eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

# D1.3 Revision of Requirements and Architecture Design

**Version 1.0**



## Documentation Information

| | |
|---|---|
| **Contract Number** | 9555558 |
| **Project Website** | www.eFlows4HPC.eu |
| **Contractual Deadline** | 31.08.2022 |
| **Dissemination Level** | PU |
| **Nature** | R |
| **Author** | Jorge Ejarque (BSC) |
| **Contributors** | Rosa M. Badia(BSC), Yolanda Becerra(BSC), Loïc Albertin (Atos), Anna Queralt (BSC), Domenico Talia (DtoK Lab), Jedrzej Rybicki (FZJ), Alessandro D'Anca (CMCC), José Flich (UPV) |
| **Reviewer** | Fabrizio Marozzo (DtoK) |
| **Keywords** | Requirements, Architecture, 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 |
|---|---|
| **0.1** | Proposed Table of Contents |
| **0.2** | Update of Pillars Requirements |
| **0.3** | Update of Architecture Components |
| **0.4** | Update of Component interactions |
| **1.0** | Update with internal reviewer comments |
| | |
| | |

# Table of Contents

# 1. Executive Summary

This document presents an update of the work performed in WP1 regarding the requirements and architecture of the eFlows4HPC workflow platform. In the first part of the deliverable, requirements for the eFlows4HPC platform have been reviewed, checking if they are still valid, if the priority is still correct and evaluating their status of implementation. Regarding requirements from HPC sites, the survey has been extended to new HPC centres, which are not part of the project. We have received answers from CSC (hosting the LUMI supercomputer), HLRS (hosting Hawk and Vulcan supercomputers), IT4I (hosting Karolina and Barbora supercomputers), CINECA (hosting Marconi supercomputer), EPCC (hosting Archer and Cirrus supercomputers) and GENCI who has provided answers for the IDRIS's Jean Zay and the CEA-TGCC's Joliot-Curie HPC clusters. The analysis of these answers has produced similar results in terms of requirements which validates the requirements gathered in the first phase of the project.

The second part of the deliverable updates the eFlows4HPC architecture. First, a new component has been included in the workflow deployment part which is in charge of creating tailored container images for specific HPC systems, and other components (HPCWaaS, Software Catalogue and Workflow registry) have been updated to reflect some implementation decisions taken during the first phase of developments. Apart from the components update, a deployment diagram has been defined to clarify the different parts of the infrastructure and where the different components of the software stack are deployed. The main use cases have been updated, including the credential management and the component interactions have been updated derived by the aforementioned changes.



# 2. Introduction

One of the main current barriers for the adoption of HPC is the complexity of developing, deploying and executing complex workflows in federated HPC environments. New scientific and industrial applications require to implement workflows that combine traditional HPC simulation and modelling with big data analytics (DA) and machine learning (ML) algorithms. The integration of these different technologies in a single workflow increases the complexity of managing its entire life-cycle. Starting from the development phase, the integration of different HPC, DA and ML workflow phases requires additional programming efforts, for example by introducing some glue code which deals with the execution and data integration between the different parts of the workflow. In the deployment phase, users are required to perform complex software installations in HPC systems 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 for reliability requirements to several clusters, the required time and resources will increase. Finally, in the execution phase all the different components must be orchestrated in a dynamic and intelligent way in order to make an efficient use of resources.

The eFlows4HPC project aims at widening the access to HPC to newcomers, and, in general, to simplify the development, deployment and execution of complex workflows in HPC systems. It proposes to simplify this process in two ways. From one side, the eFlows4HPC software stack aims at providing the required functionalities to manage the lifecycle of these kind of complex workflows; from the other side, it introduces the HPC Workflow as a Service (HPCWaaS) concept, which leverages the software stack to widen access to HPC by the different communities. This service offering tries to bring the Function as a Service (FaaS) concept to the HPC environments trying to hide all the complexity of a HPC Workflow 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 implementation of forthcoming efficient HPC and data-centric applications can be developed with our proposed novel workflow technologies.

This document presents the updates in the requirements and the architecture of the eFlows4HPC software stack and the HPCWaaS concept. Section 3 updates the requirements from the different eFlows4HPC

stakeholders (Pillars, Software components and HPC sites), and Section 4 updates the software stack architecture including the main usage cases of the HPCWaaS methodology and the relation of the requirements with the software stack components.

# 3. Revision of the Requirements

The requirements gathering process for the eFlows4HPC platform was split in three main parts according to the main stakeholders on the architecture: the project pillars, that provide requirements about functionalities for implementing, deploying and executing their different workflows; the software component owners, that provide requirements about how HPC simulators as well as ML/DA frameworks and other software components of a workflow must be deployed and executed in the computing infrastructure; and finally, the HPC system administrators that provide the requirements in order to interact with HPC systems according to their security constraints and usage model.

The next paragraphs provide the update of the requirements gathered in the first period which are still valid in the second phase of the project and if their priority is still appropriate. Apart from the ID, descriptions and priority, we have also added a new column about the status of the requirement at M20. The possible status are: *Done*, when the requirement has been implemented and validated; *Pending to Validate*, when the functionality is already implemented but it is not validated by the pillar workflows; *In Progress*, when the implementation of a solution for this requirement is under development; and *Pending*, when the implementation is still to be started.

## 3.1. Requirements from Pillars

The following tables present 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), and the name, description and priority assigned by pillar teams, and the status at M20.

Most of the Pillar I requirements with Must priority have been already done. The only *Pending* requirement of this category is P1-2 “Storing of hyper-reduced model”. In the first version of the workflow, the hyper-reduced model is serialised as a JSON file, which can be easily stored and transferred. However, this format is not very efficient for being loaded in the solver, so this requirement is still open. Regarding the other requirements with less priority, most of them are *In Progress* or *Pending to Validate*, except the P1-4 and P1-8, about the including clustering models and the ML inference in the workflow, which are still *Pending*.



*Table 1. Summary of requirements from Pillar I*

| ID | Name | Description | Priority | Status M20 |
|---|---|---|---|---|
| **P1-1** | Distributed SVD | Requires an optimised distributed SVD to analyse large scale data. This is one of the most computationally intensive steps since such a matrix will be very large | Must | Done |
| **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 | Pending to Validate |
| **P1-3** | DNN model | Artificial Neural Networks (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 | In Progress |
| **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 | Pending |

| | | | | |
|---|---|---|---|---|
| **P1-5** | Persistent storage | Requires persistent storage for data to be consumed between the steps | May | In Progress |
| **P1-6** | Restart | Workflow programming and management have to allow re-start the ROM computation according to validation results. | Should | Pending to Validate |
| **P1-7** | Workflow Orchestration | Workflow management is also required through the phases to coordinate the execution of the different computing steps | Must | Done |
| **P1-8** | ML inference | Simulation code requires accessing the ML trained model | May | Pending |
| **P1-9** | Deployment | Deployment of the model and the required software in the cloud. This requires carrying around all the data needed to start from scratch a complete hyper-reduced model. | May | In Progress |

Regarding Pillar II, most of the requirements identified as *Must* are done except P2-3, P2-6 and P2-8. In P2-3, the integration with the workflow manager is already done, but it is not supporting multiple HPC infrastructures at the same time, so the requirement is not fully completed yet. Regarding P2-6 and P2-8, the implementation of the functionality is in place but the validation from the pillar's workflow is not finished. The only pending requirement in this Pillar is P2-7 which is about the integration of AI models in the ensemble member pruning. The others are *In Progress* or *Pending to Validate*.

