D6.1 Report on requirements and core modules definition

Status: UNDER EC REVIEW

Dissemination Level: public

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.

Document Description

D6.1 Report on requirements and core modules definition			
Work Package number WP6			
Document type	Deliverable		
Document status	UNDER EC REVIEW	**Version**	1
Dissemination Level	Public		
Copyright Status	This material by Parties of the interTwin Consortium is licensed under a [Creative Commons Attribution 4.0 International License]().		
Lead Partner	CSIC		
Document link	**https://documents.egi.eu/document/3950**		
DOI	**https://doi.org/10.5281/zenodo.8036987**		

D6.1 Report on requirements and core modules definition

Author(s)	- Isabel Campos (CSIC) - Donatello Elia (CMCC) - Germán Moltó (UPV) - Ignacio Blanquer (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) - Levente Farkas (EGI) - Andrea Manzi (EGI)
Reviewers	- Sandro Fiore (UNITN) - Marcin Płóciennik (PSNC)
Moderated by:	- Andrea Manzi (EGI) - Sjomara Specht (EGI)
Approved by	Christian Pagé (CERFACS) on behalf of TCB

D6.1 Report on requirements and core modules definition

Revision History			
Version	**Date**	**Description**	**Contributors**
V0.1	12/03/2023	ToC	Isabel Campos (CSIC)
V0.2	25/05/2023	First draft ready for review	Isabel Campos (CSIC) and the other authors
V0.3	26/05/2023	Version containing reviewers comments	Isabel Campos (CSIC) and the other authors
V0.4	30/05/2023	Version addressing reviewers comments	Isabel Campos (CSIC) and the other authors
V0.5	05/06/2023	Version ready for TCB approval	Isabel Campos (CSIC) and the other authors
V0.6	08/06/2023	Version approved by TCB	Christian Pagé (CERFACS)
V1.0	12/06/2023	**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 is 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

Terminology / Acronyms: **https://confluence.egi.eu/display/EGIG**

D6.1 Report on requirements and core modules definition

Table of Contents

D6.1 Report on requirements and core modules definition

Table of Figures

D6.1 Report on requirements and core modules definition

Executive summary

This deliverable provides an overview of the global requirements that the target users of a Digital Twin Engine (DTE) have identified, versus the available technical solutions to satisfy them. The requirements have been matched with the core modules of the solutions that can fulfil those requirements, at least in a first approximation. Obviously this being an early deliverable we expect that both requirements and core modules available are refined in the course of the project.

The first part of the document is a summary of the high-level goals and general features/modules required by each application. The subsequent sections of the document will detail the DTE core components that will provide these features.

The first core feature of a DTE is the ability to define and execute a workflow that will take advantage of the other components to build out the functionalities the application requires. Regarding workflow composition modules, there are two very general features demanded by most DTE developers: support for AI workflows, and data acquisition & event-driven triggering of workflows, which also supports data fusion capabilities.

The general architecture to support AI-based workflows in the DTE engine differentiates between two roles, the DTE developer, focusing on training and validation of the models, and the end-user who can exploit those pre-trained models, for example to re-training them for specific modelling. As for the event-driven architecture, it leverages Apache Kafka as event-ingestion system, and OSCAR as serverless event-processing system, to build a comprehensive solution able to use common file and object storage systems and the PaaS orchestrator as a back-end to the infrastructure (Cloud) required to actually perform data processing.

The DTE features a specific module as well for quality assurance aiming to tackle the early validation of the DTs, before being deployed as a "living DT". The main component is the Software Quality Assurance as a Service (SQAaaS), which provides graphical and programmatic interfaces to compose CI/CD pipelines. The DTE developer will be able to choose among a set of special-purpose libraries to assess QA criteria relative to models and to data. The list of criteria will act as quality gates during the validation of a given DT workflow.

The components to support Big Data analytics aim to provide support for the deployment of data analytics environments, on top of cloud resources. The deployment layer requires a set of topology templates and recipes (a TOSCA templates repository) that will contain the definition of the data analytics software components and the underlying infrastructure required to execute them. The main modules include the possibility to deploy elastic Kubernetes clusters on demand, support to Daskhub environments to execute JupyterHub data analytics, and Volcano to support the deployment of batch systems on top of HPC resources. Additionally, Horovod will be available to support distributed learning frameworks based on common tools such as, among others, Tensorflow, Keras, and PyTorch.

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.

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".

1.2 Document Structure

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 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 data analytics. Finally, we conclude with the description of the next steps towards the first release of the DTE core modules, to take place by fall of 2023.

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.

- **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.
  - 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, leading to at least 8 GPUs necessary. Computational costs can be high; therefore attention needs to be paid to sufficient per-GPU performance next to the number of total GPUs available.

- **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, Graph Neural Networks and Generative Adversarial Networks) will be adopted as modelling tools for learning the underlying mapping between drivers and

outcomes in the past and generalising it to future projection data, where a strong climate change signal is emerging.

As the use case is quite data-driven, besides the actual ML models, 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**: Generative Adversarial Networks, Convolutional Neural Networks, Graph Convolutional Networks

- **ML dataset**: 4D climate time series. Size is in the order of 2TB.

- **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 on HPC resources a distributed Training Framework (e.g., Horovod, PyTorch DDP) 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, Delft-FEWS, 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
- 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, Delft-FEWS, 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

D6.1 Report on requirements and core modules definition

- **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).

- **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

The most important capability 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 processes to the next one. Workflows may have steps that run in parallel.

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

The DTE to be built as part of the interTwin project, and which can be used both by expert users (DTE developers) and regular users (Scientists), will use 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 user (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 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 solution adopted for DTE workflow composition is 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

The goal of this component is to implement 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 has to satisfy the following requirements: i) detect when new data that requires processing is made available; ii) perform data stage and pre-processing (e.g. to perform data cleansing or data quality assessment) and iii) delegating the complex data processing into external workflow management systems in charge of enacting the execution on resources that can be dynamically provisioned from a Cloud-based infrastructure.
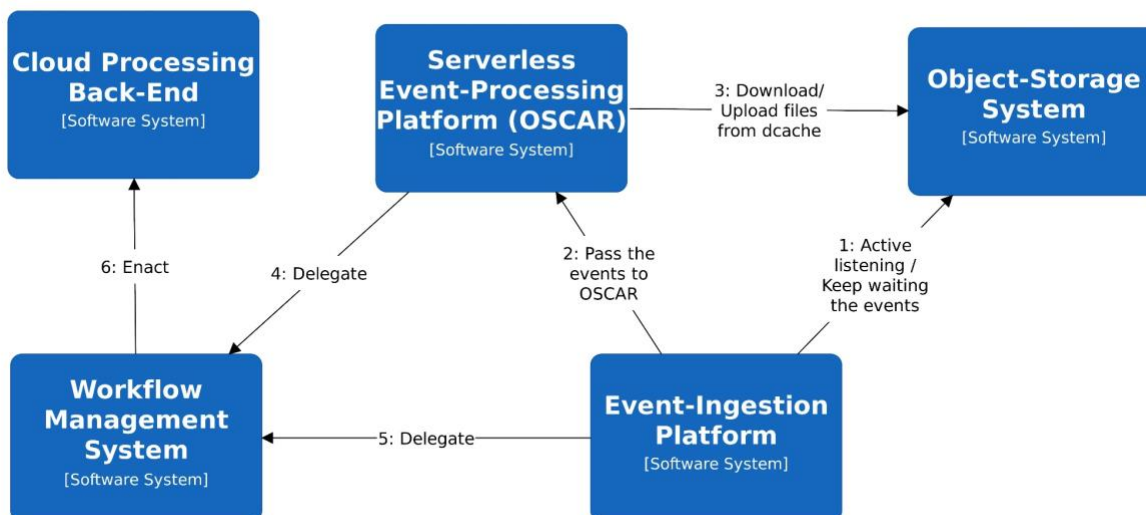
Figure 1 - General architecture for data ingestion and event-driven triggering of workflows

To this aim, **Figure 1** shows the overall architecture for data acquisition and event-driven triggering of workflows, including the high-level components described below.

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.

There are many file-storage systems such as dCache or Onedata. There are also a wide variety of object-storage systems such as MinIO, OpenStack's Swift or NextCloud. We plan to use dCache as an exemplary file storage system due to its support for the Server-Sent Events (SSE) specification, a server push technology that allows a client to receive updates from a server through an HTTP-based connection. It supports several transfer protocols such as WebDav and authentication mechanisms such as OIDC, which facilitates the integration with use cases' technologies.

Event-Ingestion System

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

There are several event-ingestion systems such as those in the category of Publication/Subscribe (i.e., Pub/Sub) systems, as is the case of Apache Kafka, which allows developers to connect programs to each other in different languages and across a large number of nodes. We plan to use Apache Nifi as an exemplary event-ingestion system, 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. In particular, dCache can be used as a source for events via a client for the SSE support. The benefits of an 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.

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 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.

There are different serverless event-processing systems such as OpenFaaS, Knative, or Nuclio. We plan to use OSCAR as an exemplary 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 a two-level elasticity (in terms of the number of pods and the number of nodes) and provide efficient execution of asynchronous 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.

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 to support 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-base 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.

- 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. 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 user interface, as well as an API to trigger the workflow.

When choosing the technology to be adopted for the top-level workflow of the DTE, openEO1 process graphs and Common Workflow Language (CWL)2 were considered as possible options. It was debated if multiple choices should be offered, and the consensus was that that 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 disparate technological stacks. In the end, CWL was selected as the language in which 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 cwltool[3]) which can be used to describe portable reusable workflows.

- CWL has gained much traction and is currently widely supported in practice by popular workflow management systems and engines such as Streamflow or Apache Airflow (CWL-Airflow[4] in particular).

3.2.1 General description and functionalities

For the purpose of defining the top-level Digital Twin Engine (DTE) workflow the usage of Common Workflow Language (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 Snakemake[5,] Streamflow6, Apache Airflow7, and ecFlow[8] . Apache Airflow was selected as the tool to orchestrate and execute the top-level workflow of the interTwin DTE, based on these arguments:

---

[1] https://openeo.org

[2] https://www.commonwl.org

[3] https://github.com/common-workflow-language/cwltool

[4] https://cwl-airflow.readthedocs.io/en/latest/

[5] https://snakemake.readthedocs.io/en/stable/

[6] https://streamflow.di.unito.it/

[7] https://airflow.apache.org

[8] https://www.ecmwf.int/en/newsletter/166/computing/ecflow-5-brings-benefits-member-states

- **User-friendly web interface**: 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 composing workflows with different orchestration solutions.

- **Extensibility**: 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 you to develop custom wrappers for other workflow orchestration solutions and incorporate them seamlessly into Airflow.

