }
}

```

6.3.4. Crops Classification Workflow

Ecere originally intended to demonstrate reproducibility in the context of a crop classification workflow previously developed for the 2020-2021 *Modular OGC API Workflows (MOAW)* project led by Ecere in collaboration with several other OGC members and other organizations, for which financial support was provided by Natural Resources Canada’s GeoConnections program.

A major difficulty faced with this workflow was reliance on input sentinel-2 data Level 2A data provided from an OGC API – *Coverages* end-point. A persisting issue with the EuroDataCube implementation that previously provided this capability prevented its use. As an alternative, Ecere spent efforts on improving the data cube capabilities of the GNOSIS Map Server to serve that purpose, with a goal to provide efficient access to sentinel-2 data sourced from Cloud Optimized GeoTIFF and cataloged with STAC metadata. The development of those capabilities presented its own set of challenges, described below, and still remained to be completed at the end of Testbed 18. The ability to demonstrate the crop classification workflow was therefore still pending the completion of those datacube capabilities.

The output from previous executions of this workflow as well as training data are shown below. The workflow uses a Random Forest machine learning prediction algorithm to classify crops based on Earth Observation imagery from ESA sentinel-2 across multiple seasons.

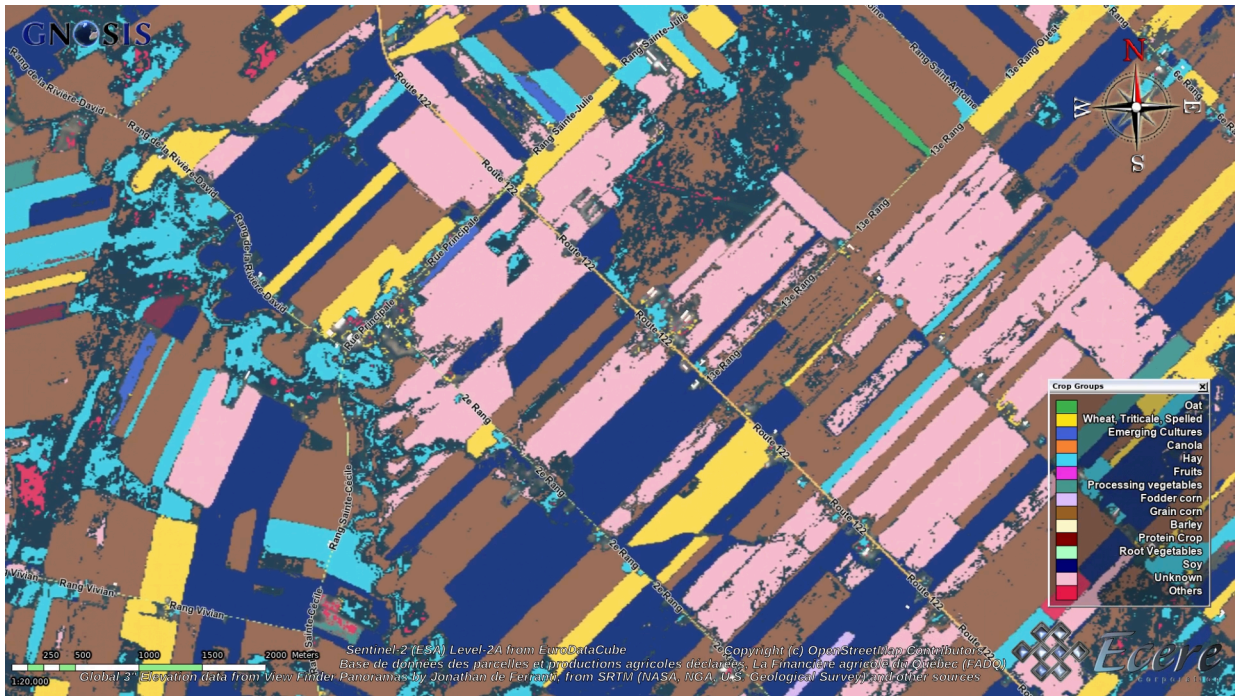


Figure 11 – Crop Classification Workflow output visualized in Ecere’s GNOSIS Cartographer client (from 2020-2021 MOAW GeoConnections project)

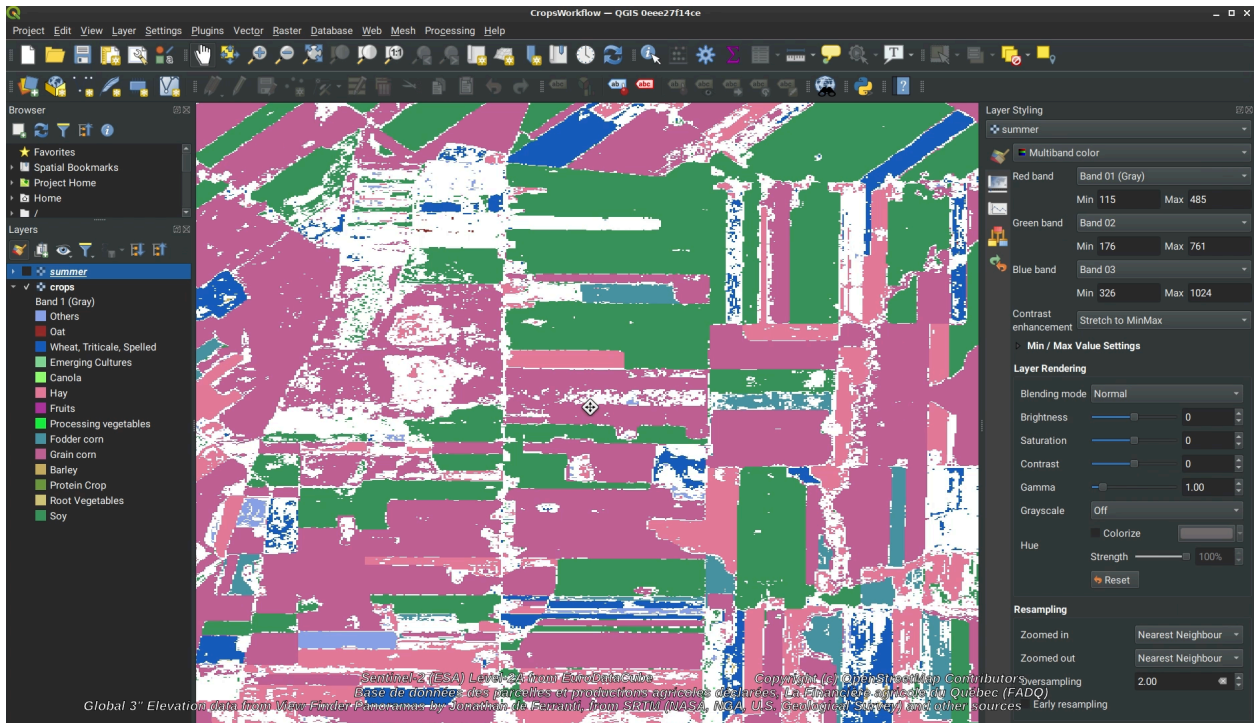


Figure 12 – Crop Classification Workflow output visualized in QGIS client (from 2020-2021 MOAW GeoConnections project)

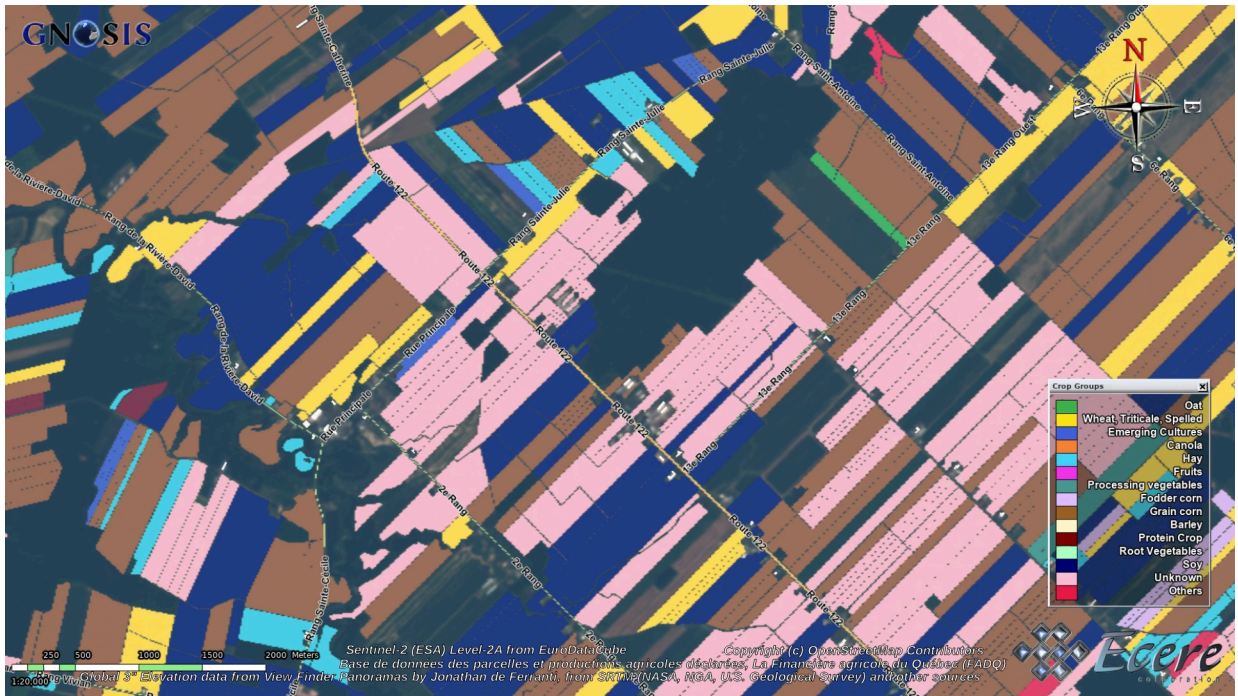


Figure 13 – Parcels training data for crop classification workflow (from 2020-2021 MOAW GeoConnections project)

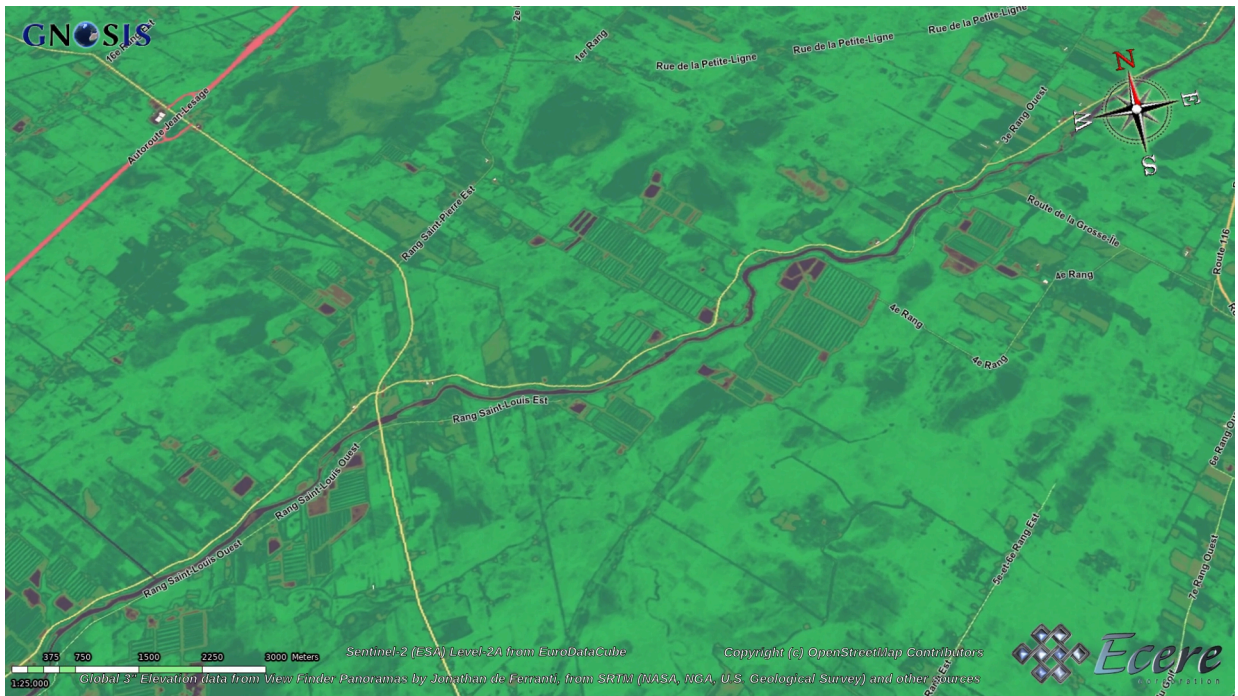


Figure 14 – Enhanced Vegetation Index used as training data for crop classification workflow (from 2020-2021 MOAW GeoConnections project)

The workflow referenced a sentinel-2 OGC API collection as an input. The machine learning model was trained on the *Parcels and Reported Agricultural Productions Database* of *La Financière agricole du Québec*. Some aspects of the original workflow were initially hardcoded directly in the

RFClassify process. These included the selection of time intervals for three different season in a specific year, cloud masking, and the calculation of an Enhanced Vegetation Index from specific bands. The same data preparation steps were also used as input to training the Random Forest model. The ability to express some of these steps within the workflow definition was discussed in [Processes issue #279](#) An eventual extension to *Processes* facilitating the progressive training, selection, testing and management of a machine learning model was also suggested.

A prototype workflow from these discussions, intending to be deployed as a workflow taking in as inputs a *modis_data* (low spatial resolution, high temporal resolution) and a *sentinel2_data* (high spatial resolution, low temporal resolution) follows.

