RESEARCH ARTICLE

REVISED Towards a sustainable astronomical data infrastructure: Optimising linking data from the Rucio datalake to the users areas within the SKA Regional Centres Network

[version 2; peer review: 2 approved with reservations]

Manuel Parra-Royón[1], Julián Garrido-Sánchez[1], Susana Sánchez-Expósito[1], Laura Darriba-Pol[1], Jesús Sánchez-Castañeda[1], M. Ángeles Mendoza[1], Jeremy Coles[2], Sean McConkey[2], Rohini Joshi[3], Rob Barnsley[4], Jesús Salgado[4], Lourdes Verdes-Montenegro[1]

[1]Extragalactic Astronomy, Instituto de Astrofisica de Andalucia, Granada, Andalusia, 18008, Spain
[2]Battcock Centre for Experimental Astrophysics, University of Cambridge Department of Applied Mathematics and Theoretical Physics, Cambridge, England, CB3 0HE, UK
[3]Institute for Data Science, Hochschule fur Technik FHNW, Windisch, Aargau, 5210, Switzerland
[4]SKA Organisation, Macclesfield, England, SK11 9FT, UK

v2 **First published:** 09 Jan 2026, **6**:18
https://doi.org/10.12688/openreseurope.22118.1
Latest published: 20 Apr 2026, **6**:18
https://doi.org/10.12688/openreseurope.22118.2

Abstract

The distributed architecture of the SKA Regional Centre Network (SRCNet) aims to provide scientific communities worldwide with efficient computational and storage resources to exploit the massive data volumes produced by the SKA Observatory (SKAO). Given the amount of SKAO data, traditional data management paradigms — where data is transferred to computational resources— are no longer feasible. Instead, computational workflows must increasingly be relocated closer to data storage locations, emphasizing efficient data access strategies and avoiding unnecessary duplication or redundancy. In this context, we present PrepareData, a modular and extensible data delivery service developed within SRCNet prototyping activities. Our proposal for this service addresses the critical challenge of redundant data transfers and duplication at both node and user levels by enabling seamless delivery of requested datasets from local Rucio Storage Elements (RSEs) directly into users' working environments. PrepareData operates as a local service within each SRCNet node and it is integrated into a broader ecosystem of

Open Peer Review

Approval Status ? ?

	1	2
version 2 (revision) 20 Apr 2026		
version 1 09 Jan 2026	? view	? view

1. **Alexandra Theodoropoulou**, University of Patras, Patras, Greece

2. **Rana Veer Samara Sihman Bharattej Rupavath**, National Louis University, Chicago, USA

Any reports and responses or comments on the article can be found at the end of the article.

federated services. Specifically, we designed and evaluated two distinct yet complementary implementations to avoid unnecessary data duplication and to enable a dynamic data bridge between the RSEs and the user storage areas, through: (1) a filesystem-based solution leveraging CephFS, which uses shared filesystem mount points and bind mounts to ensure consistent and immediate data availability of the data across computational nodes, and (2) a Kubernetes model using Persistent Volumes and Persistent Volume Claims, dynamically injecting data into a user's areas. To tackle this work we detail the architectural design and development, the technical implementation, the integration of both solutions with science enabling tools, such as JupyterHub, CARTA or virtually any application, and finally we provide a performance evaluation. This contribution provides a scalable and sustainable blueprint for data delivery in federated scientific infrastructures, supporting the broader goals of green computing and efficient resource utilisation.

Plain Language Summary

Modern scientific projects such as the Square Kilometre Array Observatory (SKAO) produce extremely large amounts of data — far more than traditional research systems have handled before. Scientists around the world need efficient ways to access and work with these vast datasets without unnecessary delays or wasteful copying of information. In the SKA Regional Centre Network (SRCNet), data are stored in distributed storage systems. However, these storage systems are not directly visible to the software tools scientists use for analysis, such as interactive notebooks, visualisation platforms, or specialised processing services. To make the data usable, it must be moved or linked into the scientists' working environments. Our research focuses on a service called *PrepareData*, designed to manage this delivery process in a way that is both fast and resource-efficient. We describe and test different technical methods for exposing data to users, including techniques that avoid repeatedly copying large files. We ran experiments to measure how fast and scalable each method is under realistic conditions. The results show that by linking data instead of copying it, *PrepareData* can reduce delays and lower the burden on storage systems. This leads to better performance for scientists and less wasted computing and storage resources. These improvements are especially important for major international science projects, where efficient data delivery can accelerate research and reduce environmental impact.

Keywords

Data preparation, Federated data infrastructure, SKAO, Mount propagation, Storage orchestration, Cloud-native architectures, Green Computing

This article is included in the Horizon 2020 gateway.

This article is included in the Cloud-based Technologies collection.

Corresponding authors: Manuel Parra-Royón (mparra@iaa.es), Julián Garrido-Sánchez (jgarrido@iaa.csic.es)

Author roles: Parra-Royón M: Conceptualization, Investigation, Methodology, Software, Writing – Original Draft Preparation; **Garrido-Sánchez J**: Conceptualization, Funding Acquisition, Methodology, Project Administration, Supervision, Writing – Review & Editing; **Sánchez-Expósito S**: Conceptualization, Funding Acquisition, Investigation, Methodology, Project Administration, Writing – Review & Editing; **Darriba-Pol L**: Writing – Review & Editing; **Sánchez-Castañeda J**: Resources, Writing – Review & Editing; **Mendoza MÁ**: Writing – Review & Editing; **Coles J**: Writing – Review & Editing; **McConkey S**: Software, Writing – Review & Editing; **Joshi R**: Conceptualization, Software; **Barnsley R**: Software; **Salgado J**: Conceptualization, Writing – Review & Editing; **Verdes-Montenegro L**: Funding Acquisition, Project Administration, Supervision

Competing interests: No competing interests were disclosed.

Grant information: The author(s) declared that no grants were involved in supporting this work.

How to cite this article: Parra-Royón M, Garrido-Sánchez J, Sánchez-Expósito S *et al.* **Towards a sustainable astronomical data infrastructure: Optimising linking data from the Rucio datalake to the users areas within the SKA Regional Centres Network [version 2; peer review: 2 approved with reservations]** Open Research Europe 2026, **6**:18 https://doi.org/10.12688/openreseurope.22118.2

First published: 09 Jan 2026, **6**:18 https://doi.org/10.12688/openreseurope.22118.1

REVISED **Amendments from Version 1**

This revised version of the manuscript includes several improvements aimed at increasing its clarity, consistency, and reproducibility.

All bibliographic references have been carefully updated to follow a more homogeneous citation style, including, whenever possible, the complete metadata available. In addition, all references previously pointing to arXiv preprints have been replaced by their corresponding peer-reviewed journal articles or conference proceedings when published versions were available.

To improve the readability of the work, the Introduction now includes an explicit paragraph at its end describing the main objectives of the study and its specific contributions.

The Results section has also been expanded with additional paragraphs describing the experimental and deployment environment used for the tests, so that readers can more easily reproduce the reported experiments.

Several clarifications have been introduced regarding the interpretation of the comparative results. In particular, the comparison presented in Table 4 should be understood as a quantitative evaluation within a specific testbed context, rather than as a universal ranking of alternative approaches. The intention is not to establish that one solution is inherently better than another, but to provide a context-dependent comparison under controlled experimental conditions.

The Conclusions section now also includes an explicit discussion of the current limitations of the study, clarifying how these limitations may affect both the interpretation of the results and the reproducibility of the proposed approach.

Finally, it is important to note that the scope of this work is primarily focused on the design, deployment, and performance of the proposed solutions, rather than on aspects such as the full lifecycle management of prepared data or their security model, which remain relevant directions for future developments.

Any further responses from the reviewers can be found at the end of the article

1. Introduction

The Square Kilometre Array Observatory[1] (SKAO) and its distributed network of SKA Regional Centres [1] (SRCNet) face unprecedented challenges in managing data scale and movement, with SKA data volumes expected to reach exabyte levels.[2] The SKAO Data Lake is orchestrated with Rucio,[3] a policy-driven data management system that automates replication, placement, and lifecycle control across heterogeneous storage infrastructures. In Rucio, data are persisted in *Rucio Storage Elements* (RSEs), logical storage endpoints that abstract diverse backends (e.g., Ceph,[4] dCache,[5] EOS,[6] and cloud object stores) under a unified namespace. Crucially, replicating every dataset to *all* RSEs is infeasible at SKA scale due to storage, bandwidth, and operational costs.[7] Instead, datasets must be placed selectively on a minimal subset of SRCs (via their local RSEs) to balance scientific demand against resource constraints.

Access to these data and the associated compute capacity is provided, among others, through the SRCNet *Science Gateway*, which integrates data discovery, resources selection, and orchestration of data processing across SRC nodes.[8] Within representative SRCNet use cases,[9] scientists can launch science services — such as Jupyter Notebooks or visualisation tools — on the infrastructure where the data physically reside—typically the local RSE at a chosen SRC. However, because RSEs expose a storage abstraction optimised for global data management rather than user-facing runtimes, datasets residing in an RSE are not directly visible or consumable by end-user applications. Local SRC nodes therefore require a mechanism to *deliver* data from the RSE into user workspaces (e.g., home directories or service-attached paths) in a way that is both functional and efficient for analysis, visualisation, and processing tools.

A straightforward mechanism is to copy the requested data from the local RSE into the user area prior to execution. While operationally simple, this approach incurs duplication overheads, can introduce non-negligible preparation latency for large files, and exerts sustained pressure on user quotas and shared storage—particularly under concurrent demand. Although adequate for testing or low-volume scenarios, it is unlikely to remain viable at the operational scale foreseen for SRCNet.