- **Strong community and support**: Airflow has a large and active community of users and developers, making it easier to find help, examples, and resources for integrating it with other workflow orchestration solutions. Additionally, the project is backed by the Apache Software Foundation, which provides long-term stability and support.

- **Wide range of built-in operators**: Airflow includes a wide range of built-in operators for various tasks and integrations, making it easier for users to create complex workflows without needing to develop custom code.

- **Scalability and flexibility**: Airflow is designed to be scalable and can handle large-scale, distributed workflows. It supports parallel task execution, dynamic task creation, and execution on various distributed computing environments, including Kubernetes and Apache Mesos.

- **Language and platform support**: 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.

As an example, the case of HEP Machine Learning workflows, whose requirements have been captured in **Section 2.3,** can be supported by Apache-Airflow.
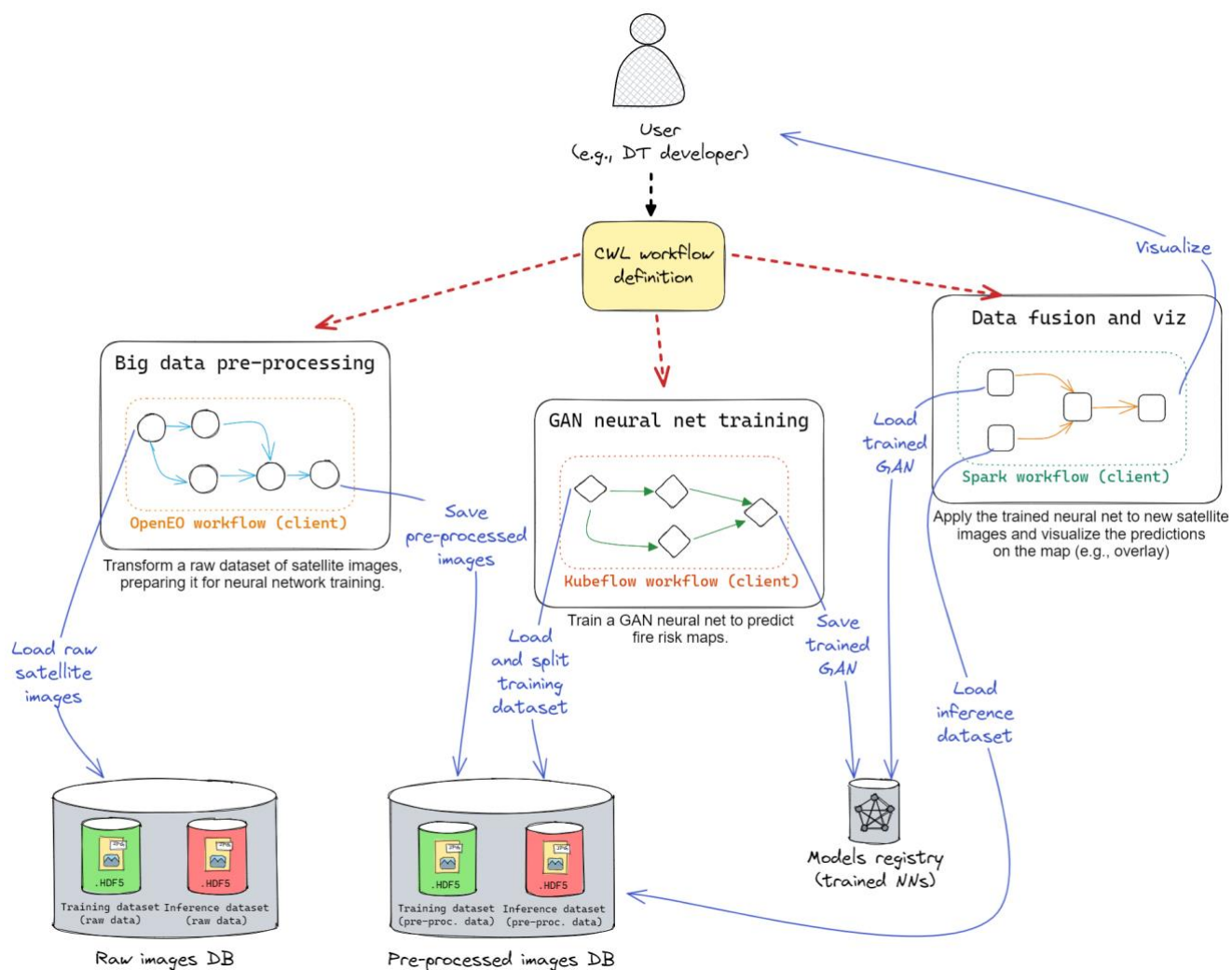
Figure 2 - Example of workflow composition for EO applications.

Figure 2 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 will have to provide 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 will need to deliver interfaces for DT Developers and eventually DT User willing to execute DT Application 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.

Apache Airflow

One of the most promising solutions for the definition and execution of the CWL workflow is Apache 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.

Apache 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 you 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 for users to create complex workflows without needing to develop custom code. Airflow is designed to be scalable and can handle large-scale, distributed workflows. It supports parallel task execution, dynamic task creation, and execution on various distributed computing environments, including Kubernetes and Apache Mesos. 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

ecFlow[9] 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/

- **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

openEO[10] is an API specification[11] to discover, access and process earth observation data and define abstract processing workflows. It has grown into an independent open source community[12] standard in the earth observation community and is steered through the openEO project steering committee[13].

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 virtual data cubes;

- 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);

- Processes make the nodes in this graph and are pre-defined[14]. 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;

- Asynchronous processing of graphs through batch jobs and management of jobs;

- Authentication and authorization of any exposed micro service;

---

[10] https://openeo.org/

[11] https://api.openeo.org/

[12] https://github.com/open-eo

[13] https://openeo.org/psc.html#members

[14] https://processes.openeo.org/

- 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)S, XYZ, WCS or OGC API tiles.

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 R[15] and Python[16,] as well as the JavaScript based web-editor.

Streamflow

StreamFlow[17] 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.

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

---

[15] https://openeo.org/documentation/1.0/r/

[16] https://open-eo.github.io/openeo-python-client/

[17] https://streamflow.di.unito.it/

D6.1 Report on requirements and core modules definition

Delft-FEWS

Delft-FEWS[18] (Flood Early Warning System) is an advanced software system designed to provide early warning and forecast capabilities for hydrologic hazards such as floods, droughts, and other water-related events. It is developed and maintained by Deltares and has been used globally in numerous countries.

Delft-FEWS is highly flexible and can be tailored to specific needs of a local, regional, or national water system. It integrates a wide range of water level, flow, and rainfall data (from both real-time monitoring and forecast models) into a single, cohesive platform. This data can be used to run simulations and generate forecasts, helping decision-makers take timely action in response to developing hydrologic events.

The system includes a sophisticated suite of data handling and visualisation tools, enabling users to interactively explore and analyse their data. Advanced warning and alerting features can automatically notify users when certain hydrologic conditions or thresholds are exceeded.

Delft-FEWS is an open data handling platform, meaning it is not tied to any specific models. Instead, it can interface with a wide range of third-party hydrological and hydraulic models, allowing for considerable flexibility in the way it is used.

Delft-FEWS offers several workflow management features, designed to support its users in managing, executing, and tracking the complex tasks associated with forecasting and early warning processes, including:

- **Automation of tasks**: Delft-FEWS can be set up to automatically run data imports, model simulations, and data exports at specified intervals. This is important in operational settings where timely, regular forecasts are needed.

- **Workflows**: Users can define a series of steps, known as a workflow, that should be executed in a specific order. These workflows may include data retrieval, model execution, post-processing, forecast visualisation, etc. Once set up, workflows can be run manually or automatically or can be triggered when certain messages are logged, or thresholds are crossed.

- **Tracking and auditing**: Delft-FEWS records the execution of all tasks, allowing users to track when certain tasks were completed, who completed them (in case of manual intervention), and whether there were any errors or issues.

- **Manual intervention**: While Delft-FEWS can automate a lot of tasks, it also provides capabilities for users to manually intervene in workflows. For example, users might want to manually adjust forecast parameters or inspect and modify model inputs/outputs before they are used in further steps of the workflow.

- **Alerts and notifications**: Delft-FEWS can be set up to notify users when certain conditions are met, such as a forecasted water level exceeding a certain threshold. This is an integral part of the early warning functionality of the system.

---

[18] https://www.deltares.nl/en/software-and-data/products/delft-fews-platform

D6.1 Report on requirements and core modules definition

- **Distributed forecasting**: Delft-FEWS supports distributed forecasting, where different parts of a workflow are run on different machines. This can be used to distribute computational load or to ensure that certain tasks are executed on machines that have the necessary software or hardware requirements.

The capabilities of Delft-FEWS can be extended and customised to meet the specific needs of individual users or organisations, allowing for a wide range of workflow management applications within the context of hydrologic forecasting and early warning.

Ophidia

Ophidia[19] 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 biodiversity, solid Earth, and environmental data.

The framework exploits an array-based storage model, leveraging the datacube abstraction from OLAP systems, 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 (more than 50 currently available). Among the others it supports statistical analysis, time series processing, data intercomparison, subsetting, multi-model analysis, etc.

The framework was enhanced a few years ago to target very large-scale applications through a new runtime environment based on the MPI+X paradigm and tighter integration with HPC systems. 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, such as 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 definition20. The workflow engine is able to handle complex workflow in the form of Directed Acyclic Graphs 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 bindings) can be used to interact with the engine to send and execute workflow documents with the defined JSON format. In addition, a Python module recently developed, named esdm-pav-client, can also be used to create the workflow description in programmatic way, run it on the server and monitor its execution via a Python API.

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

---

[19] https://ophidia.cmcc.it/

[20] https://ophidia.cmcc.it/documentation/users/workflow/workflow_basic.html

- 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;

- 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 execution:* one between to sub-workflows is executed according to a condition to be checked at run-time, similarly to an if-else statement;

- *iterative execution*: a sub-workflow can be executed multiple times, similarly to a loop statement;

- *parallel execution*: of sub-workflows: a sub-workflow can be executed in parallel multiple times on different input data or with different operations on the same dataset;

- *interactive execution:* the execution of a task is triggered by an event (e.g., the availability of a file/data);

- *interleaving execution*: the execution of a task is triggered by another workflow.

These constructs can also be nested to support execution of very complex and dynamic workflows.

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.

3.3 Provenance in Workflows

Workflow and provenance are two faces of the same medal. While the former addresses the execution of multiple computational and data tasks over a set of machines, the latter relates to the tracking and management of the lineage information.