```
{
  "process" : "https://research-alpha.org/ogcapi/processes/randomForestPredict",
  "inputs" : {
    "trainedModel" : "https://research-alpha.org/ogcapi/models/sentinel2ModisLandCover",
    "data" :
      [
        { "$ref" : "#/components/monthlyInput", "{month}" : 1 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 2 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 3 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 4 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 5 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 6 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 7 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 8 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 9 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 10 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 11 },
        { "$ref" : "#/components/monthlyInput", "{month}" : 12 }
      ]
  },
  "components" :
  {
    "modis":
    {
      "$input" : "modis_data",
      "format": { "mediaType": "application/netcdf" },
      "datetime" : { "year" : { "{datetime}.year" }, "month" : "{month}" }
    },
    "sentinel2":
    {
      "$input" : "sentinel2_data",
      "format": { "mediaType": "image/tiff; application=geotiff" },
      "filter" : "scene.cloud_cover < 50 and cell.cloud_cover < 15",
      "sortBy": "cell.cloud_cover(desc)",
      "datetime" : { "year" : { "{datetime}.year" }, "month" : "{month}" }
    },
    "monthlyInput":
    {
      { "$ref" : "#/components/modis" },
      { "$ref" : "#/components/sentinel2" },
      {
        "process" : "https://research-alpha.org/ogcapi/processes/PassThrough",
        "inputs" : {
          "data" : { "$ref" : "#/components/sentinel2" },
          "properties" : { "evi" : "2.5 * (B08 - B04) / (1 + B08 + 6 * B04 + -7.5 * B02)" }
        }
      }
    }
  }
}
```

```
}  
  }  
    }  
      }
```

6.3.5. Considerations for Workflow Reproducibility:

A definition of a workflow using *OGC API – Processes – Part 3: Workflows & Chaining* facilitates reproducibility in several ways:

- sharing the definition together with the resulting data, whether that data is pre-processed or processed on demand;
- identifying the *Processes* and *Collections* involved by URIs;
- directly embedding algorithms deriving field values as expressions;
- re-executing the workflow from different clients; and
- re-executing from different servers implementing the same processes.

This approach provides the ability to reproduce output, verify reproducibility, and re-produce (re-use) the science for different areas and/or resolutions of interest, or with similar data sources.

Each of the URI used within a workflow, as well as the workflow itself, could also possibly be mapped to a Digital Object Identifier (DOI).

6.3.6. Datacube Implementation

Ecere extended the Datacube capabilities of its *GNOSIS Map Server* to provide integration with very large repositories of Earth Observation imagery hosted in the cloud available as Cloud Optimized GeoTIFF and indexed using STAC.

STAC client capabilities were developed, but several limitations with server implementations led to a focus on using a local database to store and rapidly query the assets metadata.

A relational database following the STAC data model as well as the *STAC Electro-Optical Extension Specification* was pre-populated with the metadata retrieved from the Amazon Web Services deployment of Cloud Optimized GeoTIFF for sentinel-2 Level 2A managed by Element 84.

Support for efficiently accessing the data relevant to a particular area and resolution of interest from Cloud Optimized GeoTIFF was improved by implementing support for accessing overviews as well as using HTTP range requests.

A goal of the implementation is to allow efficiently accessing the data at any scale. This proved to be challenging since tens of thousands of granules need to be processed to cover the whole

globe for any time instant. The following images show some initial success in generating a low-resolution covering the entire Earth.

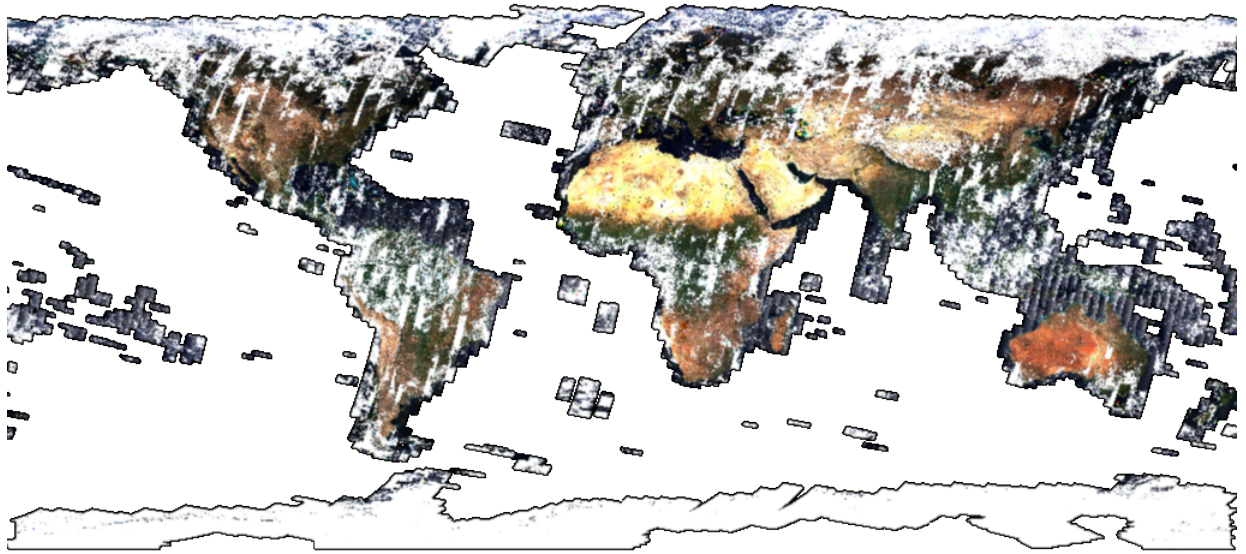


Figure 15 – Sentinel-2 (ESA) Datacube presented as an OGC API collection on maps.gnosis.earth, sourced from Cloud Optimized GeoTIFF managed by Element 84 hosted on Amazon Web Services

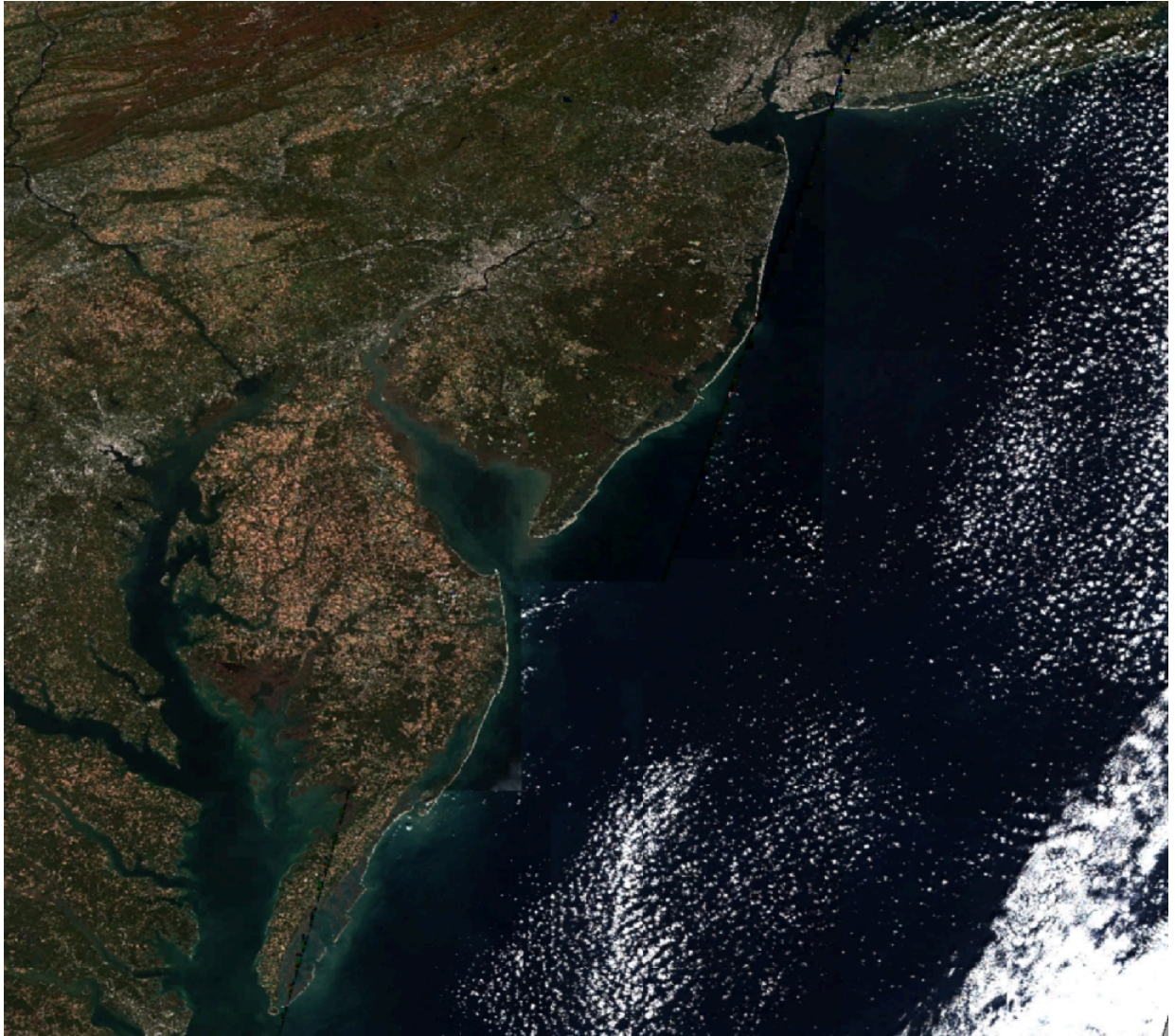


Figure 16 – Sentinel-2 (ESA) Datacube subset as OGC API collection on maps.gnosis.earth representing East Coast US, sourced from Cloud Optimized GeoTIFF managed by Element 84 hosted on Amazon Web Services

For these initial experiments, the low-level overviews were used, which in addition to only including the red, green and blue bands, are limited to an 8-bit precision value for each of those bands.

6.3.6.1. STAC relational database

By using a relational model to cache the STAC metadata, content for the sentinel-2 COG granules sourced from 20 million JSON files taking up 200 gigabytes could be stored in a single SQLite file taking up only 2.2 gigabytes once compressed. In addition to reducing the storage space required for caching the metadata from large archives, this database format facilitates and minimizes the bandwidth requirements for exchanging large amount of STAC metadata.

The list of tables set up for the STAC relational database included the following:

ASSETS_BASE_URL	STRINGS_NAME	STRINGS_type
ITEMS	STRINGS_baseurl	eo_bands
ITEMS_ASSETS	STRINGS_constellation	eo_bands_combinations
ITEMS_rtree	STRINGS_platform	proj_shape
ITEMS_rtree_node	STRINGS_rel_href	proj_transform
ITEMS_rtree_parent	STRINGS_role	
ITEMS_rtree_rowid	STRINGS_title	

Entries from tables such as those holding string values are referenced from other tables using integer keys, and are employed to reduce redundancy and conserve space. The item assets URLs are separated into base URLs and relative *href* strings, each stored in their corresponding table and referenced by an integer key in the ITEMS_ASSETS table. The ITEMS table contains the metadata properties at the scene or granule level, whereas the ITEMS_ASSETS table contains the properties for the individual assets of each items. In the case of the sentinel-2 data hosted on AWS, individual assets are available for each band, as well as for additional resources such as thumbnails and metadata. An R-tree is used to index the items for efficient spatial queries.

The ITEMS table is created with the following SQL command.

```
CREATE TABLE ITEMS (
  ID INTEGER PRIMARY KEY,
  `constellation` INTEGER REFERENCES STRINGS_constellation,
  `created` TEXT,
  `datetime` TEXT,
  `eo:cloud_cover` REAL,
  `itemID` TEXT UNIQUE,
  `gsd` INTEGER,
  `platform` INTEGER REFERENCES STRINGS_platform,
  `proj:epsg` INTEGER,
  `sentinel:data_coverage` REAL,
  `sentinel:grid_square` TEXT,
  `sentinel:latitude_band` TEXT,
  `sentinel:product_id` TEXT,
  `sentinel:sequence` TEXT,
  `sentinel:utm_zone` INTEGER,
  `sentinel:valid_cloud_cover` INTEGER,
  `updated` TEXT,
  `view:off_nadir` INTEGER)
```

The ITEMS_ASSETS table is created with the following SQL command.

```
CREATE TABLE ITEMS_ASSETS (
  ID INTEGER PRIMARY KEY,
  ITEM INTEGER REFERENCES ITEMS,
  `NAME` INTEGER REFERENCES `STRINGS_NAME`,
  `title` INTEGER REFERENCES `STRINGS_title`,
  `type` INTEGER REFERENCES `STRINGS_type`,
  `role` INTEGER REFERENCES `STRINGS_role`,
  `rel_href` INTEGER REFERENCES `STRINGS_rel_href`,
  asset_base_href INTEGER REFERENCES ASSETS_BASE_URL,
  gsd INTEGER,
  `proj:shape` INTEGER REFERENCES proj_shape,
  `proj:transform` INTEGER REFERENCES proj_transform,
  `eo:bands` INTEGER REFERENCES eo_bands_combinations)
```

The name string table for the sentinel-2 metadata identifying each different band and other asset types is shown below.

ID s

```

-- -----
1  AOT
2  B01
3  B02
4  B03
5  B04
6  B05
7  B06
8  B07
9  B08
10 B09
11 B11
12 B12
13 B8A
14 SCL
15 WVP
16 info
17 metadata
18 overview
19 thumbnail
20 visual

```

An entry in ITEMS appears as follows.

```

          ID = 1
    constellation = 1
          created = 2020-08-28T03:31:33.284Z
          datetime = 2019-02-22T21:36:06Z
    eo:cloud_cover = 100.0
          itemID = S2A_1EDP_20190222_0_L2A
          gsd = 10
          platform = 1
          proj:epsg = 32701
    sentinel:data_coverage = 2.65
    sentinel:grid_square = DP
    sentinel:latitude_band = E
    sentinel:product_id = S2A_MSIL2A_20190222T212231_N0211_R014_T01EDP_
20190222T224440
          sentinel:sequence = 0
          sentinel:utm_zone = 1
    sentinel:valid_cloud_cover = 1
          updated = 2020-08-28T03:31:33.284Z
    view:off_nadir = 0

```

An entry in ITEMS_ASSETS appears as follows.

