

interTwin

D6.3 Updated report on requirements and core modules functionalities

Status: Under EC Review

Dissemination Level: public



Funded by the
European Union

Disclaimer: Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them

D6.3 Updated report on requirements and core modules functionalities

Abstract


Key Words

Core modules, Workflow management, Data fusion, Real time data Acquisition, AI, Big Data, Quality, TOSCA

interTwin co-designs and implements the prototype of an interdisciplinary Digital Twin Engine (DTE). The developed DTE will be an open source platform that includes software components for modelling and simulation to integrate application-specific Digital Twins. InterTwin WP6 will provide the core DTE modules to be integrated by WP7 and to be executed on infrastructure and components delivered by WP5.

The current document consists of a report on the high-level description of WP6 components and the related design based on the C4 model. Additionally, this deliverable includes the set of requirements derived from the analysis of the use cases and updates on the development of components after the first release in Feb 2024.



Document Description			
D6.3 Updated report on requirements and core modules functionalities			
Work Package number WP6			
Document type	Deliverable		
Document status	UNDER EC REVIEW	Version	1.0
Dissemination Level	Public		
Copyright Status	 <p>This material by Parties of the interTwin Consortium is licensed under a Creative Commons Attribution 4.0 International License.</p>		
Lead Partner	CSIC		
Document link	https://documents.egi.eu/document/3952		
DOI	https://zenodo.org/records/13709618		
Authors	<ul style="list-style-type: none"> • Isabel Campos (CSIC) • Donatello Elia (CMCC) • Germán Moltó (UPV) • Ignacio Blanquer (UPV) • Estíbaliz Parceró (UPV) • Alexander Zoechbauer (CERN) • Eric Wulff (CERN) • Matteo Bunino (CERN) • Andreas Lintermann (FZJ) • Rakesh Sarma (FZJ) • Pablo Orviz (CSIC) • Alexander Jacob (EURAC) • Sandro Fiore (UNITN) • Miguel Caballer (UPV) • Bjorn Backeberg (DELTARES) • Mariapina Castelli (EURAC) • Juraj Zvolensky (EURAC) 		



D6.3 Updated report on requirements and core modules functionalities

	<ul style="list-style-type: none"> • Iacopo Ferrario (EURAC) • Michele Claus (EURAC) • Rufai Omowunmi Balogun (EURAC) • Massimiliano Fronza (UNITN) • Andrea Manzi (EGI) • Davide Donno (CMCC) • Emanuele Donno (CMCC) • Cosimo Palazzo (CMCC) • Andrea Cristofori (EGI)
Reviewers	<ul style="list-style-type: none"> • Sandro Fiore (UNITN) • Marcin Płóciennik (PSNC)
Moderated by:	Andrea Anzanello (EGI)
Approved by	Christian Pagé (CERFACS) on behalf of TCB

Revision History			
Version	Date	Description	Contributors
V0.1	01/06/2024	ToC update from D6.1	Isabel Campos (CSIC)
v0.2	29/07/2024	Updated sections 2,3, 4, 5, 6 from D6.1	All contributing authors
v0.3	08/08/2024	Added Executive Summary and Conclusions. Sent for review.	Isabel Campos (CSIC)
v0.4	16/08/2024	Internal review	Sandro Fiore (UNITN) Marcin Płóciennik (PSNC)
v0.5	22/08/2024	Version ready for TCB review	Isabel Campos (CSIC)
v0.6	29/08/2024	TCB review	Christian Pagé (CERFACS)
v0.7	03/09/2024	Version ready for QA	Isabel Campos (CSIC)
V1.0	06/09/2024	Final	



Terminology / Acronyms	
Term/Acronym	Definition
DT	Digital Twin, a digital representation of an actual physical product, system, or process that serves as the effectively indistinguishable digital counterpart of it for practical purposes, such as simulation, integration, testing, monitoring, and maintenance
DTE	Digital Twin Engine, a platform to build DTs
CLI	Command Line Interface
GUI	Graphical User Interface
API	Application Programming Interface, aka programmatic interface of a computer system through which other computer systems can interact with it
REST API	API that conforms to the design principles of REST, or REpresentational State Transfer architectural style
SQL	Structured Query Language, a domain-specific language used in programming and designed for managing data held in relational database management systems
CI/CD	In software engineering, CI/CD is the combined practices of Continuous Integration and Continuous Delivery
AI	Artificial Intelligence
ML	Machine Learning is a branch of AI and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy

Table of Contents

1 Introduction	11
1.1 Scope	11
1.2 Document Structure	12
2 Requirements	14
2.1 Astrophysics - Noise detector DT	14
2.1.1 Description	14
2.1.2 Requirements in terms of core capabilities	14
2.2 Noise simulation for Radio Astronomy	15
2.2.1 Description	15
2.2.2 Requirements in terms of core capabilities	15
2.3 High Energy Physics - Detector simulation	16
2.3.1 Description	16
2.3.2 Requirements in terms of core capabilities	16
2.4 High Energy Physics - Lattice QCD Simulations	17
2.4.1 Description	17
2.4.2 Requirements in terms of core capabilities	17
2.5 Climate Change Future Projections of Extreme Events (storms & fire)	17
2.5.1 Description	17
2.5.2 Requirements in terms of core capabilities	18
2.6 Climate Change Impacts of Extreme Events (storms, fire, floods, drought)	18
2.6.1 Description	18
2.6.2 Requirements in terms of core capabilities	18
2.7 Early Warning for Extreme Events (floods & drought)	19
2.7.1 Description	19
2.7.2 Requirements in terms of core capabilities	19
3 Components for advanced workflow composition	21
3.1 Data acquisition & event-driven triggering of workflows	22
3.1.1 General description and functionalities	22
3.1.2 Interfaces	25
3.1.3 Technology stack	25
3.1.4 Interaction with other components	26
3.2 Workflow composition	27
3.2.1 General description and functionalities	28
3.2.2 Interfaces	29
3.2.3 Technology stack	30



D6.3 Updated report on requirements and core modules functionalities

3.2.4 Interaction with other components	35
3.3 Provenance in Workflows	35
3.4 Data Fusion	37
3.4.1 Data Fusion in Workflows	37
3.4.2 Processing Fusion in Workflows	38
3.4.3 Data Fusion for Artificial Intelligence	39
4 Components for AI workflows	40
4.1 Training module	43
4.1.1 General Description and functionalities	43
4.1.2 Interfaces	47
4.1.3 Technology stack	47
4.1.4 Interaction with other components	48
4.2 Model Registry	48
4.2.1 General description and functionalities	49
4.2.2 Interfaces	49
4.2.3 Technology stack	49
4.2.4 Interaction with other components	49
4.3 Metric Logger	50
4.3.1 General Description and functionalities	50
4.3.2 Interfaces	50
4.3.3 Technology stack	50
4.3.4 Interaction with other components	51
4.4 Machine Learning model deployment	52
4.4.1 General description and functionalities	52
4.4.2 Interfaces	54
4.4.3 Technology stack	54
4.4.4 Interaction with other components	55
5 Components for Quality Assurance	57
5.1 Software Quality Assurance as a Service	57
5.1.1 General Description and functionalities	57
5.1.2 Interfaces	59
5.1.3 Technology stack	59
5.1.4 Interaction with other components	60
6 Components for Big Data Analytics	61
6.1 Deployment of Big Data Analytics tools	61
6.1.1 General description and functionalities	61
6.1.2 Interfaces	63
6.1.3 Technology stack	63



D6.3 Updated report on requirements and core modules functionalities

6.1.4 Interaction with other components	64
6.2 Elastic Kubernetes clusters on demand	64
6.2.1 General Description and functionalities	64
6.2.2 Interfaces	64
6.2.3 Technology stack	65
6.2.4 Interaction with other components	65
6.3 Horovod environment	65
6.3.1 General Description and functionalities	65
6.3.2 Interfaces	66
6.3.3 Technology stack	66
6.3.4 Interaction with other components	66
6.4 KubeFlow clusters	66
6.4.1 General Description and functionalities	66
6.4.2 Interfaces	67
6.4.3 Technology stack	67
6.4.4 Interaction with other components	67
6.5 Ophidia Cluster	67
6.5.1 General description and functionalities	67
6.5.2 Interfaces	68
6.5.3 Technology stack	68
6.5.4 Interaction with other components	68
6.6 openEO clusters	68
6.6.1 General Description and functionalities	68
6.6.2 Interfaces	69
6.6.3 Technology stack	69
6.6.4 Interaction with other components	69
6.7 Airflow clusters	69
6.7.1 General Description and functionalities	69
6.7.2 Interfaces	69
6.7.3 Technology stack	69
6.7.4 Interaction with other components	69
6.8 EOEPKA ADES clusters	70
6.8.1 General Description and functionalities	70
6.8.2 Interfaces	70
6.8.3 Technology stack	70
6.8.4 Interaction with other components	70
6.9 ecFlow clusters	70
6.9.1 General Description and functionalities	70



D6.3 Updated report on requirements and core modules functionalities

6.9.2 Interfaces	71
6.9.3 Technology stack	71
6.9.4 Interaction with other components	71
6.10 MLFlow server	71
6.10.1 General Description and functionalities	71
6.10.2 Interfaces	71
6.10.3 Technology stack	71
6.10.4 Interaction with other components	71
6.11 yProv server	71
6.11.1 General Description and functionalities	71
6.11.2 Interfaces	72
6.11.3 Technology stack	72
6.11.4 Interaction with other components	72
7 Conclusions	73
8 References	74

Table of Figures

Figure 1 - General overview of the core components and related functionality	12
Figure 2 - General architecture for data ingestion and event-driven triggering of workflows	23
Figure 3 - Integration of OSCAR, interLink, and itwinai for scalable event-driven processing of AI workloads across Cloud and HPC	26
Figure 4 - Example of workflow composition for EO applications.	29
Figure 5 - High-level schematic view of the use of the PyOphidia module for generating and managing the provenance document of a workflow.	36
Figure 6 - Example OpenEO processing fusion	39
Figure 7 - Detailed view on the architecture of the ML component	42
Figure 8 - Detailed view on the architecture of ML training module	44
Figure 9 - Kubeflow-based example of an illustrative AI pipeline	46
Figure 10 - Detailed view on the architecture of the ML deployment module	53
Figure 11 - Detailed view on the architecture of the quality assurance module	59
Figure 12 - General architecture of the data analytics tools	62



Executive summary

This deliverable updates the previous D6.1 (*Report on requirements and core modules definition*). For completeness it also provides the overview of the global requirements that the target users of a Digital Twin Engine (DTE) have identified. In D6.1 these requirements were matched with the core modules of the solutions that can fulfil those requirements in a first approximation.

This document describes the developments that took place in the course of the last year towards refining and extending the set of core modules to build the DTE. Several uncertainties and pending technical decisions have been made in the light of the use cases, technological evolution and integration opportunities. Overall efforts have been made towards a standardisation and improvement of the interoperability properties of the core modules.

- Regarding **advanced workflow composition** the project moved in the direction of exploiting the CWL standard as a common ground. Interfacing with popular and community workflow managers such as Apache Airflow, Streamflow or ecFlow has been a priority. The usage of the openEO API for Earth Observation using CWL as standard in the workflow made substantial progress in the last year. The effort to integrate community solutions such as the execution and workflow manager engine Ophidia needs to be highlighted, for instance a python-based client for Ophidia has been developed including support for the tasks descriptions to be written in CWL format.
- **Real-time data acquisition and event-driven triggering of workflows** is one of the most advanced and complex stacks developed in interTwin. Our middleware stack is able to detect when new data arrives, perform data stage and pre-processing, including quality control, and finally delegate into an external workflow management system that will enact the execution on resources dynamically provisioned on Clouds or on HPC-based infrastructure.
- We have also tackled the general issue of **provenance in workflow execution**, integrating the yProv service with the general framework of WP6. This relies on the W3C PROV family of standards for its information model. The project is testing the yProv functionality in climate analytics workflows and HEP pipelines.
- The core components to support **AI workflows** are one of the project priorities. In this context the itwinai virtual environment has been greatly enhanced during the last year including features such as Hyperparameter Optimization and support to distributed training in Cloud and HPC infrastructures. The framework enables the DT developer to define modular AI workflows, fostering code reuse and streamlining development.
- The **Quality Assurance** component is evolving to support requirements of data quality evaluation in data spaces, and support the inclusion of QA in workflow composition for **model validation**. For instance it has been enhanced with capabilities to include specific criteria, such as AI/ML validation. The Quality and uncertainty tracing module allows testing (micro)services in a black-box manner. For instance, once a pre-trained ML model is deployed as a container, it can be



D6.3 Updated report on requirements and core modules functionalities

easily tested, and the DT developer can provide user case-specific cases to validate the performance of the model.

- Finally at the deployment layer the **big data analytics tools** are designed as a deployment layer using OASIS TOSCA templates to describe the virtual resources and software components needed to deploy data analytics applications. Kubernetes clusters, Horovod environments, Kubeflow, and Ophidia, Airflow, ADES, ecFlow, MLFlow and yProv are among the tools provided in the deployment layer. All the templates are stored in a public repository, accessible to end users via the WP5 tools (PaaS orchestrator dashboard). The recipes needed to install/configure every software component are stored in a set of public repositories using Ansible language. In the context of WP5, the PaaS orchestrator by means of the Infrastructure Manager interacts with the computing infrastructure providers to deploy the virtual resources as needed.

All the WP6 components delegate its execution to backends deployed via the interfaces and APIs made available by WP5.

1 Introduction

1.1 Scope

The general objective of the WP6 is enhancing Digital Twins with horizontal capabilities for exploiting generic workflows able to link observational and model data.

Progressing towards this goal requires tackling a number of technological challenges. For example, we need to expand the paradigm of “serverless” computing to Digital Twins by implementing a generic framework for real-time data acquisition and processing that builds on event-triggered execution of workflows. This is a generic capability required by (most) Digital Twins aiming at performing automated validation of models using real-time observational data.

The usage of Artificial Intelligence techniques such as Machine Learning, for a variety of purposes (from optimisation of simulations to model building) requires a significant push on distributed training and the related advanced optimization (such as Hyper Parameter Optimization). A generic framework needs to be devised to plug in ML models and data pipelines that can be interfaced to the computing and data resources in the backend.

In turn, the computing and data resources need to support a number of components and best practices to efficiently run data analytics. In the framework of WP6, this implies the implementation of recipes for general-purpose data analytic environments to be deployed on demand on top of the Cloud resources, or the provision of a seamless HPC and Cloud resources interface with container workload management services that are able to interact with HPC resources. One of the main challenges (and source of innovation) is on the “on-demand” provisioning, horizontal scaling and integration of the workflow mechanisms.



D6.3 Updated report on requirements and core modules functionalities

A general model quality validation strategy needs to be developed as well, to enhance Digital Twins with the capability to implement best practices and standard quality measures to support model validation. The work here is inspired by DevOps practices, to exploit automation, Continuous Integration and Delivery (CI/CD) to create a comprehensive environment for model quality assessment and validation, together with the evaluation of FAIR data quality integrated, in the pipeline for observed and simulated data. The final objective here is the implementation of Model Quality Validation “as a Service”.

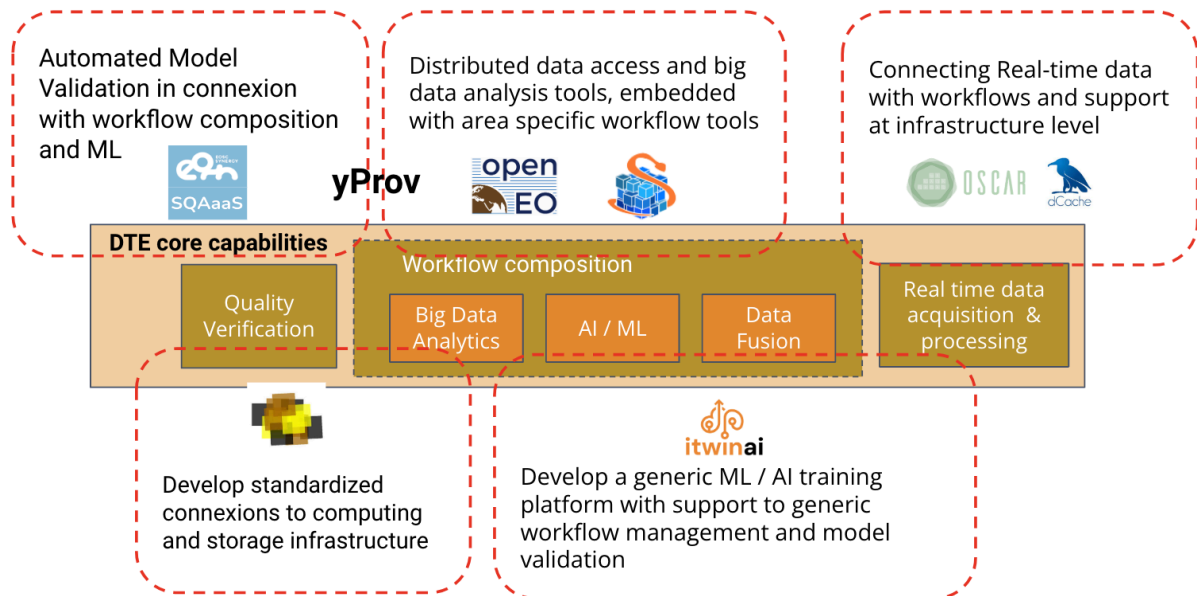


Figure 1 - General overview of the core components and related functionality

1.2 Document Structure

This document is an updated version of the deliverable D6.1 [6] **Report on requirements and core modules definition**. Most of the information are therefore common to the two documents with the following sections updated:

- Section 3.1 has been updated with the integration performed;
- Section 3.2 has been extended to include ADES and remove Delf-FEWS;
- Section 3.3 has been extended with the latest design;
- Section 3.4 has been rewritten;
- Section 4 has been heavily upgraded;
- Section 5 has been extended with uncertainty quantification component;
- Section 6 has been extended with new TOSCA templates developed in the second part of the project.

This deliverable is structured as follows. [Section 2](#) contains a very high level summary of the main applications used to derive the requirements and a list of the required functionalities that fall in the scope of “core” services; [Section 3](#) describes the core components that will be implemented for advanced workflow composition and the components that deliver data fusion capabilities; [Section 4](#) describes the architecture

D6.3 Updated report on requirements and core modules functionalities

and components related to Artificial Intelligence workflows; [Section 5](#) describes the core components dedicated to support quality assurances; [Section 6](#) describes the components dedicated to support big data analytics.



2 Requirements

2.1 Astrophysics - Noise detector DT

2.1.1 Description

The discovery of gravitational waves was one of the main scientific results in recent years and was awarded the Nobel Prize in 2017, opening a new era in the study of the cosmos and paved the way to multi-messenger astronomy. Besides getting ready for their next observation runs, the gravitational-wave community is designing a next-generation observatory, the Einstein Telescope, which was recently included in the EU ESFRI roadmap.

The sensitivity of Gravitational Waves interferometers is limited by noise; its reduction and subtraction are one of the most important and challenging activities in this research area. The Digital Twin of an interferometer is meant to realistically simulate the noise in the detector, in order to study how it reacts to external disturbances and, in the perspective of the Einstein Telescope, to be able to detect noise “glitches” in quasi-real time, which is currently not possible. This will allow the low-latency search pipelines to veto or de-noise the signal, sending out more reliable triggers to observatories for multi-messenger astronomy.