According to the National Institute of Standards and Technology (NIST)[21] data provenance is defined as follows: *"It is an equivalent term to chain of custody. It involves the*

---

method of generation, transmission and storage of information that may be used to trace the origin of a piece of information processed by community resources."

Storing information about the origin and history of a data fosters trust and re-usability. It represents a best practice too, as metadata in its essence contributes to data documentation.

Making provenance FAIR means that such metadata information must be findable, accessible, interoperable, and re-usable. Interoperability is key, both in terms of service interfaces and data model. To this end a provenance data model (PROV-DM) has been provided by the W3C consortium in 2013 to represent provenance information in a standardised way.

Such a formal data model describes any physical, conceptual, and virtual object. Specifically, at its core the PROV data model [**R1**] defines the use and production of entities by activities, which may be influenced in various ways by agents. **Figure 3** shows the relationship between these three elements.
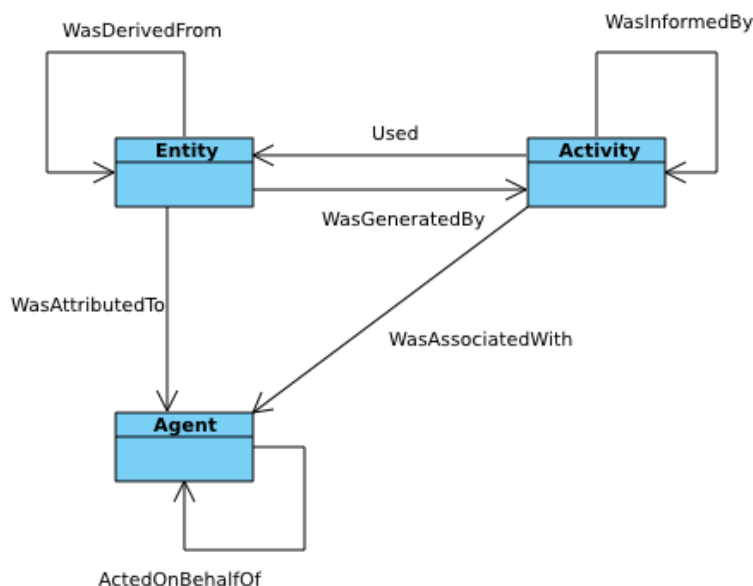
Figure 3 - PROV Data Model

Such a data model is adopted by the yProv service, which represents a cross-domain service able to track, manage, query, and mine provenance information.

Such service consists of a RESTful interface with a persistent database in its back-end.

The RESTful API allows the implementation of two different scenarios:

- *batch*: through a document-based API to manage provenance offline, after the end of the workflow;
- *online*: through a set of methods to manage provenance tracking at runtime.

In the interTwin project, such a service will help tracking provenance information associated to climate analytics workflows both in batch and real-time workflows. Though its primary application domain is climate, its cross-domain and standard API definition enables its re-use over multiple different domains. The service is also intended to

manage scenarios with multi-level provenance information. In this respect a relevant use case in interTwin is the climate analytics experiments based on the Ophidia, which represents an example of a second-level provenance workflow engine.

3.4 Data Fusion in Workflows

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.

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 four 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.
- Implementation of processes preparing results for visualisation in collaboration with specific thematic modules for visualisation.

The data fusion components are strongly dependent on the exact data sources used and will have a more detailed description in the second release of this document (Deliverable 6.3).

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 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), but the models could also be re-trained on new data, if desired. 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 hyper-parameter optimization. The component for AI workflows in digital twins does not exist at the moment, and it is part of the set of the innovations introduced by interTwin. Therefore, the proposed architecture is likely to change throughout the lifespan of the project, to accommodate use cases' needs.

Training an ML model involves loading some (pre-processed) dataset from the storage and splitting it into training and validation. In the case of online learning, the training dataset is a stream. 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., Horovod22, RaySGD23), and hyper parameter optimization (HPO). Once a model is trained, its performances are 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".

Once an ML model passes domain-specific validation, performed by the "Quality and uncertainty tracing" DTE's module, it can be deployed. 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 computing 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 easy to understand, whereas experienced users would like to have finer control of the underlying

---

[22] https://github.com/horovod/horovod

[23] https://docs.ray.io/en/releases-1.11.0/raysgd/raysgd.html

D6.1 Report on requirements and core modules definition

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. 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 of the scientist. If the scientist has some special needs, they can outsource changes to a DT developer.
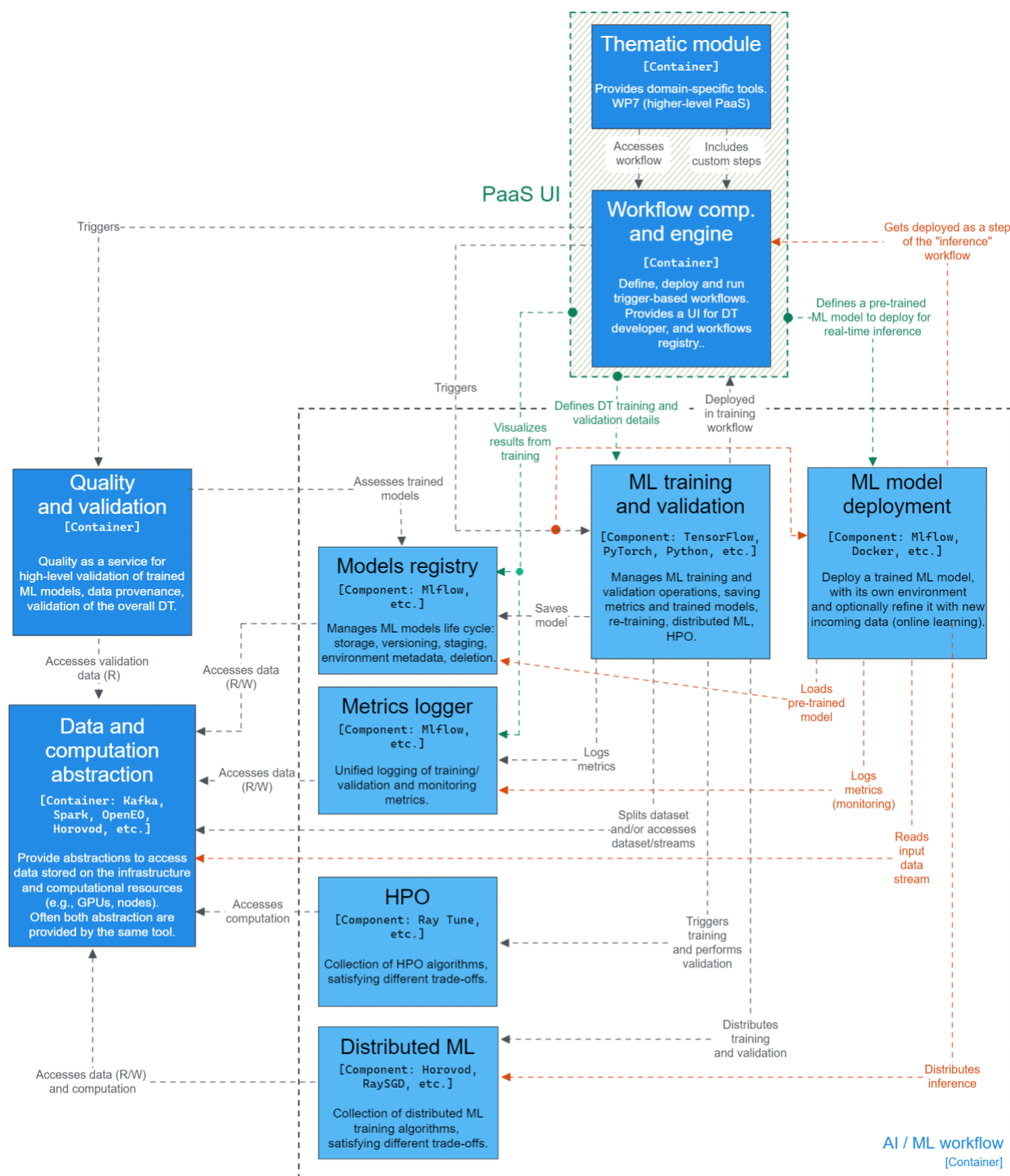
Figure 4 - Detailed view on the architecture of the ML component

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, interactions with other AI/ML components, the required technology stack, and even the interactions with other tasks will be 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 **Figure 5.**

4.1.1 General Description and functionalities

This module provides various functionalities to the DTE for training their AI models. The module will allow 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 will be required. In collaboration with WP5, support will be provided for job scheduling, for example with Slurm, Cron, etc. JupyterLab24 is currently being investigated as a programming interface for this module. Furthermore, some use-cases could potentially require online training functionality, which means training the AI models on-the-fly with new data, perhaps from real time sensor outputs. In this regard, besides normally used offline training, the module will also permit online training. The primary components of this module are summarised here.

Custom interTwin environment

Requirements have been gathered in the initial phase of the project. This will be continued through consultation with individual use-cases from WP4 and WP7 to gather further elaborate requirements in terms of modules necessary to train the AI models. Based on the survey, a custom interTwin Python-based environment is currently being developed for the use-cases to allow training on HPC resources. For instance, at the moment, it is observed that the open-source Python packages, PyTorch[25] and TensorFlow[26] are widely used across multiple use-cases. Other packages will continuously be updated in the environment over the course of the project. Furthermore, if required by individual use-cases, it will also be possible to load their own custom environment and requisite support will be provided in this regard.