SRCNet prototyping teams developed *PrepareData*, a service designed to deliver data from the RSE to the user workspaces, initially based on the copy strategy but open to include other additional strategies. In this context, we have designed and developed a local data-delivery strategy that avoids wholesale copying by *exposing* RSE-resident datasets within user workspaces. First, *operating-system–native* mechanisms (symbolic links and `mount --bind`) enable

[1]SKA Regional Centres Network: https://www.skao.int/en/science-users/119/ska-regional-centres

efficient linking from the RSE into user workspaces-visible paths. Second, *Kubernetes-native* mechanisms (based on Persistent Volumes and Persistent Volume Claims, PVs/PVCs) allow per-user dataset attachment within orchestrated services. Together, these approaches support both standalone services (e.g., SLURM-based workflows and other non-orchestrated applications) and Kubernetes-hosted environments (e.g., CANFAR,[10] visualisation tools, Jupyter Notebooks or virtually any Kubernetes service) without duplicating data. This design is shaped by the need to accommodate heterogeneous computing and services, and data storage environments, aiming to minimize redundant transfers within the local SRCNet nodes and fostering the adoption of mechanisms aligned with green computing.

This work addresses the problem of efficiently delivering datasets stored in local RSEs to user workspaces within SRCNet without unnecessary data duplication. We investigate alternative delivery strategies that allow direct access to RSE-resident data while maintaining performance and scalability. The main contributions are: (i) the design of operating-system–native mechanisms (symbolic links and bind mounts) and Kubernetes-native approaches (PVs/PVCs) for dataset exposure; (ii) their integration into the PrepareData service to support heterogeneous execution environments; and (iii) their assessment as a scalable alternative to traditional copy-based methods, reducing storage overhead, latency, and redundant data movement. Our evaluation of these methods focuses on functional deployment and performance characteristics rather than on comprehensive security validation, policy enforcement, or lifecycle management.

These resulting mechanisms enable an end-to-end workflow in which scientists can select an SRC through the Science Gateway, run analyses close to the data at the local RSE, and seamlessly access those datasets from their runtime environments. By combining these selective data placements across RSEs with local delivery mechanisms that eliminate redundancy, *PrepareData* bridges the gap between global data management and user-centric execution, providing a scalable and operationally efficient solution to consider given the performance requirements of SKA-scale science.

This paper is organised as follows. Section 2 provides an overview of existing approaches for user data provisioning within large-scale scientific infrastructures, high-lighting current limitations and motivating factors. Section 3 presents the architectural landscape of the SRCNet, outlining the operational and technical requirements that drive the need for the *PrepareData* service. The development, functional components and integration of the proposed solution are described in Section 4, including the data delivery strategies and modular architecture. Test experiments and performance results are presented in Section 5. Finally, Section 6 discusses future directions, including support for hybrid deployment models and interoperability across distributed RSEs.

2. Related work

The delivery of large-scale scientific datasets to users is a recurring challenge in Big Science platforms such as SKAO,[11] LOFAR,[12] and other observatories. A foundational contribution in this domain in 13, presents a scalable data delivery framework for astronomical observatories. Their system implements automated staging, user-aware transfer policies, and metadata tracking while reducing redundant copying—principles that align closely with the goals of *PrepareData.*

Several large-scale scientific collaborations have developed specialized data management systems to address the challenges of data duplication and efficient user access. ATLAS[3] experiment at CERN, developed Rucio to manage over a billion files across more than 120 data centres, facilitating policy-driven data placement and minimizing unnecessary data replication.[5] In ATLAS, data were organised into a very large number of relatively small files, a granularity well suited to replication across distributed storage. By contrast, SKAO products will consist of much larger files (≈ TB), making replication to every RSE prohibitively costly in both storage and bandwidth. Another approach working at CERN based on EOS[14] storage system, combined with CERNBox, offers users synchronized access to their data across various platforms, reducing the need for redundant data transfers. Another example is the dCache service, used by DESY [2] and Fermilab [3], where it provides a unified virtual filesystem that allows seamless access to distributed datasets without the need for multiple copies[15] but within the context of High Energy Physics. For instance, in another science context[16] proposes a memory-based caching mechanism that mitigates file system contention and accelerates data availability using in-memory staging on compute nodes. Similarly[17] introduces a scalable service for petascale systems, enabling intermediate staging from compute nodes to dedicated staging nodes before final persistence.

Additional other contributions address complementary needs in data workflows. For instance,[18] explores the seamless integration of interactive Jupyter workflows with distributed datasets via shared file systems, while in 19 evaluates the performance characteristics of different storage backends in large-scale image analysis pipelines, highlighting trade-offs between latency, throughput, and data isolation.

[2]DESY: Deutsches Elektronen-Synchrotron.
[3]FermiLab: Fermi National Accelerator Laboratory.

Beyond domain-specific platforms, the broader literature emphasizes key requirements for efficient data access in scientific infrastructures. Deduplication has been proposed as a critical energy-saving and scalability mechanism in cloud and distributed storage environments. The DD-ECOT framework[20] demonstrated up to 92.8% storage reduction via chunk-level deduplication in parallel processing contexts, significantly lowering bandwidth and storage costs. Khan *et al.*[21] describe a fault-tolerant, cluster-wide deduplication implementation in Ceph environments, showing high robustness and near-optimal space savings with minimal performance impact. Such approaches directly support the green-computing vision by reducing redundancies and operational energy usage. Furthermore, the energy-efficient storage architecture proposed in 22 uses online deduplication for VM image stores, reporting a reduction of up to 66% in energy consumption while preserving performance, an indication of how data preparation strategies that avoid duplication can contribute to sustainability.

At the orchestration level,[23] propose a time-driven data placement strategy that combines edge and cloud computing to minimize data transfer latency, adapting the placement dynamically based on bandwidth and storage constraints. While their context differs, the principle of bringing computation close to data resonates with the "move computation to data" model adopted in SRCNet.

Within green computing research, general principles like reuse, reduction, and support —Green Data Mining— have emerged as design tenets for sustainable data workflows.[24] These works illustrate the importance of reducing data duplication, supporting heterogeneous back-end infrastructures, enabling lifecycle-aware access control, and integrating robust user authentication and authorisation. They provide strong motivations and justification for the architecture behind *PrepareData*, which couples RESTful API interaction, asynchronous state management, and modular backing implementations to deliver efficient, scalable, and environmentally-conscious data delivery in distributed scientific environments like the SRCNet.

3. Problem statement

While the SRCNet has 16 member countries, its initial phase is focused on the more advanced initiatives from nine countries, notably including the Spanish SRC.[25] Each imitative is setting up an SRC that will collectively deliver access to around 700 PB of data per year to the global SKAO user scientist community. As the volume and complexity of scientific data continue to grow across distributed research infrastructures, ensuring efficient, scalable, and user-transparent data delivery mechanisms has become a critical challenge.

An initial phase of the work focused on enabling user access to data through the deployment of a Rucio-based datalake and the development of connectors for both interactive tools and batch processing environments. This provided a preliminary solution for transferring data from the RSE to the user workspace, but also exposed several limitations related to data distribution and repeated data movement to the local sessions. To address these issues, subsequent efforts concentrated on integrating the RSE, the user area and a linking mechanism capable of exposing datasets without redundant transfers. This line of work ultimately led to the design and implementation of the *PrepareData* system. In this context, the ability to prepare and expose data to end users—while minimizing duplication, latency, and resource contention—is essential for interactive and high-performance scientific workflows in next generation observatories in line with SKAO and the SRCNet.

Figure 1 illustrates the architectural foundation of the SRCNet ecosystem, which comprises multiple SRC nodes coordinated by the Global SRCNet services. Each SRC node hosts a range of scientific services and resources that are accessible both through a Science Gateway and independently. In this figure are highlighted several key components involved in managing data and orchestrating user workflows, including the DM-API (SRCNet Data Management API), SC-API (SRCNet Site Capabilities API), SRCNet Permissions API, and then local services exposed by the SRC nodes through the SRCNet GateKeeper [4] (GK)—such as the *PrepareData* service or the SODA[26] cutout service—, and other top level SRCNet local services like CANFAR [5], JupyterHub or CARTA,[27] an image visualisation and analysis tool, among others. *PrepareData* and other dependent services need GK to: (1) validate that the user has the necessary permissions to access a given dataset and its corresponding Rucio namespace within the SRCNet node; and (2) verify that the requested service is available and operational within the selected SRC node.

One of the central challenges within the current SRCNet architecture lies in efficiently delivering scientific data from the local Rucio Storage Element (RSE) of each SRC node to the user's working environment. In this federated infrastructure, user workspaces are provisioned locally within each SRC, yet the RSEs are not directly accessible or visible to users due

[4]GK repository: https://gitlab.com/ska-telescope/src/src-dm/ska-src-dm-da-service-gatekeeper/
[5]CANFAR is a science platform that integrates, among other things, a data visualisation environment, notebooks, and Linux desktops.

