Data Logistics Service in eFlows4HPC

Jedrzej Rybicki*, Christian Böttcher*
* Juelich Supercompuging Center, Juelich, Germany
{j.rybicki, c.boettcher}@fz-juelich.de

***Abstract*—Modern scientific endeavors often require complex, data-intensive workflows leveraging distributed and heterogeneous computing and data resources. Such workflows often include multiple steps of classical simulations, but increasingly also ML and AI components. As a result, they use not only HPC, but also Cloud-like resources. Efficient and user-friendly execution and management of such workflows pose many challenges. In this paper, we share our experience in implementing three such workflows in the eFlows4HPC project. We focus, however, on the data management dimension of the workflows. How to ensure the timely availability of the required data, how to move data to and from compute resources, and how to make the workflows complete and portable. To this end, we implemented the Data Logistics Service, integrated it with the workflow execution engine, and defined multiple data movement pipelines to cater for specific scientific needs. We will share our experience from implementation and operation of the service. This will include building a solution for continuous deployment and access management in a federated environment. On a more abstract level, we also explore how the presented approach fits into the vision of the FAIR paradigm.**

Keywords*—*High Performance Computing, Distributed Data, Cloud

I. Introduction

Modern science requires computations that involve multiple steps such as simulations, big data analytics using machine learning, and application of artificial intelligence (AI). In addition, such workflows often use heterogeneous resources such as HPC, various accelerators, and Clouds. Finally, the whole is driven by data and constrained by the need for transparency, reproducibility, and portability. All this poses a set of challenges that significantly raise the entry bar for scientists, especially newcomers. The eFlows4HPC project aims to enable seamless execution of complex workflows by providing a software stack and a set of services to implement three scientific workflows from geoscience, Industry 4.0, and climate modeling [1].

In this paper, we describe our experience in supporting project-specific workflows (Pillars) with a focus on the data-related dimension. We will expand on the previously proposed Data Logistics Service (DLS) [2] and show how it was used to address these scientific challenges. In particular, we describe what data movements were required, which sources and targets it had to be integrated with. We will share our approach of making the data movement pipelines more generic while at the same time making the data more visible.

For the sake of portability, container technologies (Docker and Singularity) were used to encapsulate scientific codes. On an abstract level, container images are just files that need to be present on the site before the execution starts. Some containers runtime are able to retrieve images from image repositories on Internet, but this does not work on the HPC compute nodes as they usually have very restricted or no Internet access. Thus, the DLS was also used to schedule and conduct the image transfers in the stage-in phase.

Lastly, the workflows in the project were utilizing ML and AI. Powerful HPC resources were used to create models fulled by data. The creation (training) of such models require lot of time and computation but the trained models can often be reused. To extract the models and make them available, Data Logistics pipelines were used. The models were staged-out to an external model repositories and made available there with addition metadata describing both training parameters and model performance.

It should be clear by now, that many different data movement formalizations (data pipelines) were required. In the technology we used as the basis for the Data Logistics Service, the pipelines are defined in a very popular programming language (Python). However, the pipelines need to be deployed to the DLS to make them available for the workflows' developers and users. For this purpose, we implemented an automatic continuous deployment (CD) process that was able to retrieve the new pipelines from the project repository, test them to verify that no regression bugs had been added, and deploy the pipelines to the running instances of the Data Logistics Service. Updates to the software that runs the service itself were also handled by the CD process.

An often overlooked problem when developing distributed services is the credentials management and authentication and authorization solution (AAI). In our case the problem was twofold. Firstly, we needed to provide access to the Data Logistics Service, so that the users can view the data pipelines and also information about their execution. Furthermore, the workflows executions should be able to perform data movements on users behalf. In this paper, we show how we addressed the problems, providing a working solution based on mature technologies.

The rest of this paper is structured as follows: To set up the context we will describe related work in Section II and shortly introduce the project eFlows4HPC itself in Section III. We then proceed to describe how the DLS was implemented and operated in the project (Section IV) and show a workflow implementation and produce an example of image transfer optimization (Section V). We conclude our work with a recap and future look in Section VI.

