Evoland
LAND MONITORING EVOLUTION

D2.15 EvoLand infrastructure – final version

Work Package 2 – Method Development

Authors: Daniel Thiex (Sinergise)

Date: 30.09.2025

Funded by the European Union

Full Title	**Evolution of the Copernicus Land Service portfolio** (integrating novel EO data and latest machine learning algorithms to continuously monitor the status, dynamics and biomass of the land surface)			
Project number	101082130		**Acronym**	EVOLAND
Start date	01.01.2023		**Duration**	36 months
Granting authority	European Health and Digital Executive Agency (HaDEA)			
Project Coordinator	Vlaamse Instelling voor Technologisch Onderzoek N.V. (VITO)			
Date of delivery	**Contractual**	M33	**Actual**	M33
Type	R - Document, report		**Dissemination level**	PU - Public
Lead beneficiary	Sinergise			
Lead author	Daniel Thiex		**Email**	daniel.thiex@sinergise.com
Other authors	Gustav Janse van Rensburg (Sinergise),			
Reviewer(s)	Tim Ng (VITO)			
Keywords	Cloud infrastructure, Operationalisation, openEO Platform,			

Document Revision History				
Version	**Issue date**	**Stage**	**Changes**	**Contributor**
0.1	26.09.2025	Draft	Create initial draft	Sinergise
1.0	30.09.2025	Final	Review and finalization	Sinergise, Vito

Acronyms

Acronym	Description
API	Application Programming Interface
GDAL	Geospatial Data Abstraction Library
GIS	Geographic Information System
GPP	gross primary production
LC	Land Cover
LSC	Land surface characteristics
NDVI	Normalized difference vegetation index
UDF	User defined function
VS	Visual Studio

Contents

List of Tables

Applicable documents

ID	Applicable document
[AD1]	Grant Agreement
[AD2]	D1.5 Infrastructure requirements report - interim version
[AD3]	D1.6 Infrastructure requirements report – final version
[AD4]	D2.12 Best practices for running large scale EO processes in the cloud - interim version
[AD5]	D2.13 Best practices for running large scale EO processes in the cloud - final version

1 Introduction

This document is the **final version** (D1.15) of the "**EvoLand Infrastructure**" report for the EvoLand project [AD1] which incorporated lessons learnt from the first round of prototype development (completed in M17) as well as new changes that the second round of prototype development (M22 – M33) brought.

An interim version (D1.14) of the report was delivered in M15. The infrastructure used can have a significant impact on the scalability and cost efficiency (execution time and release cycles can also be categorised in the cost category) of the methods and prototypes developed. It is therefore important to have a good overview of the existing infrastructure. The interim version of the report was used to identify shortcomings and questions that were now answered in the final version.

The presented report will first present and evaluate the infrastructure used within phase 1 of the project followed by an update from phase 2 on the infrastructure proposed and used for the second iteration of the prototype development. The last chapter of the infrastructure section then focuses on weaknesses, strengths, and reasoning behind the proposed infrastructure both for phase 1 and phase 2.

Where appropriate, references to the "Infrastructure Requirements Report" [AD2] and the "Best Practices on Cloud Processing Report" [AD34] are drawn.

Before the report concludes with a summary, Chapter 3 looks at different factors that need to be considered from an infrastructure perspective concerning up-scaling the prototype sites and producing the results at an operational scale.

2 Infrastructure

Earlier in the project (M8 and M22), an analysis of infrastructure requirements was carried out [AD2, AD3], in which available infrastructure solutions (Hyperscalers, DIASes, Copernicus Data Space Ecosystem) were presented based on a prototype requirements assessment (at the time given) and a proposal was made for the infrastructure to be used in the EvoLand project.

It is important to bear in mind that the development of the prototypes (D3.1/3/5) at the interim version of this report was not yet finalised (and was still in its infancy at the time) at the time and that the intensive work on the prototypes in recent months has better defined the requirements, especially with regard to the concrete processes required to implement the developed methods. On the infrastructure side, the proposed infrastructure (openEO Platform) has been further improved (cf. 2.1 Infrastructure proposed), not only, but also thanks to the requirements defined by the prototypes within the EvoLand project.

At M33 we concluded the prototype development and can therefore present the final infrastructure used (2.4 2.4Infrastructure used in phase 2) to develop the prototypes within phase 2.

2.1 Infrastructure proposed in phase 1 (interim version of this report)

As part of AD2, we proposed to use the openEO Platform[1] as a large-scale cloud infrastructure in the project (cf. Chapter 4: Infrastructure proposal within AD2) and as we will see in the next chapter (cf. 2.3 Infrastructure used in phase 1 (interim version of this report)), some of the prototypes are already being developed there.

In these cases, there is a close collaboration between the backend providers offering the infrastructure and the developers of the prototypes (this is especially true for the VITO backend), which has led to the implementation of missing required features.

Below are some of the recent improvements that are worth mentioning. However, not all of the improvements listed are a direct result of the prototype development. This is one of the advantages of using an existing infrastructure. Due to the, albeit small, already established user base and the integration into other projects (e.g. Open-Earth-Monitor or Copernicus Data Space Ecosystem), the infrastructure is continuously improved even outside the direct requirements of the prototype, which ultimately benefits all users.

More recent (since AD2) development:

[1] https://openeo.cloud/

- Improved processing capabilities for synchronous jobs (increased speed and supported extent)
- Post processing of data, enabling new output file formats (netCDF and Zarr)
- Extended openEO UDF API to allow algorithms that change (e.g. increase) the resolution. The new API also supports renaming of bands within the UDF.
- OpenEO batch jobs can now write their output to a user-defined object storage bucket. This reduces the need for file manipulation in the user scripts, which is especially useful for operational setups.
- Export of vector data to GeoParquet
- The load_stac process can now read netCDF samples generated by openEO, allowing to build caches of EO training data.
- Various improvements to support more complex compositing methods, such as best available pixel and max-NDVI.
- Multiple time intervals can now be specified instead of just one when loading data.