```

          ID = 9665
          ITEM = 484
          NAME = 5
          title = 5
          type = 1
          role = 1
          rel_href = 5
    asset_base_href = 1450
          gsd = 10
          proj:shape = 2
    proj:transform =
          eo:bands = 4

```

6.3.6.2. Draft Coverages Standard

One mechanism for accessing the datacube is the draft *OGC API – Coverages Standard*. As defined in *OGC API – Coverages – Part 1: Core*, coverage resources follow the resource path `{datasetAPI}/collections/{collectionId}/coverage`. The GET HTTP method returns the coverage, and all components supported by the selected representation. A particular encoding is selected using HTTP content negotiation, for example GeoTIFF.

A number of query parameters are available to refine queries. The `scale-factor`, `scale-axes` and `scale-size` allow to request a downsampled version of the coverage. The `subset` parameter supports retrieving only a subset of a coverage, with ranges for axes named by their abbreviations, as defined in the CRS. In the case of EPSG:4326 and CRS84, those abbreviations are *Lat* for latitude, and *Lon* for longitude. An example subset parameter: `?subset=Lat(37:41),Lon(-77:-72.5)`. The `properties` parameter will dictate that only a subset of fields (bands) available be returned. An example: `?properties=B02,B03,B04`. The `bbox` parameter defined in *OGC API – Common – Part 2: Geospatial Data* is an alternative mechanism to select the spatial subset of the coverage to retrieve. An example: `?bbox=78.750000,30.937500,81.562500,33.750000`. The `datetime` parameter is an alternative mechanism to select the temporal subset of the coverage to retrieve, as an interval (trim operation) or for a time instant (slice operation). An example: `?datetime=2022-11-23`.

Deriving new field values from existing ones is also possible through an extension. This capability is detailed in *OGC API – Processes – Part 3: Workflows & Chaining* and may also eventually be defined as an extension to *OGC API – Coverages*, as query parameters. This supports defining a custom field as an evaluation of arithmetic expression, using specified coverage fields as input variables to the expression. For example: [https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/coverage.png?properties=\(B8-B4\)/\(B8+B4\)](https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/coverage.png?properties=(B8-B4)/(B8+B4)).

Explicitly naming the derived field may be possible in a future implementation e.g., as `NDVI:(B8-B4)/(B8+B4)`.

6.3.6.3. Challenges

Currently, STAC servers will not return a large number of items, and the draft *OGC API Features – Part 3: Filtering Standard* is not yet well supported.

Even once implemented, there remains the issue that a large number of scenes (whether on a first retrieval of entire set or for delta updates) are still much larger than required, mainly due to not being relational in nature, requiring the transfer of the same data several times for multiple occurrences.

The development to support access to the full set of Sentinel-2 COGs bands, overviews, and full resolution turned out to be a major effort that was still under development at the end of the task.

6.3.7. Lessons Learned

There is a need for more efficient transfers of large sets of STAC metadata. Such transfers should be based on a relational data model, and ideally retrieved as compressed binary data. A use case is the ability to provide efficient local filtering capabilities based on that metadata as part of an OGC API implementation presenting a large collection of data such as the sentinel-2 archive made up of millions of scenes / granules. Such capability would allow to rapidly mirror an entire STAC metadata catalog in bulk, then maintain it up to date by synchronizing the removal, update and addition of entries.

Workflows defined using extended OGC API – Processes execution requests specifying input data and processes via URLs provide the ability to reproduce outputs and verify reproducibility and facilitate reuse for different areas and resolutions of interest, as well as with similar data sources.

6.4. GeoLabs

6.4.1. Goals of Participation

To support OGC SWGs / DWGs efforts to advance the Standards Baseline, GeoLabs performed the following as part of OGC's Testbed 18.

- Investigated ways to integrate the OGC API – Processes – Part 2: Deploy, Replace, Undeploy within the current ZOO-Kernel implementation.
- Review of current specifications.
- Interactions with the OGC API – Process Standards Working Group (SWG).
- Investigated ways to integrate the OGC API – Processes – Part 3: Workflows within the current ZOO-Kernel implementation.

For Developers:

- The work completed during Testbed 18 provides a ready-to-use environment for integrating new features within the tale for documenting and demonstrating new features in the future development of the involved Open Source software (ZOO-Project and Mapserver)

The demonstrated reproducible workflow from GeoLabs enables the end user to run OGC API – Processes and OGC API – Feature server instances and interact with them either from within the tale or remotely. This workflow is implemented without the user needing to install

anything and only relying on the wholetale.org infrastructure, which ensures reproducibility of the environment to execute the given workflows.



Figure 17 – GeoLabs Wholetale Workflow

6.4.2. Contributed Workflow and Architecture

GeoLabs evaluated the use of the WholeTale.org infrastructure for providing the following tales.

- [ZOO-Project Introduction tale](#)
- [Deploy ADES on Microsoft Azure Kubernetes cluster](#)
- [TB18-MS-OGC-Tale](#)

GeoLabs contributed to the OGC API – Processes – Part 2: Deploy, Replace, Undeploy Process providing schema and a tentative implementation (<http://tb18.geolabs.fr:8080/djay/wps3/>) and to Part 3: Workflow Draft Standards by providing schema and a tentative implementation (to be determined).

6.4.2.1. Publishing a tale on Wholetale.org illustrating the OGC Web Services' use

Geolabs worked on multiple [Tales](#) using the infrastructure offered by <https://www.wholetale.org>. A tale is defined as a reproducible environment that runs on Wholetale.org or a local machine. The initial goal was to review how to integrate the MapServer and ZOO-Project (including OrfeoToolBox support) Open Source Software which implements OGC APIs, to then be able to incorporate a complex workflow within a tale.

The initial goal was to create what GeoLabs envisioned as a “FAIRy tale” where a user would get a transparent introduction to the OGC API – Processes interactions by performing deep learning with real-world remote sensing images in a reproducible manner. The idea was to integrate the [OTBTF](#) as a process within the ZOO-Project environment and produce the associated Jupyter notebook to then ease the interactions and present the model training and the application results.

The [ZOO-Project-Introduction-Tale](#) was the first tale made during this investigation. This tale contains multiple Open Source Software toolkits:

- [OrfeoToolBox 6.6](#)
- [MapServer 7.7-dev](#) (was upcoming 8.0, now out)
- [ZOO-Project 2.0-dev](#)

When development began on this tale, MapServer 8.0 which introduces OGC API – Features support was not yet available. GeoLabs planned to use Version 8.0 to illustrate the automated publication of API endpoints available in the ZOO-Project resulting from past participation in the [OGC OSGeo ASF Code Sprint in March 2022](#). GeoLabs used the ASF Code Sprint version during Testbed 18 by relying on the latest MapServer main branch available on GitHub that would not be conducive to a reproducible workflow due to active development. In addition, the integration within the ZOO-Project required minor modifications that GeoLabs contributed back to the MapServer project. The GeoLabs fork of the MapServer GitHub repository could be used as a stable source to build the tale.

In addition to the software, GeoLabs successfully integrated the [MapServer OGC Web Services Workshop Material](#) into the tale, translating it to run from the Jupyter notebook environment. Once this task was achieved, work was conducted to include the [OrfeoToolBox Workshop 2018](#) material within the tale as well.

The screenshot below shows a sample part of the OrfeoToolBox Workshop Material integration, an example of extracting a message hiding in the image using the BandMath application.

Message 4

In `image4.tif`, a message has been hidden in the 2 least significant bits of the image.

This modification can not be detected by human eyes, but could be revealed by isolating the values of those 2 bits.

Use the `BandMath` application to isolate the 2 least significant bits in the image (encoded on 12 bits), to reveal the message.

Note: The `rint()` function allows to round a floating point value to nearest integer in `BandMath` application

Solution 4

To reveal the 4th message, we are going to isolate the 2 least significant bits using the `BandMath` application:

```
[107]: !otbcli_BandMath -il ./WorkshopData/stegano/image4.tif \
      -out WorkshopData/stegano/output_decoded4.tif \
      -exp "imbl-4*rint(imbl/4)"

2022-07-08 09:42:29 (INFO): Loading kwl metadata from official product in file ./WorkshopData/stegano/image4.tif
2022-07-08 09:42:29 (INFO): Default RAM limit for OTB is 128 MB
2022-07-08 09:42:29 (INFO): GDAL maximum cache size is 1600 MB
2022-07-08 09:42:29 (INFO): OTB will use at most 8 threads
2022-07-08 09:42:29 (INFO): Image #1 has 1 components

2022-07-08 09:42:29 (INFO): Estimated memory for full processing: 76.2558MB (avail.: 128 MB), optimal image partitioning: 1 blocks
2022-07-08 09:42:29 (INFO): File WorkshopData/stegano/output_decoded4.tif will be written in 1 blocks of 2000x2000 pixels
Writing WorkshopData/stegano/output_decoded4.tif...: 100% [*****]2022-07-08 09:42:29 (WARNING): Skipping GCPs saving to prevent GDAL from
assigning a WGS84 projref to file (WorkshopData/stegano/output_decoded4.tif)
(0 seconds)

The following expression does not contain the 2 least significant bits, and the difference with original image thus reveals the message.
```

$$4 * \text{rint}\left(\frac{\text{imbl}}{4}\right)$$

```
[126]: import matplotlib.pyplot as plt
import cv2

w = 10
h = 10
fig = plt.figure(figsize=(20, 10))
columns = 2
rows = 1
tiffs=["WorkshopData/stegano/image4.tif", "WorkshopData/stegano/output_decoded4.tif"]
for i in range(1, columns*rows + 1):
    img = cv2.imread("./"+tiffs[i-1], cv2.IMREAD_UNCHANGED)
    subPlot=fig.add_subplot(rows, columns, i)
    subPlot.set_title(tiffs[i-1])
    plt.imshow(img)
plt.show()
```

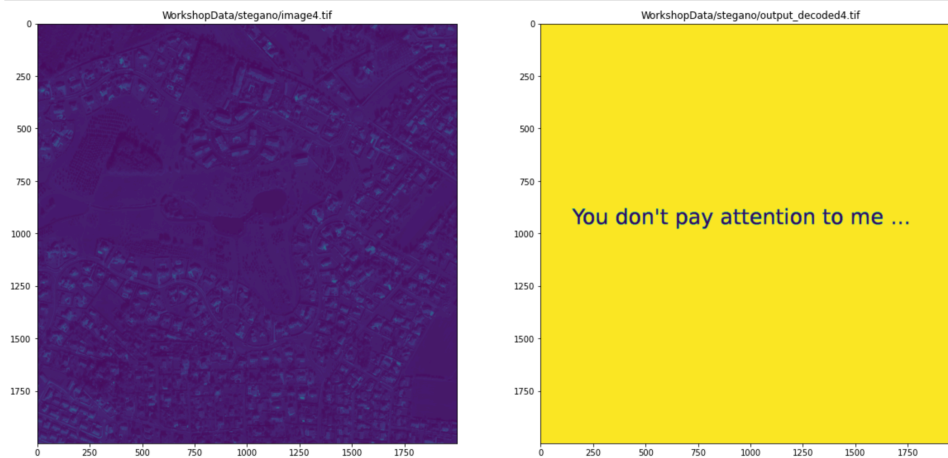


Figure 18 – GeoLabs OTB Workshop Material integration

After successfully incorporating the Open Source Software and the various workshop materials, issues that were identified in using the ZOO-Project from within the tale were addressed. By doing so, GeoLabs realized that having everything in the same tale leads to having a critical size for the container to be run. Also, the time required to build the tale and then deploy it on Wholetale.org is significant. Consequently, the participants decided to split the tale into smaller parts, starting with the MapServer OGC Web Services Workshop.

The tale was named TB18-MS-OGC-Tale. This tale embeds the MapServer 8.0 release and contains the material ported to the Jupyter notebook. Additionally, it provides introductions on interacting with the following OGC Web Services: WMS, WFS, WCS, and the OGC API – Features – Part 1: Core.

The first step of the Index notebook consists of starting a web server from within the tale. Then, the user accesses the `OGC_Web_Services_Workshop` notebook, which contains the instructions

first to fetch the required dataset and adapt the MapServer's map files to work with the 8.0 version. Then the user adds an identifier to the rivers dataset.

The figure below illustrates a trivial example of the use of the WMS GetLegendGraphic and GetMap requests combined with OGC Style Layer Description (SLD) styles that can be applied on demand using the dedicated select list.

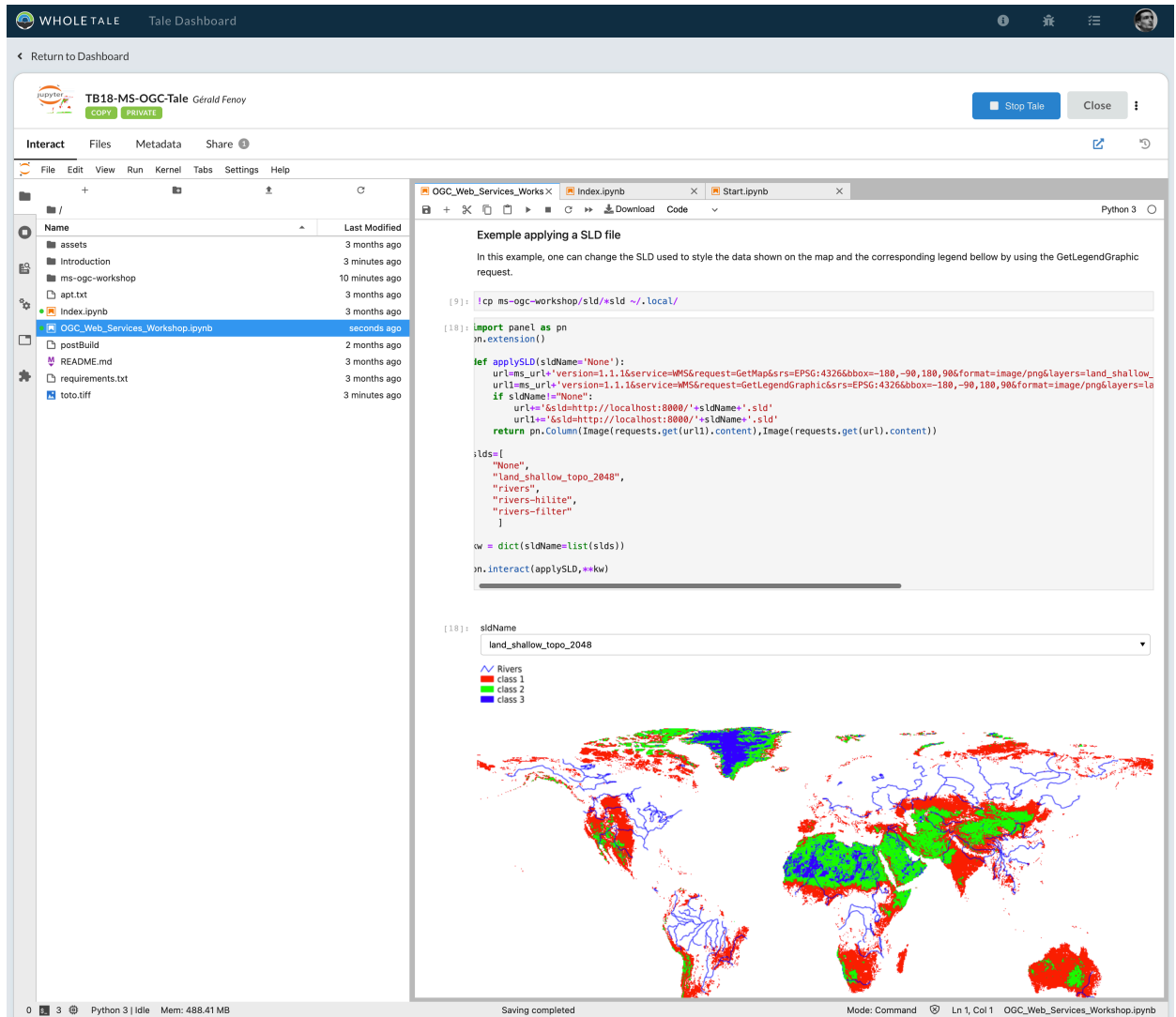


Figure 19 – GeoLabs MapServer OGC Web Map Service

The figure below illustrates how to interact with an OGC API – Features – Part 1: Core Server instance to show the collection's items in an HTML table and display them on top of the OpenStreetMap base layer.


```
[166]: import geopandas as gpd
from pyproj import CRS
import requests
import geojson

# Specify the url for web feature service
url='http://localhost:8000/cgi-bin/mapserv?map='+os.environ['REPO_DIR']+'/MapServer/ms-ogc-workshop/service/config.map'

# Specify parameters (read data in json format).
params = dict(service='WFS',
              version='2.0.0',
              request='GetFeature',
              typeName='rivers',
              outputformat='geojson'
            )

# Fetch data from WFS using requests
r = requests.get(url, params=params)

#print(r.content)

# Create GeoDataFrame from geojson
data = gpd.GeoDataFrame.from_features(geojson.loads(r.content))
# Clean overall cell
# Check the data
data.head()
```

```
[166]:
```

	geometry	NAME	SYSTEM
0	LINestring (124.00678 56.47258, 123.25956 56.6...	Aidan	Lena
1	MULTILINESTRING ((-61.27730 -3.60706, -60.6846...	Amazon	Amazon
2	LINestring (73.98818 37.49952, 73.52595 37.528...	Amu Darya	
3	LINestring (122.63956 49.99730, 120.47874 49.2...	Amur	
4	LINestring (105.07841 51.93053, 103.92959 51.7...	Angara	

