SAND2021-12885C

Sandia National Laboratories

Exceptional service in the national interest

PORTABILITY AND REPRODUCIBILITY CONSIDERATIONS WITH CONTAINERS

SIXTH ANNUAL CROSS RESEARCH SYMPOSIUM

OCTOBER 14TH, 2021

PRESENTED BY

ANDREW J. YOUNGE

Acknowledgements:
Shane Canon, Kevin Pedretti, Jay Lofstead, Si Hammond

ECP SUPERCONTAINERS

Joint DOE effort - Sandia, LANL, LBNL, LLNL, U. of Oregon

Ensure container runtimes will be scalable, interoperable, and well integrated across DOE
- Enable container deployments from laptops to Exascale
- Assist Exascale applications and facilities leverage containers most efficiently

Three-fold approach
- Scalable R&D activities
- Collaboration with related ST and AD projects
- Training, Education, and Support

Activities conducted in the context of interoperability
- Portable solutions
  - Optimized E4S container images for each machine type
  - Containerized ECP that runs on Astra, A21, El-Capitan, …
- Work for multiple container implementations
  - Not picking a “winning” container runtime
- Multiple DOE facilities at multiple scales

SUPERCONTAINERS

MOTIVATION FOR CONTAINERS

- Containerized computing is being adopted across HPC landscape
- Many potential benefits
  - Prescriptive deployment
  - Modern DevOps
  - Portability of containers
  - Potential to reproduce workloads later *
  - Flexible software ecosystem

SUPERCONTAINERS

- Several potential tools and container runtimes available today
  - Scale from your workstation to a supercomputer
- Eases barrier of entry for complex or emerging software ecosystems

REPRODUCIBILITY IN HPC

- Reproducibility is a cornerstone of quality science!
  - Consistent results across studies aimed at answering the same **scientific** question
  - Critically important in conducting computational science today
- DOE/NNSA must extend the lifecycle without underground testing
- Rely on modeling and simulation apps to perform this task
  - incorporate a multitude of physics and engineering models
  - Executed on leadership-class supercomputers
- Long-term studies take years
  - Any particular simulation may not seem important at the time
  - Later analysis may prove to demonstrate value in an old simulation
  - Need to reproduce & reevaluate runs many months or years later!
- Can containers help or hinder?

PORTABILITY & REPRODUCIBILITY PROBLEMS

- Containers promise the potential to improve flexibility for developers
  - Support of user-defined software stacks
  - Potential impact in portability and reproducibility
- Current implementations fall short of delivering on promises
  - System must still match the host micro-arch
  - System must be capable of exploiting specialized hardware in HPC
    - High speed, low latency interconnects
    - Specialized instructions & extensions
    - Advanced GPUs and accelerators
  - Require runtimes to leverage host libraries in containers for performance