A complete list of all improvements can be found in the open source repositories of the different backends that make up the openEO Platform: for the Vito Terrascope Geopyspark driver[2] and for the Sentinel Hub openEO Python driver[3]. There is currently no public changelog for the integrated EODC backend.

2.2 Infrastructure proposed in phase 2

As we laid out in AD3, we did not change the infrastructure proposed within phase 1 but rather extended our recommendation to rely, next to openEO within openEO Platform, also on the increasingly emerging openEO instances within the Copernicus Data Space Ecosystem[4]. In addition to the reproducibility and scalability advantages of the cloud, we also stressed the advantages of combining local set-ups for a quick initial development with cloud solutions.

Not only but also thanks to the usage and the feedback by the EvoLand prototype developers to the openEO backends, the backends further evolved and improved through the development phase. The most relevant improvements (Full changelog[5]) since the last version of this report are:

- A prototype for exporting to the new ZARR format.
- Multiple improvements for exporting data to object storage and metadata to STAC API's.

[2] https://github.com/Open-EO/openeo-geopyspark-driver/blob/master/CHANGELOG.md
[3] https://github.com/Open-EO/openeo-sentinelhub-python-driver/releases
[4] https://dataspace.copernicus.eu/analyse/openeo
[5] https://openeo.dataspace.copernicus.eu/openeo/CHANGELOG

- STAC 1.1 support for data import/export is being progressively rolled out.
- Reading and on the fly georeferencing of Sentinel-3 data
- Improvements to sampling of raster datasets, typically to collect training data for ML models
- Improvements to tooling that supports large scale processing.
- COG output format received small fixes to ensure that output is fully "dissemination ready"
- Initial support for docker based 'CWL' processing, allowing to integrate any software that runs in Docker containers.

2.3 Infrastructure used in phase 1 (interim version of this report)

At this time of the project, both method development (T2.4 – T2.7) and prototype development (T3.1 – T3.5) are at a stage where they are intensively testing their workflows through running code and are therefore heavily reliant on the use of computing resources (infrastructure).

The resources used can be roughly divided into three categories:

- Local machines (powerful personal devices): always available, setup entirely determined by the owner, easy installation of additional software, if any, limited expandability of processing capabilities
- Internal infrastructure (mainly GPU clusters operated by the partner itself): Familiarity due to in-house set-up, finite but more scalable than local machines, often higher costs but internal billing
- Cloud infrastructure (openEO Platform): new ecosystem to get used to, finite, but much more scalable than internal infrastructure, more cost effective

Within the method development we see that currently most methods use a combination of all three resource groups (T2.5 and T2.6), one combination of internal and cloud infrastructure (T2.4) and one only local processing (T2.7) (Table 1).

Table 1. Infrastructure used within the method development.

	Local machine	Internal infrastructure (GPU cluster within HPC/Linux cluster)	Cloud infrastructure (openEO/openEO Platform)
T2.4 Weakly supervised learning	No	Yes	Yes (for data collection)
T2.5 Improved resolution	Yes	Yes	Yes
T2.6 Continuous monitoring	Yes	Yes	Yes
T2.7 Biomass mapping	Yes	No	No

On the prototype side two prototypes rely on all three groups (C4, C5), two on internal and cloud infrastructure (C8, C11), three on local machine and internal

infrastructure (C1, C2, C3) and four solely on internal infrastructure (C6, C7, C9, C10) (Table 2).

Table 2. Infrastructure used within the prototype development.

	Local machine	Internal infrastructure (GPU cluster within HPC/Linux cluster)	Cloud infrastructure (openEO/openEO Platform)
C1 Continuous forest monitoring	Yes	Yes	No
C2 Forest disturbance mapping	Yes	Yes	No
C3 Forest biomass mapping	Yes	Yes	No
C4 Cover crop type mapping	Yes	Yes	Yes
C5 Crop-/grassland GPP monitoring	Yes	Yes	Yes
C6 Small landscape features mapping	No	Yes	No
C7 Improved water bodies mapping	No	Yes	No
C8 Continuous imperviousness monitoring	No	Yes	Yes (for data collection)
C9 automated land use mapping of urban dynamics	No	Yes	No
C10 Continuous mapping of LSC	No	Yes	No
C11 On-demand LC mapping	No	Yes	Yes

On the software side partners rely on a mix of

- different GIS software (ArcGIS Pro, Qgis, GDAL),
- different python packages for the data collection (e.g. rasterio, sensors-io, pandas), and
- python packages for the deep-learning part (e.g. PyTorch, Hydra, lightning).

In general, the strong use of python within the EO model development is also reflected within the EvoLand project as all methods and all prototypes are being developed in python (T2.7 additionally having some parts being developed in matlab). Next to specific programs used for the data preparation (first bulled above) a lot of the work is done directly in code editors like VS Code or Juypter Notebooks (cf. Infrastructure in use 02/2024 overview table in the

Appendix).

2.4 Infrastructure used in phase 2

Looking at the method development we see that all four methods rely on the same infrastructure in phase 2 (Table 3) as in phase 1 (Table 1). This infrastructure is a combination of all 3 categories, with most (T2.5 and T2.6) using all 3 options (local, internal and cloud infrastructure), and only T2.7 relying fully on local processing.

Table 3. Infrastructure used within the method development.