```
[330]: import folium
import panel as pn

pn.extension(sizing_mode="stretch_width")

m = folium.Map(location=[40, 10], zoom_start=4, control_scale=True, prefer_canvas=True)
folium_pane = pn.pane.plot.Folium(m, height=400)

style = {'fillColor': '#f5f5f5', 'lineColor': '#000000', 'weight': 10.2}
#polygon = folium.GeoJson(geojson.loads(r.content), style_function = lambda x: style).add_to(m)

style_function = lambda x: {'fillColor': '#ffffff',
                             'color': '#6666ff',
                             'fillOpacity': 0.1,
                             'weight': 6.2}

highlight_function = lambda x: {'fillColor': '#000000',
                                 'color': '#ee00ff',
                                 'fillOpacity': 0.50,
                                 'weight': 6.2}

polygon = folium.features.GeoJson(
    data = geojson.loads(r.content),
    style_function=style_function,
    control=False,
    highlight_function=highlight_function,
    tooltip=folium.features.GeoJsonTooltip(
        fields=['NAME'],
        aliases=['NAME'],
        style=("background-color: white; color: #333333; font-family: arial; font-size: 12px; padding: 10px;")
    )
).add_to(m)

folium.LayerControl().add_to(m)

folium_pane.object = m

folium_pane
```

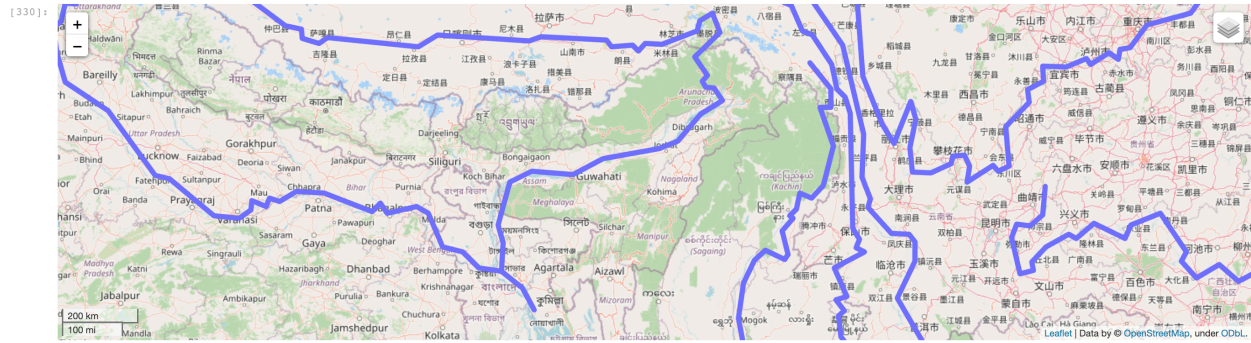


Figure 20 – GeoLabs MapServer OGC API - Fatures

The figure below illustrates how to interact with a WCS server instance through the GetCoverage request to access the raster dataset and display it from the tale using OpenCV.

Using TB18-MS-OGC-Tale from within the wholetale.org platform, the participants decided to publish the tale. To do so, a new tale was created from the wholetale.org website using the dedicated GitHub repository as a source and tests were conducted to ensure that everything worked as expected when running the tale. As the tests were conclusive, another copy of the tale was created using the same process and this copy was published using the wholetale.org tools and Zenodo (DataONE was also available but not considered during this research). The automated publication on the Zenodo platform led to the reception of a DOI for the tale. As quoted in the official wholetale.org documentation, in the Planned Features section, the container images are not published or exported in a way that would ensure the reproducibility of the processing environment. Indeed, only the files stored on wholetale.org for the tale are published, which includes only the Jupyter notebook and other files. As such, when a tale is published it contains multiple files used by repo2docker, which wholetale.org invokes when a user decides to build or run a tale.

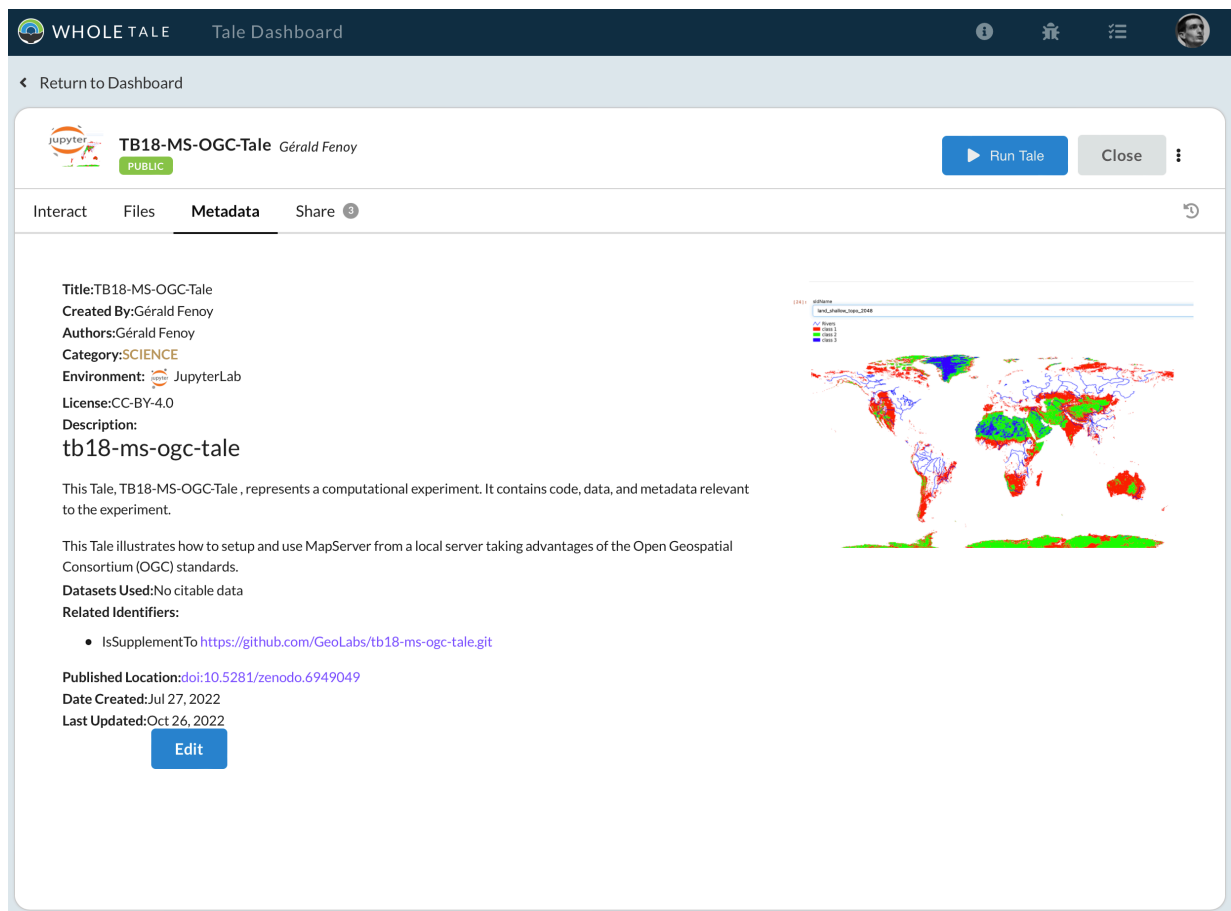


Figure 21 – GeoLabs MapServer OGC FAIRy tale preview

Consequently, when a user starts a tale, the container image may be built from scratch again. Therefore, there is no guarantee that this build will succeed. Indeed, it is possible to reproduce a run on the same Docker image once built, and it can be ensured that this image copy will be available for a long time. Unless the tale is run on the same Docker image, there is no way to ensure providing a reproducible environment on wholetale.org as the image may be rebuilt from time to time, and the software component may introduce issues.

```
View Logs

Forcing build.
[Repo2Docker] Looking for /wholetale/repo2docker_config in /
[Repo2Docker] Loaded config file: /wholetale/repo2docker_config.py
Picked Local content provider.
Using local repo /host/tmp/tmpck2e7jvj.
# syntax=docker/dockerfile:experimental
FROM buildpack-deps:bionic

# Avoid prompts from apt
ENV DEBIAN_FRONTEND=noninteractive

# Set up locales properly
RUN apt-get -qq update && \
apt-get -qq install --yes --no-install-recommends locales > /dev/null && \
apt-get -qq purge && \
apt-get -qq clean && \
rm -rf /var/lib/apt/lists/*

RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && \
locale-gen

ENV LC_ALL en_US.UTF-8
ENV LANG en_US.UTF-8

 Scroll to Bottom 
```

Figure 22 – GeoLabs Wholetale.org build run

GeoLabs evaluated the possibility of automatically building and publishing a binary Docker image and retrieve a DOI associated with it. Multiple ways are available, such as utilizing GitHub to publish the binary layers that constitute a container image and create a release with a Zenodo webhook activated for the repository’s releases. Nevertheless, if it is possible to acquire a DOI for the container binary image, there still needs to be support for using this DOI to create a new container. This publication process would provide a long-term solution for preserving the Binary Docker image and producing an associated DOI. The security considerations inevitably come with long-term preserved binary Docker images, as the software included in the container image may have vulnerabilities.

While creating this reproducible computational environment, GeoLabs realized that the build process requires a specific version of every software library/toolkit included within the build process because it is a prerequisite that the container image be built in a reproducible manner.

Reducing the complexity and the number of software components included in the container reduces the time required to build the container from wholetale.org and the resulting size of the binary container image.

Relying only on the processing facilities that wholetale.org offers for building and running container images sounds nonrealistic for embedding all the dependencies required by such a complex software.

6.4.3. Publishing a Tale to Interact With ADES Running on Microsoft Azure Kubernetes Cluster

GeoLabs investigated the creation of a tale that would illustrate how to install the [Application Deployment and Execution Service \(ADES\)](#) on a Microsoft Kubernetes cluster. From the [GitHub](#)

repository associated with this tale, it is noted that the apt.txt, requirements.txt, and postBuild files are short. Indeed, the number of software libraries/toolkits required are reduced to a minimum as ADES should deploy on a preconfigured Microsoft Kubernetes cluster. The ADES solution uses the ZOO-Project as a processing engine to offer the OGC API – Processes – Part 1: Core support. In addition to the Part 1 support, it also adds the capability to Deploy and Undeploy processes defined using the CWL format.

The tale contains multiple notebooks based on the original ADES wiki pages. Users can use them to connect their Azure Kubernetes cluster to deploy MiniIO and ADES. Then, they can Deploy an application and execute it remotely. When set up on an Azure Kubernetes cluster from this tale, the result produced by ADES will be a STAC catalog stored on the dedicated MiniIO S3 bucket.

The figure below shows a result of a process execution through ADES once fetched from the S3 bucket after extracting information from the STAC catalog.

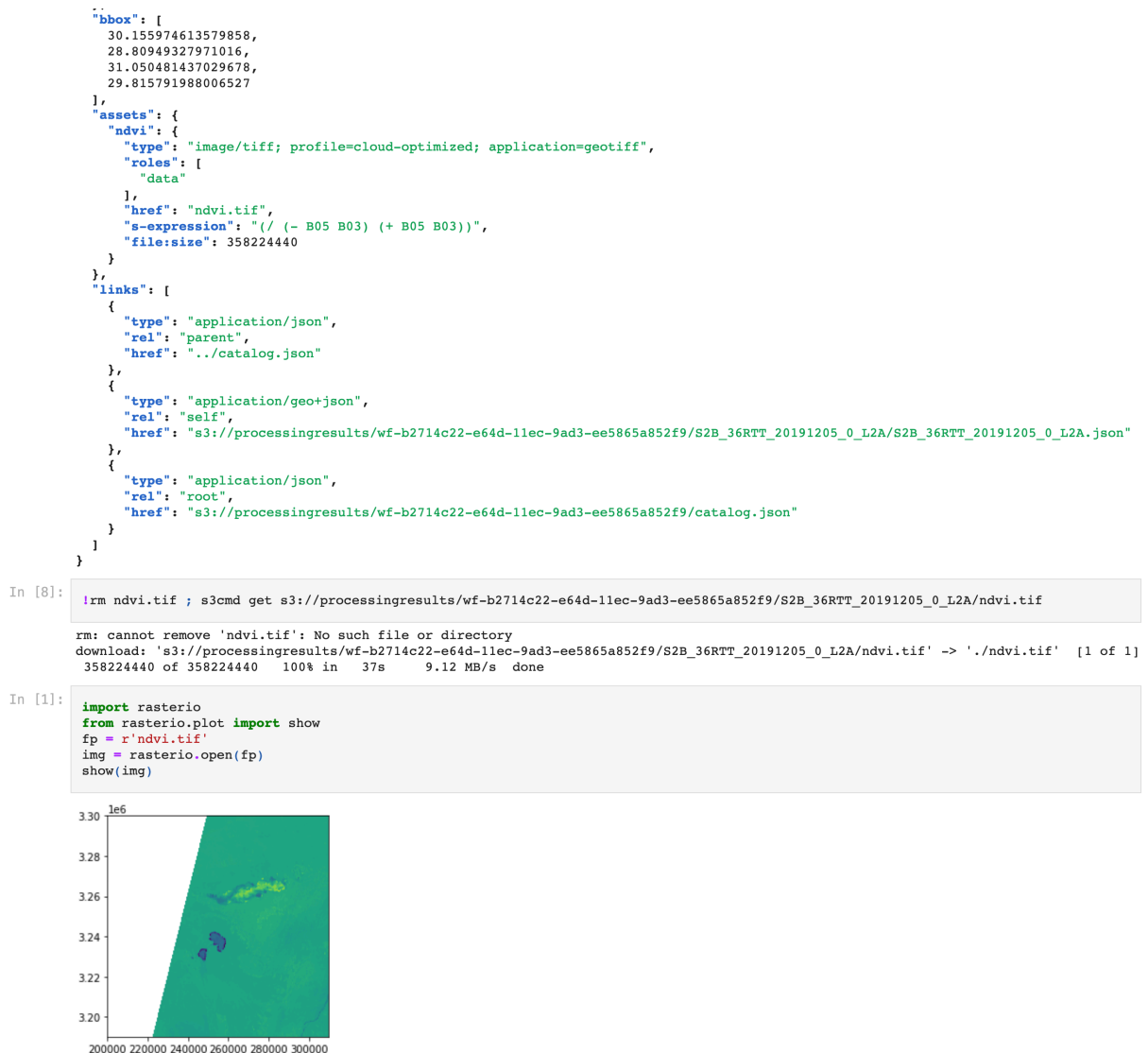


Figure 23 – GeoLabs Wholetale.org build run

When work began on this reproducible ADES deployment on an Azure Kubernetes cluster, ADES implementation provided DeployProcess and UndeployProcess as processes instead of properly defined requests in an OpenAPI as it is for other operations.

GeoLabs also reviewed the OGC API – Processes – Part 2: Deploy, Replace, Undeploy extension and provided the missing schemas and text content to the Draft Standard. Discussions were had with Ecere and Cubewerx on both Part 2 and Part 3 in dedicated meetings to avoid overlap with the ongoing Part 1 discussions. When signs of progress were observable, these were discussed with the SWG during bi-weekly meetings.

GeoLabs added a new conformance class to the Part 2 Draft Standard which enables the client application to detect that the OGC API – Processes – Part 2 Server Instance supports “application/cwl” payload to Deploy and Replace a process. IANA recognizes the “application/cwl” and “application/cwl+json” mime types. “application/cwl+yaml” is under discussion.

This led to the update of the implementation used by ADES in line with the ongoing Part 2 specification that was modified over time. The OpenAPI generated by the ZOO-Project environment from ADES provides a tentative implementation of the OGC API – Processes – Part 2: Deploy, Replace, Undeploy draft specification.

The modified Part 2 GeoLabs is working on introduces support for the POST method on the /processes path and the PUT and DELETE methods on the /processes/{processId} for Deploy, Replace, and Undeploy operations, respectively. The request payload for the POST and PUT HTTP methods is not restricted. An OGC Application Package conformance class was in the Draft Standard for interoperability purposes. Therefore, the request body may vary depending on the conformance class supported by the Server Instance and can have the “application/ogcappkg+json” or “application/cwl” content type. As the draft evolves, more content types should be supported including additional conformance classes, such as one for OpenEO. Once a working application to Deploy and Undeploy processes relying on “application/cwl” files, GeoLabs added a conformance class and is open to working on missing conformance classes. The ADES implementation was modified to support this new Standard with the CWL conformance class support. The payload can be of both content types and works similarly behind the scenes of ADES.

Having reached this level in both the content of the Draft Standard and having a tentative implementation led to the security considerations discussion. Having the capability of deploying and executing users’ services in a restricted manner is required. This is to reduce or restrict, for instance, the number of nodes and namespaces created during the process execution per user. Also, offering access to ADES on a user’s own Azure Kubernetes cluster may require a significant amount of funding to cover the cost of user executions. If there is no security in place and no restriction at the core of the processing engine, there is no control over the price, either.

As such, GeoLabs first introduced a security section in the Draft Standard mentioning that the OpenAPI can be protected to restrict or manage user access to specified requests and paths. Consequently, the OpenAPI security schema was added in the official ZOO-Project GitHub repository main branch. Therefore, when a user deploys a ZOO-Project Server Instance, they can receive the published OpenAPI containing the security schema definitions and restricted access per requested methods and paths.

The implementation is currently limited to Basic, Bearer, and API keys. If the OpenAPI presents some request methods and paths as secured, they need to remain as secured.

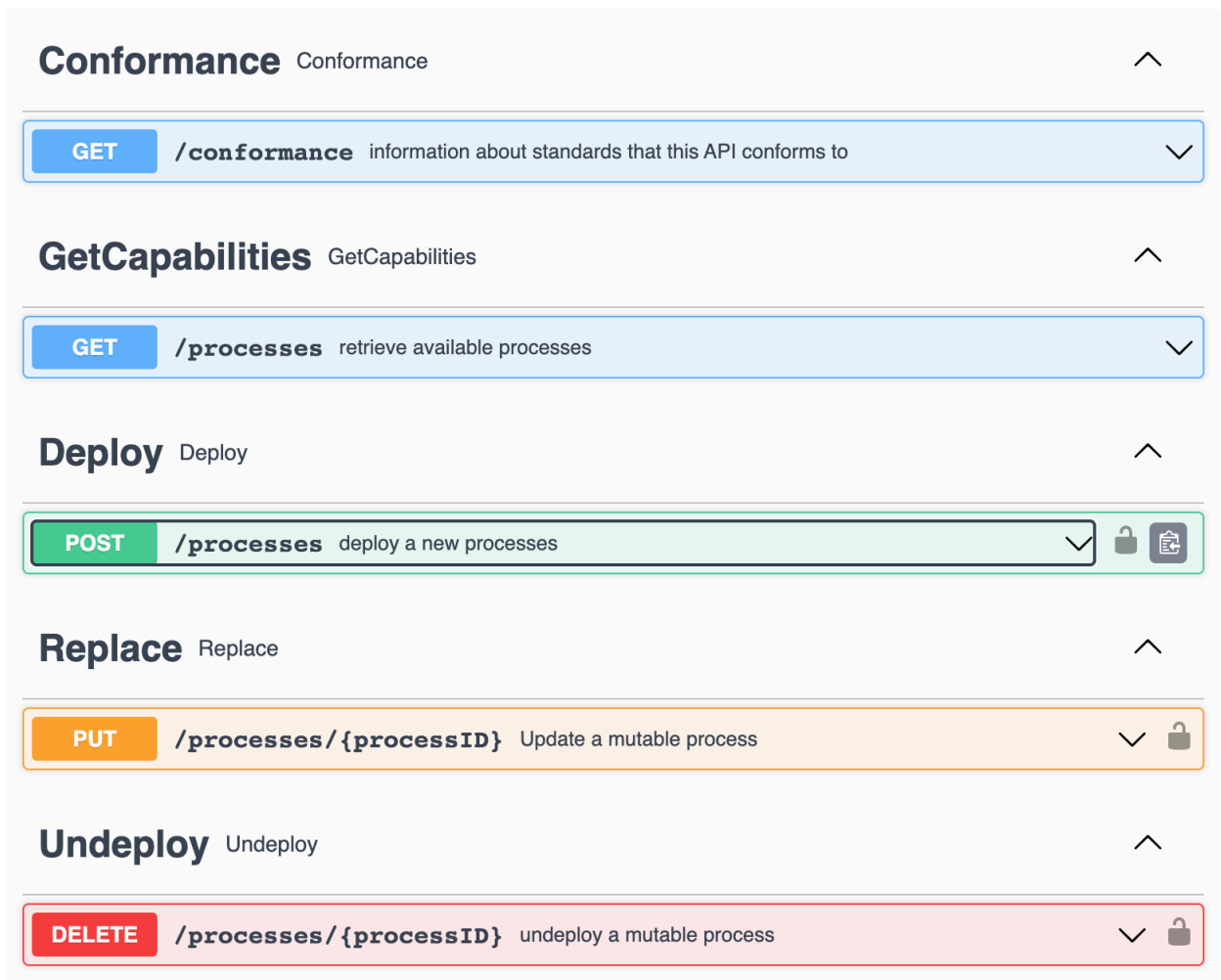


Figure 24 – GeoLabs Wholetale.org build run

The participants decided to implement security as a process to ease the development and integration of new security schemes in the ZOO-Project platform. To provide an example of such a security process, an example was implemented in C that provides HTTP Basic Authentication. This example uses an Apache password file and verifies that the furnished user and password correspond to the one stored in the file. If the test fails and the couple (request method, path) is secured, it results in an error message with a 401 or 403 response header, depending on the authentication error. In other cases, the request continues and returns the expected result. It is a trivial implementation, but it illustrates well how easy it is to integrate a new security scheme within the ZOO-Project environment.

Deploy Deploy

POST /processes deploy a new processes

Deploy a new processes

Parameters Try it out

No parameters

Request body *required*

Mandatory OGC Application Package in CWL format

Examples:
 Example 0: Deploy snuggs process

Example Value | Schema

