D1.1 Requirements, Metrics and Architecture Design

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	30.06.2021
Dissemination Level	PU
Nature	R
Author	Jorge Ejarque (BSC)
Contributors	Rosa M. Badia (BSC), Yolanda Becerra (BSC), Anna Queralt(BSC), François Exertier (Atos), Domenico Talia (DtoK), Salvatore Giampà (DtoK), Jedrzej Rybicki (FZJ), Bernd Schuller (FZJ), Björn Hagemeier (FZJ), Alessandro D'Anca (CMCC), Donatello Elia (CMCC), José Flich (UPV)
Reviewer	Jedrzej Rybicki (FZJ)
Keywords	Requirements, Architecture, metrics, workflows

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Change Log

Version	Description Change
V0.1	Proposed table of contents
V0.2	Components and HPC requirements
V0.3	Architecture description
V0.4	Pillars requirements
V0.5	Metrics and DA/ML frameworks differentiation
V0.6	Ready for internal review
V1.0	Final version with reviewer comments addressed

Table of Contents

1. Executive Summary

This document presents the work performed in WP1 regarding requirements gathering, design of the eFlows4HPC workflow platform and selection of the metrics to evaluate improvements in the Pillars workflows.

Requirements for the eFlows4HPC platform have been gathered from three sources (Pillars, Software components, and HPC sites). From Pillars we have gathered their requirements with regard the required functionalities for implementing, deploying and executing and managing the data of their different workflows. The process to obtain these requirements has been through a template proposed by WP1 asking for relevant information about the Pillars' workflows. Once the templates were filled, several meetings between WP1 and each of the Pillars (WP4, WP5 and WP6) were organized to discuss and understand the requirements. From this process, a total of 33 different requirements from the Pillars. The requirements are summarized in Table 1, Table 2 and Table 3, and provide requirements on the characteristics needed in the workflow management system, in the data and storage management, in the type of artificial intelligence tools, in the software deployment tools, portability, usability, interoperability and accessibility.

The second source of requirements are the software components which must be deployed in the computing infrastructure, either HPC simulators required by the workflow or components of the eFlows4HPC software stack such as Machine Learning (ML) or Data Analytics (DA) frameworks, runtimes and data management tools. We have studied the deployment and execution processes of this software to identify the required functionalities in the different phases of the workflow lifecycle. Table 4 summarizes the findings for this second set of requirements, which include aspects related to software deployment and access to specific HPC hardware to get the expected performance.

Finally, constraints from HPC data centres must be also considered in order to reduce the barriers for adopting the eFlows4HPC methodologies and software stack. Supercomputers are singular infrastructures shared by multiple users at the same time. System administrators have to preserve the security of the data processed while keeping the performance of the whole system. For this reason, supercomputers have several constraints in terms of accessibility and usability which have to be taken into consideration when producing software or services using these systems. To gather these constraints, we have conducted a survey with different HPC centres involved in the project. Table 5 summarizes the constraints posed by the HPC centres which mainly relate to the deployment of services that persist between executions or require external connectivity, the access protocol for login and data transfer, software management packages, file system and job scheduler. From the conclusions of the analysis of these surveys, a set of requirements were derived (see Table 6).

In total, a list of 38 requirements were derived. From this list of requirements, we have identified the components that can provide the required functionality (Table 7).

Another set of activities of the WP1 in this initial phase was devoted to the design of the eFlows4HPC architecture. This document also presents the details of the first version of this architecture including the description of the components organized in functional groups (Workflow Accessibility/Reusability, Workflow Development, Workflow Deployment and Execution and Data management), the main usage cases and the component interactions derived from these cases. WP1 partners were organized in working groups to analyze and compare the different components that have similar or related characteristics. The overlaps, differences and possible interactions were discussed.

The difference phases in the lifetime of an eFlows4HPC workflow were designed: development, deployment and execution, defining what we have called HPC Workflows as a Service (HPCWaaS). In the first phase, the workflow developer with use the extended TOSCA, PyCOMPSs and Data logistics Pipelines to design the workflow. The workflow will be based on different types of components: HPC solvers or simulators, ML and DA tools, as well as data sources. These elements will be available in the Data Catalog, the Model Repository and the Software Catalog. Once available, the workflow will be published as a service in the Workflow Registry. Users will be able to select the workflow and request its deployment. For the deployment, the Ystia Orchestrator and the Data Logistics Service will collaborate to deploy the different workflow components in the HPC centres and in the auxiliary cloud (used for the services that cannot be executed in HPC due to its constraints). Finally, the workflow will be executed, orchestrated mainly by the PyCOMPSs runtime, in cooperation with the Ystia orchestrator. The workflows will be very dynamic, not only because the required executions (task graph) is generated at run but also because the PyCOMPSs runtime is able to react to failures and exceptions, enabling to define a very dynamic and adaptative behaviour. To exploit new architectures such as accelerators or the EPI, specific optimized kernels will be integrated in the workflow executions. The envisaged environment also integrates solutions for persistent storage (dataClay and Hecuba) that will enable to exploit new storage paradigms, and optimized kernels.

The last part of this deliverable provides the definition of a set of selected common metrics according to the aspects related to the project technical objectives. These metrics will be used in different phases of the project to evaluate the improvements introduced by eFlows4HPC technologies in the pillars' workflows.

2. Introduction

Traditionally, High-Performance Computing (HPC) has been used to provide computational resources, software environments and programming models to enable the execution of large-scale e-science applications with the objective of generating predictions of real processes (weather forecasting, wave propagation, protein interaction ...). Recently, with the introduction of Big Data and Artificial Intelligence (AI) technologies, these applications have evolved to more complex workflows where traditional HPC simulations are combined with data analytic (DA) and machine learning (ML) algorithms. However, the combination of these different technologies in a single workflow require to dedicate a lot of effort to manage the integration of different frameworks in different phases of the workflow lifecycle. Starting from the development phase, where the integration of different workflow HPC, DA and ML parts requires additional programming efforts, passing through the deployment phase, where different tools and frameworks must be deployed in the infrastructure, and the execution phase, where the execution of all the different components must be orchestrated in a dynamic and intelligent way.

The eFlows4HPC project aims at delivering a workflow platform that consists of the software stack and an additional set of methodologies that will enable the integration of HPC simulation and modelling with big data analytics and machine learning in scientific and industrial applications. From one side, the eFlows4HPC software stack aims at providing the required functionalities to manage the lifecycle of such complex workflows; form the other side, it introduces the HPC Workflow as a Service (HPCWaaS) concept. It will apply the Function as a Service (FaaS) concept to the HPC environments which will hide all the complexity of a HPC Workflow deployment and execution to end users. These project outcomes demonstrate, through three application Pillars

with high industrial and social relevance (manufacturing, climate and urgent computing for natural hazards), how the realization of forthcoming efficient HPC and data-centric applications can be developed with the proposed novel workflow technologies.

This document reports the work performed in requirements gathering and the eFlows4HPC architecture definition. It is organized as follows. Section 3 reports the process of gathering the requirements from the different eFlows4HPC stakeholders (Pillars, Software components, and HPC sites). Then, the software stack architecture, the main usage cases of the HPCWaaS methodology, and the relation of the requirements with the software stack components is presented in Section 4. Finally, Section 5 describes the metrics selected to evaluate the implemented workflows according to the main technological areas related to the project objectives.

3. Requirements & Constraints

The requirements gathering process for the eFlows4HPC platform has been split in three main parts. The main source of requirements are the project Pillars. These pillars are the uses cases representing the user communities (manufacturing, climate and urgent computing) which will benefit from the complex workflows targeted by the eFlows4HPC project. They will drive and validate the implementation of eFlows4HPC platform providing the required functionalities for implementing, deploying and executing and managing the data of their different workflows.

The second source of requirements are the software components which must be deployed in the computing infrastructure, either HPC simulators required by the workflow or components of the eFlows4HPC software stack such as ML or DA frameworks, runtimes and data management tools.

Finally, constraints from HPC data centres must be also considered in order to reduce the barriers for adopting the eFlows4HPC methodologies and software stack.

3.1. Requirements from Pillars

Collecting the requirements from the pillars has been performed in collaboration between WP1 and WP4, 5 and 6 which correspond to the different Pillars of the project. This process is usually complex due to the differences in the terminology used in the Pillars domain and the one used by software developers and architects. To coordinate this process WP1 proposed a template (Appendix A) pointing out the relevant information to extract from the Pillars' workflows. This information was trying to identify what are the requirements for the eFlows4HPC platform in the different phases of the workflow lifecycle. These templates have been filled by the teams working on the different workflows defined by the pillars, and their results have been analyzed by the Pillars teams together with the WP1 team. The results of this analysis are reported in deliverables D4.1, D5.1 and D6.1, and the following tables are the summary of the requirements from the different pillars. These tables contain an identifier (ID) to easily identify the source of the requirement (P1: Pillar 1, P2: Pillar 2, and P3: Pillar 3), the name, description and priority assigned by pillar teams. More details about these requirements can be found in the mentioned deliverables.

Table 1. Summary of requirements from Pillar I: Digital twin in Manufacturing.

ID	Name	Description	Priority
P1-1	Distributed SVD	Requires an optimized distributed execution of the Singular Value Decomposition (SVD) algorithm to analyze large scale matrices which can exceed the memory of different computing nodes of a cluster.	Must
P1-2	Storing of hyper-reduced model	Requires storing and transferring the meshes and the trained ML model needed to reconstruct the hyper-reduced model, together with the solver executable needed to run it.	Must
P1-3	ANN model	Require Artificial Neural Networks (ANN) (probably convolutional) to train autoencoders. This may provide an attractive option to improve the reduction ratio of the reduced model. Here both training data and the output to be used in the inference step need to be saved.	May
P1-4	Clustering model	Clustering algorithms as an option to improve the reduction ratio. Here both training data and the output to be used in the inference step need to be saved.	Should
P1-5	Persistent storage	Requires persistent storage for data to be consumed between the steps	May
P1-6	Restart	Workflow programming and management have to allow re-start the Reduced-Order-Model (ROM) computation according to validation results.	Should
P1-7	Workflow orchestration	Workflow management is also required through the phases to coordinate the execution of the different computing steps.	Must
P1-8	ML inference	Simulation code requires access to the ML trained model.	May
P1-9	Hyper Reduced Model Deployment	Once the hyper reduced model is computed it may be deployed in a computing infrastructure,(such as a Cloud) to be accessible and reusable by final users. This requires to deployment the model together with software and data needed to run a complete hyper-reduced model from scratch.	May

Table 2. Summary of requirements from Pillar II: Dynamic and adaptive workflows for climate modelling.

