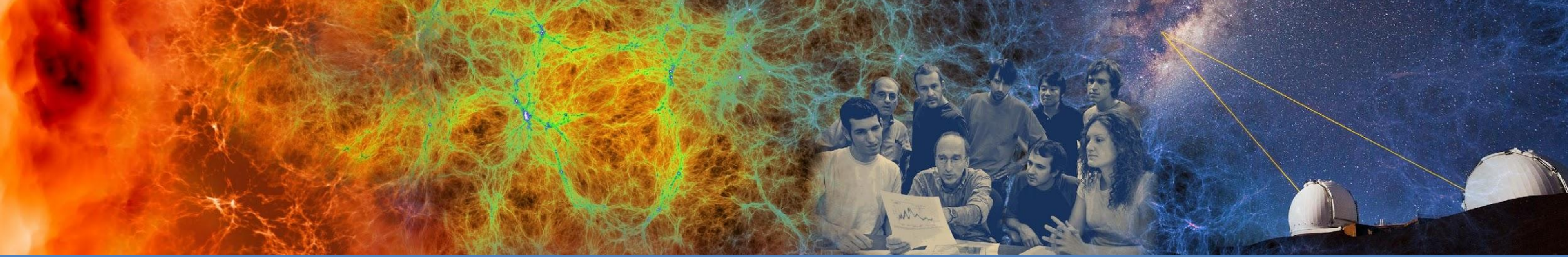
Adopting standardized container runtimes at NERSC

Cray User Group 2022
Monterey

Aditi Gaur, Shane Canon,
Dan Fulton, Laurie Stephey,
Doug Jacobsen
NERSC/LBNL

Outline

- Container environments at NERSC
  - Motivations to replace Shifter
- Bringing Podman to NERSC
- Early insights from scaling experiments
- What we have learned



Containers at NERSC

Key ideas behind HPC runtimes

What do we want from an HPC Container Runtime?

- Security - It needs to work well in a multi-tenant/multi-user environment (e.g. no special privileges for containers).
- Performance - It needs to leverage capabilities without sacrificing performance (e.g. GPUs, Interconnect).
- Scalability - It needs to scale to the full system size without significant impacts.
- Standardization - It should leverage existing standards and integrate easily with the broader ecosystem.
- Example implementations:
  - Shifter (used at NERSC), CharlieCloud, Singularity

What has changed in the past few years

- Improved support for running “root-less” containers (Docker, Podman)
- Modern kernels support broader set of features needed to use modern container runtimes (eg. user namespaces)
- More alignment between “industry” and HPC due to things like AI/ML workloads
- Emergence of Kubernetes and the ecosystem of tools around it.

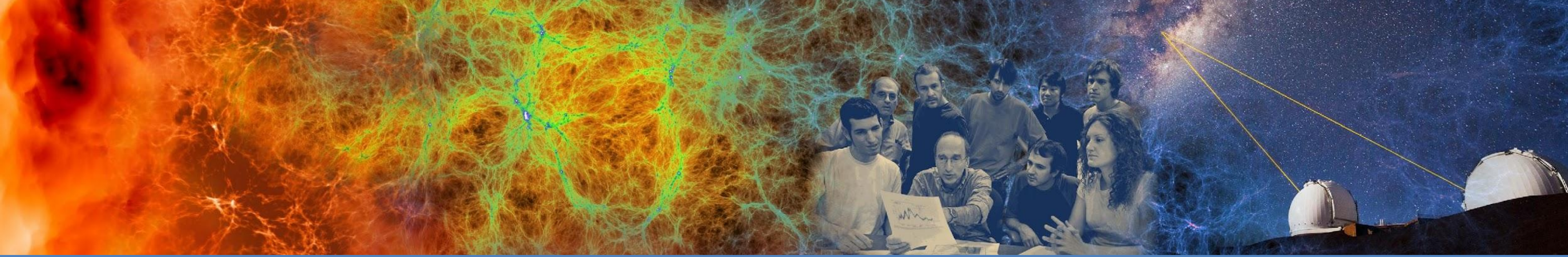
Podman as a potential replacement for Shifter

Why Podman?

- Already has momentum for replacing Docker under the hood in many installations
- Can run fully rootless using user namespaces, uid/gid map, fuse-overlayfs, and other tricks.
- It is a full runtime (e.g. can support local image builds too)
- Open development but with strong Vendor support from IBM/Redhat and others
- Recipes for how to integrate with GPUs and MPI
- Community support and uptake
- Podman leverages OCI runtimes and CRI-O stack.
  - runc/crun OCI runtimes
  - containers/storage library for managing storage
  - containers/image library for managing images

What are the gaps?