2.1.2 Requirements in terms of core capabilities

- **Notebook platform:** JupyterLab is currently used. Notebooks will be used during R&D, but the DT pipeline should have a microservices architecture.
- **ML architecture:** Generative Adversarial Networks. Particular attention will need to be paid to the training stability, especially regarding the with regard to online learning.
- **ML dataset:** 2D images (frequency-time heatmap). Size is in the order of 10s of TB.
- **ML framework:** Pytorch and Tensorflow should be made available in the Notebooks and in the pipeline environment (Linux containers).
- **Distributed ML:** For training heavy models or large datasets on HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) is necessary.
- **Streaming platform:** the detector pushes data using Apache Kafka:
 - input consists of about 50 AUX channels for a total of 2 MB/s plus the Strain channel at 160 kB/s;
 - output is a stream of denoised data to low-latency search pipelines (year 2), rate comparable to the strain channel (160 kB/s).
- **ML model storage:** only the most recent trained model (and the previous one for rollbacks) is needed by the DT pipeline, most likely on a POSIX filesystem. A database or online service for retrieval of model history will be beneficial for offline analysis.



D6.3 Updated report on requirements and core modules functionalities

- **ML monitoring tools:** plan to use Tensorboard to monitor the behaviour of the training process (accuracy, convergence...). Notifications to the DT operator would be interesting to have, as well as monitor processing in general (e.g., Prometheus + Grafana).
- **Event-based platform:** trigger retraining when experimental conditions (detector) change and the simulated data start to deviate from the input stream.
- **User notifications:** user intervention might be needed when the conditions for re-training are met. Would be interesting having online notifications (not just emails).

2.2 Noise simulation for Radio Astronomy

2.2.1 Description

The Digital Twin of the MeerKAT radio telescope is meant to better separate the background signals from the (usually weak) astrophysical effects, especially considering the ever-increasing amount of data streams delivered by the antennas. A major challenge is to identify the large amount of man-made noise signals (mostly from satellites and mobiles). Radio Astronomy sensors collect increasing amounts of data requiring massive parallel computing both in the online and offline phases. This requires dynamic filtering of data in a real-time manner via ML from huge data streams amounting to several Petabytes per day, and a feedback loop from analysis to sensor control.

The above goals will be achieved by harmonisation of real-time and near-real-time streams, such as earth observation data or radio telescopes with AI and ML modelling workflows, and effective detection of noise signals in real-time with continuous training of ML algorithms for immediate and optimised control of the physical twin.

2.2.2 Requirements in terms of core capabilities

- **Tools to transfer input data:** we need to push and/or pull to/from an archive (of the observatory that operates the telescope or other institutes interested in the data), which differs for each telescope and project. Given the size of the data (hundreds of TBs) it would be beneficial having UDP-based protocols to transfer.
- **ML architecture:** convolutional/recurrent Neural Networks, long short-term memory networks, neural ODE (computational speed needs to be evaluated in case of online learning)
- **ML dataset:** 2D images (frequency-time heatmap). Size is in the order of 10s of TB.
- **ML framework:** tensorflow should be made available in the Notebooks and in the pipeline environment (Linux containers).
- **Distributed ML:** for training heavy models or on large datasets using HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) is necessary.



D6.3 Updated report on requirements and core modules functionalities

- For the future we aim at implementing in a more computationally efficient way (e.g., C++) using distributed training on a HPC cluster (multiple GPUs at once, e.g., as the model does not fit on a single GPU). An approach to multithreaded computation will be needed, such as MPI or in-memory frameworks such as Apache Spark.

2.3 High Energy Physics - Detector simulation

2.3.1 Description

Particle detectors measure different particle properties when interacting with the materials that the detectors consist of, at the Large Hadron Collider (LHC) experiments. More specifically, the detectors called calorimeters are key parts of the detectors' set up, which are responsible for measuring the energy of the particles. In a collider, the emerging particles travel through the detector and interact with the materials either electromagnetically or hadronically. During this interaction cascades of secondary particles are created. The modelling of these matter interactions is performed by Monte Carlo calculations, which in order to produce an instance of the interactions of a particle with the detector, depend on repeated random sampling. These simulations have a crucial role in High Energy Physics (HEP) experiments, and at the same time are slow and resource intensive.

Therefore, there is a need for faster simulations. The main motivation for fast simulations is to incorporate other faster alternative simulation techniques. For this purpose, machine learning has been utilised as a fast simulation technique to speed up detector simulations. Our use case leverages a variant of a GAN developed for HEP applications called 3DGAN, where the detector output was generated employing three dimensional convolutions, a powerful approach for retaining correlations in all three spatial dimensions.

2.3.2 Requirements in terms of core capabilities

The use case aims at comparing simulated data generated with the ML model with previously generated based simulated data using MonteCarlo methods (by GEANT4). The format file is different (HDF5 vs ROOT) but the accompanying metadata are the same. CI/CD pipelines to automate the comparison and check the integrity of the metadata can be interesting.

The current case is meant as a static synthetic model of a detector. We could think of extending this to an application capable of modelling in real time the behaviour of a detector in different operation conditions (beams and accelerator configurations) and therefore include continuous retraining on real data. That would require event-driven execution.

- **ML architecture:** 3D Generative Adversarial Networks. The envisioned model will have ~5 million parameters, needing at least 8 GPUs. Computational costs can be high; therefore attention needs to be paid to sufficient per-GPU performance next to the number of total GPUs available.



D6.3 Updated report on requirements and core modules functionalities

- **ML dataset:** 3D images (energy deposition in 3D). Size is in the order of 10s of GB.
- **ML framework:** Pytorch and Tensorflow should be made available in the Notebooks and in the pipeline environment (Linux containers).
- **Distributed ML:** for training heavy models or large datasets on HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) is necessary.

2.4 High Energy Physics - Lattice QCD Simulations

2.4.1 Description

The aim of Lattice QCD is shedding light on the properties of Quantum Chromodynamics in the limit of low energies/strong couplings, where perturbation theory breaks down, and numerical approaches become mandatory. In interTwin are exploring two use cases addressing the status of Lattice QCD simulations: a classical scenario, with large scale simulations in HPC; and a second scenario, Machine Learning-based simulations, an area under development in the community, at the proof of concept level, therefore requiring few resources.

2.4.2 Requirements in terms of core capabilities

- HPC simulations require access to HPC resources with Infiniband. Data sharing in a Data Lake requires mainly infrastructure services from WP5. In order to connect to the services, Jupyter notebooks are becoming a useful tool.
- For the Machine Learning based simulations the requirements in terms of tools are ML libraries such as Tensorflow, Pytorch, etc. CI/CD pipelines to automate the checking of the convergence of the simulations towards the target acceptance rate.

2.5 Climate Change Future Projections of Extreme Events (storms & fire)

2.5.1 Description

This Digital Twin application is related to the prediction of Extreme Weather Events (EWEs), in particular storms and fires, in future projection scenarios (e.g., CMIP6) with the aim of giving an indication about the temporal trend and the geographical occurrence of such events across the globe due to climate change. ML models (e.g., Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs)) that learn the underlying mapping between drivers and the historical occurrence of such EWEs, will be adopted. The data-driven products generated can generalise to future projection data, where a strong signal of climate change is evident.

As the use cases are data-driven, besides the actual ML models and the experimental setup, pre-processing pipelines will play a relevant role in order to prepare the data for the training and inference phases. What-if scenarios will be made available for the end-users (e.g., climate scientists) for highlighting relevant changes of such EWEs in future projections.



2.5.2 Requirements in terms of core capabilities

- **Workflow composition:** support for execution of large-scale data cube-based workflows, parallel multi-model workflows and workflows integrating community-based climate tools (e.g., Ophidia).
- **Provenance:** climate workflows should be also documented in terms of provenance metadata to enable full tracking of lineage information.
- **ML architecture:** Convolutional Neural Networks, Graph Convolutional Networks.
- **ML dataset:** up to 4D climate data. Overall size is at most 1TB.
- **ML framework:** PyTorch, PyTorch Lightning and PyTorch Geometric should be made available in the Notebooks and in the pipeline environment (Linux containers).
- **Distributed ML:** for training heavy models or on large datasets on HPC resources a distributed Training Framework (e.g., PyTorch, PyTorch Lightning, Horovod) is necessary.
- **CI/CD validation pipelines:** for trained ML models.

2.6 Climate Change Impacts of Extreme Events (storms, fire, floods, drought)

2.6.1 Description

The Digital Twin application for Climate Change Impacts of Extreme Events on Floods (hereafter referred to as FloodAdapt Climate Change Impact) uses the same process-based models as the Digital Twin application for Flood Early Warning in coastal and inland regions ([Section 2.7](#)). The main addition is that in FloodAdapt Climate Change Impact, end-users (e.g., decision makers) can define scenarios such as building a dam wall or doubling the rainfall, that modify the input data for the process-based models. These changes are managed by the FloodAdapt backend.

2.6.2 Requirements in terms of core capabilities

- **Workflow composition:** reuse workflows already developed by CERFACS, Deltares, and EURAC. This will need a workflow backend that will be able to call a mix of processes and sub-workflows involving amongst others openEO, FloodAdapt, StreamFlow and ecFLOW.
- **Data fusion:** generic data fusion components combined with custom Python scripts for preprocessing forcing and boundary condition data.
- **Container workflow management** for running containerised models on heterogeneous computing infrastructures.
- **Batch queue system** for running models at scale on HPC/HTC infrastructures.
- Data is to be queried and processed via the **openEO syntax**. That includes a backend with openEO interface, geodata in an openEO compatible format



D6.3 Updated report on requirements and core modules functionalities

- Possibility to store intermediate and final workflow results at the cloud provider.
- Support for model sharing via **Jupyter notebooks** or **docker containers**.
- **ML framework:** Tensorflow should be made available in the Notebooks and in the pipeline environment (Linux containers).
- **ML architecture:** Kmeans, Clustering
- **ML Dataset:** 4D satellite time series. Size is in the order of 100s of GB. Update frequency is 5-10 years.
- **Distributed ML:** For training heavy models or on large datasets on HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) is necessary.

2.7 Early Warning for Extreme Events (floods & drought)

2.7.1 Description

For Early Warning for Extreme Events, two Digital Twin applications are planned: (1) Flood Early Warning in coastal and inland regions (hereafter referred to as FloodAdapt Early Warning), and (2) Drought Early Warning in alpine regions. Process-based models will be combined with ML and Deep Learning models using Earth Observation data to support early warning of floods and droughts. This includes developments towards globally relocatable models which require data pre-processing pipelines similar to those needed by ML and Deep Learning models.

2.7.2 Requirements in terms of core capabilities

- **Workflow composition:** reuse workflows already developed by Deltares, EURAC, and TUW. This will need a workflow backend that will be able to call a mix of openEO, FloodAdapt, and ecFLOW sub-workflows.
- **Container workflow management** for running containerised models on heterogeneous computing infrastructures.
- **Data fusion:** generic data fusion components combined with custom Python scripts for preprocessing forcing and boundary condition data.
- **Batch queue system** for running models at scale on HPC/HTC infrastructures.
- **FAIR data quality evaluation** to assess FAIRness of output from process-based and data-driven models.
- **ML framework:** Tensorflow should be made available in the Notebooks and in the pipeline environment (Linux containers). PyTorch to implement the model architecture. Tensorboard for training analysis.
- **Distributed ML:** for training heavy models or on large datasets on HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) is necessary. Also, Dask would be useful for distributed hyper parameter tuning. NVIDIA GPU (with the NVIDIA driver version 510.85.02 and CUDA version 11.6).



D6.3 Updated report on requirements and core modules functionalities

- **ML architecture** (type of model): Recurrent Neural Networks (Long Short-Term Memory networks and/or Gated Recurrent Unit), encoding/decoding using Convolutional Neural Networks (for climate data downscaling).
- **ML Dataset:** 4D satellite time series. Climate data (Copernicus European Regional Reanalysis, System 5 seasonal forecasts). Size is under investigation.



3 Components for advanced workflow composition

One of the most important capabilities of a Digital Twin Engine (DTE) is the ability to organise and run a chained list of data acquisition and/or processing steps into what is known as a workflow. At every stage in a workflow, the current step has a connection to the next one, which means data of earlier steps process to the next one. Workflows may have steps that run in parallel.

Workflow orchestration is the process of configuring, managing, and coordinating tasks automatically, which may contain different disparate systems.

The DTE to be built as a result of the interTwin project, and which can be used both by expert users (DTE developers) and regular end users (Scientists), is using the following approach:

- **Create/Reuse an intuitive User Interface (UI):** develop a user-friendly, web-based UI that allows users to create, modify, and manage workflows without needing to understand the underlying complexity of the workflow orchestration solutions. The UI should provide drag-and-drop functionality, visual representation of tasks, and easy configuration of task parameters. Some steps that are part of the workflow could expose specific UI's or APIs to the DTE expert users (e.g., to configure specific aspects of a DTE step, to collect results of a specific task, etc.).
- **Create a programmatic interface** (aka an Application Programming Interface, or API): define a programmatic interface for the DTE, as a contract between DTE implementers and DTE users, that allows the DTE to be triggered by other software components, most notably by data-related events.
- **Adopt a top-level workflow orchestration technology:** reusing existing work as steps in a DTE's workflow is a crucial capability of the DTE. The challenge is that most of this work is in the form of existing workflows, based on different workflow composition and/or workflow execution frameworks. Therefore, picking any single workflow orchestration and execution technology for the interTwin DTE would mean asking researchers to redo most of this work. At the same time, the top-level entry point into the DTE should be well defined. Pick a single workflow description language, and workflow orchestration tool that supports that language, which can then accommodate existing research work/tools as is, as sub-workflows.
- **Abstract workflow orchestration solutions:** develop a set of reusable components or modules that encapsulate the functionality of the various workflow orchestration solutions supported by the project. These components should provide a consistent interface for interacting with the underlying systems, making it easier for users to incorporate different workflow solutions without needing to understand their specific implementation details. Additional



D6.3 Updated report on requirements and core modules functionalities

components, supporting additional workflow orchestration frameworks can be added in the future, as needed.

- **Provide pre-built templates:** offer a collection of pre-built workflow templates that cover common use cases and can be easily customised by users. These templates should include examples of how to call complex sub-workflows from different workflow orchestration solutions and provide best practices for structuring and organising workflows.
- **Ensure scalability and flexibility:** design both the high-level entry point, and the components that can be used as steps in this top-level workflow, to be scalable and flexible, allowing users to create workflows that can grow in complexity and size over time. This may involve providing support for parallel execution, distributed computing environments, and cloud platforms.
- **Offer documentation and support:** provide comprehensive documentation, tutorials, and support resources to help users get started with the high-level entry point and learn how to create and manage workflows. This should include guides for working with different workflow orchestration solutions and examples of how to call complex workflows.

By following the approach described above, we can create a high-level entry point that makes it easy for users with different expertise levels to compose workflows using various workflow orchestration solutions. This will enable users to focus on the high-level logic and structure of their workflows, without needing to understand the intricacies of the underlying systems.

Data acquisition and how to kick off the DTE workflow in reaction to data-driven events is described in [Section 3.1](#). The solutions adopted for DTE workflow composition are covered by [Section 3.2](#), while provenance in workflows is addressed by [Section 3.3](#) and data fusion by [Section 3.4](#).

3.1 Data acquisition & event-driven triggering of workflows

This component offers a generic framework for real-time data acquisition and processing that builds on event-triggered execution of workflow engines.

3.1.1 General description and functionalities

The real-time data acquisition and processing framework for the DTE that supports event-triggered execution of workflow engines satisfy the following requirements: i) detects when new data that requires processing is made available; ii) performs data stage and pre-processing (e.g., to perform data cleansing or data quality assessment) and iii) delegates the complex data processing into external workflow management systems in charge of enacting the execution on resources that are dynamically provisioned from a Cloud or HPC based infrastructure.



D6.3 Updated report on requirements and core modules functionalities

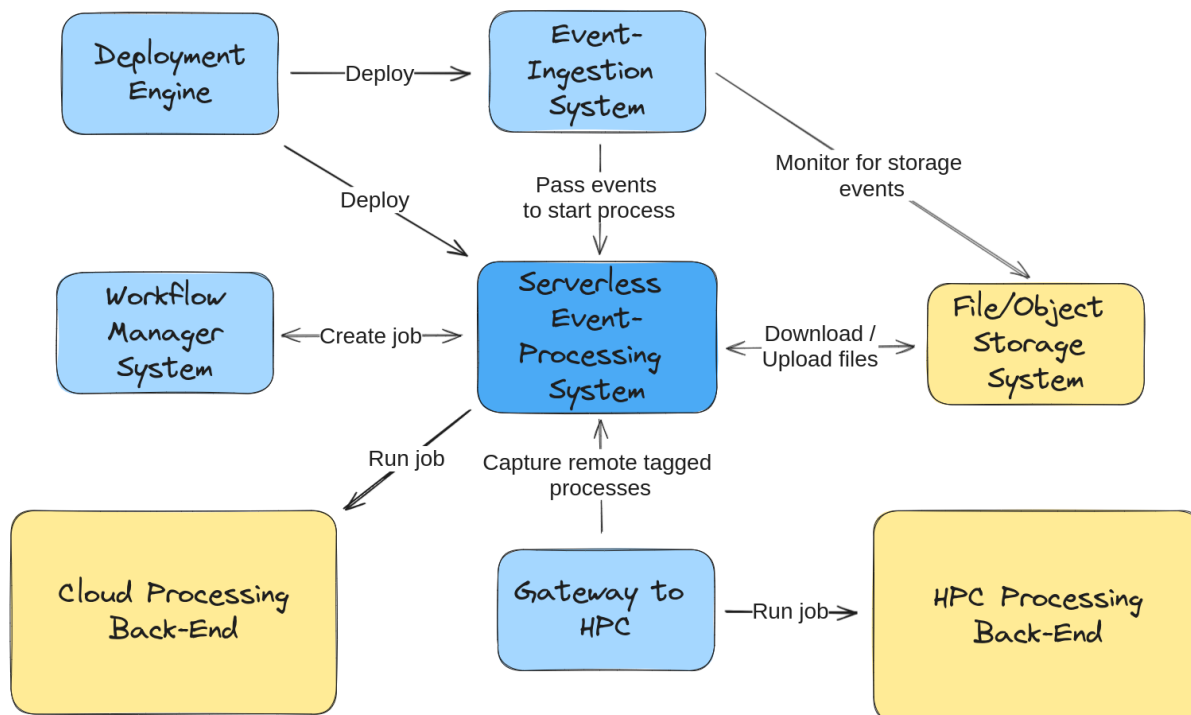


Figure 2 - General architecture for data ingestion and event-driven triggering of workflows

To this aim, **Figure 2** shows the overall architecture for data acquisition and event-driven triggering of workflows, including the high-level components described below. Yellow boxes represent infrastructure-level components for storage and processing, while blue boxes represent software/service components.

Serverless Event-Processing System

The serverless event-processing system is in charge of receiving data pre-processing requests from the event-ingestion system to perform additional data transformations that may not be performed within the event-ingestion system itself. This can be due to a lack of support for certain operations or the dependency on external tools that may be packaged as Docker images, which may not be able to run directly within the event-ingestion system.

In order to address the challenges and limitations of the event-ingestion systems, we adopted the OSCAR serverless event-processing system since it leverages the high scalability provided by elastic Kubernetes clusters, which are dynamically provisioned via the Infrastructure Manager on multi-Clouds and uses the CLUES elasticity manager to achieve two-level elasticity (in terms of the number of pods and the number of nodes) and provide efficient execution of data-processing requests.

OSCAR also leverages Knative under the hood to support low-latency synchronous requests. OSCAR services are triggered in response to events and execute user-defined scripts on dynamically provisioned containers out of user-defined Docker images, thus facilitating the integration with external workflow engines where the actual complex processing can take place.