ID	Name	Description	Priority
P2-1	Execution robustness	Management of fault tolerance during the workflow execution including checkpoints or retries. For example, during a large execution if a node fails, the workflow must be able to recover and continue to the end.	Should
P2-2	Portability	Workflow components should be portable to various types of HPC infrastructures.	Should
P2-3	Integrated workflow management	Requires the management of task dependencies, execution of parallel simulations on different HPC infrastructures, management of batch jobs (submission, monitoring, cancellation), management of conditional paths in a transparent way.	Must
P2-4	Integration with long-term archive/repository storage	Results may be stored in long-term storage for archiving purposes, second use (e.g., downstream services) and/or to satisfy FAIRness policies.)	May
P2-5	Workflow adaptability	Capability to easily manage, cancel, replace and add components invocations in the workflow, for instance allowing the execution starting from the n-th step.	Should
P2-6	Access to intermediate in-memory results	The workflow should be able to retrieve data/intermediate outputs of the running processes directly from memory.	Must

P2-7	AI integration for ensemble member pruning	Support for applying Machine Learning techniques on intermediate data of running members to compute the members that will be discarded at a given step of the simulation.	Should
P2-8	ML/DL capabilities	Requires the support for training and inference of Neural Network models for example for Tropical Cyclone detection.	Must
P2-9	DA capabilities	Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis.	Must
P2-10	High Performance Computing support	Climate models have to be executed on computing infrastructures capable of providing a large amount of processing and memory resources.	Must
P2-11	Multi-member analysis	Support for concurrent execution of sub-workflows starting from different inputs (configurable) and comparison of the sub-workflows results.	Must

Table 3.Summary of requirements from Pillar III: Urgent Computing for natural hazards

ID	Name	Description	Priority
P3-1	Urgent computing access	Priority access to HPC computational resources.	Must
P3-2	Data accessibility	Some data required in HPC computations is stored in external repositories and some computational results must be required for post processing in external services. So, some High-performance data management mechanism between external repositories and HPC facilities is required in order make data accessible in the infrastructure required by the computation	Should
P3-3	Data replication	Redundancy of data is required in different phases of the workflow execution. Source data must be replicated in different location to assure a high-availability computation as well as avoiding time consuming data transfers (e.g. computational meshes). Intermediate data generated by large computation must be also considered in order to avoid losing data in case of failures.	Must
P3-4	Execution robustness	Support for the management of fault tolerance during the workflow execution including checkpoints or retries. For example, during a large execution if a node fails, the workflow must be able to recover and continue to the end.	Must
P3-5	Infrastructure interoperability	Interoperability between computations performed in different infrastructures used in the Pillar (e.g., HPC clusters and external servers).	Must
P3-6	Portability and Reusability	Workflow and its components must be portable and reusable to several infrastructures and users.	May
P3-7	Streaming data source	Management of streaming data sources in real-time from external agencies or servers	Must
P3-8	Integrated workflow manager	Support for the management of task dependencies, execution of parallel simulations on different HPC infrastructures, management of batch jobs (submission, monitoring), management of conditional paths, and coordination of microservices invocations	Must
P3-9	Integration with permanent storage	Support for access to external data repositories (R/W) such as EUDAT Data Storage services (e.g. B2DROP). Support for final storage in long-term storage for second use and/or to satisfy FAIRness policies.	Must
P3-10	Inference with online/offline ML models	Support to the use of inference from Online and/or offline trained ML models by Earthquake and Tsunamis emulators as steps in its workflows.	Must
P3-11	DA integration	Predictive and prescriptive data analytics to assist some building blocks in analysis and decision tasks.	May
P3-12	Workflow malleability	Capability to cancel and add new components invocations at run-time.	Should

Note that some requirements from different pillars have defined similar functionalities such as P2-1 and P3-4 regarding execution robustness; P2-2 and P3-6 about components portability and reusability; P1-7, P2-3 and P3-8 about workflow management and orchestration; and P2-4 and P3-9 regarding integration with permanent storage. From now on, these similar requirements will be considered a single requirement in order to avoid effort duplication.

3.2. Requirements and Constraints from Components

Another important goal of the project is enabling the portability and reusability of complex workflows simplifying its deployment and execution. For this reason, the eFlows4HPC software stack must support the deployment and coordinate the execution of the software components required by required by the Pillars' workflows (mainly HPC software and DA/ML frameworks) as well as the eFlows4HPC software stack components which must also be deployed in the computing infrastructure to manage the workflow execution and data.

We have studied the deployment and execution processes of this software to identify the functionalities that the eFlows4HPC platform should support to achieve the mentioned goal. The following table provides a summary of these functionalities. The first part of the table focuses on the software deployment aspects. It includes the support for the different deployment models required by the used software as well as the access to the specific HPC hardware to ensure the execution will get a similar performance as if it was manually deployed by the user.

Table 4. Requirements from software components

ID	Name	Description	Priority
CMP-1	Access to HPC specific devices	Workflows developed with eFlows4HPC stack must be able to access the specific HPC hardware such as High-Performance networks, accelerators or special CPU vector instructions.	Must
CMP-2	Support optimized kernels	Workflows developed with eFlows4HPC stack must be able to support the architecture-optimized kernels and libraries.	Must
CMP-3	Service deployments	The eFlows4HPC software stack should support the deployment of data bases and services required by the DA and ML frameworks in auxiliary Cloud and HPC centers.	Must
CMP-4	Service invocation	Workflows developed with eFlows4HPC stack must support the invocation of services.	Must
CMP-5	Multi-node execution support	Workflows developed with eFlows4HPC stack must support the execution of applications distributed in different computing resources (such as MPI applications).	Must
CMP-6	Multicore execution support	Workflows developed with eFlows4HPC stack must support the execution of applications with multi-threaded/multi-process using several cores.	Must

3.3. Constraints from HPC Centers

The last source of requirements is provided by HPC centres. Supercomputers are complex infrastructures shared by different users at the same time. System administrators have to preserve the security of the data processed while keeping the performance of the whole system. For this reason, supercomputers have several constraints in terms of accessibility and usability which have to be taken into consideration when producing software or services using these systems. Not fulfilling these constraints can prevent the adoption of a certain technology in these computing environments.

To gather these constraints, we have conducted a survey with different HPC centres. We have prepared a questionnaire (available in Appendix B) with questions related to the main objectives of the eFlows4HPC platform including access and security aspects, available services, software management tools, and execution restrictions. In the first phase of the project, this questionnaire has been sent to HPC system administrators which are involved in the project or used by the Pillars to get feedback and identify common constraints and produce a first set of requirements for the eFlows4HPC architecture. In the second phase, we plan to extend this questionnaire and to include other HPC sites in PRACE and EuroHPC.

Table 5. Summary of HPC sites survey

Site		BSC	CMCC	FZJ	PSNC	AWI	DKRZ
Access & Security	**Access**	SSH	SSH(VPN)	SSH	SSH/GSISSH	SSH	SSH
	Identity	SSH Keys	VPN Cert / SSH Keys	SSH Keys	SSH Keys, x509(PRACE)	SSHkeys	SSHkeys
	UNICORE	Testing	No	Yes	No	No	No
	In Conn.	SSH	Only VPN	SSH	SSH, GSISSH, ARC	Only from AWI	On request
	Out Conn.	Not Allowed	Allowed	No SSH	ssh, gsissh, arc, http, ftp	Allowed	On request
Cluster Nodes Restrictions	**Login**	Restricted	No MPI	Restricted	Restricted	Restricted	Restricted
	Service	Not Allowed	Yes	Not Allowed	Yes	No	On request
	Compute	No restrictions	No restrictions	No restrictions	No restrictions	No restrictions	No restrictions
Queue System Shared disk	**Queue system**	Slurm/LSF	LSF	Slurm	Slurm	Slurm	Slurm
	Shared disk	GPFS	GPFS	GPFS, HPST(BB)	cNFS(Home), Lustre	BeeGFS	Lustre
Software Management	**Modules**	Yes	Yes	Yes	Yes	Yes	Yes
	Installation tools	Conda (EB and Spack testing)	Conda	Easy-build	Conda	Not provided	Conda Spack
	Containers	Singularity	Singularity	Singularity	Singularity	Singularity	Singularity
Data Infrastructure	**Data Interfaces**	SCP GridFTP	SCP FTP, GridFTP HTTP, openDAP	SCP SFTP UFTP	SCP	SCP	SCP GridFTP Swift/S3
	Storage Levels	(NVMe), SSD, GPFS HSM:GPFS/Tape	GPFS Archive	HPST GPFS HSM	cNFS Lustre (no specific clean up after job)	/dev/shm, SSD BeeGFS, Tape	Lustre (HDD, NVMe) HSM

Table 5 provides a summary of the results obtained in this survey, where we can see that in some aspects, the answers are quite dispersed but in others we can see commonalities. The main findings of the survey are:

- We cannot rely on services which must persist between executions or require external connectivity. Most clusters either do not provide nodes suitable to install them in a user level or it could have connectivity or execution restrictions.
- SSH and SCP is the common remote shell and transfer protocol supported by all the systems

- For software management, all systems use Modules for managing the environment of the installed software, and for containers, Singularity is supported by all the systems.
- All systems have a shared file system, however other types of storage and its usage varies depending on the site.
- Slurm is the most extended queue system but not the only one. Implementing tools supporting this system will cover a considerable number of sites. However, the queue system is the key component in an HPC site, so they are going to adopt the eFlows4HPC stack if it is not supported it cannot be extended to support it.

Table 6. Requirements from HPC sites

ID	Name	Description	Priority
HPC-1	HPC cluster access support	The interactions between eFlows4HPC software stack and the HPC systems must at least support the SSH protocol	Must
HPC-2	HPC data transfers support	Transfers to/from HPC clusters must support at least the SCP protocol	Must
HPC-3	Singularity container support	The usage of containers in the HPC system must be compatible with singularity containers	Must
HPC-4	Infrastructure service deployment	The eFlows4HPC software stack cannot rely only on services which require to be installed with privileged nodes or users in the supercomputing clusters.	Must
HPC-5	Queue system	Supported queue systems must include at least Slurm and must provide extension mechanisms to provide other queue systems.	Must

4. Architecture

This section provides the details of the designed architecture for achieving the eFlows4HPC project objectives and the requirements presented in the previous section. It is organized as follows, first an overview of the architecture is presented followed by a description of the main functionalities of the architecture components.

4.1. Overview

The eFlows4HPC software stack will be composed of a set of software components, organized in different functional groups (Figure 1). The first group provides the syntax and programming models to implement these complex workflows combining typical HPC simulations with HPDA and ML. A workflow implementation consists of three main parts: a description about how the software components are deployed in the infrastructure (provided by an extended TOSCA [1] definition); the functional programming of the parallel workflow (provided by the PyCOMPSs Programming Model [2]); and data logistics pipelines to describe data movement to ensure the workflow data are available in the computing infrastructure when required.

The second group consists of a set of services, repositories, catalogues, and registries to facilitate the accessibility and re-usability of the implemented workflows (Workflow Registry), their core software components such as HPC libraries and DA/ML frameworks (Software Catalog) and its data sources and results such as ML models (Data Registry and Model repository).

Finally, the lowest groups provide the functionalities to deploy and execute the workflow based on the provided workflow description. From one side, this layer provides the components to orchestrate the deployment and coordinated execution of the workflow components in federated computing infrastructures. On the data management side, it provides a set of components to

manage and simplify the integration of large volumes of data from different sources and locations with the workflow execution.

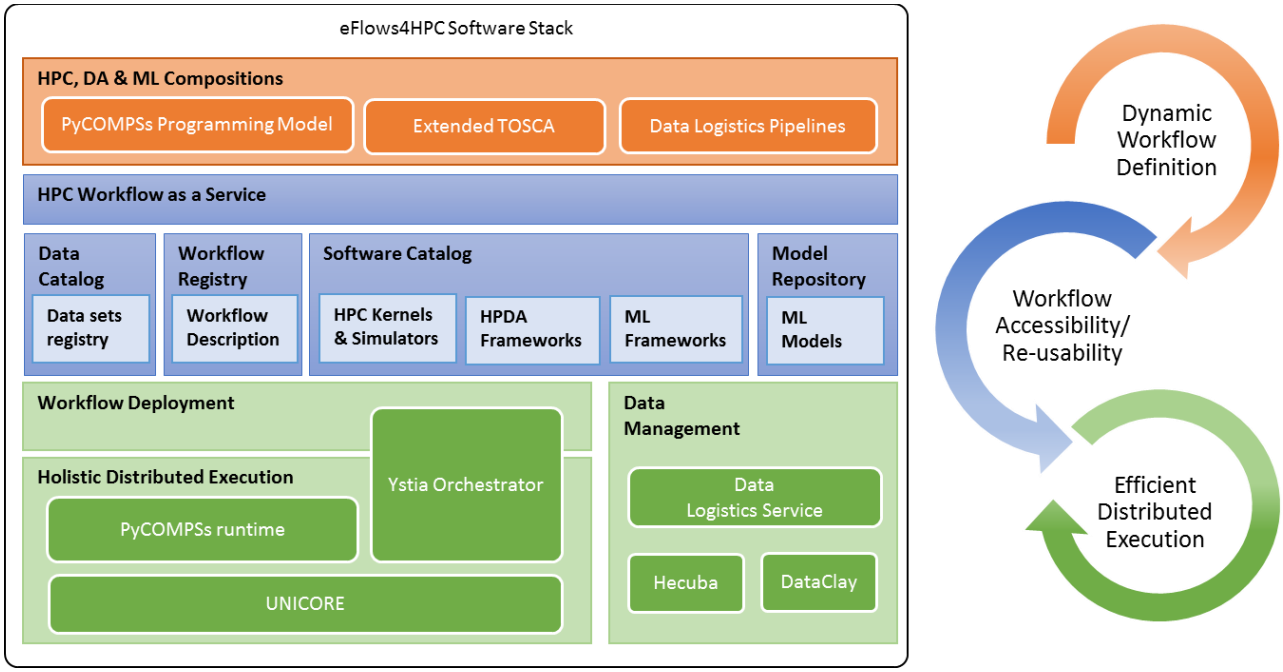

Figure 1. eFlows4HPC Software Stack Overview.