*Table 2. Summary of requirements from Pillar II*

| ID | Name | Description | Priority | Status M20 |
|---|---|---|---|---|
| **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 | In Progress |
| **P2-2** | Portability | Workflow components should be portable to several HPC infrastructures. | Should | In Progress |
| **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 | In Progress |
| **P2-4** | Integration with permanent 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 | Pending to Validate |
| **P2-5** | Workflow adaptability | Capability to easily manage, cancel, replace and add components invocations in the workflow, for example to start the workflow from building block n. | Should | In Progress |
| **P2-6** | Access to intermediate in-memory results | The workflow manager should have the capability to retrieve data/intermediate outputs of the current running members of an ensemble on execution time directly from memory. | Must | Pending to Validate |
| **P2-7** | AI integration for ensemble member pruning | Support for applying AI techniques on intermediate data of running members to compute the members that will be discarded by the ESM workflow manager at a given step of the simulation. | Should | Pending |
| **P2-8** | ML/DL capabilities | Require the support for training and inference from Neural Network models as workflow steps. | Must | Pending to validate |
| **P2-9** | DA capabilities | Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis. | Must | Done |
| **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 | Done |

| | | | | |
|---|---|---|---|---|
| **P2-11** | Multi-member analysis | Support for concurrent execution of sub-workflows starting from different inputs (configurable) and intercomparison of the sub-workflows results. | Must | Done |
| **P2-12** | Usability | Easiness to run/manage the workflow and workflow blocks. | May | In Progress |

Regarding Pillar III, none of the requirements are fully done. However, most of them are *Pending to be validated* by the pillar workflows. It is due to some parts of the Pillar's workflows having been implemented during the first phase of the project. Therefore, the validation of these requirements is still missing. Regarding the *Pending* requirements they are: P3-3 about managing data replication, P3.5 about the integration of the workflow with the infrastructure services, and P3-7 about supporting streaming data sources.

*Table 3. Summary of requirements from Pillar III*

| ID | Name | Description | Priority | Status M20 |
|---|---|---|---|---|
| **P3-1** | Urgent computing access | Priority access to HPC computational resources | Must | In Progress |
| **P3-2** | Data interoperability | High-performance data transfers between HPC facilities | Should | Pending to Validate |
| **P3-3** | Data replication | Redundancy of data in different external repositories to assure a high-availability service. Moreover, replication of large data (e.g. computational meshes) in HPC facilities to avoid time-consumed transferences. Data redundancy should consider data replication with data stored at operation. | Must | Pending |
| **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 | Pending to Validate |
| **P3-5** | Infrastructure interoperability | Interoperability between different services (eg. Data Logistics Service, HPC clusters and microservices-based infrastructure) | Must | Pending |
| **P3-6** | Portability | Workflow components must be portable to several infrastructures | May | In Progress |
| **P3-7** | Streaming Data Source | Management of streaming data sources in real-time from external agencies or servers | Must | Pending |
| **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, cancellation), management of conditional paths, and coordination of microservices invocations | Must | Pending to Validate |
| **P3-9** | Integration with permanent storage | Support for access to external data repositories (R/W) such as EUDAT. Support for final storage in long-term storage for second use and/or to satisfy FAIRness policies | Must | Pending to Validate |
| **P3-10** | Inference of 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 | In Progress |

| | | | | |
|---|---|---|---|---|
| **P3-11** | Data Analytics integration | Predictive and prescriptive data analytics to assist some building blocks in analysis and decision tasks | May | In Progress |
| **P3-12** | Workflow malleability | Capability to cancel and add new components invocations at run-time | Should | Pending to validate |

## 3.2. Requirements from Components

Another important goal of the project is the integration of HPC, DA and ML techniques in complex workflows for simplifying its deployment and execution, and enabling their reusability. For this reason, the eFlows4HPC software stack must support the deployment and coordinate the execution of the HPC software and DA/ML frameworks required by the Pillars' workflows as well as the eFlows4HPC software stack components which must also be deployed in the computing infrastructure to manage the workflow execution and data. The following table provides a summary of the requirements identified in the first phase of the project, and their status at M20. In this first phase, we have completed all the requirements for supporting the different types of software invocations (CMP-4, CMP-5 and CMP-6). Regarding the others, the requirements for supporting HPC optimizations (CMP-1 and CMP-2) are *In Progress*, and the requirement for supporting service deployments (CMP-3) is still *Pending*.

*Table 4. Summary of requirements from Components*

| ID | Name | Description | Priority | Status M20 |
|---|---|---|---|---|
| **CMP-1** | Access to HPC-specific devices | Workflows developed with the eFlows4HPC stack must be able to access the specific HPC hardware such as High-Performance networks, accelerators or special CPU vectorial instructions. | Must | In Progress |
| **CMP-2** | Support Optimised kernels | Workflows developed with eFlows4HPC stack must be able to support the architecture-optimised kernels and libraries | Must | in-progress |
| **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 the auxiliary cloud and HPC centres | Must | Pending |
| **CMP-4** | Service Invocation | Workflows developed with eFlows4HPC stack must support the invocation of services | Must | Done |
| **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 | Done |
| **CMP-6** | Multicore execution support | Workflows developed with eFlows4HPC stack must support the execution of applications with multithreaded/multiprocess using several cores | Must | Done |



## 3.3. Constraints from HPC Centres

The last source of requirements is provided by HPC centres. Supercomputers are singular 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.

During the first phase of the requirements analysis, we produced a survey with different HPC centres. We prepared a questionnaire (available in D1.1[1]) asking 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 second phase of the project, this questionnaire has been sent to system administrators of HPC centres which are not involved in the project, to validate whether the common limitations and supported technologies identified in the first phase are still valid for these new HPC centres. We have received answers from CSC (hosting the LUMI supercomputer), HLRS (hosting Hawk and Vulcan supercomputers), IT4I (hosting Karolina and Barbora supercomputers), CINECA (hosting Marconi supercomputer), EPCC (hosting Archer and Cirrus supercomputers) and GENCI who has provided answers for the IDRIS's Jean Zay and the CEA-TGCC's Joliot-Curie HPC clusters. The answers of the survey provided by these HPC centres are summarised in the following table.

