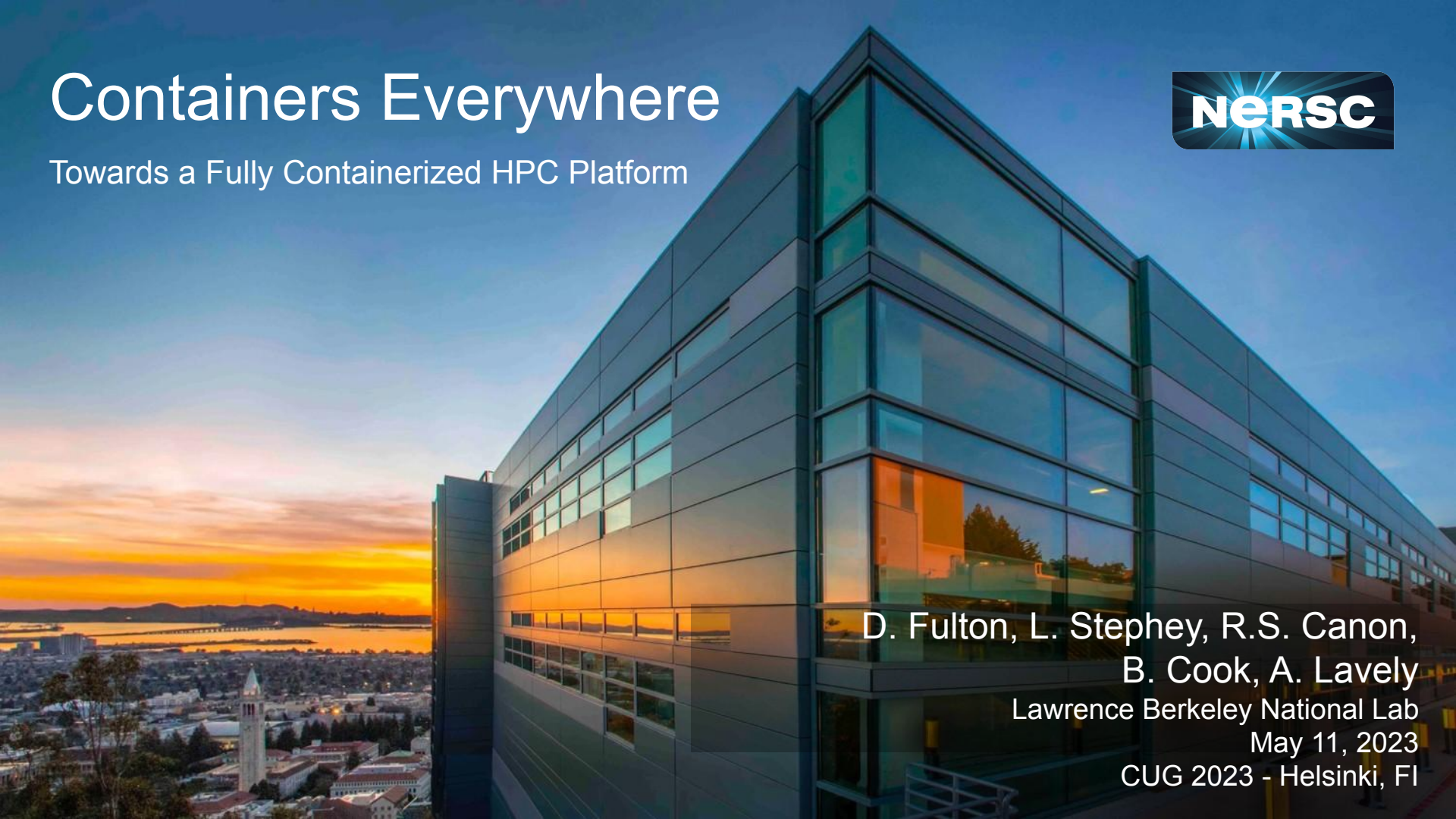
Containers Everywhere

Towards a Fully Containerized HPC Platform



D. Fulton, L. Stephey, R.S. Canon,
B. Cook, A. Lavely
Lawrence Berkeley National Lab
May 11, 2023
CUG 2023 - Helsinki, FI