4.2. Component Descriptions

Next paragraphs provide the details of the functional groups of the eFlows4HPC software stack and the description of the components in each group

4.2.1. Workflow Definition

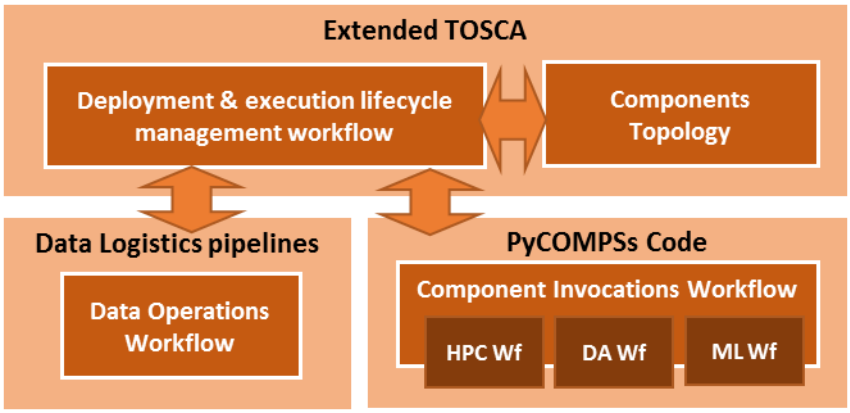

Figure 2. Workflow Definition Overview.

As introduced before, the second group is in charge of providing the syntax and programming models to implement complex workflows combining typical HPC simulations with HPDA and ML phases. The workflow definition consists of three parts. At a higher level, an extended version of the TOSCA standard is used to describe the high level execution lifecycle of the workflow. At a lower level, the PyCOMPSs programming model that provides a lower level task-based parallelization, which is used to orchestrate the invocations of the different workflow computations which can include just a simple task execution or sub-workflows implemented with different HPC libraries or DA/ML frameworks. Finally, the Data Logistics pipelines define the

required data operations to properly execute the workflow. Next paragraphs provide more details about these models.

4.2.1.1. Extended TOSCA

The higher-level part of eFlows4HPC workflow description will be defined using the TOSCA standard [1]. A workflow in the eFlows4HPC platform will correspond to a TOSCA application, designed as an assembly of components, that may be interconnected and that defines standard operations to manage their execution lifecycle. TOSCA defines two types of workflows.

- Declarative workflows: they are workflows to install, start, stop, uninstall (and run for jobs) based on the component's standard operations and the relationships between these components.
- Imperative workflows: they user-defined workflows that can override declarative workflows

TOSCA extensions and imperative workflows will be leveraged, to address specific kinds of services to deploy, such as HPC jobs, or to implement specific aspects addressed in the project like dynamic workflows or data awareness. To support dynamicity in this part of the eFlows4HPC workflow, users can define variables in the TOSCA descriptions whose value can be updated by workflow's steps allowing the possibility to modify other workflow steps. TOSCA descriptions can also support Inputs/Outputs, span hybrid/heterogeneous infrastructures and rules based scheduling (Drools-Like / Cron-Like).

Developers can define the high-level workflows with TOSCA either in plain-YAML or graphically designed using the Alien4Cloud[1] web user interface (UI).

4.2.1.2. PyCOMPSs Programming Model

PyCOMPSs [2] is the Python binding of the COMPSs framework [3] that facilitates the development of parallel computational workflows for distributed infrastructures. It offers a programming model based on sequential development - the application is a plain sequential Python script - where the user annotates the functions to be run as asynchronous parallel tasks with Python decorators. This decorator also contains a description of the function parameters, such as type and direction, etc. which is vital for building the dependency graph where tasks are represented as nodes and data dependencies between tasks as edges.

At execution time, tasks are created for each decorated function invoked from the code and forwarded to the COMPSs Runtime which handles data dependency analysis, task scheduling and data transfers. The task creation is performed in an asynchronous way, and once the runtime has added a given task to the dependency graph, the execution of the main Python code continues, possibly generating new tasks. With this aim, PyCOMPSs manages future objects: a representative object is immediately returned to the main program when a task is invoked. A future object returned by a task can be involved in subsequent asynchronous task calls and PyCOMPSs will automatically find the corresponding data dependencies without requiring to wait for the actual result of the task.

PyCOMPSs can be extended to support different tasks and data types. For instance, it is able to support multi-node and multi-core tasks, which are used to integrate MPI and OpenMP

[1] http://alien4cloud.github.io/

applications as tasks of a workflow just adding a specific @mpi decorator in a function which represents the execution of an MPI application [4]. Following the same approach, it can be used to integrate the execution of binaries (@binary) or containerized applications (@container). Users can also use the @constraint decorator to define resource requirements for a task such as number of CPUs, GPUs or memory size.

Regarding data types, PyCOMPSs not only supports standard parameters (primitive types, files, objects and collections or dictionaries of the mentioned types), it also supports the definition of streams to exchange data between tasks. It allows users to create hybrid workflows combining data and task flows in the same application [5].

Another important feature of PyCOMPSs is the workflow dynamicity. PyCOMPSs workflows are created at execution time from a Python code, so it is very easy to program different workflow branches from previously generated values. However, dynamicity can be also generated by failures or exceptions. Scientific workflows usually implement hyper-parameter searches in huge spaces performing loads of simulations with different input parameters. It is very likely that some of these simulations fail, but they should not imply a failure in the whole workflow. For those simulation tasks, developers can provide hints to ignore these failures, or to cancel their successors. It is also possible that a solution is found before finishing all the execution. For this purpose, PyCOMPSs supports parallel try-except blocks where if a specific exception is raised in one of the tasks of the block all the remaining tasks will be cancelled [6].

4.2.1.3. Data Logistics Pipelines

Workflows for Data Logistics Service are created in Python as Directed Acyclic Graphs (DAGs) and called Pipelines. They have some similarities with ETL (Extract, Transform, Load) approaches. Pipelines include additional metadata describing execution details: frequency, retries, parallelism, backfill, etc.

The idea is to build the workflows as pure functions, no orthogonal concerns should be included (invocation/scheduling/input-output locations). Such an approach makes testing of the pipelines easier. Furthermore, such functions achieve idempotence. In case we have to rerun the pipeline, the results should be the same, e.g., a measurement series is recreated rather than new measurements are added. This results in a powerful ability to recreate the data sets in a reproducible way. If some tasks fail, they can be restarted. In the worst case, the work will be done twice but the data will not be duplicated.

DAGs are composed of operators and sensors. Sensors are able to detect new data in the sources whilst the operators are responsible to conduct processing, transformations, and transfers. There are a number of operators already available in the Data Logistics Service, and the list can be easily extended.

4.2.2. Workflow Accessibility and Reusability

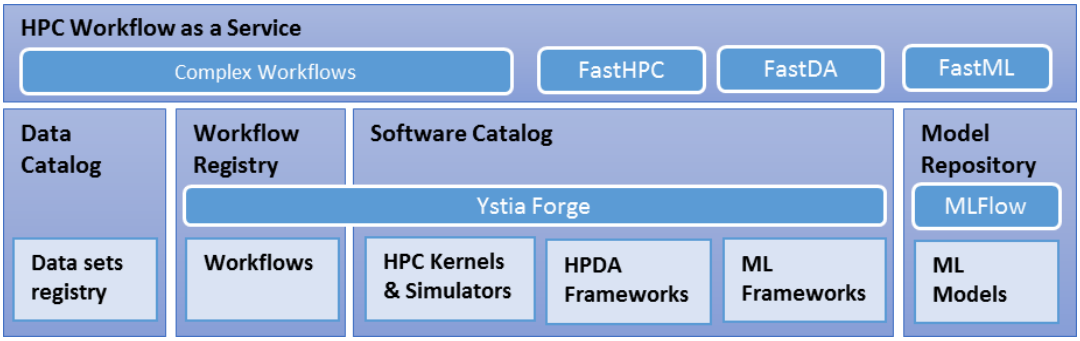

Figure 3. Workflow Accessibility and Reusability Layer.

This Workflow Accessibility and reusability group provides the required functionalities to manage workflows and their main components in an easily accessible and reusable way. It is composed by the HPC Workflow as a Service component which provides the main entrypoint for users to simplify the workflow accessibility, and by different catalogs, registries, and repositories to enable the reusability of the workflow components. Next paragraphs provide details about the functionality provided by these components and their baseline technology.

4.2.2.1. HPC Workflow as a Service (HPCWaaS)

The HPC Workflow as a Service provides an API and GUI to manage simple and complex workflows. It relies on the underlying components Data Catalog, Workflow Registry, Software Catalog and Model Repository (described in the following sections), to support workflow construction to respectively:

- specify access to data within workflows,
- re-use, customize, store workflows,
- specify usage of software components within workflows,
- use existing ML models and store resulting models from workflows.

Applications to be managed in the project typically make use of HPC simulation, Data Analytics, and/or ML. Therefore, corresponding workflows will include any combination of these respective kinds of processing, or domains.

Workflows to be handled may then be categorized as

- simple workflows of three kinds:

  1. Traditional HPC simulation workflows, composed of intensive computation on high volumes of simulation data, represented by the “FastHPC” module in the figure.
  2. Data Analytics workflows, involving Data Analytics software usage, typically used for data preparation or post simulation processing data analysis, represented by the “FastDA” module in the figure.
  3. Machine Learning workflows, designed to manage ML Models training and inference (MLOps), handled through the FastML component.

- Complex workflows are any combination of these simple workflows. A complex workflow typically combines Data Analytics workflows (e.g. for pre and post processing), HPC simulation workflows complemented by ML workflows.

While Complex Workflows, FastHPC and FastDA workflow interfaces will be defined within the scope of the project, the FastML product API/GUI will be the baseline for ML workflows. Currently FastML will provide the following features (through a Rest API, CLI, and GUI):

- Model Training Management focused on HPC deployment (including User Management, permissions, Cluster Resource Monitoring)
- Support of TensorFlow, Keras, PyTorch, scikit-learn, although any other ML/DL framework can be used as far as its Docker Image is made available
- Model tuning through Hyper Parameter Optimization features (automated GridSearch and RandomSearch for now)
- Jupyter notebooks integration including "Fairing" which is the ability to launch any code from a Notebook on an HPC cluster as a job

The HPCWaaS component will be used as the main entry point for workflow developers and users, whose main usages and interactions are described in Section 4.3.

4.2.2.2. Data Catalog (DC)

The following describes the architecture of eFLows4HPC Data Catalog. The service will provide information about data sets used in the project. The catalog will store information about their locations, schemas, and additional metadata.

The Data Catalog is implemented with FastAPI[2], a modern Python framework allowing for flexibility, easy extensibility in the future, and quick deployment. The Data Catalog offers an API well-document in the (industry-standard) Swagger[3] format as well as a web-based GUI.

The primary use case for the Data Catalog is to store information about the data sets which can be then used by the Data Logistics Service to facilitate the required data movements. Secondly, the Catalog will improve the visibility of the data sets used and created in the project, enabling possible reuse and collaboration in spirit of FAIR data principles. To this end, the Data Catalog offers a possibility to describe the items with a rich set of metadata.

4.2.2.3. Workflow Registry (WR)

As introduced in Section 4.2.1, Workflows will be described at the higher level using the TOSCA standard [1], they will be managed as TOSCA Topology templates (also named TOSCA Application templates), as described in Section 4.2.1.1. Using this paradigm, the lifecycle of an application can be described, from resource allocation, to deployment and execution. Workflow designers will use this formalism to describe a high-level application workflows which will be linked with the PyCOMPSs workflows and data logistic pipelines to define the lower level computational and data workflows. These descriptions will be stored in Workflow Repository, from which descriptions can be fetched either to deploy and run the corresponding workflow, or to be customized to address a new use case.

