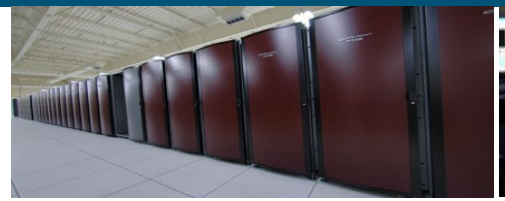
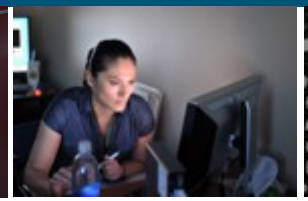
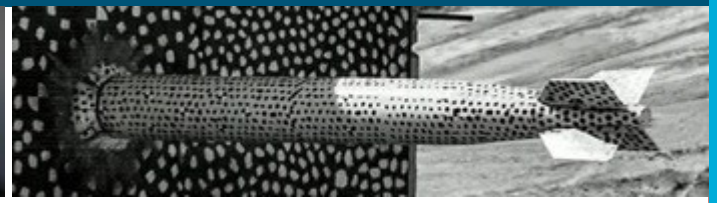
Leveraging Containerization for DevOps with Sandia's HPC Workloads

PRESENTED BY

Andrew J. Younge, PhD

ajyoung@sandia.gov

Unclassified Unlimited Release
DUSA: DIS-CS
SAND2018-7367 PE

Motivation

Multiple drivers exist for using containers in HPC

1. Containers to help with DevOps
   - Development of HPC apps on workstations which can port & scale to supercomputers
   - Utilize testbed/cloud resources for functionality and feature testing
   - Leverage latest Cloud craze in Docker containers
2. Containers to aid in emerging HPC workloads
   - Support for deep learning / machine learning software ecosystems
   - Large-scale data analytics & in-situ workload ensembles
   - Streaming analytics & non-batch jobs

Container features wanted in HPC

- **BYOE** - Bring-Your-Own-Environment.
  - Developers define the operating environment and system libraries in which their application runs.
- **Composability**
  - Developers explicitly define how their software environment is composed of modular components as container images
  - Enable reproducible environments that can potentially span different architectures
- **Portability**
  - Containers can be rebuilt, layered, or shared across multiple different computing systems
  - Potentially from laptops to clouds to advanced supercomputing re- sources
- **Version Control Integration**
  - Containers integrate with revision control systems like Git
  - Include not only build manifests but also with complete container images using container registries like Docker Hub.

Container features not wanted in HPC

- **Overhead**
  - HPC applications cannot incur significant overhead from containers
- **Micro-Services**
  - Micro-services container methodology does not apply to HPC workloads
  - 1 application per node with multiple processes or threads per container
- **On-node Partitioning**
  - On-node partitioning with cgroups is not necessary (yet?)
- **Root Operation**
  - Containers allow root-level access control to users
  - In supercomputers this is unnecessary and a significant security risk for facilities
- **Commodity Networking**
  - Containers and their network control mechanisms are built around commodity networking (TCP/IP)
  - Supercomputers utilize custom interconnects w/ OS kernel bypass operations

Container Vision @ Sandia

- Support software dev and testing on laptops
  - Working builds then can run on supercomputers
  - May also leverage VM/binary translation
- Let developers specify how to build the environment AND the application
  - Users just import a container and run on target platform
  - Many containers, but can have different code “branches” for arch, compilers, etc.
  - Not bound to vendor and sysadmin software release cycles
- Performance matters!
- Want to manage permutations of architectures and compilers
  - x86 & KNL, ARMv8, POWER9, etc
  - Intel, GCC, LLVM