---

[24] https://jupyter.org/

[25] https://pytorch.org/

[26] https://www.tensorflow.org/
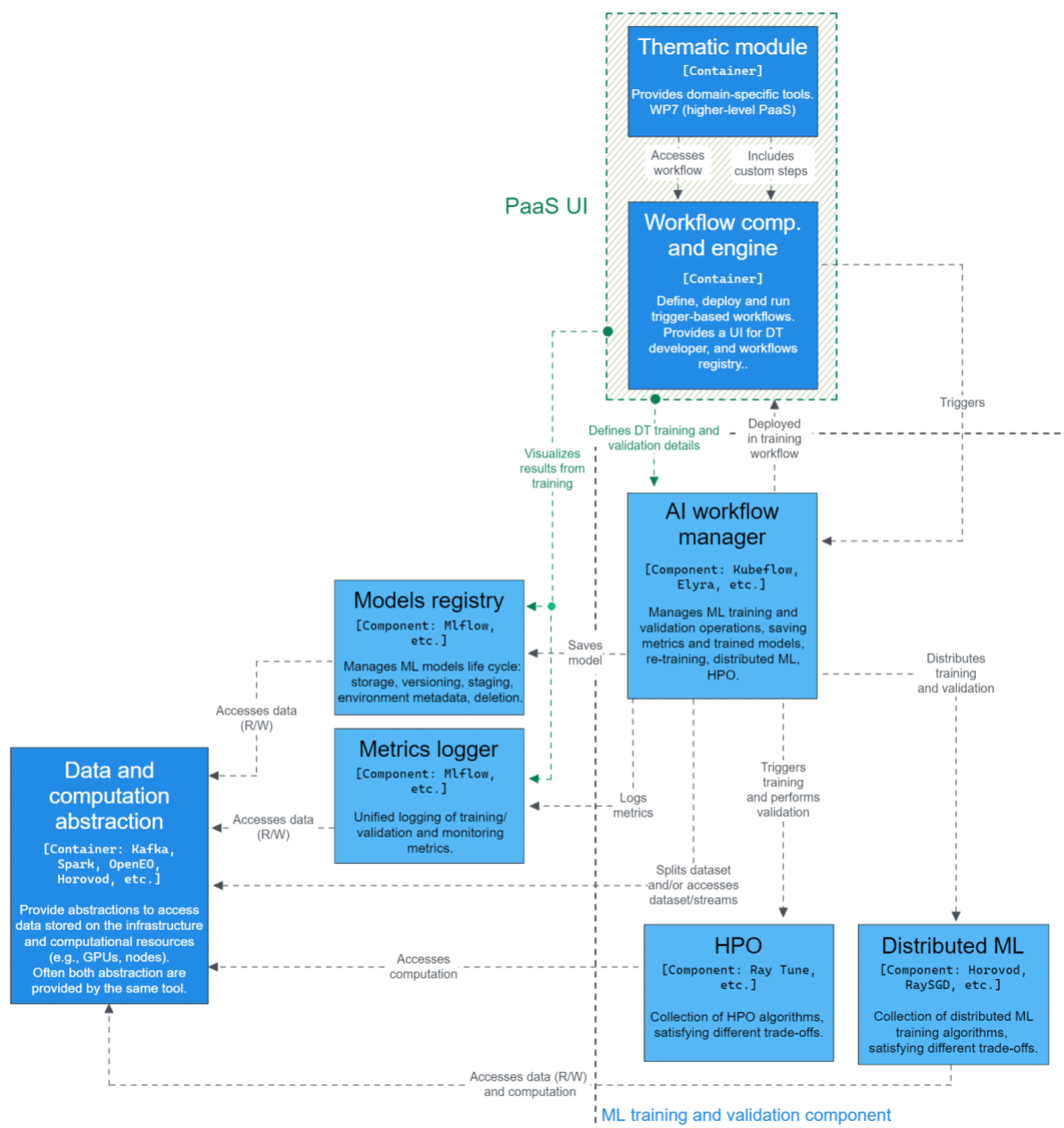
Figure 5 - Detailed view on the architecture of ML training module

Distributed training

The training module will provide 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 will be provided, which distributes a single model to several workers, such that each worker only executes a fraction of the full model.

D6.1 Report on requirements and core modules definition

There are multiple open-source frameworks that provide distributed training functionality. PyTorch Distributed Data Parallel (DDP) module[27], Horovod developed by Uber, Helmholtz Analytics Toolkit (HeAT)[28] from the Helmholtz Association and DeepSpeed[29] from Microsoft are a few of the available frameworks. In the course of the project and depending on the requirements, these and other possible distributed training frameworks will be continuously added and updated to the training module.

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 HyperBand[30] and BOHB[31] provide solutions for HPO. These will be implemented through open-source HPO frameworks such as RayTune[32] and DeepHyper[33] to provide the functionality.

Workflow Pipeline Editor

The AI workflow consists of multiple components which can broadly be categorised into three parts, namely preprocessing, training, and data processing. An AI pipeline example with the basic components is shown in **Figure 6**, which is based on the Elyra pipeline editor[34]. In the initial phase of the project, Elyra has been identified as one of the solutions for constructing the workflow pipeline. Elyra is available as an extension to the JupyterLab environment. In the requirements gathering phase, it is seen that the JupyterLab or notebook platform is used across multiple use-cases.

---

[27] https://pytorch.org/docs/stable/distributed.html

[28] https://github.com/helmholtz-analytics/heat

[29] https://github.com/microsoft/DeepSpeed

[30] https://arxiv.org/abs/1603.06560

[31] https://arxiv.org/abs/1807.01774

[32] https://docs.ray.io/en/latest/tune/index.html

[33] https://deephyper.readthedocs.io/en/latest/

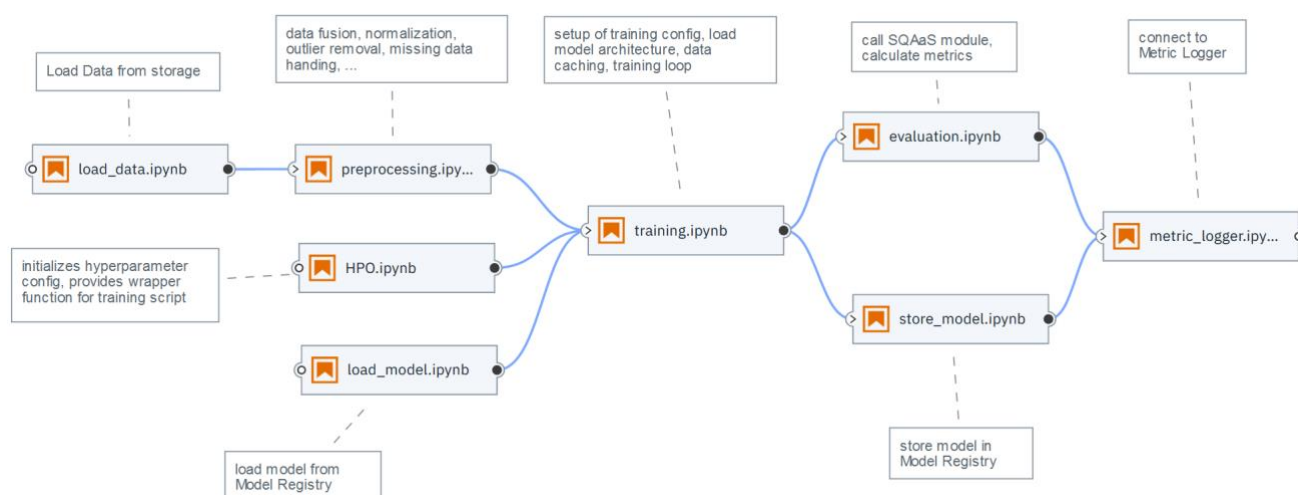[34] https://elyra.readthedocs.io/en/latest/

Figure 6 - Elyra-based example of an illustrative AI pipeline

The preprocessing step involves operations such as data cleaning, data transformation, data fusion, etc. Many of these operations will be part of Data Fusion, described in **Section 3.4** and the details on the functionalities of the preprocessor in the AI module will be decided over the course of the project. The training component consists of various modules defined concretely in **Section 4.1**. Finally, the post-processing component involves performing elementary tests on the performance of the models and updating the model registry. This module requires close interaction with the Quality Assurance components dedicated to validation and verification of the trained models, described in **Section 5**.

Workflow Management Tool

The solutions to launch the AI pipeline in cloud/HPC infrastructure are currently being investigated and in the initial phase, multiple frameworks have been identified. Apache Airflow[35], an open-source Python-based platform, is one of the workflow management tools that is being considered. Airflow offers many functionalities to define and schedule workflows and then monitor these with a built-in User Interface (UI) platform. Airflow uses Directed Acyclic Graphs (DAGs) to orchestrate the workflows and executes them according to the specified dependencies in the pipeline. It manages the scheduling and execution of the workflows, which could be scheduled or triggered by external events[36]. Airflow provides a high-level interface to external platforms with Hooks[37], which for instance will facilitate communication with external databases. Furthermore, due to the modular setup, Airflow should have good scalability. The Airflow deployment across the HPC infrastructure is also under active development at Jülich Supercomputing Centre (JSC), one of the partner HPC centres, which will further benefit the project in the long term.

---

[35] https://airflow.apache.org/

[36] https://airflow.apache.org/docs/apache-airflow/1.10.2/scheduler.html

[37] https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/connections.html

Another workflow management tool that is being investigated is Kubeflow[38], which is also an open-source platform. 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 (similar to Elyra), 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.

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 or Apache Airflow-like 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)
- Communicate with external environment (e.g., CLI, REST APIs)
- Distributed AI framework (e.g., DeepSpeed, Horovod, PyTorch DDP, etc.)
- Drivers to access (parallel) computing infrastructure (e.g., CUDA drivers)
- Job scheduling support (e.g., SLURM)
- Container (e.g., Docker) engine support to deploy models in infrastructure
- Workflow orchestrator to launch training of AI models in infrastructure (e.g., Airflow, 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 to a certain checkpoint to resume with new configurations (e.g., learning rate) without the need to completely discard the

---

[38] https://www.kubeflow.org/

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.

Data fusion will provide a (preprocessed) dataset URI on disk or as object storage. The DT developer needs to implement the loading of the preprocessed data into `_getitem_()` for `torch.utils.data.DataLoader` or `map()` for `tensorflow.data.dataset.`

Quality and uncertainty tracing: During the training procedure, the model will be validated with a training and a validation loss. If the loss envelope is considered bad according to predefined metrics (threshold, sliding window average of loss, etc.) the training will be stopped, and a warning message will be raised. The training module will monitor and visualise the training and additionally propagate this information to the Quality Assurance component to complement the quality assessment for a certain model. If a custom loss is needed, the DT developer has to define the desired loss function using Pytorch/Tensorflow logic exclusively in order to preserve differentiability. We envision a sample loss function that the DT developer can use as a guideline.

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 review workflows. 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 and Azure, or custom deployment scripts).

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 tools that is investigated for the Model Registry is MLFlow. 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.

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:

- 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 than not all the training is lost and allows to fall back to models 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 post-processed 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. Its Software Quality Assurance as a Service (SQAaaS) module will be called on models in the Model Registry and conduct unit tests with given corner cases provided by the use cases. It will certify the quality of each model in the registry according to the FAIR (Findable, Accessible, Interoperable and Reusable) quality assessment. The quality certification will be stored alongside the model itself and made visible to the DT user to enable it to easily choose among (pre)trained models.

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.

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

Under the MLFlow framework mentioned above there is the feature MLFlow Metric Logger. It 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 used with programming languages like Python, Java, R or 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.