[2] https://fastapi.tiangolo.com

[3] https://swagger.io

The Ystia suite implements a “TOSCA Application Templates'' repository. Its Github project proposes a public instance of such a repository[4]. It will be more appropriate that an instance dedicated to the eFlows4HPC project will be created. The repository management is implemented by the Alien4Cloud, component of the Ystia software stack, which provides an API and GUI for this purpose. The Ystia orchestrator (Yorc), described in Section 4.2.3.1 will then be able to handle the deployment of components and application templates stored in this repository. Alien4Cloud also provides a GUI to facilitate the design of TOSCA applications.

4.2.2.4. Software Catalog (SC)

The Software Catalog is in charge of storing the software components descriptions that may be used within the workflows. It will contain the TOSCA descriptions of the software to be used in definition of high-level workflow. The stored software description are TOSCA Components that describe the software component lifecycle, i.e., how they should be provisioned, deployed, started, stopped, and undeployed on the infrastructures when used in a workflow.

As described in Section 4.2.2.3, the high-level workflow is itself described within a TOSCA Application template, which is built as a composition of the TOSCA Components stored in this catalog.

The Software Catalog will be handled in the same TOSCA repository, implemented by the Ystia suite (and materialized as “Ystia Forge” in Figure 3) as the Workflow Registry. The TOSCA repository stores both TOSCA Application Templates and TOSCA Components.

4.2.2.5. Model Repository (MR)

The Model Repository shall store, ML models as well as information about the ML lifecycle like performance metrics and training parameters. Currently MLflow[5] is considered the candidate technology to offer this functionality.

The Model Repository offers the possibility to track the lifecycle progress (MLflow Tracking). The interaction with this part can be done either directly from the scientific code (in any language), using its CLI. The tracking allows users to see and compare the different versions of the models and their performance (also using web-based GUI). MLflow Projects allows to package data science code and models. This enables sharing of the computation outcomes and reproducibility of the experiments. The results of tracking as well as the project itself can be registered to make them available to be viewed by other researchers.

There are currently some first efforts undertaken to integrate the MLflow Tracking into FastML. The further plans include creation of a central repository where ready models created in the project will be published for the sake of transparency, reusability, and project visibility. Lastly, to support everyday scientific work, an on-demand ad-hoc instance of MLflow can be provided.

[4] https://github.com/ystia/forge

[5] https://mlflow.org

4.2.3. Workflow Deployment and Execution

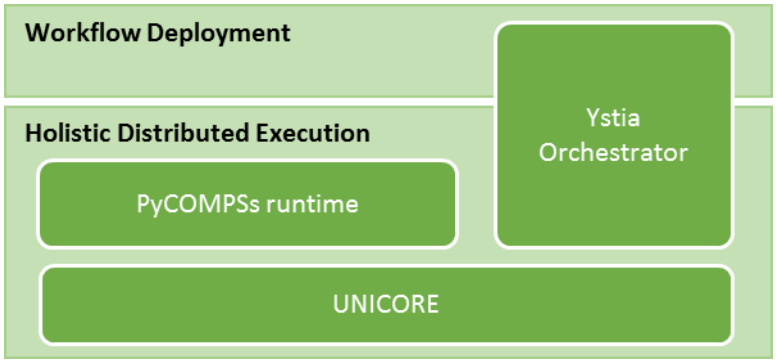

Figure 4. Workflow Deployment and Execution Layer.

The workflow deployment and execution functionalities in eFlows4HPC are provided by the Ystia Orchestrator, which manages the deployment and execution lifecycle at a higher-level, the COMPSs runtime, which provides the lower level execution orchestration, and UNICORE which provides services for uniform access to the federated computing infrastructure.

4.2.3.1. Ystia Orchestrator (Yorc)

The Ystia orchestrator suite[6] will be used for the high-level workflow execution, which consists of components provisioning and deployment on the available infrastructures (including resource allocation), applications launching and stopping, and the subsequent undeployment.

Yorc supports application lifecycle management over hybrid infrastructures, it is TOSCA native and designed for large scale. It is the core orchestrator engine of the Ystia suite, which also includes a "Forge" for hosting TOSCA components and application templates, and relies on a companion software, Alien4Cloud, that provides functions to design TOSCA applications and to handle a catalog of TOSCA components and applications. Alien4Cloud provides a Web interface (as well as an API) to facilitate application design, deployment and supervision, as well as for handling the TOSCA catalog.

Yorc supports the concept of "jobs" (through a TOSCA extension), as well as Container based applications. It supports workflows based on TOSCA imperative workflows (as explained in Section 4.2.1.1).

4.2.3.2. COMPSs Runtime

The COMPSs runtime is the component to transparently manage PyCOMPSs tasks. PyCOMPSs creates tasks for each decorated function invocation in the user code and forwards them to the COMPSs Runtime, which asynchronously handles them. Once a task is submitted to the runtime it analyses the data used by the task detecting data dependencies from previous tasks. Based on these data dependencies, the available resources and data locations, tasks are scheduled and executed in the remote resources which can be located in clusters, grids or clouds (IaaS or CaaS offerings). The execution of tasks is performed in a transparent way for the user, requesting the required data transfers, spawning the computation according to the task type (binary, MPI, containerized ,...) in the allocated resources and synchronizing the tasks results when required.

[6] https://ystia.github.io/

The runtime is also in charge of applying the failure reaction policy defined by the user in an autonomous way, cancelling the required tasks (in case of cancel successors or exceptions raised in a try-except block) restoring the data coherence (setting default values or removing the expected generated data) and continue with the application execution [6].

Apart from the mentioned features, the COMPSs runtime can also infer the maximum parallelism of an application based on the generated task dependency graph. This information can be used to detect when the application can be accelerated with more resources or it is wasting resources. In these situations, the COMPSs runtime increases or decreases the resources to make an efficient use of resources if the computing infrastructure supports it [7].

The COMPSs runtime can be deployed in two modes. It can be deployed as a master-worker application, where the master node executes the main code and schedules tasks to the workers nodes which are in charge of executing the tasks. The second deployment mode is a multi-agent application where each computing node deploys a COMPSs agent which is able to analyze, schedule and execute tasks and collaborate with other agents to execute applications. It provides a more flexible runtime which is able to better adapt to nested applications and highly distributed infrastructures.

4.2.3.3. UNICORE

UNICORE (UNiform Interface to COmputing REsources)[7] provides tools and services for building federated systems, making high-performance computing and data resources accessible in a seamless and secure way for a wide variety of applications in intranets and the Internet.

UNICORE is an infrastructure-level service, which offers RESTful APIs[8] for HPC job submission and management (on top of a batch scheduler such as Slurm), data access, data movement and workflows. It is easily integrated with federated AAI solutions and the HPC site's user/project management. As an infrastructure service, it is deployed and operated by the HPC site. UNICORE solves the critical task of user authentication and authorization in a federated environment, and allows integration of HPC compute and data into web applications, without any compromise in security.

The UNICORE compute service component offers an abstraction layer over the site's batch resource manager (e.g. Slurm), which can be used to create portable jobs. However, also low-level interaction with the resource manager is possible. This component also provides access to the HPC site's file system(s), solving common data management issues such as data transfer and pre/post job data staging from/to external data sources.

The UNICORE workflow engine provides a REST API for submitting and managing workflows that can span multiple UNICORE-enabled computational resources. The workflow execution engine and workflow description (JSON format) offer a wide range of control constructs and other features: while, repeat and for-each loops, if-else blocks and plain groups are supported, which can be nested to any depth. Workflow variables can be defined, modified using scripts and used in jobs, if-else conditions or for loop control. Workflows can be halted at user-defined points and continued later, allowing to integrate user-made decisions and user modifications of workflow variables.

[7] https://www.unicore.eu

[8] UNICORE REST API documentation: https://sourceforge.net/p/unicore/wiki/REST_API

In summary, UNICORE can help solve integration of HPC into federated applications, allows for the creation of cross-site workflows and provides added value such as site-to-site data movement (between POSIX filesystems), access to external data or sharing of HPC data sets.

4.2.4. Data Management

Figure 5. Data Management Layer.

Data management features in eFlows4HPC can be split into two kinds of functionalities. From one side, the orchestration of data operations which is supported by the Data Logistics Service; and the in-memory/persistent storage management which is managed by Hecuba and dataClay.

4.2.4.1. Data Logistics Service (DLS)

Data Logistics Service provides scientific users with a means to prepare, conduct, and monitor data movement and transformations. The primary use case is the aggregation of data from distributed locations, transformation into the required form, and keeping the local copies up-to-date [8].

Data Logistics Service is based on Apache Airflow[9]. Its main parts are scheduler, metadata store, executor, and a set of workers. Airflow executes data transformation pipelines (DAGs) defined by users. As already explained in Section 4.2.1.3, these definitions are just a normal Python code, lowering the entry barrier. Once a DAG is created it will be passed to execution by the scheduler (based on pre-defined requirements like execute once or periodically). Executor dissects the DAG into single tasks and passes them over to available workers. Workers execute the jobs, and store the information about the execution in the metadata store. The content of the store can be used to monitor the correctness, and performance of the tasks' execution. The users can view the information through the GUI.

In the project the Data Logistics Service will be used in the phase of staging workflows to execution, making sure that the data required for computation are available. Also, the results of the computation can be moved out of the processing facility, and registered automatically in the Data Catalog.

4.2.4.2. Persistent Data Management

In this section we present two solutions, Hecuba and dataClay, that aim at facilitating the utilization of persistent data in applications. Both solutions implement a common API that allows programmers to manipulate all the data as regular Python objects, regardless if they are persistent (stored in disk) or volatile (stored in memory). Both solutions can be integrated with PyCOMPSs to enhance data locality and to optimize the mechanism of passing parameters to tasks. The target

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

applications of Hecuba are applications that use big amounts of data and can benefit from the parallel and asynchronous access that offer highly distributed key-value data stores. The target applications of dataClay are object-oriented applications that aim to avoid data movements by executing the object methods locally to the data.

4.2.4.2.1. Hecuba

Hecuba is a set of tools that provide programmers with transparent and efficient access to key-value databases. The current implementation can interact with Apache Cassandra [9], which is a wide-column store that follows the key-value data model. Hecuba code is organized into two different layers.

The first layer implements a Python API that allows programmers to access the persistent data as regular in-memory Python objects. This layer implements, transparently to the programmer, the data model in the database and keeps the mapping between both data structures: the in-memory objects defined by the programmer and persistent data. Hecuba also supports the definition of nested objects. Moreover, Hecuba is fully compatible with the NumPy library [10]. Hecuba implements the management of NumPy ndarrays storing them in a distributed fashion across the database nodes, to benefit from the database architecture and to increase the potential parallelism degree in accessing this data type.

The second layer is implemented in C++ and contains the code to interact with the database backend. This code implements some optimizations in the access to the database as data prefetching, data caching and enhancing data locality. This layer defines its own interface which is used by the Python layer and which can be used by any non-python application to benefit from the optimizations in the data access.

Hecuba is integrated with PyCOMPSs to enhance data locality and to avoid data serialization when tasks access persistent objects.

To facilitate the utilization of key-value data stores on an HPC environment, Hecuba comes with a set of scripts to automate the configuration and deployment of the database, considering the particularities of queue-based systems. Also, these scripts facilitate the creation of snapshots of the database and recover them in the following job allocations. The snapshotting mechanism provides a fast and easy way of persisting data between executions without requiring a global and shared database service in the HPC installations.

4.2.4.2.2. dataClay

dataClay [11] is a distributed data store that manages data in the form of objects, with their properties and relationships, enabling the representation of complex data structures (matrixes, lists, graphs, …). These data structures are directly stored in a persistent device (Non-volatile memory, SSD …) without the need of any transformations. The physical location and format of persistent data is transparent to the application developer.

The objects stored in dataClay also include the methods that enable their manipulation (retrieve or update the data they contain, or perform arbitrary computation on them). In this way, data locality is exploited as data does not need to be transferred to the application to be processed, but only the results of the methods defined.

