Journal of Information and Data Management, 2026, 17:1, doi: 10.5753/jidm.2026.5723

Provenance Support for Containerized Workflow Analyses in High-Performance Computing Environments

Liliane Kunstmann [**PESC-COPPE-Universidade Federal do Rio de Janeiro** | ***lneves@cos.ufrj.br***]
Débora Pina [**PESC-COPPE-Universidade Federal do Rio de Janeiro** | ***dbpina@cos.ufrj.br***]
Daniel de Oliveira [**IC-Universidade Federal Fluminense** | ***danielcmo@ic.uff.br***]
Marta Mattoso [**PESC-COPPE-Universidade Federal do Rio de Janeiro** | ***marta@cos.ufrj.br***]

PESC-COPPE-UFRJ - Av Horácio Macedo 2030, Bloco H, sala 319 - Cidade Universitária, 21941-914

Received: 22 March 2025 • **Published:** 13 March 2026

Abstract Deploying scientific workflows in High-Performance Computing (HPC) environments presents several challenges due to variations in computational infrastructure, execution environments, and resource availability. Containers offer a way to ease workflow deployment and foster reproducibility. However, effective use of containers requires more than just access to container images. Understanding container provenance is essential, as it provides detailed information on image creation, configuration, and execution history, which is critical when deploying workflows across different architectures and container engines. Existing provenance support focuses on tracking container actions and standalone processes, but does not relate it to the provenance of workflows. To address this limitation, we represent container metadata as provenance and relate it to the provenance captured by the workflow execution. This approach enables workflow deployment with multiple container configurations in HPC environments while being compliant with the W3C-PROV standard for structured container provenance. The proposed model was evaluated in a real scientific machine-learning workflow. The evaluation assessed how provenance data can improve traceability, support workflow reproducibility, and facilitate containerized workflow analyses.

Keywords: Container, Provenance, Workflows, Machine Learning

1 Introduction

Scientific workflows are chained activities with a data flow between [de Oliveira *et al.*, 2019]. The workflow activities comprise multiple programs, software libraries, and software dependencies [Orzechowski *et al.*, 2018]. These workflows are usually deployed in various environments due to their resource requirements and scientific nature [Orzechowski *et al.*, 2018]. Their design and initial development are performed on personal computers, and later are migrated to High-Performance Computing (HPC) environments where they can be properly executed. So, their execution demands a deployment process to be performed multiple times in heterogeneous environments. Current machine learning experiments share the same profile of scientific workflow deployment.

Containerization can assist in deploying applications in multiple and heterogeneous environments [Merkel, 2014]. Containers are a lightweight form of virtualization that packages an application and its software dependencies into a single, self-contained unit [Merkel, 2014]. This executable unit is called a container image. Using containers to deploy scientific workflows requires a careful containerization design [Kunstmann *et al.*, 2024b]. This design includes deciding which container images to use for the workflow activities and how to group these images in containers, known as container composition. There are three alternatives to compose the workflow container. In the fine-grained composition, each workflow activity is containerized in independent containers. The other extreme is the coarse-grained composition that uses a single container to group all workflow activities. In the hybrid composition, workflow activities are grouped in a few containers to avoid one single container or managing the many containers of the fine-grained. This containerization design is implicit and yet needs to be known for reproducibility. Provenance data is often associated with workflow execution to add trust and reproducibility [Costa *et al.*, 2013]. The workflow provenance is not aware of the containerization design. It is important to make a workflow provenance container-aware to represent how the container images are chosen and grouped to workflow activities, with the necessary ports to access those images *etc*.

Current approaches to container provenance focus on capturing I/O operations and tracking container behavior through low-level function calls. They do not relate container provenance to workflow provenance traceability. Analyzing these provenance data is further complicated because related work does not comply with standard representations like W3C PROV [Moreau and Groth, 2013] and the Open Container Initiative (OCI) annotations[1].

We present a workflow provenance deployment approach for containerized workflow analyses to represent globally the traceability of the workflow execution. We argue that both workflow and container provenance are essential for the analysis and reproducibility of the workflow, as they are complementary. Our approach aims at providing workflow analysis like "*which container images were used for a specific workflow activity*" and traceability on the base images of the workflow execution path.

[1]`https://opencontainers.org/`

This manuscript extends the work presented in [Kunstmann *et al.*, 2024a] with new experiments and by providing a deeper evaluation of current provenance support for workflow traceability with HPC profiling and container metadata. We use `ProvDeploy` [Kunstmann *et al.*, 2022] to evaluate our container-aware workflow provenance approach. `ProvDeploy` is a framework that deploys containerized workflows in HPC environments, supporting different container compositions. We contribute to a provenance data model based on the W3C PROV and the OCI annotations to represent container descriptions, requirements, recipes, and files. We explore our container-aware workflow provenance with two real workflows, DenseED [Freitas *et al.*, 2021], a scientific machine-learning workflow, and Sciphy [de Oliveira *et al.*, 2012], a phylogenetic trees evaluation workflow. We use ProvDeploy to deploy them in HPC and evaluate provenance queries to explore traceability in containerized workflow analyses. We show how the provenance of multiple containers can help to analyze and reproduce the workflow.

The remainder of this manuscript is structured as follows. Section 2 presents container provenance-related work. Section 3 presents `ProvDeploy` and the provenance data model. Section 4 presents a provenance query evaluation and the query support on current approaches. Section 5 shows the use and evaluation of provenance queries through `ProvDeploy` in two real scientific workflows, and Section 6 concludes this manuscript.

2 Workflow Containerization and Current Provenance Support

This section presents challenges in containerizing workflows and container provenance. Current approaches for container provenance are limited to representing container actions and processes without relating them to workflow and its activities. We did not find related work for workflow provenance traceability that is container-aware and vice versa. Capturing and relating containers with workflow provenance for analysis is an open problem, particularly in HPC.

Provenance support for workflows is not new [Silva *et al.*, 2020]. However, many solutions that claim to support provenance do not represent the typical relationships that define the derivation paths for traceability [Schlegel and Sattler, 2023; Pina *et al.*, 2024] or cannot capture provenance in HPC. PROV-IO$^+$ [Han *et al.*, 2024] is an exception to capture workflow provenance for HPC environments. It shares similarities with `ProvDeploy` like using an extensible W3C PROV-compliant data model and providing relationships through prospective provenance (*p-prov*), where steps of the workflow are represented before execution like a recipe to be followed, in addition to retrospective provenance (*r-prov*), which is captured during execution [Freire *et al.*, 2008]. Despite a rich workflow provenance, PROV-IO$^+$ like current provenance systems is unaware of container provenance.