D6.3 Updated report on requirements and core modules functionalities

File/Object Storage System

The file/object-storage system provides a solution for the storage of data to be analysed, whether it is temporary or long-term storage, as is the case of data lakes. For fault-tolerance and high-availability reasons, the data is typically stored in a distributed approach among a large number of heterogeneous servers while providing a unified vision of a virtual filesystem that can be accessed through a variety of protocols.

The serverless event-processing system OSCAR supports several storage providers like MinIO, Amazon S3, OneData, and WebDAV storage providers (e.g., OpenStack's Swift, NextCloud).

The framework now includes support for dCache, a file storage system known for managing large-scale scientific data across multiple nodes. This integration has brought several advantages, including improved data availability due to dCache's ability to replicate data across multiple storage nodes. Also, the ability to detect new data uploaded in dCache to trigger its data processing within an OSCAR cluster.

Event-Ingestion System

The event-ingestion system is responsible for receiving the notification events from the file/object-storage system or data source and provides the ability to execute simple transformation data flows using the built-in components supported by the system.

In the development of this system, we have employed Apache NiFi, since it allows us to craft flows that will take data from a large variety of different sources, enrich the data, and route it to several destinations.

The benefits of this event-ingestion system lie, among others, in decoupling the rate at which files can be uploaded to the file-storage system to the one used for data processing. This acts as an infinite message queue that elastically grows to manage the asymmetry between data producer and consumer.

We have integrated this event-ingestion system with several data sources to leverage the benefits of Apache NiFi, like Amazon S3 and dCache. In the case of dCache, we have successfully incorporated it as a source for events through a client for the SSE (Server-Sent Events) support in our event-ingestion system.

We also integrated Apache Kafka, a Publish/Subscribe system and a distributed event streaming platform known for its high-throughput, fault-tolerance, and durability. In our setup, the event-ingestion system consumes data from a specific Kafka topic and it's stored in an internal buffer for its processing. When the buffer is full, the data is sent to an object storage system (e.g., MinIO) which triggers the execution of an OSCAR service. This allows OSCAR to process data close to real-time as it is delivered, enhancing the responsiveness and efficiency of our system.

This multi-source approach not only diversifies our data intake but also increases the flexibility of our system. It allows us to handle a wide range of use cases and adapt to various data environments.



D6.3 Updated report on requirements and core modules functionalities

DCNiOS

DCNiOS¹ is an open-source Data Connector for Apache NiFi and OSCAR. DCNiOS has been developed to facilitate the deployment of dataflows to achieve integration between a source of events like a storage system (such as Kafka or dCache) and OSCAR Services.

This tool allows users to set up dataflows using simple YAML configuration files. These files detail data sources and destination endpoints together with intermediate steps processes.

DCNiOS also comes with a command-line interface (CLI). This feature enables us to deploy and adjust the NiFi flow at runtime. For instance, we can change the data processing rate, making our system adaptable to varying needs.

3.1.2 Interfaces

The interfaces vary for the different subcomponents, as follows:

- **File/Object-Storage System:** these systems typically support several protocols and interfaces for file uploading/downloading. For example, dCache supports FTP (File Transfer Protocol), NFS (Network File System), and WebDAV (Web Distributed Authoring and Versioning), an extension of HTTP (Hypertext Transfer Protocol).
- **Event-Ingestion System:** these components support several specifications to comply with the Publish/Subscribe (Pub/Sub) approach to gather the events. In particular, Apache NiFi relies on HTTP to create an SSE client and the corresponding subscription into dCache in order to receive the file upload events into the file-storage system.
- **Serverless Event-Processing System:** these systems rely on HTTP-based requests and events from the underlying object-storage system. For example, OSCAR allows receiving events from MinIO buckets to perform event-driven processing. It can also receive triggering requests via HTTP into the OSCAR Manager's API.

3.1.3 Technology stack

The technology stack to support the deployment and execution of this component is:

- A **Cloud Management Platform** (e.g., OpenStack or OpenNebula) or a public or federated Cloud on which to perform the automated provisioning of these components via the PaaS Orchestrator and the Infrastructure Manager (IM).
- The **PaaS Orchestrator**, an open-source TOSCA-based engine to provide Cloud resource selection, and infrastructure provision and customization through the Infrastructure Manager.

¹ <https://github.com/interTwin-eu/dcnios>



D6.3 Updated report on requirements and core modules functionalities

- The **Infrastructure Manager**, an Infrastructure as Code (IaC) tool to support the deployment of TOSCA-based description of complex application architectures (e.g., Kubernetes clusters) on multiple Cloud back-ends.
- **Kubernetes**, a container orchestration platform which allows to coordinate the execution of multiple container-based services on a distributed infrastructure. This will facilitate the automated deployment of the subcomponents in an automated fashion.
- A **Container Registry** (e.g., Docker Hub, GitHub Container Registry) to host the Docker images required to deploy the subcomponents within a Kubernetes cluster.

Of course, this component also involves the corresponding packages to provision Apache NiFi, OSCAR clusters, and the corresponding workflow management solution.

3.1.4 Interaction with other components

These components provide the ability to trigger the execution of workflows upon certain file events in an external storage system. There are also additional integrations carried out with other tools in the ecosystem, described as follows and shown in the following figure:

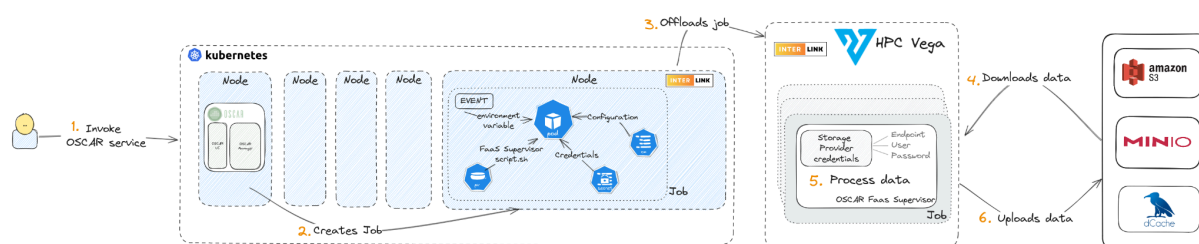


Figure 3 - Integration of OSCAR, interLink, and itwinai for scalable event-driven processing of AI workloads across Cloud and HPC

OSCAR-interLink Integration

OSCAR was integrated with interLink, a framework that allows the execution of a Kubernetes pod on any remote resource capable of managing a Container execution lifecycle. This allowed OSCAR jobs to be offloaded into an external HPC cluster through interLink, where additional scarce computing resources, such as GPUs, may be available.

OSCAR-itwinai Integration

We successfully integrated OSCAR and itwinai, an open-source Python library designed to compose and run ML workflows. This integration allows for efficient execution of inference AI pipelines in response to incoming data events, enabling real-time predictions on new samples. Furthermore, the ability to offload tasks to remote HPC systems for accelerated predictions on GPU hardware enhances the speed and efficiency of ML tasks.



D6.3 Updated report on requirements and core modules functionalities

OSCAR-Jupyter Integration

We provide the ability to deploy Jupyter Notebooks in an OSCAR cluster with automatic mounting of the object-storage (e.g., MinIO or dCache) to facilitate data processing from a visual environment which can be customised by the user. This facilitates the post-processing phase after the data ingestion and processing done inside the OSCAR cluster, to perform further interactive analysis.

The workflow composition and workflow enactment and execution will be described in [Section 3.2](#).

3.2 Workflow composition

The workflow composition subsystem is devoted to facilitating the definition of complex workflows. The challenge is to support the definition of workflows whose steps might already have been implemented or which require specific workflow engines. This need has emerged from the requirements analysis included in [Section 2](#), therefore it is important to allow reuse of existing workflows as building blocks (sub-workflows) in the DTE's workflow.

The workflow composition solution adopted for the interTwin DTE, as described in [Section 3](#), is to have a high-level entry point for users to compose workflows that in turn can call complex sub-workflows built with any of the workflow orchestration technologies supported by the project. The adopted high-level entry point should have both an easy-to-use UI, as well as an API to trigger the workflow. The subsystem should also support the execution of Workflows directly from one of the Sub-workflow engines as needed by the DT Application developers.

When choosing the technology to be adopted for the top-level workflow of the DTE, openEO² process graphs and Common Workflow Language (CWL)³ were considered as possible options. It was debated if multiple choices should be offered, and the consensus was that this would complicate the DTE architecture and implementation, without adding much benefit, as either solution still allows for the flexibility needed to reuse existing work as sub-workflows with different technological stacks. CWL was selected as the language to describe the top-level workflow of the interTwin DTE, on these considerations:

- OpenEO is primarily designed for Earth observation data processing and would need to be adapted and extended with additional concepts to accommodate the broader scope of workflows in a general purpose DTE.
- CWL supports open consensus-based standards for command line data analysis workflows and tools, and also provides a reference implementation (the cwltool⁴) which can be used to describe portable and reusable workflows.

² <https://openeo.org>

³ <https://www.commonwl.org>

⁴ <https://github.com/common-workflow-language/cwltool>



D6.3 Updated report on requirements and core modules functionalities

- CWL has gained much traction and is currently widely supported by popular workflow management systems and engines such as StreamFlow⁵ or Apache Airflow (CWL-Airflow⁶ in particular).

3.2.1 General description and functionalities

For the purpose of defining the top-level DTE workflow the usage of CWL as a glue to execute isolated workflow steps which eventually will run in different workflow engine backends is envisaged. Please note that workflow composition is highly dependent on the application at hand, therefore the most sensible approach is adopting a neutral standard such as CWL as the requirement for the general architecture of the DTE.

When selecting the software to orchestrate and execute the top-level CWL-based workflows of the interTwin DTE, multiple platforms that include CWL support were analysed, including Streamflow^{7 a} and Apache Airflow⁸, others like Ophidia have already started supporting CWL as an activity in the project.

An example of EO workflows defined with CWL with subworkflows is shown in Figure 2.

⁵ <https://streamflow.di.unito.it/>

⁶ <https://cwl-airflow.readthedocs.io/en/latest/>

⁷ <https://streamflow.di.unito.it/>

⁸ <https://airflow.apache.org>



D6.3 Updated report on requirements and core modules functionalities

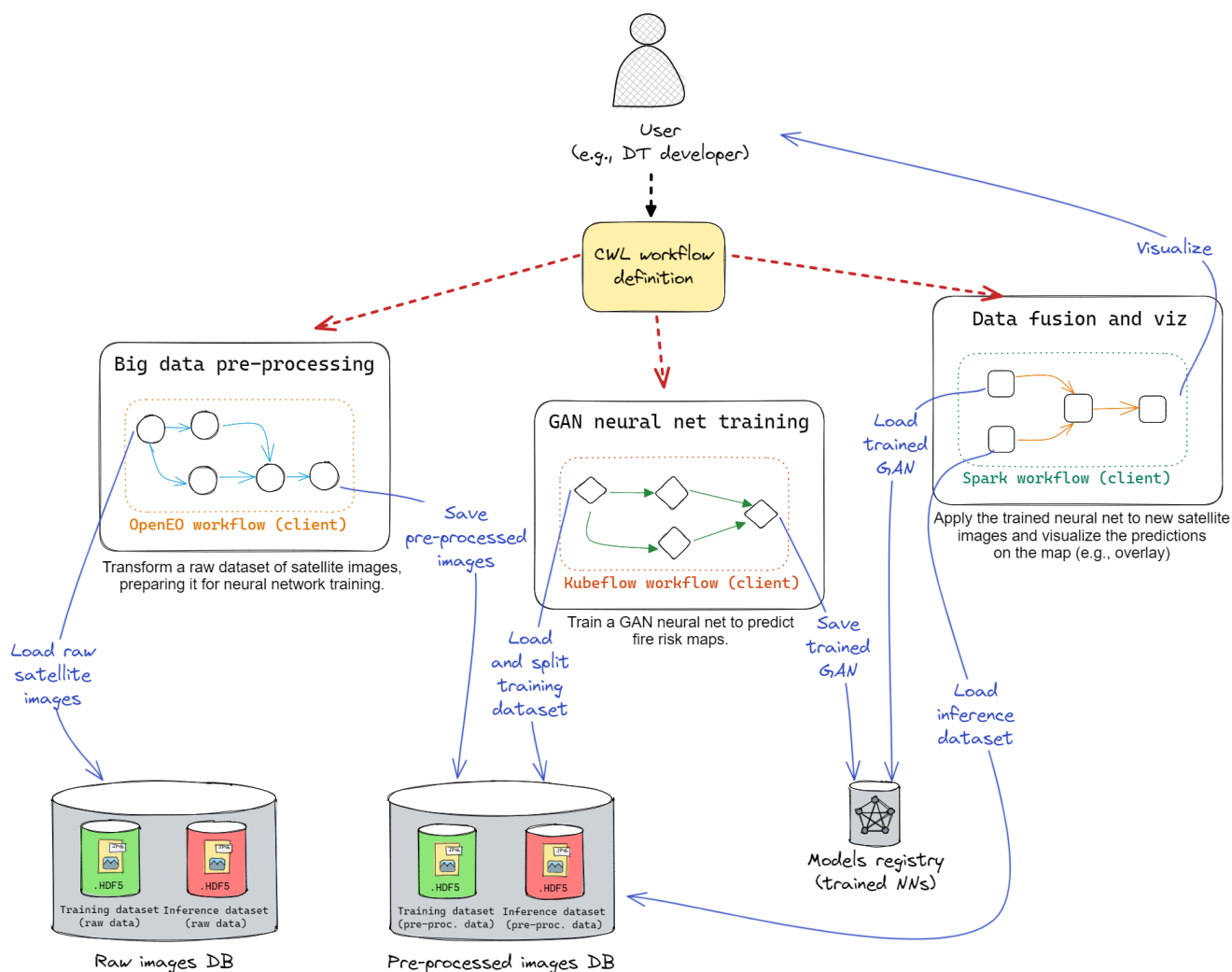


Figure 4 - Example of workflow composition for EO applications.

Figure 4 shows the generic usage model of workflow composition for the EO machine learning where the end-user defines the workflow following the standard of CWL.

3.2.2 Interfaces

The workflow composition tool provides CLI tools and a GUI so that DT Developers could set up and execute Workflows for the definition of DT applications, by integrating thematic modules tailored to their needs (developed by WP7).

Therefore, from one side the Workflow composition tool needs to deliver interfaces for DT developers and eventually DT users willing to execute DT applications workflows. On the other side the component should be responsible for the execution of the workflows delegating its execution to the backends that are deployed via the PaaS Orchestrator and the interfaces/API that are made available by WP5.



3.2.3 Technology stack

In this section we list the possible entrypoint for definitions of the Workflows to be executed and Workflow Management engines requested/suggested by the user communities to be used as sub workflows management engines.

Apache Airflow

One of the most promising solutions for the definition and execution of the CWL workflow is Apache Airflow or simply Airflow. Airflow comes with a built-in, easy-to-use web interface that allows users to manage, monitor, and visualise their workflows. This interface can be extended and customised to provide a more intuitive experience for users who are compositing workflows with different orchestration solutions.

Airflow has a plugin system that enables users to create custom operators, sensors, and other extensions to add new functionality or integrate with other systems. This feature allows the users to develop custom wrappers for other workflow orchestration solutions and incorporate them seamlessly into Airflow. Airflow includes a wide range of built-in operators for various tasks and integrations, making it easier to create complex workflows without needing to develop custom code. It is designed to be scalable, can handle large-scale, distributed workflows and supports parallel task execution, dynamic task creation, and execution on various distributed computing environments, including Kubernetes and Apache Mesos. Furthermore Airflow supports Python as its primary scripting language, which is widely used and accessible to users with varying expertise levels. It also runs on multiple platforms, including Linux, macOS, and Windows.

ecFlow

ecFlow⁹ is a workflow management system developed and maintained by the European Centre for Medium-Range Weather Forecasts (ECMWF). It is designed to handle complex workflows, particularly in the field of weather forecasting and numerical weather prediction. ecFlow has several features that make it suitable for managing large-scale, compute-intensive workflows:

- **Dependency management:** ecFlow allows to define complex dependencies between tasks, ensuring that they are executed in the correct order.
- **Scalability:** ecFlow can manage workflows that consist of thousands of tasks running across a distributed computing environment.
- **Fault tolerance and error handling:** ecFlow provides mechanisms for handling errors, retries, and failure recovery to ensure that your workflows continue running even in the presence of failures.
- **Monitoring and visualisation:** ecFlow includes a web-based user interface (ecFlowUI) that allows you to monitor the progress of your workflows, visualise dependencies, and perform various management tasks.
- **Extensibility:** ecFlow supports custom scripting using Python, which allows it to extend its functionality and integrate it with other tools and platforms.

⁹ <https://ecflow.readthedocs.io/en/latest/>



D6.3 Updated report on requirements and core modules functionalities

- **Environment agnostic:** ecFlow can be used on various platforms, including Linux, macOS, and Windows.

ecFlow is particularly well-suited for use cases involving large-scale, compute-intensive workflows, such as weather forecasting, climate modelling, and other scientific simulations.

openEO

openEO¹⁰ is an API specification¹¹ to discover, access and process Earth Observation data and define abstract processing workflows. It has grown into an independent open source community¹² standard in the earth observation community and is steered through the openEO project steering committee¹³.

The core API is defined following open API version 3 for the setup of a RestFul http-based API. The main concept is to have openEO as middleware between clients and back-end implementations. It exposes virtual views on data independent of the actual organisation of it. It implements the concept of virtual data cubes that can represent gridded or vector data. Also, the definition of processing is defined on an implementation agnostic process graph, allowing for very different implementations of processing frameworks in any kind of compute environment.

The main features of openEO are:

- Decoupling of data model and physical storage through data cubes¹⁴ in order to represent data along with time dimensions which can be queried and subsetted.
- Decoupling of process model and processing infrastructure through graph-based workflow descriptors;
- Workflows in openEO are described through the so called openEO process graph. The process graph follows the principles of a directed acyclic graph (DAG)¹⁵;
- Processes make the nodes in this graph and are pre-defined¹⁶. Custom processes can be added however to extend the functionality;
- Metadata catalogues functionality for data discovery and querying;
- Metadata functionality for description of available processes for the graph-based processing;
- Synchronous processing of graphs;

¹⁰ <https://openeo.org/>

¹¹ <https://api.openeo.org/>

¹² <https://github.com/open-eo>

¹³ <https://openeo.org/psc.html#members>

¹⁴ <https://openeo.org/documentation/1.0/datacubes.html#what-are-datacubes>

¹⁵ https://en.wikipedia.org/wiki/Directed_acyclic_graph

¹⁶ <https://processes.openeo.org/>



D6.3 Updated report on requirements and core modules functionalities

- Asynchronous processing of graphs through batch jobs and management of jobs;
- Authentication and authorization of any exposed micro service;
- Integration of custom user data;
- Integration of custom user processing scripts in R or python through so called User Defined Functions;
- Definition of higher-level processes based on simple processes based on User Defined Processes;
- Dynamic execution of workflows based on web service request, through secondary web services such as WM(T)S¹⁷, WCS¹⁸ or OGC API tiles¹⁹.

Apart from the main features and capabilities of the openEO API itself, it should be mentioned that openEO has a set of open source implementations, both on the client and server side, by very active user communities. Extensive documentation for all components is available especially for the user facing resources in form of the client libraries in R²⁰ and Python²¹, as well as the JavaScript based web-editor.

StreamFlow

StreamFlow²² is a workflow management system that focuses on the parallel and distributed execution of complex scientific applications. It is designed to handle both cloud and edge computing environments and aims to provide an easy-to-use framework for developing, deploying, and managing scientific workflows. StreamFlow offers a flexible framework for developing and managing complex workflows for applications that require high levels of parallelism, distributed computing, and cross-domain interoperability.