The architecture of dataClay consists of a Logic Module, which is the main authority for metadata, such as the structure and methods of each object type, or the locations of objects in the different Backends. Each backend stores a set of objects and is also in charge of executing the methods

associated with them. Objects are instantiated in memory in the backends, ready to be used and thus avoiding disk accesses during execution in order to improve performance.

dataClay is integrated with the PyCOMPSs runtime, providing the appropriate interface to allow PyCOMPSs to get the locations of objects and create replicas (for read-only tasks) or versions (for read-write tasks), so that it can schedule the execution of tasks in the most appropriate location.

4.2.5. HPDA/ML Frameworks

One of the main reusable components of complex workflows are the HPDA and ML frameworks. They provide toolkits to easily program and efficiently execute DA algorithms or ML training and inferences. Therefore, every workflow running some of these codes will require to deploy and invoke these frameworks. Next paragraphs provide the description of the frameworks we plan to use during the project.

4.2.5.1. Ophidia

The Ophidia HPDA framework represents an open source solution, developed by CMCC, targeting the challenges related to management and analysis of scientific multi-dimensional data by joining HPC paradigms and Big Data approaches [12][13]. The framework provides in-memory, parallel, server-side data analysis and I/O and an internal storage model, based on the datacube abstraction inherited from the Online Analytical Processing (OLAP) systems, and a hierarchical organization to partition and distribute large amounts of multi-dimensional scientific data.

The framework is primarily used in the climate science domain, although it has also been successfully exploited in other domains (e.g., astronomy, seismology, and smart cities) thanks to its flexible architectural design and storage model.

Ophidia aims to provide a full software stack for data analysis at scale. The Ophidia Server represents the framework front-end component exposing multiple interfaces, such as SOAP or OGC-WPS. The server manages the interactions with the client-side and supports multiple execution modes ranging from interactive analysis to batch processing and workflows of analytics operators. Interactions with the server can be triggered through the Ophidia Terminal, a Command Line Interface, as well as the PyOphidia module, the Ophidia Python bindings.

In terms of data management and analysis features, the framework supports around 50 operators for both sequential metadata management and parallel datacube processing, including for example data subsetting, aggregation, comparison and import/export for domain specific data formats (e.g., NetCDF).

This second group of operators can be applied in parallel, exploiting a hybrid MPI+X approach, on the datacube fragments (i.e., chunks of data) distributed over the in-memory Ophidia storage layer. Each fragment is organized as a collection of multi-dimensional binary arrays. A wide set of low-level libraries implemented as User Defined Functions (UDF) and called Ophidia Primitives are provided to support the management and parallel processing of n-dimensional arrays.

4.2.5.2. Parallel Social Data Analytics (ParSoDA)

Parallel Social Data Analytics (ParSoDA) is a framework useful for parallel processing and analysis of social on top of a given parallel runtime environment [14]. Currently, ParSoDA is implemented on top of three environments: Hadoop, Spark, and PyCOMPSs. This means ParSoDA is currently available as a library for Java and Python.

A ParSoDA application is structured as a workflow that can be composed of the following stages:

- *Data Acquisition*: data are collected from one or more social media sources or data sets and converted into a common format, which is easily readable by the next stages;
- *Data Filtering*: data items are filtered by a chain of Boolean predicates which check some conditions that make the data eligible for next processing stages;
- *Data Mapping*: all the filtered data are modified according to a cascade of functions, in such a way more refined data are obtained from the filtered ones;
- *Data Partitioning*: data are partitioned using a group key; all items with the same group key are sorted according to a sort key and stored into shards; this stage is useful for optimizing data locality on the underlying runtime environment, because all the elements of a shard are sent to a node, which will compute the remaining stages on them;
- *Data Reduction*: all the elements with the same group key are reduced or aggregated into a single final item or data structure that is associated to the same group key;
- *Data Analysis*: the output data from the reduction stage are analyzed in order to extract or mine the target patterns;
- *Data Visualization*: finally, the results of the analysis are visualized in different graphical formats.

ParSoDA simplifies the development of data analysis applications by providing their deployment on the different nodes of a distributed computing system. The framework also exposes some built-in functions for all the different stages. For example, ParSoDA provides some predefined crawling function useful for collecting data from different sources during the Data Acquisition stage. Moreover, the set of functions can be enriched and a developer can create its own functions for a specific application.

4.2.5.3. Data Mining Cloud Framework (DMCF)

The Data Mining Cloud Framework (DMCF) [15] is a service-oriented distributed software framework that aims to effectively execute complex workflows for data analysis applications on cloud systems. In particular, DMCF has been designed to exploit data-driven parallelism and it uses in-memory distributed storage of data in order to maximize data locality and reduce read/write latency to secondary storage. Moreover, DMCF allows for the creation of workflow applications directly through a programming interface that defines two alternative programming languages for workflow programming: VL4Cloud that is a visual language used for graphic development of applications, and JS4Cloud, a textual language based on JavaScript. Thanks to the adoption of these languages, DMCF simplifies the development of applications, requiring a low level of programming skills.

The DMCF runtime enables the parallel execution of service-oriented data analysis workflows on multiple Cloud machines, so as to improve performance and ensure scalability of applications. To this end, the runtime implements data-driven task parallelism that automatically spawns ready-to-run workflow tasks to the Cloud resources, considering dependencies among tasks and current availability of data to be processed. Parallelism is effectively supported by the data and tool array formalisms where the array cardinality automatically determines the parallelism degree at runtime.

DMCF is implemented on the Microsoft Azure cloud platform, and it also includes a data-aware scheduler that uses Hercules [16] in order to exploit in-memory storage of temporary data. Hercules is a software component which aims to reduce access latency to/from secondary storage, by implementing a RAM disk on each node of the system. The data-aware scheduler of DMCF is

based on two task queues: a global waiting queue, in which every task is stored into before it is polled by a compute node of the system for execution; a local queue for each compute node where locally activated tasks are stored into. The global queue is useful to protect task metadata from node failures, because it is distributed among the cloud platform, so if a virtual node fails the global queue can be accessed anyway. Locally activated tasks are these polled by a node and prepared to be executed on it. Each node has at least a thread which polls tasks from the local queue, executes them and updates their metadata. In particular, the scheduler aims to execute the task whose parents are terminated and for which the current node is the best one with respect to data allocation.

The DMCF allows a user for storing input and output files of the workflows into the native storage of Azure, where the user can access them, while speeding up the execution performance by using the Hercules in-memory storage for temporary files that are created and re-used on the compute nodes during the execution

4.2.5.4. Dislib

The Distributed Computing Library (dislib) [17] is a distributed machine learning library written in Python that enables large-scale data analytics on HPC infrastructures. Inspired by scikit-learn, dislib provides an estimator-based interface that improves productivity by making models easy to use and interchangeable. This interface also makes programming with dislib very easy to scientists already familiar with scikit-learn. Furthermore, dislib provides a distributed data structure that can be operated as a regular Python object, hiding the underlying distribution details to the final user. The combination of this data structure and the estimator-based interface makes dislib a distributed version of scikit-learn, where communications, data transfers, and parallelism are automatically handled behind the scene.

Dislib is built on top of PyCOMPSs, offering good scalability and performance in distributed infrastructures, including clusters, Clouds, and supercomputers. Since PyCOMPSs automatically manages the infrastructure and the distribution of the computation, dislib applications can run in multiple platforms without changing the source code, and without having to worry about platform specific details, such as IP addresses and storage devices. In addition to this, dislib applications can also include custom PyCOMPSs tasks to perform data pre-processing or post-processing in parallel, or to combine computational workloads with data analytics.

In addition to data management methods, dislib provides algorithms for clustering, classification, decomposition, and model-selection for parameter tuning among others. Dislib abstracts developers from all the parallelization details, and allows them to build large-scale machine learning workflows in a completely sequential and effortless manner.

The main concepts around dislib are:

- **Distributed arrays**: The built-in 2-dimensional arrays that can be operated in parallel, and that are used as the main input for the different algorithms. Distributed arrays store samples and labels in a distributed way that works as a regular Python object from the user point of view.
- **Data handling:** Methods for loading data from files in common formats, such as CSV and LibSVM.
- **Unified interface**: scikit-learn inspired interface for the different algorithms (i.e., fit, predict, etc.). This makes dislib's interface easy to learn for the users already familiar with scikit-learn, and allows a smooth transition of existing codes from scikit-learn to dislib.

Dislib can be easily integrated into any existing PyCOMPSs application, and can run in any computing platform supported by the COMPSs runtime.

4.2.5.5. Helmholtz Analytics Toolkit (HeAT)

HeAT [18] has been introduced in order to exploit distributed memory architectures as well as GPU driven computation and make it available through an easy to use NumPy-like interface. HeAT utilizes PyTorch as a node-local eager execution engine and distributes the workload on arbitrarily large high-performance computing systems via MPI. It provides both low-level array computations, as well as assorted higher-level algorithms. With HeAT, it is possible for a NumPy user to take full advantage of their available resources, significantly lowering the barrier to distributed data analysis.

HeAT hinges on the concept of the *DNDarray*, which is an inherently distributed n-dimensional array akin to NumPy's ndarray. The *DNDarray* object is a virtual overlay of the disjoint PyTorch tensors, which store the numerical data on each MPI process. A *DNDarray*'s data may be redundantly allocated on each node, or one-dimensionally decomposed into evenly-sized chunks with a maximum size difference of one element along the decomposition axis. This data distribution strategy aims to balance the workload between all processes. During computation, API calls may redistribute data items. However, completed operations automatically restore the uniform data distribution.

HeAT is used in various application domains, such as Earth system modeling, structural biology, neuroscience, and aeronautics and aerospace. When compared to similar frameworks, HeAT achieves speedups of up to two orders of magnitude. It has proven to be significantly faster than Horovod for training on the ImageNet classification and CitiScapes semantic segmentation tasks [19].

4.2.5.6. European Distributed Deep Learning library (EDDL)

The European Distributed Deep Learning library (EDDL) is an open source library for Distributed Deep Learning and Tensor Operations in C++ for CPU, GPU and FPGA. The main goal of EDDL is to serve as a European Library for training and inference operations over Deep Learning neural networks. The main properties of the library are listed here:

- EDDL works around the concept of tensors. Tensors can be instantiated and operated with tensor operations. Current supported operations include tensor manipulations, image operations, indexing & sorting, linear algebra, logic functions and mathematical functions.
- Tensors can be instantiated and operated transparently on different target hardware devices, such as CPU, GPU, and FPGA. Tensor operations are abstracted away from the target hardware and, as such, allows the end user to transparently use different hardware devices for his/her operations with no prior knowledge.
- With EDDL a neural network with standard layers used in other software packages such as TensorFlow/Keras can be used. Currently, layer types supported are: Core layers, auxiliary layers, activation layers, data augmentation and transformation, convolutions, pooling, normalization, reduction, and recurrent layers (among others). The library is Open Source and allows adding new kinds of layers, if needed.
- The EDDL library includes import/export functionality mostly working with ONNX formats, therefore, is able to natively run a generated ONNX model with no additional effort from other frameworks.

- Python version of EDDL is available as a parallel Open Source project.[10]
- EDDL is being adapted with COMPSs. Distributed training can be defined and run with EDDL in companion with COMPSs [20].

4.2.5.7. Framework Differentiation

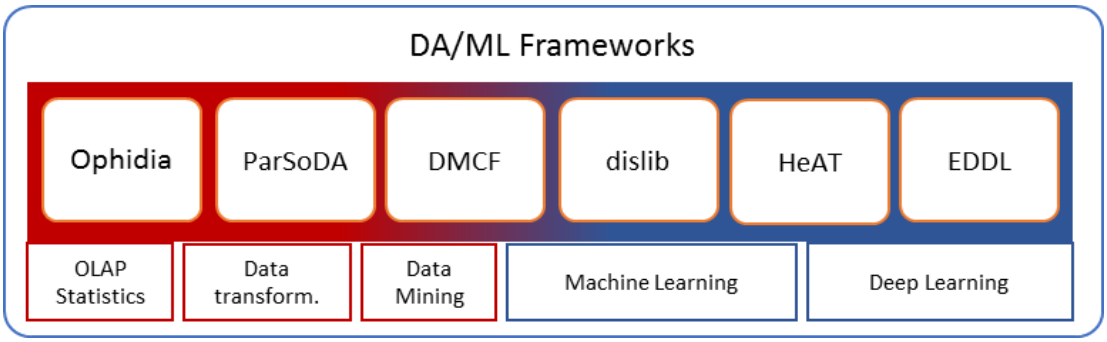

Figure 6. HPDA/ML framework differentiation by functionality.