*Table 5. Summary of the results of the second round of the HPC centres survey*

| Site | | CSC | HLRS | IT4I | CINECA | GENCI | EPCC |
|---|---|---|---|---|---|---|---|
| **Access & Security** | **Access** | SSH | SSH | SSH | SSH | SSH | SSH |
| | **Identity** | SSH Keys | Password SSH Keys | SSH Keys | SSH Keys | SSH keys/ Password | SSH keys |
| | **UNICORE** | No | No | No | yes | No | No |
| | **In Conn.** | SSH | SSH from VPN or user IPs | SSH | SSH, GridFTP | restricted IP list from EU | SSH/ICMP |
| | **Out Conn.** | Allowed | Not allowed | HTTP(s), ssh scp, rsync | Allowed | None by default | DNS blocked |
| **Cluster Nodes Restrictions** | **Login** | Restricted | Restricted | Restricted | Restricted | Restricted | Restricted |
| | **Service** | No | Not Allowed | Not Allowed | Not Allowed | with restrictions | on request |
| | **Compute** | No SSH | No restrictions | No restrictions | No restrictions | No restrictions | No restrictions |
| **Queue System Shared disk** | **Queue system** | Slurm | PBS | PBS | Slurm | Slurm | Slurm/PBS |
| | **Shared disk** | Lustre | NFS (Home) Lustre (Scratch) | GPFS, HPST(BB) | Lustre/GPFS | GPFS | Lustre/XFS /CEPH |
| **Software Management** | **Modules** | Yes | Yes | Yes | Yes | Yes | Yes |
| | **Installation tools** | Spack, Easy-build | Conda | Easy-build | Spack, Conda | Easybuild, Conda (with connectivity restrictions) | Conda |
| | **Containers** | Singularity | Singularity (production) | Singularity | Singularity | Singularity/ PoCC | Singularity |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | podman/ docker (evaluation) | | | | |
| **Data Infrastructure** | **Data Interfaces** | SCP | SCP UFTP, GridFTP | SCP rsync iRODS-B2SAFE | SCP, SFTP, GridFTP, UFTP, rsync | SCP | SCP |
| | **Storage Levels** | NVMe, Lustre over SSD/HDD, object storage (ceph) | NFS/Lustre HPSS (Archive) Quobyte (Object Store) | NVMe Lustre CESNET (Archive) | Local disk Shared Tape | Scratch: GPFS over SSD Work: GPFS over HDD Storage: DMF (GPFS over HDD + Tape) | Lustre (HDD, NVMe)/ XFS/NVMe CEPH(HDD) |

Analysing the answers of the table, we can see the results are similar to the ones obtained in the previous deliverable (D1.1):

- SSH and SCP is still the common remote shell and transfer protocol supported by all the systems. So, it is still recommended that eFlows4HPC platform must support these two protocols to interact with HPC Systems
- In this second round, we still see that the availability to deploy services or daemons in login or service nodes is very limited, due to most clusters either do not provide nodes suitable to install them at user level and login nodes have limited connectivity or execution restrictions, such as execution time and memory. So, the eFlows4HPC platform can not rely on services deployed in the HPC system which must persist between executions or require external connectivity. The usage of service must be limited to the computation execution or deployed outside the HPC cluster.
- For software management, all systems use Modules for managing the environment of the installed software and containers. Singularity is still the engine supported by most of the systems. Another restriction regarding containers due to network connectivity, images can not be downloaded from external registries. So, they have to be managed as files which must be copied to the sites using the supported protocols.
- All systems have a POSIX accessible 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 amount of sites. However, the queue system is the key component in an HPC site and system administrator will not change to adopt the eFlows4HPC stack. For this reason, components of the stack that interacts with HPC systems must include mechanisms to support several queue systems.

The following table summarises the identified constraints and limitations in terms of requirements for the eFlows4HPC architecture as well as their implementation status after the first phase of the project. In this case, all the requirements have been already implemented.

*Table 6. Summary of the requirements from HPC centres*

| ID | Name | Description | Priority | Status M20 |
|---|---|---|---|---|
| **HPC-1** | HPC Cluster | The interactions between eFlows4HPC software stack and the | Must | Done |

| | | | | |
|---|---|---|---|---|
| | access support | HPC systems must at least support the SSH protocol | | |
| **HPC-2** | HPC Data Transfers support | Transfers to/from HPC clusters must support at least the SCP protocol | Must | Done |
| **HPC-3** | Singularity Container support | The usage of containers in the HPC system must be compatible with singularity containers. | Must | Done |
| **HPC-4** | Infrastructure Service deployment | The eFlows4HPC software stack can not rely only on services which require to be installed with privileged nodes or users in the supercomputing clusters. | Must | Done |
| **HPC-5** | Queue System | Supported queue systems must include at least Slurm and must provide extension mechanisms to provide other queue systems. | Must | Done |

# 4. Architecture Update

During the first phase of the project, we tried to design and implement methodologies and mechanisms to achieve the required functionalities. During this period, we have identified missing components and redefined their interactions in order to better implement these functionalities. In this section, we provide details of the major changes included in the new version of the architecture. It will describe the new components in the software stack as well as the updated component interactions. More details about the components not updated in this deliverable can be found in Deliverable 1.1 [1].

Figure 1 shows the overview of the eFlows4HPC software stack. As in the first year, it is will composed of a set of software components, organised in different layers: The first layer provides the syntax and programming models to implement and automatically operate complex workflows combining typical HPC simulations with HPDA and ML; The second layer consists of a set of services, repositories, catalogues, and registries to facilitate the accessibility and re-usability of the implemented workflows, software components, data sources and results; Finally, the lowest layers provide the functionalities to automate the deployment and execution of the workflow. This layer provides the components to orchestrate the deployment and coordinated execution of the workflow components in federated computing infrastructures. Moreover, 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. All the actions performed in the layer are performed according to the workflow description provided by the first layer of the architecture and the metadata stored in the services of the second layer.



## 4.1. Component Updates

Most of the eFlows4HPC components of the Software Stack are the same as described in Deliverable 1.1. In this second version of the architecture, a new component is introduced (Container Image Creation) and the HPC Workflow as a Service, Workflow Registry and Software Catalogue are updated.

### 4.1.1. HPC Workflow as a Service (HPCWaaS)

The HPC Workflow as a Service component provides the interfaces to Workflow developers and final workflow users to manage the different parts of the workflow's lifecycle. It is composed of two subcomponents that offer the interfaces according to the user role. Workflow developers interact with *Alien4Cloud* to develop and deploy workflows and the final workflow users interact with the *Execution API* to execute the deployed workflows.

On the one hand, *Alien4Cloud* is a web GUI which allows developers to create and deploy workflows as TOSCA topologies in a user-friendly way. This TOSCA topology describes the deployment and operation procedures of a workflow which can be deployed in different environments (HPC sites). In the case of development capabilities, it allows developers to create topologies from scratch and save them in the Workflow Registry, or reusing the existing ones to create new workflows or to deploy the same workflows in new environments.

On the other hand, the *Execution API* is a REST API and a CLI which allow the workflow users to see the deployed services in the different environments, to manage the credentials in the different environments and to execute the deployed workflows using these credentials.