Some of the key features of StreamFlow include:

- **Hierarchical Workflow Composition:** Enables modular and reusable workflow components by allowing sub-workflows as tasks in higher-level workflows.
- **Resource Abstraction:** Simplifies resource management by defining resources independently of tasks, making it easy to switch configurations.
- **Cross-Domain Interoperability:** Supports various application domains and integrates with domain-specific languages (DSLs) and tools.
- **Fault Tolerance and Recovery:** Automatically detects and retries failed tasks, with checkpointing to resume execution from the last saved state.

¹⁷ <https://www.ogc.org/standard/wms/>

¹⁸ <https://www.ogc.org/standard/wcs/>

¹⁹ <https://ogcapi.ogc.org/tiles/>

²⁰ <https://openeo.org/documentation/1.0/r/>

²¹ <https://open-eo.github.io/openeo-python-client/>

²² <https://streamflow.di.unito.it/>



D6.3 Updated report on requirements and core modules functionalities

- **Scalability:** Optimises resource utilisation with parallel and distributed computing, suitable for cloud and edge environments.

Ophidia

Ophidia²³ is a CMCC research effort addressing scientific big data challenges. It provides a High-Performance Data Analytics (HPDA) framework for the analysis of scientific multi-dimensional data, targeting primarily the climate change domain, although it has been effectively used also with solid Earth, and environmental data.

The framework exploits an array-based storage model, leveraging the datacube abstraction, and a hierarchical storage organisation to partition and distribute large multi-dimensional scientific datasets over multiple nodes. It provides a platform for server-side in-memory computation through a large set of parallel operators, supporting statistical analysis, time series processing, data intercomparison, subsetting, multi-model analysis, etc.

Besides running single parallel operators, Ophidia provides a workflow management system for running complex scientific analysis composed of hundreds of tasks; this engine is integrated with the Ophidia server front-end. Different interfaces are provided to interact with the server for the submission of the workflow execution plan, including OGC/WPS or WS-I.

The workflow description request is written in JSON format according to a set of keywords defined in the workflow schema definition²⁴. The workflow engine is able to handle complex workflow in the form of DAG of tasks. Tasks can be defined using each of the Ophidia data analytics operators, including external Python/bash scripts or binary executables.

From the client side, the PyOphidia module (i.e., the Ophidia Python bindings²⁵) can be used to interact with the engine to send and execute workflow documents with the defined JSON format. The module can also be used to build the workflow document in a programmatic way, through a recent extension. Moreover, in the context of the project the PyOphidia library is being extended with some utilities to support execution of workflows of Ophidia tasks written in CWL format. These additional capabilities translate a workflow in CWL format, by using the CWLtool, in the native Ophidia JSON format before submitting the workflow description to the front-end server. It is worth mentioning that only part of the CWL standard is supported.

The workflow engine takes care of handling the whole execution flow:

- translates the JSON document into an “execution plan”, i.e., an ordered list of single tasks that are managed by the resource manager;
- handles the scheduling of the different tasks based on their dependencies and available resources;
- tracks and monitors the execution status of each task;

²³ <https://ophidia.cmcc.it/>

²⁴ https://ophidia.cmcc.it/documentation/users/workflow/workflow_basic.html

²⁵ <https://pyophidia.readthedocs.io/en/latest/>



D6.3 Updated report on requirements and core modules functionalities

- handles task failures, potentially resubmitting the same task for execution.

The Ophidia workflow system supports different types of abstractions; besides data and flow dependencies and simple task definition, it provides flow control constructs to handle conditional, iterative and parallel execution of sub-workflows (i.e., a subset of the workflow tasks).

Application Deployment and Execution Service (ADES)

The ADES originated from the Earth Observation Exploitation Platform Common Architecture (EOEPCA) project²⁶. It is a Kubernetes-based CWL execution service designed to provide a seamless and easily scalable cloud-based application execution environment. Interaction with ADES is managed via OGC API Processes, allowing users to deploy, undeploy, view, and execute processes.

When a workflow is submitted, a job is created. The ADES first performs initial setup tasks, such as pulling the necessary Docker images specified in the CWL. A pod is then spawned, serving as the execution environment for the application. During the stage-in process, all required data is ensured to be available for the application's execution. The outputs are stored in an S3 bucket, accompanied by a STAC²⁷ (Spatio Temporal Asset Catalog) catalogue that provides metadata for each application output.

OGC Application Package

According to the Open Geospatial Consortium (OGC) best practices document, an Application Package is formally defined as "a platform-independent and self-contained representation of an application, providing executables, metadata, and dependencies such that it can be deployed to and executed within an Exploitation Platform"²⁸. The Application Package is designed to free developers from the constraints of existing exploitation platforms, ensuring platform independence so that it can be executed on any underlying hardware. Its self-contained nature allows developers to use any programming language or framework.

An Application Package comprises two main components: a Docker Image and a CWL file.

The Docker Image contains all the application software, dependencies, and environment variables necessary to run the application. It must also provide a command line interface for interaction, as the application is deployed without further user input. According to OGC best practices, an OGC Application Package stages in and out data using a STAC.

The CWL file describes all command line tools required to run the workflow, with all inputs and outputs defined and linked semantically. Additionally, metadata about the application is stored in the CWL document.

²⁶ [Earth Observation Exploitation Platform Common Architecture - EOEPKA Portal](#)

²⁷ <https://stacs.org/>

²⁸ [OGC Best Practice for Earth Observation Application Package](#)



3.2.4 Interaction with other components

The workflow composition and workflow execution will interact with the real time acquisition component, as it will trigger the execution of workflows upon data arrival. The component will also trigger SQaaS service pipelines as part of the workflow execution to enable Model validation and Data FAIRness. Finally, the provenance component described in the next section will also be integrated to support lineage metadata tracking of the interTwin workflows. To this end, initial work has already been performed for the integration of provenance support in the Ophidia framework (described in the next section).

3.3 Provenance in Workflows

Workflow and provenance are two faces of the same medal. While the former addresses the coordinated execution of multiple tasks over a set of machines, the latter relates to the historical record of data from its original sources. In this respect, provenance represents valuable accompanying documentation for scientific data and experiments, providing historical context, documenting data transformations, and addressing trust and authenticity of data.

Additionally, provenance is a key enabling factor for reproducibility, thus playing a relevant role in Open Science. On the same line, to facilitate findability, accessibility, interoperability and reusability, provenance documents should also adhere to FAIR principles.

However, due to the complexity of scientific workflows and applications, provenance needs to be managed across multiple and multifaceted processes and services, which introduces the need for linking provenance documents as well as expressing provenance information at multiple levels.

In the former case, linking provenance information needs the use of Persistent Identifiers (PIDs) to reliably find, use and cite associated documents, while in the latter different levels of granularity for provenance could serve very different scenarios, according to users' needs to investigate such information at different levels of depth.

All the aspects presented before pose interesting challenges at the level of services, interfaces, infrastructure, libraries and tools as well as applications.

The core module yProv relies on the PROV family of standards for its internal information model and it enables core provenance functionalities through an interoperable service interface linked to a persistent graph database in its back-end.

In the interTwin project, such a service is being extended to help tracking provenance information associated to climate analytics workflows both in batch and real-time workflows. The service is also intended to manage scenarios with multi-level provenance information allowing scientists to drill-down into specific processes according to their needs. In order to access the provenance information recorded by the service, a provenance explorer GUI is under implementation and will be finalised for the final interTwin release.



D6.3 Updated report on requirements and core modules functionalities

Besides the yProv service and GUI, a new library (prov4ML) is also part of the provenance software ecosystem under development. Such a library aims at tracking provenance information within learning tasks, thus providing useful insights about AI training processes.

Though its primary application domain is climate, its cross-domain and standard API definition enables its re-use over multiple different domains, thus multiplying exploitation opportunities within and outside the project, across a variety of different use cases.

Some relevant examples include provenance support in climate analytics workflow (which links to the Ophidia big data framework), more general scientific workflows (which links with WFMSs) as well as AI training processes. The last scenario is being implemented in the context of different use cases like the extreme events Digital Twin (T4.5 - Tropical Cyclones detection) and deep learning processes for High Energy Physics (which also addresses an integration with [itwinai](#)).

Concerning the Ophidia framework, an extension for supporting provenance is being implemented in the PyOphidia library. Such a feature is still on the development branches of the library and will be part of the next WP6 release. Through this extension provenance documents in JSON format (following the W3C-PROV standard) can be generated by the user through a method of the library, once the Ophidia workflow execution is complete. The provenance document can then be sent to the yProv service for a fine grained analysis of the workflow provenance structure. An initial integration of the functionalities has already been tested. [Figure 5](#) shows how the interaction among the different components of the Ophidia framework and the yProv service are carried out.

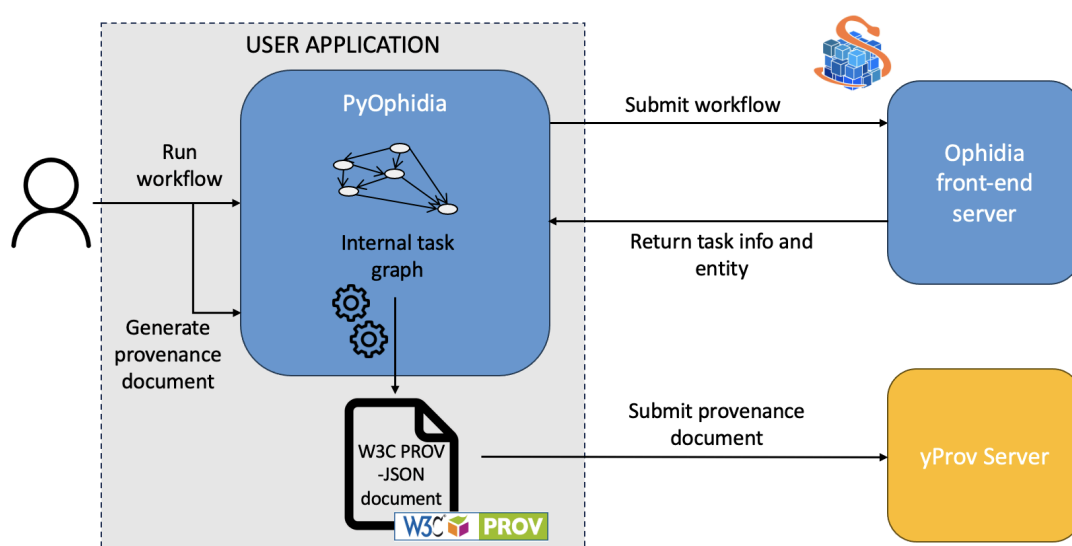


Figure 5. High-level schematic view of the use of the PyOphidia module for generating and managing the provenance document of a workflow.

3.4 Data Fusion

Data fusion is an important component in the implementation of workflows consisting of multiple heterogeneous data sources and is the output of Task 6.3.

The main challenge in this task is the combination of domain specific datasets and tools in the general workflow of interTwin. Datasets need to be prepared to be used in generic components taking care of the analytics framework and artificial intelligence. By the end of processing, results from multiple model runs need to be re-integrated for visualisation purposes. It's important to note that data fusion requirements mainly come from the environmental use cases, where it is necessary to perform fusion on a large number of heterogeneous datasets and indicators to process the data. Therefore the design and the technology used as base for the data fusion components implementation are tailored to that domain. In the context of the environmental use cases data from various sources need to be integrated covering climate model output with satellite imagery as well as various sources of vector data.

In total three tasks are hence covered by this activity:

- Definition of guidelines for the implementation of thematic modules in order to be interoperable with the general workflow in terms of data exchange.
- Implementation of processes for merging datasets from different sources in collaboration with developers of thematic modules, including gridded and vector data.
- Implementation of processes for preparing data for ingestion into AI workflows.

3.4.1 Data Fusion in Workflows

OpenEO facilitates the preparation and seamless integration of data from several sources, hence enabling data fusion in workflows. These workflows make use of OpenEO's ability to carry out crucial data fusion and preprocessing operations, guaranteeing that data is appropriately staged for the workflow's later phases.

The data is sent back to OpenEO when the workflow's processing portion is finished. It can now either be saved as the final result or go through additional post-processing. This method makes it possible to link several workflows together. Since it allows for multiple data fusion events, this design greatly improves the flexibility and scalability of the data processing pipeline, making it ideal for complicated processing workflows.

In order to allow for dynamic access of data during various steps in the workflow, a common architecture for data discovery and access is required. In analogy of ongoing activities in the Destination Earth this will be based on STAC (SpatioTemporal Asset Catalog) as the main data catalogue, describing collections of online accessible data through various technologies. The Thematic Module developed in WP7 raster2stac²⁹ addresses the need of having a single entripoint to create STAC compliant metadata

²⁹ <https://pypi.org/project/raster2stac/>



D6.3 Updated report on requirements and core modules functionalities

from a variety of raster datasets in an unified manner. More so, the STAC metadata works seamlessly with downstream tools that can directly access datasets provided in the interTwin data lake interfacing with Rucio³⁰ or alternatively on S3 bucket. In order to achieve this, a specific procedure to register into STAC catalogue datasets uploaded to Rucio is under implementation by WP5.

Additionally, the openEO concept of virtual data cubes allows to handle multidimensional data from multiple domains (e.g.: remote sensing and climate analysis) in the same way, since the processes are generalistic and can handle them independently from their source, given the data collections metadata (provided as STAC documents). A key development done for the openeo-processes-dask³¹ Thematic Module in WP7 consists in extending the capabilities of loading and processing data coming from the generated STAC Collections, making the implementation flexible enough to handle several different cloud native file formats (COGs, ZARR, Kerchunk).

3.4.2 Processing Fusion in Workflows

interTwin provides a host of different data processing engines depending on the scientific domain and applications used in the manifestation of a specific digital twin.

Complex workflows often require data from multiple sources. OpenEO, combined with OGC Application Packages, enables seamless data fusion by supporting the integration of multiple datasets. These datasets can be preprocessed using a variety of cloud-optimized processes, such as reprojection, aggregation, and temporal and spatial filtering. These preprocessing steps facilitate data fusion, preparing the data for further analysis.

Once the fused data is ingested into an Application Package, additional specific preprocessing may occur, such as atmospheric correction or co-registration of SLC SAR data. The Application Package allows users to run their custom algorithms or models, such as examples present in the Thematic Modules. The results are then made available in OpenEO via STAC for post-processing or storage. **Figure 6** shows how the processing workflow looks like expressed in an OpenEO Process Graph.

³⁰ <https://rucio.cern.ch/>

³¹ <https://github.com/Open-EO/openeo-processes-dask>



D6.3 Updated report on requirements and core modules functionalities

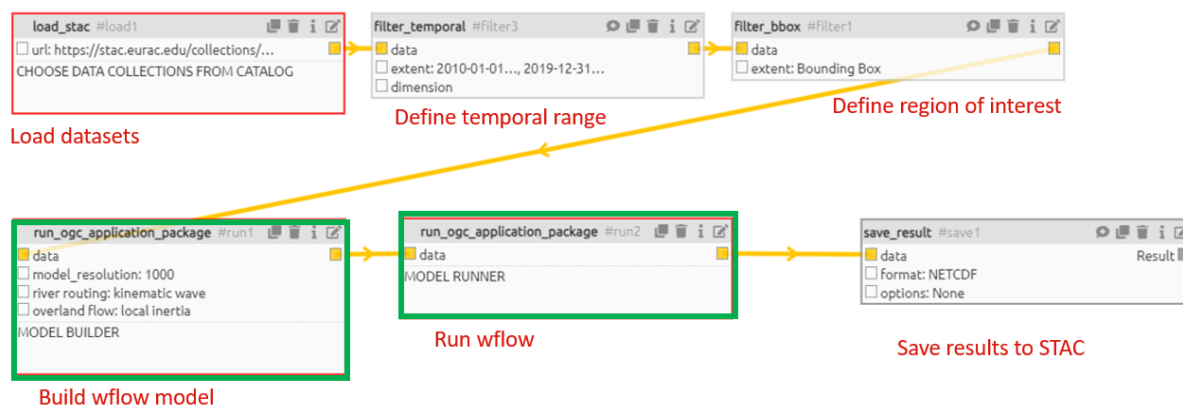


Figure 6. Example OpenEO processing fusion.

3.4.3 Data Fusion for Artificial Intelligence

Machine learning models require that multi-dimensional data inputs should be: co-located spatially and temporally, eventually reshaped to be ready for training, split into train, test and validation and scaled to a numerical range to improve convergence.

Data fusion by combining the abilities described in the previous two subsections on data fusion and processing fusion in workflows are powerful tools to perform those operations on the input data for further enhancement in data driven models.

OpenEO reproducible workflows ensure that input data are correctly integrated, the results stored in the data lake and the corresponding metadata uploaded into the STAC catalogue.

The process of developing ML models can be characterised by two phases: a) an experimentation phase where the user is heavily experimenting on the choice of the model architecture, model structure and the overall workflow; b) a consolidation and inference phase where the overall process has reached a certain stability and the user may want to re-train the model on other input data (e.g., geographical regions), perform additional hyperparameter tuning or simply running the model in inference.

In InterTwin, these two phases are enabled by partially different sets of technologies.

In the experimentation phase, the [itwinai](#) module offers the highest flexibility whilst also covering some common data preprocessing steps such as data splitting and scaling, whereas the workflow composition and execution is managed by KubeFlow (see 4.1.1).

In the consolidation and inference phase, the relevant itwinai's functionalities and pipeline execution are wrapped and exposed in a CWL Application Package, whereas a customisable openEO process graph takes care of chaining the data fusion steps to the itwinai-based CWL and seamlessly executing the overall workflow.



4 Components for AI workflows

The Artificial Intelligence (AI) subsystem in the proposed Digital Twin Engine (DTE) is intended for data-driven Digital Twin (DT) models and is the output of Task 6.5. This subsystem is mainly devoted to two macro-operations: training and deployment of machine learning (ML) models. In this context, the DT developer is mostly focusing on ML training workflows, which also include hyperparameter tuning and validation of ML models. The DT application user is generally more interested in the deployment of pre-trained ML models on its preferred infrastructure (e.g., cloud services, on-premises servers, HPC systems). In the more general case the models will be re-trained by the scientific users to input new data, or to focus in particular areas of the parameter space. Figure 4 depicts the internals of the AI subsystem, showing how the ML training and deployment modules interact with other components, such as distributed training, metrics logger, models registry, and hyperparameter optimization (HPO). This AI subsystem developed in the context of the interTwin project is called **itwinai**³², which is a Python library to support large-scale AI applications in scientific DT applications.

Training an ML model involves loading some (pre-processed) dataset from the storage and splitting it into training, validation and test datasets. The DT developer inputs the details of the ML model from the Platform-as-a-Service user interface (PaaS UI), like a thematic module, including the loss function, evaluation metrics, neural network architecture, optimizer type, etc. The user can choose among a collection of tools for both distributed training (e.g., Horovod³³, DeepSpeed³⁴), and HPO, such as Ray Tune³⁵. Once a model is trained, its performance is assessed on the validation dataset (ML-level validation). ML logs, like metrics, are saved on disk and made available to the user for future inspection by means of the "metrics logger", whereas the best models are saved in the "models registry". In the case of online inference, the input is a stream of data that is updated over time. The full deployed DT includes the possibility to perform real-time inference on the trained models.

An ML model is deployed after it passes domain-specific validation performed by the "Quality and uncertainty tracing" DTE's module, when validation is required.. The user chooses a pre-trained ML model from the model registry, which is going to be served as a step in the overall DT inference workflow. There may be multiple versions available for the same ML model and the user can choose which version to deploy as the "living" DT model. Once the full DT is deployed as a workflow, it can process real-time streams of data from the real world, and the experimenter can interact with it, like performing experiments and making predictions.