Figure 6 summarizes the features offered by the different DA and ML frameworks. Ophidia and ParSoDA focus on providing data analysis features, in particular Ophidia provides OLAP analytics operations and *ParSoDA* provides a set of data transformations (partitioning, mapping, merging, etc) and visualization functions. On the other hand, dislib, HeAT and EDDL are providing different algorithms for machine learning model training and inference. In particular, dislib and HeAT are targeting machine learning methods and HeAT and EDDL also target deep learning with different types of neural networks. In the middle, we have DMCF which provides some data mining and machine learning algorithms.

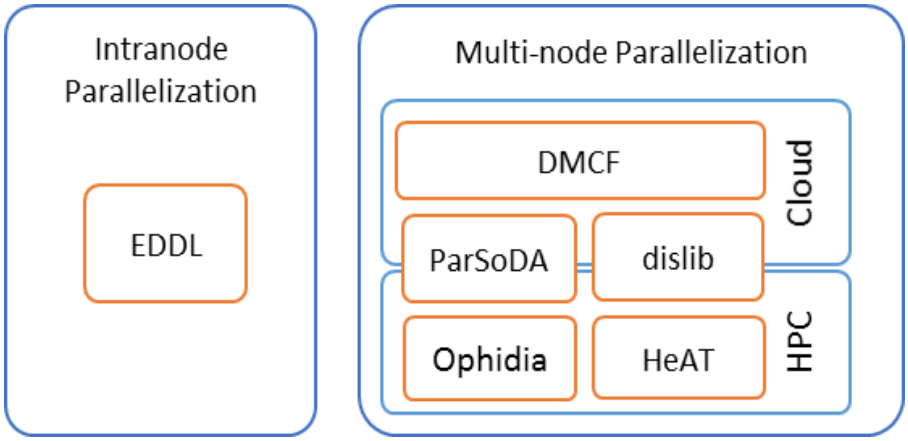

Figure 7. HPDA/ML Framework differentiation by target parallelism and platform.

Apart from the functional classification, we can also differentiate the frameworks by the target parallelization platform as depicted in Figure 7. For instance, EDDL is targeting the parallelism inside a node such as multi-core CPUs and accelerators (GPUs, FPGAs, ...), the rest of frameworks are focused on multi-node parallelization. In this part, we can differentiate three types of frameworks according to the target environments. On one hand, DMCF relies on cloud services for executing the algorithms, so it is currently targeting just cloud environments. On the other hand, Ophidia and HeAT rely on MPI for multi-node parallelism so their algorithms can benefit from high-performance networks in HPC clusters. Then, ParSoDA and dislib rely on infrastructure agnostic technologies so their algorithms can work in both environments. Note that this classification is

[10] https://github.com/deephealthproject/pyeddl

done based on the main targets of the frameworks. However, due to the background libraries or the combinations with other tools they can support more complex environments and workflows. For instance, HeAT relies on PyTorch which allows to operate with GPUs; EDDL can be combined with dislib and PyCOMPSs for supporting distributed and federated training; and Ophidia, ParSoDa, dislib and EDDL operations can be decomposed in PyCOMPSs tasks creating integrated DA/ML workflows.

4.3. Usage and Component Interactions

One of the main current barriers for the adoption of HPC is the complexity of deploying and executing the workflows in federated HPC environments. Usually, users are required to perform software installations in complex infrastructures which are beyond their technical skills. Therefore, having the workflows ready for execution in a supercomputer could take large amounts of time and human resources. If it needs to be replicated to several clusters, the required time and resources will increase. To widen the access to HPC to newcomers, and, in general, to simplify the deployment and execution of complex workflows in HPC systems, eFlows4HPC proposes a mechanism to offer HPC Workflows as a Service (HPCWaaS) following a similar concept as the Function as a Service (FaaS) in the Cloud, but applying it for workflows in federated HPC environments. The goal is to hide all the HPC deployment and execution complexity from the final end users of the workflows in such a way that executing workflows only requires to perform a simple call to a REST API. It will also provide a mechanism to enable the sharing, reuse, and reproducibility of complex workflows

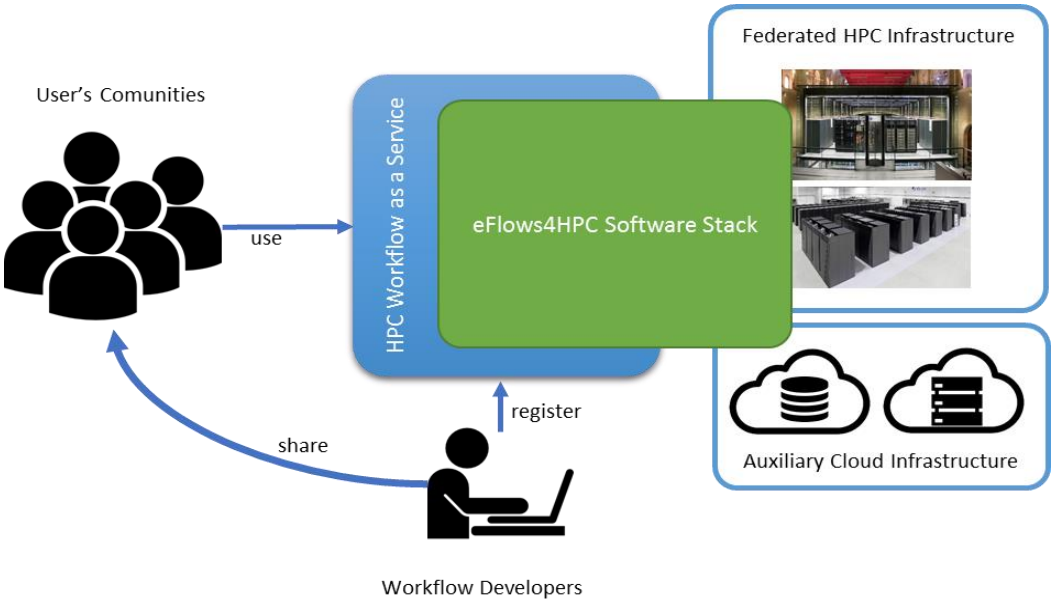

Figure 8.HPC Workflow as a Service usage cases overview.

Figure 8 shows an overview of how the proposed model works. The HPC Workflow as a Service is built on top of the eFlows4HPC software stack in order to provide the required functionality to develop, deploy and execute the complex workflows. Interactions of the users with HPCWaaS is done in two phases: one for workflow developers and another for workflow user communities. At development time, workflow developers are in charge of building the workflow using the first two layers of the eFlows4HPC stack. Once the workflow creation is finished, the workflow is registered in the HPCWaaS to make it available to the final users. After a successful registration, the workflow

developer receives a service endpoint from the HPCWaaS that other users can invoke to use the developed workflow. It will be automatically deployed and executed in the computing infrastructure using the rest of eFlows4HPC stack functionalities. Following paragraphs provide more details about how the different eFlows4HPC components interact to provide the required functionality in the different usage cases.

4.3.1. Workflow Development

One key part of the challenge mentioned in the introduction, is the implementation of complex workflows that combine HPC, HPDA, and ML frameworks. eFlows4HPC proposes two mechanisms to achieve this challenge as depicted in Figure 9. On the one hand, the software stack provides a set of registries, catalogs and repositories, providing workflow developers with the means to store the core components (HPC, DA, and ML frameworks) and the required data and ML models in such a way that they can be easily reused in different workflows and infrastructures. On the other hand, we propose the definition of a workflow description which enables the combination of the different workflow components. From this workflow description, the third layer of the eFlows4HPC software stack can be used to automatically deploy and execute the workflow in the Computing Infrastructures.

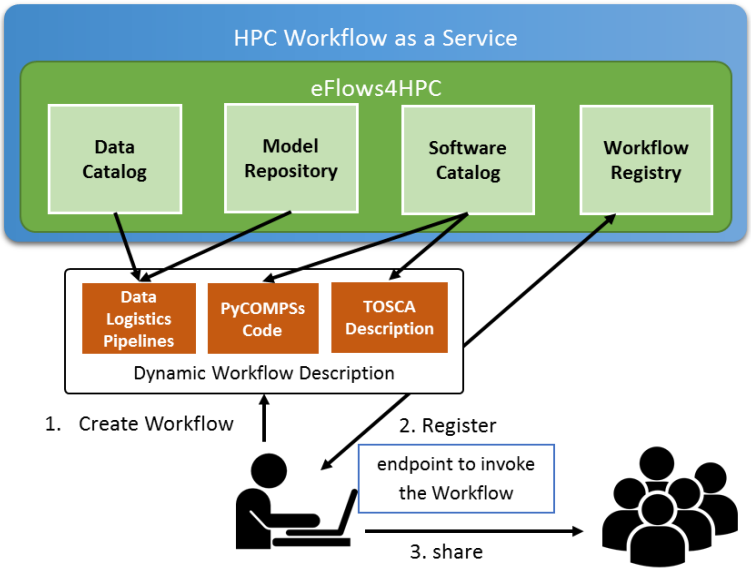

Figure 9. Workflow development usage case.

The proposed workflow description is composed of a combination of an Extended TOSCA syntax, the PyCOMPSs programming model, and a set of Data Logistics pipelines. In the first part, TOSCA (an orchestration standard) allows developers to specify which software and services are required; and how each component should be deployed, configured (linked to each other), started, stopped and deleted. In the second part, the PyCOMPSs programming model will provide the logic of the different components of the overall workflow. PyCOMPSs is a task-based programming model that enables the development of workflows that can be executed in parallel on distributed computing platforms. It is based on programming sequential Python scripts, offering the programmer the illusion of a single shared memory and storage space. While the PyCOMPSs task-orchestration code needs to be written in Python, it supports different types of tasks, such as Python methods, external binaries, multi-threaded (internally parallelized with alternative programming models such as OpenMP or pthreads), or multi-node (MPI applications). Thanks to the use of Python as a programming language, PyCOMPSs naturally integrates well with data analytics and machine learning libraries, most of them offering a Python interface. Finally, in the last part of the workflow

description, the data logistics pipelines allow developers to describe how the workflow data are acquired, moved and stored during the workflow execution in order to ensure the data is available in the computing infrastructure when required.

As mentioned before, the workflow description is registered and stored in a workflow registry by means of the HPCWaaS interface. The result of this registration will produce a service endpoint which can be later used to invoke the execution of the workflow.

4.3.2. Workflow Deployment and Execution

When users want to execute the registered workflows, they only need to invoke the endpoint provided at the end of the workflow development phase. As result of this invocation, the last layers of the eFlows4HPC software stack are used to provide an automatic and holistic workflow deployment and execution in federated computing HPC infrastructures. This functionality is provided by the cooperation of several components at different levels. At the highest level (Figure 10), the Ystia Orchestrator (Yorc) is in charge of orchestrating the deployment of the main workflow components in the computing infrastructures and managing their lifecycle (configuring, starting services) as described in the TOSCA part in the workflow description. In parallel to the component deployment, the data logistics part of the description is used by the Data Logistics Service to set up the required data movements, such as the data stage-in and stage-out, or periodical transfers to synchronize data produced outside the HPC systems.