Containerizing applications is almost straightforward, whereas composing workflow activities into containers is challenging [Lampa *et al.*, 2019; Novella *et al.*, 2019]. The coarse-grained composition helps deployment; however, replacing and reusing workflow activities is easier with a fine-grained composition. Reproducing a workflow execution without container composition awareness can be challenging. In HPC workflows, there are often restrictions on image building [Priedhorsky *et al.*, 2021]. Containers are not isolated like virtual machines; they rely on the host OS to execute their isolated processes and can be affected by other containers, their configurations, and the execution environment [Straesser *et al.*, 2023; Wofford *et al.*, 2022].

All those issues increase workflow containerization challenges. Like scientific workflows, machine learning (ML) often adopts containers through ML studios and other cloud-hosted services that provide access to computing resources. In the ML context, users face challenges in container compositions and limited provenance support [Schlegel and Sattler, 2023; Pina *et al.*, 2024].

Capturing container provenance is addressed by a few approaches, with variations in what data is collected and how it is stored, depending on their goals. Most of these approaches [Shaffer *et al.*, 2023; Chen *et al.*, 2021; Abbas *et al.*, 2022; Ahmad *et al.*, 2020; Han *et al.*, 2024] automatically collect metadata from containers of single applications or microservices. However, this metadata does not relate to the application artifacts, has low-level traces, lacks workflow support, and is available only for *post-mortem* analysis, *i.e.* only after execution. Our analysis finds that current related work limits workflow traceability analyses and the reproducibility of container images. `ProvDeploy` aims at complementing current workflow provenance to make it container-aware.

We discuss and compare some of these approaches with `ProvDeploy` in Table 1. The column *Container & Workflow* specifies if the provenance captured can represent container provenance related to workflow. *Provenance Awareness level* details the level of container provenance representation if it represents single applications, microservices, or workflows. *Provenance graph* specifies if the approach allows the derivation of a provenance graph. *Data Model* describes whether provenance is represented following a standard such as W3C PROV or in an *ad-hoc* way. *Access Availability* indicates whether the provenance data are available for analysis at runtime or *post-mortem*. *Query Support* indicates the query support for analyzing provenance data.

Chen *et al.* [2021] discuss the challenges of sound and clear provenance tracking for microservices proposing `CLARION` a namespace and container-aware provenance tracking solution. Similarly, `ALASTOR`, proposed by Datta *et al.* [2022], enables tracking of suspicious events in serverless applications. `PACED` [Abbas *et al.*, 2022] is designed to detect container escape attacks through the isolation of cross-namespace events. Satapathy *et al.* [2023] discuss the lack of provenance data capture for microservice applications and propose `DisProTrack` for capturing provenance from microservices in an integrated way, handling parallel calls inherent to microservices.

Modi *et al.* [2023] discuss the challenges of capturing container provenance for standalone applications using container namespaces. They examine *post-mortem* container provenance from auditing tools like `PROV-CRT` and introduce

[2]Universal Provenance Graph

Table 1. Comparison of provenance support for containerized workflows.

Container Provenance Solution	Container & Workflow	Provenance Awareness Level	Provenance Graph	Data Model	Access Availability	Query Support
`CLARION` [Chen *et al.*, 2021]	No	Microservice	No	N/A	*Post-mortem*	N/A
`ALASTOR` [Datta *et al.*, 2022]	No	Microservice	Yes	*Ad-hoc*	*Post-mortem*	Provenance graph
`PACED` [Abbas *et al.*, 2022]	No	Microservice	No	N/A	Runtime	N/A
`DisProTrack` [Satapathy *et al.*, 2023]	No	Microservice	Yes	UPG [2]	*Post-mortem*	RegEx
[Modi *et al.*, 2023]	No	Single Application	Yes	*Ad-hoc*	*Post-mortem*	Hypergraph
[Wofford *et al.*, 2022]	No	Single Application	Yes	*Ad-hoc*	Runtime	SQLite
`PROV-CRT` [Ahmad *et al.*, 2020]	No	Single Application	Yes	W3C PROV	Runtime	Jupyter Interface
ContainerEnv [Olaya *et al.*, 2022]	Yes	Workflow	No	*Ad-hoc*	*Post-mortem*	Jupyter Interface
`ProvDeploy`	Yes	Workflow	Yes	W3C PROV OCI	Runtime	MonetDB

a hypergraph-based model for tracking provenance in single containerized applications. Their model is limited to *post-mortem* analysis of single applications.

Wofford *et al.* [2022] propose requirements for capturing the provenance of HPC applications and the issues related to hardware metadata capture. They design and implement a container-based provenance capture system that is limited to a single application.

`PROV-CRT` [Ahmad *et al.*, 2020] is a provenance module integrated into container runtimes such as LXC and Docker. It tracks and audits provenance during container creation and execution. `PROV-CRT` captures provenance at the granularity of system calls, which, although difficult to analyze, enables verification and validation of computations during container replay by comparing audited provenance data. This approach is limited to a single application.

Olaya *et al.* [2022] is the closest approach to `ProvDeploy`. They propose ContainerEnv, a tool that combines container data from Singularity with workflow data from activities in a record trail. This approach assumes a fine-grained composition to capture data from each containerized activity. They represent this record trail inside each data container separately. However, if the workflow adopts a coarse-grained composition, ContainerEnv can no longer capture provenance from workflow activities. Their approach provides a Jupyter interface to process this data from the containers to join the traces into a workflow graph. There is no independent provenance graph to be accessed by third-party tools. Since their provenance depends on Singularity containers, they rely on Singularity/Apptainer compatibility and availability. Their record trail is not based on W3C PROV relationships, which forces the user to learn a new representation, and it is only available for *post-mortem* analysis that also relies on data container availability.