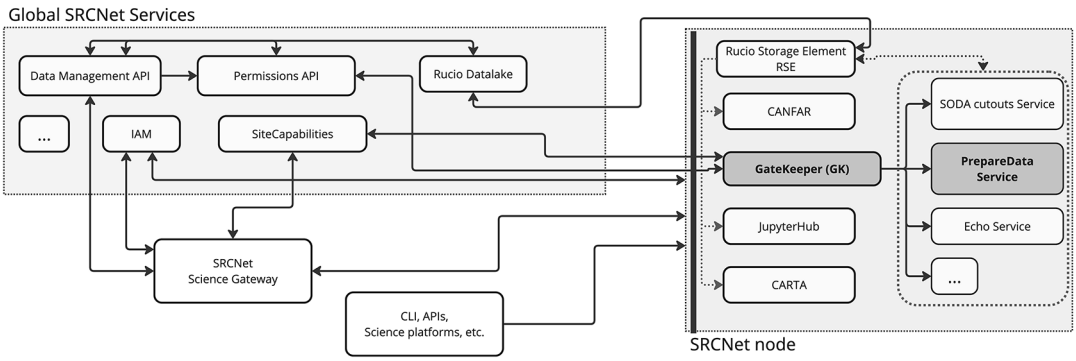

Figure 1. Architecture of the SRCNet services ecosystem, distinguishing between global services (SRCNet Global Services) and local services (SRCNet nodes), together with their dependencies. The diagram highlights the mandatory services required at SRC nodes and the specific integration of the PrepareData service. Access to PrepareData and other data-related services is protected and authorised through the GK component.

to their design as controlled backend storage end-points. Consequently, an intermediary mechanism is required to make these datasets available within each user's environment, enabling seamless use by scientific tools such as JupyterHub, CANFAR, or CARTA. The most direct way to achieve this delivery is to copy the required datasets from the RSE into the user's workspace before the analysis begins. This approach ensures native accessibility from user applications but introduces significant limitations, including data duplication, latency from repeated transfers, and inefficient utilisation of local storage capacity. In large-scale scientific environments such as the SRCNet, where multiple users may request overlapping datasets, this model rapidly becomes unsustainable in terms of performance, cost, and scalability.

Given this context, *PrepareData* is conceived as the service responsible for bridging the gap between data stored in the local RSE and the user-accessible workspace (see Figure 2). The service must not only ensure data availability and accessibility for the user but also do so in a way that respects the architectural heterogeneity of the SRCNet. This heterogeneity includes both standalone services (e.g., batch systems such as SLURM or interactive environments) and Kubernetes-based orchestrated deployments, as well as heterogeneous backend storage systems ranging from POSIX filesystems to Ceph- or xrootd, dCache-based RSEs.

Under these conditions, simply extending the default copy-based behaviour of *PrepareData* is not sufficient. New delivery strategies are needed that avoid unnecessary local data movement, instead leveraging operating-system and orchestration-level capabilities to expose RSE data efficiently within user environments. These strategies must preserve user transparency—data appearing natively within the user's workspace—, and ensure authorised access to the data, while optimising system efficiency by reducing bandwidth consumption, data duplication, and I/O overhead across SRC nodes.

This work addresses these challenges by presenting both currently deployed operational solutions and new approaches under development within the SRCNet prototyping framework. The proposed mechanisms target two main scenarios:

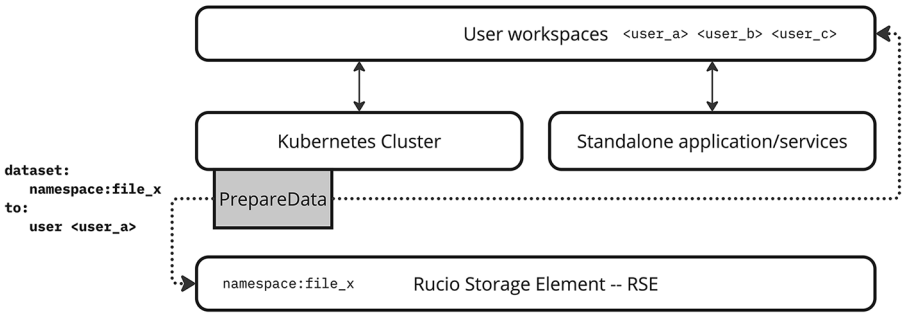

Figure 2. *PrepareData* acts as a bridge between the local RSE of the SRC and user storage areas, enabling access to data in user areasthat are accessible from services running both standalone and on Kubernetes. Here, *PrepareData* runs on Kubernetes.

(i) services operating in standalone mode and (ii) services deployed within Kubernetes clusters. For the latter, we focus on two complementary data delivery strategies—mount propagation and the use of Persistent Volumes and Persistent Volume Claims (PVs/PVCs)—which are designed to minimise duplication, reduce data-access latency, and establish a sustainable, resource-efficient model for user data provisioning in distributed scientific environment.

4. Development of PrepareData

In the current distributed infrastructure of the SRCNet, user home directories are currently isolated per SRC site, requiring a mechanism to expose requested data locally (see Figure 3). This functionality must abstract the underlying heterogeneity of storage back-ends and it provides a uniform mechanism for dataset exposure. It must support two primary usage scenarios: services executed within orchestrated environments (e.g., Kubernetes) and standalone applications running directly on the hosts system. The solution must also ensure that data access complies with user permissions and that the exposed datasets are correctly mapped to the corresponding user workspace without duplicating data unnecessarily. Among the constraints, the system must operate under limited storage quotas, maintain low latency in access times, and guarantee isolation between user sessions. At the same time, the service needs to interoperate with other SRCNet services through agreed interfaces in order to maintain independence among the SRCNet services.

Figure 4 illustrates a sequence of operations of a use case triggered when a dataset is staged to a SRC. In the current implementation, users must explicitly initiate the staging process and specify where data should be transferred. However, the final design aims to make this entirely transparent to the user: the system should automatically determine the

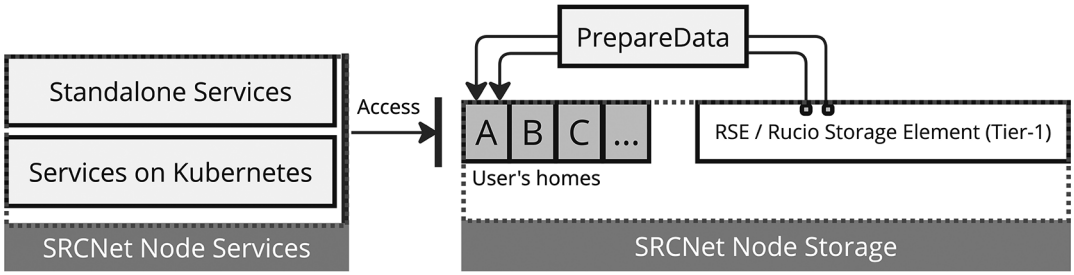

Figure 3. This diagram illustrates how services in local SRCNet nodes support both standalone and Kubernetes based deployments, and how *PrepareData* acts as an intermediary between the Rucio RSE (local Rucio storage) and the user workspace (user homes) within the SRCNet node storage.

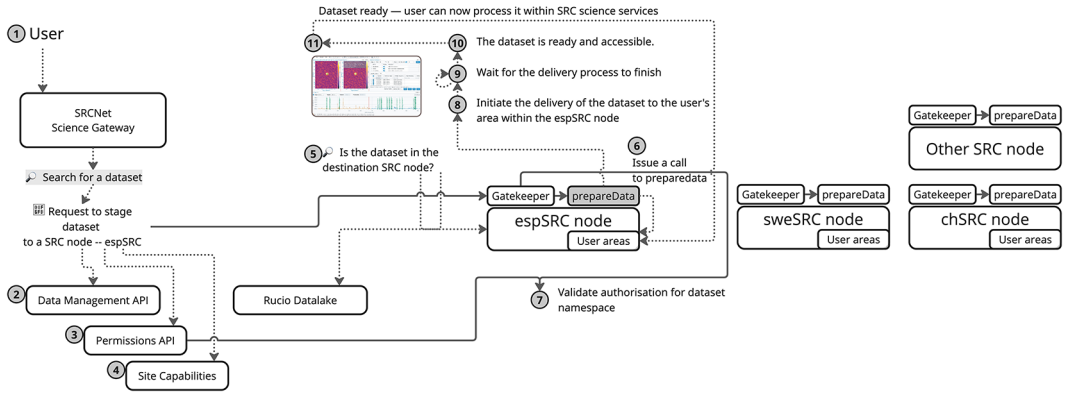

Figure 4. This diagram shows the workflow of operations and services involved in this specific use case, from the moment a scientist accesses the SRCNet Science Gateway until the requested data are made available in their user workspace.

appropriate destination without requiring manual intervention. The workflow for the current implementation is composed of the following steps:

1. An user initiates a request to prepare a dataset at a specific SRC site.

2. A preliminary check is performed to determine whether the dataset is already available at site. If not, a staging request is issued to transfer the dataset asynchronously to the corresponding RSE to access the selected dataset.

3. Metadata and site-specific information are retrieved from the SRCNet registry and information services (SRCNet Site Capabilities).

4. If the dataset is not present in the RSE, a request is made to trigger the asynchronous data transfer.

5. If the dataset is already available, a request is issued to the *PrepareData* service deployed at the SRCNet node.

6. Authorisation mechanisms validate that the user has the required privileges to execute data preparation actions at the site.

7. The *PrepareData* service is invoked, launching a local job that places the dataset into the user's area — typically via copy, symbolic links, mount propagation, or PVs/PVCs.

8. The system monitors the execution status of the preparation job and waits for it to reach a terminal state, typically READY.

9. Once the job is completed successfully, the dataset becomes accessible within the user's area in the selected SRCNet node.

10. Finally, the user opens a science-enabling application (CARTA, JupyterHub, etc.) to explore/use the requested data.

In order to ensure consistent interfaces across all SRC nodes and enable seam-less integration with other components of the SRCNet ecosystem, the *PrepareData* functionality must expose a Web service with a standardized API. This is achieved through a RESTful API, which provides a homogeneous interaction layer regardless of the underlying infrastructure. The API is designed to receive as input an OIDC token –used for user authentication and authorisation– as part of the `HTTP Authorization Bearer` header, along with a payload containing a list of datasets to prepare for the local user. Each dataset entry specifies the logical dataset name, the corresponding path in the local RSE, and the user-relative mount point (typically under the user's home directory). The API specification has been formally defined using the OpenAPI standard.[28] An example API call by using `curl` command is illustrated below in the next listing:

```
curl -X 'POST' \
'http://localhost:8000'\
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer $SKA_TOKEN$' \
-d '[
[
"testing:galactic_centre.fits",
"testing/84/1c/galactic_centre.fits",
"./testing"
]
]'
```

In this request, the service will prepare the dataset galactic centre.fits, identified logically under the testing namespace, whose actual location in the RSE is: `testing/84/1c/galactic_centre.fits`. The file will be made accessible within the user's area, allocated in the subdirectory `./testing` relative to the user home. The invocation of the *PrepareData* functionality is designed to be intermediate by GK, as illustrated in Figure 4. This architectural

decision ensures consistency in access control and direct invocation of the *PrepareData* API is only permitted in local testing environments; in production scenarios, all external requests are routed through the GK.