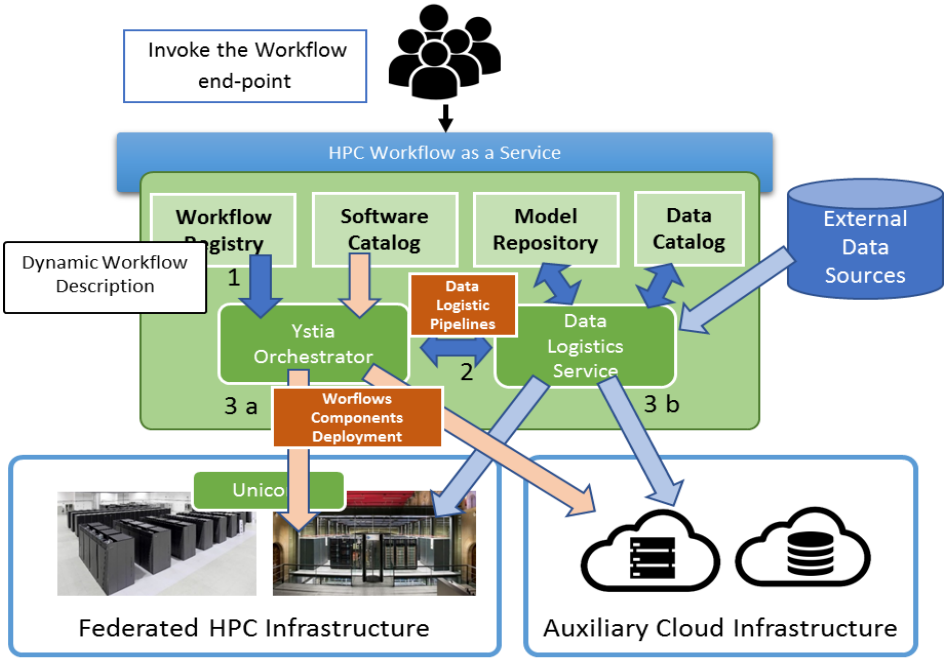

Figure 10. Workflow deployment

Once the workflow components and data are deployed, Yorc submits the execution of the main workflow processes in the computing infrastructure (Figure 11). This step can be supported by UNICORE, which is in charge of managing the federation of HPC compute and data resources in order to make them available to users in a secure way. At the lowest level, the COMPSs runtime will coordinate the invocations of the workflow components implemented with the PyCOMPSs task-based programming model. As mentioned before, COMPSs supports several task types which

can include HPC simulations, DA transformations, etc. The runtime dynamically generates a task-dependency graph by analyzing the existing data dependencies between the invocations of tasks defined in the Python code. The task graph encodes the existing parallelism of the workflow, which can be used to schedule the executions in the resources already deployed by Yorc. Based on this scheduling the COMPSs runtime can interact with the different HPC, DA and ML runtimes in order to coordinate the resources usage performed by the different invocations to avoid overlaps and getting the maximum performance to the available resources. Apart from the dynamic task graph generation, the COMPSs runtime is also able to react to task failures and exceptions in order to adapt the workflow behavior accordingly. These functionalities, together with similar features provided by Yorc at a higher level, offer the possibility of supporting workflows with a very dynamic behavior, that can change their configuration at execution time upon the occurrence of given events, such as failures or exceptions.

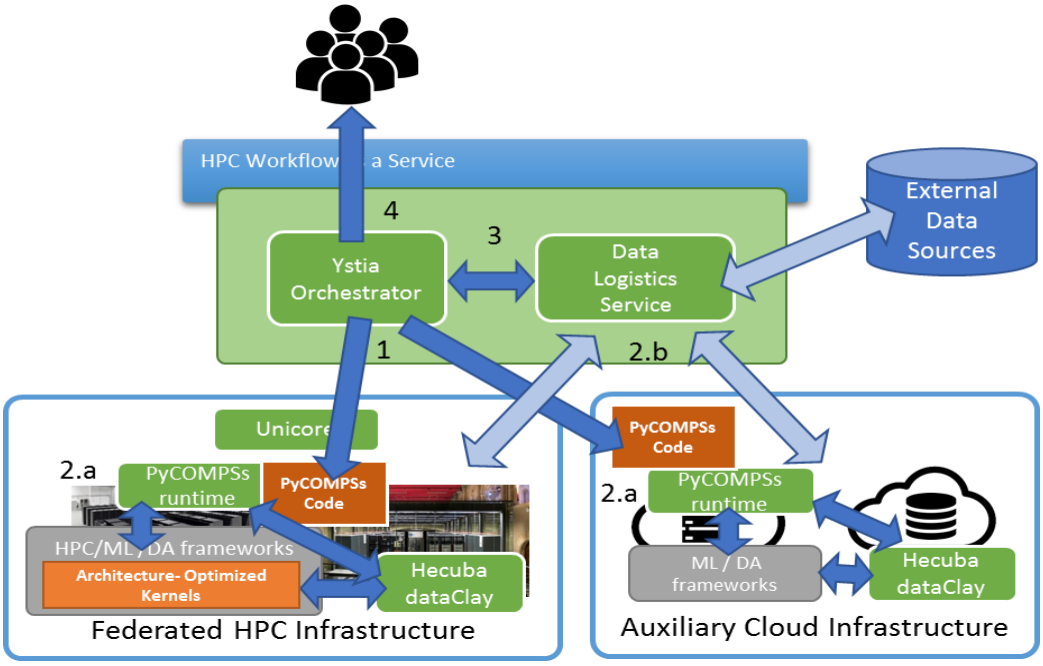

Figure 11. Workflow execution

Finally, regarding the integration of the data management and computation, the eFlows4HPC stack also provides two solutions for persistent storage: Hecuba (based on key-value databases) and dataClay (object-oriented distributed storage). These solutions can be used in PyCOMPSs applications to store application objects as persisted objects, either in disk or in new memory devices, such as NVRAM or SSDs, enabling to keep data after the execution of the application. This changes the paradigm of persistent storage in HPC, dominated by the file system, to other more flexible approaches. By using persisted objects, application patterns such as producer-consumer, in-situ visualization or analytics, can be easily implemented.

4.4. Requirement Fulfilment by Architecture Components

The following table provides the relationship between the components of the eFlows4HPC architecture and the requirements extracted from the different sources. For each requirement we have identified which components are involved in providing the required functionality.

Table 7. Requirements-to-component matrix

Requirement		Components Involved												
		HPCWaaS	DC	SC	WR	MR	HPDA -Fw.	ML -Fw.	PyCOMPSs	YORC	UNICORE	DLS	Hecuba	dataClay
P1-1	Distributed SVD							x	x					
P1-2	Storing of hyper-reduced model					x								
P1-3	ANN model					x		x						
P1-4	Clustering model					x	x	x						
P1-5	Persistent storage												x	x
P1-6	Restart								x	x	x			
P1-8	Workflow orchestration							x						
P1-9	ML inference	x								x				
P2-1/P3-4	Hyper Reduced Model Deployment	x				x			x	x	x	x		
P2-2/P3-6	Portability and reusability	x		x	x					x				
P1-7/ P2-3/P3-8	Workflow Orchestration / Integrated workflow management	x							x	x	x			
P2-4/P3-9	Integration with permanent storage		x									x	x	x
P2-5	Workflow adaptability								x	x				
P2-6	Access to intermediate in-memory results								x				x	x
P2-7	AI integration for ensemble member pruning							x	x				x	x
P2-8	ML/DL capabilities					x		x						
P2-9	DA capabilities						x							
P2-10	High Performance Computing support								x	x	x			
P2-11	Multi-member analysis						x	x	x	x			x	x
P3-1	Urgent computing access									x	x			
P3-2	Data accessibility											x		
P3-3	Data replication		x									x	x	x
P3-5	Infrastructure interoperability								x	x	x	x		
P3-7	Streaming Data Source								x			x	x	x
P3-10	Inference of online/offline ML models					x		x						
P3-11	DA integration						x		x				x	x
P3-12	Workflow malleability								x	x	x			
CMP-1	Access to HPC specific devices			x			x	x	x	x				
CMP-2	Support optimized kernels			x			x	x	x	x				

CMP-3	Service deployments	x								x				
CMP-4	Service Invocation								x					
CMP-5	Multi-node execution support								x	x	x			
CMP-6	Multicore execution support								x					
HPC-1	HPC Cluster access support									x	x			
HPC-2	HPC Data Transfers support										x	x		
HPC-3	Singularity Container support						x	x	x	x			x	x
HPC-4	Infrastructure Service deployment									x	x	x		
HPC-5	Queue System								x	x	x			

5. Metrics

To evaluate the improvements introduced by the eFlows4HPC methodology and architecture, we have defined a set of end-user metrics which have been divided in two parts: common workflow metrics selected from the technical areas targeted by the project, and pillar specific metrics, which are focused on measuring the improvements related to the scientific process of each pillar. The following paragraphs present the common workflow metrics while pillar specific metrics are presented in deliverables D4.1, D5.1 and D6.1.

The eFlows4HPC project will improve different aspects of the lifecycle of the complex workflows (development, deployment and execution). To measure these improvements, we have defined a set of relevant metrics from different areas related to the technical project objectives. Next paragraphs introduce the considered areas and afterwards Table 8 presents the selected metrics.

- **Development & Maintenance**: Metrics of this area are intended to measure how difficult the development of a software and its maintenance is. They are normally measured by tools to inspect source codes (such as Sonar) providing the lines of code, cyclomatic complexity or the number of duplicated code blocks. Lines of code provides a general metric of the effort for programming and maintaining a code: an increment/reduction of lines of code, implies an increment/reduction of effort to program, understand and modify. The cyclomatic complexity is a metric to measure how complex a piece of code is, so it is mainly affecting the maintenance. Finally, the duplicated blocks metric is also affecting the maintainability because a change in a duplicated code must be done in several places. Therefore, we consider lines of code metric as the most relevant for development and maintenance. Due to the project being mainly focused on the workflow development and the integration of its main components, this metric will be only applied to workflow codes but not in software used inside the workflow components (simulators, libraries etc.).
- **Accessibility & Deployment:** Metrics of this area are aimed at measuring the deployment process of a workflow in terms of how portable or reusable a workflow is, and how long a developer or an automated process will take to deploy the workflow in a selected infrastructure.
- **Performance**: Metrics of this area focus on the quality of a workflow execution. These quality metrics are normally calculated by measuring the execution time of a workflow in different conditions such as input data sets or infrastructure configurations to know if the execution is efficient and scalable. These metrics can be applied to the whole workflow

execution, but also to kernels whose optimization will considerably affect the total workflow performance.

- **Data Management**: Another important area when executing distributed computing applications is data management. Metrics of this area are focused on evaluating how efficient a workflow execution is in terms of data management. These metrics can provide information about the number of data operations, the size of these operations and duration of these operation which is crucial to evaluate if the execution is using a proper data locality
- **Reliability:** Failures can happen when executing complex workflows in large computing infrastructures. There are some failure recovery techniques to mitigate these failures (such as retries, data replication or execution redundancy) which can be implemented by the user or transparently supported by the workflow manager. Metrics on the reliability area are proposed to evaluate how tolerant a workflow is to unexpected failures.
- **Energy & Cost**: Despite most scientific HPC users are not paying for its usage, the maintenance of HPC infrastructures has a cost, not only in the initial investment or the system administration but also in the energy consumed during the operation. Metrics defined in this area measure the energy consumed and the cost associated to a workflow execution. Some infrastructure monitoring systems are providing accounting information about the resource usage and energy consumption of the executed jobs. These metrics for the workflows can be calculated as the aggregated metric for all the jobs executed by a workflow.