ML model deployment: During deployment the pre- or user defined metrics will be visualised in order to monitor if the deployed model is working correctly. 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 7.**

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.
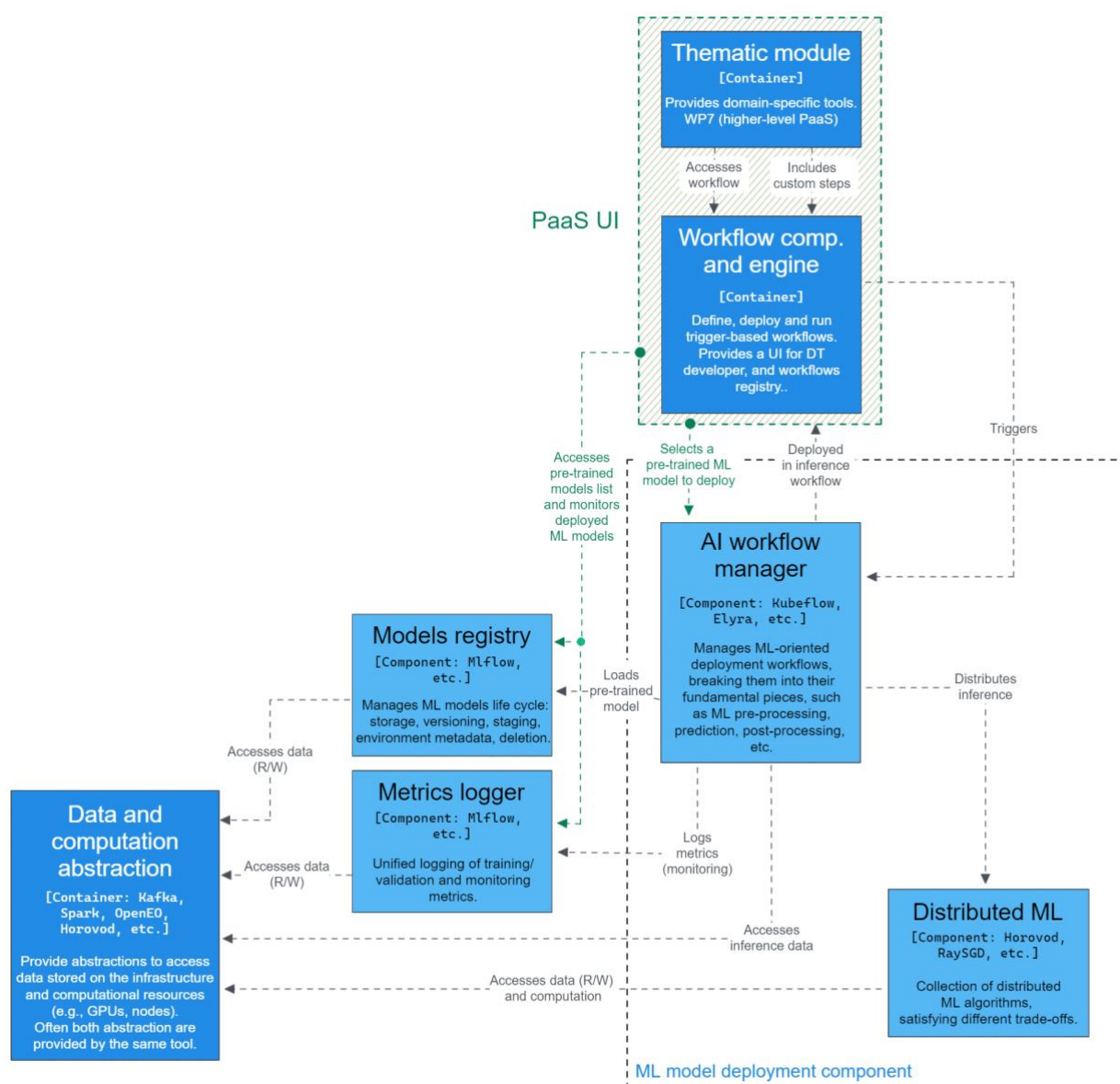

Figure 7 - - Detailed view on the architecture of the ML deployment module

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 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, OpenStack APIs, or Apache Airflow-like APIs.

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 orchestrator (such as **Argo workflows** or **Apache Airflow**).

- 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

- 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 (WP5): 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 Kubernetes-like APIs. Furthermore, the computing infrastructure will provide computing resources such as GPUs (preferably) or CPUs, to run the deployed ML model.

Data infrastructure (WP5): 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.

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 some UI provided by the DTE PaaS. 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 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**, in addition to the compliance with FAIR principles, foreseen QA work shall also include criteria for data quality, e.g., check data and metadata integrity after data ingestion.

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

Software Quality Assurance as a Service (SQAaaS) provides graphical and programmatic interfaces to compose continuous integration and delivery (CI/CD) pipelines. These pipelines are then used as fail-fast systems so that any indication of a failure is reported promptly during the development life cycle.

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.

The architecture of the SQAaaS module is depicted in **Figure 8**, and its primary components are summarised below.

SQAaaS API server

This is the key component of the SQAaaS platform. It provides two main features: 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 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.

D6.1 Report on requirements and core modules definition

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 a 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,[39] DeepChecks,[40] and Great Expectations.[41]

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.

---

[39] https://github.com/EOSC-synergy/FAIR_eva

[40] https://github.com/deepchecks/deepchecks

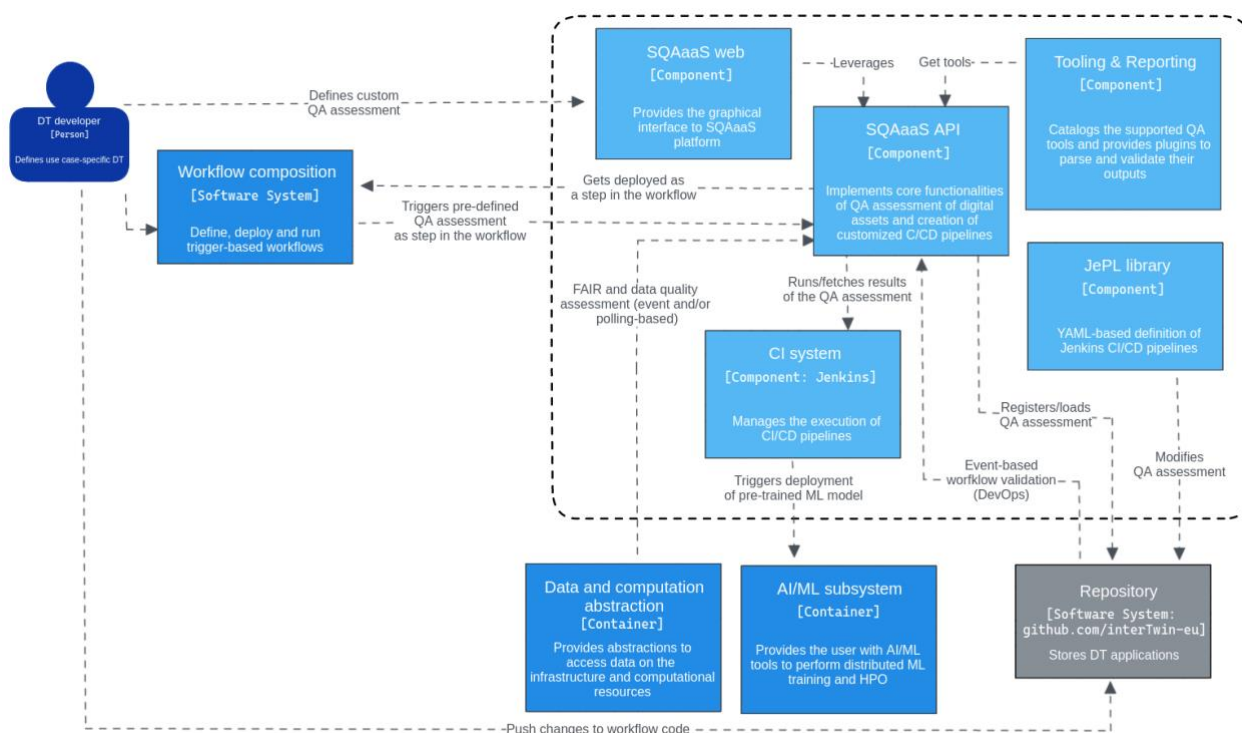[41] https://github.com/great-expectations/great_expectations

Figure 8 - 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 though 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 integrated with the GitHub platform, where the InterTwin project has already an

organisation available[42]. 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 additional 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.

---

[42] 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 Daskhub environments, **Section 6.4** covers Volcano and Horovod environments, **Section 6.5** covers Hadoop and Spark, while Section 6.6 deals with Kubeflow clusters. Finally, **Section 6.7** covers Ophidia.

6.1 Deployment of Big Data Analytics tools

This section describes the deployment module of the Big Data Analytics subsystem. **Figure 9** 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.
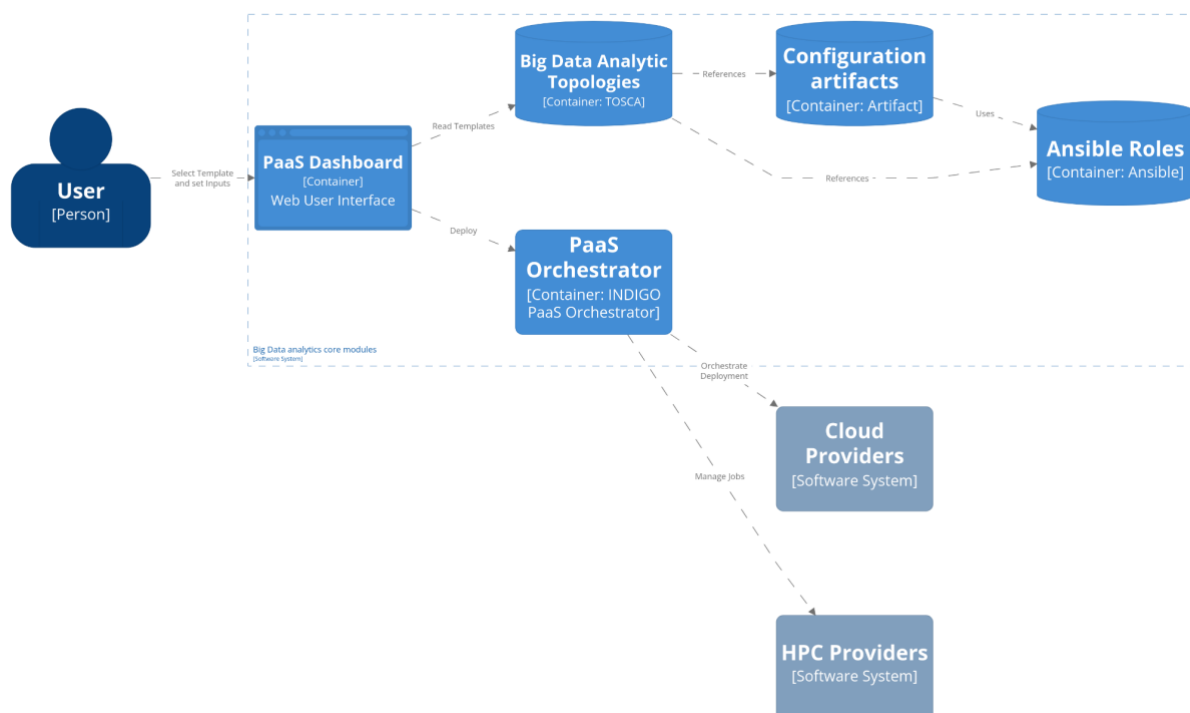
Figure 9 - General architecture of the data analytics tools

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

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 Manager[43] (IM) is the component that actually deploys and configures the virtual infrastructure once the site has been selected.