Container Vision @ Sandia

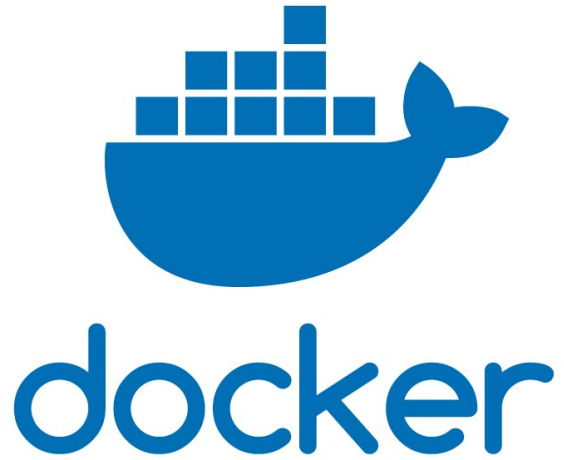
- Developers specify exactly their runtime environment
  - OS, version
  - Third-party libraries (TPLs)
  - How to compile
- Can share environment as a container with other developers
  - Quickly get env to new developer
  - Provide software as a container to analysts
  - Developer makes changes, triggerd container build with CI, validated image
- Leverage same container image on different clusters or supercomputers

Trilinos Muelu Container Example

```
FROM ajyounge/dev-tpl

WORKDIR /opt/trilinos

# Copy files to image

COPY do-configure /opt/trilinos/

# Download Trilinos source tarball

RUN wget -nv https://trilinos.org/oldsite/download/files/trilinos-12.8.1-Source.tar.gz -O /opt/trilinos/trilinos.tar.gz

# Extract Trilinos source file

RUN tar xf /opt/trilinos/trilinos.tar.gz -C /opt/trilinos/

RUN rm -f /opt/trilinos/trilinos.tar.gz

RUN mv /opt/trilinos/trilinos-12.8.1-Source /opt/trilinos/trilinos

RUN mkdir /opt/trilinos/trilinos-build

# Compile Trilinos

RUN /opt/trilinos/do-configure

RUN cd /opt/trilinos/trilinos-build && make -j 3

#Link in a directory, and then set the workdir

RUN ln -s /opt/trilinos/trilinos-build/packages/muelu/doc/Tutorial/src /opt/muelu-tutorial

WORKDIR /opt/muelu-tutorial
```