II. Related Work

There are products out there to support data movement, with prominent examples of FTS [3], IPFS [4], or iRODS [5]. They can be thought of as distributed content storage networks. Internally, they move the data and provide ways to retrieve objects or files. From our point of view, such solutions could be easily integrated into the DLS if required by the users and thus be treated as data repositories. However, the direct application of the underlying data movement for our use case is not possible. It would require the target execution environment to somehow integrate with the storage solution by installing the necessary software and enabling product-specific protocols, which not all participating HPC centers are willing to do. Our solution is much less intrusive in this regard.

We also wanted the data movement definitions to be part of the workflow description. In our vision, the workflow should describe how to prepare and perform a particular scientific task, rather than just assuming that the data (or software) is already in place. This also means that a data movement solution like Globus on-line [6], will not be sufficient. The DLS offers a way of shareable formalization of the data movement in a general programming language. We believe that above-mentioned products are compatible with our solution. They can be either sources, or targets of the data movement, or can even facilitate the data movements triggered by the DLS. To some extend this is similar to the situation that the high-level workflows in eFlows4HPC can cater different workflow system for the computations. The DLS can use different storage solutions and transfer protocols.

The idea of making scientific computations FAIR is spreading rapidly, gaining support from research communities, and funding agencies alike. The effort is crucial to make the science sustainable. This trend also arrived at the workflow communities. There are first works pointing out FAIR challenges specific to workflows like Wilkinson et al [7]. One can argue that the solution proposed by the eFlows4HPC by publishing the workflows in an open repository or provide an API to reuse existing workflows, is already a substantial step towards the vision of FAIR research. On the DLS part, the formalization of the data movement in a human and machine-readable way makes finding, accessing, and reusing of the data much easier. The DLS also supports variety of (standardized) data transfer protocols to increase the interoperability of the solution.

III. The eFlows4HPC Project

This section will provide short overview of the problems the project eFlows4HPC is tackling and will help to derive requirements that were used to design and implement the Data Logistics Service. We will not be very extensive in our descriptions and rather focus on the data dimension of the described use cases.

A. Use-case Overview

The eFlows4HPC project delivers a workflow software stack and an additional set of services to enable the integration of HPC simulations and modelling with big data analytics and machine learning in scientific and industrial applications. The software stack allows for the creation of innovative adaptive workflows that efficiently use computing resources and incorporate novel storage solutions [1].

The project is driven by three scientific use-cases. The first one comes from the field of manufacturing and is implementing Digital Twin for complex manufacturing objects. The vision is to obtain a model that will accompany the actual physical object (requiring deployment on the edge) and make prediction of the properties and states of the actual device in the real-time. Such model cannot be too complicated due to limited computation capabilities present. Thus the idea is to use HPC systems to build high-fidelity models (Full Order Model, FOM) and then use order reduction process to create useful, yet less detailed model (Reduced Order Model, ROM). The development of the FOM is driven by the training data obtained in experiments and the creation of the FOM is a multi-step workflow as described above.

Second Pillar application of eFlows4HPC stems from climate modelling. The scientific task at hand is tropical cyclone modeling [8]. The modeling requires multiple two dimensional fields like pressure, temperature, wind velocity, etc. Typical end-to-end Earth System Modeling (ESM) workflow consists of different steps including data pre-processing, HPC simulations, visualizations and produces large amount of output data that need to be managed. The challenge is further amplified by the fact that ESM often uses ensemble simulations to asses the uncertainty of the models, increasing the number of simulations significantly.

The last use case covered in eFlows4HPC is the urgent computing for natural hazards [9]. The scenario here is to apply instant HPC simulation as soon as earthquake is detected to assess the probability and magnitude of emerging tsunami. Here the model is driven by the obtained data and also required some pre-processing and data describing particular geographical area. Often ensemble simulations are used as a means of assessing the uncertainty. The results need to be analyzed, visualized and distributed quickly to decision-makers.