This work is a team effort

Daniel Fulton
(NERSC)

Laurie Stephey
(NERSC)

Shane Canon
(NERSC)

Brandon Cook
(NERSC)

Adam Lavely
(NERSC)

Special thanks to NERSC colleague Christopher Samuel (Computational Systems Group)
for help installing Slurm 23.02

Overview

- Containers at NERSC today
- NERSC's container wishlist
- *Containers everywhere!* What? Why?
- Design requirements for putting *containers everywhere*
- Implementation examples of putting *containers everywhere*
- Outstanding issues/challenges with *containers everywhere*

Containers at NERSC Today

- User-facing container runtimes (batch jobs and interactive tools)
  - Shifter
  - Podman-HPC
- Packaging complex applications (user or staff managed)
  - CP2k
  - NWChem
  - And more...
- Deployment of HPC system services:
  - Slurm
  - Monitoring
  - Shifter Image Gateway

What's Missing?

- Container orchestration…
  - … for complex user workflows with HPC resources.
  - … for persistent user services with HPC resources.
  - … for staff provided (non-system) HPC services.
- Hidden/seamless containers for novice users.
- Automated continuous deployment for staff provided services.

Containers Everywhere!

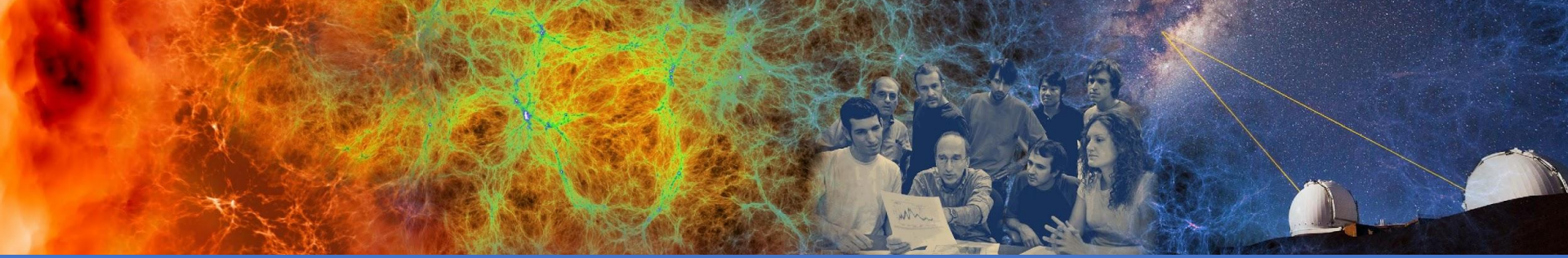
A ***containers everywhere*** HPC strategy is one where the container is the primary building block of software, deployment, and orchestration at all levels of an HPC system.

- Not just the packaging method, but also more flexible and scaleable application architecture and continuous deployment.
- Together, we believe this can provide more flexibility in the HPC system to more easily incorporate new hardware, and support a broader set of science use cases.
- Is *containers everywhere* viable?
  Cloud hyperscalers are already doing it!

Microsoft Azure

amazon web services

Google Cloud Platform



Design Requirements

Stakeholders

- ***HPC Admin*** - An HPC administrator.
- ***HPC Staff Member*** - An HPC systems engineer providing services, software, or other support without root privileges.
- ***Experienced Container User*** - An HPC user who is experienced with building and running containers.
- ***Novice Container User*** - An HPC user who is a novice with the use of containers.

HPC User Motivations

- Easier to install system packages and rootful applications.
- Granular control over performance.
- Enhanced collaborations, controlling and sharing application runtimes and application development environments.
- More stability, flexibility, portability, reproducibility.
- Democratized containers for all users, including novices.

HPC Operator Motivations

- Decouple system maintenance from user environment.
- Easier/more frequent system updates.
- Allow non-root staff to contribute to system configuration.
- Rolling maintenances using orchestration (Kubernetes).
- More granular versioning.
- Synergy with cloud environments:
  - Test HPC environments in cloud
  - Portable system environments
  - Easier cloud bursting

Use Cases

Key Point: There are *many* use cases to satisfy all the way through stack from control plane to user applications.

Reproduce Trad. HPC Functionality

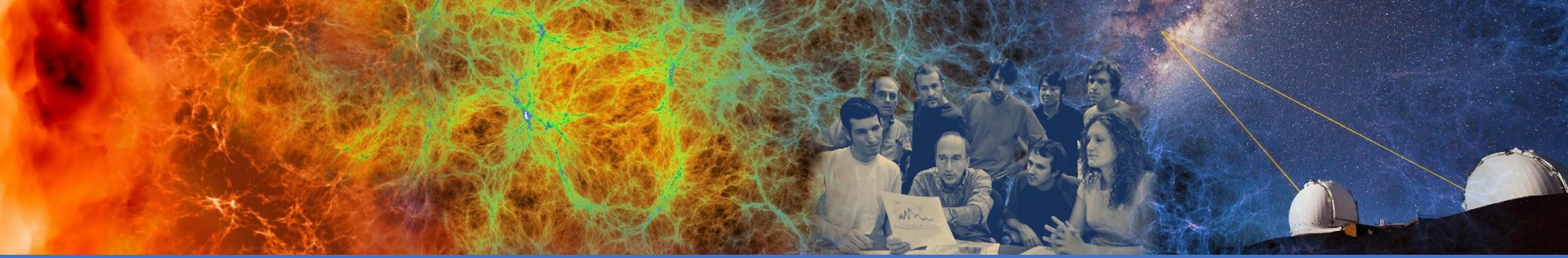
- Interactive login into container.
- Routine file editing
- Code build/compile
- Running a scheduled batch job.
- Debugging/performance analysis
- Run metadata intensive application
- Sharing data with project collaborators
- Compose application runtime environment
- Troubleshoot a user session.

Add Additional Functionality

- Compose an interactive development environment
- Sharing a composed environment
- User-owned persistent complex services.
- Advanced Resource and Access Control
- Rolling HPC Maintenances
- Sandboxing System Updates
- Sandboxing Non-admin staff
- Continuous Deployment of Staff provided software/service

Design Milestones

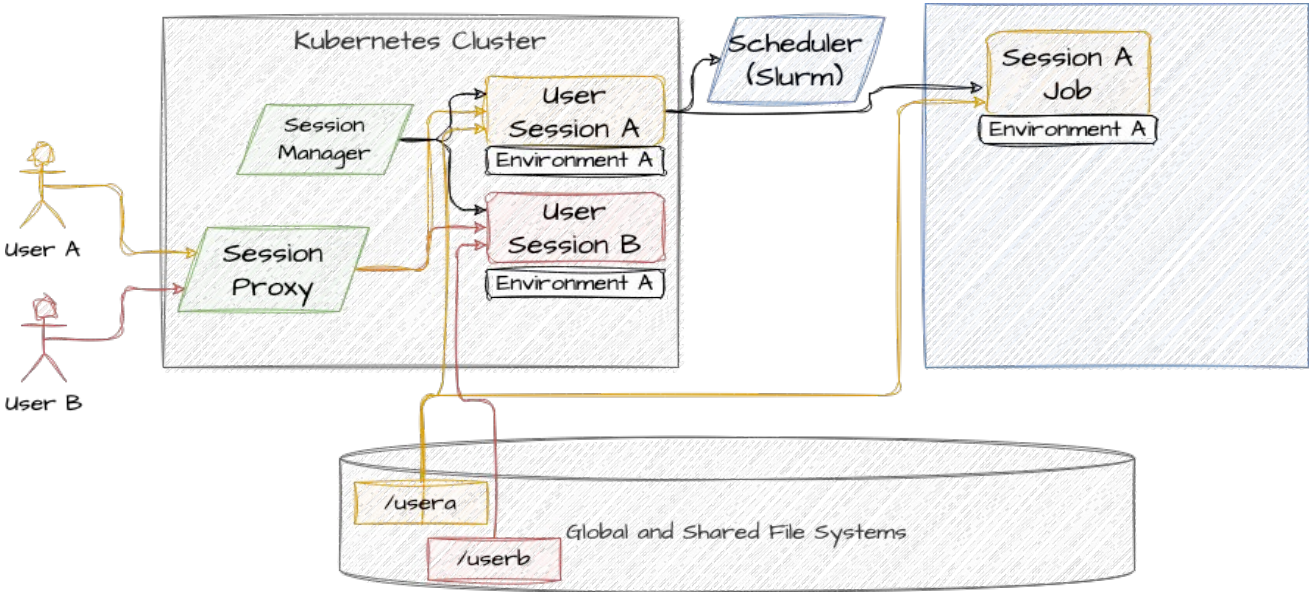
Minimum Viable	Simple, Loveable, Complete	"Kitchen Sink"
Minimal impact to ***Novice Container User***	Users may customize login environment containers	Users can specify resource access control in containers
Secure. All user containers run in user namespaces.	Containers-in-containers	Kubernetes-in-kubernetes
Default login and development environments are in containers	User environment and system software containers are decoupled	Fully containerized system software
Compute jobs in containers.	Containerized service plane (staff deployments)	Containerized service plane (user deployments)
Session management.		



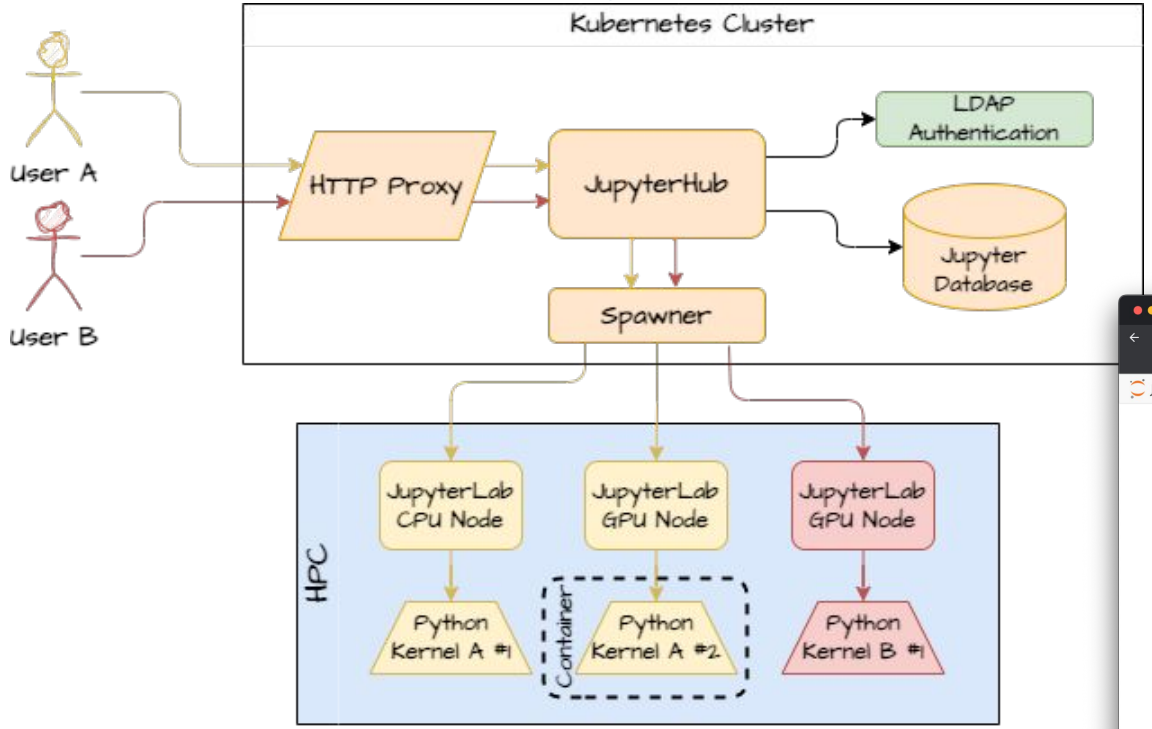
Implementation

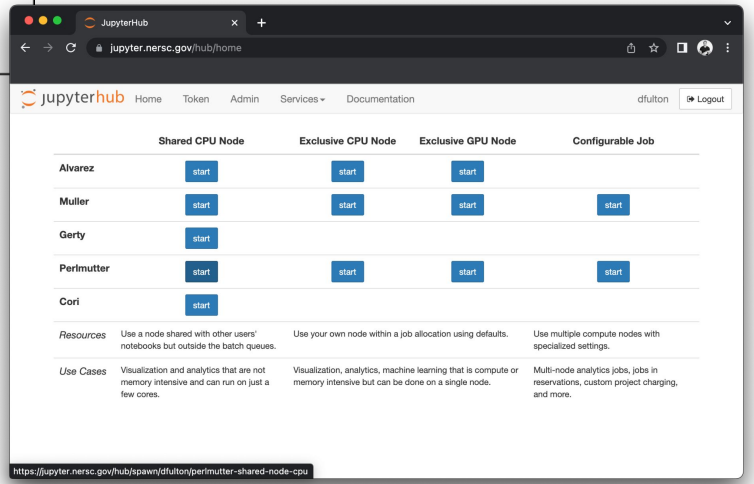
Example: Session Proxy Architecture

Session Proxy and *Session Manager* ensure user session persistence across container rescheduling and logins.

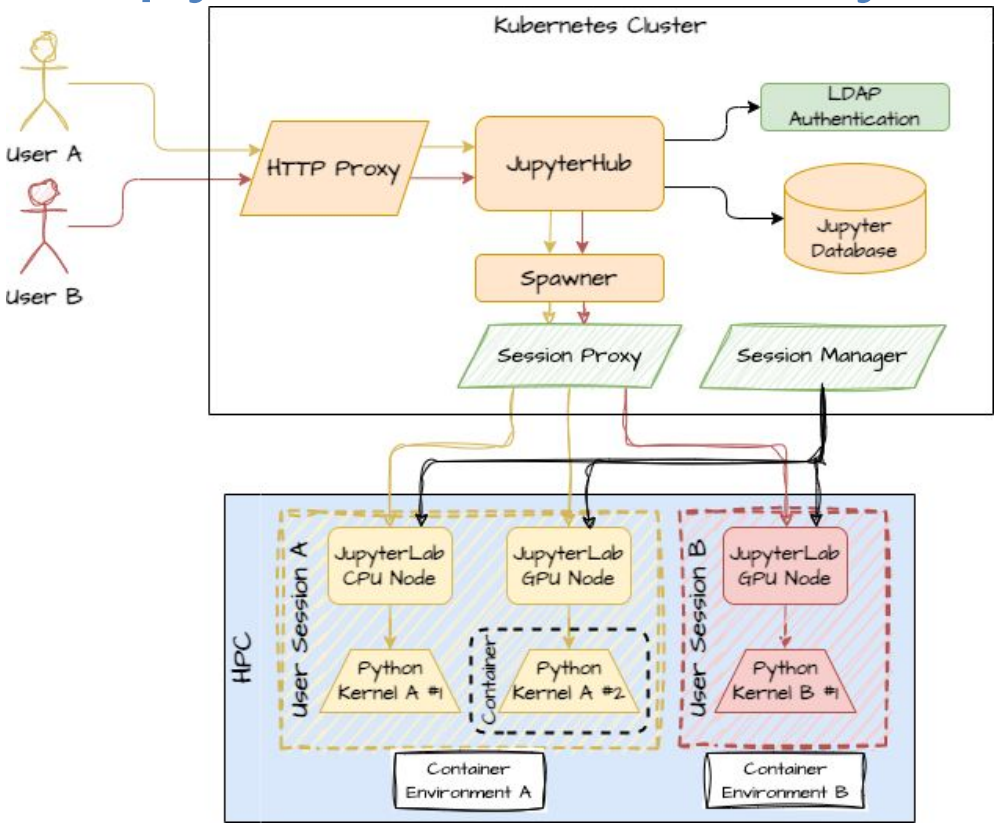


Example: Jupyter Uses Container Today





Example: Jupyter/Session Proxy Integration



Example: Running a Debugger

Key ideas: May require strace to be installed in the image. May also require expertise that novice users won't have. Utility wrapper could help.