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. **Figure 10** shows the architecture for deployment of Big Data Analytics tools using on-demand Kubernetes clusters.

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.

---

[43] https://www.grycap.upv.es/im/index.php

6.2.2 Interfaces

Kubernetes objects are managed through the Kube apiserver using the Kubernetes API[44]. 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 Dashboard[45] is a Graphical User Interface (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 charts[46] 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 GitHub[47]. This recipe comprises the following components:

Kubernetes

Kubernetes (K8s) is an open source platform for automating the deployment, scaling, and management of containerized applications.[48]

Elastic Cloud Computing Cluster

The Elastic Cloud Computing Cluster (EC3)[49] 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.

---

[44] https://kubernetes.io/docs/concepts/overview/kubernetes-api/

[45] https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

[46] https://helm.sh/es/docs/topics/charts/

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

[48] https://kubernetes.io/es/

[49] https://eosc-portal.eu/news-and-events/news/elastic-cloud-compute-cluster-ec3
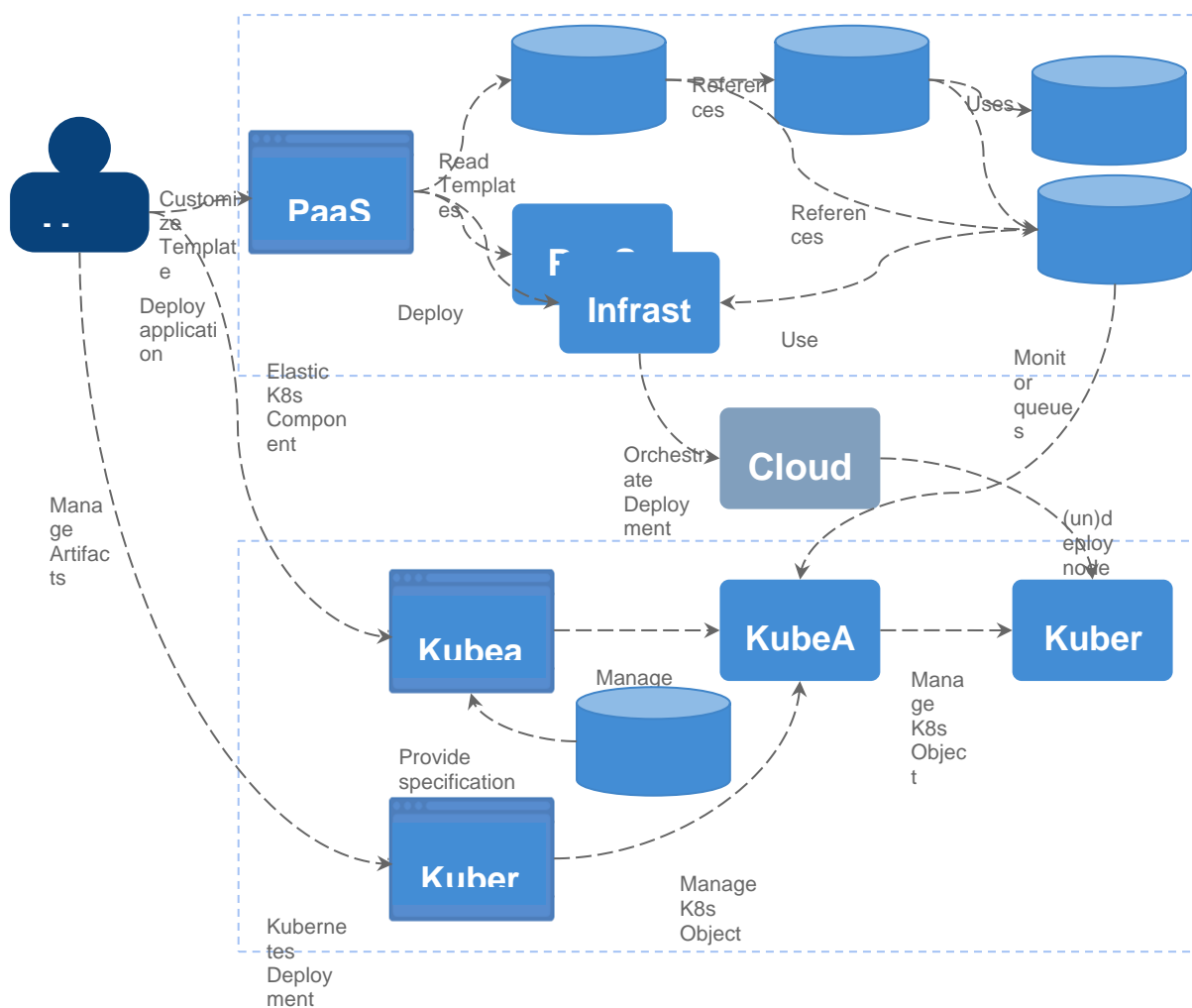
Figure 10 - General architecture for deploying data analytics tools on Kubernetes clusters

6.2.4 Interaction with other components

The on-demand elastic Kubernetes cluster module acts as the framework using which Daskhub environments (see **Section 6.3**), Volcano and Horovod environments (see **Section 6.4**), Kubeflow clusters (see **Section 6.6**), and Ophidia clusters (see **Section 6.7**) will be deployed.

6.3 Daskhub environment

This section describes how Daskhub environments can be as part of the Big Data Analytics subsystem, and **Figure 11** illustrates the involved architecture.

6.3.1 General description and functionalities

DaskHub is a Helm chart that deploys both Dask Gateway and JupyterHub on a Kubernetes cluster[50]. DaskHub is an evolution of the Pangeo Helm chart, which came out

---

[50] https://artifacthub.io/packages/helm/dask/daskhub

of that community's attempts to do big data geoscience on the cloud. DaskHub provides a scalable and flexible environment for data analysis and processing. With DaskHub, users can easily launch and scale Dask clusters within a JupyterHub environment, allowing for seamless integration of interactive computing and parallel processing.

Dask Gateway is an option for deploying Dask clusters[51]. A Dask cluster is a set of Dask workers that are managed by a Dask scheduler. The workers are responsible for executing tasks and the scheduler is responsible for coordinating the execution of those tasks across the workers. It is centrally managed, meaning that administrators do the heavy lifting of configuring the Gateway, while users simply connect to the Gateway to get a new cluster. Dask workers can execute any Python function. As such, Dask, an open-source library for parallel computing in Python[52], can be utilised to parallelize and distribute the execution of any Python code. However, it should be noted that using Dask's high-level APIs can facilitate the parallelization and distribution of computations.

JupyterHub is a multi-user server that brings the power of notebooks to groups of users[53]. It gives users access to computational environments and resources without burdening them with installation and maintenance tasks. Users, including students, researchers, and data scientists, can get their work done in their own workspaces on shared resources.

6.3.2 Interfaces

Once deployed, users can interact with DaskHub through the JupyterHub interface. JupyterHub provides a web-based interface where users can launch Jupyter notebooks and access computational resources. From within a Jupyter notebook, users can launch and scale Dask clusters to perform parallel computations.

6.3.3 Technology stack

The DaskHub environment can be deployed on a Kubernetes cluster using a Helm chart. To do this, first add the repository by running the command helm repo add dask https://helm.dask.org, then install the chart with the command helm install my-daskhub dask/daskhub --version 2023.1.0.

This chart will deploy a standard Dask Gateway deployment using the Dask Gateway helm chart and a standard JupyterHub deployment using the JupyterHub helm chart. The Dask Gateway is configured to use JupyterHub for authentication and JupyterHub is configured to proxy Dask Gateway requests and set Dask Gateway-related environment variables. Chart configurable parameters can be modified on installation in config.yaml. JupyterHub will be available at the proxy-public external IP.