B. The eFlows4HPC Solution

The challenges in the eFlows4HPC project are twofold. The first is to provide support for the implementation and optimisation of the above-mentioned workflows. This is achieved through close collaboration between domain experts (workflow developers) and computer scientists from HPC centres. These teams describe high-level workflows in the open standard TOSCA (OASIS Topology and Orchestration Specification for Cloud Applications [10]). The workflows are intended to be a complete description of the scientific process and consist of data movement

descriptions as well as execution steps. In the execution steps, lower level workflows such as PyCOMPSs [11] can be used. Once a workflow is defined, it can be executed by tools implementing the TOSCA standard.

Second challenge is to share the workflows within the community, where other users beside the workflow developer can use, execute, and modify them. To this end, an HPC Workflow-as-a-Service (HPCWaaS) solution was implemented. In short, it offers a simple API that the end-users can use to trigger a workflow execution. The execution can be then provided with input parameters defining, e.g. which dataset should be used as input, or what parameters should be passed to HPC simulation, etc.

It should be clearly stated that we strive for portability of the workflows. Thus, the preferred way of defining computation steps is with help of Singularity containers, which are then deployed to the target location. To summarize, the eFlows4HPC workflows include parts describing the required datasets, and the software used. During the execution the data pipelines are triggered to transfer required data, and stage-in containers with the software. Afterwards the computations are started. It should be stressed that the workflow orchestrator can also build the containers for particular target architecture [12], resulting in fully-fledged, complete description of the workflow.

IV. Design of the Data Logistics Service

Equipped with a basic understanding of the challenges the project is tackling, especially in terms of data management, we are ready to devise set of requirements that will guide the implementation of the Data Logistics Service. It should be a standalone service, that provides an API for integration with a higher-level orchestration engine. The primary goal of the DLS will be to ensure that the data required by certain computations are in the right place (infrastructure) at the right time. The resulting data movement can be categorized as stage-in phase, stage-out of the results, and data movement between infrastructures (e.g. intermediate results traveling from HPC to Cloud). The description of the data movement should be abstract, yet understandable to humans, and executable without their participation. Furthermore, it should be possible to extend the service to integrate new data storages, e.g., new repositories. Lastly, we want to provide users with a self-service for the users allowing them to add new data movement descriptions. We call the formalization of a data movement a *data pipeline*. In general, each described workflow will require at least two data pipelines (stage-in, and -out).

To further refine the requirements for the Data Logistics Service, we started the project with the implementation of a minimal workflow. This workflow was intended to use all the essential moving parts of the eFlows4HPC software stack, present the idea to the end user, drive the integration of the software elements, refine the requirements, and potentially identify missing parts. The minimal workflow consists of a stage-in phase (where a dataset should be ingested into an HPC system), a computational step (in the singularity container), and a stage-out of the results into an external data repository. As a data repository for stage-in and stage-out datasets, EUDAT B2SHARE [13] was used. For the HPC access, it was decided to start with SSH as access protocol as this was the least common denominator across the centers in the project. Thus, the stage-in and stage-out pipelines became HTTP API to SSH transmissions (and other way round for the stage-out). The minimal workflow also required the presence of the Singularity image in the HPC center, but from the data point of view this was again a transfer pipeline between the HTTP API of the image repository and HPC storage accessible through SSH.

A. Implementation of the Data Pipelines

After implementing the minimal workflow, we were ready to proceed to solicit the Pillars for more specific requirements. We used the minimal workflow to showcase the functionality and make the discussions more concrete. This process iterated thorough the project lifetime and resulted in data pipelines as listed in Table I).

It can be seen that most popular data protocol used was indeed SSH due to its high proliferation among the participating centers. Most popular repositories were EUDAT B2SHARE and EUDAT B2DROP. In two cases a generic HTTP-based pipeline was used. Also the interaction with the image repositories was based on HTTP transfer. To this end we implemented pretty efficient pipelines that avoided the storage on DLS and just copied two streams.

The source code of the pipelines implemented in the project can be found in the open GitHub repository [14]. The DLS was based on Apache Airflow [15]. In this software, the data pipelines are defined as directed acyclic graphs (DAGs). In short, that means that the task forming the pipeline depend on each other, and there are no circular dependencies.