PODMAN ON THE ASTRA SUPERCOMPUTER

Astra ideal test environment for Podman container build experiments

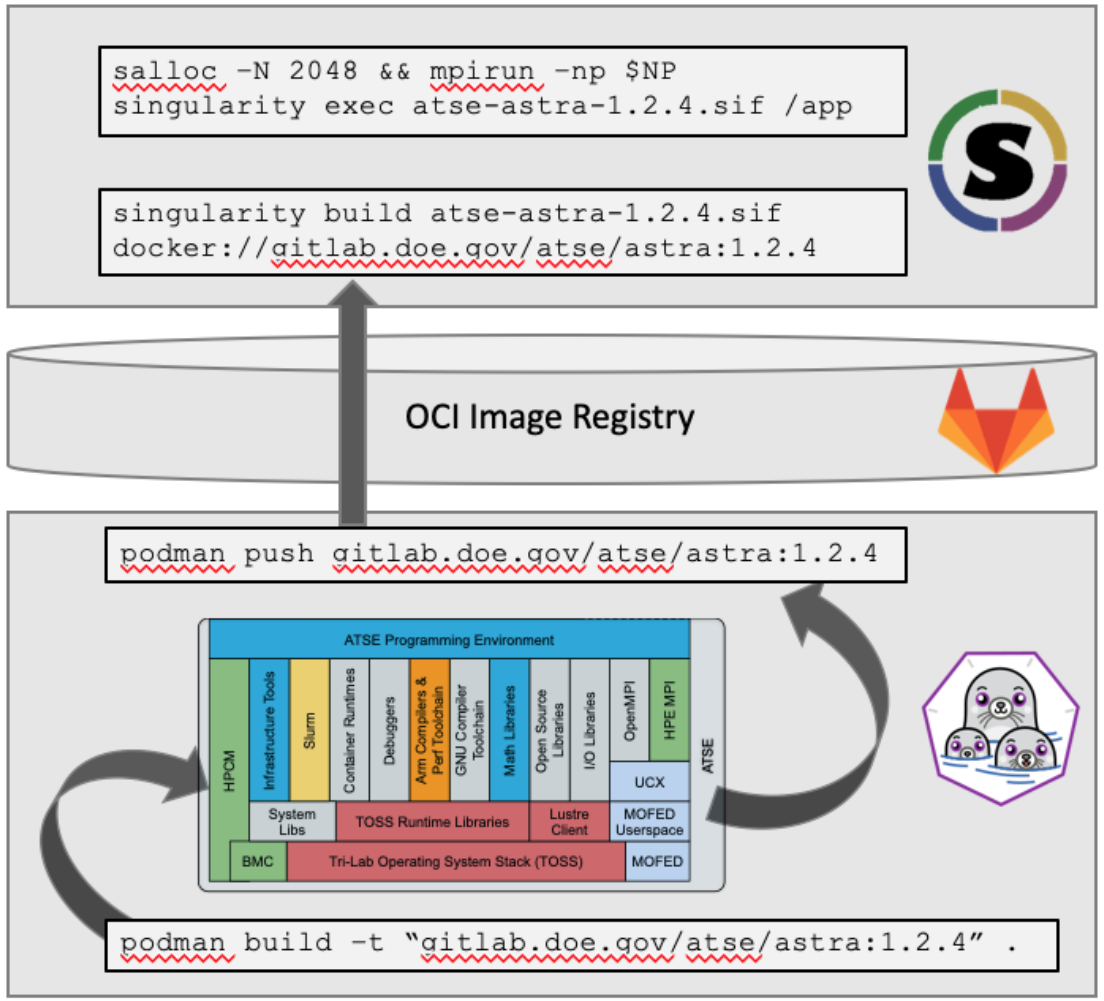
- 1st Arm-based Supercomputer on Top500 (Nov17)
  - 5000+ Marvel ThunderX2 processors (aarch64 armv8)
  - Significant need for container builds on Arm architecture
- Astra a prototype system => increased flexibility for R&D

Building Advanced Tri-lab Software Environment (ATSE) and HPC apps => in a container

- Built directly on Astra login nodes with Podman
- Now using Spack
- Pushed OCI images to site Gitlab container registry
- Singularity to run at scale => can use any HPC container runtime (with some relativity)