4.1 Service implementation

The *PrepareData* service has been implemented in Python, leveraging the FastAPI[29] framework for building a high-performance RESTful web interface, and served using the uvicorn [6] ASGI server. The exposed API is defined using the OpenAPI specification. To support multiple data preparation implementations in a modular and extensible way (see details in 4.2), *PrepareData* adopted the object factory design pattern, which allows us to integrate our custom implementations seamlessly. Since the delivery of data from the RSE to the user's area is not instantaneous and may vary significantly depending on the specific implementation used (e.g., linking vs. copying), the *PrepareData* service implements an asynchronous processing –without blocking subsequent operations– model as depicted in the Figure 5.

To handle asynchronous tasks, the service relies on Celery [7], a distributed task queue, which enables the delegation of long-running or delayed tasks (see Figure 5). Upon initiation of a data preparation request, a unique preparation identifier (job id) is generated and returned to the client. This identifier can be used in subsequent API calls to query the status of the operation.

The life-cycle of a data preparation request is captured through a finite set of states, where in the *Pending* state, the preparation task has been accepted and is currently being processed. Once the requested data have been successfully made available in the user's home directory, the process enters the *Completed* state, indicating that the data are ready to be used by a scientific service. Finally, in the *Error* state, the data preparation has failed due to several conditions: a) the service may be unable to stage or link the dataset from the local RSE to the user's area; b) the requested dataset may no longer exist in the RSE due to expiration or policy-driven deletion; c) the user's home directory may be unavailable because no session has been initiated; d) or there may be insufficient storage space to accommodate the requested data.

4.2 Data preparation implementations

The *PrepareData* service offers a flexible mechanism for integrating multiple implementations within its codebase based on *objects factory*. This design principle allows for the modular inclusion of different implementations without modifying the core service logic. As a result, each SRC could deploy one or several implementations tailored to its available infrastructure and the needs of the services it exposes.

4.2.1 Copy-based delivery

The copy-based approach represents the most time-consuming scenario among the supported data preparation strategies. It is primarily designed as a baseline implementation to validate the operational correctness of the *PrepareData* service. In this mode, data are physically copied from the local RSE to the user's area.

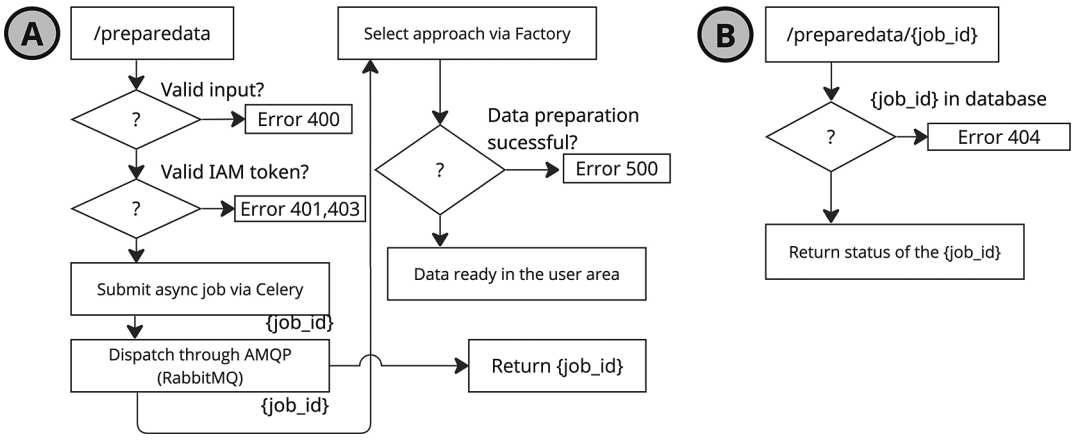

Figure 5. Diagram of the two *PrepareData* calls: (A) the initial request with parameters, and (B) the status query to check the state of the data preparation job.

[6]uvicorn: https://uvicorn.dev/
[7]celery: https://docs.celeryq.dev/en/stable/

Its current implementation leverages standard operating system call and wrappers (e.g., cp) to perform file transfers. This approach does not maintain any internal registry or tracking metadata about which files were delivered to which user, nor the context of the data preparation job. As a result, this prevents *PrepareData* from executing automated garbage collection or lifecycle control (e.g., revocation, expiration) over the prepared datasets.

Additionally, this method is not using any caching or deduplication logic. If a dataset is already available either at the RSE or even within a user's home directory, the copy is performed again without verification. Consequently, repeated requests for the same dataset by the same or different users will result in multiple independent copies in their respective user areas. This behavior may lead to unnecessary storage consumption and redundant data replication, which is unsustainable at scale. A more optimal implementation of copy-based delivery could have been achieved by introducing, for example, mechanisms to detect and avoid redundant copies when the dataset is already present at the target RSE; however, such enhancements were considered out of scope for this work.

The total preparation time includes two distinct phases: (i) a potential staging period, if the dataset is not already present in the local RSE, and (ii) the copy time from the RSE mount point to the user's home directory.

4.2.2 Mount bind and mount propagation

This approach leverages a Linux kernel feature known as `mount --bind`, which allows a file or directory that is already part of the filesystem to be remounted at another location. Unlike symbolic links, `mount --bind` operate at the kernel level and **effectively** mirror a directory tree from one path to another within the same namespace, preserving permissions and access controls.

In the context of *PrepareData*, this mechanism enables dynamic linking of files located within the RSE directly into a user's area, which may reside on a separate volume or filesystem. With all, this method ensures data encapsulation, as users are only granted access to the files within their designated home directories, even though the underlying storage is shared. It also prevents privilege escalation or unauthorized browsing of unrelated directories.

However, this approach requires root-level privileges (or equivalent capabilities) to execute `mount –bind` operations on the hosts. Furthermore, since Kubernetes workloads are typically distributed across multiple nodes, the bind operation must be consistently applied on all nodes where the target pod might be scheduled. To address this, *PrepareData* uses `mountPropagation: HostToContainer` [8] to propagate the mounted path from the host into the containerized workload in Kubernetes. Figure 6 presents the high-level architecture of the proposed solution. The diagram illustrates how the *PrepareData* service is integrated with a backend database responsible for managing metadata associated with each data preparation request.

For each request, the implementation registers a unique transaction identifier (`job id`), the target user, the original dataset path within the RSE, and the destination path within the user's area. In addition to this core metadata, the database stores timestamps such as creation and expiration among other. This enables future lifecycle management features, such as automated permissions revocation or unlinking of datasets once their access window expires.

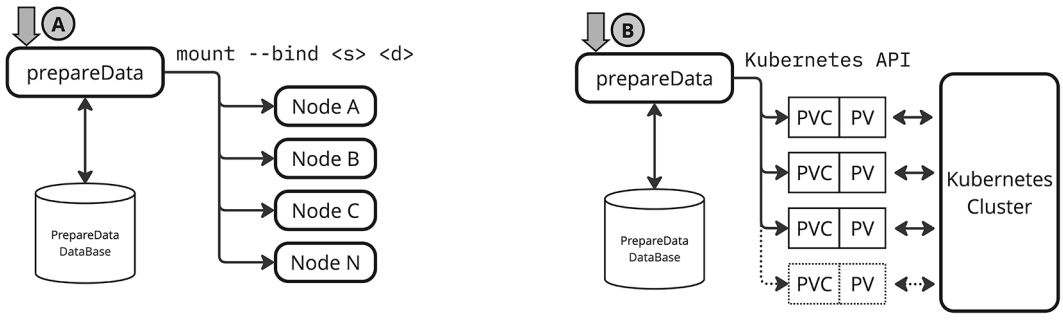

Figure 6. Scheme of operation of mount and PV/PVCs approaches.

[8]HostToContainer feature: https://kubernetes.io/docs/concepts/storage/volumes/#mount-propagation

4.2.3 Persistent volume-based linking

Given that a significant portion of the scientific services deployed in the SRCNet architecture are or will be deployed on Kubernetes, *PrepareData* have also support native data provisioning mechanisms compatible with the orchestrator storage model. Kubernetes provides a storage abstraction through PVs and PVCs. A PV represents a piece of storage in the cluster, provisioned either statically or dynamically, while a PVC is a request for storage by a user or service. The linkage between the two enables dynamic mounting of specific directories or datasets directly into containerized services, following a declarative provisioning model. Furthermore, using the `SubPath` capability, it is possible to mount a specific file or subdirectory from a volume into a container, rather than the volume's root directory. Additionally, the use of `StorageClass` definitions allows administrators to define different backend types (e.g., CephFS, Cloud Storage, NFS, local-path, among other).

In this approach the *PrepareData* service acts by creating a PV/PVC pair that maps the requested RSE dataset to a dedicated mount path within the user's home directory on the pod itself. At runtime, the corresponding Kubernetes deployment must be customized to recognize which PVCs are associated with the user. These PVCs must be mounted as volumes during pod instantiation, ensuring that the dataset is accessible within the user's containerized session. This design pattern is effective for data delivery in orchestrated environments, but it introduces a coupling between *PrepareData* and the instantiation logic of the consuming service.