B. Data Catalogue

One of the first lessons learned from implementing the workflows was to make the pipelines as generic as possible. We found that there was a list of common repositories used to store the data, and that users would want to run the same workflows for different input datasets. Also, an iterative process where the output of one workflow is used as input for the next run of the workflow was a common scenario. Therefore, we decided to add a component to our service stack called the Data Catalogue (DC). This component stores information about available (and produced) datasets. It can be understood as a simple key/value store. Where the key is the record identifier used as a parameter for the stage-in. The value is a set of metadata (freely defined by the user) with a prominent field describing the location of the data in an external repository. Armed with the ability to define custom datasets in the Data Catalog, the user is able to run a workflow defined by the workflow developer to analyze a particular dataset.

TABLE I: Data pipelines

Source	Target	Comment
B2DROP	SSH, Unicore	Stage-in form WebDAV based repository
SSH	SSH	Copy between two HPC centers
HTTP(S)	SSH	Simple stage-in from any resource accessible thorough HTTP
B2SHARE	SSH	Data objects stage in
SSH	B2SHARE	Data objects stage out (with metadata management)
Model repository	SSH	Stage-in of existing ML models
SSH	Model repository	Registration of ML model with a repository
Image repository	SSH	Container stage-in

Datasets created during a workflow execution are also automatically registered in the Data Catalog. Here we needed, however, to make the process a little bit more flexible. Most of the scientific repositories required some minimal set of metadata to be stored along the actual data. A good example is, often required, the Dublin Core Metadata Element Set (DCMES [16]). In our approach, the user can define the values for the metadata required by the target repository in the Data Catalogue. The identifier of this record in the DC will be passed over to the stage-out pipeline and upon registration to the external repository, the record will be used as a "cookie-cutter" schema for the newly registered dataset. After the registration the location of the dataset in the external repository is registered in the Data Catalogue and can be used as an input for the stage-in pipelines. This approach allows for pretty high flexibility, the users of Data Logistics Service can reuse the existing pipelines to stage-in and -out their datasets. They don't have to modify (or even understand) the pipelines. Finally, the process can be executed without human intervention by the workflow orchestration software.

C. Pipeline Deployment

Despite the flexibility provided by the usage of Data Catalogue, sometimes a need arises to develop a new pipeline. Typically, a new source of data (repository) need to be integrated, or the workflow developer needs more sophisticated usecase, e.g., a pipeline that not only transfers the data but also transforms them in a particular manner. To speed up the devolvement and deployment of such pipelines we implemented a continuous deployment strategy. This is implemented with help of gitlab pipelines. Each time a new pipeline is added to the repository, a new (test) instance of the Data Logistics Service is created. If the test doesn't brake the service (e.g. due to some programming error), the test service is promoted to the main instance and the new pipeline is available to the users. The same process is used to maintain the periodic updates of the service internals. In the 2 years of the project more than 400 automatic deployments were conducted. By enabling such quick turnovers, we were able to focus on the features development, and always provide the users with most up-to-date solutions. The automatic deployment process also benefits the users who want to extend the set of data movement functionalities. They can add new pipelines in an easy, self-service manner without need to involve the service admins.

D. Credentials and Access Management

Since workflow users and workflow developers require different levels of access to the DLS and the Data Catalogue, we needed to implement some kind of access management. In particular, we needed to prevent changes to the datasets in the Data Catalogue and to the pipelines in the DLS by unauthorized people. In a first step, which was executed early in the project, this meant adding a simple login mechanism to the Data Catalogue, which supported a local user database with some basic access levels - mainly it was used to lock the public out of making any modifications to the data. For the sake of transparency and goal of sharing results of the project, the read-only access to the Data Catalogue is granted. For the DLS, it was easy to use the login mechanisms that the underlying software (Apache Airflow [15]) used, which was also based on a local user database. Here, we already differentiated between read-only access for workflow users, and a more comprehensive access for developers, who could also manually execute the pipelines and look at lower-level logs.