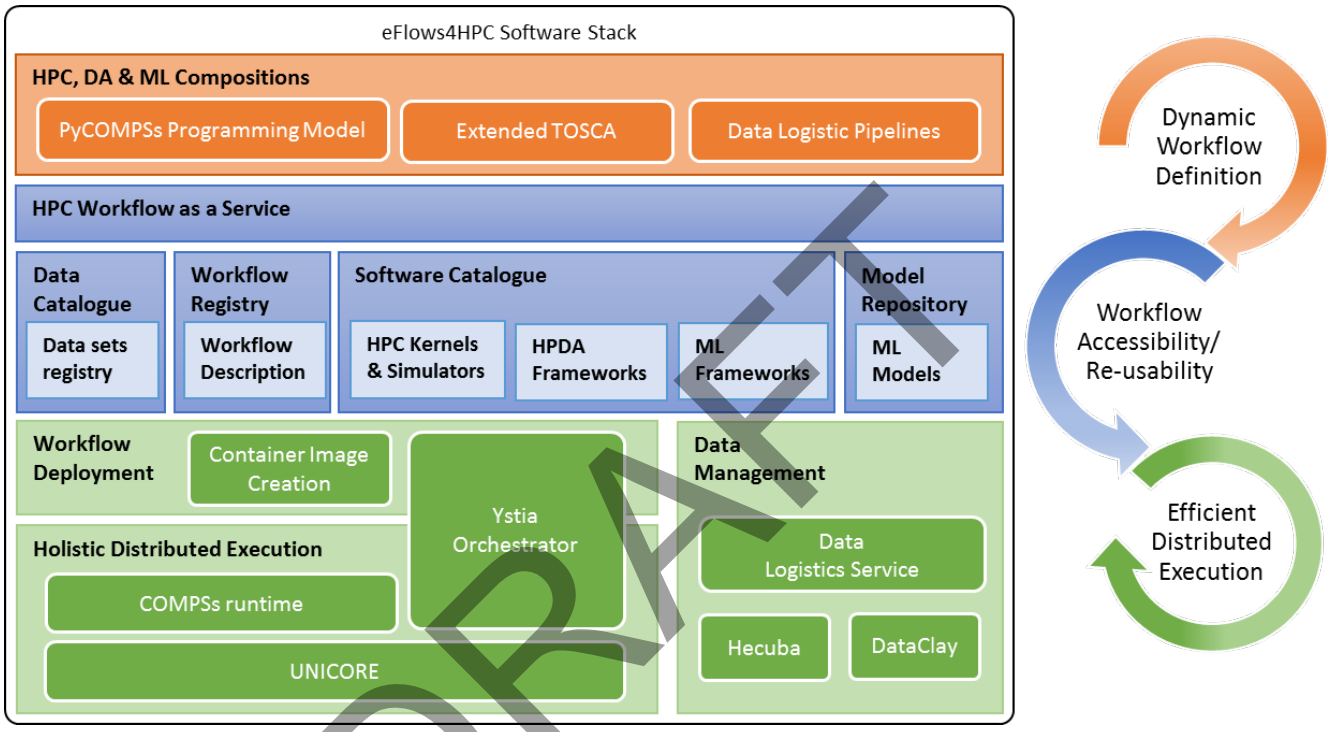


*Figure 1.eFlows4HPC Software Stack Overview*



## 4.1.2. Workflow Registry (WR)

The Workflow registry is a GIT repository which allows developers to store the developed workflows. Workflows are stored in different folders following the structure indicated in Figure 2. Inside the workflow folder, there is a *tosca* subfolder which is used to store the TOSCA description[2]. It is a topology with the relationship of workflow components including the data pipelines and PyCOMPSs computational workflows as steps of the overall workflow. Then, there is a set of folders that include the PyCOMPSs code and the software requirements of each workflow step.

To include new workflows in the registry, workflow developers have to create a new fork or branch of the git repository. In this fork/branch, they have to include a new folder for the workflow with a subfolder for the TOSCA description and the different workflow steps, as explained above. Finally, to make it available for the community, they have to create a pull request of the branch/fork containing the new workflow description to the main branch. This pull request will be reviewed by the community and included in the repository.

```
workflow-registry
  |- workflow_1
  |    |- tosca
  |    |    |- types.yml              TOSCA description of the different components involved in the workflow
  |    |       ...
  |    |- step_1
  |    |    |- spack.yml              Sofware requirements for this workflow step as a Spack environment specification
  |    |    |- src                    PyCOMPSs code of the workflow step
  |    |       ...
  |    |- step_2
  |         ....
  |- workflow_2
  |     ...
```

*Figure 2. Structure of the descriptions stored in the Workflow Registry*

### 4.1.3. Software Catalogue (SC)

The Software Catalogue is a GIT repository which allows software owners and workflow developers to store the description of the software used in the workflows in a way that the eFlows4HPC Software Stack can manage it transparently to final users. Figure 3 shows an example of software descriptions. It is following a structure compatible with software repositories of HPC build systems such as Spack[3] or Easybuild[4]. In the case of the figure, we have shown the example for the Spack system. All Software packages are stored in a *packages* folder. Inside this folder, there is a subfolder per software package where developers have to include: the package.py file with the installation description according to the Spack schemas, and different *invocation.json* files to describe the different ways to invoke the software.



```
software-catalog
  |- packages
  |    |- software_1
  |    |    |- package.py             Installation description following the Spack package format
  |    |    |- invocation.json        Description of the software invocation
  |    |       ...
  |    |- software_2
  |         ....
  |- cfg                              Spack configuration used by the Image Creation Service
  |
  |- repo.yaml                        Spack description of for this repository
```

*Figure 3. Structure of the description stored in the Software Catalogue*

To include new software packages in the catalogue, developers have to create a new fork or branch of the git repository. In this fork/branch, they have to include a new subfolder for the software with the installation and invocation descriptions, as explained above. Finally, to make it available for the community, they have to create a pull request of the branch/fork with the new software description to the main branch. This pull request will be reviewed by the community and included in the repository.

### 4.1.4. Container Image Creation (CIC)

The Container Image Creation is a component which automates the creation of the container images tailored to a specific platform. This component leverages specialised HPC builders (such as Spack[3] or

Easybuild[4]) multi-platform container build tools (such as buildx[1]) and the information provided by the eFlows4HPC Software Catalogue and Workflow Registry to automatically create optimised container images required to execute a workflow in a specific HPC machine.