Demonstrated first on-platform container build, but prototype has limitations

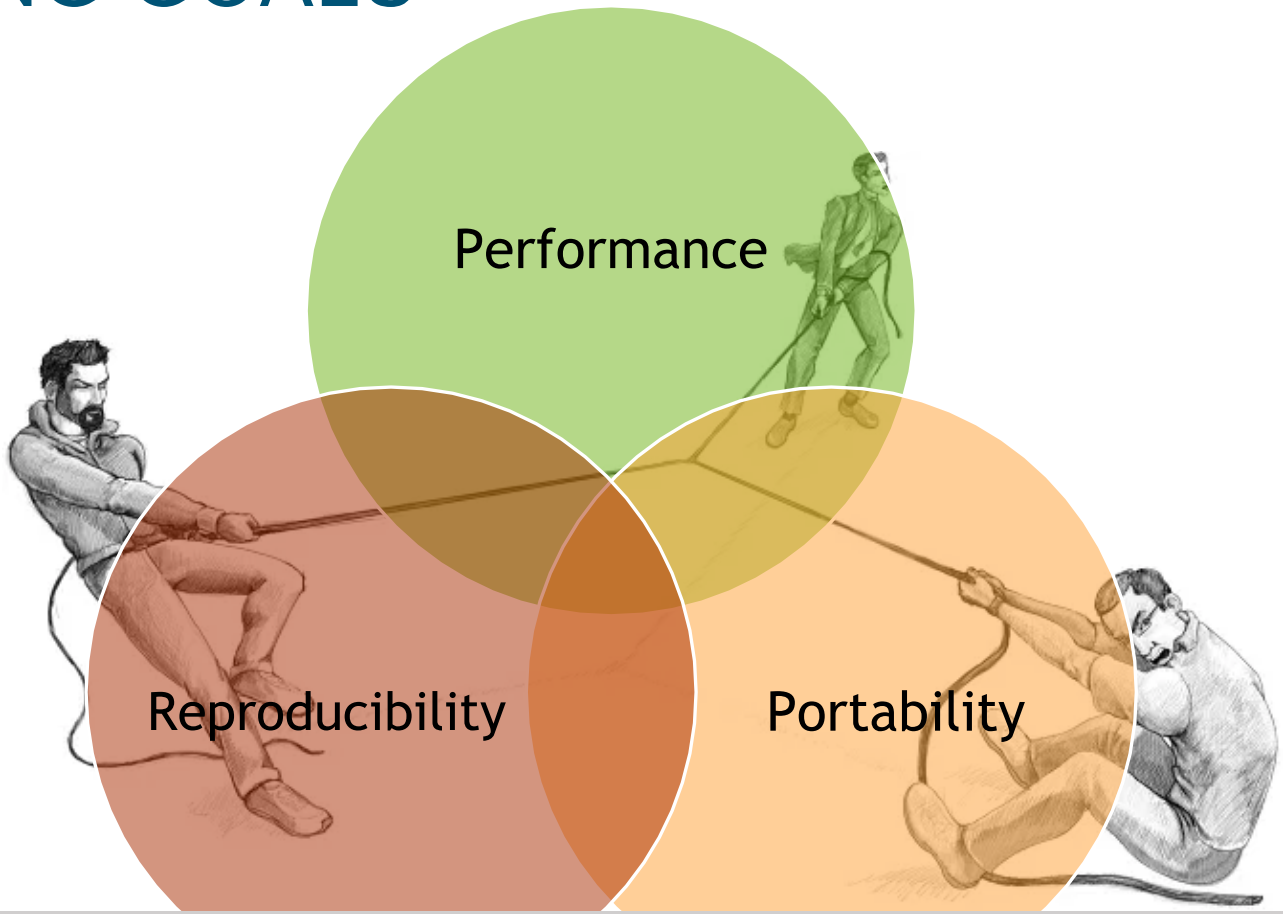
- RHEL7 missing several features
  - Overlay, FUSE, NFS client features
- Collaborating with Red Hat to productionize on RHEL8



CONSIDERATIONS

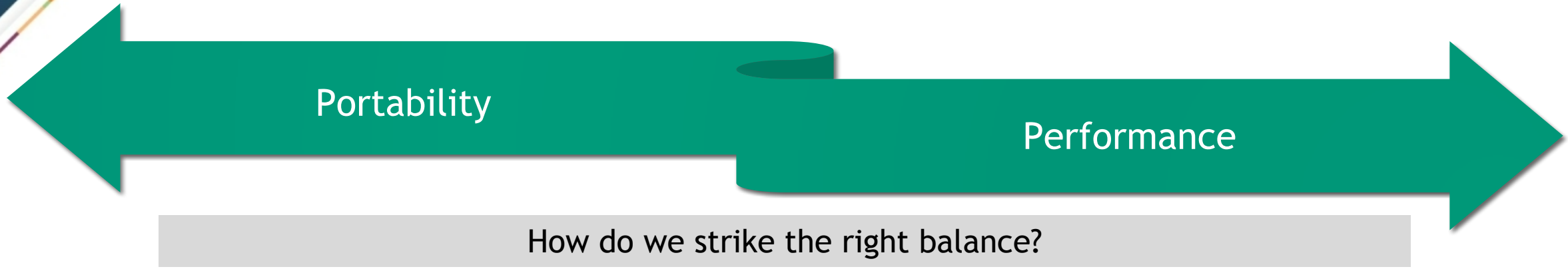
- HPC applications need to use specialized interconnects & libraries not found or optimized for in base OS packages.
- Typical container solution requires mapping in libraries which can cause host-to-container incompatibilities.
- There are differing methodologies in container runtimes for incorporating GPUs and accelerators.
- Users may not know if a given container image can be ported to a different HPC system.
- If portability is possible, users may not know what performance implications exist when running a container on a different HPC system.