However, after some time, we realized, that across multiple deployments, it became rather unpractical to manually manage user access at each different location. To this end we started to manage an Identity Provider (IdP) based on the software Unity [17]. Focusing on the relevant details of Unity for the DLS, Unity can act as an OAuth2 [18] provider, which is a protocol for authenticating users via an external user management system. Using this, we could then implement another login path for users and developers, where Unity verifies the credentials and provides the required access levels to the DLS. This was then also extended to some other eFlows4HPC services, resulting in a true Single-Sign-On experience for the users.

The second challenge related to access management was the question on how to enable DLS access to data repositories, or HPC systems. Since this was not unique to the DLS (the workflow orchestrator also needed access on the users behalf), we decided not to duplicate the solution but rather use the same approach the orchestrator used and integrate the external component: Vault with the DLS. Vault is a mature solution for managing secrets and sensitive data [19]. It is widely used and acknowledged as a good and secure solution [20]. In this manner, all the data transfers were conducted on user-behalf rather than with a general project-wide credentials which would constitute a security risk and would not be accepted by the HPC

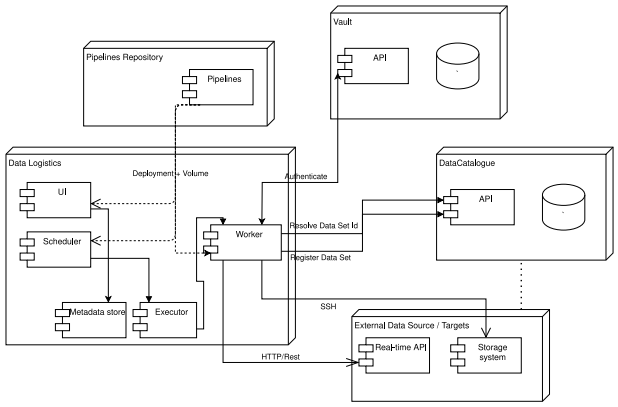

Fig. 1: Component overview of the DLS.

centers.

E. Resulting DLS Architecture

The development of the Data Logistics Service was driven by the requirements obtained of the three user communities participating in the eFlows4HPC project, as well as technical requirements from the participating HPC centers. They were then refined through the process of implementing the minimal workflow (described above) and the subsequent implementation of the Pillars' workflows. An overview of the final components interaction can be found in Figure 1. This overview does not include the infrastructural cornerstones like IaaS Cloud offering [21] used for deployment of the components or the gitlab pipelines responsible for the continuous deployment. Also for clarity the higher-level services (workflow orchestrator, HPCWaaS, etc) are omitted.

V. Example

Let us now look at a concrete example. This will be based on the excerpts from the natural hazard workflow. As already explained, the high-level workflows are defined using Alien4Cloud and stored in the TOSCA standard. Users or workflow creators don't have to worry about the code describing the workflow, as there is a web-based GUI to quickly assemble it from available blocks. Internally, the blocks and their connections are described in the topology file. As an example, let us analyze how this looks like. The whole workflow can be found in the open GitHub repository [22].

The code for the workflow can be seen on Listing 1. We have omitted large chunks of code to capture only the most important parts. First, the workflow build blocks in `topology_template` section. We see a block for the image transfer. Since all executions in the workflow run in containers, we ingest the image to the HPC site. The two remaining blocks of the workflow are the low-level workflow execution with PyCOMPSs and the upload of the results from the model repository. The latter block also includes the name of the DLS pipeline (`model_search_upload`).

Listing 1: Workflow description

```
topology_template:
  ImageTransfer:
    type: dls.ansible.nodes.DLSDAGImageTransfer
    properties:
      dag_id: "transfer_image"
  Model_search_results_upload:
    type: dls.ansible.nodes.DLSDAGModelSearchUpload
    properties:
      dag_id: "model_search_upload"
  PyCOMPSJob_run:
    target: PyCOMPSJob
workflows:
  modelSelection:
    steps:
      PyCOMPSJob_run:
        target: PyCOMPSJob
      Model_search_results_upload_run:
        target: Model_search_results_upload
```

The actual computation is done using the PyCOMPS framework, a snippet of the Python code can be seen in Listing 2. The code performs a typical parameter search for an ML model. It then serializes the results of the model search (i.e., the performance of the intermediate models) as well as the best model.