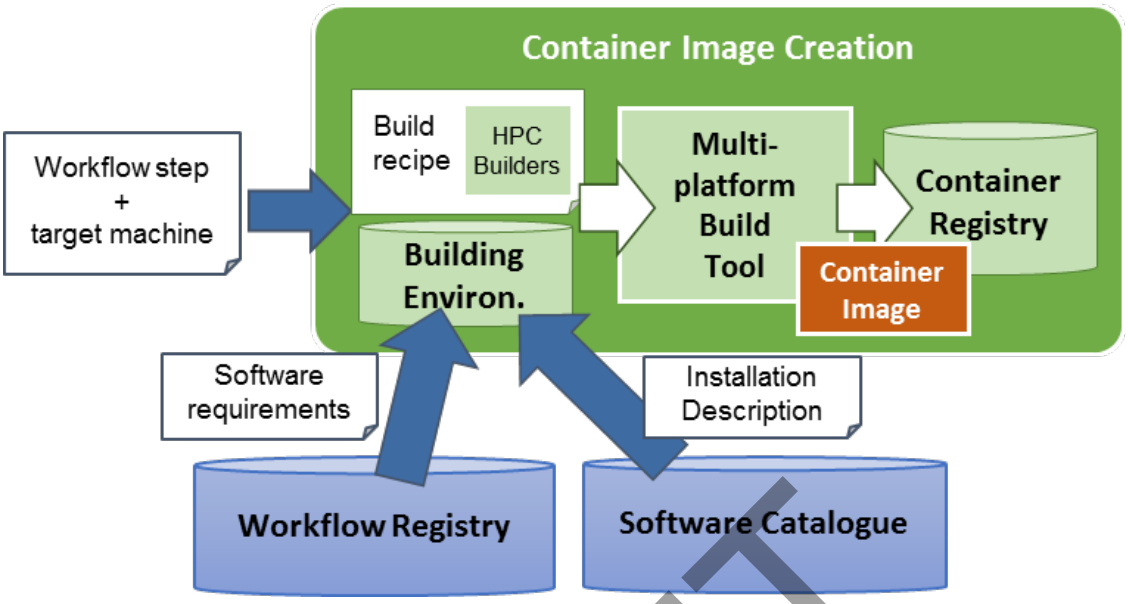


*Figure 4. Container Image Creation Overview*

As depicted in Figure 4, given a workflow registered in the Workflows registry and a description of a target platform (such as CPU architecture, available MPI versions and accelerators), the CIC orchestrates the creation of the workflow image for this platform, by generating a container building environment and a recipe with the required software and executing this recipe in the multi-platform container build tool. At the end of this process, the generated image will be stored together with a metadata description to avoid creating again the same image for a similar platform.

## 4.2. Software Stack deployment

Figure 5 shows the different components of the eFlows4HPC Software stack according to their deployment. Two types of components are identified. The *Gateway Services* are the components deployed in resources which are external to the computing infrastructures but accessible by the developers and final users through standard HTTPS protocols. They are used to expose the HPC Workflows as a Service interfaces and repositories to allow the workflow reusability and to coordinate the workflow lifecycle outside the computing infrastructure. The *Runtime Components* are the components which are used during the execution of the workflow and they must be deployed in the computing infrastructure. The interactions between the Gateway services and runtime components are limited to either standard SSH/SCP protocols (which is common in all HPC sites and Cloud Environments) or the Unicore services, if the HPC site supports this middleware. These runtime components can be also deployed inside the workflow container images together with other HPDA/ML frameworks required by the workflows.

## 4.1. Usage and component interactions

This section provides the description of the component interactions in order to implement the required functionalities for the Software Stack and the HPC Workflow as a service methodology. Figure 6 shows an overview about how the proposed solution works and the main usage cases of the different actors (developers and final users). The HPC Workflow as a Service offering is built on top of the eFlows4HPC software stack in order to provide the required functionality to develop, deploy and execute the complex services. The different usages of the HPCWaaS are organised depending on the actor: developers are in

[1] https://docs.docker.com/build/buildx/

charge of the workflow development and deployment, and the final user's communities are performing the execution of the deployed workflows. Next paragraphs provide more details about how the different eFlows4HPC components interact to provide the required functionality in the different use cases. First, we will provide the details about how a workflow is developed. Then, we will explain how the developers' and final users' credentials are managed to delegate the access to the computing infrastructure in order to perform the deployment and execution. Afterwards, we will provide the details about the workflow deployment process. Finally, we will provide the details about the workflow execution performed by the final users.

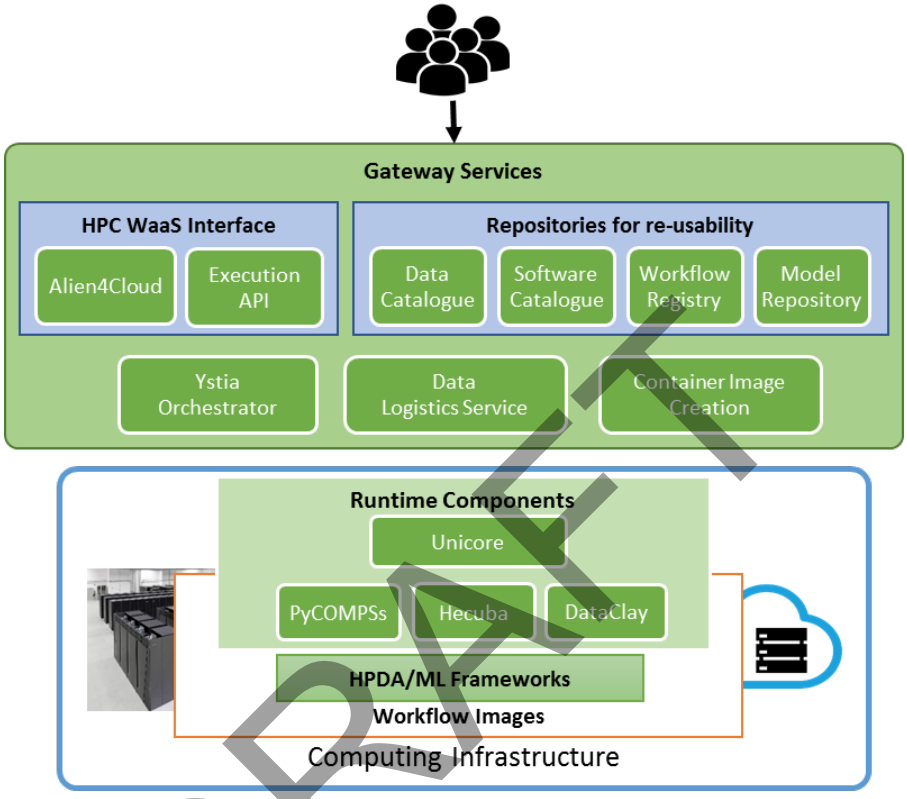


*Figure 5.eFlows4HPC Software Stack Deployment Diagram*



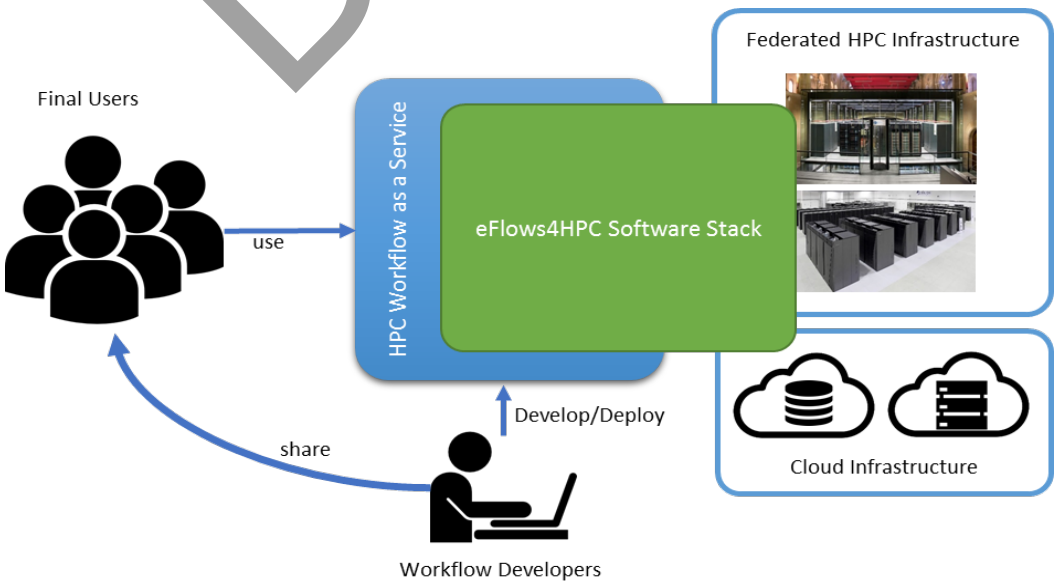


*Figure 6. HPC Workflow as a Service usage cases overview*

### 4.1.1. Workflow development

One key part of the mentioned challenges is the implementation of complex workflows that combine HPC, HPDA, and ML framework which can be easily shared and reused by different users and systems. These functionalities are provided by different components of the eFlows4HPC software stack (Figure 1). On the one hand, it provides a set of registries, catalogues 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 and combined. 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.

