The Journal of Open Source Software

Discovering the SUPER in computing - dagster-slurm for reproducible research on HPC

Hernan Picatto[2], **Maximilian Heß**[2], **Georg Heiler**[1,2], and **Martin Pfister**[3]

1 Complexity Science Hub Vienna (CSH) **2** Austrian Supply Chain Intelligence Institute (ASCII) **3** AI Factory Austria

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Christoph Junghans

Reviewers:

- @MoraruMaxim
- @jan-janssen

Submitted: 31 October 2025
Published: unpublished

In partnership with

AMERICAN ASTRONOMICAL SOCIETY

This article and software are linked with research article DOI 10.3847/xxxxx, published in the Journal of Open Source Software.

Summary

Dagster is a modern data orchestrator that emphasises reproducibility, observability, and a strong developer experience (Elementl, 2024b). In parallel, most high-performance computing (HPC) centres continue to rely on Slurm for batch scheduling and resource governance (Yoo et al., 2003). The two ecosystems rarely meet in practice: Dagster projects often target cloud or single-node deployments, while Slurm users maintain bespoke submission scripts with limited reuse or visibility. This paper introduces **dagster-slurm**, an open-source integration that allows the same Dagster assets to run unchanged across laptops, CI pipelines, containerised Slurm clusters, and Tier-0 supercomputers. The project packages dependencies with Pixi (prefix.dev, 2024), submits workloads through Slurm using Dagster Pipes (Elementl, 2024a), and streams logs plus scheduler metrics back to the Dagster UI.

The key contribution is a unified compute resource (`ComputeResource`) that hides SSH transport (including password-only jump hosts and OTP prompts), dependency packaging, and queue configuration while still respecting Slurm's scheduling semantics. The project ships two production-ready execution modes—`local` for laptop/CI development and `slurm` for one-job-per-asset submissions—and two stable launchers: Bash for script-based workloads and Ray for multi-node distributed computing. Experimental support for Spark, session-based allocation reuse, and heterogeneous jobs is under active development.

Statement of Need

Research software engineers (RSE) and data scientists increasingly face cross-environment workflows. The 22nd edition of the Research Software Engineering (RSE) International Survey reports that every participating country now marks HPC as an important RSE skill (Hettrick et al., 2022). Reproducibility challenges in HPC environments are well documented, and Antunes et al. (Antunes & Hill, 2024) provide comprehensive coverage of issues unique to HPC in their survey. Courtes et al. (Courtès, 2022) further examine the tension between reproducibility and performance, demonstrating that these goals need not be mutually exclusive.

The 2024 Community Workshop on Practical Reproducibility in HPC produced a comprehensive report highlighting cost-effective reproducibility challenges and the need for tools that bridge development and production environments (Keahey et al., 2025). A rich ecosystem of Python-based HPC workflow tools already exists. Parsl (Babuji et al., 2019) provides a parallel scripting library with multi-site execution, dependency tracking, and SSH-based remote submission. Executorlib (Janssen et al., 2025) extends Python's standard `concurrent.futures.Executor` to HPC schedulers, offering per-function resource control, local-to-cluster portability, and support for heterogeneous workloads combining MPI and GPU libraries. Jobflow (Rosen et al., 2024) manages complex computational workflows with decorator-based task definitions, and PSI/J (Hategan-Marandiuc et al., 2023) offers a portable submission interface across schedulers. The Common Workflow Language community maintains an extensive catalogue of further systems (Common Workflow Language Community, 2024). These frameworks provide mature solutions for constructing and executing task graphs on HPC resources, often including local execution modes for rapid prototyping.

dagster-slurm does not aim to replace any of these tools; it occupies a complementary niche. HPC workflow managers typically orchestrate *compute tasks* within the HPC ecosystem—submitting jobs, tracking task dependencies, and managing scheduler resources on one or more clusters. These task-based approaches work well for individual researchers and small teams who define end-to-end pipelines in a single codebase. A data orchestrator like Dagster operates at a different level: it models the dataflow as a *graph of persistent assets* (datasets, models, tables) rather than a graph of tasks to execute. This asset-based paradigm makes the shared state of a data platform explicit—every asset has a declared type, upstream dependencies, and freshness expectations (Elementl, 2024b)—which streamlines collaboration when many contributors work on the same dataflow graph, because changes to one asset's logic automatically propagate through the dependency structure without requiring coordination across teams. Dagster manages the end-to-end dataflow across heterogeneous, polyglot infrastructure—cloud object stores, on-premise databases, container platforms, and HPC clusters alike—while providing lineage tracking, a web UI for operational monitoring, scheduling, and alerting. dagster-slurm bridges these two worlds by extending Dagster's control plane to Slurm-managed hardware. This serves research teams whose pipelines span multiple compute tiers: data ingestion and preprocessing on institutional servers or cloud machines, GPU-intensive model training on an HPC partition, and downstream analytics or publication steps elsewhere. Teams already invested in Dagster for the non-HPC parts of their pipeline can reach supercomputers without adopting a second orchestration framework, and teams using HPC workflow managers for the compute-intensive stages can wrap those invocations inside Dagster assets to gain lineage and observability over the full dataflow. This lowers the entry barrier to sovereign AI infrastructure: data engineering and machine-learning teams without a traditional HPC background can leverage publicly funded European supercomputers through the same tooling they already use for cloud workloads, rather than having to acquire Slurm expertise first. **dagster-slurm** was created to:

- Lower the barrier to sovereign HPC adoption. HPC environments are complex; dagster-slurm provides the familiar Dagster developer experience so that beginners can reach production supercomputers without mastering Slurm scripting first.
- Enable rapid local prototyping. The same asset code runs locally without queues, eliminating long wait times during development, and can be re-pointed to a different HPC cluster when the primary system is congested.
- Orchestrate across compute tiers. A typical scientific pipeline does not run entirely on HPC GPUs—cheap preprocessing can happen on a cloud VM while only the training step targets the supercomputer. dagster-slurm keeps the full dataflow graph visible and schedulable in one place.
- Provide batteries-included environment packaging. Pixi and pixi-pack produce reproducible, self-contained bundles deployable in air-gapped HPC centres.
- Surface structured observability. Slurm job IDs, CPU efficiency, memory usage, and live log streams appear directly in the Dagster UI, benefiting beginners navigating HPC for the first time and experts managing production workloads alike.
- Encourage research software engineering best practices as outlined by Eisty et al. (Eisty et al., 2025), covering planning, testing, documentation, and maintenance across the development lifecycle.

System Overview

The integration is composed of three layers:

1. **Resource definitions** – `ComputeResource`, `SlurmResource`, `SlurmSessionResource`, and `SSHConnectionResource` are Dagster `ConfigurableResource` objects. They encapsulate queue defaults, SSH authentication (including ControlMaster fallback, password-based jump hosts, and interactive OTP prompts), and execution modes.
2. **Launchers and Pipes clients** – Launchers (Bash, Ray) translate payloads into execution plans. The Slurm Pipes client handles environment packaging (on demand or via pre-deployed bundles), transfers scripts, triggers `sbatch` or session jobs, and streams logs/metrics back through Dagster Pipes (Elementl, 2024a). Custom launchers can be added by extending the `ComputeLauncher` base class.
3. **Operational helpers** – Environment deployment scripts, heterogeneous job managers, metrics collectors, and SSH pooling utilities target HPC constraints such as login-node sandboxes, session allocations, and queue observability.

This layered approach keeps Dagster's user code agnostic to the underlying transport while retaining the full control plane visibility of the orchestrator.

dagster-slurm builds on Dagster's `ConfigurableResource` and Pipes protocols rather than on IOManagers. The system automatically transfers payload scripts and execution environments (via pixi-pack and SCP) and streams structured messages, metadata, and logs back through Dagster Pipes. Data management is deliberately left to the user because I/O strategies vary widely across HPC sites and use cases. In local development mode, datasets are typically small and reside on the developer's machine, keeping the prototyping loop fast. In production, data usually lives on shared parallel filesystems (GPFS, Lustre) or on S3-compatible object stores co-located with the cluster. The recommended pattern is a deployment-mode-aware path switch—for example an environment variable or Dagster configuration value that resolves to a local directory during development and to the shared filesystem mount on the supercomputer—so that asset code remains unchanged across tiers. This is intentional: HPC workloads typically operate on large datasets where automatic serialization across network boundaries would be impractical, and researchers retain full control over data locality and I/O strategy.

As illustrated in Figure 1, the same scalable job can follow multiple paths: direct local execution for development, automated testing through CI/CD chains, or production deployment to HPC clusters via SSH-accessible edge nodes.