One of the main challenges of this task is to provide support for a large spectrum of users with different degrees of expertise with ML workflows and MLOps best practices. This entails a tradeoff because base users would like to have a simple interface that is

³² <https://itwinai.readthedocs.io/>

³³ <https://github.com/horovod/horovod>

³⁴ <https://github.com/microsoft/DeepSpeed>

³⁵ <https://docs.ray.io/en/latest/tune/index.html>



D6.3 Updated report on requirements and core modules functionalities

easy to understand, whereas experienced users would like to have finer control of the underlying ML technicalities. This can be solved by developing two different user interfaces, considering two levels of user profiles:

- **The DT Developer** (Experienced user / ML researcher) has full control of the ML workflow, custom losses and metrics, NN architectures, ensemble methods, etc. The user provides custom training/validation scripts, interfacing directly with PyTorch, TensorFlow and MLflow. In this case, the user also provides the logic for loading non-standard data formats.
- **The Scientist** (Base user) has little experience with ML workflows and provides only high-level definitions of ML tasks. Almost everything is automated under the hood, reducing the engineering effort required from the scientist. If the scientist has some special needs, they can outsource changes to a DT developer. To achieve the desired level of abstraction for the Scientist, **itwinai** enables the DT developer to define modular AI workflows composed of reusable, reproducible, and fully-configurable steps. Scientists can reuse such steps in their AI workflows without having to understand their internal implementation, fostering code reuse and streamlining the development of AI workflows in scientific digital twin applications.



D6.3 Updated report on requirements and core modules functionalities

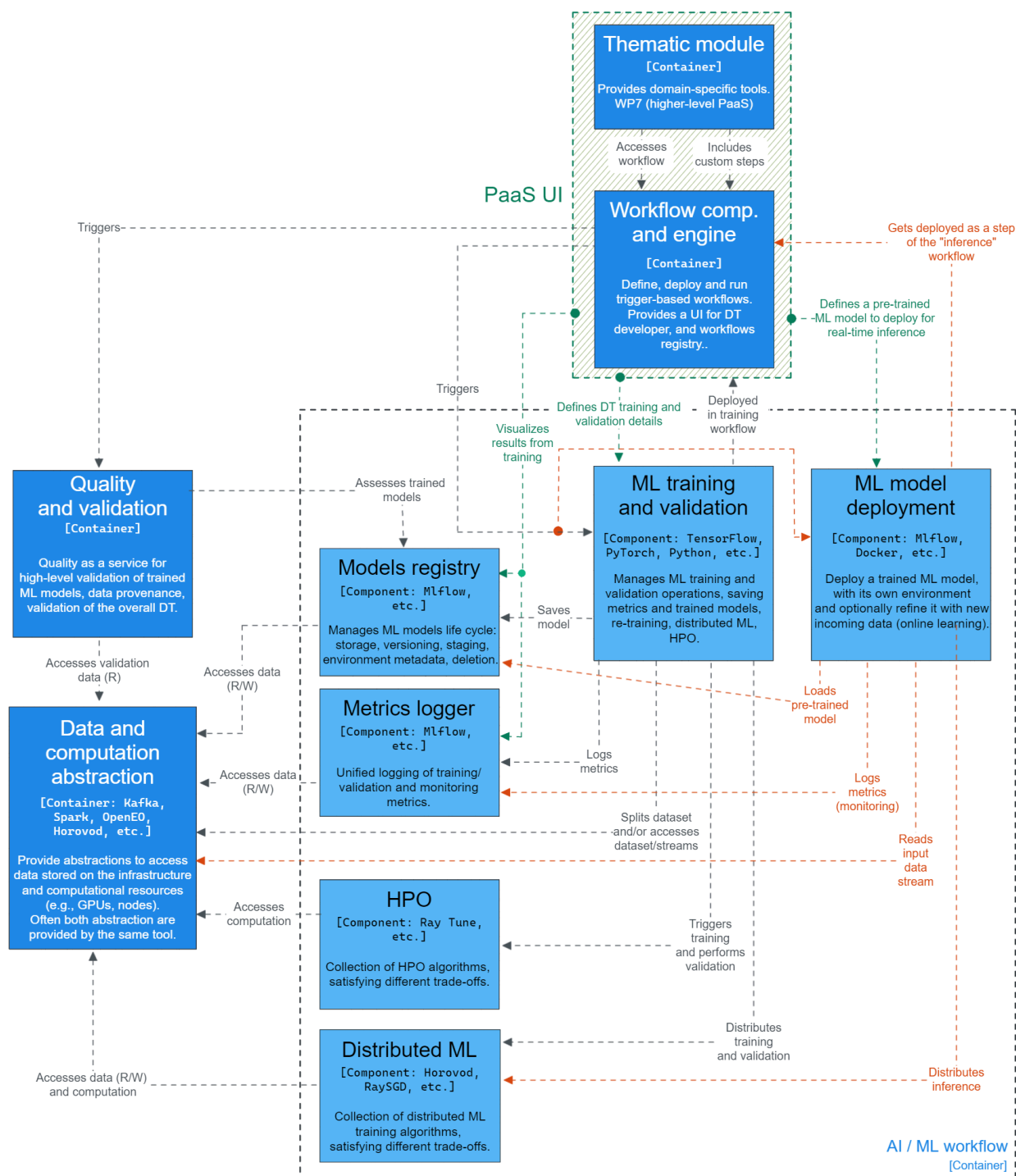


Figure 7 - Detailed view of the architecture of itwinai

An overview of the ML training component and its functionalities is described in [Section 4.1](#). The description of the Models Registry, where pre-trained ML models are stored, is provided in [Section 4.2](#), while [Section 4.3](#) describes the metrics logger component, used to log ML metrics and metadata. Finally, a description of the ML model deployment component and its sub-components is provided in [Section 4.4](#). For each of these AI/ML components, an overview of the component and its functionalities,



D6.3 Updated report on requirements and core modules functionalities

interactions with other AI/ML components, the required technology stack, and the interactions with other tasks are provided.

4.1 Training module

This section contains a description of the different components of the training module in the AI workflow engine. The architecture of this module and its interaction with the other modules of the AI workflow component is shown in [Fig. 8](#).

4.1.1 General Description and functionalities

This module provides various functionalities to the DTE for training their AI models. The module allows access points to local, cloud, and/or High-Performance Computing (HPC) resources for training the networks. The cloud and HPC resource provisioning have to be arranged on a use-case basis, in this regard, interaction with WP5 and T6.4 is foreseen. In collaboration with WP5, support will be provided for job scheduling, for example with Slurm, Cron, etc. JupyterLab³⁶ is currently being investigated as a programming interface for this module. The primary components of this module are summarised here.

itwinai virtual environment

The itwinai library employs a Python-based virtual environment for executing the workflows. At the moment, the environment supports the PyTorch³⁷ and TensorFlow³⁸ frameworks, which are decided based on the interaction with the use-cases that were integrated in the initial phase of the project. The environment creation is supported on multiple platforms, locally on a CPU-only or single-node/GPU system (for instance, on a laptop), containerized execution, and on HPC systems. In terms of HPC systems, currently the HDFML system at the Jülich Supercomputing Center (JSC) has been configured. Support for Vega EuroHPC is also being gradually added. The creation of the environment is streamlined with a Makefile. The documentation of the library outlines detailed instructions on the creation and usage of the environment. Furthermore, the itwinai library has also been released in the Python Package Index (PyPI), which enables users to directly install the library with the package management system pip. For the DT Developers, the package can be built from the source, using make targets, which are provided for various modes of execution, e.g. CPU or GPU.

³⁶ <https://jupyter.org/>

³⁷ <https://pytorch.org/>

³⁸ <https://www.tensorflow.org/>



D6.3 Updated report on requirements and core modules functionalities

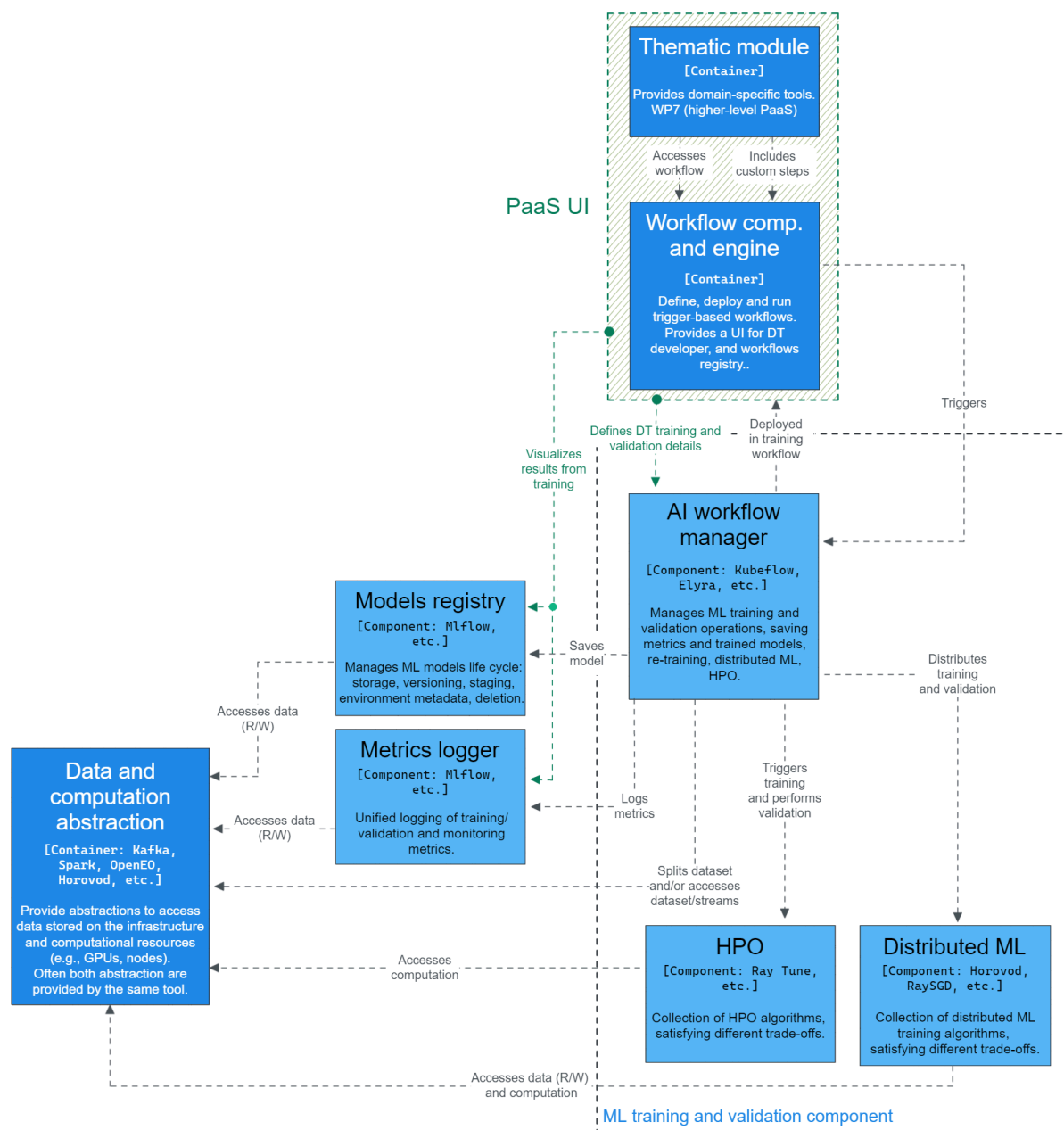


Figure 8 - Detailed view on the architecture of ML training module

Distributed training

The training module provides functionality for distributed training of the AI models, which is essential to maximise utilisation of HPC resources. Typically, training AI models in a distributed or parallel manner uses either data parallelism or model parallelism techniques. In the case of the former, batches of the dataset are distributed across the workers, while each worker receives a replica of the model. Subsequently, each worker trains the copy of the model in each training cycle (optimization step over a mini-batch), and after a certain number of cycles (dependent on the framework), the gradients are exchanged across the workers to synchronise the training. In case the AI models are too large to fit into one worker (e.g., GPU), a model parallelism feature is required, which distributes a single model to several workers, such that each worker only executes a



D6.3 Updated report on requirements and core modules functionalities

fraction of the full model. At the time of writing this report (07/2024), model parallelism in itwinai is possible by employing the ZeRO Stage 3 optimizer³⁹ provided by DeepSpeed framework through parameter partitioning.

There are multiple open-source frameworks that provide distributed training functionality. PyTorch Distributed Data Parallel (DDP) module⁴⁰, Horovod developed by Uber, DeepSpeed from Microsoft, and MultiWorkerMirroredStrategy⁴¹ from TensorFlow are the commonly-used frameworks available in itwinai for data distributed training via a simplified abstraction layer. Depending on the use-case requirements, support for other frameworks will be added to the library.

Hyperparameter Optimization (HPO)

HPO is the process of fine-tuning machine learning and deep learning models in order to improve their accuracy. This is increasingly being employed in training of AI models and is expected to be necessary for the use-cases in the interTwin project. The HPO process involves optimising the allocation of computational resources to various configurations (such as learning rate, batch size, number of filters, etc.) of the AI models. The objective is to minimise the total computational budget to find the optimal configuration with the highest accuracy. Various algorithms such as HyperBand⁴² and BOHB⁴³ provide solutions for HPO. These are implemented through the open-source HPO framework, RayTune⁴⁴. This development is still at an experimental stage.

³⁹ <https://arxiv.org/abs/1910.02054>

⁴⁰ <https://pytorch.org/docs/stable/distributed.html>

⁴¹ https://www.tensorflow.org/api_docs/python/tf/distribute/MultiWorkerMirroredStrategy

⁴² <https://arxiv.org/abs/1603.06560>

⁴³ <https://arxiv.org/abs/1807.01774>

⁴⁴ <https://docs.ray.io/en/latest/tune/index.html>



D6.3 Updated report on requirements and core modules functionalities

Workflow Pipeline Editor

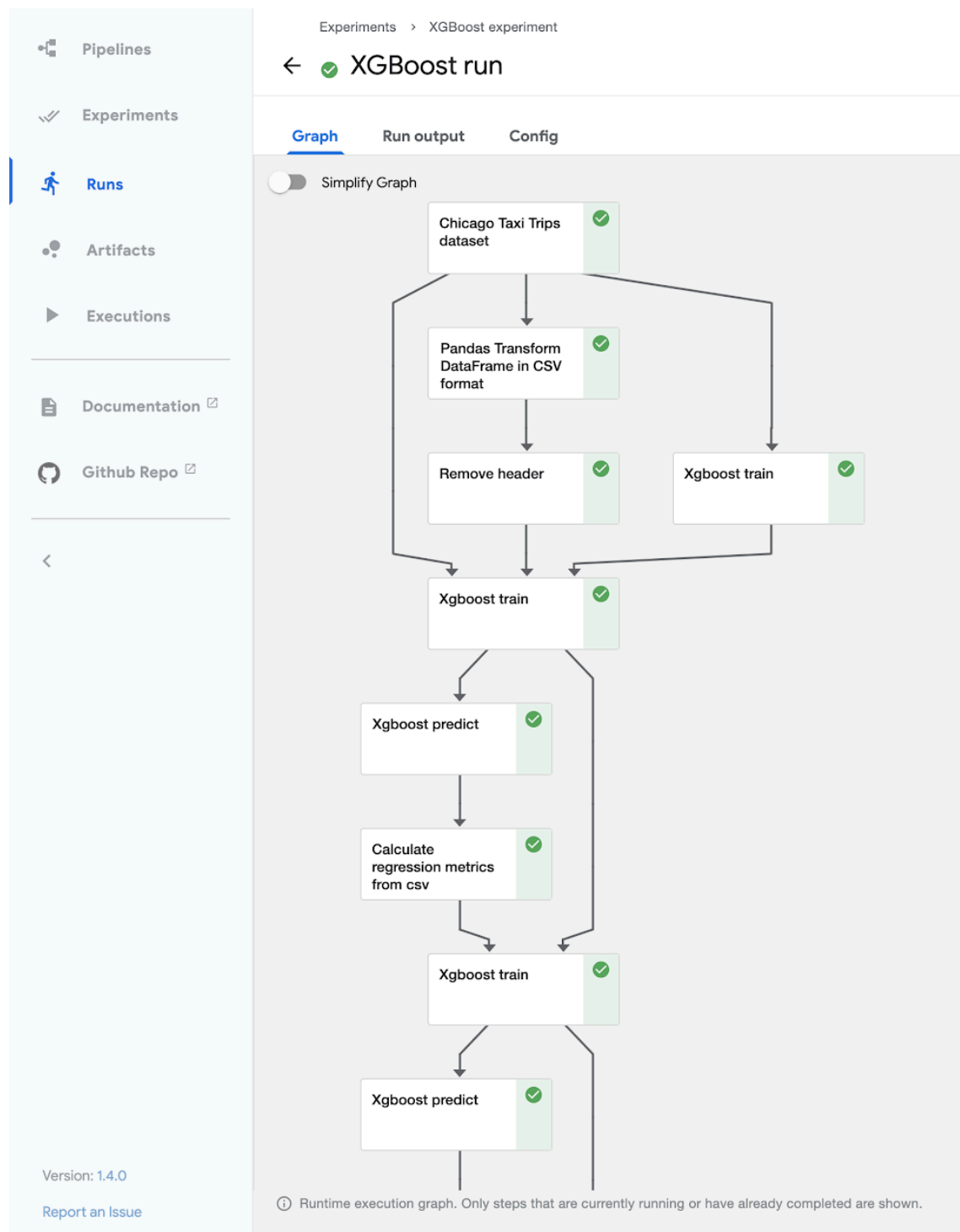


Figure 9 - Kubeflow-based example of an illustrative AI pipeline⁴⁵

⁴⁵ Image reference:
<https://www.kubeflow.org/docs/components/pipelines/legacy-v1/introduction/#the-runtime-execution-graph-of-the-pipeline>



D6.3 Updated report on requirements and core modules functionalities

The AI workflow consists of multiple components which can broadly be categorised into three parts, namely preprocessing, training, and *optionally* post-processing. An AI pipeline example with the basic components is shown in [Figure 6](#), which is Kubeflow-based pipeline definition and execution of an AI workflow.

In the exemplary pipeline, the preprocessing step involves a data transformation step. This may involve multiple operations such as data merging, splitting, preprocessing and preparation. Many of these operations will be part of Data Fusion, described in [Section 3.4](#). The training component consists of various modules defined concretely in [Section 4.1](#). In the example, the user trains an AI model, which can be conveniently defined with the pipeline-based approach in Kubeflow.

Workflow Management Tool

The solutions to launch the AI pipeline in cloud/HPC infrastructure were investigated in the initial phase of the project and multiple frameworks have been identified. Based on the requirements and AI-based functionalities, Kubeflow⁴⁶, an open-source platform, is adopted as the AI workflow management solution. The advantage of Kubeflow is that it is an ML-oriented workflow manager, which makes it suitable for most interTwin use cases. In one single package, it provides a UI for workflow composition, Tensorboards for metrics logging, models registry and models versioning, and allows to serve pre-trained ML models as standalone services, in line with MLOps best practices. These features are required by interTwin use cases. On the infrastructure side, Kubeflow depends on Argo Workflows and Kubernetes. When a Kubeflow workflow is deployed on the infrastructure, its steps are deployed as separate containers and orchestrated by the Argo Workflows manager. Also as seen in the previous section, it also allows the definition of pipelines, which are convenient for users to define their ML execution steps. A TOSCA template for launching Kubeflow instances on the cloud infrastructure at the Jülich Supercomputing Center (JSC) has already been developed by Task 6.4. Tests have already been conducted for the AI workflow definition and execution for Task 4.2.