- Scalable launch… Images are typically pulled locally and unpacked. Can use “additionalimagestore” but still requires images to be fully unpacked.
- Integration with Slurm (if needed)



Bringing Podman to NERSC

HPC support for Podman

- Podman directly mounts the various layers of the image via fuse-overlayfs
- "Flattening" the image can result in performance improvements on the parallel filesystem
- We made modifications to the underlying storage driver to allow it to mount a squashed image which is created by squashing all the layers of the image together
- Added option "`squashmount`" to the overlay driver allowing it to mount writable/temporary areas over the squashed image.
- This could potentially be used outside of Podman (e.g. Kubernetes deployments)

Podman Setup - A sample `storage.conf`

```
[storage]
  driver = "overlay"
  runroot = "/tmp/user/75535"
  graphroot = "/tmp/images"
  [storage.options]
    ignore_chown_errors = "true"
    mount_program = "/usr/bin/fuse-overlayfs"
    additionalimagestores = ["/mscratch/sd/a/agaur/images",]
    [storage.options.overlay]
    squashmount = "true"
```

Driver must be set to “overlay”

The read/write area needed for running podman is configured to user’s private /tmp area available on the login/compute nodes. Any private r/w location is a good fit here. Shared filesystems do not work for this setting.

Podman setup - storage.conf explained

```
[storage.options]
    ignore_chown_errors = "true"
```

$\longrightarrow$ To make our setup work, this value needs to be set to `"true"`

From the man page:

```
ignore_chown_errors can be set to allow a non privileged user
running with a  single UID within a user namespace to run
containers. The user can pull and use any image even those with
multiple uids.  Note multiple UIDs will be squashed down to the
default uid in the container.  These images will have no
separation between the users in the container. (default: false)
```

Podman setup - Additional Image Stores

```
additionalimagestores = ["/mscratch/sd/a/agaur/images",]
```

User created directory on “Scratch” filesystem or a user owned area on a shared filesystem

From the man page:

```
    additionalimagestores=[]

    Paths to additional container image stores. Usually these
are read/only and stored on remote network shares.
```

Podman setup - Squashmount

```
[storage.options.overlay]
    squashmount = "true"
```

`squashmount` is a NERSC customization to Podman's overlay driver and is not part of an official release yet.

This option when set to "true" tells podman to look for squashed image in its storage area and mount it, instead of mounting overlay layers. This option allows us to mount squashed images from additionalimagestores which are configured to point to a shared filesystem.

Podman User Workflow - Pull

1. Pull the image on a login node.

```
agaur@muller:login01:~> podman pull ubuntu:latest
Trying to pull docker.io/library/ubuntu:latest...
Getting image source signatures
Copying blob 125a6e411906 done
Copying config d2e4e1f511 done
Writing manifest to image destination
Storing signatures
D2e4e1f511320dfb2d0baff2468fcf0526998b73fe10c8890b4684bb7ef8290f
agaur@muller:login01:~> podman images
REPOSITORY                TAG           IMAGE ID      CREATED        SIZE     R/O
docker.io/library/ubuntu  latest        d2e4e1f51132  2 days ago     80.3 MB  false
```

Podman User Workflow - Squash the image

2. Squash and migrate it to an additional image store

```
agaur@muller:login01:~> echo $SQUASH_DIR
/mscratch/sd/a/agaur/images
agaur@muller:login01:~/container/bin> ./migrate2scratch.py mig
docker.io/library/ubuntu:latest $SQUASH_DIR
Generating squash file
Created squash image
agaur@muller:login01:~/container/bin> podman images
REPOSITORY                TAG      IMAGE ID      CREATED     SIZE    R/O
docker.io/library/ubuntu  latest   d2e4e1f51132  2 days ago  80.3 MB false
docker.io/library/ubuntu  latest   d2e4e1f51132  2 days ago  80.3 MB true
```