Running **`strace`** inside a container:

```
podman-hpc run --rm --volume /tmp:/tmp docker.io/myapp:v1.0 \
    strace -e trace=file -f -o /tmp/podman-myapp-trace \
    python3 -m myapp $(date +%s)
```

Running **`strace`** on a process in another container:

```
podman-hpc run --rm --name=test docker.io/myapp:v1.0 \
    python3 -m myapp
podman-hpc run --rm --volume /tmp:/tmp --pid=container:test \
    --cap-add sys_admin --cap-add sys_ptrace --net=container:test \
    --security-opt=seccomp:unconfined docker.io/mystrace:v1.0 \
    strace -e trace=file -f -o /tmp/podman-myapp-trace -p 1
strace: Process 1 attached
```

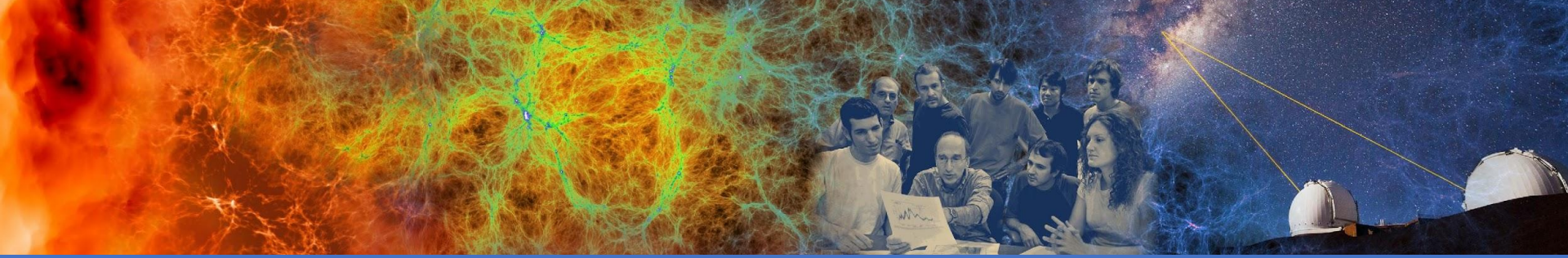
Example: Containers in Containers

Appropriate permissions need to be added to the outer container, as well as a container runtime installation, directly or via runtime hook.

```
perlmutter:~> podman run --cap-add=sys_admin,mknod
    --device=/dev/fuse --security-opt label=disable -it --rm
    quay.io/podman/stable /bin/bash

[root@180f0e912674 /]# podman pull ubuntu

[root@180f0e912674 /]# podman run ubuntu:latest echo 'hello!'

hello!
```

Example: Batch Integration

- Currently, Slurm doesn't extend the **`munge`** perimeter into containers.
- SchedMD is working on several Slurm/Kubernetes interaction models.
- One workaround would be a sidecar container to proxy requests to the workload manager.
- Ultimately, HPC operator has to resolve the philosophical conflict (high utilization vs high availability).



Future Work and Final Thoughts

Difficult Things that Need More Work

- Using Kubernetes with devices in control plane.
- User namespaces make sharing via filesystem difficult.
- Scheduling philosophy (Slurm/Kubernetes integration)
  - High utilization vs high availability
- Software environment composability
  - Probably don’t want monolithic images.
  - Probably don’t want a matrix of images.
  - Runtime composability or tools to let users compose their own.

Plans at NERSC

- Continue to support `shifter`
- Grow user base of `podman-hpc` and gather user-driven use cases with an OCI compliant container runtime. Seek open source solutions.
- Deeper exploration of orchestration and scheduler interaction.
- Develop CI/CD model for staff software middle layer. Add (more) automation into NERSC software operational model.
- Prototype session management plumbing, and/or work with HPE UAIs
- Evaluate limitations for system stack.

Final Thoughts

- Both HPC operators and user could benefit in many different ways from putting *containers everywhere*.
- We realize *containers everywhere* is ambitious (maybe even a bit crazy) and will require a substantial amount of work and collaborative effort.
- But the cloud is already doing it!
- Even if *containers* are **not** adopted *everywhere* in HPC, we are currently leaving a lot on the table. Just trying will lead to improvements.

Thank You!