COMPETING GOALS



Achieving “Ideal” Reproducibility may impact performance and portability and vice versa

CONTAINER PERFORMANCE PORTABILITY CONTINUUM



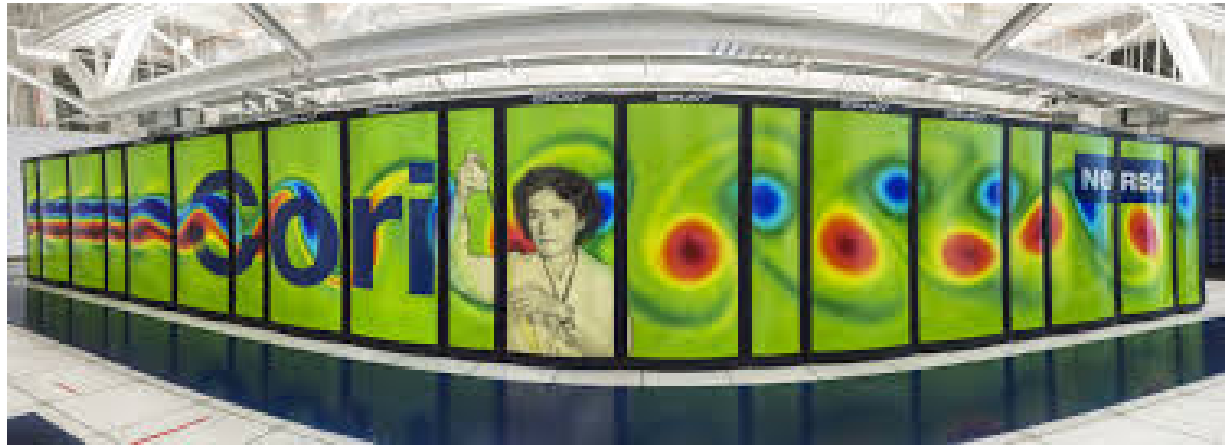
How do we strike the right balance?

- Portable container images can be moved form one resource deployment to another with ease
- Reproducibility is possible
  - Everything (minus kernel) is self-contained
  - Traceability is possible via build manuscripts
  - No image modifications
- Performance can suffer – no optimizations
  - Can't build for AVX512 and run on Haswell
  - Unable to leverage latest GPU drivers

- Performant container images can run at near-native performance compared to natively build applications
- Requires targeted builds for custom hardware
  - Specialized interconnect optimizations
  - Vendor-proprietary software
- Host libraries are mounted into containers
  - Load system MPI library
  - Match accelerator libs to host driver
- Not portable across multiple systems

STATE OF THE PRACTICE IN CONTAINERIZED HPC

- System-specific libraries are needed within a given HPC container image
- Combine bind mounts and dynamic linking to inject optimized libraries
  - From the host into the container runtime environment
  - Assert libraries are optimally configured to drive HPC hardware
  - ”Container Bypass” mechanism
- Example: MPICH-based implementations
  - Rely on ABI compatibility
    - Build from generic MPICH on container
    - Swap in CrayMPI at runtime
  - Force MPI apps to use optimal MPI
  - 2 methods
    - Replace libmpi.so directly
    - Overlay/bind mount and change LD_LIBRARY_PATH
  - Demonstrated with Shifter at full system scale on Cori