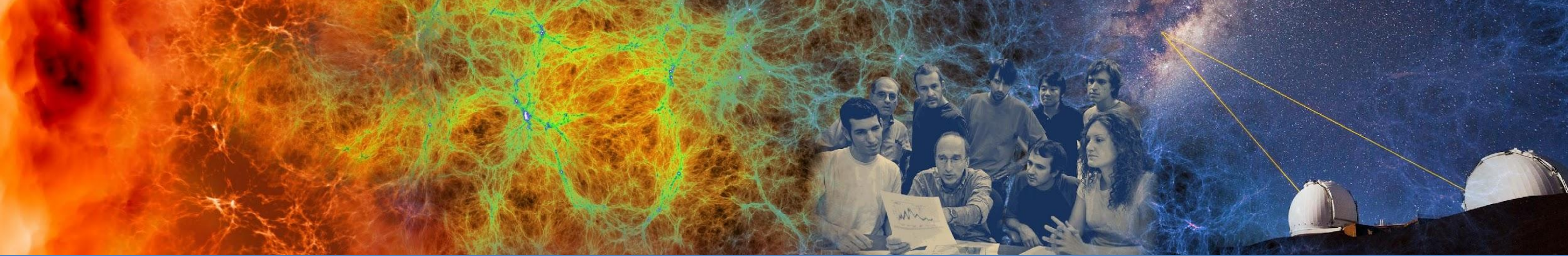
Read-only additional image store

Podman User Workflow - Access it via slurm

3. Submit a slurm job to access the image.

```
agaur@muller:login01:~/container/bin> salloc -N 1 -C gpu --userns -A
nstaff_g
salloc: Granted job allocation 45515
salloc: Waiting for resource configuration
salloc: Nodes nid001033 are ready for job
agaur@nid001033:~/container/bin> podman images
REPOSITORY                TAG      IMAGE ID     CREATED      SIZE    R/O
docker.io/library/ubuntu  latest   d2e4e1f51132 2 days ago 80.3 MB   true
```

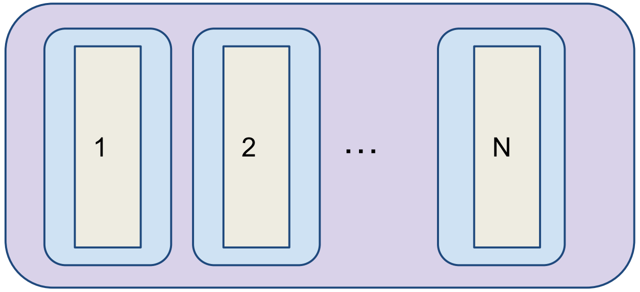
Only `additionalImageStore` is accessible on the compute node.



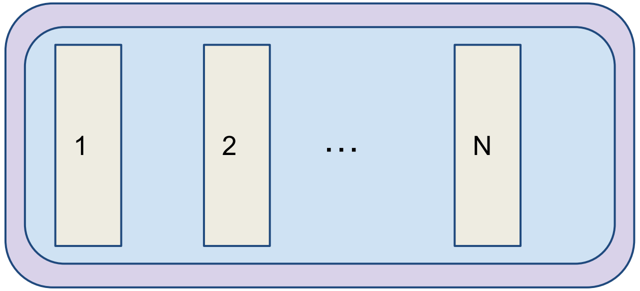
Early insights from Scaling

Execution Models

- We will be comparing four different modes: Bare Metal, Shifter, Podman, Podman "Shared"
- Podman modes:
  - Regular: A podman container per MPI rank/task
  - Shared: A podman container per node (shared by ranks on a node)
- All Podman results use the modified squash supported version



Podman Regular (per Node)



Podman Shared (per Node)

Example Batch Script - Non-shared

```
srun -N 2 -n 128 \
    podman run --rm \
       --network=host \
       --ipc=host \
       --pid=host \
       --privileged \
       -e PALS_APID \
       -e PALS_APINFO \
       -e PALS_NODEID \
       -e PALS_RANKID \
       -e PALS_SPOOL_DIR \
       -e PMI_CONTROL_PORT \
       -e SLURM_CPU_BIND \
       -e SLURM_CPU_BIND_LIST \
       -e SLURM_CPU_BIND_TYPE \
       -e SLURM_CPU_BIND_VERBOSE \
       -e SLURM_DISTRIBUTION \
       -e SLURM_NPROCS \
       -e SLURM_NTASKS \
       -e SLURM_STEP_RESV_PORTS \
       -e SLURM_UMASK \
```

```
-v /dev/xpmem:/dev/xpmem \
-v /dev/shm:/dev/shm \
-v /dev/infiniband:/dev/infiniband \
-v /var/spool/slurmd:/var/spool/slurmd \
-v /etc/libibverbs.d:/etc/libibverbs.d \
-v
/usr/lib/shifter/mpich-1.1/:/opt/udiImage/modules/
mpich \
-e
LD_LIBRARY_PATH=/opt/udiImage/modules/mpich:/opt/u
diImage/modules/mpich/dep \
       scanon/pynamic:2.6a1 \
       /pynamic/pynamic-pyMPI-2.6a1/pyMPI
/pynamic/pynamic-pyMPI-2.6a1/pynamic_driver.py
```

Needed for MPI to use the host high speed network.