	Local machine	Internal infrastructure (GPU cluster within HPC/Linux cluster)	Cloud infrastructure (openEO/openEO Platform/CDSE)
T2.4 Weakly supervised learning	No	Yes	Yes (for data collection)
T2.5 Improved resolution	Yes	Yes	Yes
T2.6 Continuous monitoring	Yes	Yes	Yes
T2.7 Biomass mapping	Yes	No	No

On the prototype side we could see some more changes from phase 1 to phase 2 with 3 prototypes (C1, C2, C10) switching (partially) their workflow to openEO/the cloud and only 1 (C8) that previously used openEO/the cloud now fully relying on internal infrastructure. In total this means the most popular infrastructure is still the internal one (some reasons for this are discussed in 0 On the software side the most interesting change compared to phase 1 is that the formerly used commercial GIS software ArcGIS Pro wasn't used anymore, and method/prototype developers fully relied on open-source software (QGIS, GDAL, Orfeo ToolBox).

All methods and prototypes still heavily use Python and different python packages for data calculations (e.g. SciPy, PySTAC) and deep learning (e.g. PyTorch, Hydra, Lightning, CatBoost).

A lot of the code was written in VS Code (used by 6 prototypes) (cf. Infrastructure in use 09/2025 overview table in the

Appendix).

Weaknesses, strengths and reasoning behind the currently used infrastructure). However, with 6 prototypes using openEO/the cloud, half of the prototypes are running on the in phase 1 proposed infrastructure.

Table 4. Infrastructure used within the prototype development.

	Local machine	Internal infrastructure (GPU cluster within HPC/Linux cluster)	Cloud infrastructure (openEO/openEO Platform)
C1 Continuous forest monitoring	Yes	Yes	Yes
C2 Forest disturbance mapping	Yes	No	Yes (inference part)
C3 Forest biomass mapping	Yes	No	No
C4 Cover crop type mapping	Yes	Yes	Yes
C5 Crop-/grassland GPP monitoring	Yes	Yes	Yes
C6 Small landscape features mapping	No	Yes	No
C7 Improved water bodies mapping	No	Yes	No
C8 Continuous imperviousness monitoring	No	Yes	No
C9 automated land use mapping of urban dynamics	No	Yes	No
C10 Continuous mapping of LSC	No	Yes	Yes
C11 On-demand LC mapping	No	Yes	Yes
C12 Tree type mapping	Yes	No	No

On the software side the most interesting change compared to phase 1 is that the formerly used commercial GIS software ArcGIS Pro wasn't used anymore, and method/prototype developers fully relied on open-source software (QGIS, GDAL, Orfeo ToolBox).

All methods and prototypes still heavily use Python and different python packages for data calculations (e.g. SciPy, PySTAC) and deep learning (e.g. PyTorch, Hydra, Lightning, CatBoost).

A lot of the code was written in VS Code (used by 6 prototypes) (cf. Infrastructure in use 09/2025 overview table in the

Appendix).

2.5 Weaknesses, strengths and reasoning behind the currently used infrastructure

Examining the utilization of local machines in the development process reveals that the majority of methods (3/4) and approximately half of prototypes (5/11) depend on this category. Local machines have the advantage of being highly flexible, as the required external software can be easily installed. This is particularly important for data preparation, when the in-situ and remote sensing data must first be cleaned and “prepared" for processing in the pipeline (in our case often for machine learning models), and possibly a reason for the high use in method development.

Above all, cleaning and preparing the data are generally not so resource-intensive and execution on the local computer often enables fast development cycles, as the computing resources are not dependent on external resources and the user has full control at all times (cf. chapter 2 Best Practices within [AD34]).

With the exception of one method (T2.4) and two prototypes (C2, C3), all methods and all prototypes rely on internal infrastructure. This is not surprising, as the local infrastructure is often easier to use in the short term for the partners working on the method/prototype, as they already know how to use it and what is available, it often already exists and does not have to be developed/set up during the course of the project and often contains much of the required data.

Currently, most methods (3 out of 4) and six of the twelve prototypes are running part or all of their workflow using openEO (on openEO Platform or CDSE). It's noteworthy that within phase 2, three new prototypes switched to openEO and only one reverted to fully relying on internal infrastructure. One of the biggest unknowns is to what extent the locally/internally developed workflows can be transferred to the openEO syntax and backends supporting openEO such as openEO Platform and CDSE. This may not be possible for the entire workflow for all prototypes, as not all steps are supported by openEO (e.g. not all machine learning models are implemented), but can only be definitively answered once the method and prototype development is complete (M17 for the first round, M31 for the second round). The fact that so many methods and prototypes are developing part of their workflows with the aim of being executed on the proposed infrastructure can be considered a success and is certainly also due to the fact that the project decided to rely on an already existing and functioning infrastructure and to further develop it instead of building a new project infrastructure from scratch. Another limiting factor mentioned, the lack of data availability, is currently being addressed as part of Task 2.3.2 (Data Access), and in the second iteration, the prototype partners will be able to integrate some of their local data into the openEO workflows. Despite improvements here (further support for loading data from STAC within openEO,

the emergence of new relevant datasets like the CDSE Sentinel-1 Mosaics), access to relevant datasets such as VHR and in-situ data remains a challenge mentioned by several prototype developers (cf. Infrastructure in use 09/2025 overview table in the Appendix). Another challenge that remains is the cost and resources needed to train models. Using cloud infrastructure for this has the advantage of scaling and descaling on demand, but regardless of the selected infrastructure, high costs due to resource-intensive training processes remain a challenge.

3 Potential operational roll-out

If we only look at the capabilities of the cloud infrastructure (openEO on openEO Platform or CDSE) itself, it is already in a situation where it can produce continental results at a reasonable cost as impressively proven by Vito within the WorldCereal project[6].

The extent to which the EvoLand project can in the end utilize those cloud infrastructure depends largely on how many of the workflows can be transferred to them (assessment to be done per method/prototype) and how to connect the steps that cannot be transferred or that are easier/more sensible done outside the platform. Tasks that only need to be done once, such as initial data clean up or training of models come to mind here.