OPENMPI USAGE IN CONTAINERS

- Container developer has 2 options
  - custom-build OpenHPC to fit HPC spec
    - IB versions, MOFED userspace drivers, PMIX, etc
    - Requires detailed knowledge of target system
    - Not portable
  - Generic OpenMPI build and use container bypass for OpenMPI
    - OpenMPI ABI incompatibilities across sub-versions!
    - OpenMPI uses RPATH -> many library dependencies have to be handled
      - Custom bind-mounts can be unweildy
    - Userspace container drivers must match host drivers (MOFED)
- Current container usage model on Astra
  - Works & scales >2000 nodes
  - Cumbersome for custom containers

THE CONTAINER BYPASS PROBLEM

- The Open Container Initiative (OCI) spec can help standardize some aspects
  - But not well fit for HPC as-is.
- HPC container runtimes cannot easily determine what libraries are needed
- Problem: host libraries can have conflicting dependencies with what's in the container

```
$ srun -n 1 shifter /app/hello
...
/app/hello: /lib/x86_64-linux-gnu/
libm.so.6: version 'GLIBC_2.23' not found
(required by /opt/udiImage/modules/mpich/
lib64/dep/libquadmath.so.0)
...
```

  - Glibc mismatch error found on upgraded Cray system
    - Host: CLE7, SLES15, glibc 2.23 vs Container: Centos6, glibc 2.17
  - Host library mounted in the container not forward compatible with container's glibc
- Problem is not theoretical, it's real. And likely to occur again with interconnects, system updates, driver changes, …

PROPOSED SOLUTIONS

What do we do to avoid mismatches between host and container libraries?

Some potential options:

1. Custom Image Labels
2. Backwards Compatible Libraries
3. A container compatibility layer
4. System-level Virtualization

1. CUSTOM IMAGE LABELS

SHIFTER

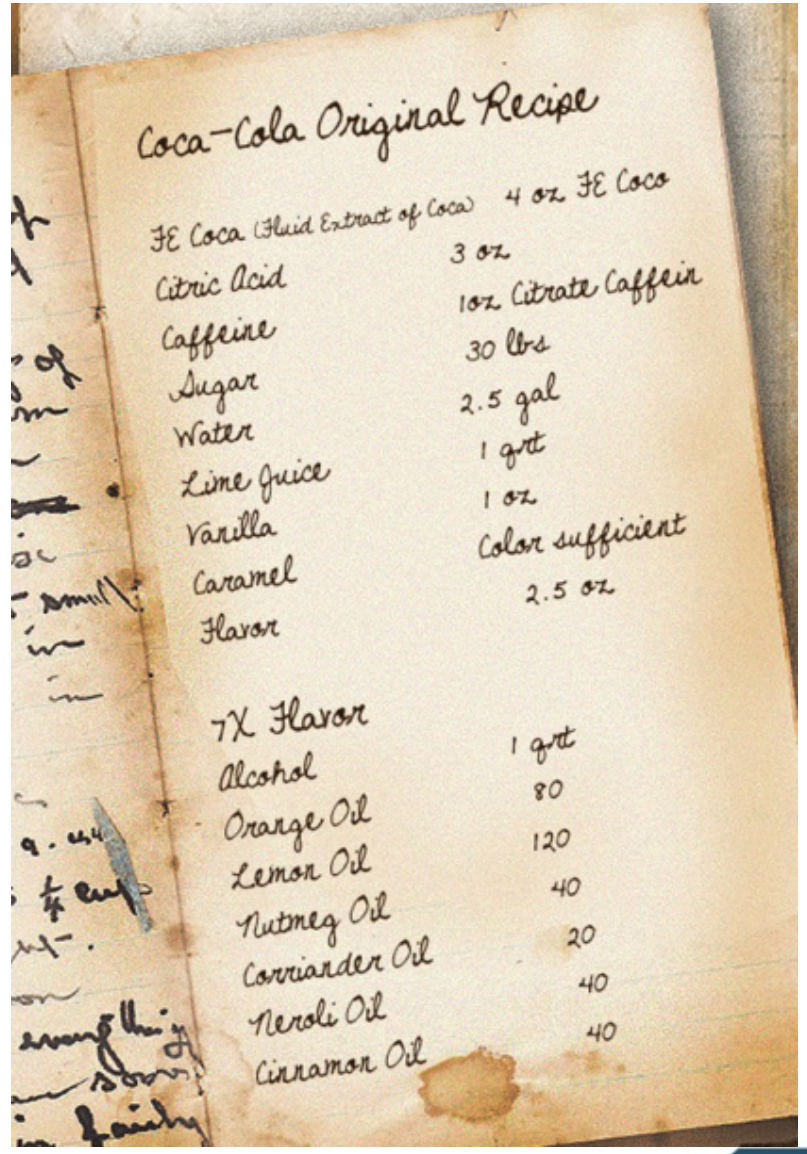
- Leverage OCI-compatible image LABELs
  - Insert directly in Dockerfile
  - Could reproduce in Singularity defs
  - Essentially embedding metadata into spec
- Labels specify expectations from the host
  - HPC container runtime intercepts labels, makes appropriate library insertion
  - Specify MPI version, Glibc expectation, etc
- Implemented prototype solution in Shifter
- Potential pitfalls:
  - Still requires container bypass
  - Requires extra work from developer

```
FROM centos:7

LABEL org.supercontainers.mpi=mpich
LABEL org.supercontainers.glibc=2.17

RUN yum -y update && \
    yum -y install gcc make gcc-gfortran \
        gcc-c++ wget curl

RUN B=mpich.org/static/downloads && V=3.2 && \
    wget $B/$V/mpich-$V.tar.gz && \
    tar xf mpich-$V.tar.gz && \
    cd mpich-$V && \
    ./configure && \
    make && \
    make install

ADD helloworld.c /src/helloworld.c

RUN mpicc -o /bin/hello /src/helloworld.c
```