---

[51] https://gateway.dask.org/

[52] https://www.dask.org/
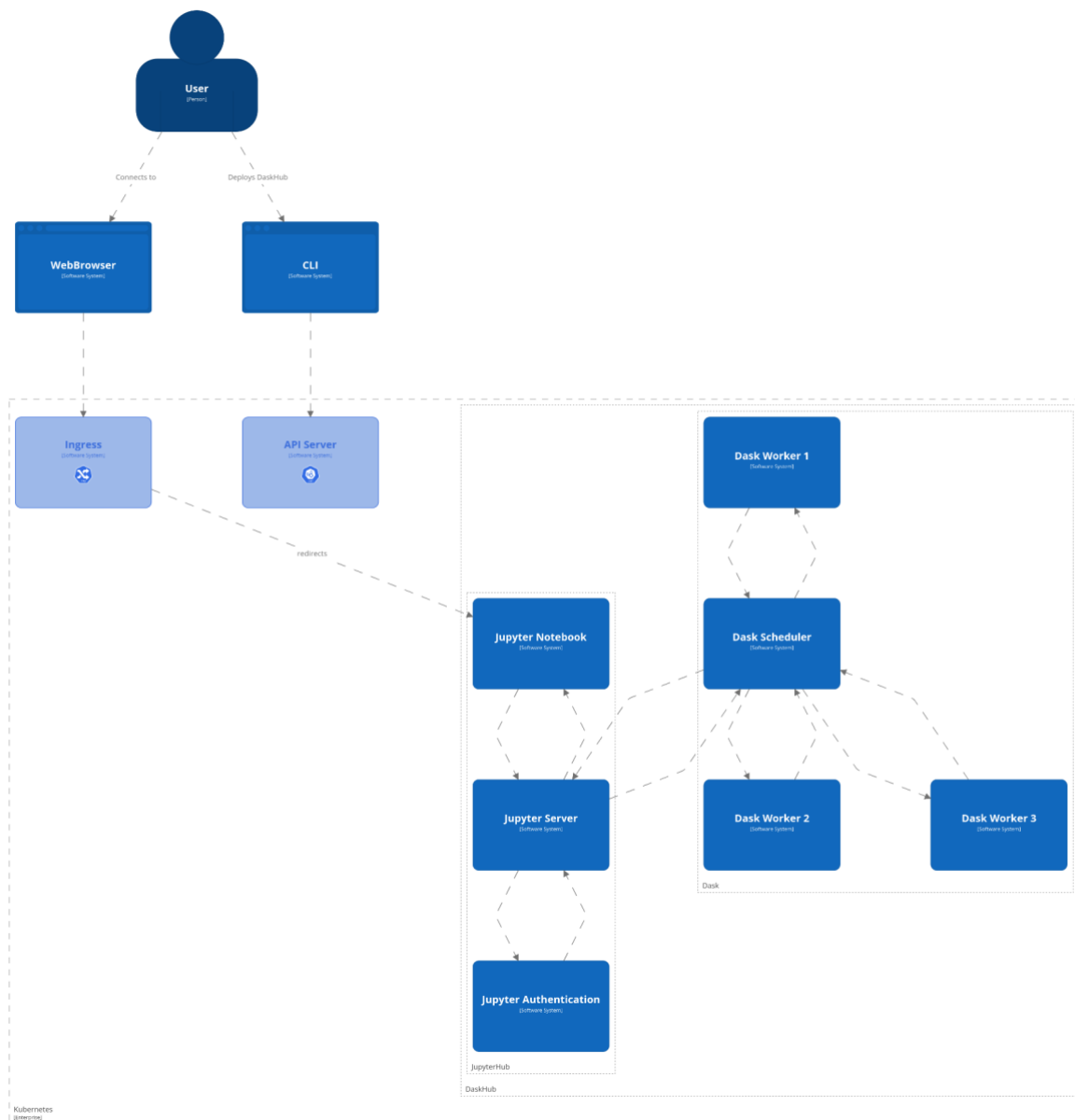
[53] https://jupyter.org/hub

Figure 11 - General architecture for using Daskhub environments for data analytics

6.3.4 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).

6.4 Volcano and Horovod environment

This section details how Volcano and Horovod environments can be as part of the Big Data Analytics subsystem, while **Figure 12** illustrates the involved architecture.

6.4.1 General Description and functionalities

Together, Volcano and Horovod can be used to run distributed deep learning workloads on a Kubernetes cluster. Volcano provides the infrastructure for managing the compute resources while Horovod provides the framework for scaling the deep learning training jobs. This combination allows users to efficiently train large deep learning models on big data sets.

Volcano

Volcano is a batch system built on Kubernetes that provides a platform for running high-performance workloads[54]. It can be used to run big data analytics jobs and other compute-intensive tasks. Volcano provides features such as job scheduling, resource management, and job management to help users efficiently run their workloads on a Kubernetes cluster.

Horovod

Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet[55]. 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.

6.4.2 Interfaces

Once deployed, users can interact with the Volcano and Horovod environment by submitting a script written in Python that includes the Horovod libraries. The Volcano system would then be in charge of running the workload and adjusting the resources accordingly. This allows users to efficiently run distributed deep learning workloads on a Kubernetes cluster.

6.4.3 Technology stack

Volcano can be installed through its Helm chart on top of Kubernetes, provided that the version of Kubernetes is 1.13 or greater and includes support for custom resource definitions[56]. This is supported by the TOSCA recipe of Kubernetes described in section 6.1. This can be performed by selecting the support of Helm charts and indicating as the name of the Helm Chart to install "volcano-sh/volcano" and **https://volcano-sh.github.io/charts** as the URL to the Helm repository.

---

[54] https://volcano.sh/en/

[55] https://horovod.ai/

[56] https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
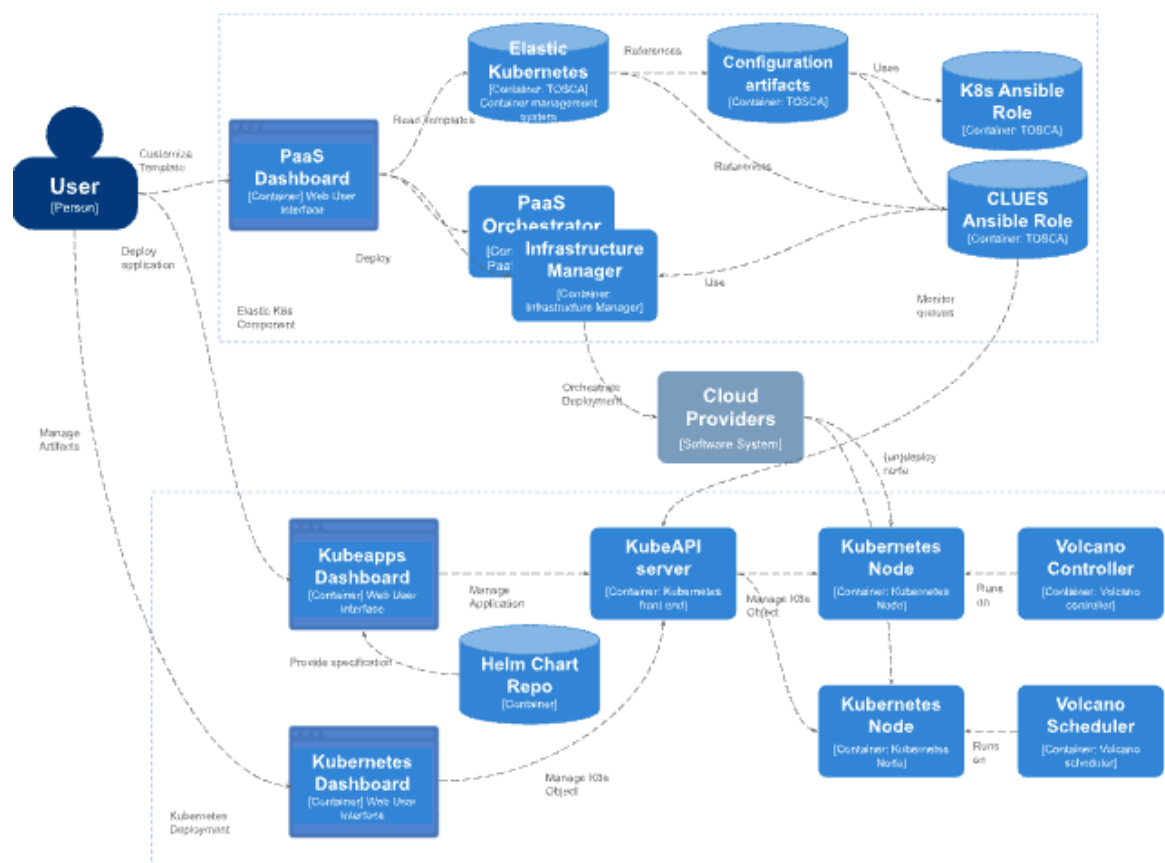
Figure 12 - General architecture for using Volcano and Horovod environments for data analytics

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).

6.5 Apache Hadoop and Apache Spark clusters

This section describes how Apache Hadoop and Apache Spark clusters can be used as a core part of the Big Data Analytics subsystem. **Figure 13** shows the architecture of the Big Data Analytics component with Hadoop and Spark.

6.5.1 General description and functionalities

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using programming models such as Map Reduce. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. It provides a distributed storage capability to split and replicate fractions of files so distributed workloads can efficiently

process concurrently data analytic tasks. Hadoop can be used as a backend for other data analytic tools such as Spark.

Apache Spark[57] is a multi-language engine for executing data engineering, data science, and machine learning on clusters. It provides high-level APIs in Java, Scala, Python, and R, and an optimised engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL[58] for SQL and structured data processing, pandas API on Spark[59] for pandas workloads, MLlib[60] for machine learning, GraphX[61] for graph processing, and Structured Streaming[62] for incremental computation and stream processing.

6.5.2 Interfaces

As Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation, it does not have a single interface. Instead, the interfaces of Hadoop are:

- The Hadoop libraries and utilities contained in Hadoop Common, these are needed by other Hadoop modules;

- The API of the Hadoop Distributed File System (HDFS), which is a distributed file system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;

- The Hadoop YARN API[63] allows for managing computing resources in Hadoop clusters and using them for scheduling applications;

- The MapReduce programming model for large-scale data processing.

Apache Spark, the unified analytics engine for large-scale data processing, provides multiple interfaces for programming clusters with implicit data parallelism and fault tolerance. The most important of these are:

- The Spark Core API is the foundation of Spark, as it provides distributed task dispatching, scheduling, and basic I/O functionalities. Libraries in multiple programming languages, including Java, Python, Scala, and R, are available;

- Spark SQL is a component on top of Spark Core that introduced a data abstraction called DataFrames, which provides support for structured and semi-structured

---

[57] https://spark.apache.org/docs/latest/

[58] https://spark.apache.org/docs/latest/sql-programming-guide.html

[59] https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_ps.html

[60] https://spark.apache.org/docs/latest/ml-guide.html

[61] https://spark.apache.org/docs/latest/graphx-programming-guide.html

[62] https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

[63] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html

data. Spark SQL provides a domain-specific language (DSL) to manipulate DataFrames in Scala, Java, Python, or .NET.

Additional Spark libraries can offer other capabilities and interfaces, notable mentions include the MLib distributed machine-learning framework, and the GrapX distributed graph-processing framework.

6.5.3 Technology stack

The TOSCA template that codes this virtual application is available in GitHub[64]. It comprises the template for a Spark working node and a Spark master, both inheriting from the Hadoop Working Node and Hadoop server specifications of the Hadoop cluster TOSCA template[65]. Subsequently, the Hadoop cluster TOSCA template could be instantiated separately by specifying the Hadoop Master and Hadoop Working nodes. In the case of the Spark cluster, the nodes implement both specifications (Spark and Hadoop) so only the Spark nodes need to be deployed. The number of nodes should be defined at deployment time.

Apache Spark

Apache Spark is installed through an Ansible Playbook stored in Ansible Galaxy[66].

Apache Hadoop

The definition of the Hadoop Master is one of the custom types of the TOSCA recipes[67]. It uses an Ansible Playbook stored in Ansible Galaxy[68].