It is still too early to answer these questions, as the first round of prototype development is still in full swing. However, a few things are already clear:

- The fact that all methods and prototypes are developed in Python is an advantage, as the openEO Python client is well supported and represents one of the main ways in which users interact with openEO Platform.
- Certain key metrics (e.g. costs or runtimes) can only be evaluated once the workflows are final (M17 for the first period and M33 for the second period).
- Due to the different requirement (cf. AD2), not all prototypes are likely to run on the openEO Platform.
- As the final report is to be submitted at the same time as the second round of prototypes, not all final developments can be included in the final report.

Even if there are no answers to the questions yet (at the interim version of this report), they can already be collected and will be answered in the final version of this report. Relevant questions are:

- How does the final infrastructure set-up look like?
- What are the costs and runtimes per prototype execution?
- How does this change when scaling up (from the prototype extent to larger extents)?
- How many and which parts of the prototypes can be transferred to the openEO Platform?
- What are the benefits/risks of the final set-up?

Being at the end of the second phase of prototype development, we can now try to answer the questions posed at the interim version of this report.

[6] https://dataspace.copernicus.eu/cases/global-cropland-monitoring-esa-worldcereal-and-openeo

Within chapter 2.5 Weaknesses, strengths and reasoning behind the currently used infrastructure we laid out how the infrastructure used by each prototype looks like in detail.

Runtimes per prototype vary from under a minute per tile (C7) up to hours (e.g. C9). One factor that greatly influences the runtime is whether the data was already collected or first needs to be searched, filtered and potentially downloaded (cloud processing with direct access to the data is advantageous here) and the temporal and spatial resolution (cf. Infrastructure in use 09/2025 overview table in the Appendix).

Another question we asked in the interim version of this report was how does scaling (temporal and spatial) affect costs and runtimes? Producing the prototype results at scale showed that runtimes increase from between several days up to 1 month for e.g. processing one year of results on European level (C3). The costs are naturally harder to assess especially for those parts of the processing that were done locally or on internal solutions. For solutions running in the cloud (using openEO) costs range from 5 to 12 openEO credits for each 100 km2 of observation. Within CDSE each user gets 10.000 free credits per month which translates into creating results for 1 observation on a country scale (between around 80.000 km2 and 200.000 km2).

Concerning the transfer of workflows to openEO, our analysis from the interim version confirmed that we can transfer some more prototypes (absolute +2) to openEO but that certain workflows will remain running locally.

The benefits and risks of the final set-up have already been described. The risks include too high a reliance on local/internal set-ups due to missing capabilities in the cloud, and thus hard-to-access cost monitoring. The benefits include the constant evolution of the infrastructure due to the choice of a public cloud federation, free resources within CDSE for testing, and significant processing and good scalability.

4 Summary

The evaluation of the established infrastructure with a view to a possible operational roll-out revealed a lot of relevant findings and helped to identify important questions for further action. Several methods and prototypes within the project are already utilising the common project infrastructure (openEO Platform), and several more are planning to test its capabilities in the coming months.

However, as mentioned above, there are also advantages (e.g. readiness, familiarity and ease of use) in using local and internal infrastructures, and several sections of the workflow are being executing on such. Helping partners to access the openEO capabilities and evaluate where it makes sense to transfer processing to the common infrastructure is the main question identified to be answered in the next month.

It should also be mentioned that the evaluation of still on-going and not yet finalized work, cannot answer all questions nor be final. Nevertheless, capturing the status quo has proven to be helpful in evaluating the next steps and determining how to proceed.

4.1 Update for phase 2

In the 2nd phase of the prototype development, we saw further adaptation of the proposed infrastructure and prototypes extending their openEO usage to the now well established CDSE. Nevertheless, the internal infrastructure remains the most popular as it is the easiest to set up and the prototype developers are familiar with the usage and capabilities.

Thanks to the continuous development within phase 2 we were able to answer all open questions we posed at the interim version of this report (cf. 2.5 Weaknesses, strengths and reasoning behind the currently used infrastructure).

Thanks to the help of the EvoLand project the openEO backends received valuable feedback whose implementation directly benefited the prototypes and the whole user base. In the future we need to continue to execute real world examples on public European Cloud infrastructure to ensure backends focus on the relevant capabilities.

In conclusion, the combination of local/internal infrastructure set-up for quick iterations on low scale with later transferring resource-intensive processes to scalable cloud solution (openEO Platform/CDSE) proved a good solution for half of the prototypes. To support more prototypes, the infrastructure needs to continue to extend on its capabilities from a processing as well as data availability point of view. The latter being already partially addressed by the option to load external data from STAC.

Appendix

- Infrastructure in use 02/2024 overview table
- Infrastructure in use 09/2025 overview table