4.1.2 Interfaces

The user will be provided with a web-based portal or API to select training hyperparameters and ML models. A selection of pre-trained models will also be available in this portal for base users. The training module can be triggered by the advanced workflow composition tool with Kubernetes-like OpenStack APIs and/or Jupyter notebook-based APIs. The trained models will be available for access in [Model registry](#) for other components such as the [Quality and uncertainty tracing](#).

4.1.3 Technology stack

The AI model is trained in a Python environment. The required software stack is listed below, corresponding to the operations:

- Train the AI model (e.g., PyTorch, TensorFlow, etc.)
- Interface to metrics logger (e.g., MLflow, WandB)

⁴⁶ <https://www.kubeflow.org/>



D6.3 Updated report on requirements and core modules functionalities

- Communicate with external environment (e.g., CLI, REST APIs)
- Distributed AI framework (e.g., DeepSpeed, Horovod, PyTorch DDP, TensorFlow MultiWorkerMirroredStrategy etc.)
- Drivers to access (parallel) computing infrastructure (e.g., CUDA drivers)
- Job scheduling support (e.g., SLURM, kubernetes)
- Container (e.g., Docker, Apptainer) engine support to deploy models in infrastructure
- Workflow orchestrator to launch training of AI models in infrastructure (e.g., Kubeflow)

4.1.4 Interaction with other components

Model registry: During the training process, the model will be periodically stored at certain time intervals (e.g., stored after every 10 epochs) through checkpoints and written to the model registry. Firstly, this acts as a safeguard mechanism. In case the model training is not proceeding well, the model can be reset from a certain checkpoint to resume with new configurations (e.g., learning rate) without the need to completely discard the model and retrain from the beginning. Secondly, this allows the convergence analysis of the models at different training stages.

Metric logger: Training and validation loss, as well as visualisation during the training process will be fed to the metric logger.

Workflow composition: The training component will be deployed in a container via Docker or Singularity. Workflow Composition will embed the container and call it once the previous steps have been completed. This tool will also provide a GUI or a file-based ML configuration (containing the model type, hyperparameters, dataset URI, etc.) setup that will write to a markup language file. The AI workflow component will then load the YAML file and set up the training module accordingly. For the DT developer, there will be the possibility of accessing an exposed Jupyter notebook directly, without going through the Workflow Composition components.

Quality and uncertainty tracing: During the training procedure, intermediate results are stored as datasets and other kinds of artefacts. The SQAaaS component will take care of assessing their quality, including the FAIRness of training datasets and, optionally, other datasets generated by AI workflows.

Computing Federation (interLink): The component developed in WP5 allows the transparent offloading of computing containers to remote computing sites (including HTC and HPC clusters). Itwinai containerized training pipelines have been already integrated with interLink and tested both Julich and VEGA HPCs.

4.2 Model Registry

This section describes the Model Registry module of the AI workflow engine.

4.2.1 General description and functionalities

The Model Registry provides a central hub for storing and sharing ML models, along with their associated metadata, such as performance metrics, hyperparameters, and deployment history.

The Model Registry allows users to register models as "production-ready" or "experimental", depending on their stage in the development process. Users can create and manage multiple versions of a model, each with its own set of metadata and artefacts, such as serialised model files, data preprocessing scripts, and model training logs.

The Model Registry also offers collaboration features such as access control, version history, and model comparison. Teams can work together to review and approve model changes before they are promoted to the production environment. Model serving and deployment can be automated using integrations with cloud platforms such as AWS, Azure, or custom deployment scripts. Once a ML model is selected to be used for inference (i.e., making predictions), it can be deployed using a number of existing services, such as KServe⁴⁷, Nvidia Triton Inference Server⁴⁸, Ray Serve⁴⁹, MLFlow Inference Server⁵⁰.

4.2.2 Interfaces

The Model Registry offers a web-based user interface for browsing and searching models, as well as APIs for programmatic access and integration with other tools in the ML ecosystem described in [Interaction with other components](#).

4.2.3 Technology stack

One of the most promising candidates for the Model Registry is MLFlow Model Registry⁵¹. MLFlow is an open source platform for the complete ML lifecycle management, including experimentation, reproducibility, deployment, and monitoring. One of its core features is the MLFlow Model Registry, which is designed to enable teams to collaboratively manage and version ML models. The Models Registry is deployed on cloud resources and provides a centralised repository of pre-trained ML models, which can be accessed from different locations, compared, re-trained on new data, or deployed for inference.

4.2.4 Interaction with other components

Training module: The Model Registry will store trained models with a certain frequency from the Training module. This will happen at two different frequencies each with a specific purpose:

⁴⁷ <https://kserve.github.io/website/latest/>

⁴⁸ <https://developer.nvidia.com/triton-inference-server>

⁴⁹ <https://docs.ray.io/en/latest/serve/index.html>

⁵⁰ <https://mlflow.org/docs/latest/deployment/deploy-model-locally.html>

⁵¹ <https://mlflow.org/docs/latest/model-registry.html>



D6.3 Updated report on requirements and core modules functionalities

- While training is still ongoing, intermediate models will be stored regularly in order to have backup models in case the training is diverging, and the model behaviour is different than expected. This ensures that not all the training is lost and allows to fall back to models saved at previous training epochs
- After the training has been completed models will be stored in the Model Registry for the purpose of evaluation from a use case perspective, i.e., the Scientist will compare different model types regarding e.g., architecture, number of parameters, datasets used with other ML and non ML model and assesses which model are fit to be post-processed and deployed.

The frequencies are use case specific and will be determined either by the DT developer or the Scientist.

ML model deployment: The Model Registry will provide a catalogue of models that can be accessed via the MLflow client. Model will be made available in the ML specific framework and in the Open Neural Network Exchange (ONNX) format. The models in the registry will be deployed in a container via the ML deployment component either as Tensorflow or PyTorch models according to use case preference and hosted as a server.

Quality Assurance: The component can query the ML deployment server. It will certify the quality of each model in the registry according to the FAIR (Findable, Accessible, Interoperable and Reusable) quality assessment.

4.3 Metric Logger

This section describes the Metric Logger module of the AI workflow engine.

4.3.1 General Description and functionalities

The metric logger allows users to define and track either predefined metrics, such as accuracy, precision, recall, mean squared error or allow the DT developer to define custom use case specific metrics. It also supports the creation of experiments, which can group together related runs of a model training or evaluation task. DT Users can compare metrics across different runs and experiments, and track how metrics change as the model is updated or new data is added. The Metric Logger also stores provenance information for AI workflows, computed at runtime by **itwinai**. Such information includes system metrics (e.g., GPU/CPU usage, carbon footprint assessment), and additional metadata needed to improve reproducibility and transparency of AI workflows in scientific digital twin applications.

4.3.2 Interfaces

The Metric Logger provides a user-friendly web-based interface for exploring and visualising metric trends over time, as well as APIs for programmatic access and integration with other components described in **[Interaction with other components](#)**.

4.3.3 Technology stack

The Metric Logger is characterised by a client-server duality.



D6.3 Updated report on requirements and core modules functionalities

DT developers access the client through Python APIs within their AI workflows. The ML logger client is accessed via a simplified abstraction layer provided by **itwinai**. This abstraction layer ensures a uniform API across popular logging frameworks, such as MLFlow⁵², Weights&Biases⁵³, and TensorBoard⁵⁴. It allows DT developers to avoid being tied to any specific framework and makes it easy to switch logging frameworks without modifying the user code.

On the other hand, the server side of the Metric Logger is an optional component, which is usually deployed on cloud resources. It allows to store ML metadata, metrics, and artefacts in a centralised location, accessible by different scientists at the same time and from different geographical locations. The Metric Logger server can be an MLFlow Tracking Server⁵⁵, Weights&Biases public or managed Platform⁵⁶, or similar. Sometimes ML workflows are executed in network-segregated environments (e.g., due to security policies on some HPCs), preventing the logging client from communicating with its server counterpart. In these occasions, the logging server will not be needed, or it will need to be integrated with other technologies, such as the Rucio Data Lake.

The Metric Logger is a flexible and scalable tool that enables users to record, visualise, and compare ML metrics during model training and evaluation. It integrates with popular ML libraries such as TensorFlow, PyTorch, and Scikit-learn, and can be accessed via a REST API.

4.3.4 Interaction with other components

Training module: While training the MLFlow Metric Logger will store and visualise pre-, or user defined metrics described above. This will allow the DT developer to assess the training progress and to intervene accordingly.

Model registry: The metrics will be stored alongside the trained models in the Model Registry. This preserves the whole training history and simplifies later comparisons and analyses of stored models.

Provenance: The Prov4ML⁵⁷ service for provenance information for AI workflows is integrated into **itwinai** as a logger, complying with the `itwinai.loggers.Logger` interface. This allows DT developers to easily log provenance information by accessing the Prov4ML functionalities from **itwinai**, potentially both during training and inference workflows.

Rucio Data Lake: The Data Lake offers a consistent distributed data management service to share data among European federated e-infrastructures, such as EuroHPC centres. This technology, once integrated with the Metric Logger, could enable DT developers to store their ML logs into the Data Lake and share them between computing sites, without requiring internet connection on compute nodes. This

⁵² <https://mlflow.org/docs/latest/tracking.html>

⁵³ <https://docs.wandb.ai/guides/track>

⁵⁴ <https://www.tensorflow.org/tensorboard>

⁵⁵ <https://mlflow.org/docs/latest/tracking/server.html>

⁵⁶ <https://docs.wandb.ai/guides/hosting>

⁵⁷ <https://github.com/HPCI-Lab/ProvML>



D6.3 Updated report on requirements and core modules functionalities

alternative is currently under exploration. More details the interTwin Data Lake developed in WP5 are given in [R4]

ML model deployment: During deployment the pre- or user defined metrics may be visualised in order to monitor if the deployed model is working correctly. Depending on the use case requirements, the DT developer can define thresholds for the metrics and should the model above or below the threshold, warning messages and automatic exception handler can be triggered.

4.4 Machine Learning model deployment

This section contains a description of the different components of the machine learning deployment module in the AI workflow engine. The architecture of the module and its interaction with the other modules of the AI workflow component is depicted in [Figure 10](#).

4.4.1 General description and functionalities

Once an ML model, like a neural network, is trained, it is served on the infrastructure as a standalone application, which receives unseen pre-processed data as input and produces the respective predictions as outputs.

This component is responsible for providing the “living” ML model of a DT, allowing anyone to query it at any time, by either requesting on-line predictions (i.e., prediction on small data, usually encapsulated in an HTTP request, in real time), or submitting batch jobs (i.e. the ML model is seen as a transformation, which is applied to a large dataset, producing another dataset of predictions as output). A DT developer or a scientist chooses a pre-trained ML model from the Models Registry and the “ML deployment component” deploys it in a container, encapsulating the minimal Python environment needed by the ML model to properly function.

D6.3 Updated report on requirements and core modules functionalities

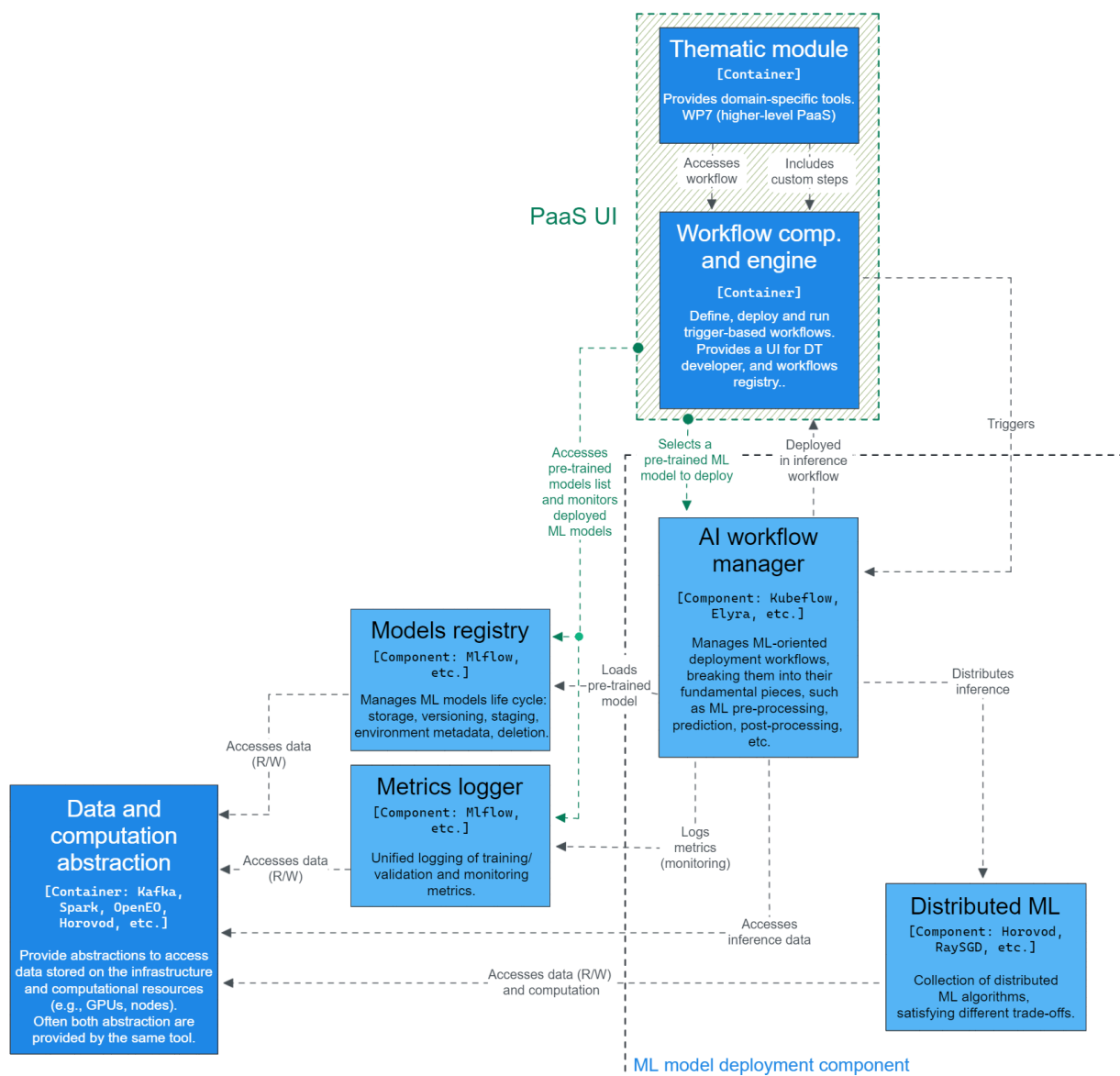


Figure 10 - - Detailed view on the architecture of the ML deployment module

This component is different from ML training in that the training component has generally a different Python environment, which, for instance, includes libraries that are needed only at training time (e.g., HPO libraries). Moreover, the Python environment used with the deployed ML model may contain deployment-specific libraries, for instance, to allow users and other modules to query the ML model for predictions, like via REST APIs. Furthermore, differently from the ML training module, the deployment of an ML model can be replicated to improve its availability when multiple queries, such as HTTP requests, are made to the same ML model. Similarly, to ML training, the deployed ML model can access resources and frameworks for distributed ML (e.g. Horovod).

A DT developer or a scientist can deploy a pre-trained ML model as a step of a broader DT workflow, which is orchestrated by the workflow composition tools and engine described in [Section 3.2](#). For instance, a minimal DT workflow including a pre-trained ML model could consist of: preprocessing of satellite images with openEO, prediction of fire risk maps with a pre-trained generative adversarial neural network (GAN), and



D6.3 Updated report on requirements and core modules functionalities

visualisation of the predictions. In this example, the pre-trained GAN can be queried by the visualisation tool to perform on-line predictions, or it could be applied to a dataset of preprocessed satellite images, in a batch processing fashion, to produce a large set of predictions all at once. In both situations, deploying the pre-trained ML model as a container allows it to be scaled up or down, according to the needs of availability for online queries or parallelization for batch processing.

4.4.2 Interfaces

The module for ML model deployment provides an API that allows the user to select and deploy a pre-trained model based on some goodness metrics. In the specific case of the selection of the pre-trained model to deploy, the DT developer can access the ML model deployment module via some web-based portal.

The ML deployment can be triggered by the advanced workflow composition through Kubernetes-like and OpenStack APIs. The OSCAR component introduced in section 3.1.1 can be also used for model serving at scale.

4.4.3 Technology stack

To begin with, the technology requirements for this component are the following:

- Interaction with the infrastructure via a Kubernetes-like API abstraction layer. Infrastructure providers may use different container management platforms, but this component expects to interact with the underlying infrastructure through some abstraction layer developed by WP5. This abstraction layer is used to deploy pre-trained models in containers as standalone services.
- On top of Kubernetes-like clusters, the deployed ML model can be triggered by some workflow orchestrators (such as [Argo workflows](#)).
- The ML model is deployed in a container (e.g., Docker) with an ad-hoc Python environment. This requires the availability of container engines.
- The pre-trained ML model is retrieved from the models registry, which shall be available on the infrastructure as part of the MLflow server (i.e. the standalone [MLflow tracking server](#)). The MLflow tracking server can be deployed on the infrastructure using its container image.

The technology stack of the ML deployment module can be summarised as follows:

- Interface with the PaaS UI and thematic module. This interface allows this component to receive the scripts to include in the deployment container, and the details of the deployment, such as unique ID of the pre-trained model to deploy, and hardware preferences.
- Client to communicate with the Models Registry. Fetches the desired ML model from the models registry using its unique ID. This could be an MLflow client.
- APIs to access the container engine, to build and deploy the image of the ML model to serve.

Once deployed, the software stack of an ML model has the following key components

D6.3 Updated report on requirements and core modules functionalities

- The model is deployed in a Python environment, including libraries to:
 - Run the pre-trained ML model (e.g., PyTorch, TensorFlow, ONNX runtime)
 - Connect to the metrics logger, like the MLflow client to contact the MLflow tracking server.
 - Communicate with the external environment (e.g., CLI, REST APIs)
 - Distributed ML for inference, such as Horovod and PyTorch DDP
- Drivers to access computing infrastructure (e.g., CUDA drivers)
- Connectors to data infrastructure, like Kafka and Rucio clients

4.4.4 Interaction with other components

The ML model deployment interacts with the following components:

Model registry: Retrieves a pre-trained model from the models registry using its unique ID. This interaction is mediated through a specific interface, which could be implemented via APIs or a client. Most likely, the models registry is going to be implemented using MLflow tracking server, therefore, the interaction with the models registry is likely to be mediated via the MLflow client. An alternative is the Kubeflow client/APIs in case the models registry is hosted on some Kubeflow instance.

Metric logger: Saves metrics concerning the deployed ML models to the logger service, for monitoring reasons. Both the ML deployment instance and the deployed ML models can access this service to constantly update the user (e.g., the DT developer) about their status.

Distributed ML: Accesses distributed ML resources to scale the inference computations when the deployed ML model is large, or to provide an alternative method to scale up when the volume of requests is large. Distributed ML provides an abstraction layer to scale the computation on multiple GPUs.

Computing infrastructure (interLink): As described above, a pre-trained ML model is deployed on the infrastructure as a container, therefore, it has to interact with the infrastructure provider using the Kubernetes-like APIs and the offloading to external clusters provided by interLink. Furthermore, the computing infrastructure will provide computing resources such as GPUs (preferably) or CPUs, to run the deployed ML model.