Similar to the mount bind approach, the use of PVs for data linking requires deployment-time awareness and injection of volume declarations, which must be dynamically associated with each user session by using a mapping database. This necessitates a certain degree of flexibility and automation in service deployment templates, especially for multi-user environments such as JupyterHub or CARTA, among others. An example of this integration is presented in the following section, where we describe a JupyterHub instance deployed within a Kubernetes cluster using *PrepareData*-generated PVCs by customising the instantiation procedure.

Figure 6 illustrates how the *PrepareData* service handles requests using a PV and PVC strategy. Similar to the `mount --bind` approach, a backend database is used to track metadata for each dataset preparation. In this case, the database also stores metadata for managing the life-cycle of dynamically created PV/PVC resources.

For each RSE dataset-user pair requested, a dedicated PV/PVC is generated via the Kubernetes API, and scoped to the appropriate namespace corresponding to the consuming service.

The PV/PVCs volumes remain available for reuse, allowing applications such as JupyterHub to dynamically discover and inject the correct PVs into user sessions. This is achieved by querying the backend database for active dataset preparations and attaching the volumes at runtime, or after a new session requested.

4.3 Integration scenarios for the SRCNet

The *PrepareData* service supports multiple integration strategies to tackle the diverse architectural and execution contexts within the SRCNet. Both implementations — `mount --bind` and Kubernetes-native PV and PVC— aim to minimise data duplication and facilitate transparent access to datasets, they differ in their system-level requirements and the degree of deployment customisation needed. Specifically, the `mount --bind` strategy is applicable to both standalone and Kubernetes-based services, whereas the PV/PVC-based model is exclusive to Kubernetes environments. The following sections detail the integration characteristics and requirements of each approach.

4.3.1 Standalone environments

Standalone environments refer to services and applications that operate outside Kubernetes orchestration, such as traditional batch systems or containerised tools running directly on host systems. For these environments, *PrepareData* leverages the native Linux `mount --bind` functionality to expose datasets stored at the RSE directly into a user's home directory or working area allowing services to access data as if it were locally present. This method is transparent to the consuming application. Consequently, no internal modification of the service logic is required, making it highly suitable for integration with resource managers such as SLURM or HTCondor. For example, a user job executed via SLURM can access a dataset located at `/mnt/rucio/.../testing/84/1c/galactic_center.fits` through a `mount --bind` to the user's area (e.g., `/home/user/testing/`), enabling immediate usage within the job script without requiring explicit data copying or staging mechanisms. *PrepareData* performs the bind operation across all relevant compute nodes to ensure availability, and relies on system privileges (via daemon or privileged containers) to execute the mount consistently. This approach remains effective provided that the file system is shared or mirrored across nodes but do not require container-level isolation.

4.3.2 Kubernetes-based services

Kubernetes has become the standard orchestration layer for many SRCNet services. Consequently, *PrepareData* has implementations based on `mount --bind` with `HostPropagation`, and PV/PVC-based delivery. Both require service-level customisation at deployment time to ensure seamless integration of user datasets.

Mount propagation

To enable full isolation and user-level data exposure, for example, the JupyterHub deployment can be configured so that each user is assigned a dedicated directory under a shared mount point, typically with this template `/mnt/users/<username>/`. This allows for direct integration with externally managed data, and provides consistency across sessions and services such as *PrepareData.*

To achieve this, the JupyterHub Helm chart `values.yaml` must be configured to,

- a) mount the parent directory `/mnt/users` into the user container, b) dynamically assign `/mnt/users/<username>` as the container's working home directory, and c) ensure mount propagation is enabled so that bind-mounted data becomes visible.

An example configuration for the `singleuser` section in `values.yaml` is shown below:

```
singleuser:
...
extraVolumes:
- name: home-mount
hostPath:
path: /mnt/users
type: Directory

extraVolumeMounts:
- name: home-mount
mountPath: /mnt/users
mountPropagation: HostToContainer

extraEnv:
NB_USER: "{username}".
HOME: "/mnt/users/{username}"
lifecycleHooks:
postStart:
exec:
command:
- sh
- -c
- >
export HOME=/mnt/users/$(whoami) && cd $HOME
```

This configuration ensures that the environment variable `$HOME` is redirected to the correct location, matching the IAM-provided username. The user is automatically placed inside their designated folder upon session start.

In combination with *PrepareData*, any dataset bind-mounted into `/mnt/users/<username>/<namespace>/` becomes directly accessible within JupyterHub without requiring any additional volume provisioning, copying, or duplication.

Dynamic injection of PVs/PVCs via spawner hooks in JupyterHub

For services running under Kubernetes, the most native integration of user-specific datasets delivered by *PrepareData* is through dynamically created PVs/PVCs. In this workflow, each dataset requested by a user is exposed as a PV/PVC and recorded in a lightweight metadata store linked to the user's IAM identity. When a new JupyterHub session is initiated,

the `KubeSpawner` pre-spawn hook, a customizable function that runs just before the user's container is launched and that enables the modification of the environment variables or the injection of PV/PVC. Therefore, this hook queries the metadata store, retrieves the relevant PVs/PVCs, and injects them into the pod specification. As a result, when the session is opened by the user datasets are seamlessly available within the container.

An example of the pre-spawn hook logic in Python is shown below:

```
...
def pre_spawn_hook(spawner):
username = spawner.user.name
pvc_list = query_user_pvcs_from_db(username)
for pvc in pvc_list:
spawner.volumes.append({
'name': pvc['name'],
'persistentVolumeClaim':
 {'claimName': pvc['name']}
 })
spawner.volume_mounts.append({
'mountPath': pvc['mountPath'],
'name': pvc['name']
 })
```

A corresponding JupyterHub configuration snippet to enable this hook is as follows:

```
c.KubeSpawner.pre_spawn_hook = pre_spawn_hook
```

For example, if a dataset is prepared as `pvc-user-namespace-m31-fits`, and designated to be mounted at `/home/jovyan/<namespace>/`, the user will see it immediately upon logging in.

5. Results and discussion

All tests were executed on a single Kubernetes compute node equipped with 8 CPU cores and 16 GB of RAM, running Kubernetes version 1.28. Both the RSE and the user areas were provided through CephFS volumes exposed via a CSI provisioner. As a consequence, all data copy operations, mount operations, and PV/PVC provisioning actions ultimately interact with the CephFS backend.

The tests were designed to evaluate three alternative mechanisms for materialising user datasets in the context of the *PrepareData* workflow: (i) copying the data directly from the RSE to the User Area, (ii) using Linux bind mounts to expose RSE subdirectories inside a user area, and (iii) dynamically constructing PVs/PVCs to attach datasets to user environments through Kubernetes' storage abstraction.

For the copy test, file sizes ranged from 100 MB up to 10240 MB. For the bind-mount and PV/PVC tests, we generated storms of 10 to 1000 mount or provisioning operations to evaluate the behaviour under increasing concurrency. All tests were repeated 10 times per configuration. It is important to note that the three test categories are *not directly comparable.* Data copies scale primarily with the size of the transferred payload and are limited by CephFS throughput. Bind-mount operations are lightweight kernel metadata operations, dominated by VFS and namespace handling. PV/PVC provisioning exercises the Kubernetes control plane and the CSI driver, and is therefore orders of magnitude slower, involving distributed coordination rather than local I/O.

Table 1 summarises the average wall-clock time required to copy datasets of increasing size from the CephFS-backed RSE to the user area.

The results exhibit an approximately linear relationship between dataset size and copy time, with an effective throughput of 1.3–1.4 GB/s for larger files. This demonstrates that CephFS provides efficient sequential throughput; however, data copying remains costly for workflows that repeatedly transfer large datasets.

Table 2 presents the results for the bind-mount test, where batches of 10–2000 mount operations were created sequentially. Besides the wall-clock time, we also report the average system CPU fraction, defined as the ratio between the system CPU time and the total elapsed time.

Table 1. Average time to copy data from the RSE to the user area. Values represent means over ten runs.

Size (MB)	Avg. Time (s)
100	0.131
200	0.206
500	0.429
1024	0.830
2048	1.510
3072	2.170
4096	2.830
5120	3.607
10240	7.213

Table 2. Bind-mount benchmark results. Columns report the number of mount operations, the average wall-clock time, and the fraction of CPU time spent in system mode (kernel).

Number of Mounts	Avg. Time (s)	Sys CPU Fraction
10	0.109	0.082
50	0.391	0.109
100	0.773	0.115
200	1.571	0.112
400	3.563	0.108
800	9.227	0.114
1000	13.242	0.123
2000	43.834	0.129

Table 3. PV/PVC test results for static provisioning. Columns report the number of PV/PVC pairs, the average wall-clock time, and the system CPU fraction.

Number of PV/PVC	Avg. Time (s)	Sys CPU Fraction
10	5.106	0.315
50	25.403	0.318
100	50.757	0.312
200	104.217	0.312
400	206.714	0.310
800	413.254	0.312
1000	517.851	0.315

Bind-mount operations scale nearly linearly with the batch size. For 100 mounts the latency remains below 1 s, while 1000 mounts require approximately 13 s. The system CPU fraction is remarkably stable (0.10–0.13), indicating a consistent amount of kernel work per mount. Although large storms of mount operations accumulate into noticeable latency for thousands of datasets, the overall behaviour remains predictable and efficient. This makes bind-mounting a viable strategy for linking large numbers of shared-read datasets in *PrepareData.*

Table 3 reports the results for the static PV/PVC provisioning test. For each dataset, one PV and its corresponding PVC were created and waited upon until the claim reached the Bound state. PV/PVC creation is significantly more expensive than bind mounting. Even small batches of 10 PVCs require over 5 s, and provisioning 1000 volumes takes more than 8.5 minutes. The system CPU fraction (0.31–0.32) is higher than in the mount test, reflecting the larger number of `syscalls` and kernel

interactions during repeated `kubectl` operations. Nonetheless, most of the elapsed time corresponds to waiting for the Kubernetes control plane (API server, controllers, CSI provisioner) rather than local CPU work.