Table 8. Common workflow metrics

Acronym	Name	Description	Area
LoC	Lines of Code	Number of Lines of code in the workflow implementation.	Development & Maintenance
DoP	Degree of Portability	Percentage of workflow components that can be reused in other infrastructures and workflows.	Accessibility & Deployment
DT	Deployment Time	Time elapsed to deploy the workflow.	Accessibility & Deployment
ET	Execution Time	Time elapsed to execute a workflow.	Performance
SU	Speed-up	Execution time improvement when running with larger resources. Calculated as: SU(N) = ET(base)/ET(N) where ET(base) is the baseline and ET(N) is the execution with N times larger resources.	Performance
Eff	Efficiency	Execution time degradation when running larger problems. Calculated as: $Eff(N) = ET_{base}(base) \ / ET_N(N)$ where $ET_{base}(base)$ is the execution of the baseline problem and infrastructure and $ET_N(N)$ is the execution time of N times larger problem and infrastructure.	Performance
TD	Transferred Data	Amount of data transferred (in bytes) by the workflow between different compute nodes of the computing infrastructure.	Data Management
DM	Data Movements	Number of transfer operations between different compute nodes of the computing infrastructure.	Data Management
IOT	I/O Time	Percentage of Execution time performing I/O operations.	Data Management
FTC	Fault-tolerant components	Percentage of workflow components that are fault-tolerant.	Reliability
CH	Core/Hour	Number hours of a CPU Core consumed by the workflow execution.	Energy & Cost
EC	Energy Consumption	Energy consumed (Wh or Joules) associated with a workflow execution.	Energy & Cost

The metrics defined above and the specific pillar metrics will be using during the evaluation process of the project. This evaluation will be performed at the end the two implementation phases defined in the project description. The pillars' teams supported by the technical teams at WP1 and WP3 will measure the defined metrics before and after applying an improvement introduced by the eFlows4HPC methodology or functionality provided by a component of the software stack. The difference between these metrics measurement will quantify the improvements done or possible side effects.

6. Conclusions

One of the main steps of a software infrastructure is the definition of its requirements and architecture. The eFlows4HPC project has conducted during these first months a key activity by collecting requirements from the Pillars' workflow applications, from the components that will compose its software stack and constraints that should be considered from the HPC centers. All this process has been performed involving key players in multiple work packages (WP1, WP4, WP5 and WP6) and external sources (HPC centers).

This document has presented the first version of the requirements and architecture, and the definition of a set of metrics for the evaluation of the workflows. A revision of this document will be performed after phase 1, in months M18 to M20. This second version will take into account the feedback received from the Pillars' on the first release of the eFlows4HPC software stack.

7. Acronyms and Abbreviations

- AAI - Authentication Authorization Infrastructure
- AI - Artificial Intelligence
- ANN – Artificial Neural Network
- API - Application Programming Interface
- CaaS - Container as a Service
- CLI - Command Line Interface
- CPU - Central Processing Unit
- D - Deliverable
- DA - Data Analytics
- DAG - Directed Acyclic Graph
- DC - Data Catalog
- DL - Deep Learning
- DLS - Data Logistics Service
- DMCF - Data Mining Cloud Framework
- DNN - Dynamic Neural Network
- EDDL - European Distributed Deep Learning library
- ETL - Extract, Transform, Load
- FaaS - Function as a Service
- FAIR - Findable Accessible Interoperable Reusable
- FPGA - Field Programmable Gate Array
- FTP - File Transfer Protocol
- GPU - Graphics Processing Unit
- GUI - Graphical User Interface
- HeAT - Helmholtz Analytics Toolkit
- HPC - High Performance Computing
- HPCWaaS - HPC Workflow as a Service
- HPDA - High-performance Data Analytics
- IaaS - Infrastructure as a Service
- ID - Identifier
- JSON - JavaScript Object Notation
- KPI - Key Performance Indicator
- M - Month
- ML - Machine Learning
- MPI - Message Passing Interface
- MR - Model Repository
- NN - Neural Network
- NVRAM - Non-Volatile Random Access Memory
- OLAP - On-Line Analytical Processing
- ONNX - Open Neural Network Exchange
- ParSoDA - Parallel Social Data Analytics
- POSIX - Portable Operating System Interface
- PRACE - Partnership for Advanced Computing in Europe

- REST - Representational State Transfer
- SC - Software Catalog
- SCP - Secure Copy
- SSD - Solid State Disk
- SSH - Secure Shell
- SVD - Singular Vector Decomposition
- TOSCA - Topology and Orchestration Specification for Cloud Applications
- UDF - User Defined Functions
- UI - User Interface
- VPN - Virtual Private Network
- WP - Work Package
- WR - Workflow Registry

8. References

[1] OASIS Standard. “Topology and orchestration specification for cloud applications version 1.0”. 2013. On-line: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

[2] Tejedor, E., et al. "PyCOMPSs: Parallel computational workflows in Python." The International Journal of High Performance Computing Applications 31.1 (2017): 66-82.

[3] Badia, Rosa M., et al. "Comp superscalar, an interoperable programming framework." SoftwareX 3 (2015): 32-36.

[4] Elshazly H, Lordan F, Ejarque J, Badia RM. “Performance Meets Programmabilty: Enabling Native Python MPI Tasks In PyCOMPSs”. In 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) (2020) (pp. 63-66). IEEE.

[5] Ramon-Cortes C, Lordan F, Ejarque J, Badia RM. “A programming model for Hybrid Workflows: Combining task-based workflows and dataflows all-in-one”. Future Generation Computer Systems. (2020); 113:281-97.

[6] Ejarque, J., Bertran, M., Álvarez, J., Conejero, J., Badia, R.M. “Managing Failures in Task-Based Parallel Workflows in Distributed Computing Environments”. In European Conference on Parallel Processing, (2020) Aug 24 (pp. 411-425). Springer

[7] Lordan, F., et al. "Servicess: An interoperable programming framework for the cloud." Journal of grid computing 12.1 (2014): 67-91.

[8] Rybicki, J. "Designing a Data Logistics and Model Deployment Service". , The Sixth International Conference on Big Data, Small Data, Linked Data and Open Data, 2020

[9] Lakshman, A., Malik, P. “Cassandra: a decentralized structured storage system”. ACM SIGOPS Operating Systems Review. Volume 44, Issue 2,1-92 pages. 2010

[10] Harris, C.R., Millman, K.J., van der Walt, S.J. et al.” Array programming with NumPy”. Nature 585, 357–362 (2020).

[11] Martí, J., Queralt, A., Gasull,D., Barceló, A., Costa,J.J., Cortes, T. “Dataclay: A distributed data store for effective inter-player data sharing”. Journal of Systems and Software 131: 129-145 (2017)

[12] S. Fiore, A. D’Anca, C. Palazzo, I. T. Foster, D. N. Williams and G. Aloisio, "Ophidia: Toward Big Data Analytics for eScience", Proc. Int. Conf. Comput. Sci., pp. 2376-2385, 2013, doi: 10.1016/j.procs.2013.05.409

[13] D. Elia, S. Fiore and G. Aloisio, "Towards HPC and Big Data Analytics Convergence: Design and Experimental Evaluation of a HPDA Framework for eScience at Scale," in IEEE Access, vol. 9, pp. 73307-73326, 2021, doi: 10.1109/ACCESS.2021.3079139

[14] L. Belcastro, F. Marozzo, D. Talia, P. Trunfio, "ParSoDA: High-Level Parallel Programming for Social Data Mining", *Social Network Analysis and Mining*, vol. 9, n. 1, 2019.

[15] F. Marozzo, D. Talia, P. Trunfio, "A Workflow Management System for Scalable Data Mining on Clouds". *IEEE Transactions On Services Computing*, vol. 11, n. 3, pp. 480-492, 2018.

[16] F. Marozzo, F. Rodrigo Duro, J. Garcia Blas, J. Carretero, D. Talia, P. Trunfio, “A Data-Aware Scheduling for Workflow Execution in Clouds”, Concurrency and Computation: Practice and Experience, vol. 29, n. 24, Wiley InterScience, 2017.

[17] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia, “dislib: Large Scale High Performance Machine Learning in Python,” in Proceedings of the 15th International Conference on eScience, 2019, pp. 96-105

[18] Götz, M.; Coquelin, D.; Debus, C.; Krajsek, K.; Comito, C.; Knechtges, P.; Hagemeier, B.; Tarnawa, M.; Hanselmann, S.; Siggel, M.; Basermann, A. & Streit, A., HeAT -- a Distributed

and GPU-accelerated Tensor Framework for Data Analytics, 2020, https://arxiv.org/abs/2007.13552

[19] Coquelin, D.; Debus, C.; Götz, M.; von der Lehr, F.; Kahn, J.; Siggel, M. & Streit, A., Accelerating Neural Network Training with Distributed Asynchronous and Selective Optimization (DASO), 2021, https://arxiv.org/abs/2104.05588

[20] Flich, J., et al. "Distributed Training on a Highly Heterogeneous HPC System." International Conference on Embedded Computer Systems. Springer, Cham, 2020.

Appendix A.

Workflows Requirements Template

Workflow overview

Identify and introduce the main workflow phases/building blocks

Workflow Requirements for eFlows4HPC Software Stack

Building blocks Requirements

Building block/phase 1:

Input/output data:

Input and output data of the building block

Computational Granularity:

coarse-grain(>secs)/ fine-grain (<1sec)

Specific software/hardware:

HPC kernels or other tools which are mandatory to run the building block

Programming Languages:

Languages of the APIs or software used in the building block

DA requirement:

Require to do whatever DA algorithm to perform …

ML requirements:

Require to support clustering …

Integration of DA, ML with HPC kernels:

The DA algorithm preprocesses the input of simultation X. The output of simulation X is the training data for the k-means clustering

Other required functionalities:

Building block/phase 2:

….

Workflow deployment /execution requirements

Deployment restrictions

Locality, Licenses, Data availability

Expected execution requirements

(Dynamicity, replication, ..).

E.g.

Before execution BB1 must be deployed and executing during the whole workflow execution.

The workflow starts executing BB1….

Data Requirements

Data flow:

Sources / Intermediate data /Outputs

Persistency requirements:

in –memory/disk

Data types/structure

collections of small items, big items, arrays…

Data creation-consumption pattern

1 to 1, 1 to N, stream…

E.g. (Maybe a graph)

The input of the workflow is … and it is an x-dimensional array of floats… it is normally stored in … must be copied to the computing location of BB 1

The output of BB1 which is a key, value sets must be used by BB2 and BB3. It is generated every iteration and can be consumed by BB2 once the data is available.

BB2 consumes BB1 data and generates ...

BB3 requires all data generated BB1 and BB2 to create a model which must be stored in a repository for re-usage.

Appendix B.

HPC System Administration Questionnaire

Background and Goal

The eFlows4HPC projects aims to build a European Stack for managing workflows for HPC. The idea is to provide a set of tools and services in order to facilitate the development, sharing, deployment and execution of complex workflow in HPC systems. We have created this questionnaire to gather what are common practices and main security constraints when accessing HPC sites. This information will be analyzed by the consortium member to detect the possible barrier for adopting the proposed eflows4HPC software stack and gathering requirements for the workflow deployment in supercomputers.

Available Infrastructure and access request

- **Supercomputers**: (Architecture + Accelerator if apply)
- **Access request procedure:**
  - PRACE access requests (yes/no)
  - Other

Access and security

- **Access to supercomputers and data transfers possibilities**
  - SSH/SCP: (Yes/No)
  - Unicore:
  - Other:
- **User identification**
  - SSH Keys: Yes/No
  - Other certificates:
- **Firewall policy:**
  - Available Income connections:
  - Available Outcome connections:
- **Restrictions on login nodes**
  - Is it possible to deploy services/deamon in login nodes to access queue system?
  - Limits interactive nodes (time, memory, num processes, ...)
- **Service nodes**
  - Do you have service nodes?
  - Is it possible to deploy new services on them?
  - Are they accessible from outside: Is there any proxy service to access services deployed in the cluster?
- **Connectivity between compute nodes:** (Enumerate restrictions if any)
- **Connectivity between login nodes to compute nodes**:

Queue system and Shared disk

- **Available Queue system** (Slurm, LSF, PBS, Other)
- **Remote execution command to spawn processes/command between nodes** (srun, blaunch, ssh,
- **Shared disk systems** (GPFS, Lustre,…)
- **Folder/nodes where Shared disk is mounted** (Example: /home in all nodes, /scratch only compute nodes)

Software Management

- **Environment management:**
  - Modules: (Yes/No)
  - Other
- **Building tools without root privileges:**
  - Easy-build: (Yes/No)
  - Spack:
  - Conda:
  - Other:
- **Containers support**
  - Docker:
  - Singularity
  - Podman:
  - Shifter:
  - Pcocc:
  - Other:

Data Infrastructure (hosting and management)

- **Data transfer nodes/services/interfaces** (Please enumerate)
- **Available data storage levels** (NVMe, SSD, HDD, Shared Disk, Archive…):
  - Persistence between execution (in storage levels)
  - Capacities (BW, size, etc.)