Label	Values	Comment
org.supercontainers.mpi	{mpich,openmpi}	Required MPI support, ABI compatibility
org.supercontainers.gpu	{cuda,opencl,rocm, etc}	Required GPU library support
org.supercontainers.glibc	Semantic version: XX.YY.Z	Specific version of GLIBC

2. BACKWARDS COMPATIBLE LIBRARIES & SOURCE CODE

- Why can't we just build everything in a container in the first place?
  - Target hardware may not be known in advance
  - May not be possible to create base container image with key software
    - Vendor proprietary, closed source, export control restrictions, etc…
- Require vendors to provide backwards compatible libraries
  - Build MPIX multiple times, with multiple versions
  - Additional build complexity for vendors
- Require vendors to provide direct source code
  - Intractable – ask Cray for CrayMPI source ;)



3. A CONTAINER COMPATIBILITY LAYER

- Construct a lower-level library for HPC container runtimes
  - Could leverage custom labels like #1
  - Make directed decisions on how to bind-mount and `LD_PRELOAD`, resolve glibc issues
- Create a common place for vendors to integrate solutions
  - Instead of building many different solutions for each device/interconnect/accelerator, HW providers implement generic solution.
- Similar approaches exist
  - `libnvidia-container` takes similar approach, but vendor-specific.
  - Container Network Interface (CNI)
    - But not general enough for HPC
- Acceptable solution requires community-wide collaboration
  - From hardware vendors, container runtime developers, system integrators, etc etc

4. SYSTEM-LEVEL VIRTUALIZATION

- If reproducibility is paramount, we should leverage ISA virtualization
  - Virtual machines & hypervisors
  - Similar to cloud implementations, multiple levels of abstraction
  - Containers atop virtual machines atop HPC hardware
- Linux kernel & host environment to match container reqs with HPC hardware
- Control kernel config, drivers, to run container images only
  - Disable arbitrary VM models, no user control
  - Users only control container images, run in userspace
  - No root access
- Concerns
  - Significant infrastructure investment?
  - Acceptable performance in HPC with hypervisors?

OPEN DISCUSSION

Containers are able to enable new mechanisms in portability and reproducibility
- Reproducibility is of critical importance for conducting quality science
- Performance is paramount in HPC

Containerization in HPC has to fix container bypass issues to deliver

We've outlined and started investigating several potential solutions
- Will more metadata _really_ fix things?

What do **you** think is the right path forward for the HPC community?

SUPERCONTAINERS

We are hiring! ajyoung@sandia.gov