The three experiments reveal distinct behaviours in terms of scalability and operational characteristics for dataset materialisation in the *PrepareData* workflow. Although the copy-based strategy exhibits good throughput and predictable performance as dataset size increases, it has a fundamental drawback: every request results in a full physical replication of the data into the user workspace. This implies a proportional consumption of storage resources and introduces cumulative overhead when the same dataset must be prepared repeatedly for different users or different sessions. Despite scaling well from a throughput perspective, the cost of repeatedly duplicating large volumes of data can therefore become prohibitive at scale.

In contrast, the approaches based on bind mounts and on PV/PVC attachments do not replicate the underlying files, but instead link a shared dataset into multiple user workspaces. This results in substantial savings in storage utilisation, as the same physical file is referenced by many users simultaneously without duplication. Both techniques are therefore conceptually aligned with scenarios where datasets are shared, reused, or accessed in a read-mostly pattern. The bind-mount approach, in particular, demonstrates excellent scalability with respect to the number of datasets: preparation times remain below one second for approximately one hundred mounts and grow linearly for larger batches. The consistently low system CPU fraction indicates that the kernel work associated with each mount operation is modest. This makes the method attractive for services that can natively operate on filesystem-level links, whether inside or outside the Kubernetes environment.

The PV/PVC strategy instead reflects the native Kubernetes model for storage attachment. Its semantics are appropriate for services designed to operate exclusively within Kubernetes, especially when persistent or isolated storage allocations are required. However, the test results show that PV/PVC-based materialisation introduces substantially higher latency per dataset, often one or two orders of magnitude above the bind-mount case. The provisioning of large numbers of PVs and PVCs imposes significant coordination work on the control-plane components and the CSI backend, leading to long preparation times when hundreds of datasets must be attached simultaneously. While these tests intentionally use exaggerated scales—far beyond typical operational scenarios—they provide a clear indication of how the system behaves under extreme load and whether each approach remains sustainable for the projected volume of users and datasets.

Taken together, the results indicate that data copying, bind mounting, and PV/PVC attachment each serve different operational scopes within *PrepareData.* Bind mounts provide an efficient mechanism for linking shared datasets via POSIX paths, while PV/PVCs offer a Kubernetes-native method for attaching storage with well-defined isolation semantics. Data copying, although predictable and throughput-driven, incurs additional storage consumption and repeated transfer costs. The test trends show that each approach scales differently depending on whether the dominant factor is data volume, filesystem operations, or control-plane activity. Ultimately, the choice among these mechanisms should be guided by the requirements of the target service—whether it prioritises data locality, Kubernetes integration, or shared access patterns—rather than by raw performance alone.

Mechanisms such as symbolic linking, shared filesystems or on-demand mounting offer not only performance advantages but also enable tracking and accountability for each data preparation request. Such implementations typically require the use of a persistent database to store metadata about each preparation, including the user identity, dataset references, and status information. This enables key operations such as expiration handling, revocation of access permissions, and querying of preparation history. The system can then maintain a comprehensive registry of active and historical data preparations, supporting more robust and policy-compliant data management across SRCs.

However, these alternative implementations may require the consuming service to be customized or reconfigured to correctly resolve and access data paths. This confirms that the data preparation implementation is tightly coupled with the service integration model, and that successful deployment depends not only on backend capabilities but also on adapting the services that consume the data to ensure seamless accessibility regardless of the underlying mechanism.

Table 4 summarises the PrepareData delivery mechanisms evaluated within a prototype SRCNet environment. The comparison is intended to highlight practical trade-offs among approaches rather than to provide a universal ranking. Because the mechanisms introduce different types of overhead and were assessed under specific infrastructure conditions, the performance indicators should be interpreted as qualitative and testbed-specific. The copy-based method offers broad compatibility but may incur significant data duplication and preparation latency. Mount-based and PV/PVC-based strategies typically reduce data movement and enable shared access, at the cost of varying levels of service customisation and deployment complexity. The columns indicate: (i) support for Kubernetes-based orchestration or standalone

Table 4. Comparison of *PrepareData* delivery mechanisms across execution environments within a prototype SRCNet testbed, highlighting qualitative trade-offs. These results should therefore be interpreted as guidance for selecting appropriate strategies depending on local SRC requirements rather than as definitive performance comparisons.

Approach	k8s/Standalone	Customisation	Estimated performance	Scalability
Copy	Both	No	Low	Low
Mount	Both	Yes/No	High	Very High
PV/PVC	k8s	Yes	High	High

execution, (ii) whether user-facing service adaptation is required, (iii) qualitative performance observed in the test environment, and (iv) scalability characteristics under concurrent use.

To ensure the reproducibility of the benchmarks and results presented in this study, the following experimental environment and procedures were established.

- The evaluation was conducted on a Kubernetes (v1.28.2) cluster consisting of one control plane node and three worker nodes. Each node was provisioned with 8 vCPUs, 32 GB of RAM, and a 100 GB local root disk, running Ubuntu 22.04 LTS with its native kernel.

- The internal networking infrastructure provided a high-speed 100 Gbps interconnect between nodes, while external internet connectivity was constrained to a 10 Gbps bandwidth. Storage was managed via Ceph (version Reef 18.2.2 stable), utilizing two distinct CephFS volumes: one dedicated to the local Rucio Storage Element (RSE) and another for the User Area. Both volumes were configured to be globally visible and mounted across all nodes in the cluster.

- All tests were performed using PrepareData version 0.1. The benchmarking suite is publicly available and can be retrieved from the Zenodo repository[29] or via the following repository: https://github.com/manuparra/preparedata-tests.git.

The testing framework evaluates three specific operational environments: copy, mount, and PV/PVC. To replicate these tests, the following prerequisites must be met:

- Administrative Access: The execution environment requires kubectl access with sufficient privileges to provision PVs/PVCs.

- Storage Visibility: Verified mounting capabilities and visibility of the CephFS-based RSE and User Area across all cluster nodes.

- Execution: Following verification, the benchmarks are triggered through the execution of the scripts `copy.sh`, `mount.sh`, and `pvpvc.sh`.

The raw output generated by these scripts constitutes the data source for the comparative analysis and the results tables detailed in this subsection.

6. Conclusion and future work

This work has addressed the problem of delivering scientific datasets from RSEs to the user working area within an infrastructure like SRCNet. The need to make data available efficiently, reliably, and in a reusable manner is central to enabling science across multiple SKA Regional Centres. Among the delivery models, the copy-based approach remains a valid and straightforward baseline. It is compatible with the widest range of services in the SRCNet ecosystem, especially legacy or standalone applications. However, this method has major disadvantages in environments with large data volumes, such as high latency, increased storage consumption, and limited support for data reuse.

To tackle these limitations, this work has introduced and implemented two alternative mechanisms for data exposure: `mount --bind` and Kubernetes-native PV/PVC injection. Both approaches have been developed to offer lightweight, efficient, and flexible data delivery with minimal overhead in terms of time, storage, and data duplication. Additionally, they support multi-user access to shared datasets without unnecessary replication.

The `mount --bind` strategy leverages native Linux kernel functionality and is ideal for all the standalone applications and cluster schedulers such as SLURM, HTCondor among other. The PV/PVC-based solution, in turn, fits naturally into container orchestration platforms like Kubernetes and integrates cleanly with user-managed services such as JupyterHub, as demonstrated via dynamic PV/PVC injection through spawner hooks. This functionality can equally be extended to other services in Kubernetes with comparable data access requirements.

Importantly, the discussion of these approaches reveals that there is no one-size-fits-all solution. The optimal strategy depends on the specific characteristics of each SRC node, including its storage backend, orchestration platform, user workload patterns, and service integration requirements. For instance, standalone HPC environments may benefit more from mount bind strategies, while cloud-native services deployed on Kubernetes clusters are better served by PV/PVC-based mechanisms. Furthermore, the *PrepareData* service itself has been designed with extensibility and modularity in mind. Its architecture supports the dynamic selection of data preparation strategies based on local configuration, enabling each SRC to adopt the most suitable approach without modifying the core logic of the service. These strategies should not be viewed as a hierarchy of technical superiority, but rather as context-specific solutions.

From a broader perspective, the work contributes to the ongoing efforts in green computing and sustainable data management. By reducing redundant data transfers and promoting reuse of existing datasets, the proposed strategies align with energy-efficient computing principles and help mitigate the environmental impact of large-scale scientific infrastructures.

This study serves as a proof-of-concept, and the results presented reflect performance trends observed within a controlled environment. First, the evaluation focuses on local data-delivery mechanisms within a single SRC context and does not assess cross-SRC scenarios or wide-area deployments. Second, performance results are derived from representative workloads and infrastructure configurations, which may not fully reflect the diversity of storage backends, network conditions and user behaviours across the entire SRCNet. Third, the proposed approaches rely on assumptions about administrative control that may not hold in all operational environments, particularly those with strict security or multi-tenancy constraints.

Nevertheless, several areas remain open for future exploration. These include the implementation of intelligent caching mechanisms, integration with object storage systems (e.g., S3, Ceph RGW), and the development of a unified metadata and access tracking layer to support fine-grained policy enforcement across SRCs. Additionally, a comprehensive performance evaluation of the PV/PVC approach under realistic SKA-scale workloads would provide further insights into its operational viability.

Ethics and consent

Ethical approval and consent were not required.

Data availability

SRCNet preparedata tests: software and data [v1.0.2]. https://doi.org/10.5281/zenodo.17772362

This project contains the following underlying data:

- `data/copy_time.csv.` Raw per-run timing measurements used to compute average copy performance. Table 1
- `data/mount_benchmark.csv.` Raw wall-clock time and CPU-usage values for mount–bind scalability experiments. Table 2.
- `data/pvc_benchmark.csv`. Raw timing and CPU metrics for PV/PVC provisioning tests. Table 3.