Our approach, implemented in `ProvDeploy`, is in line with Wofford *et al.* [2022] and Canon [2020], highlighting the importance of documenting container characteristics for analysis, explanation, and reproducibility, given that containers can employ different drivers to execute the same task. `ProvDeploy` [Kunstmann *et al.*, 2022, 2024b] is a framework that eases the deployment of scientific workflows in HPC environments with container provenance support related to workflow provenance. `ProvDeploy` was first developed as a framework to support provenance data capture in scientific applications, as it allowed using provenance services through containers. In its initial version, `ProvDeploy` included a catalog that stored metadata on container images available for execution. This catalog was later expanded to provide more detailed information about the container images. When we extended `ProvDeploy` to execute workflows, we identified the need for container provenance that also registers the workflow.

`ProvDeploy` captures environment metadata in provenance traces, and its overhead stays between 1 and 3%. The performance evaluation is outside the scope of this paper and can be found in [Kunstmann, 2024]. In ProvDeploy, container provenance is currently tied to the model presented in Figure 1, but can be extended as it is W3C Prov compliant. The workflow provenance depends on the provenance service granularity designed by the workflow developer. If the tool relies on instrumentation, only user-relevant data is captured. Automatic capture requires post-processing for integrated analysis with the container provenance data model. To the best of the authors' knowledge, Olaya *et al.* [2022] and `ProvDeploy` are the only approaches that offer container provenance at the workflow level.

3 Container-aware Provenance with `ProvDeploy`

In this section, we present a comprehensive workflow provenance data model implemented in `ProvDeploy` to become container-aware. In `ProvDeploy`'s architecture [Kunstmann *et al.*, 2024b], it corresponds to the *Catalog*, *i.e.*, all metadata necessary for `ProvDeploy` to work. `ProvDeploy` allows using different provenance services, and this model was designed to extend the provenance captured by provenance services such as DfAnalyzer [Silva *et al.*, 2020], noWorkflow [Murta *et al.*, 2015], and CWLProv Khan *et al.* [2018], making it container-aware. Using the data model presented in Figure 1, we aim to represent container provenance and provide meaningful analysis with workflow provenance without tying the user to a specific provenance service. This data model is based on OCI and Common HPC Container Conformance Initiative [3], and relevant data presented in [Priedhorsky *et al.*, 2021; Canon, 2020; Straesser *et al.*, 2023; Wofford *et al.*, 2022]. Since OCI is a standard, container providers are compliant with it, making `ProvDeploy` container-aware independent of providers.

The model presented in Figure 1 follows the W3C PROV-DM recommendation, where provenance is represented in

[3]https://github.com/container-in-hpc/container-hpc-conformance

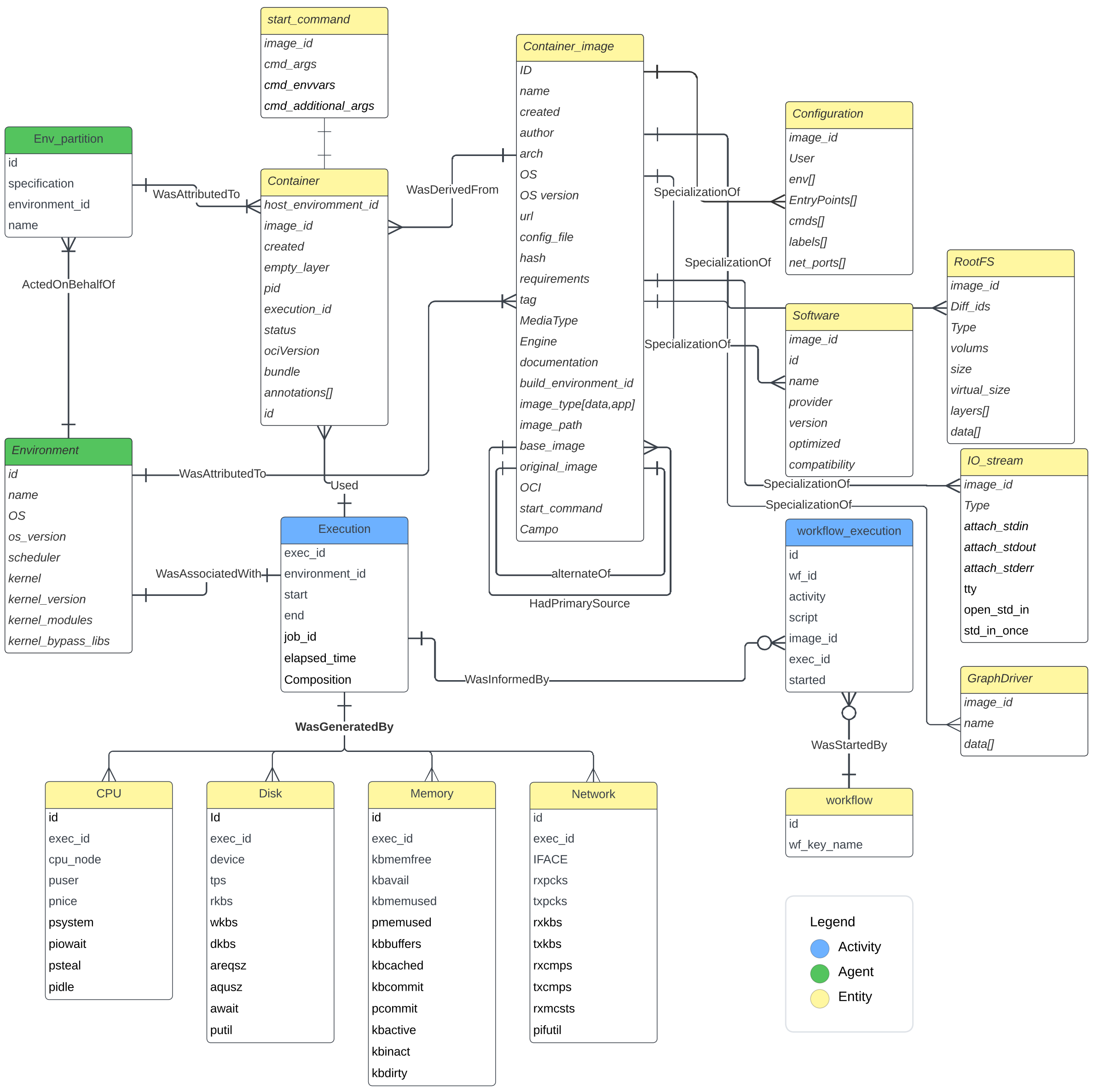

Figure 1. PROV-DM diagram of container aware provenance data model.

terms of entities (the data objects), activities (data transformations), and agents (users and systems) with their W3C PROV-DM relationships. In the proposed model, the yellow classes represent Entities, the green classes represent Agents, and the blue classes represent Activities. The colors of our class diagram are represented as stereotypes in compliance with the W3C PROV classes. This data model aims to provide information about container image origins and features, improving its findability and reliability for the rebuild process. `ProvDeploy` implements this data model using MonetDB, a column-oriented RDBMS that, as a DBMS solution, allows for querying provenance graphs and easy integration with other libraries for analysis and visualization.