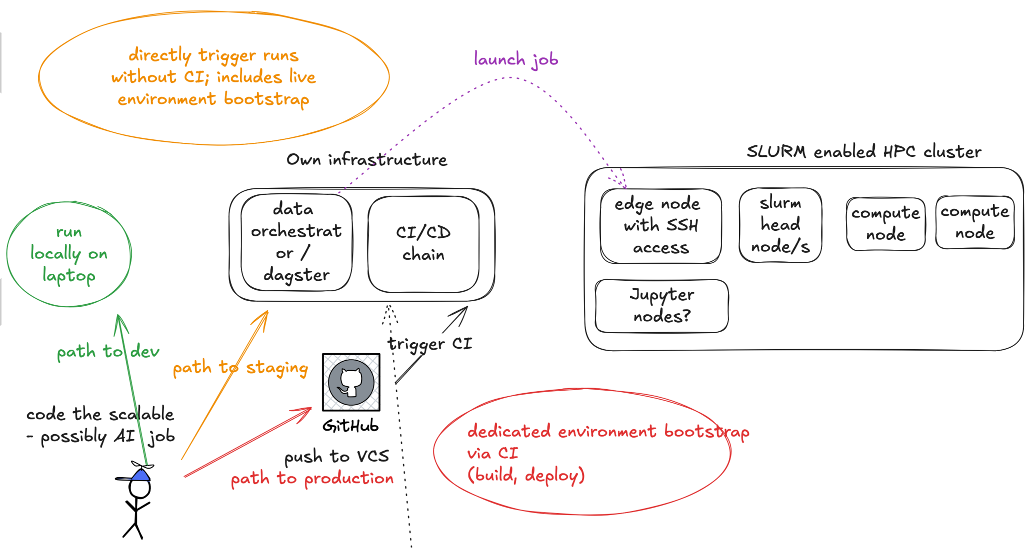

Figure 1: Architecture overview showing the progression from local development to production HPC deployment. The workflow enables researchers to code locally, test through CI/CD pipelines, and deploy to Slurm-enabled clusters with minimal changes.

Minimal usage example

```python
import dagster as dg
from dagster_slurm import (
    ComputeResource,
    ExecutionMode,
    RayLauncher,
    SlurmQueueConfig,
    SlurmResource,
    SSHConnectionResource,
)

ssh = SSHConnectionResource.from_env(prefix="SLURM_EDGE_NODE")
slurm = SlurmResource(
    ssh=ssh,
    queue=SlurmQueueConfig(partition="batch",
      time_limit="02:00:00", cpus=8, mem="32G"),
    remote_base=f"/home/{ssh.user}/dagster_runs",
)

compute = ComputeResource(
    mode=ExecutionMode.SLURM,
    slurm=slurm,
    default_launcher=RayLauncher(num_gpus_per_node=1),
)

@dg.asset(required_resource_keys={"compute"})
def train_model(context: dg.AssetExecutionContext):
    payload = dg.file_relative_path(__file__, "../workloads/train.py")
    completed = context.resources.compute.run(
        context=context,
        payload_path=payload,
        resource_requirements={"framework": "ray",
          "cpus": 32, "gpus": 1, "memory_gb": 120},
        extra_env={"EXPERIMENT": context.run.run_id},
    )
    yield from completed.get_results()
```

Local development simply swaps `ExecutionMode.SLURM` for `ExecutionMode.LOCAL`. The example project bundled with the repository demonstrates this workflow, complete with Dockerised Slurm nodes for integration testing. Session reuse and heterogeneous jobs remain under active development; early adopters can track progress in the repository milestones.

Evaluation

We validate the approach along three dimensions:

- **Reproducibility** – Integration tests run inside GitHub Actions using a containerised Slurm cluster. The pipeline provisions the environment with Pixi, deploys it once via `pixi run deploy-prod-docker`, and then runs Dagster assets through all four execution modes. This continuous integration approach aligns with emerging best practices for reproducible HPC workflows (Hayot-Sasson et al., 2025).
- **HPC readiness** – The project has been exercised on academic clusters such as VSC-5 (Austria) and Leonardo (Italy). SSH ControlMaster fallbacks, password-based jump hosts, `.bashrc` hygiene, queue/QoS/reservation overrides, and verification snippets (`squeue`,

`scontrol`) are documented for both sites.

- **Observability** – Slurm job IDs, CPU efficiency, memory, and node-hours are exposed as Dagster metadata entries, while Ray clusters stream their stdout/stderr back through Pipes. This enables conventional Dagster asset checks and alerting to operate unchanged.

Impact and Future Work

dagster-slurm lowers the barrier for research teams to adopt modern data orchestration on top of established HPC schedulers. By eliminating duplicated scripts and surfacing rich observability, the integration reduces operational toil and shortens iteration loops. Future work focuses on:

- Deepening heterogeneous job support (automatic fusion of dependent assets, richer allocation policies).
- Maturing the Spark launcher from its current experimental state to production readiness and adding support for further frameworks (MPI, GPU-accelerated libraries).
- Exploring pilot-job back ends (e.g., RADICAL-Pilot, QCG) for finer-grained scheduling inside allocations, and non-interactive OTP integrations for strict MFA environments.