Pull in MPI libraries

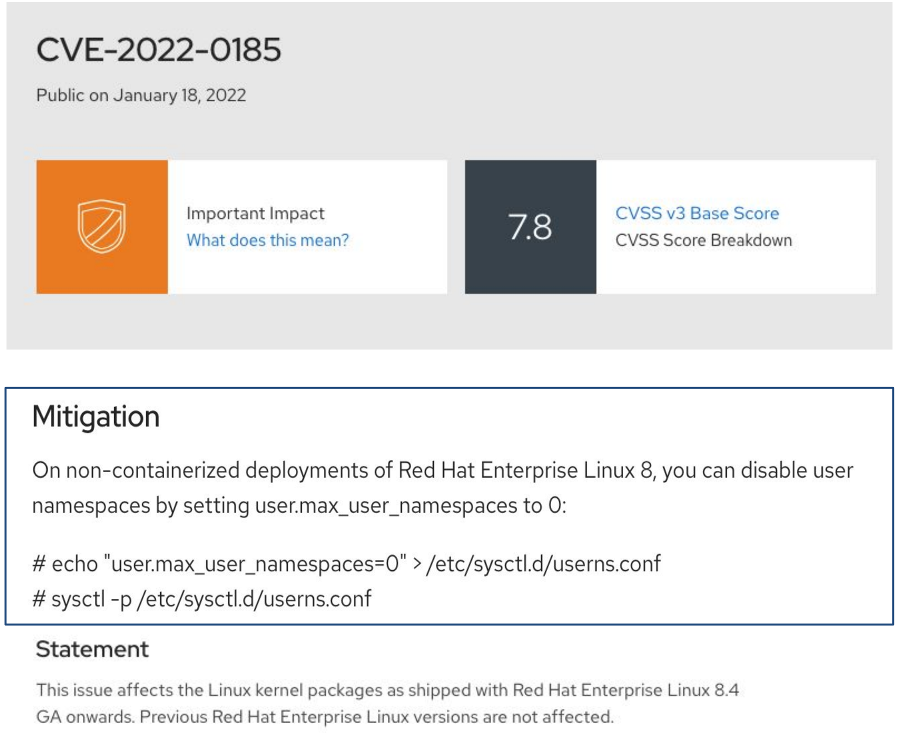
Scaling limitations

The scaling results presented here were limited by a few bottlenecks:

- Podman/slurm cgroups interaction
  - Slurm being unable to delete podman created cgroups, resulted in many nodes getting “kill task failed”
  - Cgroupv2 might help prevent some of the cgroup related issues.
  - Slurm in many cases is unable to kill podman processes if the user job times out or fails (due to any other reasons). High amount of node drains prevented big scaling runs.
  - Problem being addressed by SchedMD, and Red Hat.

Security considerations

- 3 CVE's in 2022 so far requiring mitigations to disable user namespaces
- Running user namespaces safely and securely is necessary for running rootless containers.
- Disabling user namespaces can be extremely disruptive.



CVE-2022-0185

Public on January 18, 2022

Important Impact
What does this mean?

7.8

CVSS v3 Base Score
CVSS Score Breakdown

Mitigation

On non-containerized deployments of Red Hat Enterprise Linux 8, you can disable user namespaces by setting user.max_user_namespaces to 0:

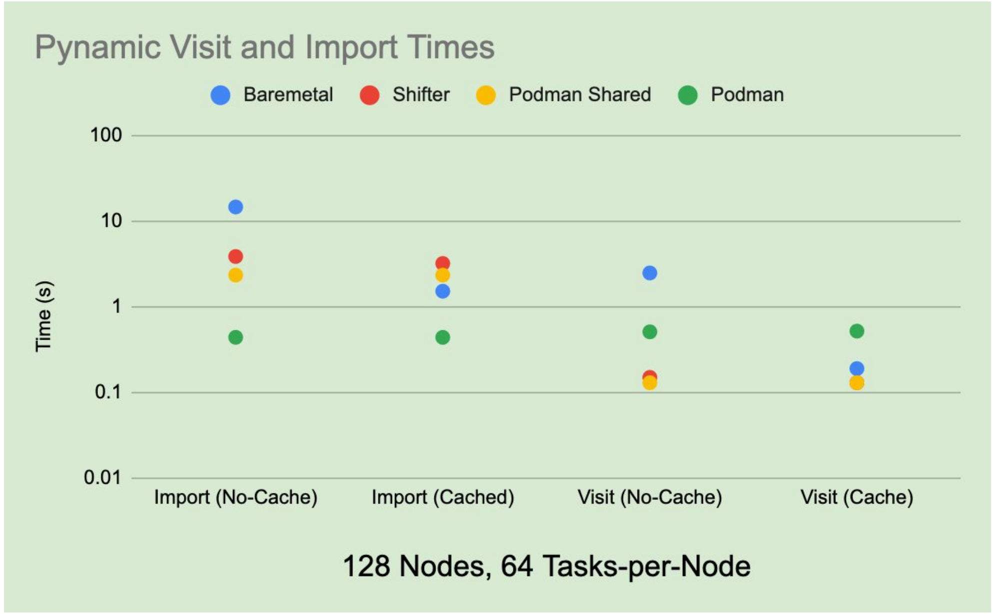
```
# echo "user.max_user_namespaces=0" > /etc/sysctl.d/userns.conf
# sysctl -p /etc/sysctl.d/userns.conf
```

Statement

This issue affects the Linux kernel packages as shipped with Red Hat Enterprise Linux 8.4 GA onwards. Previous Red Hat Enterprise Linux versions are not affected.

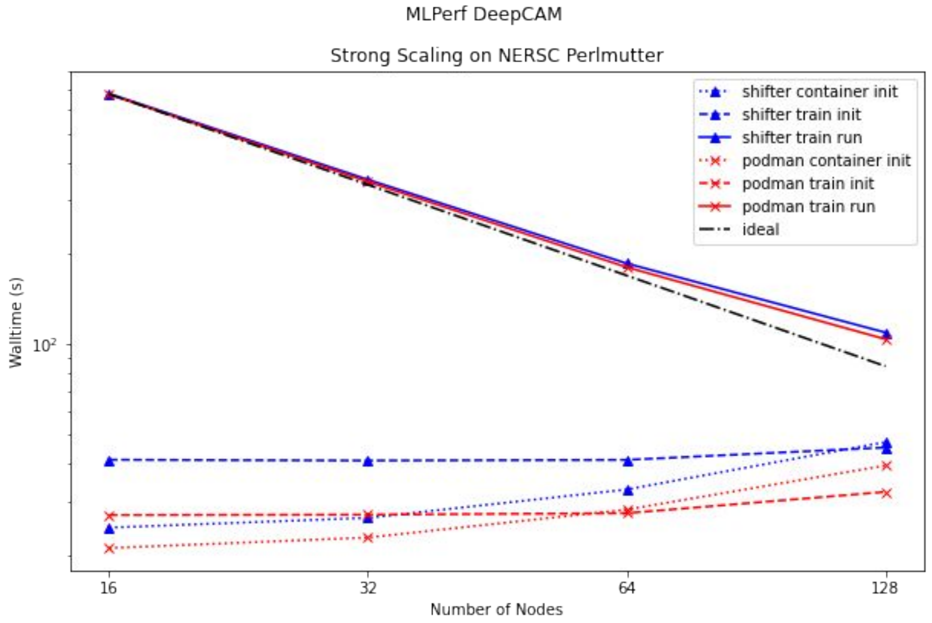
Pynamic - 128 Nodes

- Pynamic emulates complex Python application loading
- Measures Import and Visit Times
- Measured at 128 Nodes, 64 Tasks per Node on Perlmutter
- Podman + Squash is performing well even with the squashfuse overhead