The entity that allows the connection between container and workflow provenance is the entity *workflow* that has as an attribute workflow name (*wf_key_name*) that identifies it on the provenance database of the provenance capture service (*e.g., dataflow_tag* in DfAnalyzer or *hashID* in noWorkflow). Using entity *workflow*, we can relate the container provenance captured by `ProvDeploy` with the workflow provenance to be captured by a provenance system that is not container-aware. The activity *execution* associates the container composition, which can be coarse-grained (one container for the whole workflow), fine-grained (one container for each workflow activity), or hybrid (multiple, but not all, activities executed by the same container). The activity *execution* also stores *job_id* from schedulers in HPC environments, which can be used in tuning and debugging.

The entity *container_image* represents the container image details, allowing it to be rebuilt. A container image is specialized by the entities *configuration*, *rootFS* (required volumes), *software* that compose the image, *graph_driver*, and *io_stream* data. This information is stored in the container image configuration file or recipe (*config_file*) and the container image. However, there is no guarantee that recompiling the container image using the configuration file will produce an identical image [Canon, 2020] or that the configuration file is up-to-date with the image.

Additionally, container images are linked to the build environment (*build_environment_id*) in which they were created. This build environment defines the container's compatible kernel architectures. A container image can be the source for other container images, either as a primary source (*base_image*) for a new container image or by generating alternative versions with different container engines (*original_image*) through container image conversion. For instance, a container image built on Docker might be converted to Singularity. In such cases, it is no longer associated with the *config_file* but references the original container image.

The entity *container* describes the isolated process that runs using a container image and specifies the environment in which it is executed. A container relies on a container image to be executed. One single image can be used to execute multiple containers. This entity is used during *execution* activity and documents what occurred during workflow execution. Both *execution* and *workflow_execution* activities capture the expected behavior of the containerized workflow, referred to as prospective provenance. The *container* entity records the activities that were carried out, referred to as retrospective provenance. If an activity fails to start, it is not recorded. The entity *container* is specialized by the entity *start_command* that stores the commands, arguments, and environment variables used at container start because the user can start a container image with arguments that are different from what is stored in the *container_image*.

The *environment* agent describes the environment associated with an *execution*, including its kernel architecture and scheduler. HPC and cloud environments can have heterogeneous resources, so the *env_partition* was designed to detail the characteristics of the specific environment used. This partition contains the hardware specifications of the environment and is where the containers are executed.

The *execution* activity, with the *workflow_execution* activity, describes the containerized execution. This includes the composition of containers, workflow activities, and the containers they used, along with the expected execution commands and parameters needed to start each activity. These activities enhance user execution management, especially for provenance services, like DfAnalyzer, that do not natively distinguish multiple concurrent workflow executions.

Lastly, the entities *CPU*, *Disk*, *Memory*, and *Network* enable the association of profiling data with workflow execution, providing deeper insights into different container compositions. Although OCI annotations could also represent this information, we based our model on the Sysstat library, a popular tool in the HPC domain. This decision was driven by the fact that the profiling of OCI annotations is primarily tailored for Docker images, whereas Sysstat offers broader applicability. By linking profiling data to containers, `ProvDeploy` helps analyze resource utilization, particularly when resource constraints are applied to containers.

4 Provenance Query Evaluation

`ProvDeploy` aims to support a comprehensive analysis of the workflow execution in a containerized environment. Table 2 showcases a series of queries the container-aware provenance data model can address when connected to the workflow provenance model. These queries are inspired by the First Provenance Challenge[4] and are categorized into information concerning container reuse, workflow reproducibility, and insights for workflow traceability analysis. In this section, we discuss the importance of a container-aware provenance model to answer workflow queries by evaluating `ProvDeploy` and current approaches.

Query Q1 retrieves data that is usually available only during execution, most importantly, the register of the containers that were effectively used for workflow execution since registering only the images provides limited information from execution. This query can be answered by joining entities *execution, workflow_execution, container_image* and *env_partition* and setting the target workflow.

Query Q2 presents information about container images used for workflow execution, and that can help users in future workflow executions with a similar software stack. Using the associated *job_id*, we can also find more useful information about the batch job executed, such as possible tasks that were

[4]https://openprovenance.org/provenance-challenge/FirstProvenanceChallenge.html

Table 2. Container and Workflow Provenance Queries inspired by the First Provenance Challenge

#	Query	Type
Q1	Retrieve job_id, containers, and host environments involved in a workflow execution.	Reuse Reproducibility
Q2	Retrieve the container images associated with the workflow with the *job_id* and activities they performed.	Reuse Reproducibility
Q3	Retrieve all workflows that were executed with a given container image.	Reuse
Q4	Retrieve all workflow activities that were run with a given container image	Reuse
Q5	Retrieve all images that are created by a specific user.	Reuse
Q6	Retrieve all workflows where a given variable satisfied a certain condition and was deployed with a specific composition.	Analysis Reproducibility
Q7	A user has run the workflow with different compositions and in distinct environments, increasing the number of containers. Retrieve the differences in the compositions in the distinct environments.	Analysis
Q8	Which environment (host and container) executed the workflow with its best results?	Analysis Reproducibility

executed before or after the workflow and are not modeled by workflow provenance. Query Q2 joins entities *workflow, workflow_execution, container_image* and *workflow*, setting the target workflow using attribute *wf_key_name*.

Query Q3 shows the possibilities of reusing container images across multiple workflows with common software requirements, and Q4 operates similarly to Q3 but provides more detailed information on the activities executed by the container images. Query Q4 joins *container_image, workflow*, and *workflow_execution* entities.

Query Q5 explores the *container_image* entity attributes. This query helps to identify shared base images and execution architectures (*arch*). If a user plans to execute any of these container images in a different environment, the first step would be to check if the base image has a compatible version with the target kernel architecture.