```

{
  "executionUnit": {
    "href": "https://raw.githubusercontent.com/EDEPCA/app-snuggs/main/app-package.cwl",
    "type": "application/cwl"
  }
}

```

Responses

Code	Description	Links						
201	the process is deployed	No links						
	Media type <input type="text" value="application/json"/> Controls Accept header. Example Value Schema <pre>"string"</pre> Headers: <table border="1"> <thead> <tr> <th>Name</th> <th>Description</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>Location</td> <td>URL to fetch the processDescription of the deployed process</td> <td>string</td> </tr> </tbody> </table>	Name	Description	Type	Location	URL to fetch the processDescription of the deployed process	string	
Name	Description	Type						
Location	URL to fetch the processDescription of the deployed process	string						
500	A server error occurred.	No links						
	Media type <input type="text" value="application/problem+json"/> Example Value Schema <pre> { "type": "string", "title": "string", "status": 0, "detail": "string", "instance": "string", "additionalProp1": {} } </pre>							

Figure 25 – GeoLabs Wholetale.org build run

With this initial security capability in place, GeoLabs would have liked to provide a tool to identify the processes that require secured access. Indeed, there are multiple use cases where there is a process dedicated to users' authentication that returns a cookie, then other services would require authentication for them to work correctly. In the current implementation, GeoLabs decided to use the following rules. The first path (/processes/{processId}/execution) defines the general case, while the second (/processes/GivenProcessId/execution) contains exceptional ones. Meaning that for the path /processes/{processId}/execution, the security applies to every process, while for /processes/GivenProcessId/execution, the protection applies only to the GivenProcessId process.

6.4.4. GeoLab Considerations and Limitations for Reproducible Workflows

- How to offer long term registry offering access to the binary Docker image used to run the tale and assign a DOI to that binary Docker image.
 - Wholetale.org does not offer the ability to assign DOIs to the binary docker images.
 - One solution would be to use a GitHub Action for publishing binary layers of the image in a dedicated repository. Then, create a release for this repository leading to the assignment of a DOI, using the Zenodo GitHub setting for this repository. From there, a user can use the archive provided with the tale's DOI to run the tale on the user's own infrastructure.

6.4.5. Technical Interoperability Experiments

Terradue has shared a tale with GeoLabs.

6.5. Terradue

6.5.1. Goals of Participation

The goals and benefits from this effort include the following.

- The Best Practice for Earth Observation Application Package opens the possibility to enlarge the reproducibility of the EO workflows outside the EO Exploitation Platform boundaries.
- The Application Package has the potential to provide EO applications compatible with the Open Science principles (e.g. FAIR) and capable of guaranteeing their long-term reproducibility.
- End-users may reuse the best practices identified in this thread.
 - Platform owners know the scientific impact of their services.
 - EO application developers know how their applications are linked to the research impact.

6.5.2. Contributed Workflow and Architecture – Water Bodies Detection Application Package

Terradue's Testbed 18 activities use an application that detects water bodies based on the Normalised Difference Water Index (NDWI), an index derived from the Near-Infrared (NIR) and Green (G) channels of Sentinel-2 acquisitions. This formula highlights the amount of water in water bodies and thus these can be detected, with some confidence, using a thresholding approach on the NDWI. This application is the starting element for meeting the Testbed goals on how to ensure reproducible Earth Observation science in terms of application package, application container, application input data, and application output.

The Water Bodies Detection application takes as an input a list of Sentinel-2 acquisitions encoded as STAC Items referencing the assets as Cloud Optimized GeoTIFFs.

For each Sentinel-2 acquisition provided as an URL to a STAC Item, the application package orchestrates command line tools that perform the following:

- clip the green and NIR assets;
- calculate the normalized difference between green and NIR clipped assets deriving the NDWI;
- apply the Otsu automatic threshold to delineate the water bodies; and
- generate a STAC Item with the detected water body.

The application is packaged according to the OGC Best Practices for EO applications packages and thus implements a CWL Workflow element that scatters, in a fan-out pattern, the list of Sentinel-2 acquisitions defined as URLs to STAC Items on the Cloud which invokes a sub-Workflow element that scatters the green and NIR clipping, fans-in into the normalized difference, followed by the Otsu Automatic Threshold and finally creates a STAC Item with the detected water body.

The application contains four command line tools implemented in Python. The first command line tool uses GDAL to clip the remote green and NIR assets that are encoded as COGs. The second command line tool uses numpy and GDAL to apply the normalized difference between two bands. The third command line tool applies the sklearn implementation of the Otsu Automatic Threshold operator. The fourth command line tool implements the STAC Item generation using the pystac library.

Each command line tool is included in a container image and referenced in the CWL document with the CommandLineTool class describing the Application Package.

This Application Package, encoded as CWL, is used as an artifact for the exploratory activities Terradue conducted during this Testbed.

6.5.3. Application Package Software Configuration Management

The SCM has the task of tracking and controlling changes in the software as a part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.

The Water Bodies Detection Application Package SCM relies on git and has a dedicated software repository hosted on GitLab, available at the URL <https://gitlab.com/terrადue-ogc/testbed-18/water-bodies-detection>.

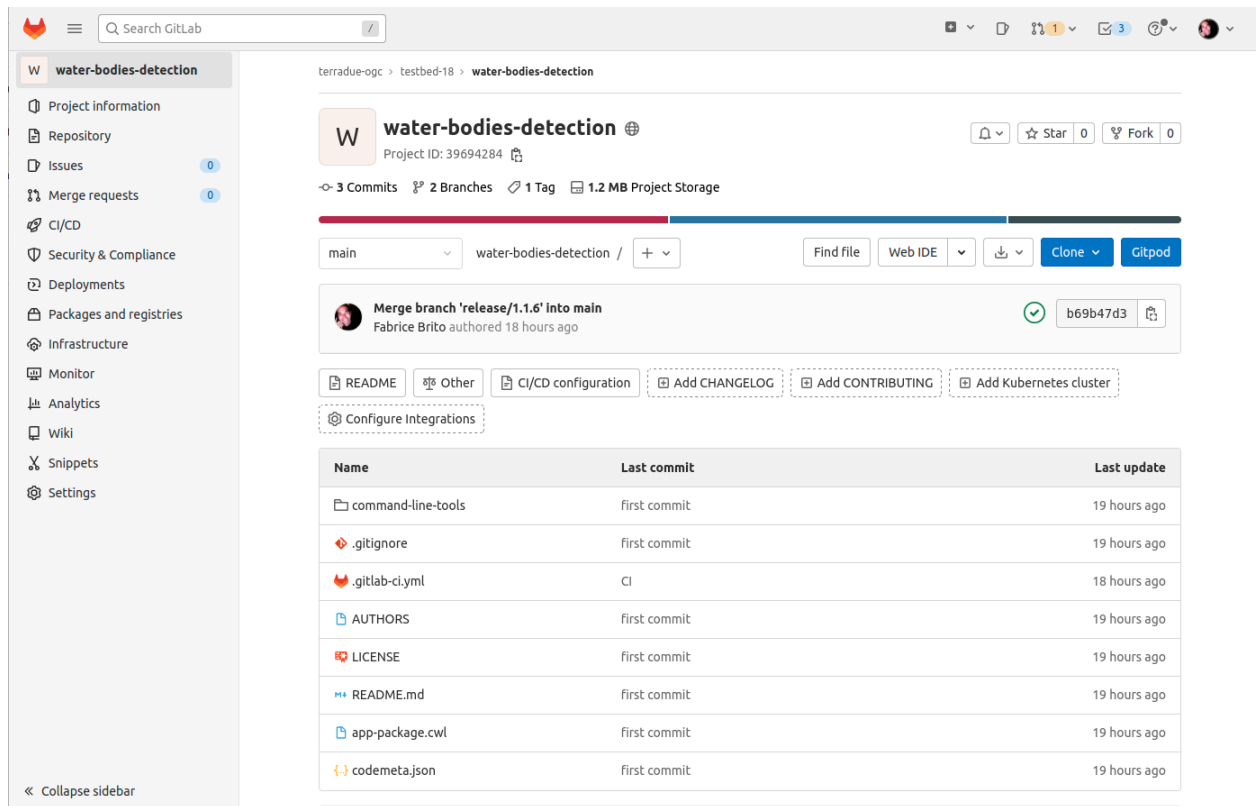


Figure 26 – Application Package software configuration management

Terrადue looked at the Software Heritage approach to assign an identifier to the Water Bodies Detection SCM and followed its best practices and recommendations.

To reference or cite the objects present in the Software Heritage archive, permalinks based on SoftWare Heritage persistent IDentifiers (SWHIDs) must be used instead of copying and pasting the url from the address bar of the browser as there is no guarantee the current URI scheme will remain the same over time.

First, the code must be:

- hosted on a repository publicly accessible (Github, Bitbucket, a GitLab instance, an institutional software forge, etc.) using one of the version control systems supported by (Subversion, Mercurial and Git); and

- include, at the top level of the source code tree, the following files:
 - **README** containing a description of the software (name, purpose, pointers to website, documentation, development platform, contact, and support information, ...)
 - **AUTHORS**, a list of all the persons to be credited for the software.
 - **LICENSE**, the project license terms. For Open Source Licenses, the standard SPDX license names are used. For large software projects and developers, the REUSE (<https://reuse.software/>) process and tools can be an option to look at.
 - **codemeta.json**, a linked data metadata file that helps index the source code in the Software Heritage archive and provides an easy way to link to other related research outputs.

While the first three files to be included in the repository are self-explanatory, the `codemeta.json` of the code meta project deserves further explanation.

The code meta project motivation (<https://codemeta.github.io/>) is reported below:

Research relies heavily on scientific software, and a large and growing fraction of researchers are engaged in developing software as part of their own research (Hannay et al 2009). Despite this, infrastructure to support the preservation, discovery, reuse, and attribution of software lags substantially behind that of other research products such as journal articles and research data. This lag is driven not so much by a lack of technology as it is by a lack of unity: existing mechanisms to archive, document, index, share, discover, and cite software contributions are heterogeneous among both disciplines and archives and rarely meet best practices (Howison 2015). Fortunately, a rapidly growing movement to improve preservation, discovery, reuse, and attribution of academic software is now underway: a recent NIH report; conferences and working groups of FORCE11, WSSSPE & Software Sustainability Institute; and the rising adoption of repositories like GitHub, Zenodo, figshare & DataONE by academic software developers. Now is the time to improve how these resources can communicate to each other.

With the goal to address the fact that there are many different metadata vocabularies, the CodeMeta project started an effort to tackle this situation which led to the development of a sort of Rosetta stone for translating from one vocabulary to the other: the CodeMeta crosswalk table and the creation of the CodeMeta vocabulary, as an extension of the SoftwareApplication and SoftwareSourceCode classes found in the vocabulary of the Schema.org initiative. Metadata information conformant to the CodeMeta vocabulary can be represented in JSON format, typically named `codemeta.json`.

The initial `codemeta` JSON file for the Water bodies detection application package was generated manually using the `codemeta` generator built by the Software Heritage and then added to the software repository. The generator is available at the URL <https://codemeta.github.io/codemeta-generator/>.

The software repository was submitted for archival at the Software Heritage and it has the following identifier:

swh:1:dir:afaf7ddf712f8f3f0c66bd16de495d31a67106b9;origin=https://gitlab.com/terradue-ogc/testbed-18/water-bodies-detection.git;visit=swh:1:snp:79dd9bcb0327d0cd75abceccb7bccbe51e889a3;anchor=swh:1:rev:b69b47d3dc663031cee7793813e01a2db6f582e7

and the permanent link:

<https://archive.softwareheritage.org/swh:1:dir:afaf7ddf712f8f3f0c66bd16de495d31a67106b9;origin=https://gitlab.com/terradue-ogc/testbed-18/water-bodies-detection.git;visit=swh:1:snp:79dd9bcb0327d0cd75abceccb7bccbe51e889a3;anchor=swh:1:rev:b69b47d3dc663031cee7793813e01a2db6f582e7>

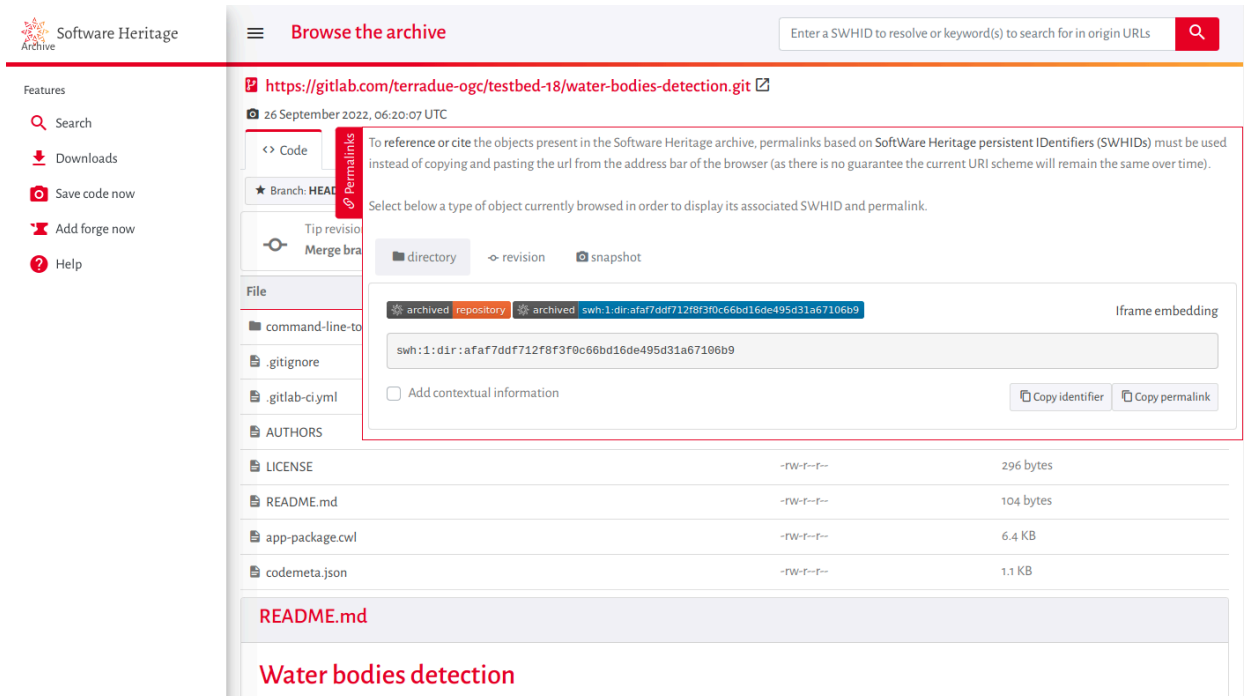


Figure 27 – Application Package on Software Heritage

6.5.4. Application Package Containers

The Water Bodies Detection application package follows the OGC Best Practices for EO Application Packaging and in particular the command line tools are containerized and published on container registries.

This section addresses the steps needed to achieve a higher level of the determinism for container images references.

Container images are tagged when built, for example:

```
docker build -t docker.terradue.com/crop:1.1.6 .
```

The application package reflects this container image with:

```
- class: CommandLineTool
  id: crop
```

```
hints:
  DockerRequirement:
    dockerPull: docker.terradue.com/crop:1.1.6
```

With this tag, and when pulling this container image from the container registry, the user expects to have pulled the right container. What happens if, by mistake, the developer or continuous integration tool builds another container image with the same tag? The application package would be, at this stage, broken.

A deterministic approach benefits from the sha256 signature which acts as an immutable identifier of the container image, and this can be obtained with:

```
docker inspect docker.terradue.com/crop:1.1.6 | jq '.[0]["RepoDigests"][0]'
```

This returns:

```
docker.terradue.com/crop@sha256:ec2d8e71ab5834cb9db01c5001bde9c3d6038d0418ad085726b051b4359750e1
```

The application developer updates his/her application package with:

```
- class: CommandLineTool
  id: crop

  hints:
    DockerRequirement:
      dockerPull: docker.terradue.com/crop@sha256:ec2d8e71ab5834cb9db01c5001bde9c3d6038d0418ad085726b051b4359750e1
```

While this approach is a way to ensure a deterministic identification of a container image, Docker currently has an issue with its own builds where two separate builds from the same Dockerfile result in differing hash values. Nevertheless this has a limited impact on the application package.

Terradue activities also attempted to identify a mechanism for assigning a DOI to a container image but these lead to a manual process that cannot be reused for the reproducibility of an application package execution. Instead, the container's sha256 is relied upon to reference or cite it.

6.5.5. Continuous Integration

The Water Bodies Detection application package Continuous Integration includes the following steps.

- Build and push the container images to a container registry.
- Release the application package and publish the artifact (CWL document) with the updated container image references.
- Publish the release to the Software Heritage.

The build and push containers step uses a container tag but then inspects the container to retrieve its sha256. These values are used in the application package release step and

the references to the containers are deterministic satisfying the determinism required for reproducibility.

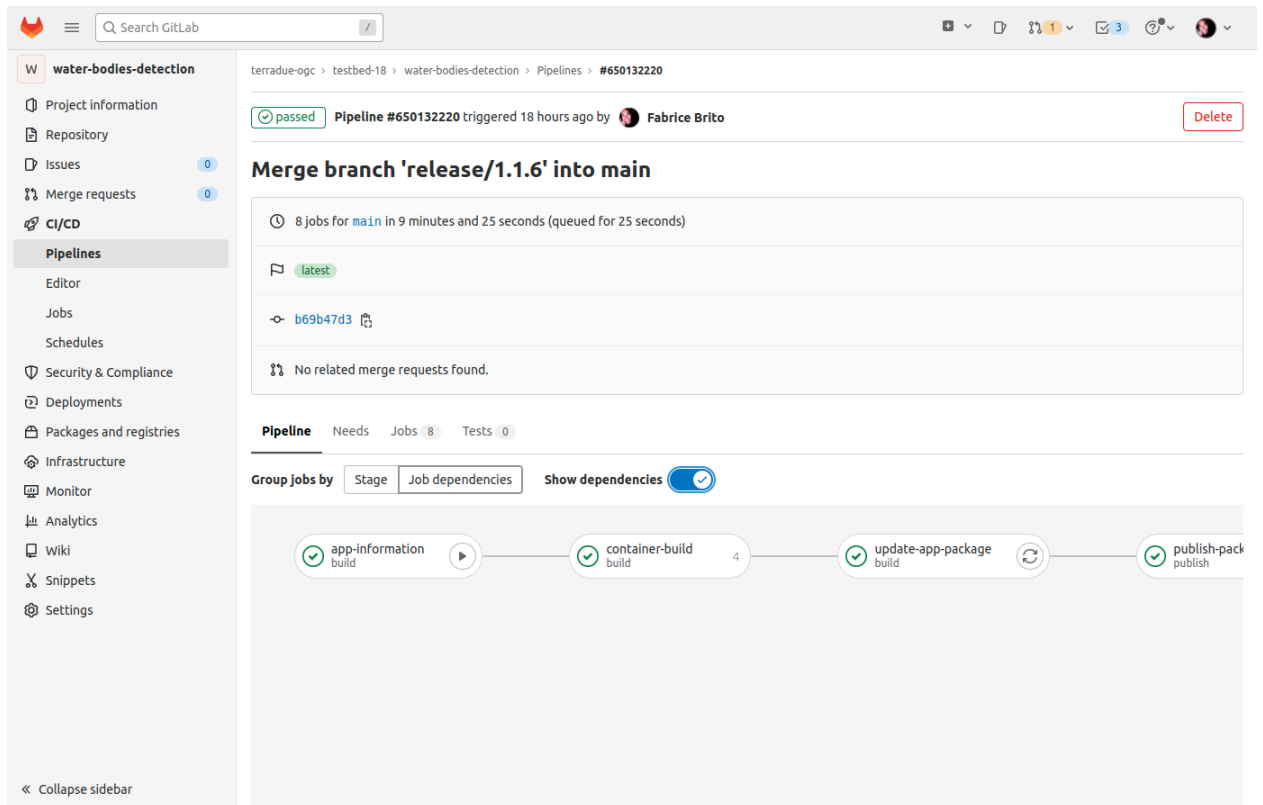


Figure 28 – Application Package Continuous Integration on Gitlab.com

The last step of the Continuous Integration provides a Software Heritage configuration that saves the repository to the Software Heritage whenever a release is triggered. This step ensures that each release gets a Software Heritage identifier.

The Water Bodies Detection application package execution relies on a CWL runner and for the purposes of assigning identifiers to the application package and the inputs/outputs the official CWL runner was used.

6.5.5.1. Application Package

The OGC Best Practices for EO Application Packages recommends enriching the application package with concepts from schema.org that are used whenever possible and linked with their RDF encoding.

While enforcing the version as a mandatory, the Best Practices fosters the provision of additional information to encourage a correct citation.

The elements listed are:

- *author*: The main author of the Application Package. <https://schema.org/author>

- *citation*: A citation or reference to a publication, web page, scholarly article, etc. <https://schema.org/citation>
- *codeRepository*: Link to the repository where the Application code is located (e.g. SVN, github). <https://schema.org/codeRepository>
- *contributor*: A secondary contributor to the Application Package. <https://schema.org/contributor>
- *dateCreated*: The date on which the Application Package was created. <https://schema.org/dateCreated>
- *keywords*: Keywords used to describe this application. Multiple entries in a keywords list are delimited by commas. <https://schema.org/keywords>
- *license*: An URL to the license document that applies to this application. <https://schema.org/license>
- *releaseNotes*: Description of what changed in this version. <https://schema.org/releaseNotes>

During the Testbed activities, Terradue looked at how to assign a Digital Object Identifier (DOI) to an application and identified a solution following the same approach with the *sameAs* element defined as the URL of a reference webpage that unambiguously indicates the item's identity – <https://schema.org/sameAs>.

For the Water Bodies Detection application package, the metadata section becomes:

```
$namespaces:
  s: https://schema.org/
s:author:
- class: s:Person
  s:affiliation: Planet Earth
  s:email: john.doe@somedomain.org
  s:name: Doe, John
s:contributor:
- class: s:Person
  s:affiliation: Planet Earth
  s:email: jane.doe@somedomain.org
  s:name: Doe, Jane
s:sameas: https://doi.org/10.5072/zenodo.1107209
s:softwareVersion: 1.1.6
schemas:
- http://schema.org/version/9.0/schemaorg-current-http.rdf
```

At this stage, the application package is findable and citable.

The DOI is assigned by the Zenodo platform.

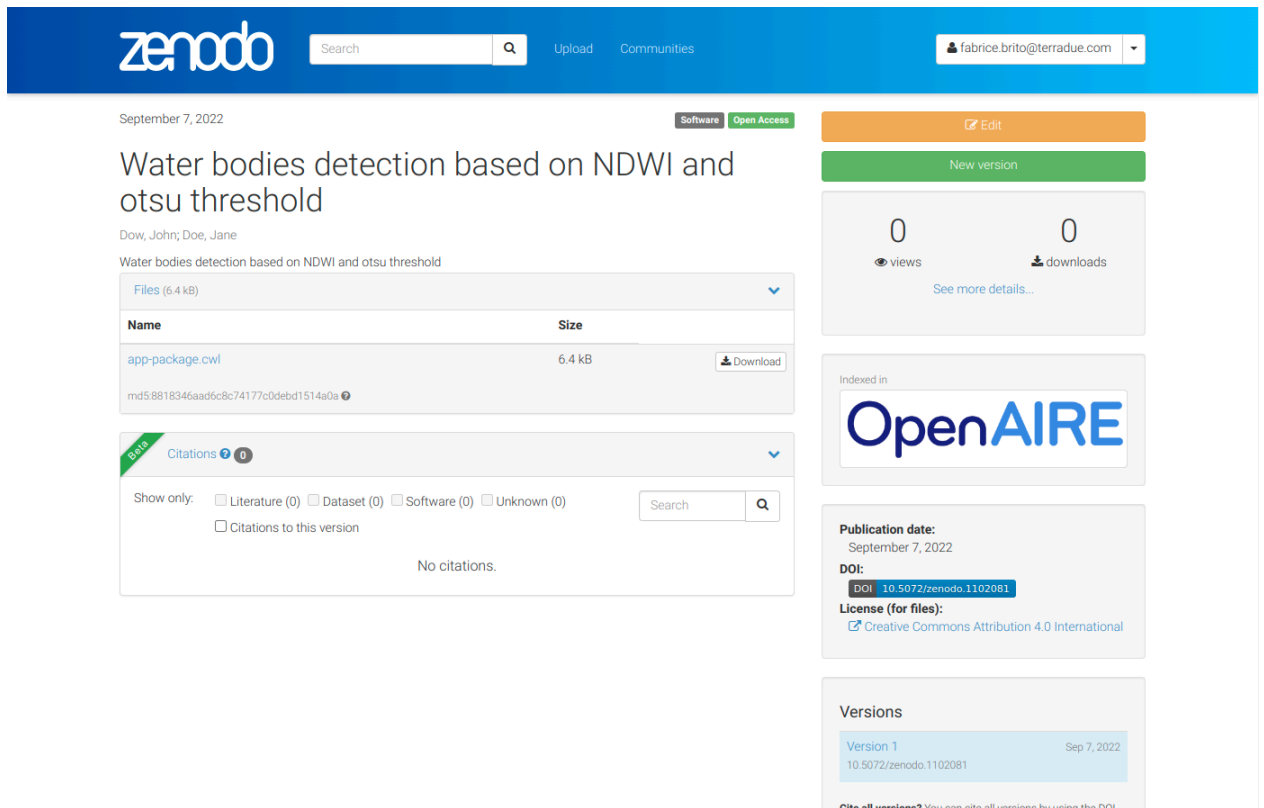


Figure 29 – Application Package on Zenodo

6.5.6. Input Dataset

The Water Bodies Detection application package takes as an input an array of STAC Items URLs describing Sentinel-2 acquisitions.

An execution of this application package will then take a finite list of those URLs. The list of URLs was aggregated into a STAC Collection. The STAC Collection Specification defines a set of common fields to describe a group of Items that share properties and metadata. This definition fits the purposes of aggregating the STAC Items into a single element that can be cited and assigned a digital identifier.

The STAC Collection specification may be extended using STAC Extensions. During Testbed 18, a subset of the STAC Extensions (listed below) were identified as more relevant to the Testbed goals:

- The *Scientific Citation Extension* adds the ability to indicate from which publication data originates and how the data itself should be cited or referenced.
- The *Datacube Extension* specifies the datacube related metadata, especially their dimensions, and potentially more in the future.

The referencing capability of the *Scientific Citation Extension* is a key element to meet the goals of reproducibility and citation of an application package execution. If a DOI is assigned to the

STAC collection that aggregates the STAC Items used as inputs, the input dataset becomes citable and findable.

The *Datacube Extension* allowed us to enrich the input dataset STAC Collection with additional information on the temporal, spatial, and radiometric dimensions. The temporal enrichment provides the list of acquisition dates, the spatial provides the extent of x and y dimensions, and, finally, the radiometry provides clear and detailed information about the radiometric characteristics of the Sentinel-2 data.

During the Testbed activities, Terradue implemented a Jupyter notebook that takes as an input the list of STAC Items of the input dataset, produces a STAC Collection enriched with a DOI (provided by the Zenodo platform) and the datacube STAC extension fields.

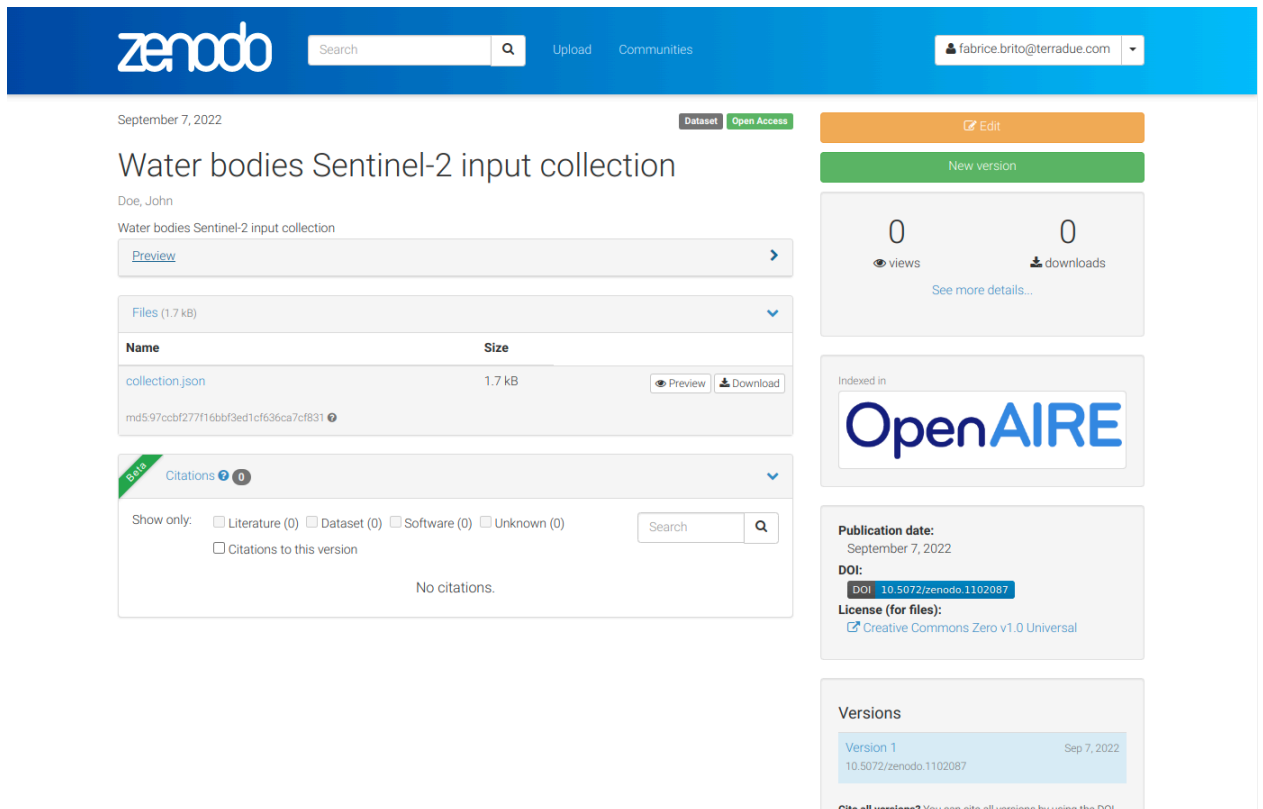


Figure 30 – Input collection on Zenodo

6.5.7. Output Dataset

The output dataset as generated by the application package execution is a STAC Catalog with links to STAC Items representing and describing the water bodies detected in a provided Sentinel-2 acquisition.

Terradue implemented a similar approach as the one following for the input dataset.

- Describe the output dataset as a STAC Collection.
- Add the *Scientific Citation Extension* to the output dataset STAC Collection.

- Add the *Datacube Extension* to the output dataset STAC Collection.
- Add the *File Extension* to add the results' md5.

Terradue created a notebook implementing these steps and, as done for the input dataset, added the code to interact with Zenodo's API to assign a DOI to the output dataset STAC collection.

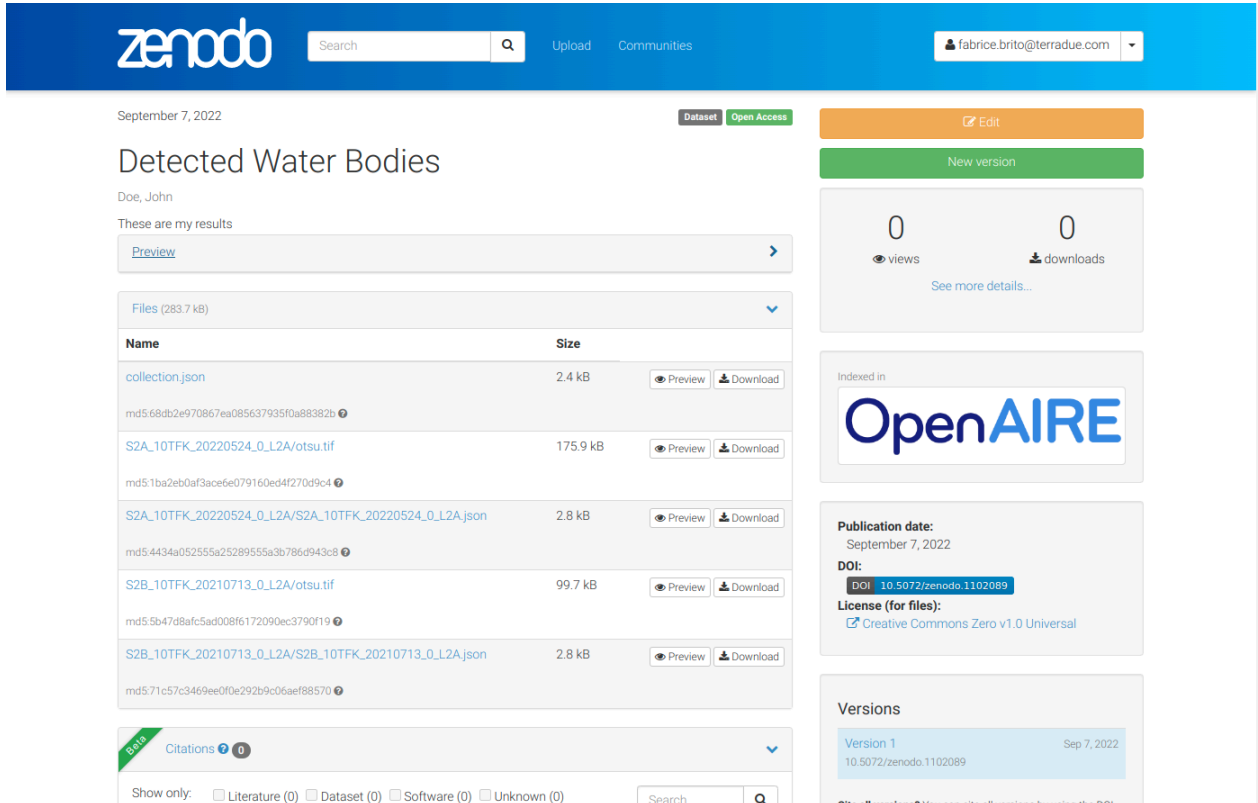


Figure 31 – Output collection on Zenodo

6.5.8. Retrospective Provenance of an Execution

During the Testbed activities, Terradue looked at how to address the retrospective provenance of an Application Package execution. The Common Workflow Language community world on CWLProv, a format for the methodical representation of workflow enactment, associated artifacts, and capturing and using retrospective provenance information.

The official CWL runner, cwltool, implements the CWLProv approach and it was quite straightforward to have the application package execution produce the retrospective provenance folders and files in the Bagit format.

The content of the Bagit CWLProv folder includes all the information to run the application package against the set of input parameters, intermediary files, and output results. All these files have their md5sum listed in metadata files allowing verifying the results between repeated runs.

6.5.9. Reproducible Earth Observation science

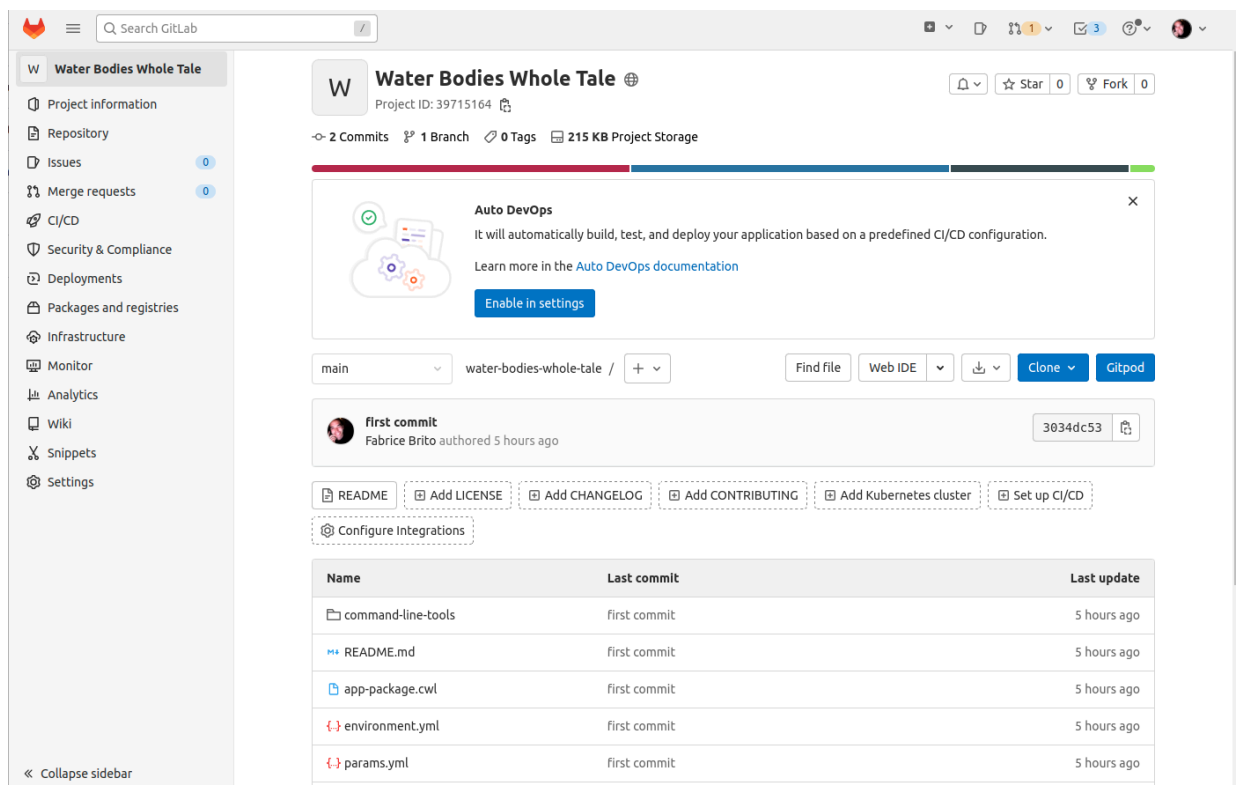
Terradue's Testbed 18 activities demonstrated that scientific Earth Observation workflows can be reproducible with existing standards and tools. Starting with the package of the workflow in an application package, the assignment of unique identifiers during its development, deployment, and instantiation to ensure the workflow reproducibility was demonstrated.

The activity supported the definition of best practices for reproducibility using existent specifications and tools. CodeMeta is used to describe the application elements, EO Application Package (OGC XXX) to package the workflow components, Docker to create the necessary containers, and CWLProv to define the full application instantiation provenance. With this exercise the essential ingredients of a reproducible scientific Earth observation workflow were identified.

6.5.10. Water Bodies Detection on Whole Tale

As part of Testbed 18 activities, Terradue also explored how Whole Tale could be used for the reproducible Earth observation workflows. This section provides Terradue's experience in using the Whole Tale platform to run the Water Bodies Detection application package.

Terradue created a dedicated Github repository as the seed for the tale. The seed repository is available at the URL: <https://gitlab.com/terrardue-ogc/testbed-18/water-bodies-whole-tale>.



The screenshot shows the GitLab interface for a repository named "Water Bodies Whole Tale". The repository is located at the path "water-bodies-whole-tale" on the "main" branch. It has 2 commits, 1 branch, 0 tags, and 215 KB of project storage. The repository was created by Fabrice Brito 5 hours ago. The repository contains several files, including "command-line-tools", "README.md", "app-package.cwl", "environment.yml", and "params.yml". The repository is currently empty of code, showing only the file list and commit history.

Name	Last commit	Last update
command-line-tools	first commit	5 hours ago
README.md	first commit	5 hours ago
app-package.cwl	first commit	5 hours ago
environment.yml	first commit	5 hours ago
params.yml	first commit	5 hours ago

Figure 32 – Software repository for seeding the Whole Tale

The Tale is available at: <https://dashboard.wholetale.org/run/62bb160e4c73b407324772e2>

The repository contains the following.

- The python scripts and a CWL document containing the Water Bodies Detection Application Package.
- A CONDA environment file including the cwlttool CONDA package.
- A postBuild file that repo2docker uses to configure the additional CONDA environments simulating the environment isolation the containers provide.

While the Tale uses JupyterLab as the runtime environment, the execution of the Application Package is done on a Terminal with a CWL runner. The Terminal is provided by the JupyterLab instance triggered by Whole Tale.

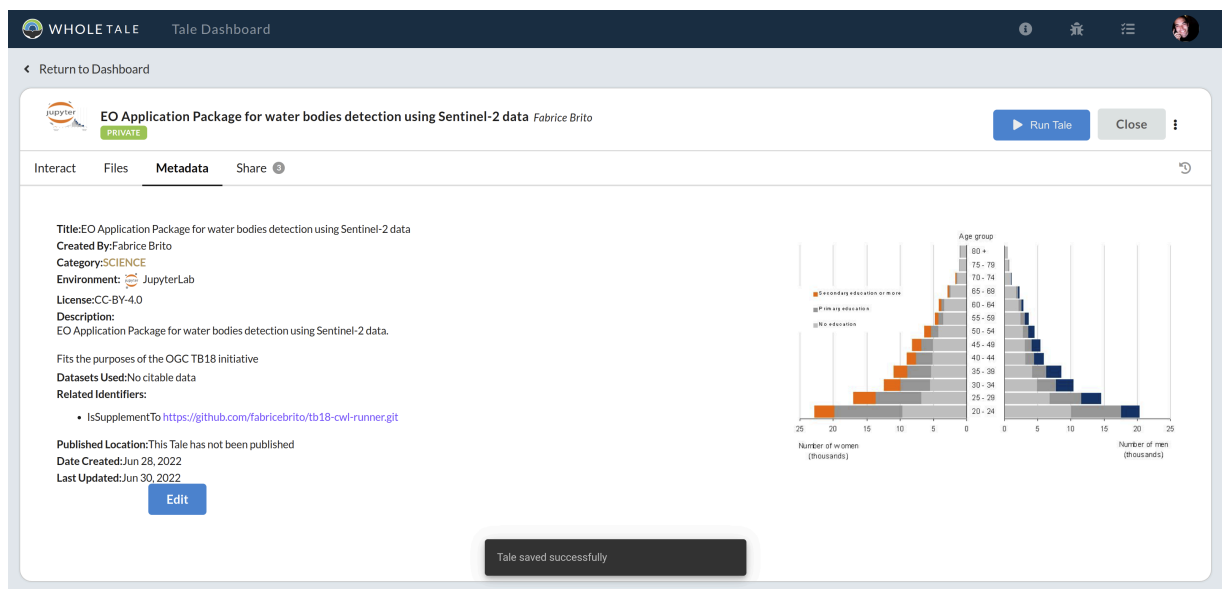


Figure 33 – Application Package on Whole Tale

6.5.10.1. Running the Tale

The Whole Tale dashboard (<https://dashboard.wholetale.org/>) shows the tales that belong to the logged user or tales that they have access to. The Water Bodies Detection Tale is run using a JupyterLab environment and its included shell terminal using the CWL runner.

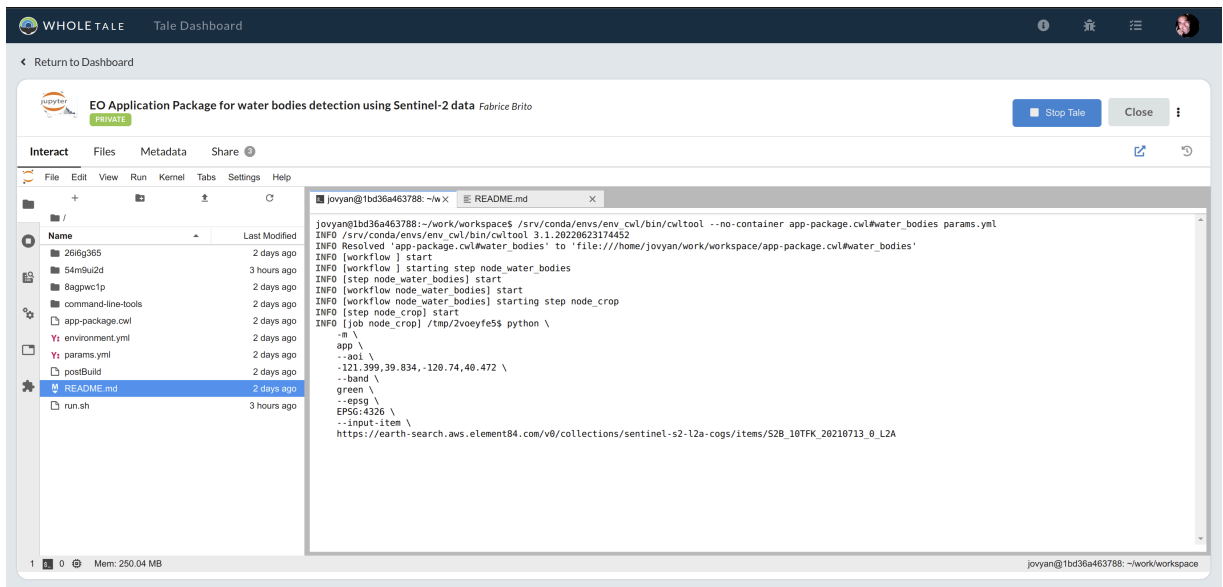


Figure 34 – Running the Application Package on Whole Tale

The execution generates a folder with the results. The Whole Tale workspace can be saved at different moments in time.

6.5.10.2. Creating Tale Runs

The Tale was enriched with a run.sh script that invokes the cwltool CLI and generates a new folder with the results. This is an automated execution of the Tale. As with the previous runs, these are saved and accessible in the Tale workspace.

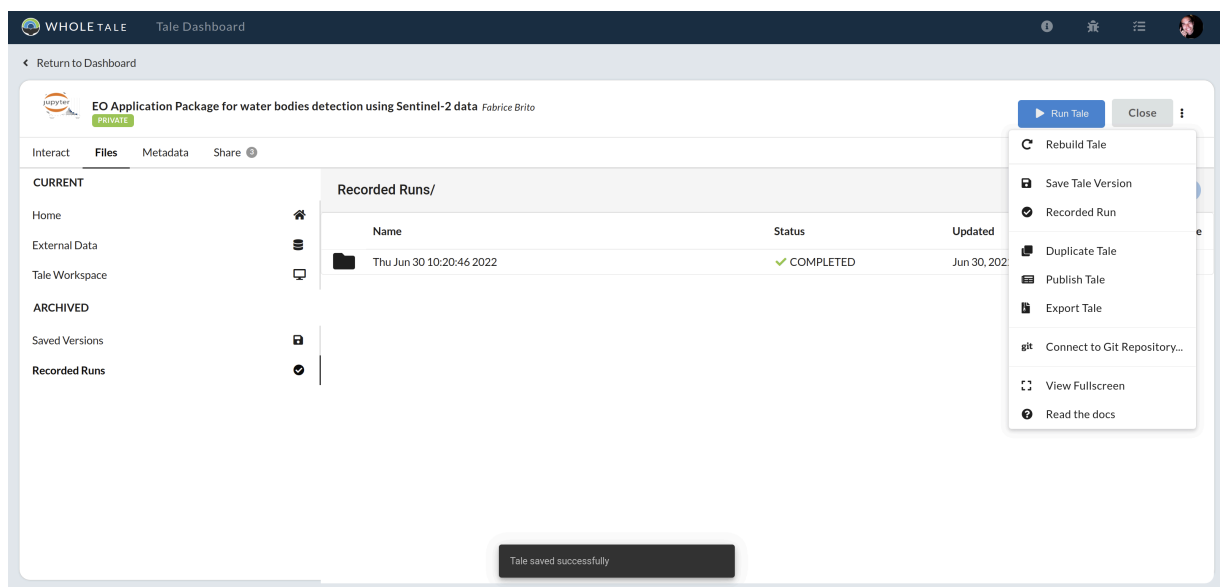


Figure 35 – Recorded runs of the Application Package on Whole Tale

6.5.10.3. Referencing Data in a Tale

The Sentinel-2 STAC items were added as two HTTP data references. Terradue's opinion is that this didn't provide additional value for the Tale execution or reproducibility.

6.5.10.4. Tale Export and Local Execution

The Tale export function uses the bagit format to create an archive that, once extracted, will use Docker to configure and set a JupyterLab instance on a local PC. Nevertheless, using the content right out of the box caused the local execution to fail because the postBuild script wasn't invoked during the container image creation and thus the CONDA environments providing the executables for the Application Package CLI were missing.

This issue hampers the assigning of a DOI with platforms such as Zenodo as the repeatability and reproducibility is not ensured.

6.5.11. Link to the Original Software Repository

While the tale is created starting from an initial git repository, the original link to that git repo is lost.

6.5.12. Whole Tale Assessment

Terradue identified additional functionalities needed for developing, running, and sharing Application Packages on Whole Tale which include the following.

- Running containers as the Application Package advocates the command line tools are encapsulated in containers with all the software dependencies.
- Guarantee the link with the original git repository to version evolutions of the Tale back to git.
- A better export experience to reproduce the environment with all its elements including the post build step.

6.5.13. Considerations and Limitations for Application Packages Reproducibility

EO Application Packages are created using the following process.

- Application developers create containers with their runtime environment, dependencies, and application binaries.

- Application developers/integrators describe the Application package using the Common Workflow Language.