Evoland
LAND MONITORING EVOLUTION

EvoLand Consortium

vito Remote Sensing

International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

CLS
COLLECTE LOCALISATION SATELLITES

CESBIO

SINERGISE

JOANNEUM RESEARCH

cnes

DLR

GAFAG
an e-GEOS (ASI / Telespazio) Company

evenflow

Candidate CLMS prototype	Description	T2.4	T2.4	T2.5	T2.6	T2.7
		Weekly Supervised Learning	Weekly Supervised Learning	Improved Spatial/Temporal/Spectral Resolution	Continous Monitoring	Biomass Monitoring
Lead		**UT3**	**DLR**	**UT3**	**VITO**	**JR**
Person working on the method/protoype (main lead)		**Jordi INGLADA (CESBIO)**	**Nassim Ait-Ali Braham (DLR)**	**Julien MICHEL (CESBIO/CNES)**	**Wanda De Keersmaecker**	**Heinz Gallaun**
Infrastructure	Infrastructure you are currently using to develop the method/prototype (e.g. local machine, internal solution, openEO Platform...)	Data-set collection: openEO for S1, S2, AGERA-5 and DEM Model training: CNES HPC infrastructure (including A100 GPU nodes) Jean Zay Idris (CNRS) HPC infrastructure (A100 and V100) under grant 2023-AD010114923 For production / inference: CNES HPC infrastructure, OpenEO platform, and local machine	SSL4EO-S12 (ESA Sentinel-1/2) and SpectralEarth (DLR EnMAP) model pretraining: JUWELS Booster Juelich Supercomputer GPU (NVIDIA A100s) cluster; SSL4EO-S12 data preparation: Google Earth Engine, EnMAP data preparation: DLR internal	**For Dataset Collection:** CNES HPC infrastructure (CPU) for sen2venµs extended dataset OpenEO for Landsat-Sentinel2 time-series (up-coming) **For Training:** CNES HPC infrastructure (including A100 GPU nodes) Jean Zay Idris (CNRS) HPC infrastructure (A100 and V100) under grant 2023-AD010114835 **For production / inference:** CNES HPC infrastructure, OpenEO platform, and local machine	**Disturbance classification** (JR): openEO platform, local desktop workstations, Linux cluster running openSuSe 15.5 (SLURM managed)	JR: local desktop workstations UT3: laptop
Infrastructure (Internal)	Describe the hardware of the internal solution, ie, CPU, GPU etc.	NVIDIA V100 and A100 gpus nodes fom HPC infrastructures listed above	DLR terrabyte GPU cluster (NVIDIA A100s)	NVIDIA V100 and A100 gpus nodes fom HPC infrastructures listed above	JR: Linux cluster hardware: 10 physical machines, different CPUs up to Core i7 12700, RAM typically 64 GB	JR: Core i9 12900, RAM 64 GB UT3: Intel Core i7-7820HQ CPU @ 2,90GHz, RAM 16 GB
Software	List any program you are using to process/modify the data	openEO / rasterio / pandas for data collection pytorch / lightning / hydra for deep-learning onnxruntime for model export and end-user tools	PyTorch Lightning + Hydra + common Python geospatial packages (e.g., rasterio, etc.); self-supervised models trained (ResNet and ViT backgones) and publicly available through TorchGeo Python library (model weights on HuggingFace)	rasterio / sensorsio / dask / pandas for data collection pytorch / lightning / hydra for deep-learning onnxruntime for model export and end-user tools	JR: QGIS, Python 3 standard library plus numpy, scipy, sklearn, gdal or similar geodata I/O backends	JR: ArcGIS Pro, Python 3 standard libraries UT3: Matlab, QGis, Gdal, OrfeoToolBox
Programming language	Language you are using (e.g. R, Python...)	full python stack	Python	full python stack	JR: Python	JR: Python UT3: Matlab
Shortcoming/Blockers	Describe any shortcoming/Blockers you are facing with the currently used Infrastructure	**For training :** CNES HPC limited number of GPUs has been a shortcoming at the end of 2023, but it has been solved by requesting the grant on Jean-Zay HPC, which has far more processing power. **For inference :** Producing data required by prototypes on OpenEO will be challenging because of the lack of GPU processing.	**data preparation:** limited openEO budget	**For training :** CNES HPC limited number of GPUs has been a shortcoming at the end of 2023, but it has been solved by requesting the grant on Jean-Zay HPC, which has far more processing power. **For inference :** Producing data required by prototypes on OpenEO will be challenging because of the lack of GPU processing. It must be stressed however that the developped OpenEO UDF allows to process smaller areas if a large coverage is not required	JR: None for method development	JR: None for method development UT3: None
Scalability of current approach	How well does your current approach work if you'd need to execute it on e.g. over whole Europe or 1x per month	Needs to be assessed.	model inference for feature vectors over all of Europe heavily depends on SSL backbone picked and available hardware (CPU vs. GPU); **model pre-training** - SpectralEarth: implemented with (scalable) multi-GPU node training; SSL4EO-S12: no multi-GPU node training enabled, yet	Single Image Super-Resolution is embarassingly parallel (e.g. each product can be processed independantly). Therefore, this is only a matter of how much processing power is available and what is the volumetry required by prototypes.	Disturbance classification (JR): should scale approx. linearly w.r.t. the number and size of disturbance patches to be classified	JR: scales approx. linearly w.r.t. the number of tiles processed UT3: same
Timeline	How long does one iteration (one cycle) to execute your pipeline once currently takes (in days)?	Needs to be assessed.	needs assessment	A full Sentinel2 product can be processed in 5 hours (CPU, single thread), 50 minutes (CPU, 8 threads) or 5 minutes (GPU)	Needs assessment (what exactly is "one iteration")	JR: for non-parametric estimation, for one S2 granule processing time is ~5 hours (excluding processing of composites such as S2 compositing over vegetation period) UT3: for parametric estimation, for one PALSAR tile (1° x 1°), a few minutes
Are you planning to transfer you workflow to openEO	Yes/No (If no, why not. What are the blockers?)	Only the inference step	If PyTorch Lighning models can be integrated for inference that is a potential option depending on how easy the plug-and-play works	Yes, already in progress: https://framagit.org/jmichel-otb/openeo_superresolution	JR: depends on requirement of GAF regarding prototype production, not foreseen at the moment	JR: not foreseen at the moment UT3: same
Comments/Questions		N. A.	N/A	N. A.		N.A.

