# Union of Containers: Layered Software Stacks with Cross-Container Dependency Reuse for HPC

Parmanand Sinha
Research Computing Center, University of Chicago
Chicago, IL, USA
pnsinha@uchicago.edu

***Abstract*—E4S publishes one container image per accelerator target, and each image repeats the CPU-generic majority of the stack. In this paper, we describe a layered alternative: a shared provider holds the architecture-generic dependencies, thin specialized layers hold only GPU-compiled packages and reuse the provider's builds, and the layers compose at run time through absolute RPATH entries under fixed paths. Results indicate that a specialized layer compiles one of twenty buildable packages. At one site, splitting the stack costs five files and 26 ms of startup, while each further GPU target adds 1,412 files where a monolithic build adds 10,772. Rebuilding a layer is 1.8 to 2.8 times faster, and application performance is unchanged.**

***Index Terms*—containers, E4S, Spack, RPATH, dependency reuse**

## I. Introduction

E4S publishes one image per accelerator target [1], [2], each with device code for one architecture (e.g., the `cuda80` image's `libkokkoscore.so` carries sm_80 cubins and no PTX). A center supporting several GPU generations therefore holds several images whose CPU-side contents are identical. The file count forced this work: the `cuda80` image holds 1,826,534 filesystem entries, and unpacking it on Anvil hit a one-million-file scratch quota 69% of the way through.

## II. Design

The mechanism has two requirements. (i) Binaries carry absolute `DT_RPATH` entries under both canonical roots, so the loader resolves cross-container dependencies with no search-path configuration or relocation. (ii) The specialized build consumes the provider's installed packages as already-satisfied dependencies. This work uses Spack's `shared_linking: rpath` and upstream chaining [3]; the compiler, CUDA and MPI come from the base image as externals. Because `ld.so` performs the resolution, the trees are not tied to one runtime [4], [5]. EESSI shares one base through a CVMFS namespace [6] and XaaS defers specialization through IR [7]; this work binds independently built trees and measures the reuse.

## III. Evaluation

We evaluated the pattern with pinned inputs (Spack v1.2.1, `spack-packages` e4s-26.06.1, base image `e4s-cuda-26.06-cuda80`) on Midway3 (GPFS, V100) and Anvil (GPFS, A100 and H100). The provider holds 36 buildable packages; specialized layers hold AMReX for sm_70, sm_80 and sm_90, and heFFTe for sm_80. The monolithic comparator builds the identical 24-spec graph in one tree.

TABLE I
Build time for one more GPU target

| Site, target | Monolithic | Layered | Ratio |
|---|---|---|---|
| Midway3, sm_70 | ∼13 min | ∼5 min | 2.6x |
| Anvil, sm_90 | 857 s | 309 s | 2.8x |
| Anvil, sm_80 | 1,019 s | 564 s | 1.8x |

Each AMReX layer compiled one of twenty buildable packages (95.0% reuse) at both sites, and heFFTe reused twenty of twenty-one. In Table I, the sm_90 monolith also compiled FFTW for the benchmark (2.1x without it); the sm_80 layered figure is whole-job time, so that row likely understates the gap.

On Anvil, one target in the layered arrangement holds 10,777 files (`find -type f`) against 10,772 for the monolith; each further target adds 1,412 instead of 10,772, i.e., 7.6 times fewer. Across the three AMReX layers, 1,397 of 1,412 files are byte-identical.

An AMReX benchmark ran in 0.112 s whether layered, monolithic, or from the stock E4S image on an A100, and within 0.9% on an H100. The second tree adds 26 ms to startup when bound. One provider served two containers at once. The Apptainer-built trees passed a GPU test under Charliecloud unchanged.

## IV. Constraints

Four constraints bound the pattern. (i) Reuse fails across Spack versions because drift in bundled defaults changes dependency hashes. (ii) Spack ignores a standalone `upstreams.yaml`, and the build silently recompiles everything. (iii) Concurrent containers binding one `/tmp` path each need a private `/tmp`. (iv) First-touch listing of the provider on GPFS is 4.4 times slower than warm at both sites, but container startup never walks the tree.

## V. Conclusion

This work contributes to the literature on container-based deployment of HPC software stacks. Splitting the stack costs almost nothing for one target and saves files and build time for every target after that, with no perceptible impact on application performance. The next test is sharing one provider

across multiple nodes. Environments, scripts, and evidence logs are at https://github.com/pnsinha/union-of-containers.

## ACKNOWLEDGMENT

This work was supported by the Better Scientific Software Fellowship Program, a collaborative effort of the U.S. Department of Energy (DOE), Office of Advanced Scientific Research via ANL under Contract DE-AC02-06CH11357 and the National Nuclear Security Administration Advanced Simulation and Computing Program via LLNL under Contract DE-AC52-07NA27344; and by the National Science Foundation (NSF) via SHI under Grant No. 2435328. This work used Anvil through ACCESS allocation CIS261295 (NSF grants #2138259, #2138286, #2138307, #2137603, #2138296) and UChicago RCC resources.

## REFERENCES

[1] J. M. Willenbring, S. S. Shende, and T. Gamblin, "Providing a flexible and comprehensive software stack via Spack, an extreme-scale scientific software stack, and software development kits," *Computing in Science & Engineering*, vol. 26, no. 1, pp. 20–30, 2024.

[2] E4S Project, "E4S: Extreme-scale Scientific Software Stack," 2026. [Online]. Available: https://e4s.io/

[3] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, et al., "The Spack package manager: bringing order to HPC software chaos," in *Proc. SC '15*, 2015, art. 40.

[4] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, art. e0177459, 2017.

[5] R. Priedhorsky and T. Randles, "Charliecloud: unprivileged containers for user-defined software stacks in HPC," in *Proc. SC '17*, 2017, art. 36.

[6] B. Dröge, V. Holanda Rusu, K. Hoste, C. van Leeuwen, A. O'Cais, and T. Röblitz, "EESSI: a cross-platform ready-to-use optimised scientific software stack," *Software: Practice and Experience*, vol. 53, no. 1, pp. 176–210, 2023.

[7] M. Copik, E. Alnuaimi, A. Kamatar, V. Hayot-Sasson, A. Madonna, T. Gamblin, et al., "XaaS containers: performance-portable representation with source and IR containers," in *Proc. SC '25*, 2025.