Queries Q6, Q7, and Q8 are more complex queries and integrate container and workflow provenance. These queries join container entities *execution, workflow_execution* with the corresponding entities of the workflow provenance model. Once container provenance is joined with the workflow provenance, the user can explore queries using an interval starting from the *execution_start* and ending on the added *elapsed_time*. Those queries help to understand the impact of the container composition on workflow performance in different environments.

Queries Q1, Q2, and Q7 cannot be answered using only workflow provenance. While developing a scientific workflow, users often explore multiple algorithms and software libraries, resulting in numerous containers and container images combined with different sources and environments. In current approaches, containers are not linked with their associated container images, environments, libraries, or workflows. This information may be scattered across logs, files, and scripts. Without a predefined structure to represent this information, analyzing it becomes increasingly difficult and may not be possible, even though it is essential for reproducibility.

Queries Q3, Q4, Q5, and Q7 can improve container image reuse and reduce the execution and storage of unnecessary container images to execute a workflow, since a container image can execute multiple workflow activities and generate new container images. In addition, with the data model of `ProvDeploy`, we can track a container image derivation path to rebuild it.

These queries are used to discuss the state of the art solutions for analyzing workflow provenance in containerized workflows. A summarization of the results is shown in Table 3. Unlike `ProvDeploy`, these solutions record their data using logs and unstructured metadata, which are typically captured per container, resulting in scattered and disconnected data sources. They require different amounts of post-processing to relate data to address the queries outlined.

Among the queries analyzed, Q4 is the one answered by all approaches. However, some approaches, including `CLARION`, `ALASTOR`, and `DisProTrack`, do not store activity or workflow data explicitly. Instead, this information can only be inferred from the container data that these approaches record. On the other hand, `ProvDeploy` captures both activity and workflow provenance directly, along with all other relevant provenance data. By doing so, it ensures that these data are explicitly available and can be queried using SQL.

Table 3. Query support for the approaches presented in section 2.

Container Provenance Solution	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
`CLARION`	-	-	-	Yes	-	-	-	-
`ALASTOR`	-	-	-	Yes	-	-	-	-
`PACED`	Yes	-	-	Yes	-	-	-	-
`DisProTrack`	-	-	-	Yes	-	-	-	-
Modi *et al.* [2023]	-	Yes	-	Yes	-	-	-	-
Wofford *et al.* [2022]	Yes	-	-	Yes	-	-	-	-
`PROV-CRT`	-	-	-	Yes	Yes	-	-	-
ContainerEnv	-	-	Yes	Yes	-	Yes	-	-
`ProvDeploy`	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

`PACED` and Wofford *et al.* [2022] are approaches that emphasize the importance of the execution environment and, as a result, can incorporate environment-related data into their tracking. This capability enables these approaches to provide answers to query Q1. However, they do not establish explicit records or associations between the executed workflows and the container images. Similarly, Modi *et al.* [2023] is the only approach that can partially address query Q2. Although it supports querying certain aspects of workflow execution, it does not manage environment-specific data, such as *job_id*, which limits its ability to fully respond to the query.

`PROV-CRT`, as one of the first approaches for data collection in container engines, stands out as the only approach capable of answering Q5, as the data capture primarily occurs during the generation of container images. Lastly, the approach by Olaya *et al.* [2022] stands out by addressing queries that require workflow provenance associated with containers. However, their record trail depends on data available from Singularity, which limits its query support and the compatible kernel architectures to execute workflows.

The model implemented in `ProvDeploy` supports all the presented queries by modeling the relationships between the different metadata associated with containerized execution. Its adoption of W3C-PROV makes it extensible and facilitates the integration of plugins and interoperability. By adopting OCI annotations, we ensure that the model is not tied to a specific container engine. Additionally, we assume the existence of a provenance service that captures workflow data, which becomes container-aware. Without `ProvDeploy`, the user would have to browse and post-process HPC metrics files and container metadata from separate files and structure data to analyze the two together.

5 Evaluation of the Container Provenance Model

`ProvDeploy` was used to deploy DenseED and SciPhy workflows in HPC executions. In this section, we evaluate the container provenance model to support queries during and after the execution of the workflows.

5.1 Environment Setup

DenseED and SciPhy were deployed with `ProvDeploy` in Santos Dumont (SDumont)[5] Supercomputer. SDumont has

[5]`https://sdumont.lncc.br/`

an installed processing capacity of around 5.1 Petaflop/s (5.1 x 1015 float-point operations per second), presenting a hybrid configuration of computational nodes in terms of the available parallel processing architecture. For DenseED we used two different computational nodes: a CPU node and a GPU node. The CPU node has two CPUs with an Intel Xeon E5-2695v2 Ivy Bridge 2.4GHZ processor, 24 cores (12 per CPU), and 64GB DDR3 RAM. The GPU node is part of SDumont's expanded partition BullSequana X that has two CPUs with an Intel Xeon Skylake 2.1 GHz processor, 48 cores (24 per CPU), 384GB RAM, and four GPUs NVIDIA Volta V100. For SciPhy we used only the CPU node. In both nodes, we used Linux RedHat 7.6 operating system, and Singularity 3.8 for the container engine. We have used a personal computer (CPU Intel Core i7-10700K processor, 3.80GHz, 16 cores, and 15,5 GiB) to execute DenseED and generate the container images to be deployed in SDumont. In both experiments, the workflows already used provenance from DfAnalyzer [Silva *et al.*, 2020] due to its W3C Prov compliance that demands no postprocessing, but `ProvDeploy` can deploy other provenance services.

5.2 DenseED Workflow

DenseED is a scientific Machine Learning workflow proposed by Freitas *et al.* [2021], based on a Physics-guided Convolutional Neural Network (PCNN). DenseED uses the PCNN as a surrogate model for Reverse Time Migration (RTM) calculations to quantify uncertainties in seismic analysis. Solving RTM equations is a CPU-intensive and time-consuming task. To quantify uncertainties, RTM has to be calculated many times. The architecture, presented in Figure 2, shows convolutional layers with Encoder-Decoder dense blocks to manage high-dimensional inputs and outputs.