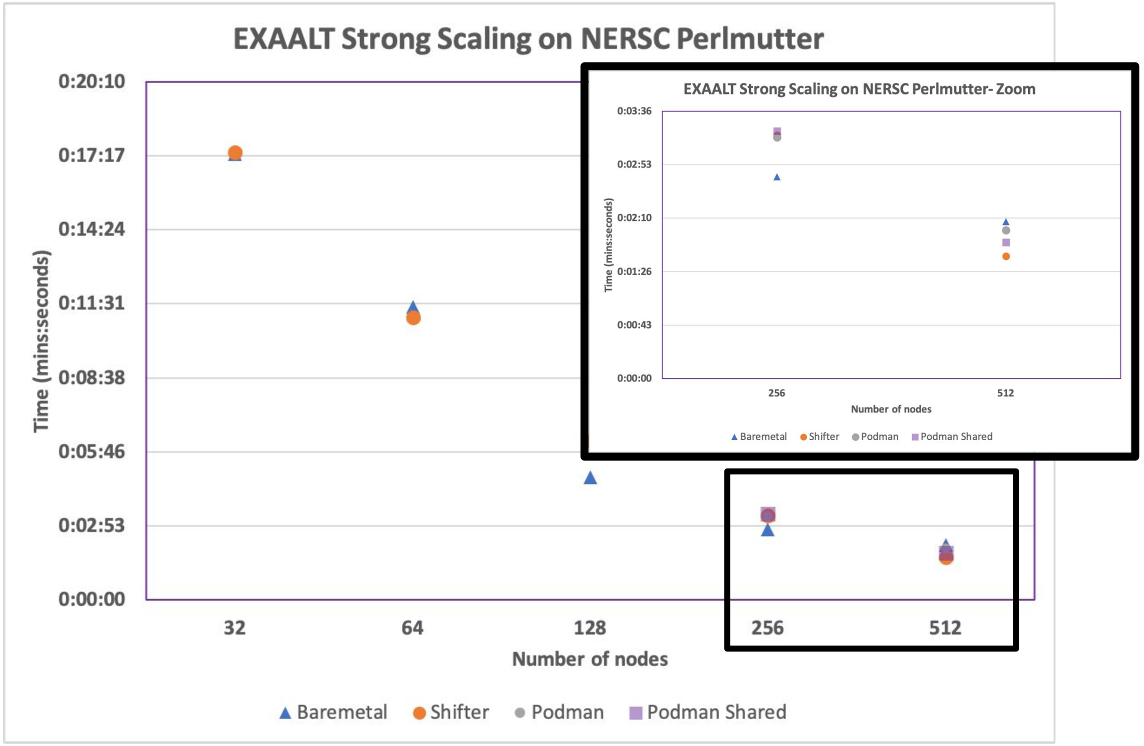
ML GPU Scaling

- MLPerf DeepCAM training with reduced size dataset.
- Using `podman-exec` execution model.
- Comparable runtime performance between `podman` and `shifter`.
- No obvious scaling concerns on GPUs.



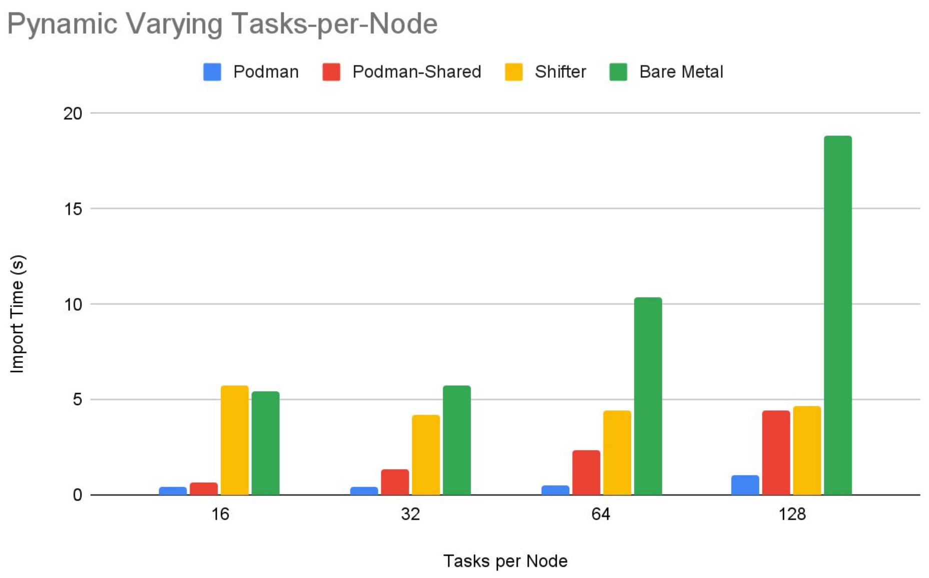
Scaling with EXAALT (MPI + GPU)

- ECP app EXAALT uses LAMMPS
- GPU app with CUDA-aware MPI
- Podman runs up to 512 nodes with performance close to Shifter and baremetal
- Podman data collected only at large scale due to node crashes