Candidate CLMS prototype	C1	C2	C3	C4	C5	C6
	Continous forest monitoring	Forest disturbane mapping	Forest biomass mapping	Cover crop type mapping	Corpland/grassland GPP monitoring	Small landscape features mapping
Lead	**GAF**	**GAF**	**GAF**	**VITO**	**VITO**	**CLS**
Person working on the method/protoype (main lead)	**(supported by DLR)**			**Roel Van Hoolst**	**Kasper Bonte**	**Faucqueur Loïc**
Infrastructure	*DLR support* - SSL4EO-S12 model fine-tuning for single timestamp binary forest maps: JUWELS Booster Juelich Supercomputing GPU cluster (NVIDIA A100), DLR terrabyte GPU cluster (NVIDIA A100); unsupervised forest change detection on before-after timestamps: single GPU (no HPC cluster, desktop device sufficient)			local machine, Terrascope VM, openEO platform	local machine, Terrascope VM, openEO platform	Internal solution
Infrastructure (Internal)				16GB RAM, integrated GPU, intel Core vPRO i5	16GB RAM, integrated GPU, intel Core vPRO i5	CPU Processing server : - CPU : 52 - RAM : 252GB GPU server available if needed
Software	*DLR support* - PyTorch Lightning + Hydra + common Python geospatial packages (e.g., rasterio, etc.) + Jupyter notebooks (Jupyter Lab)			VS code	VS code	VS Code
Programming language	*DLR support* - Python			Python	Python	Python
Shortcoming/Blockers	*DLR support* - **data preparation:** accuracy of data annotation for training single timestamp model limited by annual product (HRL 2018); **model training:** models trained on external/non-project infrastructure (required additional proposal)			Limited amount of in-situ data	Limited amount of in-situ data	In-situ data availability EO data availability
Scalability of current approach	*DLR support* - single (binary forest map) and dual timestamp (forest change detection) input does not require full time series input (inference proven to run on commodity CPU); limitation: model fine-tuning requires at least a single GPU			Should be evaluated	Should be evaluated	to be evaluated
Timeline						
Are you planning to transfer you workflow to openEO				Yes	Yes	Not assessed yet
Comments/Questions						

Candidate CLMS prototype	C7	C8	C9	C10	C11
	Improved water bodies mapping	Continous imperviousness monitoring	Automated land use mapping of urban dynamics	Continous mapping of land surface characteristics	On-demand land cover mapping
Lead	**CLS**	**CLS**	**CLS**	**VITO**	**VITO**
Person working on the method/protoype (main lead)	**Christophe Fatras**	**Sadri Haouet**	**Eve poitevin**	**Wanda De Keersmaecker**	**Nick Gutkin**
Infrastructure	Internal solution	Internal solution Data collection : openEO for EO data (sentinel-2)	Internal solution	Internal solution: Terrascope VM, GPU platform of VITO	local machine, TerraScope VM, openEO platform
Infrastructure (Internal)	Internal server with CPU & GPU	CPU Processing server : - CPU : 52 - RAM : 252GB GPU server available for DL development	CPU	Training on GPU (30 TB local storage, 48 (Intel) CPU cores,370GB CPU memory, 8 RTX 2080 Ti Nvidia GPU's, 11 GB GPU memory), inference on CPU	16GB RAM, integrated GPU, intel Core vPRO i5
Software	VS code	Python librairies: data handling: rasterio, pandas, geopandas, nmpy, sklearn ML / DL: Broceliande, Tensorflow, Keras	VS code	VS Code	VS Code
Programming language	Python	Python	Python	Python	Python
Shortcoming/Blockers	No access to VHR for validation yet	In situ data availability on all test sites and representative enough regarding the prototype goal	No access to VHR for validation yet	None so far	Takes a long time to get feedback from openEO, sometimes errors are not clear
Scalability of current approach	First approach potentially scalable to global operation - still under investigation Super-resolution too long for a global approach; still working on that			No problems at this point	Definitely challenges with scalability, may need to incorporate tiling of training data and/or predictions
Timeline		Not assessed yet			0,04 days (1 hr)
Are you planning to transfer you workflow to openEO	Needs to be investigated	Not assessed yet		Yes, but I'm not sure if all steps can be implemented in OpenEO	Yes, will already be written and implemented fully in openEO. Only potential blocker is lack of other machine learning models in openEO besides Random Forests
Comments/Questions	NA				