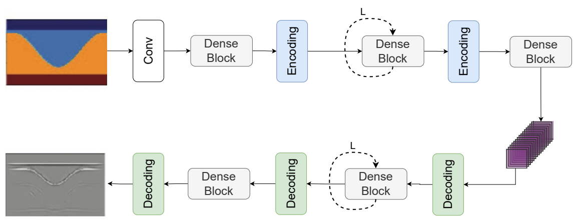

Figure 2. DenseED architecture [Freitas *et al.*, 2021].

The workflow provenance of DenseED was previously modeled to capture hyperparameters and training/test metrics like R^2 and RMSE. Specifically, K determines the DenseED growth rate, and L the number of layers in the dense block, both stored by the model. The system relies on TensorFlow, which requires specific compatibility with Python, the C compiler, Bazel, CUDA, and cuDNN when using GPUs. TensorFlow is available through containers, simplifying this process by enabling GPU usage with specific flags. Users can explore alternative container images from public registries like NGC (NVIDIA GPU Cloud), Docker, and Binder if a workflow's container image is incompatible with the available GPU or the kernel architecture.

Additionally, containers ease the exploration of heterogeneous CPU hardware with minimal effort. Public registries like NGC provide users with optimized TensorFlow container images for different NVIDIA GPU series, allowing users to explore multiple TensorFlow versions and features on different GPU devices. This exploration is not simple, and container provenance helps to identify the best version for reusing the container images, in addition to tracking performance changes. To ensure the reproducibility of these executions over time, container provenance provides relevant information about the original images and, if necessary, helps identify suitable replacement images or rebuild new ones.

To find the best container composition for DenseED, the workflow was executed with three containerization compositions as shown in Table 4. Querying container provenance helps to choose the best composition and fine-tune containerization decisions according to the target execution environment. `ProvDeploy` supports all queries listed in Table 2, and we discuss some of these queries with DenseED executions.

Table 4. Containerization compositions with DenseED.

Compositions	Description
Coarse-grained	A single container encompassing all dependencies for provenance data capture and running DenseED.
Partial modular	Two containers: A container with DenseED and a second container with the DBMS and provenance data, and the provenance service.
Provenance modular	Three containers: the first with DenseED, the second with the provenance service, and the third with the DBMS and provenance data.

Table 5 shows how query Q5 explores container images created by the user 'Liliane' from various workflows. The images tagged *dfanalyzer*, *py-readseq-modelGenerator*, and *java-readseq-modelGenerator* are all based on a Java image that Liliane did not create and share the amd64 kernel architecture. When reproducing this workflow with any of these container images, it is beneficial to check if the base image has a version compatible with the new architecture.

DenseED explores variations of its CNN architecture (convolutional, discriminator/generator, conditional, *etc.*), training metrics, and parallelization techniques. It also explores multiple execution environments with different TensorFlow and CUDA versions (*i.e.*, multiple containers), and these changes impact the training execution time.

Analyzing these different details over time becomes increasingly complex, leading to a trial-and-error process. Without provenance, it is difficult to reproduce the same results or match DenseED with its compatible containers. For instance, even using a single environment like SDumont, users face this challenge since there are multiple GPU partitions with distinct devices available (*e.g.*, K40 and V100). These devices require different combinations of cuDNN, CUDA, TensorFlow, and other software, resulting in distinct container images. Also, these combinations are affected by the image origins (*e.g.*, Docker Hub, NCG, *etc.*), which will ignore the GPUs if deployed in an incompatible partition.

Using `ProvDeploy`, this information is captured and available through the proposed data model implemented at the workflow provenance database, filling gaps in the execution information, as required in query Q7. Table 6 shows the results of query Q7 for each environment with different container compositions. Query Q7 requires joining the entities *execution*, *env_partition*, *workflow_execution*, and *workflow*

Table 5. Q5 - Container images generated by user 'Liliane';

id	author	arch	vendor	image tags	description	Env name	image type	Base Image
8	Liliane	amd64	Singularity	denseed	DenseED with DfAnalyzer and MonetDB	liliane-imac20	application	tensorflow
9	Liliane	amd64	Singularity	provData	DfAnalyzer and MonetDB	liliane-imac20	provenance	dfanalyzer
1	Liliane	amd64	Docker	icc	Intel compiler with dfa-lib-cpp	liliane-ubuntu	application	N/A
3	Liliane	amd64	Singularity	dfanalyzer	Container of dfanalyzer	liliane-ubuntu	provCollector	java
4	Liliane	amd64	Singularity	monetdb	Container of provdeploy database	liliane-imac20	database	N/A
5	Liliane	amd64	Singularity	py-readseq-modelGenerator	ReadSeq, python2, java, raxml, dfa-lib-python with telemetry, and psutils for python applications	liliane-iMac20	application	java
6	Liliane	N/A	N/A	java-readseq-modelGenerator	ReadSeq, python2, java, raxml, dfa-lib-python with telemetry, and psutils for Java applications	liliane-iMac20	application	java

with those from the DenseED provenance data model, including hyperparameters and metrics within a constrained interval from the *execution* entity. We present the executions of container compositions with the shortest elapsed time in each environment and the lowest R^2 for the DenseED execution.

Table 6. Q7 - Execution of container compositions with the smallest elapsed time per environment and the best R^2.

job_id	Composition	Elapsed time(m)	Env Name	R^2
10898072	Partial modular	4.01	sequana_gpu	0.9979
10898073	Provenance modular	4.10	sequana_gpu	0.9978
10898075	Coarse-grained	4.19	sequana_gpu	0.9975
10905992	Provenance Modular	20.55	cpu	0.9976
10901804	Coarse-grained	21.10	cpu	0.9978
10900921	Partial modular	21.32	cpu	0.9976
N/A	Coarse-grained	32.78	liliane-iMac20-1	0.9975
N/A	Partial modular	33.53	liliane-iMac20-1	0.9977

In Q7, the differences in R^2 values are small, but as expected, the elapsed time increases according to the type of environment. The workflow container composition also varies with the environment specification. In resource-limited scenarios such as the 'liliane-iMac20-1', a personal computer, the coarse-grained composition was the best considering the elapsed time. The coarse-grained composition involves a single container image, allowing it to expand and freely use available resources. In resource-limited scenarios, other compositions tend to experience resource competition between containers. In addition, they require extra time to start and stop containers, which accumulates over time, increasing the overall elapsed time. The difference in compositions for the same environment was less than two minutes, but in more complex workflows, this difference may increase.

Query Q8, shown in Table 7, identifies the best result in DenseED with the highest R^2 and the lowest RMSE.