- Applications needing more complex Data Flow Management can use a local catalogue encoded using STAC as a data manifest for application inputs and outputs metadata.
- Applications can be deployed in a Cloud provider and invocable in a service compliant with the OGC API Process specification.

EO Application Packages use the Common Workflow Language (CWL) for orchestrating command line tools in containers. The CWL is an open standard:

- for describing analysis workflows and tools;
- portable and scalable across a variety of software and hardware environments, from workstations to cluster, cloud, and high performance computing (HPC) environments; and
- designed to meet the needs of data-intensive science, such as Bioinformatics, Medical Imaging, Astronomy, Physics, Geology, and Chemistry.

The CWL standards support workflow automation, scalability, abstraction, provenance, portability, and reusability.

Complementary to the CWL efforts, Terradue identified and addressed the considerations and limitations to increase the Application Package alignment with the FAIR principles.

6.5.14. Technical Interoperability Experiments

The Whole Tale workflow developed by Terradue was shared amongst the Testbed 18 participants.

6.5.15. Lessons Learned

The adoption of software configuration management is a stepping stone for properly managing the EO applications source code. Yet, it has been made apparent that it is not enough for this code to be cited and preserved. The codemeta approach provides a solid metadata schema and acts as a manifest when interacting with the Software Heritage, the initiative ensuring code preservation, and curation and assigning an identifier. Some journals ask for the codemeta json file in the repository before the associated paper is published.

EO applications rely on container images exposed in container registries. It is important to consider that image tags do not ensure the deterministic approach required to make sure the right container image is referenced in an application package. The deterministic approach for container images rely on their sha256 hash instead of their tag. Regarding container images identifiers (e.g. DOI), although it is possible to assign a DOI using the Zenodo platform with a series of manual steps, it has been determined that this is not an adoptable practice as runtime

environments (e.g. kubernetes image pullers) would not know how to consume a DOI instead of a container image reference.

EO Application Packages metadata rely on the schema.org metadata scheme. The OGC Best Practices for EO Application Packaging identifies a set of mandatory and recommended fields. This approach was extended during the testbed to include a DOI with the schema.org *sameas* field. EO Application Packages can have an identifier following this practice.

Regarding an EO Application Package execution, Terradue has identified an approach to assign an identifier to the inputs and results by identifying amongst the current STAC extensions the ones that support citation and a clear description of what data was used as input and what results were produced. The STAC extensions used were: *STAC Scientific Extension*, *STAC Datacube Extension*, and the *STAC File Extension*. By creating input and output STAC collections referencing the STAC Items used as inputs, the STAC Items produced, Collection and Items with the STAC extensions fields, both the inputs and results become citable through their identifiers.

Retrospective provenance using CWLProv comes as a powerful mechanism to trace the inputs, outputs, and intermediary files of an Application Package execution. Using the Bagit format of directories and files (including their hash), the retrospective provenance enriches the execution and provides an approach to verify the reproducibility of an execution against another.

7

OVERALL LESSONS LEARNED AND FUTURE WORK

OVERALL LESSONS LEARNED AND FUTURE WORK

7.1. The Expansion of Whole Tale's Capabilities

During the Testbed 18 activities Whole Tale was shown to be a useful resource for the advancement of reproducible workflows. However, there is a limitation with Whole Tale where a binary Docker image cannot be assigned a DOI through Whole Tale, which affects the reproducibility of the workflow. A work-around is to use a GitHub Action for publishing binary layers of the image in a dedicated repository. Then, create a release for this repository leading to the assignment of a DOI, using the Zenodo GitHub setting for this repository. From there, a user can use the archive provided with the tale's DOI to run it on the user's own infrastructure. Additional research and discussions with the Whole Tale team is recommended to add this functionality to Whole Tale in order to improve the reproducibility of workflows.

7.2. The Influence on Randomness in Reproducible Machine Learning Workflows

Randomness is important for deep learning models and applications for optimization and generalization, however this randomness inherent in the models makes true reproducibility a challenge. Randomness for deep learning models can be introduced through hardware, the environment, software frameworks, functions, and algorithms. Some initial identification of the ways to address randomness in deep learning models has been addressed in the Testbed 18 activities, however additional research, discussion, and identification of best practices for how to address randomness in deep learning models is recommended in order to improve reproducibility of deep learning workflows.

7.3. How Health-Related Workflows Will Benefit from FAIR data

Findable, Accessible, Interoperable, and Reproducible health and social data will better enable predictive analytics and risk stratification in healthcare, and advance the capabilities of global public health and emergency response communities. Two Health Geospatial Workflows that

were developed during the 2019 and 2021 OGC Disaster Pilots that will benefit from FAIR practices are the Mortality Risk Index and the Health Risk Index.

7.3.1. Mortality Risk Index

The Mortality Risk Index (MRI) utilizes data on population demographics and the prevalence of comorbidities to identify the risk to the underlying population of severe illness, hospitalizations, or mortality due to a specific disease. Most recently, the MRI was adapted to predict mortality for the COVID-19 pandemic. Health and policy decision makers can utilize the MRI to identify areas where measures to control the spread of disease are most warranted due to the concentrations of at-risk populations as well as allocate resources necessary to treat the infected. The health and social demographics data leveraged in producing the MRI are joined together down to the lowest granularity available (national, state, and county levels in the U.S. / Peru). The more granular data available through FAIR sources will increase the quality, and make more actionable, the results from the MRI.

7.3.2. Health Risk Index

The Health Risk Index (HRI) identifies the health and medical needs of a disaster-impacted population a priori enabling the emergency response and public health communities to plan for the provisioning of resources needed to meet those needs as a component of the overall disaster response planning. This early identification of needs will expedite health and emergency medical response, saving lives and reducing the overall cost of the response.

To provide this foresight, the HRI merges data on social determinants of health and health data to identify an impacted population's health risk from a natural disaster. This information can be utilized by emergency operations managers and on-the-ground emergency medical responders to ensure that their efforts address the unique health challenges caused by the specific natural disaster.

As in the case of the MRI, in scenarios where authoritative data sources may be unavailable within the time frame needed to support critical decision-making, alternate data sources that are still FAIR can provide valuable and actionable insights.



ANNEX A (INFORMATIVE) REVISION HISTORY



ANNEX A (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-03-07	r2	Jerome St-Louis	6.3	Ecere Updates
2022-11-18	0.1	Paul Churchyard	all	initial version



BIBLIOGRAPHY





BIBLIOGRAPHY

- [1] CodeMeta 2022 – <https://github.com/codemeta/codemeta>
- [2] CORINE Land Cover, 2022 – <https://land.copernicus.eu/pan-european/corine-land-cover>
- [3] DVC by iterative.ai, 2022 – <https://dvc.org/>
- [4] Non-deterministic algorithm, 2021 – <https://www.engati.com/glossary/non-deterministic-algorithm#:~:text=Non%2Ddeterministic%20models%20are%20primarily,be%20followed%20is%20equally%20preferable.>
- [5] Sentinel Online Overview, 2022 – <https://sentinel.esa.int/web/sentinel/missions/sentinel-2/overview>
- [6] GDAL, 2022 – <https://gdal.org/>
- [7] GeoTIFF, 2022 – <https://www.heavy.ai/technical-glossary/geotiff#:~:text=GeoTIFF%20is%20a%20public%20domain,be%20used%20in%20GIS%20applications.>
- [8] Versioning, provenance, and reproducibility in production machine learning. 2021 – <https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005>
- [9] Tuning Parameters and Overfitting, 2019 – <http://www.feat.engineering/tuning.html>
- [10] Landsat Science, 2022 – <https://landsat.gsfc.nasa.gov/>
- [11] Data Cube Interoperability, 2021 – <https://www.ogc.org/projects/initiatives/gdc>
- [12] Jacovella-St-Louis, J., Vretanos, P.A.: OGC API – Processes – Part 3: Workflows and Chaining (draft). Open Geospatial Consortium, <https://opengeospatial.github.io/ogcna-auto-review/21-009.html> (2023).
- [13] OGC API – COVERAGES, 2022 – <https://ogcapi.ogc.org/coverages/>
- [14] OGC API – FEATURES, 2022 – <https://ogcapi.ogc.org/features/>
- [15] PyTorch, 2022 – <https://pytorch.org/>
- [16] Mission, 2022 – <https://www.softwareheritage.org/mission/>
- [17] ISEA Based DGGs – <http://webpages.sou.edu/~sahrk/dgg/isea.old/isea.html>
- [18] Building a reproducible machine learning pipeline. *arXiv preprint arXiv:1810.04570*(2018)
- [19] Whole Tale Reproducibility Simplified, 2022 – <https://wholetale.org/>

- [20] About Zenodo, 2022 – <https://about.zenodo.org/>
- [21] Resource Description Framework(RDF) Model and Syntax Specification 1999 – <https://www.w3.org/TR/PR-rdf-syntax/Overview.html>