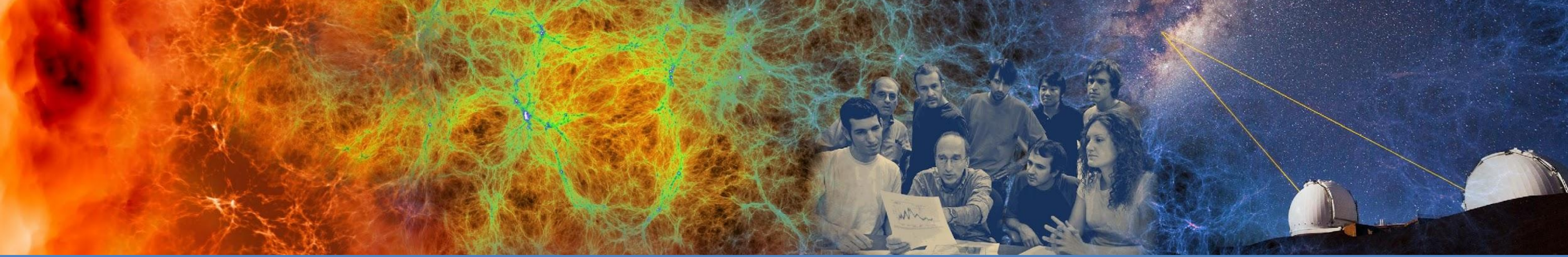


Squashfs Performance and Scaling

- Squash-fuse driver and kernel driver can be a bottleneck
- Some advantages to running a separate container per task (e.g. non-shared)
- There could be other impacts of running this way that offset this advantage.
- Podman and Shifter out-perform bare-metal for imports.



Pynamic Import Times Non-cached
Single node varying Tasks-per-Node



What we have learned

User Experience

- Sites ought to have a plan for subuids/subgid’s early on.
    - Once subuid’s and subgid’s are set for a given user, changing them may make the files created with those subuid’s unusable.
    - Additionally shared filesystems (like Lustre) still require a single user ID owning files. Does not understand subuid/subgid’s.
    - At NERSC, we are hoping to provide a constant subuid/subgid mapping via LDAP. And for shared filesystems - `` `ignore_chown_errors` `` flag helps us avoid chown errors.
- Desirable to have `podman pull` step also convert the images to squashed format and migrate it to additional stores.

Future Work

- Production
  - After security fixes: expand access to NERSC early users
  - Develop documentation, trainings, and improve sharp edges
- Improve and tweak OCI hooks
  - Develop hook for MPI/CUDA-aware MPI
  - Develop hooks for filesystem mounts like CVMFS
- Explore Slurm integration
  - Cleanup, cgroups
- Work with community to merge squashfs support into main branches

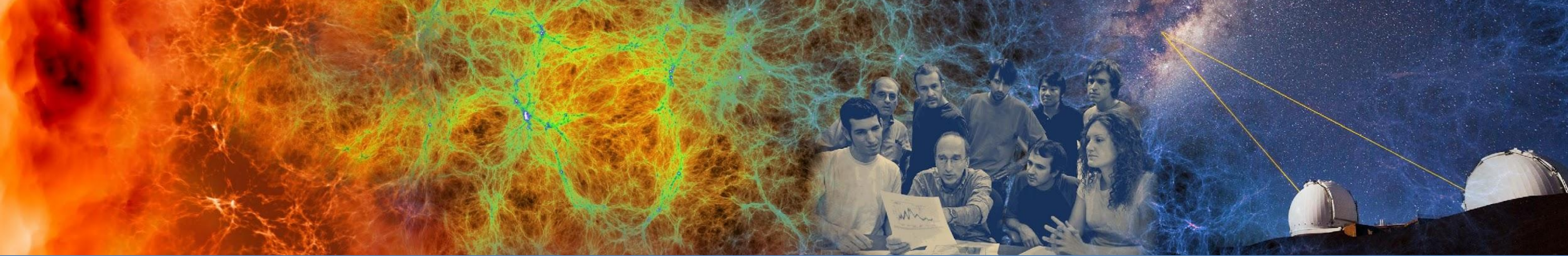
Cut for Time

There is much more to discuss, some things we've skipped:

- OCI hooks to use GPU's and MPI inside containers
- Leveraging Podman to support unprivileged kubernetes to enable complex workflows. (e.g. usernetes)
- Plans around using Podman to support local builds on login nodes. (or UAN's in Cray Shasta systems)
- Podman + NFS/GPFS/Lustre issues

Conclusions

- Podman appears to be a feasible container solution with some enhancements.
- Early benchmarks demonstrate that it can perform well (but further testing is needed).
- Giving users ability to build and run with Podman will greatly improve containers workflow and usability at NERSC
- Using a standard container framework means we can engage with the larger OCI community