No datasets used in this study are subject to restrictions, and no embargo applies. All materials are fully accessible for reuse and replication.

Software availability

SRCNet preparedata software v1.0.2. https://doi.org/10.5281/zenodo.17772362

- - All source code is also available in a public GitHub:

GitHub: https://github.com/manuparra/preparedata-tests

Acknowledgements

MP, SSE, JG, JS, MM, LD, EJ and LVM acknowledge financial support from the grant CEX2021–001131-S funded by MICIU/AEI/ 10.13039/501100011033 and from the grant TED2021-130231B-I00 funded by MICIU/AEI/ 10.13039/501100011033 and by the European Union NextGenerationEU/PRTR, acknowledges financial support from the grant PID2021-123930OB-C21, PID2024-155817OB-I00 funded by MICIU/AEI/ 10.13039/501100011033 and by ERDF/EU, and by the grant INFRA24023 (CSIC4SKA) funded by CSIC. MP, JG, SSE, LVM acknowledge support from the European Science Cluster of Astronomy and Particle Physics ESFRI Research Infrastructures project that has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 824064. Prototype of an SRC (SPSRC) service and sup-port funded by the Ministerio de Ciencia, Innovación y Universidades (MICIU), by the Junta de Andalucía, by the European Regional Development Fund (ERDF) and by the European Union NextGenerationEU/PRTR. The SPSRC acknowledges financial support from the Agencia Estatal de Investigación (AEI) through the "Center of Excellence Severo-Ochoa" award to the Instituto de Astrofísica de Andalucía (IAA-CSIC) (SEV-2017-0709) and from the grant CEX2021–001131-S funded by MICIU/AEI/ 10.13039/501100011033.

References

1. SKA Observatory: **SKAO foundational document.** 2013. Accessed August 19.
Reference Source

2. Scaife AMM: **Big telescope, big data: Towards exascale with the Square Kilometre Array.** *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences.* 2020; **378**(2166): 20190060.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

3. Barisits M, Beermann T, Berghaus F, *et al.*: **Rucio: Scientific data management.** *Computing and Software for Big Science.* 2019; **3**(1): 11.
Publisher Full Text

4. Fuhrmann P, Gülzow V: **dcache, storage system for the future**. In *European Conference on Parallel Processing* (pp. 1106–1113). Springer; 2006.
Publisher Full Text

5. Peters AJ, Sindrilaru EA, Adde G: **EOS as the present and future solution for data storage at CERN.** *J Phys Conf Ser.* 2015; **664**(4): 042042.
Publisher Full Text

6. Bonaldi A, Brüggen M, Burkutean S, *et al.*: **Square Kilometre Array science data challenge 1: Analysis and results.** *Mon Not R Astron Soc.* 2020; **500**(3): 3821–3837.
Publisher Full Text

7. SRCNet: **SRCNet v0.1 implementation plan (Technical Report SRC-0000009-01).** 2025.

8. Square Kilometre Array Observatory (SKAO): **SRCNet use cases (Technical Report SRC-0000004-01).** 2025.

9. Gaudet S, Hill N, Armstrong P, *et al.*: CANFAR: The Canadian Advanced Network for Astronomical Research. In *Software and Cyberinfrastructure for Astronomy (SPIE).* 2010; (Vol. **7740**, pp. 577–586).

10. Hartley P, Bonaldi A, Braun R, *et al.*: **SKA science data challenge 2: Analysis and results.** *Monthly Notices of the Royal Astronomical Society.* 2023; **523**(2): 1967–1993.
Publisher Full Text

11. van Haarlem MP, Wise MW, Gunst AW, *et al.*: **LOFAR: The low-frequency array.** *Astronomy & Astrophysics.* 2013; **556**: A2.
Publisher Full Text

12. Villard E, *et al.*: **Advanced data products for radio observatories.** *Astron Comput.* 2025, September 12: 101003.
Publisher Full Text

13. Elmsheuser J, Di Girolamo A: **Overview of the ATLAS distributed computing system.** *EPJ Web Conf.* 2019; **214**: 03010.
Publisher Full Text

14. Mkrtchyan T, Chitrapu K, Garonne V, *et al.*: **dCache: Inter-disciplinary storage system.** *EPJ Web of Conferences.* 2021; **251**: 02010.
Publisher Full Text

15. Wozniak JM, Sharma H, Armstrong TG, Wilde M, Almer JD, Foster I: **Big data staging with MPI-IO for interactive X-ray science**. In *2014 IEEE/ACM International Symposium on Big Data Computing* (pp. 26–34). IEEE; 2014.
Publisher Full Text

16. Abbasi H, Wolf M, Eisenhauer G, *et al.*: **DataStager: Scalable data staging services for petascale applications.** *Cluster Computing.* 2010; **13**: 277–290.
Publisher Full Text

17. Kluyver T, Ragan-Kelley B, Pérez F, *et al.*: *Jupyter Notebooks—a publishing format for reproducible computational workflows*: IOS Press; 2016. 87–90.
Publisher Full Text

18. Peters A, Sindrilaru E, Adde G: **EOS as the present and future solution for data storage at CERN.** In *J Phys Conf Ser* (Vol. **664**, 042042), IOP; 2015.
Publisher Full Text

19. Zhou Q, *et al.*: **Energy efficient algorithms based on VM consolidation for cloud computing: Comparisons and evaluations**. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)* (pp. 489–498). IEEE; (2020).
Publisher Full Text

20. Khan A, Lee C-G, Hamandawana P, Park S, Kim Y: **A robust fault-tolerant and scalable cluster-wide deduplication for shared-nothing storage systems**. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (pp. 87–93). IEEE; 2018.
Publisher Full Text

21. Li H, Dong M, Liao X, *et al.*: **Deduplication-based energy efficient storage system in cloud environment.** *The Computer Journal.* 2015; **58**(6): 1373–1383.
Publisher Full Text

22. Lin B, Zhu F, Zhang J, *et al.*: **A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing.** *IEEE Transactions on Industrial Informatics.* 2019; **15**(7): 4254–4265.
Publisher Full Text

23. Schneider J, Seidel S, Basalla M, *et al.*: **Reuse, reduce, support: Design principles for green data mining.** *Business & Information Systems Engineering.* 2023; **65**: 65–83.
Publisher Full Text

24. Garrido J, Darriba L, Sánchez-Expósito S, *et al.*: **Toward a Spanish SKA Regional Centre fully engaged with open science.** *J Astron Telesc Instrum Syst.* 2021; **8**(1): 011004.
Publisher Full Text

25. Bonnarel F, Salgado J, Allen M, Barnsley R, Baumann M, Boch T, *et al.*: **Prototyping access from visualisation tools to SKA science images and cubes stored in a Rucio data lake through IVOA discovery and access services**. In Jacques A, Seaman R, Gandilo N, *et al*. (Eds.), *Astronomical Data Analysis Software and Systems XXXIII*. 2025; **541**: 72.
Publisher Full Text

26. Comrie A, Wang K-S, Hwang Y-H, *et al.*: **CARTA: The Cube Analysis and Rendering Tool for Astronomy (4.1.0) [Software].** *Zenodo.* 2024.
Publisher Full Text

27. Ponelat JS, Rosenstock LL: **Designing APIs with swagger and OpenAPI (424).** 2022.

28. Voron F: *Building data science applications with FastAPI*: Packt Publishing; 2023.

29. Parra-Royón M: **manuparra/preparedata-tests: SRCNet preparedata tests v1.0.2 [Software].** *Zenodo.* 2025.
Publisher Full Text

Open Peer Review

Current Peer Review Status: ? ?

Version 1

Reviewer Report 20 March 2026

https://doi.org/10.21956/openreseurope.23938.r70163

?

Rana Veer Samara Sihman Bharattej Rupavath
[1] National Louis University, Chicago, Illinois, USA
[2] National Louis University, Chicago, Illinois, USA

The article presents **PrepareData**, a data-delivery service designed for the **SKA Regional Centre Network (SRCNet)** to make datasets stored in **Rucio Storage Elements (RSEs)** accessible within users' working environments without repeatedly copying large files. The work proposes and evaluates two non-copy approaches alongside the copy baseline:

- A **filesystem-based approach** using Linux mechanisms (e.g., **bind mounts**, and/or symbolic linking concepts) to "expose" data into user workspaces.
- A **Kubernetes-based approach** using **Persistent Volumes (PVs)** and **Persistent Volume Claims (PVCs)** to inject datasets into user environments dynamically.

The paper describes architectural integration (e.g., via a gateway component for authorization) and reports performance experiments comparing these strategies in a prototype environment, emphasizing scalability, reduced duplication, and sustainability/"green computing" considerations.

2) Is the work clearly and accurately presented, and does it cite the current literature? **Partly**
What works well

- The core problem is clearly motivated: enabling access to very large datasets in distributed infrastructure without wasteful repeated copying.
- The paper provides useful architectural/workflow figures and concrete implementation detail (not purely conceptual).

What needs improvement (must address)

- **References appear inconsistent/incomplete in format** in places (per the review text), and some citations may lack basic bibliographic information.
- The reviewer flags that **some references are preprints (arXiv) rather than peer-reviewed sources and** suggests replacing them where possible with peer-reviewed, published literature.

How authors can address

- Standardize reference formatting (journal, year, volume/issue, pages, DOI where available).
- Replace arXiv/preprints with peer-reviewed versions when they exist; if none exist, justify

why the preprint is used and ensure complete metadata.

3) Is the study design appropriate and does the work have academic merit? **Yes**
Strengths
- The study tackles a real, relevant systems problem in scientific data infrastructure.
- The solution is practical and aligns with established "move compute to data" concerns and scalable data-management needs.
- The architecture and implementation are described in enough detail to be useful to practitioners.