Data infrastructure: An ML model can be deployed to transform a large dataset, in a batch processing manner. Alternatively, the deployed ML model can take advantage of (near) real-time data acquisition to perform real-time predictions on input data streams, producing an output stream of predictions by using the DCNios component detailed in section 3.1.1

Thematic module / PaaS UI (WP7): Similarly, to the ML training component, the ML deployment module is instantiated by the user (e.g. the DT developer) through the UI provided by the DTE PaaS by means of Jupyter Notebooks. The user selects the pre-trained model to deploy from the models registry, and defines other deployment configurations, such as the number of replicas, or hardware preferences. An important



D6.3 Updated report on requirements and core modules functionalities

aspect is the definition of connectors to the pre-trained dataset: the DT developer shall provide a Python script to load the dataset items in memory, by defining the logic to parse and identify the items of the dataset stored on disk, like, the logic defined by extending Pytorch's Dataset class. Another similarity with the ML training module is that the deployment of an ML model provides the user (e.g., the DT developer) with statistics concerning the ML model, through the metrics logger component, for real-time monitoring,

Advanced workflow composition: The ML deployment can be triggered by the advanced workflow composition. Both components are likely to be deployed as containers on the infrastructure, thus the trigger received from the advanced workflow composition should be in the form of Kubernetes-like or Apache Airflow-like APIs. The interaction could be via HTTP requests, message queues, or other.

Quality and uncertainty tracing. The Software Quality Assessment as a Service logic provided by the "Quality and uncertainty tracing" module, allows testing (micro)services in a black-box manner. Once a pre-trained ML model is deployed as a container as a standalone service, it can be easily tested in a black-box manner. The DT developer can provide use case-specific test cases to validate the performance of the model.



5 Components for Quality Assurance

The main goal of the Quality Assurance (QA) component of the DTE is to perform domain-specific (functional, behavioural) validation, in a black-box manner, avoiding clashes with any complementary evaluation task, such as the validation of performance metrics done as part of the AI workflows subsystem. However, based on the collected requirements summarised in [Section 2](#), current assessment capabilities might be extended to include the evaluation of specific criteria related to (i) data quality (such as those related to the [data quality dimensions](#)) and (ii) AI & ML validation. In addition, the interfaces of the SQAaaS platform are expected to grow in order to integrate with data spaces and workflow engines.

5.1 Software Quality Assurance as a Service

The DTE features a specific module for quality assurance (QA) that aims at tackling the early validation of the DTs, before being deployed as a “living DT”. The main module that provides this functionality is the Software Quality Assurance as a Service (SQAaaS).

5.1.1 General Description and functionalities

The SQAaaS platform provides graphical and programmatic interfaces to compose continuous integration and delivery (CI/CD) pipelines. These pipelines might be then used as fail-fast systems and/or quality gates, so that any indication of a failure is detected promptly, which enables to adjust the runtime behaviour of the workflows composed through the DTE.

By means of the SQAaaS platform, the DT developer can choose among a set of special-purpose, open-sourced libraries and tools to assess QA criteria relative to models and/or data. The list of criteria (and their means of validation) is then wrapped into CI/CD pipelines, which once executed, act as quality gates during the validation of a DT workflow.

Accordingly, the resultant CI/CD pipelines are meant to be triggered both on-demand, e.g. as the final acceptance check of a pre-trained ML model before moving it to production (see also the [components for AI workflows](#)), or as a response to events, such as those generated by data ingestion systems (integration with [workflow composition tools](#)) or by repository platforms as a result of new changes in the source code of the model as depicted in [Figure 11](#).

The primary components in the SQAaaS architecture are summarised below.

SQAaaS API server

This is the key component of the SQAaaS platform. It provides two central building blocks: i) the composition of CI/CD pipelines, and ii) the quality assessment of three types of digital assets: source code, (web)services and data. Whilst the latter, known as Quality Assessment and Awarding, provides a more general and comprehensive analysis (and crediting) of a given digital object through the execution of a fixed set of



D6.3 Updated report on requirements and core modules functionalities

criteria and tools, the former, coined as Pipeline as a Service, facilitates the task of tailoring CI/CD pipelines by selecting the required criteria and the tools to be used in each stage of the pipeline.

SQAaaS tooling & reporting

The SQAaaS platform has built-in support for a series of open source tools that cover the validation of individual quality criteria. Tools are selected based on the popularity and adequate support within the given community. As an example, `pycodestyle` (Python's PEP8 checker) is one of the supported tools to cover the "code style"-related criteria. The support for a given tool is accompanied by an associated reporting plugin that is in charge of validating its output.

New libraries and tools will enrich the current catalogue in order to deal with the requirements of the DTE implementation, both in terms of data quality and model validation. These tools will assist DT developers to uncover issues in early stages with extended capabilities on data integrity, cleansing, formatting, profiling, and FAIR compliance, as well as model performance. Examples of tools already identified are FAIR-EVA,⁵⁸ DeepChecks,⁵⁹ and Great Expectations.⁶⁰

Jenkins Pipeline Library

Jenkins Pipeline Library (JePL) provides the integration with the CI solution (Jenkins) through YAML descriptions, where the pipeline work is defined. The file is structured according to the set of quality criteria being used. The library relies on containers to set the environment for the execution of the checks clustered in each criterion being defined.

Uncertainty Quantification (UQ)

Uncertainty Quantification (UQ) aims to enhance the reliability of ML/AI models by establishing trust in their predictions. This process helps scientists determine the extent to which they can trust and accept the results obtained from scientific workflows involving ML/AI models. To achieve this, UQ requires probabilistic distributions rather than just deterministic predictions or forecasts. In classification problems, the model should produce discrete probabilistic distributions, while in regression models, it should generate continuous probabilistic distributions.

This component will be integrated with SQAaaS to be triggered programmatically, helping the end user understand the uncertainties inherent in the predictions. This enables users to determine whether the results can be trusted or require further investigation. The integration will be achieved by containerizing the workflow, which provides various tools and metrics for uncertainty quantification. This container will be triggered whenever a new pre-trained model is available in the registry, utilising the tooling and reporting components to deliver reports and feedback on the results. Also,

⁵⁸ https://github.com/EOSC-synergy/FAIR_eva

⁵⁹ <https://github.com/deepchecks/deepchecks>

⁶⁰ https://github.com/great-expectations/great_expectations



D6.3 Updated report on requirements and core modules functionalities

integrate it with `itwinai` for allowing DT developers to gain an initial understanding of potential uncertainties before deploying their models.

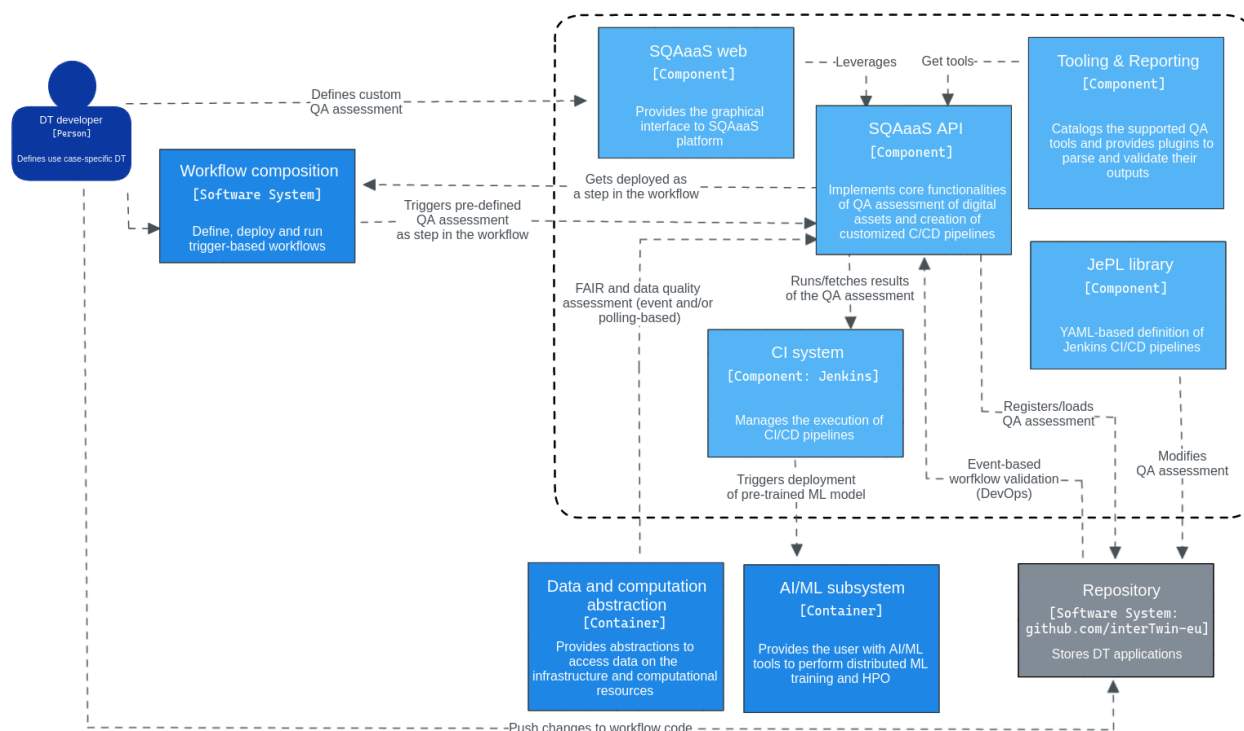


Figure 11 - Detailed view on the architecture of the quality assurance module

5.1.2 Interfaces

The SQAaaS platform provides an API that allows the composition and triggering of QA assessments, which shall be leveraged by this component. In the particular case of the composition of a customised QA assessment or CI/CD pipeline, the DT developer has the additional possibility of accessing the SQAaaS platform through the web portal.

5.1.3 Technology stack

The technology stack used to deploy and operate the SQAaaS platform is described below.

CI system

This component is where the CI/CD pipelines described through the JePL library run (i.e., Jenkins). The SQAaaS cloud instance already provides a Jenkins CI server with a set of agents to distribute the work. The SQAaaS can also be deployed on-premises via Kubernetes. In this case, a Kubernetes Jenkins operator is deployed in the cluster.

Code repository platform

The SQAaaS platform relies on a Git-based repository platform to operate (i.e., store, track, and access) the pipelines being created. The SQAaaS API stores each defined QA assessment as a JePL pipeline in an individual code repository. The API is currently



D6.3 Updated report on requirements and core modules functionalities

integrated with the GitHub platform, where the InterTwin project has already an organisation available⁶¹. This approach facilitates the maintenance of the QA assessment, acting as a catalogue of JePL pipelines so that they can be re-defined and re-triggered.

Besides using GitHub for the operational needs of the SQAaaS platform, it can be additionally leveraged to store workflow definitions created by the workflow composition tools, and thus, configure the integration with the SQAaaS' CI server in order to validate them upon changes.

Container registry

The libraries and tools that will be used in the CI/CD pipelines, both the ones having built-in support and/or the specifically selected by the user, shall be available as container images in some Docker registry.

5.1.4 Interaction with other components

Workflow composition: The workflow composition tool will allow the DT developer to add QA assessments in one or multiple steps during the workflow composition. The added value of following this approach is the early detection of any issue, and thus, having the capability to interrupt the workflow execution as soon as any of the pre-defined quality standards are not met (quality gate).

ML model deployment: Before performing the black-box validation, the model needs to be deployed. In the case of ML-based models, the AI/ML module will provide the "ML model deployment component" that fetches the appropriate version of the model from the registry and deploys it for further validation.

Data acquisition and event-driven triggering of workflows: The event-driven architecture implemented by Task 6.1 can be leveraged to perform QA work on the data to be used to train and operate the DT (DataOps-like approach). For instance, if connected with a data lake or registry (aka Data Computation and Abstraction) it would facilitate the quality attributes of the data stored there. Polling-based solutions can also be considered.

⁶¹ <https://github.com/interTwin-eu>



6 Components for Big Data Analytics

An overview of the deployment of the Data Analytics tools is provided in [Section 6.1](#), while subsequent sections will detail the specific tools that are involved in Big Data Analytics. Specifically, [Section 6.2](#) describes Kubernetes clusters, [Section 6.3](#) presents Horovod environments, [Section 6.4](#) covers Kubeflow environments, [Section 6.5](#) covers Ophidia , while [Section 6.6](#) deals with openEO clusters, [Section 6.7](#) describes Airflow clusters, [Section 6.8](#) describes EOEPKA ADES clusters, [Section 6.9](#) shows ecFlow clusters, [Section 6.10](#) covers MLFlow. Finally, [Section 6.11](#) covers the yProv server.

6.1 Deployment of Big Data Analytics tools

This section describes the deployment module of the Big Data Analytics subsystem. [Figure 12](#) shows how the modules of the Big Data Analytics deployment work together. Each of these modules is described in more details below.

6.1.1 General description and functionalities

The goal of the deployment layer is to create a set of topology templates and recipes for general-purpose data analytic environments to be deployed on demand on top of the cloud resources.

The cloud topology templates will be created using the OASIS TOSCA Simple YAML specification [\[R2\]](#). They will describe all the virtual resources and the software components required to deploy the final application. Furthermore, they will provide the user with a set of input parameters enabling them to customise the application configuration.

All the defined templates will be stored in a public repository that will be made available to the users by the Orchestrator Dashboard. It will render the templates, enabling the users to set the defined input parameters and will contact the PaaS Orchestrator that will be in charge of processing the TOSCA template and creating all the required cloud resources, configuring the selected big analytics tool, and making it available for the final user. Finally, the Dashboard will show the TOSCA specified output values with the required information to connect with the application (endpoints, credentials, etc.).

The artefacts with the recipes to configure the desired big analytics tools will be described using the Ansible Language (in Ansible terms, “playbooks”). All the ansible playbooks referenced in the TOSCA templates will also be stored in a public repository. Moreover, the main installation recipes will be packaged as Ansible roles thus enhancing their reusability in different playbooks. Also, all the defined Ansible roles will be stored in public repositories and made available for the playbooks using the ansible galaxy tool.



D6.3 Updated report on requirements and core modules functionalities

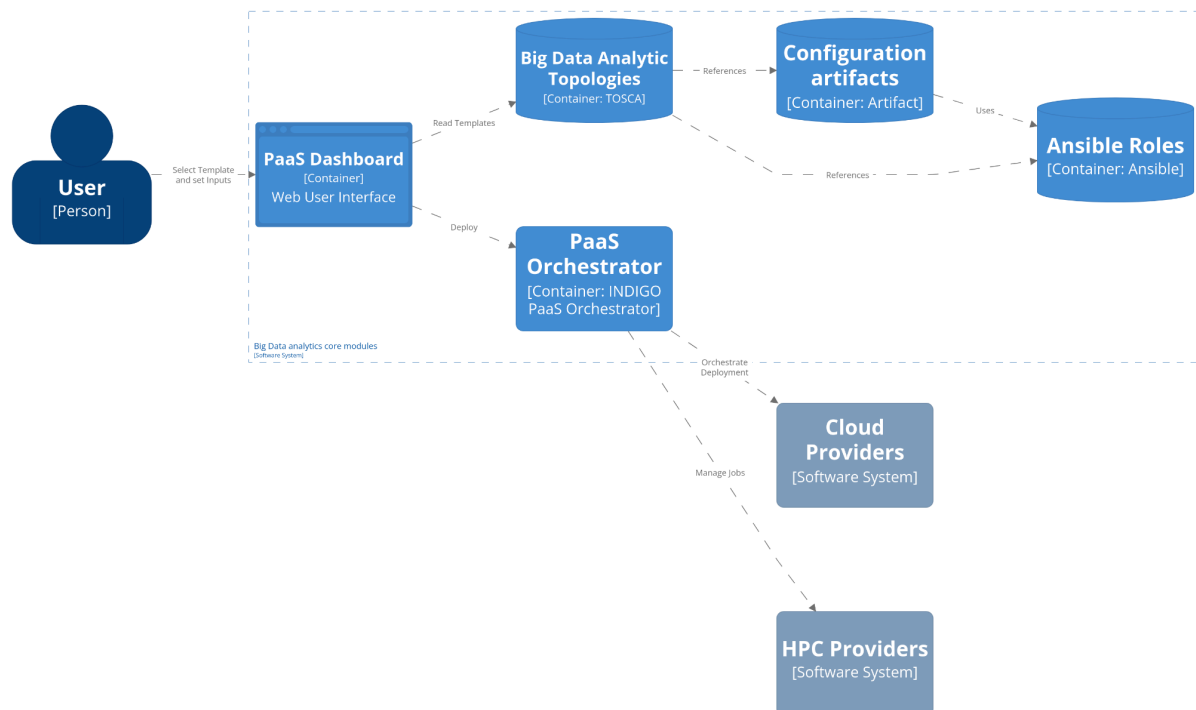


Figure 12 - General architecture of the data analytics tools

PaaS Orchestrator

The Platforms as a Service (PaaS) Orchestrator allows federating heterogeneous resource providers and orchestrates the deployment of TOSCA templates, selecting the best provider according to criteria like the data location, the SLA and monitoring information. It provides a set of APIs to create, monitor and manage the deployments.

Orchestrator dashboard

The PaaS Orchestrator dashboard is a web application that enables users to easily interact with the services of the PaaS, particularly the Orchestrator, to create TOSCA-based deployments. The dashboard provides a user-friendly interface for managing and monitoring deployments.

Big Data Analytics TOSCA templates repository

This repository will store the set of TOSCA templates with the definition of the software component and the underlying virtual infrastructures needed to execute the Big Data Analytics tools. Some of these TOSCA templates will require the creation of new TOSCA custom types to define particular elements (mainly software components) of the cloud topology. These new custom types will also be defined in a separate YAML file but stored in the same repository.

Configuration artefacts repository

TOSCA templates define the topology and the set of components needed to fully deploy an application, but these templates refer to a set of artefacts with the recipes needed to install/configure every software component needed. These artefacts will also be stored



D6.3 Updated report on requirements and core modules functionalities

in a public repository (or set of them). These artefacts will be defined using the Ansible DSL language.

Ansible roles repositories

The artefacts defined to install/configure software components may also use Ansible roles. Ansible roles are ways of getting content contributions from various Ansible Developers, they are similar to libraries in programming languages. These roles use a central service provided by Ansible (Ansible Galaxy) to search for the required roles to be used in an Ansible playbook, but the actual recipes are stored in public GitHub repositories.

6.1.2 Interfaces

For the final user the Orchestrator Dashboard offers a graphical web UI where any user without knowledge about TOSCA can easily deploy their own Big Data Analytics tools with the underlying virtual infrastructure required.

The PaaS Orchestrator also offers a REST API (<https://indigo-dc.github.io/orchestrator/>) that can be used programmatically to create the required virtual infrastructures.

6.1.3 Technology stack

The PaaS Orchestrator needs the presence of the following services:

SLA Manager

The SLA Manager (SLAM) stores all Service Level Agreements (SLAs) of the user and allows for their programmatically retrieval.

Configuration Management DataBase

The Configuration Management DataBase (CMDB) is where the Cloud sites store the necessary information for delivering their services and/or resources. Information like service endpoints, contact emails, people responsible for the services, planned/unplanned interventions, and downtime, etc. The information stored in the CMDB is regularly reviewed and validated.

Monitoring System

The monitoring system collects and generates Availability and Reliability metrics for the services registered in CMDB, via monitoring probes deployed at each Cloud site. It allows querying monitoring metrics through a REST interface.

Cloud Provider Ranker

Cloud Provider Ranker (CPR) receives all the information retrieved from the aforementioned services and provides the ordered list of the best sites.