Listing 2: Model training

```
rf = RandomForestRegressor(n_estimators=15, try_features
     = 'third')
searcher = GridSearchCV(rf, parameters, cv=3)
pd_df = pd.DataFrame.from_dict(searcher.cv_results_)
pd_df.to_csv(pd.csv')
searcher.best_estimator_.save_model('model.dat')
```

The final part of the workflow is to upload the results of the model search along with the best model to the model repository. This task is delegated to the Data Logistics Service and performed by a dedicated pipeline. The main parts of the pipeline can be seen in Listing 3. After accessing the serialized results from the previous step (see: Listing 2) via SSH protocol (this part is omitted in the pipeline code). The results are parsed and uploaded to the model repository. In particular, a distinction is made between model parameters (uploaded with function `mlflow.log_param`) and model evaluation metrics (`mlflow.log_metric`). Based on the metric results, the best model is identified and added to the repository (`mlflow.log_artifact`).

Listing 3: Model upload pipeline

```
e = client.get_experiment_by_name(experiment_name)
df = pd.read_csv('pd.csv')
for i, p in enumerate(dct['params'].values()):
    with mlflow.start_run(experiment_id=e) as run:
        for parname, pvalue in p.items():
            mlflow.log_param(key=parname, value=pvalue)
        for m in metrics:
            mlflow.log_metric(key=m, value=dct[m][i])
        if dct['rank_test_score'][i]==1:
            mlflow.log_artifact(local_path='model.dat')
```

Given the workflow described above, we were able to optimize its performance. In eFlows4HPC, workflow optimization was a multifaceted process. It included optimizing the machine learning libraries used in the workflows, setting the right parameters for execution, and using high-performance storage solutions. For the Data Logistics Service, an obvious area for improvement were the transfers. Fig. 2 shows the transfer times for the

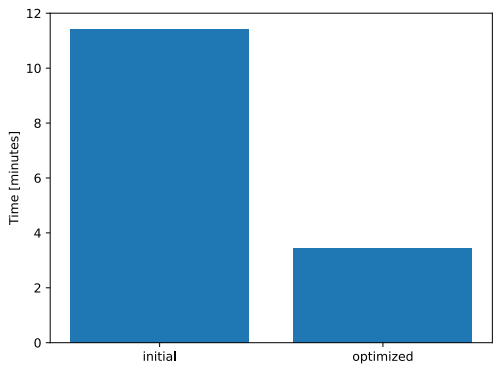

Fig. 2: Image transfer times

container image. The size of the image was 640MB, and it was transferred from an image repository with HTTP-based access to an HPC site accessible via SSH. By using transfer optimizations such as SSH and HTTP pipelining, as well as direct memory access, we were able to significantly reduce the transfer times. Note that all workflows developed in the eFlows4HPC project were run in containers, so any image transfer optimizations trickled down to all executions. Furthermore, the same optimizations were applied to all HTTP-to-SSH transfers.

VI. Conclusion

In this paper, we shared our experience in implementing the Data Logistics Service. This service was a central part of the eFlows4HPC software stack, used to implement real scientific workflows. We explain the key requirements and design decisions made in the process. We believe that the solution can be used in the modern scientific infrastructures as it provides ways of making data movements (an everyday activity in modern research) easy, shareable, and generic. In a broader sense, it contributes to the vision of FAIR research. Especially with the remaining parts of the eFlows4HPC solution, it is a demonstration that the scientific workflows can be implemented in such a way that they are transparent, efficient, and portable.

In our future work, we plan to extend the Data Logistics Service beyond simple data transfer but to off-load parts of data processing activities into the pipelines. That would have clear benefit to save the more expensive HPC resources, and would also contribute to reproducibility of such operations. We show some initial evaluation of the Data Logistics performance, in the future an extended comparison with existing products might be worth exploring.

Acknowledgment

This work 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, and Norway. In Germany, it has received complementary funding from the German Federal Ministry of Education and Research (contracts 16HPC016K, 6GPC016K, 16HPC017 and 16HPC018).

References

[1] J. Ejarque, R. M. Badia, L. Albertin, G. Aloisio, E. Baglione, Y. Becerra, S. Boschert, J. R. Berlin, A. D'Anca, D. Elia, F. Exertier, S. Fiore, J. Flich, A. Folch, S. J. Gibbons, N. Koldunov, F. Lordan, S. Lorito, F. Løvholt, J. Macías, F. Marozzo, A. Michelini, M. Monterrubio-Velasco, M. Pienkowska, J. de la Puente, A. Queralt, E. S. Quintana-Ortí, J. E. Rodríguez, F. Romano, R. Rossi, J. Rybicki, M. Kupczyk, J. Selva, D. Talia, R. Tonini, P. Trunfio, and M. Volpe, "Enabling dynamic and intelligent workflows for HPC, data analytics, and AI convergence," *Future Generation Computer Systems*, vol. 134, pp. 414–429, 2022.

[2] J. Rybicki, "Designing a Data Logistics and Model Deployment Service," ALLDATA 2020, The 6th International Conference on Big Data, Small Data, Linked Data and Open Data. IARIA, Feb. 2020, pp. 22–26.

[3] A. A. Ayllon, M. Salichos, M. K. Simon, and O. Keeble, "FTS3: New data movement service for WLCG," *Journal of Physics: Conference Series*, vol. 513, no. 3, p. 032081, jun 2014.

[4] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[5] A. Rajasekar, *iRODS primer: integrated rule-oriented data system*. Morgan & Claypool Publishers, 2010, vol. 12.

[6] I. Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.

[7] S. R. Wilkinson, G. Eisenhauer, A. J. Kapadia, K. Knight, J. Logan, P. Widener, and M. Wolf, "F... workflows: when parts of FAIR are missing," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, Oct. 2022.

[8] G. Accarino, D. Donno, F. Immorlano, D. Elia, and G. Aloisio, "An ensemble machine learning approach for tropical cyclone localization and tracking from ERA5 reanalysis data," *Earth and Space Science*, vol. 10, no. 11, p. e2023EA003106, 2023.

[9] D. Talia and P. Trunfio, "Urgent computing for protecting people from natural disasters," *Computer*, vol. 56, no. 04, pp. 131–134, apr 2023.

[10] "Topology and orchestration specification for cloud applications," OASIS, Standard, Nov. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[11] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, and J. Labarta, "PyCOMPSs: Parallel computational workflows in Python," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.

[12] J. Ejarque and R. M. Badia, "Automatizing the creation of specialized high-performance computing containers," *International Journal of High Performance Computing Applications*, vol. 37, no. 3, pp. 272–287, 2023.

[13] EUDAT. B2SHARE. [Online]. Available: https://www.eudat.eu/services/userdoc/b2share

[14] eFlows4HPC. Pipelines repository. [Online]. Available: https://github.com/eflows4hpc/dls-dags

[15] Apache Airflow. [Online]. Available: https://airflow.apache.org

[16] "ISO 15836-1:2017 – information and documentation – the Dublin Core metadata element set," International Organization for Standardization, Standard, May 2017.

[17] UNITY. [Online]. Available: https://unity-idm.eu/

[18] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6749

[19] HashiCorp. Vault. [Online]. Available: https://www.vaultproject.io

[20] I. Kuzminykh, B. Ghita, and S. Shiaeles, "Comparative analysis of cryptographic key management systems," in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer International Publishing, 2020, pp. 80–94.

[21] B. Hagemeier, "HDF Cloud – Helmholtz Data Federation Cloud Resources at the Jülich Supercomputing Centre," *Journal of large-scale research facilities*, vol. 5, p. A137, 2019.

[22] eFlows4HPC. Pillar iii workflow. [Online]. Available: https://github.com/eflows4hpc/workflow-registry/blob/main/Pillar_III/