Optional strengthening
- Make the contribution claims explicit (see next section).

4) Are sufficient details of methods and analysis provided to allow replication by others? **Partly**

What is present
- Implementation details: service implemented in Python (FastAPI), asynchronous processing via Celery, API described, integration patterns described.
- Performance experiments are described (copy timing vs dataset size; "storms" of mount operations and PV/PVC provisioning), including hardware/software context (e.g., Kubernetes version, CephFS backing).

What needs improvement (must address)
- The reviewer indicates the manuscript needs **a clearer, explicit objectives/contributions section**, preferably near the end of the Introduction.
- The reviewer also suggests **limitations must be clearly presented**, rather than implied—this affects how replicable and scoped the results appear.

How authors can address
- Add a short subsection: **"Objectives and Contributions"** (what exactly is evaluated; what is claimed).
- Add a subsection near the end (before Conclusions or within it): **"Limitations"** (prototype constraints, single-node setup, what was not evaluated such as multi-site behavior, security enforcement validation, lifecycle enforcement in practice, etc.).
- Include a replication checklist: versions, configuration parameters, key scripts/commands, and where to find them (if not already clearly consolidated).

5) Statistical analysis and interpretation: **Not applicable**
The work is primarily systems/performance benchmarking; the selected response in the document is "Not applicable."
(If the venue expects statistical treatment: authors could still add uncertainty measures - e.g., variance/CI across repeated runs - but that would go beyond what the reviewer marked.)

6) Are all the source data underlying the results available to ensure full reproducibility? **Partly**
What is present
- A **Data availability** section exists and points to a **Zenodo deposit** that includes raw data files (e.g., CSVs for copy time, mount benchmarks, PVC benchmarks).
- A **Software availability** section exists, also pointing to Zenodo and a public GitHub repository.

Why only "Partly" (likely)

- Even with data and code shared, full reproducibility often also requires clear environment/configuration capture (cluster settings, CephFS/CSI config, resource limits, etc.). The reviewer's "Partly" suggests something is missing or not fully transparent.

How authors can address (must address)

- Add a "Reproducibility" paragraph listing:
  - exact versions (Kubernetes, CSI driver, CephFS, OS/kernel),
  - cluster topology (nodes, CPU/RAM, network assumptions),
  - key configuration settings and how to run the benchmarks end-to-end.
- Ensure the repository/Zenodo package contains (or links to) scripts that reproduce tables/figures directly from raw data.

7) Are the conclusions adequately supported by the results? **Partly**

Main issue (must address)

- The review explicitly notes that the **performance comparison wording (especially Table 4)** risks sounding more definitive/universal than the experimental setup supports.
- The tests evaluate different overhead types (copy time vs mount operations vs control-plane provisioning latency), so they are **not strictly apples-to-apples**; the paper itself even notes they are "not directly comparable," but the summary framing may still read like a universal ranking.

How authors can address

- Reframe conclusions to emphasize **trade-offs and context-specific suitability**, not "winner" claims.
- Adjust Table 4 wording (and surrounding text) to avoid universal performance rankings; tie each method to its intended operational context and what exactly was measured.
- Explicitly state what is demonstrated (prototype performance trends) and what is not (full production-scale multi-site evaluation).

Points that must be addressed to make the article scientifically sound (per the review's reservations)

1. **Add an explicit statement of objectives and main contributions** (near end of Introduction).
2. **Add a clear limitations section** (before/within Conclusions).
3. **Fix and standardize references**; ensure complete bibliographic info and replace preprints with peer-reviewed sources where possible.
4. **Clarify the performance comparison framing** (especially Table 4 and related wording) so it does not read as a universal ranking beyond the scope of the experiments.
5. **Improve reproducibility documentation** beyond sharing data/code (environment/config details and clear "how to reproduce" instructions).

Is the work clearly and accurately presented and does it cite the current literature?

Partly

Is the study design appropriate and does the work have academic merit?

Yes

Are sufficient details of methods and analysis provided to allow replication by others?

Partly

If applicable, is the statistical analysis and its interpretation appropriate?
Not applicable

Are all the source data underlying the results available to ensure full reproducibility?
Partly

Are the conclusions drawn adequately supported by the results?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: ICT; Management Information Systems; Decision Support Systems; Business Analytics; Data Analytics; Digital Transformation.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 06 Apr 2026

Manuel Parra Royón

We sincerely thank the reviewer for the detailed and constructive comments, which have helped us improve the clarity, methodological transparency, and reproducibility of the manuscript. In response to the comments, all bibliographic references have been carefully revised and updated to follow a more homogeneous citation style, including, whenever possible, the complete metadata requested. In addition, all previous references to arXiv preprints have been replaced by their corresponding peer-reviewed journal articles or conference proceedings whenever published versions were available. We have added a new paragraph at the end of the Introduction explicitly clarifying the main objectives of the study and its concrete scientific and technical contributions. Likewise, an additional paragraph has been incorporated into the Conclusions to explicitly discuss the limitations of the current work, making it easier for readers to understand how these limitations may affect both the reproducibility of the proposed approach and the interpretation of the reported results. Several new paragraphs have been incorporated into the Results section describing in detail the experimental and deployment environment required to reproduce the tests. This environment is now fully specified to facilitate complete reproducibility of the reported experiments, including a list of the required components and configuration items needed to deploy and execute the tests. Additionally, the outputs generated by the testing framework, which are publicly available through Zenodo and the associated GitHub repository, are directly exported in CSV format, corresponding to the data included in the manuscript. The Zenodo record itself also provides the execution instructions required to reproduce the test workflow. We have introduced an explicit clarification stating that the comparison presented in Table 4 should not be interpreted as a universal ranking of approaches, but rather as a qualitative and context-dependent evaluation performed within

a specific testbed scenario. Finally, to reinforce the scope and current maturity of the work, the Conclusions now include an additional paragraph explicitly describing the type of development represented by this contribution and its present limitations, helping delimit the current applicability of the proposed solution.

Competing Interests: No competing interests were disclosed.

Reviewer Report 12 March 2026

https://doi.org/10.21956/openreseurope.23938.r70476

?

Alexandra Theodoropoulou
[1] University of Patras, Patras, Greece
[2] University of Patras, Patras, Greece

This is a real technical contribution with practical value for SRCNet and the paper is not weak in concept. The core problem is well motivated, namely how to expose RSE-resident data to user workspaces without repeated copying in a heterogeneous environment of standalone and Kubernetes-based services. The architecture and workflow figures are also useful and the paper gives concrete implementation detail rather than staying abstract. That said, the manuscript still needs certain revisions before final indexing.

The paper needs a brief, explicit paragraph, preferably near the end of the Introduction, that clearly states the study objectives and the main contributions of the paper, instead of mentioning them implicitly and in a narrative form. The same goes for the limitations of the study; they need to be clearly presented, preferably at the end of the paper (before Conclusions or as a subsection in Conclusions.)

One point that needs some further clarification is the way the performance comparison is presented, especially in Table 4, because the current wording can sound broader and more definitive than the actual evaluation setup supports. This does not mean that the reported results are invalid; rather, the concern is that the three approaches were tested under a limited prototype environment and do not measure exactly the same type of overhead, so they should not read as a universal ranking. A better way to present this would be to soften the comparative wording and explain more clearly that the findings reflect the specific testbed and mainly show different trade-offs and practical use cases, rather than one method being generally superior in all settings.

The discussion of access control and operational safeguards would benefit from clearer wording, because some parts of the manuscript may give the impression that these aspects were fully

validated in practice, while they are mainly presented at the architectural level. This does not mean that the design is not sound; the concern is simply that the current evaluation focuses on performance and deployment behavior rather than on security enforcement, revocation or lifecycle control. A clearer presentation would be to state more explicitly that these features are part of the proposed system design and intended management model, while the present experiments demonstrate functional integration and performance under the tested setup.

The references seem inconsistent in their format, lacking basic information. Also, there are a few preprints/papers that have not been peer-reviewed (arXiv database). Authors need to find all the information required for their references (e.g. ref no22) and replace the preprints with peer-reviewed, published papers.

Is the work clearly and accurately presented and does it cite the current literature?
Partly

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Partly

If applicable, is the statistical analysis and its interpretation appropriate?
Not applicable

Are all the source data underlying the results available to ensure full reproducibility?
Partly

Are the conclusions drawn adequately supported by the results?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: ICT, Management Information Systems, Decision Support Systems, Business Analytics, Data Analytics, Digital Transformation

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 06 Apr 2026

Manuel Parra Royón

We sincerely thank the reviewer for the careful reading of the manuscript and for the constructive observations provided. Following these comments, we have improved the wording surrounding the context of Table 4, explicitly reinforcing that the comparison does not address security enforcement, revocation mechanisms, or lifecycle control aspects, but

is instead focused on the performance behaviour of each preparation flavor under the specific experimental conditions considered. The paragraph preceding Table 4 has been revised to clearly state that the intention is not to establish that one study or approach is universally better than another, but rather to provide a quantitative and context-dependent comparison within the specific testbed used in this work. In addition, we have strengthened the Introduction by adding an explicit paragraph describing the main objectives and the specific contributions of the work. Likewise, the Conclusions section now includes a dedicated discussion of the current limitations of the study. These additions clarify that the main scope of the manuscript is centred on the design, deployment, and performance evaluation of the proposed solutions, rather than on the full lifecycle management of prepared datasets or their associated security models. Finally, all bibliographic references have been carefully revised and updated to follow a more homogeneous citation format, including, whenever possible, the complete metadata requested. Furthermore, all citations previously referring to arXiv preprints have been replaced by their corresponding peer-reviewed journal articles or conference proceedings whenever published versions were available.

Competing Interests: No competing interests were disclosed.