Candidate CLMS prototype	Description	T2.4	T2.4	T2.5	T2.6	T2.7
		Weekly Supervised Learning	Weekly Supervised Learning	Improved Spatial/Temporal/Spectral Resolution	Continous Monitoring	Biomass Monitoring
Lead		UT3	DLR	UT3	VITO	JR
Person working on the method/protoype (main lead)		Jordi INGLADA (CESBIO)	Nassim Ait-Ali Braham (DLR)	Julien MICHEL (CESBIO/CNES)	Wanda De Keersmaecker	Heinz Gallaun (JR)
Infrastructure	Infrastructure you are currently using to develop the method/prototype (e.g. local machine, internal solution, openEO Platform...)	OpenEO for data access, CNES and Idris computing center (jean-zay) for GPU processing power.	LRZ Terrabyte Cluster and JUWELS Booster Module from Forschungszentrum Jülich	OpenEO for data access, CNES and Idris computing center (jean-zay) for GPU processing power.	Internal solution and openEO	local machine
Infrastructure (Internal)	Describe the hardware of the internal solution, ie, CPU, GPU etc	GPFS + H100 GPUs	A100 GPUs	GPFS + H100 GPUs	Intel core Ultra 5 135U, 16GB RAM	Intel(R) Core(TM) i9, 64GB RAM, GPU: NVIDIA GeForce RTX
Software	List any program you are using to process/modify the data	Python stack : pytorch + hydra + lightning	Python, pytorch lightning, hydra	Python stack : pytorch + hydra + lightning	openEO, Python	Python, Scipy
Programming language	Language you are using (e.g. R, Python...)	Python	Python	Python	Python	Python
Shortcoming/Blockers	Describe any shortcoming/Blockers you are facing with the currently used Infrastructure		Training models is resource intensive. Limited quota of GPU hours on super computers.	Long training time (>10 days) even on H100. This limits the number of experiments.		
Scalability of current approach	How well does your current approach work if you'd need to execute it on e.g. over whole Europe or 1x per month	We did not perform inference on openEO, but computing times should be of several minutes per tile on GPU and about 2 hours on CPU (considering time series of covering 1 year)	It varies drastically depending on the model and available hardware (data loading time, GPU availability, etc). For a ResNet-50 model, a Sentinel tile can be processed in a few minutes (10-20) with a CPU (it also depends on the stride). If a high end GPU is available (and depending which), the inference time can be reduced to less than a minute.	For super-resolution, inference time for one MGRS tile and one date is 6 minutes on GPU and 6h on a single CPU. For Landsat/Sentinel-2 fusion, inference time for full year of Landsat and Sentinel-2 acquisition generating 15 synthesis (24 dates/year) on a single MGRS tile is 1 hour on GPU, and 130 minutes using 16 CPUs.	Inference was performed in openEO. It costs about 11 credits to process a 10 x 10 km block (10m resolution and single image). Most likely this can be reduced to 6 credits by further optimization of the workflow	processing over whole EEA38 countries ~1 month for each year for which biomass is estimated, on a PC (i9, 64GB RAM, GPU) as described above. (only if quaterly mosaics are provided as input e.g. by S2GM service). Integration of L- or P Band SAR timeseries required.
Operational role out	List a few things that should happen next in order to transfer your workflow into an operatinal state				Model was already trained globally, so applicable globally. Further model optimization to reduce confusion in the vegetation classes may strengthen its usage for downstream applications	In addition to Sentinel 1 & 2, also L- or P Band SAR time series (e.g. from NISAR / BIOMASS satellites which were launched recently) are required in ecosystem regions characterised by high biomass such as in Central Europe or tropics.
Timeline	How long does one iteration (one cycle) to execute your pipeline once currently takes (in days)?	See scalabiity section.	See above.	See scalabiity section.	10km x 10km block (single observation) with 10m resolution takes 11 openEO credits, possibly reduced to 5 credits	for wall-to-wall EEA38, ~30 days for one yearly biomass product, by using one PC as described above.
Are you planning to transfer you workflow to openEO	Yes/No (If no, why not. What are the blockers?)	Done for the selected methods	Currently uncertain.	Done for super-resolution. For Landsat / Sentinel-2 fusion it would require evolution of OpenEO capabilities	Inference is in openEO	Currently uncertain
Lessons learned from the 1st round of prototype development	Please list at least 3 things you learned/took from the 1st round of prototype development					
Comments/Questions						

Candidate CLMS prototype	C1	C2	C3	C4	C5
	Continous forest monitoring	Forest disturbane mapping	Forest biomass mapping	Cover crop type mapping	Corpland/grassland GPP monitoring
Lead	GAF (supported by DLR)	GAF	GAF	VITO	VITO
Person working on the method/protoype (main lead)	Elmer Ekinzog	Elmer Ekinzog	Fahad Jahangir (GAF), Heinz Gallaun (JR)	Mathilde De Vroey	Astrid Vannoppen
Infrastructure	Local machine, WekEO (CDSE backend)	Local machine	JR: local machine	Local machine, Terrascope VM, openEO (CDSE backend)	Local machine, Terrascope VM, openEO (CDSE backend)
Infrastructure (Internal)	Intel(R) Xeon(R) W-2245, 64GB RAM, GPU: NVIDIA GeForce GTX 1050 Ti	Intel(R) Xeon(R) W-2245, 64GB RAM, GPU: NVIDIA GeForce GTX 1050 Ti	JR: Intel(R) Core(TM) i9, 64GB RAM, GPU: NVIDIA GeForce RTX	16GB RAM, integrated GPU, intel Core vPRO i5	16GB RAM, integrated GPU, intel Core vPRO i5
Software	Conda, Python	Conda, Python	JR: Python, Scipy	VisualStudioCode, Qgis, gdal, openeo, CatBoost	VisualStudioCode, openeo
Programming language	Python	Python	JR: Python	Python	Python
Shortcoming/Blockers		Limited amount of in-situ data		Limited amount of in-situ data	in-situ data (icos network)
Scalability of current approach	- Data preprocessing requires a lot of resources. 3 WeKEO VMs eco2.4xlarge were for preprocessing for monthly aggregates per tile over 6 years. '- Monitoring itself is faster (~40 mins) for a typical (100 x 100 km) S2 tile on the Infrastructure (Internal).		JR: processing over whole EEA38 countries ~1 month for each year for which biomass is estimated, on a PC (i9, 64GB RAM, GPU) as described above. (only if quaterly mosaics are provided as input e.g. by S2GM service). Integration of L- or P Band SAR timeseries required.	- The heaviest step is the feature extraction in openeo: 1-2 days for 1 year and all of Belgium , i.e. approx 6 S2 tiles (note: with only 2 concurrent jobs + process could probably be optimized), should scale linearly with number of tiles. - Pre-processing, model training, inference and post-processing steps are ran locally, but are certainly scalable (depending on the availability of reference data)	scalable, we are already working on this in the framework of EEA funded hrvpp2 project.
Operational role out	Parallelization of processing to speed up pre-processing of S2 data, can be managed with VMs.	Model trained on whole of Europe. Inference can be carried out on CDSE (implementation on https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/forest_disturbance_classification/)	JR: In addition to Sentinel 1 & 2, also L or P-Band SAR time series (e.g. from NISAR / BIOMASS satellites which were launched recently) are required in ecosystem regions characterised by high biomass such as in Central Europe or tropics.	- Get more training data - Optimization of feature extraction in openeo - Parallelization of processing	currently bussy in framework of hvpp2 https://remotesensing.vito.be/services/copernicus-land-monitoring-service-hr-vpp
Timeline	Approx. 40 per tile for monitoring (could be improved by increasing chunk size, limited by RAM only).	Approx. 10 per tile (inference only).	JR: for wall-to-wall EEA38, ~30 days for one yearly biomass product, by using one PC as described above.	Approx. 1 day per tile (could be improved)	3 days for 1 year 1 tile
Are you planning to transfer you workflow to openEO				Partially done, no further transfer planned	already in openEO
Lessons learned from the 1st round of prototype development	Cloud masking by SCL was replaced by CLOUDSEN12 mask.				c3/C4 temperature dependency factor was removed in version 2, since the number of icos towers with c4 crops is very limited (i.e.not sufficient for a good callibration&validation)
Comments/Questions					