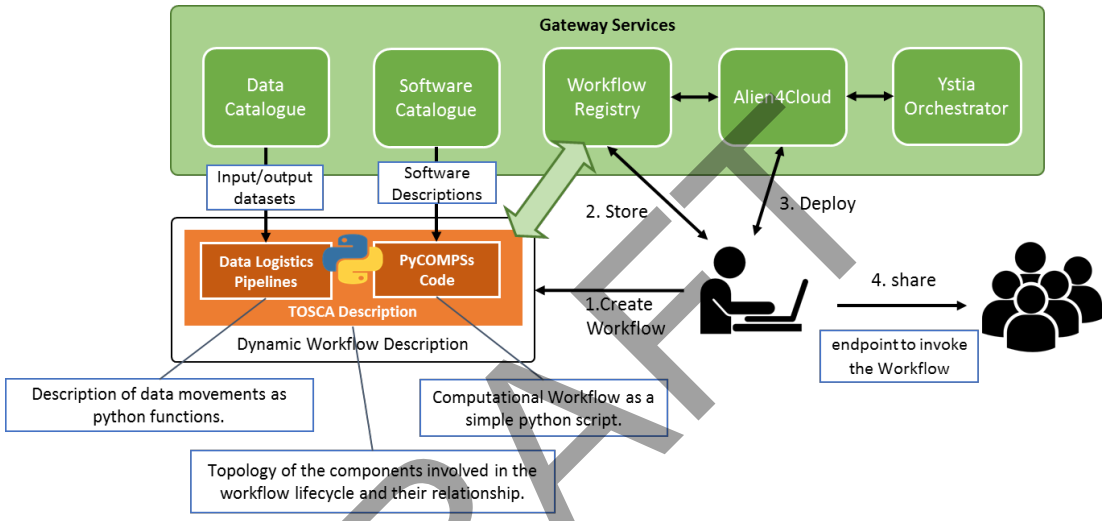


*Figure 7. Workflows development overview*



Figure 7 shows an overview about how the workflow developer interacts with the different components to develop these sharable and reusable complex workflows. The proposed workflow description is composed of a combination of Data Logistics pipelines, PyCOMPSs workflows, and an extended TOSCA description.

The data logistics pipelines allow developers to describe how the workflow data is acquired, moved and stored during the workflow execution in order to ensure the data is available in the computing infrastructure when required. The data pipelines are simple Python scripts that invoke different generic data operations to be performed on a dataset. So, the same data pipeline can be applied to different datasets, and the description of these datasets are stored in the *Data Catalogue*.

On the computational workflow side, *PyCOMPSs[5]* 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 as well as the dynamicity of the Python programming language (dealing with loop, conditionals, exceptions, etc.). 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 parallelised with alternative programming models such as OpenMP or pthreads), or multi-node (MPI applications). This PyCOMPSs mechanism is extended to enable the integration with the software descriptions stored in the *Software Catalogue*. Using this mechanism, developers can transparently include specific software invocation within their workflows by just defining a PyCOMPSs task which refers to the software description. Every time that a task is invoked in the PyCOMPSs workflow, the *COMPSs runtime[6]* will interpret this description and perform the software invocation in the computing infrastructure. In this

way, PyCOMPSs provides developers a programming model where they can naturally integrate well with data analytics and machine learning libraries, most of them offering a Python interface, as well as other types of computations such as HPC simulations.

*TOSCA* (an orchestration standard) is exploited to provide the description of the overall workflow lifecycle. It is used to describe the topology of the different components (data pipelines, PyCOMPSs workflows, or other required service/software deployment) required by the workflow and their relationship with the different steps of the workflow lifecycle. This topology is used by *Ystia Orchestrator* (*Yorc)* to orchestrate the different phases of the workflow deployment and execution.

Once the developer has finished with the workflow description, it can be stored in the *Workflow Registry* (*Step 2 in* Figure *7*) as indicated in Section 4.1.2, and deployed to the computing infrastructure by means of the *Alien4Cloud* (*Step 3 in Figure 7*). More details of the deployment use case is provided in Section 4.3.3.

### 4.1.2. Credential Management

For deploying and executing workflows in the computing infrastructure, developers and final users need to configure their credentials in order to grant the eFlows4HPC services access to the computing infrastructure on behalf of them. Figure 8 shows an overview about how credentials are managed in the eFlows4HPC architecture.

Both developers and final users have to generate a credential to access the system using the *Execution API* (SSH key pair or similar credential). This credential will be internal in a secret storage (such as Hashicorp Vault[2]) which is only accessible by the components of the *Gateway Services* (*Step 1*). As result of this procedure, a random-based token and a public key will be provided to the developer or final user. On the one hand, developers and final users have the responsibility to add the public key to the *authorized_keys* of their account in the computing infrastructure. It allows the eFlows4HPC services to access the computing infrastructure on behalf of the user. On the other hand, the random-based token is used to identify the stored credentials and it will be provided to the *Gateway services* by every time a workflow is deployed or executed. When one of these services require to interact with the computing, it queries the *Secret Storage with the provided token,* and the *Secret Storage* returns the credential to access the infrastructure. Once, the interaction with infrastructure is finished the credential is discarded.