Infrastructure Manager

Infrastructure Manager⁶² (IM) is the component that actually deploys and configures the virtual infrastructure once the site has been selected.

⁶² <https://www.grycap.upv.es/im/index.php>



6.1.4 Interaction with other components

The PaaS orchestrator by means of the Infrastructure Manager will interact with the set of cloud providers (either resources of the testbed infrastructure, the EGI Federated cloud, on-premises cloud or public cloud offerings) to deploy the virtual resources needed.

The PaaS orchestrator also allows the deployment of dockerized services and jobs on Mesos and Kubernetes clusters on HPC providers.

6.2 Elastic Kubernetes clusters on demand

This section describes the Kubernetes clusters as part of the Big Data Analytics subsystem.

6.2.1 General Description and functionalities

One of the main components required for deploying other data analytic environments is a container orchestration platform based on Kubernetes (K8s). This will facilitate the deployment of containerised data analytic components. The innovative aspect of this component will be the ability to extend and shrink the number of nodes of the K8s cluster according to the workload. This way the number of nodes in the cloud infrastructure will be minimised and adjusted to the real need.

The variation in the number of nodes of a Kubernetes cluster is performed according to two criteria:

- Powering on new nodes and adding them to the Kubernetes cluster. This is triggered when an object that has indicated a request on resources and it entered in the “pending” state due to the lack of resources. After a period of time defined in the configuration of CLUES, the deployment of a new node is triggered so the object can be scheduled.
- Powering off nodes and removing them from the Kubernetes clusters. Idle nodes that remain without processing objects allocated are powered off after a given time threshold. Powered off nodes automatically disappear from the available nodes of Kubernetes.

6.2.2 Interfaces

Kubernetes objects are managed through the Kube apiserver using the Kubernetes API⁶³. This API permits the management of any type of Kubernetes object coded into JSON or YAML formats. Authentication can be performed by means of tokens, which could be defined on the instantiation of the cluster.

In order to facilitate the management of Kubernetes objects, two additional applications can be optionally deployed with the Kubernetes cluster (supported by the TOSCA recipe). On the one hand, the Kubernetes Dashboard⁶⁴ is a Graphical User Interface

⁶³ <https://kubernetes.io/docs/concepts/overview/kubernetes-api/>

⁶⁴ <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>



D6.3 Updated report on requirements and core modules functionalities

(GUI) that can manage any type of K8s object and provides a monitoring system to explore the usage of resources. This interface facilitates the management of K8s objects especially if a Command Line Interface (e.g., kubectl) is not available. On the other hand, the TOSCA recipe is enabled to install Helm and Kubeapps, so Helm charts⁶⁵ can be deployed from the GUI of Kubeapps.

6.2.3 Technology stack

The cluster will be described as a TOSCA Infrastructure as Code blueprint, which could be deployed on top of the infrastructure through either the PaaS orchestrator or the Infrastructure Manager. The infrastructure recipe is available in GitHub⁶⁶. This recipe comprises the following components:

Kubernetes

Kubernetes (K8s) is an open source platform for automating the deployment, scaling, and management of containerized applications.⁶⁷

Elastic Cloud Computing Cluster

The Elastic Cloud Computing Cluster (EC3)⁶⁸ is a platform that allows creating elastic virtual clusters on top of Infrastructure as a Service (IaaS) providers, either public (such as Amazon Web Services, Google Cloud, or Microsoft Azure) or on-premises (such as OpenStack or OpenNebula). It uses CLUES as the elasticity management component of EC3. The CLUES system integrates with the cluster management middleware, such as container orchestrators, batch-queuing systems, or cloud infrastructure management systems, by means of different connectors.

6.2.4 Interaction with other components

The on-demand elastic Kubernetes cluster module acts as the framework using which Kubeflow clusters (see [Section 6.4](#)), Ophidia clusters (see [Section 6.5](#)), openEO clusters (see [Section 6.6](#)), Airflow clusters (see [Section 6.7](#)), EOEPKA ADES clusters (see [Section 6.8](#)) and yProv server (see [Section 6.11](#)) will be deployed.

6.3 Horovod environment

This section details how the Horovod environment can be included as part of the Big Data Analytics subsystem.

6.3.1 General Description and functionalities

Horovod can be used to run distributed deep learning workloads. It provides a framework for scaling the deep learning training jobs allowing users to efficiently train large deep learning models on big data sets.

⁶⁵ <https://helm.sh/es/docs/topics/charts/>

⁶⁶ https://github.com/grycap/im-dashboard/blob/master/tosca-templates/kubernetes_elastic.yaml

⁶⁷ <https://kubernetes.io/es/>

⁶⁸ <https://eosc-portal.eu/news-and-events/news/elastic-cloud-compute-cluster-ec3>



D6.3 Updated report on requirements and core modules functionalities

Horovod

Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet⁶⁹. It provides an easy-to-use interface for scaling deep learning training jobs across multiple nodes and GPUs. Horovod can be used to train large deep learning models on big data sets.

6.3.2 Interfaces

Once deployed, users can interact with the Horovod environment by submitting a script written in Python that includes the Horovod libraries.

6.3.3 Technology stack

Horovod is installed through a set of Ansible recipes⁷⁰.

6.3.4 Interaction with other components

Components for AI Workflow: Horovod is one of the distributed training frameworks analysed.

6.4 KubeFlow clusters

This section details how Kubeflow clusters are being used in the Big Data Analytics subsystem.

6.4.1 General Description and functionalities

Kubeflow is a free, open-source machine learning platform that allows machine learning pipelines to orchestrate complicated workflows running on Kubernetes. Kubeflow is a collection of cloud native tools for all stages of the machine learning lifecycle, including data exploration, feature preparation, model training/tuning, model serving, model testing, and model versioning. Each component of Kubeflow can be deployed separately.

The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable, and scalable. Its goal is not to recreate other services, but to provide a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures.

Kubeflow translates steps in the data science workflow into Kubernetes jobs. It is a platform for data scientists who want to build and experiment with ML pipelines. Kubeflow is also for ML engineers and operational teams who want to deploy ML systems to various environments for development, testing, and production-level serving.

It includes services to create and manage interactive Jupyter notebooks. You can customise your notebook deployment and your compute resources to suit your data science needs.

⁶⁹ <https://horovod.ai/>

⁷⁰ <https://github.com/grycap/tosca/tree/main/artifacts/horovod>



D6.3 Updated report on requirements and core modules functionalities

Kubeflow provides training operators for several ML frameworks such as TensorFlow (TFJob), PyTorch (PyTorchJob), MXNet (MXJob), XGBoost (XGBoostJob), and PaddlePaddle (PaddleJob). Furthermore, it supports a TensorFlow Serving container to export trained TensorFlow models to Kubernetes. Kubeflow is also integrated with Seldon Core, an open source platform for deploying machine learning models on Kubernetes, NVIDIA Triton Inference Server for maximised GPU utilisation when deploying ML/DL models at scale, and MLRun Serving, an open source serverless framework for deployment and monitoring of real-time ML/DL pipelines. Kubeflow Pipelines is a comprehensive solution for deploying and managing end-to-end ML workflows.

Furthermore the recipe has been extended to also include an integrated MLFlow instance that can be used to track the ML workflows.

6.4.2 Interfaces

Once Kubeflow has been deployed, users can interact with it through the Kubeflow Dashboard. The dashboard provides a central user interface for managing and monitoring Kubeflow resources such as notebooks, pipelines, and experiments.

Users can also use the Kubeflow command line interface (CLI) to interact with Kubeflow resources.

6.4.3 Technology stack

KubeFlow is installed through an Ansible recipe⁷¹ on top of an elastic Kubernetes cluster.

6.4.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

Components for AI Workflow: Kubeflow is one of the solutions analysed to launch the AI pipeline in cloud/HPC infrastructure.

6.5 Ophidia Cluster

This section covers Ophidia clusters as part of the Big Data Analytics subsystem.

6.5.1 General description and functionalities

Ophidia provides support for data-intensive analysis exploiting advanced parallel computing techniques and smart data distribution methods. It exploits an array-based storage model and a hierarchical storage organisation to partition and distribute multidimensional scientific datasets over multiple nodes. The Ophidia analytics framework can be exploited in different scientific domains (e.g., Climate Change, Earth

⁷¹ <https://raw.githubusercontent.com/grycap/ec3/tosca/tosca/artifacts/kubeflow.yml>



D6.3 Updated report on requirements and core modules functionalities

Sciences, Life Sciences) and with very heterogeneous sets of data. Further details on the framework can be found in Section 3.2.3.

Ophidia is offered using a JupyterLab instance, deployed on top of a Kubernetes cluster, jointly with a large set of pre-installed Python libraries and a ready-to-use Ophidia HPDA framework instance for running data manipulation, analysis, and visualisation.

6.5.2 Interfaces

Once deployed, users can interact with the JupyterHub interface. JupyterHub provides a web-based interface where users can launch a Jupyter server, use notebooks (e.g., the DT applications) and access computational resources. From within a Jupyter notebook, users can run the Ophidia HPDA framework to perform computations.

PyOphidia, the Ophidia Python bindings, can be used together with other libraries from the scientific Python ecosystem for implementing data analytics applications.

6.5.3 Technology stack

Ophidia is installed through an Ansible recipe⁷² on top of an elastic Kubernetes cluster.

6.5.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

Provenance: Ophidia will also interact with the provenance module (deployed as a containerized module) to track lineage metadata during (or at the end of) the analytics workflow execution. The integration between the two components is described in Section 3.3.

Workflow Composition: Ophidia is one of the proposed solutions as Workflow Management engines requested/suggested by the user communities.

6.6 openEO clusters

This section details how openEO clusters are being used in the Big Data Analytics subsystem.

6.6.1 General Description and functionalities

Earth Observation data are becoming too large to be downloaded locally for analysis. Also, the way they are organised (as tiles, or granules: files containing the imagery for a small part of the Earth and a single observation date) makes it unnecessarily complicated to analyse them. The solution to this is to store these data in the cloud, on compute back-ends, process them there, and browse the results or download resulting figures or numbers.

⁷² <https://github.com/grycap/ec3/blob/tosca/tosca/artifacts/enes/enes.yml>



D6.3 Updated report on requirements and core modules functionalities

6.6.2 Interfaces

openEO develops an open application programming interface (API) that connects clients like R, Python and JavaScript to big Earth observation cloud back-ends in a simple and unified way.

With such an API, each client can work with every back-end, and it becomes possible to compare back-ends in terms of capacity, cost, and results (validation, reproducibility).

6.6.3 Technology stack

OpenEO is installed through a Helm Chart⁷³ on top of an elastic Kubernetes cluster. Underneath it also relies on Argo-Workflows and Redis

6.6.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

Workflow Composition: openEO is one of the proposed solutions as Workflow Management engines requested/suggested by the user communities.

6.7 Airflow clusters

This section details how Airflow clusters are being used in the Big Data Analytics subsystem.

6.7.1 General Description and functionalities

Apache Airflow is a platform created by the community to programmatically author, schedule and monitor workflows.

6.7.2 Interfaces

Once Airflow has been deployed, it exposes a web interface the user can use to launch and manage the workflow executions. It is configured with a synchronisation with an external git repository where the users can edit the available workflows.

6.7.3 Technology stack

Airflow is installed through a Helm Chart⁷⁴ on top of an elastic Kubernetes cluster. Furthermore Elasticsearch will be also installed to enable store logging information.

6.7.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources

⁷³ <https://github.com/eodcgmbh/charts/tree/main/eodc/openeo-argo>

⁷⁴ <https://airflow.apache.org/docs/helm-chart/stable/index.html>



D6.3 Updated report on requirements and core modules functionalities

of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

Workflow Composition: Airflow is one of the proposed solutions as Workflow Management engines requested/suggested by the user communities.

Components for AI Workflow: Kubeflow is one of the solutions analysed to launch the AI pipeline in cloud/HPC infrastructure.

6.8 EOEPKA ADES clusters

This section details how EOEPKA ADES clusters are being used in the Big Data Analytics subsystem.

6.8.1 General Description and functionalities

EOEPKA ADES⁷⁵ is an open source processing platform that provides a server implementation of the Web Processing Service (WPS) (1.0.0 and 2.0.0) and the OGC API - Processes standards published by the OGC. It contains a minimal set of ready-to-use services that can be used as a base to create more useful services.

6.8.2 Interfaces

EOEPKA ADES provides Web Processing Service (WPS) (1.0.0 and 2.0.0) and the OGC API Processes standards published by the OGC

6.8.3 Technology stack

EOEPKA ADES is installed through an Ansible recipe⁷⁶ on top of an elastic Kubernetes cluster. It combines ZOO-Project DRU⁷⁷, for the computing part and MinIO for the storage.

6.8.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

6.9 ecFlow clusters

This section details how ecFlow clusters are being used in the Big Data Analytics subsystem.

6.9.1 General Description and functionalities

ecFlow is a workflow management system developed and maintained by the European Centre for Medium-Range Weather Forecasts (ECMWF). It is designed to handle complex

⁷⁵ <https://github.com/EOEPKA>

⁷⁶ https://raw.githubusercontent.com/grycap/tosca/main/artifacts/ades_k8s.yml

⁷⁷ <https://github.com/ZOO-Project/ZOO-Project>



D6.3 Updated report on requirements and core modules functionalities

workflows, particularly in the field of weather forecasting and numerical weather prediction.

6.9.2 Interfaces

ecFlow server provides an API to be consumed by ecFlow client to schedule and execute workflow jobs.

6.9.3 Technology stack

ecFlow will be installed through an Ansible recipe.

6.9.4 Interaction with other components

Workflow Composition: ecFlow is one of the proposed solutions as Workflow Management engines requested/suggested by the user communities.

6.10 MLFlow server

This section details how a MLFlow server are being used in the Big Data Analytics subsystem.

6.10.1 General Description and functionalities

MLFlow is an open source platform for the complete ML lifecycle management, including experimentation, reproducibility, deployment, and monitoring.

6.10.2 Interfaces

MLFlow server provides an API to interact with the model registry and logger and a GUI to show the results to the users.

6.10.3 Technology stack

MLFlow is installed through an Ansible recipe⁷⁸ that relies on docker compose to deploy all MLFlow related services.

6.10.4 Interaction with other components

Components for AI Workflow: MLFlow is used as a metrics logger tool.

Model Registry: MLFlow is used as a model registry tool.

6.11 yProv server

This section details how a yProv server is being used in the Big Data Analytics subsystem.

6.11.1 General Description and functionalities

yProv service is a cross-domain service focusing on the management of provenance documents. It consists of 3 components: (i) Web Service front-end, (ii) Graph database engine back-end (Neo4J) and (iii) a Command Line Interface (CLI).

⁷⁸ https://github.com/grycap/tosca/blob/main/artifacts/mlflow_compose.yml



D6.3 Updated report on requirements and core modules functionalities

The authentication/authorization is based on JSON Web Token (JWT), the service exposes a RESTful API (OpenAPI) providing an easy way to interact with the service and manage provenance information according to the W3C PROV family of standards.

6.11.2 Interfaces

The yProv API has been designed by following the REST principles. A list of methods and the corresponding action for each HTTP verb is available on the service repo as a Swagger document.

More in detail, five classes of resources have been identified, namely documents, entity, activity, agent and relation.

Document is the abstraction of the full provenance information associated with a workflow. For each abstract document there is a one-to-one association with a graph database. Entity, activity and agent are sub-resources of the document resource, and they allow to perform the CRUD operations on the three types of elements stored in a provenance graph according to the W3C PROV data model, thus ensuring a fine-grain control on each node stored in the graph database; similarly, relation allows to act on each single relationships of the graph.

6.11.3 Technology stack

yProv can be installed through a Helm Chart⁷⁹. It can be deployed in the cloud on a Kubernetes cluster thanks to the availability of the respective docker containers.

6.11.4 Interaction with other components

Elastic Kubernetes clusters: This module will be deployed on top of the Elastic Kubernetes framework described earlier. It requires the PaaS orchestrator or the Infrastructure Manager for being deployed on the resource provider (either resources of the interTwin testbed infrastructure, the EGI Federated cloud, on-premises cloud, or public cloud offerings).

Components Advanced Workflow Composition: yProv is the tool used for storing provenance data in scientific workflows

Ophidia Cluster: testing of Ophidia with yProv has been performed for provenance tracking within climate analytics workflows. Examples of use cases include the inference pipelines in extreme events digital twins.

MLFlow server: Prov4ML can interact with MLFlow server thus enriching the opportunities for the user to manage and view metrics tracked in terms of provenance during the training process.

Other ongoing integration activities relate to the yProv & itwinai framework as well as the yProv & SQAaaS platform. The former activity has been piloted by integrating provenance into the HEP Detector Simulation DT application whereas the latter is expanding SQA for yProv by integrating unit tests into CI/CD pipelines managed via SQAaaS.

⁷⁹ <https://github.com/HPCI-Lab/yProv/tree/main/cloud>



7 Conclusions

The second version of the interTwin Digital Twin Engine (DTE) core modules has been designed taking into consideration the evolution from the Digital Twin applications in the initial 20 months of the project and the maturity of the technological solutions available. An update on the core requirements addressing more specifically questions such as Data Fusion has been included as part of this deliverable.

The design of the modules was guided by the C4 methodology [R2], providing clear insights into the Containers, Components and Connectors that make up the DTE core modules. In the current state DTE developers can pick from the set of core modules the desired components to incorporate features such as advanced workflow composition, quality assurance models, AI/ML tools, big data analytics tools, etc.

The design also includes a list of technologies that have been analysed and selected for integration and extension/customisation. Overall effort has been placed specially in integration work to incorporate to the set of available core modules the required interoperability features. We want to stress the coordination and development effort made project-wide to come up with solutions that are deployable in infrastructures that serve different computing models, such as Cloud or HPC-batch oriented processing.

The next deliverables on WP6 (D6.4) is due in January 2025, with the description of the final release of the core components described here.

8 References

Reference	
No	Description / Link
R1	PROV-DM: The PROV Data Model , W3C Recommendation, 30 April 2013, Luc Moreau, University of Southampton. Paolo Missier, Newcastle University https://www.w3.org/TR/2013/REC-prov-dm-20130430/
R2	TOSCA Simple Profile in YAML Version 1.3, OASIS Standard. 26 February 2022, Matt Rutkowski, Chris Lauwers. Claude Noshpitz, Calin Curescu https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html
R3	C4 model in a Software Engineering subject to ease the comprehension of UML and the software . A. Vázquez-Ingelmo, A. García-Holgado and F. J. García-Peñalvo, 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 2020. DOI: 10.1109/EDUCON45650.2020.9125335
R4	interTwin D5.1 First Architecture design and Implementation Plan Diego Ciangottini, Paul Millar, Liam Atherton, Marica Antonacci, Daniele Spiga, Andrea Manzi, Renato Santana, David Kelsey, Adrian Coventry, & Shiraz Memon. DOI: https://doi.org/10.5281/zenodo.10417147
R5	interTwin D3.4 Blueprint architecture, functional specifications, and requirements analysis second version Bardaji, R., Manzi, A., Rodero, I., Geenen, T., & Warde, A. DOI: https://doi.org/10.5281/zenodo.10650440
R6	interTwin D6.1 Report on requirements and core modules definition Isabel Campos, Donatello Elia, Germán Moltó, Ignacio Blanquer, Alexander Zoechbauer, Eric Wulff, Matteo Bunino, Andreas Lintermann, Rakesh Sarma, Pablo Orviz, Alexander Jacob, Sandro Fiore, Miguel Caballer, Bjorn Backeberg, Mariapina Castelli, Levente Farkas, & Andrea Manzi. DOI: https://doi.org/10.5281/zenodo.10417153