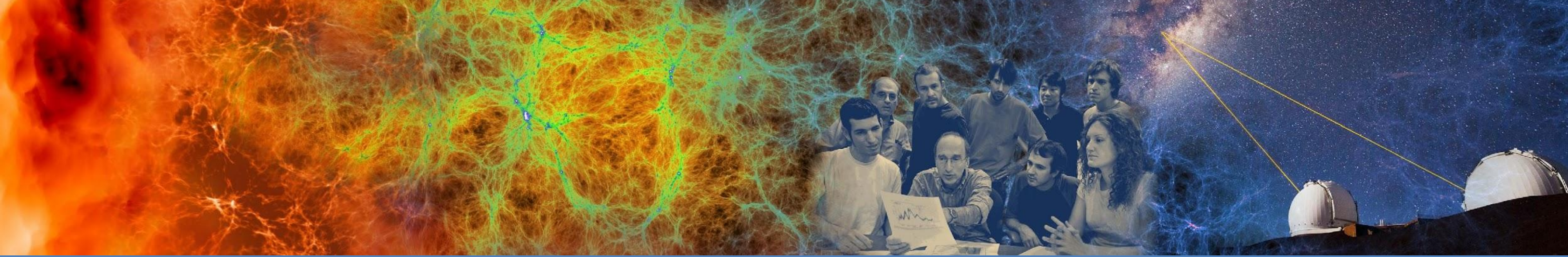


Acknowledgements

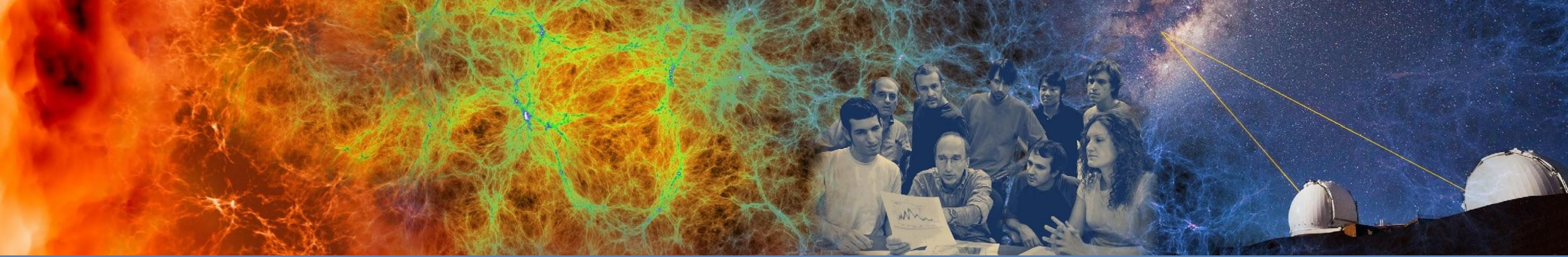
Thanks

We would like to thank

- Andrew Younge for leading bi-weekly engagement with Red Hat
- Red Hat Developers for helping us with many questions we had during development
- Rahul Gayatri, for his help in adapting the EXAALT benchmark
- Steve Farrell, for his help in adapting the MLPerf benchmark
- Gabor Torok for implementing subuid/subgid support in our LDAP database
- Rebecca Hartman Baker for giving us system time to do scale testing
- Chris Samuel for helping us debug and understand slurm cgroup interactions
- DOE Exascale Computing Project



Thank You



Appendix Slides

Containers at NERSC

- NERSC was an earlier (perhaps one of the earliest) adopters of containers for HPC
- Attractive approach to allow users to create tailored environments and provide greater flexibility
- Well-suited for complex software stacks like NVIDIA-provided PyTorch
- Imperative that users could still obtain native performance applications could scale well.
- Initially explored wrappers around Docker then developed Shifter in 2015.
- Adoption and usage modes have expanded (Complex Workflows, Jupyter kernels, Spin services)

Brief overview of Shifter

- Reused the most popular container format and tool chains (Docker)
- Replaced the run-time with something that was
  - Secure (Didn't give users any extra privs)
  - Scalable (used a novel pre-squashed format)
  - Integrated (Integrated with the batch system and MPI)
  - Incorporated many familiar Docker capabilities and syntax (like volume mounting)
- Current limitations:
  - Users must build containers elsewhere
  - Due to squashed image, container is read-only
  - Some Shifter functionality is NERSC-specific (for example, our Cray MPICH modules)