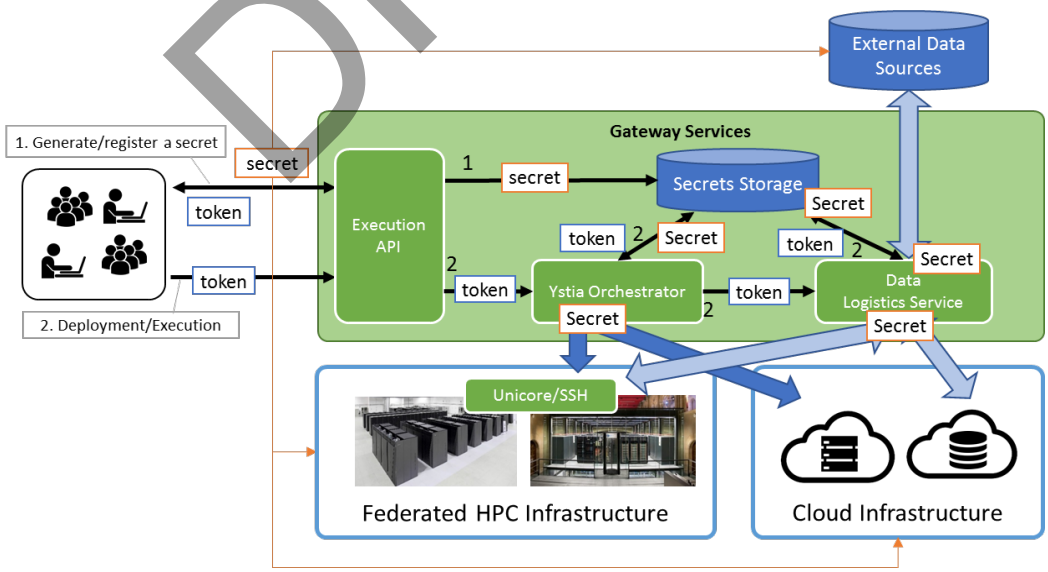


*Figure 8. Credential management overview*

[2] https://github.com/hashicorp/vault

### 4.1.3. Workflow Deployment

Figure 9 provides an overview about how components interact to provide the deployment functionalities. When developers want to execute a workflow, they use the *Alien4Cloud* interface of the HPCWaaS to indicate the workflow to deploy, select the environment (computing infrastructure) to deploy it, and provide their access token. As result of this interaction, the *Alien4Cloud* will retrieve the TOSCA description (*Step 1*) and contacts the *Ystia Orchestrator* (Yorc) is in charge of orchestrating the deployment of the main workflow components in the computing infrastructures and managing their lifecycle as indicated in the TOSCA part in the workflow description (*Step 2*). The actions orchestrated by *Yorc* include the interactions with *Container Image Creation* component to perform the creation of the container images for the selected environment (*Step 3*), and the interactions *Data Logistics Service* to set up the data pipelines to transfer the generated container images and other common datasets or models which are required by all the workflow executions (*Step 4*). The access to the HPC infrastructure can be done either SSH/SCP protocols which is the common access protocol to these types of infrastructures or through the Unicore services in the case that this middleware is available in the HPC infrastructure.

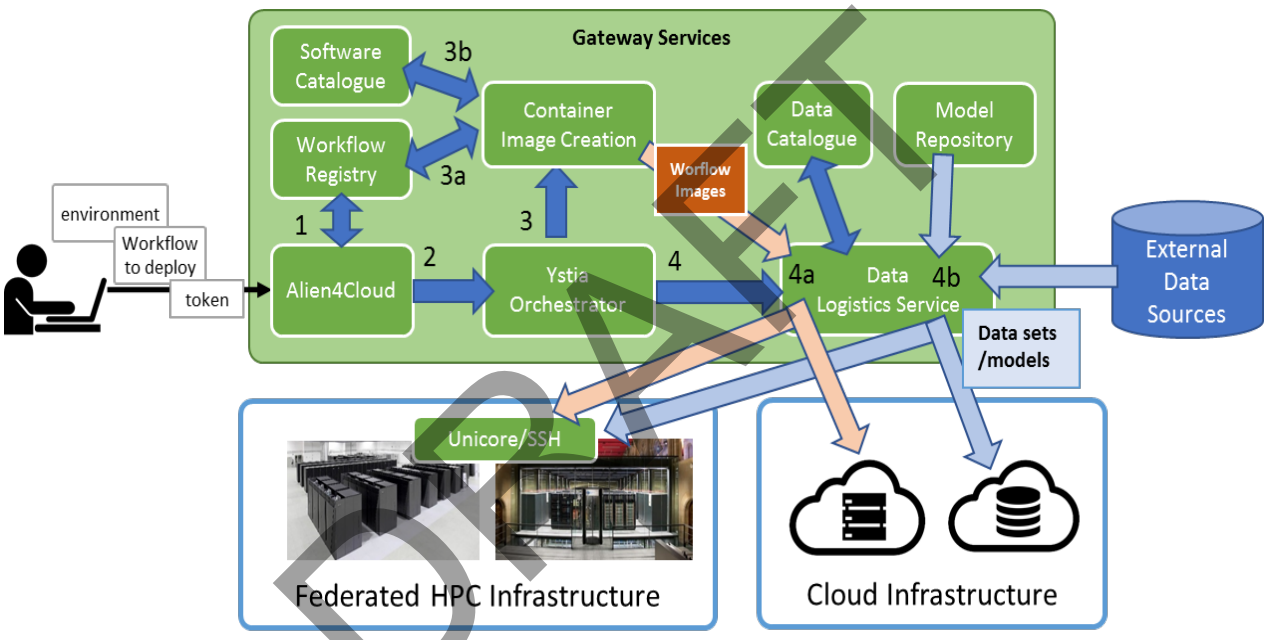


*Figure 9. Workflow deployment overview*

### 4.1.4. Workflow Execution

Once the workflow is deployed, the final users can use the *Execution API* of the HPC Workflow as a Service interface to submit the workflow executions in the computing infrastructure (Figure 10). The Execution API will contact to *Yorc* (*Step 1*) which will orchestrate the execution part of the TOSCA description. It includes the execution of the computation (*Step 2a*) where Yorc executes the PyCOMPSs workflows, and the data pipelines (*Step 2b*), where *Yorc* sets-up the data pipelines which must be active during the execution (such as the data stage-in and stage-out, or periodical transfers to synchronise data produced outside the HPC systems).

Regarding the execution computation, it is managed by the *COMPSs runtime* which coordinates the execution of the different computations implemented with the PyCOMPSs programming model in the available computing resources. As mentioned before, *PyCOMPSs* supports several task types which can include either HPC simulations as well as Data Analytics or Machine learning algorithms [7]. The *COMPSs runtime* dynamically generates a task-dependency graph by analysing 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 such 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 get the maximum performance from 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 behaviour accordingly. These functionalities, together with similar features provided by Yorc at a higher level, offer the possibility of supporting workflows with a very dynamic behaviour, that can change their configuration at execution time upon the occurrence of given events, such as failures or exceptions.