Candidate CLMS prototype	C6	C7	C8	C9	C10	C11	C12
	Small landscape features mapping	Improved water bodies mapping	Continous imperviousness monitoring	Automated land use mapping of urban dynamics	Continous mapping of land surface categories	On-demand land cover mapping	Tree types mapping
Lead	CLS	CLS	CLS	CLS	VITO	VITO	GAF
Person working on the method/protoype (main lead)	Faucqueur Loïc, Eve Poitevin	Christophe Fatras, Iris Lucas	Alexandre Pennec, Hoa Vu	Eve Poitevin, Yoann Courmont	Wanda De Keersmaecker	Manu Goudar, Joris Codde	Fahad Jahangir (GAF)
Infrastructure	Internal solution	Internal Solution	Internal solution	Internal solution	Internal solution and openEO	internal solution and openEO with Terrascope backend	local machine
Infrastructure (Internal)	51 CPUs 120G RAM	51 CPUs 70G RAM	51 CPUs 70G RAM	51 CPUs 100g RAM	Intel core Ultra 5 135U, 16GB RAM	Intel core Ultra 5 135U, 16GB RAM	Intel(R) Core(TM) i9, 64GB RAM, GPU: NVIDIA GeForce RTX
Software	VisualStudioCode, Qgis, gdal,	VisualStudioCode	VisualStudioCode, QGIS	VisualStudioCode, QGis, OTB	openEO, Python	openEO, gdal, pystac	Python, Tensorflow
Programming language	Python, bash	Python	Python	Python, bash	Python	Python, bash	Python
Shortcoming/Blockers	Availability/collection of VHR data			Uses of a lot of CPUs and RAM, and takes long time to process			
Scalability of current approach	Most impactful step is conversion of S2 time series to Super Resolved version. Once ready, computation of products is around 1h/tile. Easily scalable	Once input data is ready (shadow masks which must be computed once), the running of th algorithm is quite quick, the distinction of permanent / temporary water bodies which is a second processing step (data reprocess) is a one shot process for the moment.	- For the status layer, it can be very long for the production covering entire Europe. - For the change layer, it is more feasible (6 times less than status layer)	With the 10m output resolution, will work with low latency, but with the 5m output resolution, the computing is much longer (work to carry on on large images computing to have a scalability with this approach)	Inference was performed in openEO. It costs about 11 credits to process a 10 x 10 km block (10m resolution and single image). Most likely this can be reduced to 6 credits by further optimization of the workflow	Inference and post-processing were performed in openEO: the input features exist as stacs then it takes about an average of 12 openEO credits to process a 10 km x 10 km block (10m pixel resolution).	TempCNN models are lightweight compared to transformers or very deep networks, allowing training on standard CPUs. Inference on Sentinel-2 tiles is feasible with tiling, but processing 25 tiles already demands significant compute and storage. Scaling to country or continental-level mapping will require distributed computing
Operational role out	- Parallelization of processing to speed up pre-processing of S2 data. - Automatisation of VHR processing scalable. - Collection of VHR data remains limiting point	-Include automatic retrieving of S2 data - Parallelisation of processing -Include a cloud shadow mask as input -Industrialize the permanent / temporary disctinction phase at the end of water map estimation	- Optimize the status layer production to reduce computation time - Include external data for post-processing in order to reduce noise	- automatic retrieving of input data - work on large images optimization - get more cadastral data of different countries - train more local models (one per some s2 tiles) to fit the whole Europe	Model was already trained globally, so applicable globally. Further model optimization to reduce confusion in the vegetation classes may strengthen its usage for downstream applications	Code is ready for processing the whole globe	Workflow ready for scale-up; feasible with tiling and distributed computing, but requires scalable infrastructure for large-area mapping
Timeline	Around 1 hour/tile without taking into account collection of data/SuperResolution	-Water map : 40s per tile -Permanent/temporary disctinction : 15min for 3 years of data	- Status layer production: 2-3h/month/tile - Change layer production: 2h/tile/6 month	At 10m = approx. 1h per tile (tiles can be parallelized so it's executed in parallel) At 5m = approx. 3h per tile (tiles can be parallelized so it's executed in parallel)	10km x 10km block (single observation) with 10m resolution takes 11 openEO credits, possibly reduced to 5 credits	10km x 10km block with 10m resolution takes 12 openEO credits	the rollout of a tile takes ~1 day, could be improved.
Are you planning to transfer you workflow to openEO	No	No	No	No	Inference is in openEO	Inference and post-processing are in openEO.	
Lessons learned from the 1st round of prototype development		4 test sites might not be enough for all the water scenaries that exist.					Sentinel-2 time series with TempCNN effectively classified major tree types, but class imbalance and limited reference data for minor species reduced accuracy. Prototype validated the approach and highlighted areas for refinement
Comments/Questions							