---

[64] https://github.com/grycap/im-dashboard/blob/master/tosca-templates/spark_hadoop.yaml

[65] https://github.com/grycap/im-dashboard/blob/master/tosca-templates/hadoop_cluster.yaml

[66] https://galaxy.ansible.com/grycap/spark

[67] https://raw.githubusercontent.com/grycap/ec3/tosca/tosca/custom_types.yaml

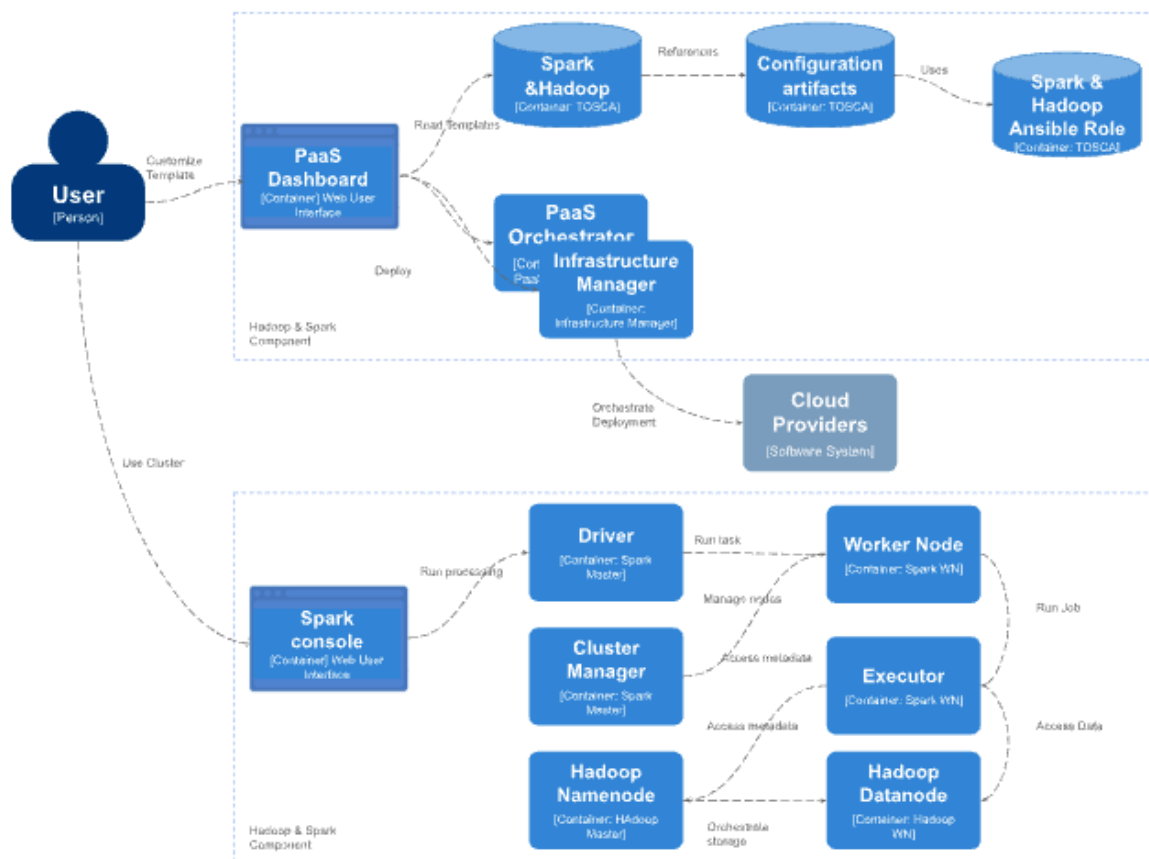[68] https://galaxy.ansible.com/grycap/hadoop

Figure 13 - General architecture for using Apache Hadoop and Apache Spark for data analytics

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).

6.6 KubeFlow clusters

This section details how Kubeflow clusters are being used in the Big Data Analytics subsystem. **Figure 14** shows the generic view of the architecture needed to us KubeFlow clusters.

6.6.1 General Description and functionalities

Kubeflow[69] 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

---

[69] https://www.kubeflow.org/

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.

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.

6.6.2 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.

6.6.3 Technology stack

KubeFlow is installed through an Ansible recipe[70] on top of an elastic Kubernetes cluster.

---

[70] https://raw.githubusercontent.com/grycap/ec3/tosca/tosca/artifacts/kubeflow.yml
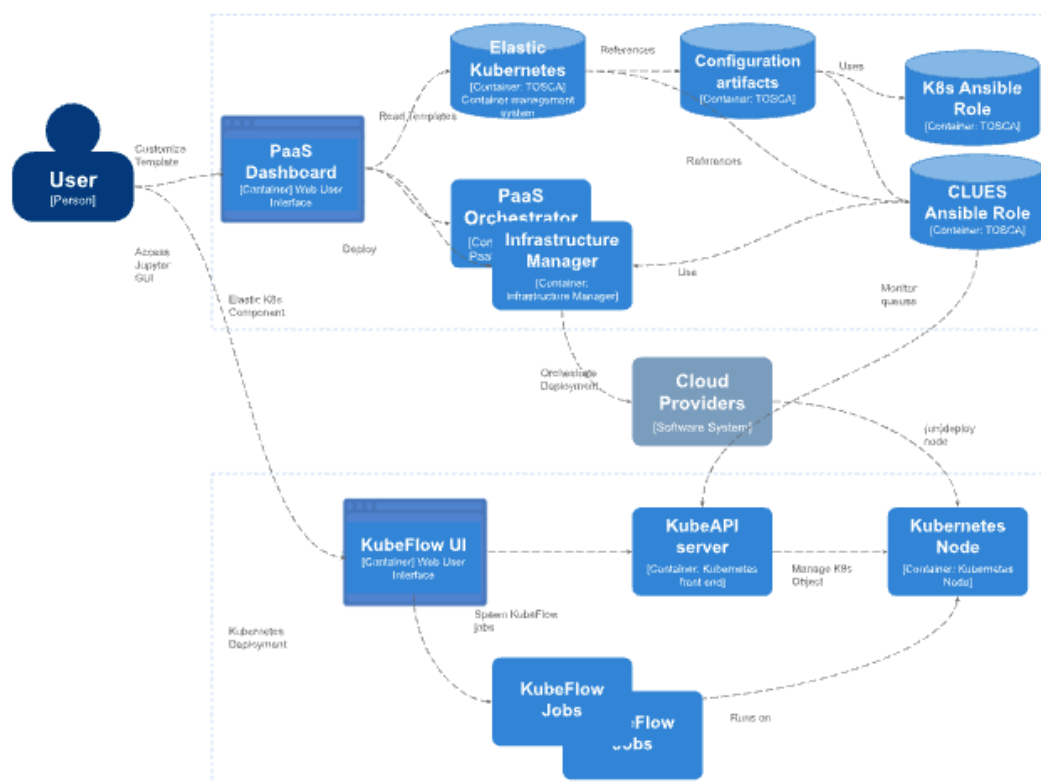
Figure 14 - General architecture for using Kubeflow for data analytics

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).

6.7 Ophidia Cluster

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

6.7.1 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 Sciences, Life Sciences) and with very heterogeneous sets of data.

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.

6.7.2 Interfaces

Once deployed, users can interact with the JupyterHub interface. JupyterHub provides a web-based interface where users can launch a Jupyter server where the user can use notebooks and access computational resources. From within a Jupyter notebook, users can launch 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.

6.7.3 Technology stack

Ophidia is installed through an Ansible recipe[71] on top of an elastic Kubernetes cluster.
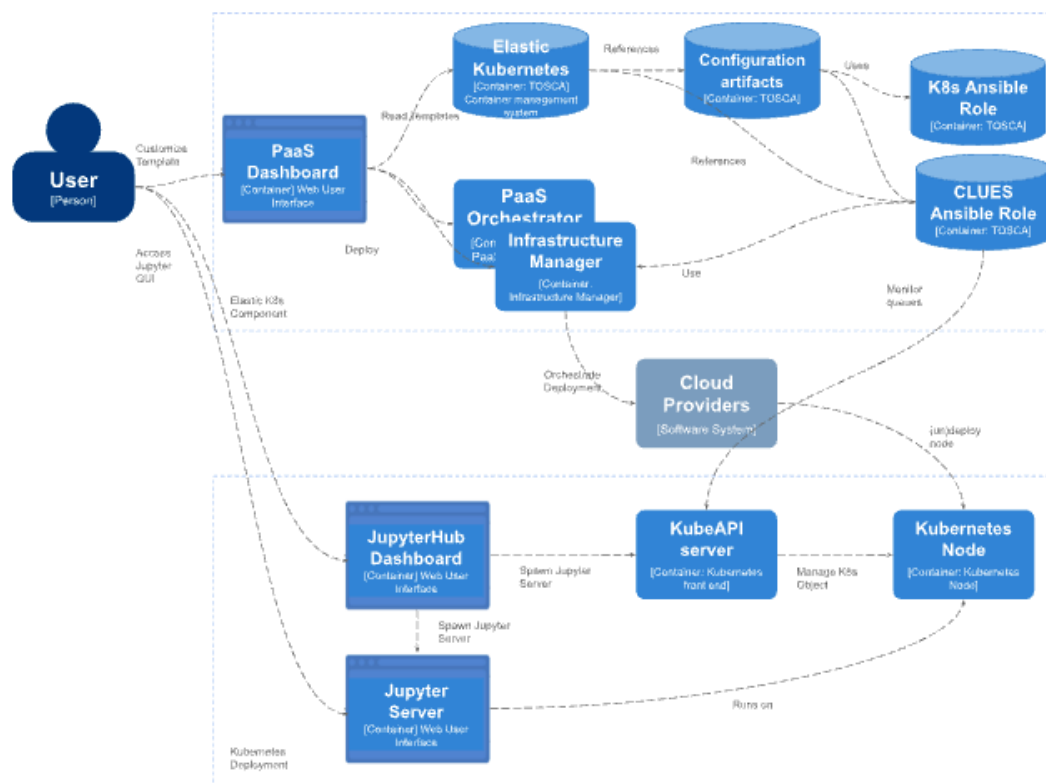

Figure 15 - General architecture for using Ophidia clusters for data analytics

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

---

[71] https://github.com/grycap/ec3/blob/tosca/tosca/artifacts/enes/enes.yml

D6.1 Report on requirements and core modules definition

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.

7 Conclusions

The first version of the interTwin Digital Twin Engine (DTE) core modules has been designed taking into consideration the requirements from the Digital Twin applications in the initial nine months of the project. 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.

The design also includes a list of technologies that have been analysed and selected for integration and extension/customisation. This Design will be updated following the first release of the component and early adopter from the DT applications, DTE thematic module integrations and requirements also from the DTE infrastructure.

The next deliverables on WP6 include in October 2023 the description of the first release of the core components (D6.2). An update on the core requirements addressing more specifically questions such as Data Fusion, and also the requirements arising from the feedback on the first release of the core services will be provided in D6.3 already during the project second reporting period.

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**