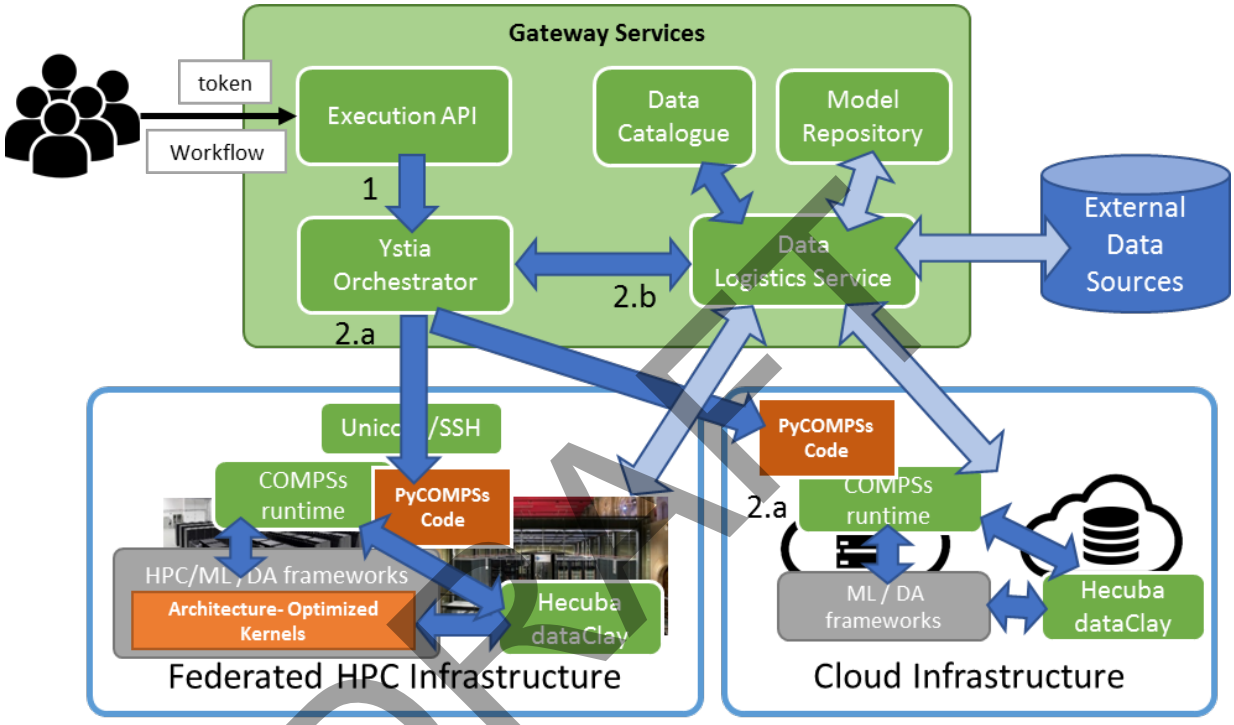


*Figure 10. Workflow execution overview*



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 visualisation or analytics, can be easily implemented.

## 4.2. 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. This table has been updated in the second phase of the project including the new CIC component.

*Table 7. Components requirements matrix. First characters of the ID indicates the source of the requirement (PX for Pillars, CMP for Components, HPC for HPC centres). The acronyms of the Components can be found in Section 7.*

| Requirements | | Components Involved | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | Description | HPCWaaS | DC | SC | WR | MR | CIC | 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 | DNN model | | | | | x | | | x | | | | | | |
| P1-4 | Clustering model | | | | | x | | x | x | | | | | | |
| P1-5 | Persistent storage | | | | | | | | | | | | | x | x |
| P1-6 | Restart | | | | | | | | | x | x | x | | | |
| P1-8 | ML inference | | | | | | | | x | | | | | | |
| P1-9 | Deployment | x | | | | | x | | | | x | | | | |
| P2-1/P3-4 | Execution Robustness | | | | | | | | | x | x | x | | | |
| P2-2/P3-6 | Portability | x | | 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 |
| **P2-12** | Usability | x | x | x | x | | | | | | | | | | |
| **P3-1** | Urgent computing access | | | | | | | | | | x | x | | | |
| **P3-2** | Data interoperability | | | | | | | | | | | | 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** | Data Analytics integration | | | | | | | x | | x | | | | x | x |
| **P3-12** | Workflow malleability | | | | | | | | | x | x | x | | | |
| **CMP-1** | Access to HPC specific devices | | | x | | | x | x | x | x | x | | | | |
| **CMP-2** | Support Optimised kernels | | | x | | | 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 | x |
| **HPC-4** | Infrastructure Service deployment | | | | | | | | | | x | x | x | | |
| **HPC-5** | Queue System | | | | | | | | | x | x | x | | | |



# 5. Conclusions

After the first implementation phase of the eFlows4HPC project, we have conducted a revision of the requirements and the architecture. Regarding the Pillar's requirements, they have been reviewed with the help of the WP4, WP5, WP6 partners to identify they gathered requirements are still valid, their priority is correct and what is the current implementation status to identify what are the most important missing

requirements to be prioritised in the following implementation phases. The same procedure has been followed for the software components requirements where we have identified some missing requirements. Regarding the HPC centres, we have updated our survey including HPC centres (which are not part of the project) in order to validate that the requirements gathered in the first phase are still relevant.

As a consequence of the first implementation phase, we have identified some gaps or not clearly defined parts in the architecture which are updated in this deliverable. A new component (CIC) has been included in the workflow deployment part which is in charge of creating tailored container images for specific HPC systems, and the HPCWaaS, Software Catalogue and Workflow Registry components have been updated. Apart from the components update, a deployment diagram has been defined to clarify the different parts of the infrastructure and where the different components of the software stack are deployed. Finally, the main use cases have been updated, including the credential management and the component interactions have been updated to include the component changes and clarify their interactions.

# 6. Acknowledgement

We want to thank the collaboration the member of the following institutions for dedicating part of their time in providing the answers to the HPC centres survey: Jesse Harrison and Henrik Nortamo from CSC, Bastian Koller and Jochen Buchholz from HLRS, Martin Golasowski from IT4I, Daniele Ottaviani from CINECA, Mark Sawyer and Kieran Leach from EPCC, and Stéphane Requena from GENCI.

# 7. Acronyms and Abbreviations

- AI - Artificial Intelligence
- API - Application Programming Interface
- CIC - Container Image Creation
- CLI - Command Line Interface
- CPU - Central Processing Unit
- D – deliverable
- DA - Data Analytics
- DAG - Directed Acyclic Graph
- DC - Data Catalogue
- DL - Deep Learning
- DLS - Data Logistics Service
- DMCF - Data Mining Cloud Framework
- 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
- 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
- ParSoDA - Parallel Social Data Analytics
- POSIX - Portable Operating System Interface
- PRACE - Partnership for Advanced Computing in Europe
- REST - Representational State Transfer
- SC - Software Catalogue
- SCP - Secure Copy
- SSD - Solid State Disk
- SSH - Secure Shell
- SVD - Singular Vector Decomposition
- TOSCA - Topology and Orchestration Specification for Cloud Applications
- UI - User Interface
- VPN - Virtual Private Network
- WP – Work Package
- WR - Workflow Registry



# 8. References

[1] eFlows4HPC Consortium. “D1.1 Requirements, Metrics and Architecture Design”, 2021.
[2] 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
[3] Gamblin, Todd, et al. "The Spack package manager: bringing order to HPC software chaos." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015
[4] Hoste, Kenneth, et al. "Easybuild: Building software with ease." 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. IEEE, 2012.
[5] Tejedor, E., et al. "PyCOMPSs: Parallel computational workflows in Python." The International Journal of High Performance Computing Applications 31.1 (2017): 66-82.
[6] Badia, Rosa M., et al. "Comp superscalar, an interoperable programming framework." SoftwareX 3 (2015): 32-36.
[7] Elshazly H, Lordan F, Ejarque J, Badia RM. “Performance Meets Programmabilty: Enabling Native Python MPI Tasks In PyCOMPSs”. In Proceeding of the 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) (2020) (pp. 63-66). IEEE.