Community contributions—issue reports, cluster-specific recipes, and new launchers—are actively encouraged at https://github.com/ascii-supply-networks/dagster-slurm. Questions and discussions are welcome on GitHub Discussions.

Acknowledgements

We thank the operations teams at the Austrian Scientific Computing (ASC) (VSC-5) and CINECA's Leonardo supercomputer for early feedback, and the Dagster community for discussions around orchestrating HPC workloads. Funding and in-kind support were provided by the Complexity Science Hub Vienna and the Austrian Supply Chain Intelligence Institute. This work was completed in part at the EUROCC AI HACKATHON 2025, part of the Open Hackathons program. The authors would like to acknowledge OpenACC-Standard.org for their support.

References

Antunes, B., & Hill, D. R. C. (2024). Reproducibility, replicability and repeatability: A survey of reproducible research with a focus on high performance computing. *Computer Science Review*, *53*, 100655. https://doi.org/10.1016/j.cosrev.2024.100655

Babuji, Y., Woodard, A., Li, Z., Katz, D. S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J. M., Foster, I., Wilde, M., & Chard, K. (2019). Parsl: Pervasive parallel programming in python. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 25–36. https://doi.org/10.1145/3307681.3325400

Common Workflow Language Community. (2024). *Existing workflow systems*. https://s.apache.org/existing-workflow-systems

Courtès, L. (2022). Reproducibility and performance: Why choose? *Computing in Science & Engineering*, *24*(3), 77–80. https://doi.org/10.1109/MCSE.2022.3165626

Eisty, N. U., Bernholdt, D. E., Koufos, A., Luet, D. J., & Mundt, M. (2025). *Ten essential guidelines for building high-quality research software*. https://doi.org/10.48550/arXiv.2507.16166

Elementl, Inc. (2024a). *Dagster pipes documentation*. https://docs.dagster.io/concepts/dagster-pipes/pipes-overview

Elementl, Inc. (2024b). *Dagster: Data orchestration platform*. https://dagster.io/

Hategan-Marandiuc, M., Merzky, A., Collier, N., Maheshwari, K., Ozik, J., Turilli, M., Wilke, A., Wozniak, J. M., Chard, K., Foster, I., Ferreira da Silva, R., Jha, S., & Laney, D. (2023). PSI/J: A portable interface for submitting, monitoring, and managing jobs. *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10. https://doi.org/10.1109/e-Science58273.2023.10254912

Hayot-Sasson, V., Hudson, N., Bauer, A., Gonthier, M., Foster, I., & Chard, K. (2025). *Addressing reproducibility challenges in HPC with continuous integration*. https://doi.org/10.48550/arXiv.2508.21289

Hettrick, S., Bast, R., Crouch, S., Wyatt, C., Philippe, O., Botzki, A., Carver, J., Cosden, I., D'Andrea, F., Dasgupta, A., Godoy, W., Gonzalez-Beltran, A., Hamster, U., Henwood, S., Holmvall, P., Janosch, S., Lestang, T., May, N., Philips, J., … Zhang, Q. (2022). *International RSE survey 2022* (Version v0.9.3). Zenodo. https://doi.org/10.5281/zenodo.7015772

Janssen, J., Taylor, M. G., Yang, P., Neugebauer, J., & Perez, D. (2025). Executorlib – up-scaling python workflows for hierarchical heterogeneous high-performance computing. *Journal of Open Source Software*, *10*(108), 7782. https://doi.org/10.21105/joss.07782

Keahey, K., Richardson, M., Tolosana Calasanz, R., Hunold, S., Lofstead, J., Malik, T., & Perez, C. (2025). *Report on challenges of practical reproducibility for systems and HPC computer science*. Zenodo. https://doi.org/10.5281/zenodo.15306610

prefix.dev. (2024). *Pixi: Reproducible package environments*. https://github.com/prefix-dev/pixi

Rosen, A. S., Gallant, M., George, J., Riebesell, J., Saez, H., Ganose, A. M., & Jain, A. (2024). Jobflow: Computational workflows made simple. *Journal of Open Source Software*, *9*(93), 5995. https://doi.org/10.21105/joss.05995

Yoo, A. B., Jette, M. A., & Grondona, M. (2003). SLURM: Simple linux utility for resource management. In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job scheduling strategies for parallel processing* (Vol. 2862, pp. 44–60). Springer. https://doi.org/10.1007/10968987_3