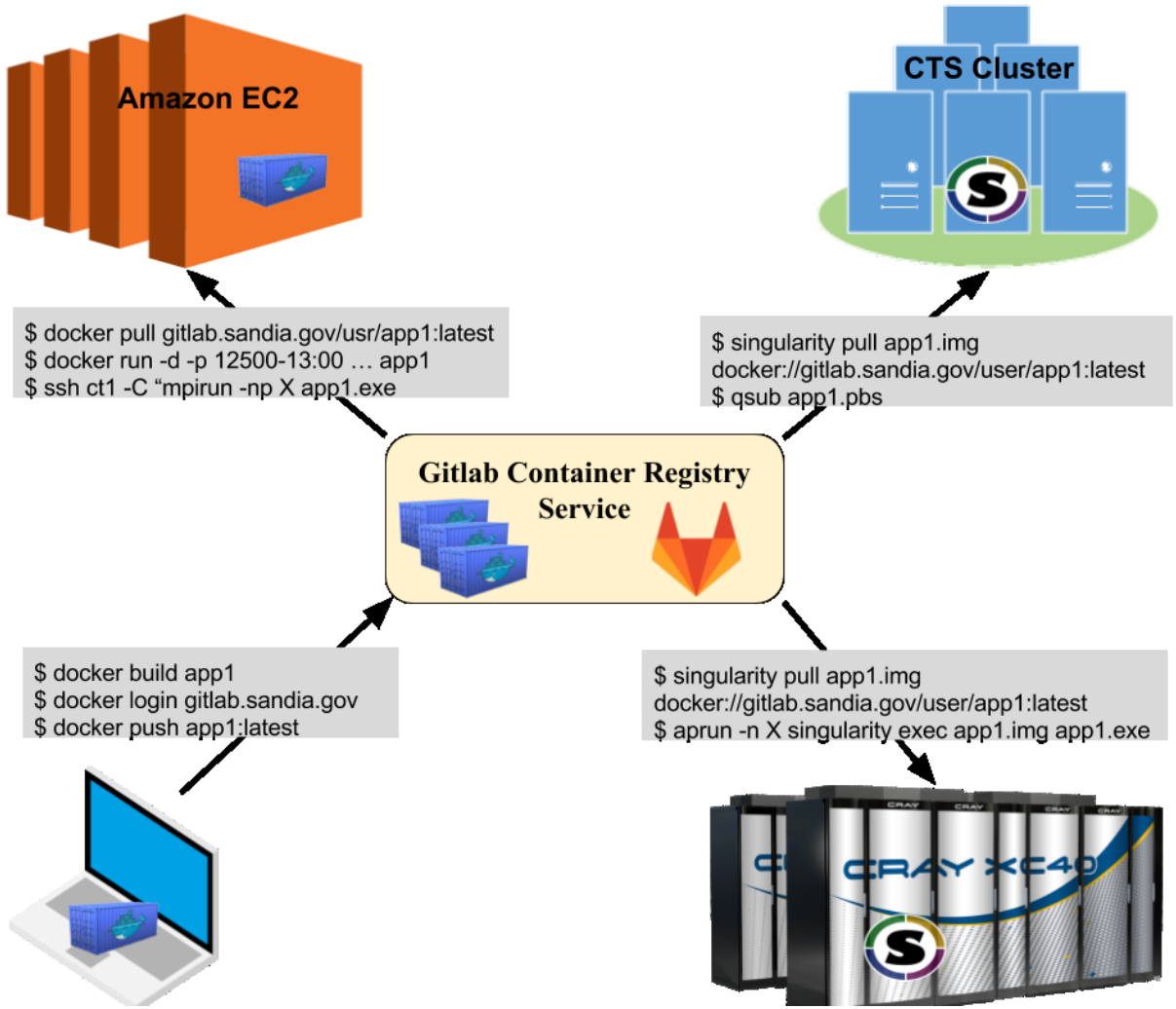
Container Runtimes

- Many different container options
  - Docker, Shifter, Singularity, Charliecloud, etc etc
- Docker containers useful for workstations
  - Allows root level builds and control on personal machine
  - NOT for HPC - Security issues, no shared resource integration
- Singularity best fit for our current HPC needs
  - OSS, publicly available, support backed by Sylabs
  - Simple image plan, support for HPC systems
  - Docker image support, as well as custom Singularity builds
  - Support for multiple architectures (x86, ARM, POWER)
  - Large HPC community support