Table 7. Q8 - Environment with the best RMSE.

job_id	Env Name	Image tag	RMSE	R^2	K	L
10898072	sequana_gpu	tensorflow, dfanalyzer	0.00093	0.9979	32	9
10897983	sequana_gpu	denseed	0.00103	0.9976	16	9
10898090	sequana_gpu	tensorflow, provdata	0.00111	0.9975	24	8

Q8 details the images associated with each composition and execution, which helps to reproduce the experiment. Q8 joins the entities *execution*, *env_partition*, *workflow_execution*, *container_image*, and *workflow* with those of DenseED provenance, such as hyperparameters and metrics within a restricted interval of DenseED training. Considering that DenseED was executed with various compositions and on different hosts, container provenance is crucial. Without it, the user would only know the values of K and L that led DenseED to the presented R^2 and RMSE. The lack of data on the image tags and the execution environment makes it much harder to reproduce the execution.

5.3 SciPhy Workflow

SciPhy [de Oliveira *et al.*, 2012] is a workflow that generates maximum-likelihood phylogenetic trees. SciPhy works with amino acid sequences and other types of biological sequences. SciPhy (Figure 3) is a script with four main activities: (i) multiple sequence alignment (MSA); (ii) alignment format conversion; (iii) selection of the best evolutionary model; and (iv) phylogenetic tree construction. Several alternative programs can be used for the MSA activity. Figure 3 illustrates six MSA alternatives (variants). The alignment conversion activity is optional and runs the ReadSeq tool. The final activity represents the construction of phylogenetic trees. The alternative programs for the Phylogenetic Tree Construction activity are RAxML or MrBayes.

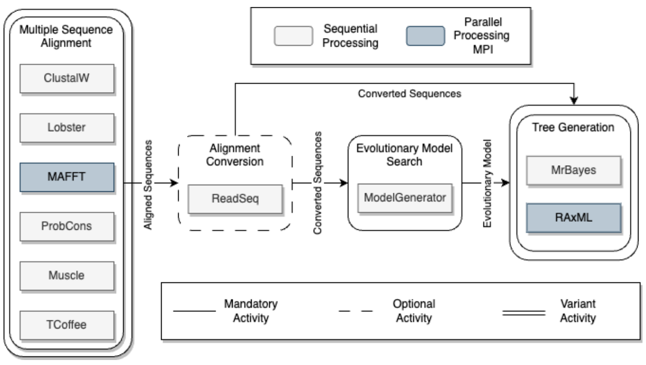

Figure 3. SciPhy workflow

The workflow provenance of SciPhy includes input parameters and files, timestamps, and the chosen algorithms. Using container provenance with SciPhy helps reuse this workflow or its activities and also improves its reproducibility. SciPhy was deployed with four containerization compositions ranging from a single container to a fine-grained composition of six containers: one container for each of the four SciPhy activities and two for provenance services and data.

Table 8 shows the result of Query Q2 with some of the container images used to execute the four compositions. We can spot the images used for fine-grained (*exec_id* = 845), coarse-grained (*exec_id* = 784), and hybrid executions. Q2 helps to identify and later reuse or replace those images, and also add the environment used for the execution. Q2 joins entities *workflow, workflow_execution, execution, container,* *and container_image*. The fine-grained composition suits exploring the variabilities of SciPhy activities. Attributes like *puser, tps, kbmemused, elapsed_time* and *wf_id* of entities *cpu, memory, disk, workflow* of Fig. 1 are used to compare the performance of the four alternative compositions. They help to check whether the overhead of multiple containers pays off the benefits of replacing activities.

Table 8. Q2 - SciPhy container compositions with container images and their activities.

image tag	exec_id	Activity
mafft	845	Mafft Execution
raxml	845	RaXML Execution
readseq	845	ReadSeq execution
model_generator	845	ModelGenerator
java	845	Provenance service
monetdb	845	Provenance persistence
java_monet	844	Provenance service, Provenance persistence
sciphy	844	SciPhy Workflow
py-readsed-modelgenerator	749	ReadSeq and ModelGenerator
coarse_sciphy	784	SciPhy workflow

Table 9 shows the result of Q1 applied to SciPhy when executed with the fine-grained composition, presenting the containers used with their scripts. The images of the containers for ModelGenerator and ReadSeq are both based on the same Java image, which can improve container start-up time and is important to trace container origins when necessary. Q1 requires entity *container*, and activities *execution*, and *workflow_execution*.

Table 9. Q1 - Scripts and containers used in a fine-grained execution of SciPhy at SDumont.

Container	script
./maft_latest.sif	mafft –quiet –retree 2 –maxiterate 1000 ...
./raxml.sif	raxmlHPC -s phylip -n phylip _raxml_tree1.singleTree -c ...
./readseq.sif	readseq -all -f=12 path_mafft > path_mafft_phy”
./modelgenerator.sif	path_aln 6
./java.sif	java -jar target/DfAnalyzer-1.0.jar
./monetdb.sif	-

6 Conclusion

Using containers to execute workflows in HPC environments can improve the portability and reproducibility of workflows. However, while containers enhance portability, they lack provenance data to assess reproducibility. In this paper, we explored container provenance aspects and introduced a container-aware provenance data model that extends an existing workflow provenance data model. Our goal is to provide container-aware provenance to improve workflow data analysis in HPC environments and improve trust in the use of containers. We evaluated this model using queries to help analyze and reproduce the workflow.

The presented model is implemented in `ProvDeploy`. For its provenance to be successfully integrated with provenance from other services, certain requirements must be met by those services. For example, they must have a key attribute that identifies the workflow, which is necessary for integrating with the container-aware provenance data model. The model also assumes a single user for each workflow activity, and we plan to explore the challenges of multi-user workflows. Despite such limitations, the presented model represents a first step towards capturing container provenance associated with workflow provenance.

Acknowledgements

We acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing SDumont HPC resources.

Funding

This paper was funded in part by CNPq, FAPERJ, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Authors’ Contributions

Daniel de Oliveira, Débora Pina, Liliane Kunstmann, and Marta Mattoso contributed to the conception and writing of this study. Liliane Kunstmann, the primary contributor, also conducted the experiments. All authors reviewed and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The code repository of `ProvDeploy` with its documentation, step-by-step guide and requirements can be accessed at `https://github.com/nevesLiliane/provDeploy`.

References

Abbas, M., Khan, S., Monum, A., Zaffar, F., *et al.* (2022). Paced: Provenance-based automated container escape detection. In *2022 IEEE IC2E*, pages 261–272. IEEE.

Ahmad, R., Nakamura, Y., Manne, N. N., and Malik, T. (2020). `PROV-CRT`: Provenance support for container runtimes. In *12th International Workshop on Theory and Practice of Provenance (TaPP 2020)*.

Canon, R. S. (2020). The role of containers in reproducibility. In *2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pages 19–25. IEEE.

Chen, X., Irshad, H., Chen, Y., Gehani, A., and Yegneswaran, V. (2021). `CLARION`: Sound and clear provenance tracking for microservice deployments. In *30th USENIX Security*, pages 3989–4006.

Costa, F., Silva, V., de Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E. S., Dias, J., and Mattoso, M. (2013). Capturing and querying workflow runtime provenance with PROV:

a practical approach. In Guerrini, G., editor, *EDBT/ICDT '13*, pages 282–289. DOI: 10.1145/2457317.2457365.

Datta, P., Polinsky, I., Inam, M. A., Bates, A., and Enck, W. (2022). ALASTOR: Reconstructing the provenance of serverless intrusions. In *31st USENIX Security*, pages 2443–2460.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. (2012). A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552. DOI: 10.1007/s10723-012-9227-2.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers. DOI: 10.2200/S00915ED1V01Y201904DTM060.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21.

Freitas, R. S., Barbosa, C. H., Guerra, G. M., Coutinho, A. L., and Rochinha, F. A. (2021). An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty. *Computational Geosciences*, 25:1229–1250.

Han, R., Zheng, M., Byna, S., Tang, H., Dong, B., Dai, D., Chen, Y., Kim, D., Hassoun, J., and Thorsley, D. (2024). PROV-IO$^+$: A cross-platform provenance framework for scientific data on hpc systems. *IEEE Transactions on Parallel and Distributed Systems*.

Khan, F. Z., Soiland-Reyes, S., Crusoe, M. R., Lonie, A., and Sinnott, R. (2018). Cwlprov-interoperable retrospective provenance capture and its challenges. In *International Provenance and Annotation Workshop (IPAW) 2018*.

Kunstmann, L., Pina, D., de Oliveira, D., and Mattoso, M. (2024a). Scientific workflow deployment: Container provenance in high-performance computing. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 457–470. SBC.

Kunstmann, L., Pina, D., de Oliveira, D., and Mattoso, M. (2024b). ProvDeploy: Provenance-oriented containerization of high performance computing scientific workflows. *arXiv preprint arXiv:2403.15324*.

Kunstmann, L., Pina, D., de Oliveira, L. S., de Oliveira, D., and Mattoso, M. (2022). ProvDeploy: Explorando alternativas de conteinerização com proveniência para aplicações científicas com pad. In *Anais do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 49–60. SBC.

Kunstmann, L. N. d. O. (2024). *APOIO À IMPLANTAÇÃO DE WORKFLOWS CONTEINERIZADOS*. PhD thesis, Universidade Federal do Rio de Janeiro.

Lampa, S., Dahlö, M., Alvarsson, J., and Spjuth, O. (2019). Scipipe: A workflow library for agile development of complex and dynamic bioinformatics pipelines. *GigaScience*, 8(5):giz044.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2.

Modi, A., Reyad, M., Malik, T., and Gehani, A. (2023). Querying container provenance. In *Companion Proceedings of the ACM Web Conference 2023*, pages 1564–1567.

Moreau, L. and Groth, P. (2013). Provenance: an introduction to prov. *Synthesis lectures on the semantic web: theory and technology*, 3(4):1–129.

Murta, L., Braganholo, V., Chirigati, F., Koop, D., and Freire, J. (2015). noworkflow: capturing and analyzing provenance of scripts. In *IPAW 2014*, pages 71–83. Springer.

Novella, J. A., Emami Khoonsari, P., *et al.* (2019). Container-based bioinformatics with pachyderm. *Bioinformatics*, 35(5):839–846.

Olaya, P., Kennedy, D., *et al.* (2022). Building trust in earth science findings through data traceability and results explainability. *IEEE TPDS*, 34(2):704–717.

Orzechowski, M., Balis, B., Pawlik, K., Pawlik, M., and Malawski, M. (2018). Transparent deployment of scientific workflows across clouds-kubernetes approach. In *2018 IEEE/ACM UCC Companion*, pages 9–10. IEEE.

Pina, D., Chapman, A., Kunstmann, L., de Oliveira, D., and Mattoso, M. (2024). DLProv: A data-centric support for deep learning workflow analyses. In *Companion of the 2024 ACM SIGMOD/PODS, Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning.*, DEEM '24, pages 77–85. ACM. DOI: 10.1145/3650203.3663337.

Priedhorsky, R., Canon, R. S., Randles, T., and Younge, A. J. (2021). Minimizing privilege for building hpc containers. In *IEEE/ACM SC*, pages 1–14.

Satapathy, U., Thakur, R., Chattopadhyay, S., and Chakraborty, S. (2023). Disprotrack: Distributed provenance tracking over serverless applications. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE.

Schlegel, M. and Sattler, K.-U. (2023). Management of machine learning lifecycle artifacts: A survey. *SIGMOD Rec.*, 51(4):18–35. DOI: 10.1145/3582302.3582306.

Shaffer, T., Phung, T. S., Chard, K., and Thain, D. (2023). Landlord: Coordinating dynamic software environments to reduce container sprawl. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1376–1389.

Silva, V., Campos, V., Guedes, T., Camata, J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. (2020). Dfanalyzer: runtime dataflow analysis tool for computational science and engineering applications. *SoftwareX*, 12:100592.

Straesser, M., Bauer, A., Leppich, R., Herbst, N., Chard, K., Foster, I., and Kounev, S. (2023). An empirical study of container image configurations and their impact on start times. In *2023 IEEE/ACM 23rd CCGrid*, pages 94–105. IEEE.

Wofford, Q., Hurd, J., Greenberg, H., Bridges, P. G., and Ahrens, J. (2022). Complete provenance for application experiments with containers and hardware interface metadata. In *2022 IEEE/ACM CANOPIE-HPC*, pages 12–24. IEEE.