Container DevOps @ Sandia

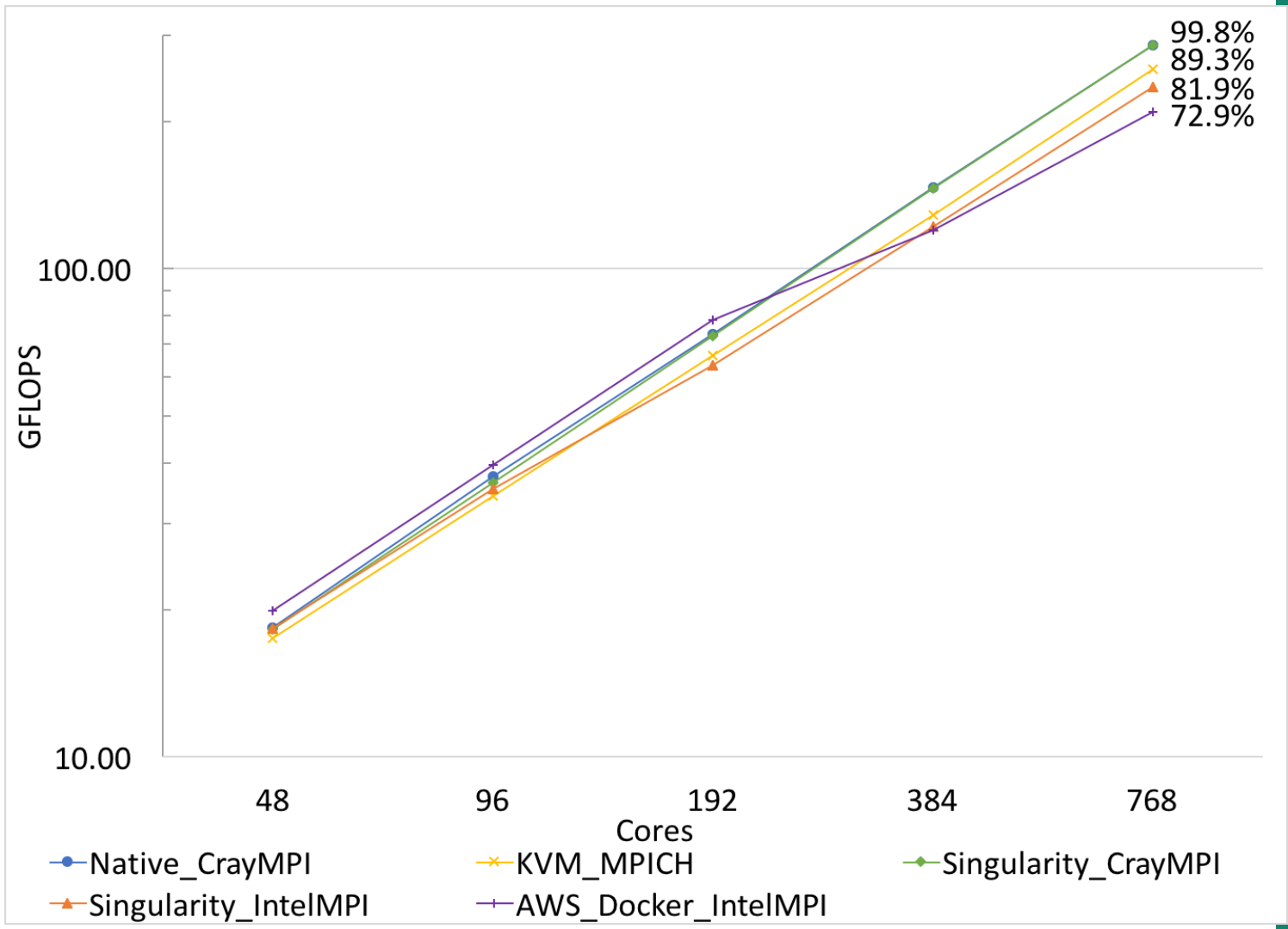
- Impractical for apps to use large-scale supercomputers for DevOps and/or testing
  - HPC resources have long batch queues
  - Dev time commonly delayed as a result
- Create deployment portability with containers
  - Develop Docker containers on your laptop or workstation
  - Leverage Gitlab registry services
    - Separate networks maintain separate registries
  - Import to target deployment
    - Leverage local resource manager
- Deployment to Cray supercomputer now possible



Younge et. al, *A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds*, IEEE CloudCom 2017

HPCG Container Performance

- Modified Cray XC supercomputer to run Singularity containers
- Create `/opt/cray` and `/var/opt/cray` on all images
- Link in Cray system software
  - XPMEM, CrayMPI, uGNI, etc
- HPCG Benchmark in Container
  - Compare Singularity on Cray
  - Compare KVM on Cray
  - Compare Amazon EC2
- Near-native (99.8%) efficiency when using Singularity on a Cray
  - Poor scalability on EC2



Younge et. al, *A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds*, IEEE CloudCom 2017

Discussion

- Containers in HPC are different than containers in the cloud
  - Running Docker alone is unacceptable on HPC
  - Need for HPC-centric containerization – Singularity
- Developing DevOps models for custom software ecosystems
- Performance _*can*_ be near native
  - Leveraging vendor libraries within a container is critical
  - Cray MPI on Aries most performant
  - Best-practices are necessary
- Leveraging container model into current & future integrated code teams deployment and testing strategy

Future Directions

Many opportunities & challenges moving forward:

- Container and library interoperability is key
  - Vendor-blessed base images
  - Facilities-blessed user-defined images – container signing?
  - Standardization on image format and ABI compatibility is necessary
- Better system software architecture is needed
  - Containers are a piece of a larger puzzle
  - Better integration with HPC scheduling systems
  - Experiment provenance possible?
- Support emerging HPC software ecosystems
  - Large-scale data analytics
  - Deep Neural Networks on supercomputers
  - Non-batch streaming workloads
  - etc etc

Questions?

ajyoung@sandia.gov

"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov