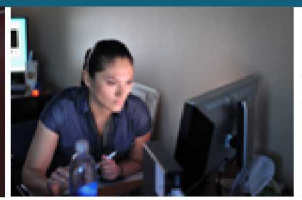
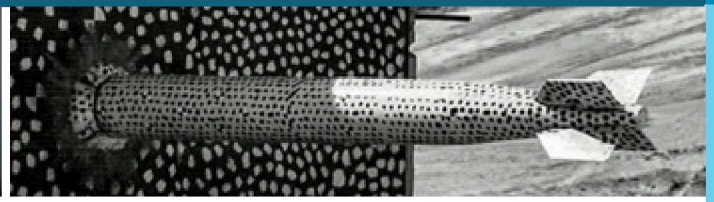
SAND2019-0594C

Deployment and Usage of Containers for Production HPC Applications





PRESENTED BY

Andrew J. Younge, Anthony M. Agelastos, Aron Warren

Sandia National Laboratories

Unclassified Unlimited Release

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Contributors (alphabetical order)

SNL org. 1422: Scalable Computer Architecture

- Simon “Si” D. Hammond

SNL org. 1423: Scalable System Software

- Gerald “Jay” F. Lofstead
- Kevin Pedretti
- Andrew J. Younge

SNL org. 9326: Scientific Applications & User Support

- Anthony M. Agelastos
- Justin M. Lamb
- Douglas M. Pase

SNL org. 9327: HPC Systems

- Erik A. Illescas
- Jeffry “Jeff” B. Ogden
- Aron Warren

Executive Summary

- ***Hypotheses***:
  - NNSA production applications that execute within HPC container runtimes exhibit native performance characteristics
  - The security criteria for applications on classified networks should allow containers and their runtimes

- ***Methodology***:
  - Build a production application natively and within a container, and compare runtimes and memory footprint
  - Discuss and formulate security requirements with appropriate teams

- ***Results***:
  - Containers with Nalu exhibited near-native performance in runtime (or better) with an extra 10 MB memory usage per rank
  - Containers can run on classified networks provided the relevant security criteria are met

Containers 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 resources
- **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

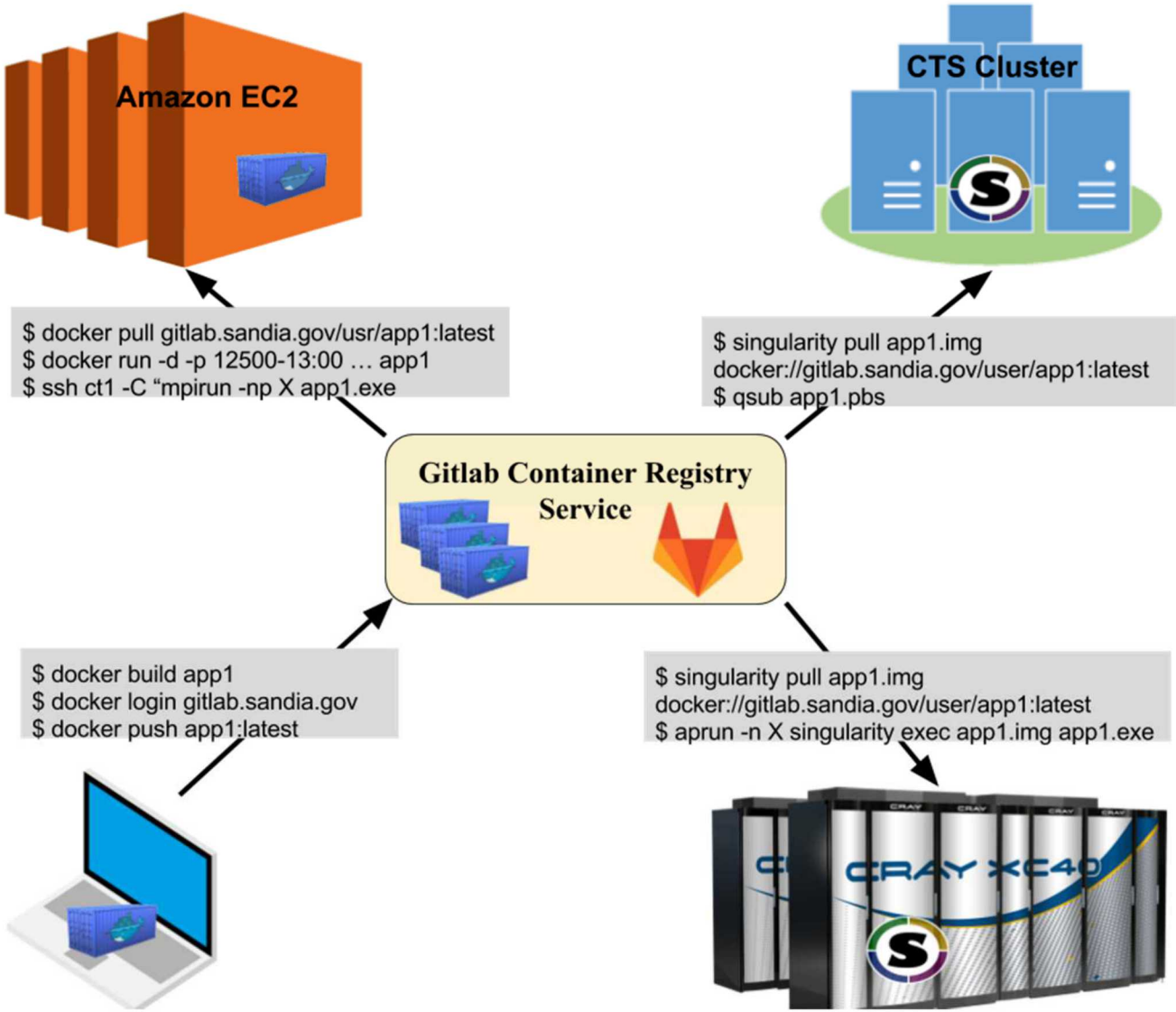
- **Overhead**
  - HPC applications cannot incur significant overhead from containers
- **Micro-Services**
  - Micro-services container methodology does not apply to HPC workloads
  - 1 application per node with multiple processes or threads per container
- **On-node Partitioning**
  - On-node partitioning with cgroups is not necessary (yet?)
- **Root Operation**
  - Containers allow root-level access control to users
  - On 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

Developing Container Vision @ Sandia

- Support software dev and testing on laptops
  - Working builds that can run on supercomputers
  - Dev time on supercomputers is extremely expensive
  - 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
- Want to manage permutations of architectures and compilers
  - x86 & KNL, ARMv8, POWER9, etc.
  - Intel, GCC, LLVM
- Performance matters!
  - Use HPC to “shake out” container implementations on HPC
  - Keep features to support future complete workflows

Container DevOps

- Impractical for apps to use large-scale supercomputers for DevOps and 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

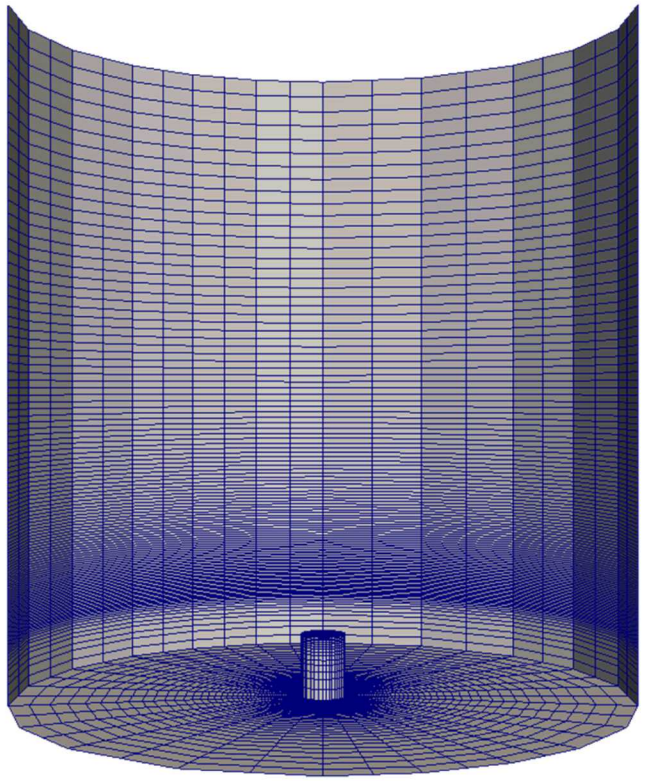


Singularity Containers

- Many different container options
  - Docker, Shifter, Singularity, Charliecloud, etc. etc.
- Standard Docker containers not good fit for running workloads
  - Security issues, no HPC integration
- Singularity best fit for our current 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 community support

Problem Description

- **SNL Nalu:**
  - A generalized unstructured massively parallel low Mach flow code designed to support energy applications of interest [1]
  - Distributed on GitHub under 3-Clause BSD License [2]
  - Leverages the SNL Sierra Toolkit and Trilinos libraries
    - Similar to bulk of SNL Advanced Simulation and Computing (ASC) Integrated Codes (IC) and Advanced Technology, Development, and Mitigation (ATDM) project applications



- **Milestone Simulation:**
  - Based on “milestoneRun” regression test [3] with 3 successive levels of uniform mesh refinement (17.2M elem.), 50 fixed time steps, and no file system output
  - Problem used for Trinity Acceptance [4] and demonstrated accordingly on Trinity HSW [5] and KNL [6], separately, at near-full scale

1. S. P. Domino, "Sierra Low Mach Module: Nalu Theory Manual 1.0", SAND2015-3107W, Sandia National Laboratories Unclassified Unlimited Release (UUR), 2015. https://github.com/NaluCFD/NaluDoc
2. “NaluCFD/Nalu,” https://github.com/NaluCFD/Nalu, Sep. 2018.
3. “Nalu/milestoneRun.i at master,” https://github.com/NaluCFD/Nalu/blob/master/reg_tests/test_files/milestoneRun/milestoneRun.i, Sep. 2018.
4. A. M. Agelastos and P. T. Lin, “Simulation Information Regarding Sandia National Laboratories’ Trinity Capability Improvement Metric,” Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2013-8748, October 2013.
5. M.Rajan, N.Wichmann, R.Baker, E.W.Draeger, S.Domino, C. Nuss, P. Carrier, R. Olson, S. Anderson, M. Davis, and A. Agelastos, “Performance on Trinity (a Cray XC40) with Acceptance Applications and Benchmarks,” in Proc. Cray User’s Group, 2016.
6. A. M. Agelastos, M. Rajan, N. Wichmann, R. Baker, S. Domino, E.W. Draeger, S. Anderson, J. Balma, S. Behling, M. Berry, P. Carrier, M. Davis, K. McMahon, D. Sandness, K. Thomas, S. Warren, and T. Zhu, “Performance on Trinity Phase 2 (a Cray XC40 utilizing Intel Xeon Phi processors) with Acceptance Applications and Benchmarks,” in Proc. Cray User’s Group, 2017.

System Description

- **SNL Doom:**
  - CTS-1 HPC platform
  - Dual E5-2695 v4 (Broadwell) processors, with AVX2, per node
  - 18 cores (36 threads) per processor, 36 cores (72 threads) total per node
  - Core base frequency 2.1 GHz, 3.3 GHz max boost frequency
  - 32 KiB instruction, 32 KiB data L1 cache per core
  - 256 KiB unified (instruction + data) L2 cache per core
  - 2.5 MB shared L3, 45 MiB L3 per processor
  - 4 memory channels per processor (8 per node)
  - DDR4 2400 MHz/s
  - 512 GB per node
  - Intel Omni-Path HFI Silicon 100 Series (100 Gb/s adapter) for MPI communications

Build & Environment Description

- **Doom Software Stack:**
  - TOSS 3.3-1 (~RHEL 7.5)
  - gnu-7.3.1, OpenMPI 2.1.1
  - hwloc-1.11.8
- **Container Software Stack:**
  - CentOS 7.5.1804 (~RHEL 7.5)
  - gnu-7.2.0, OpenMPI 2.1.1
  - hwloc-1.11.1
  - olv-plugin

Nalu Dependencies:

- zlib-1.2.11
- bzip2-1.0.6
- boost-1.65.0
- hdf5-1.8.19
- pnetcdf-1.8.1
- netcdf-4.4.1
- parmetis-4.0.3
- superlu_dist-5.2.2
- superlu-4.3
- suitesparse-5.1.0
- matio-1.5.9
- yaml-cpp-0.5.3
- Trilinos-develop-7c67b929
- Nalu-master-11899aff

- **Notes:**
  - Built with Docker, imported into Singularity
  - Container images available on Sandia GitLab repository

Open Source Software Stack Enables Greater Collaboration and Testing Across Networks and Systems

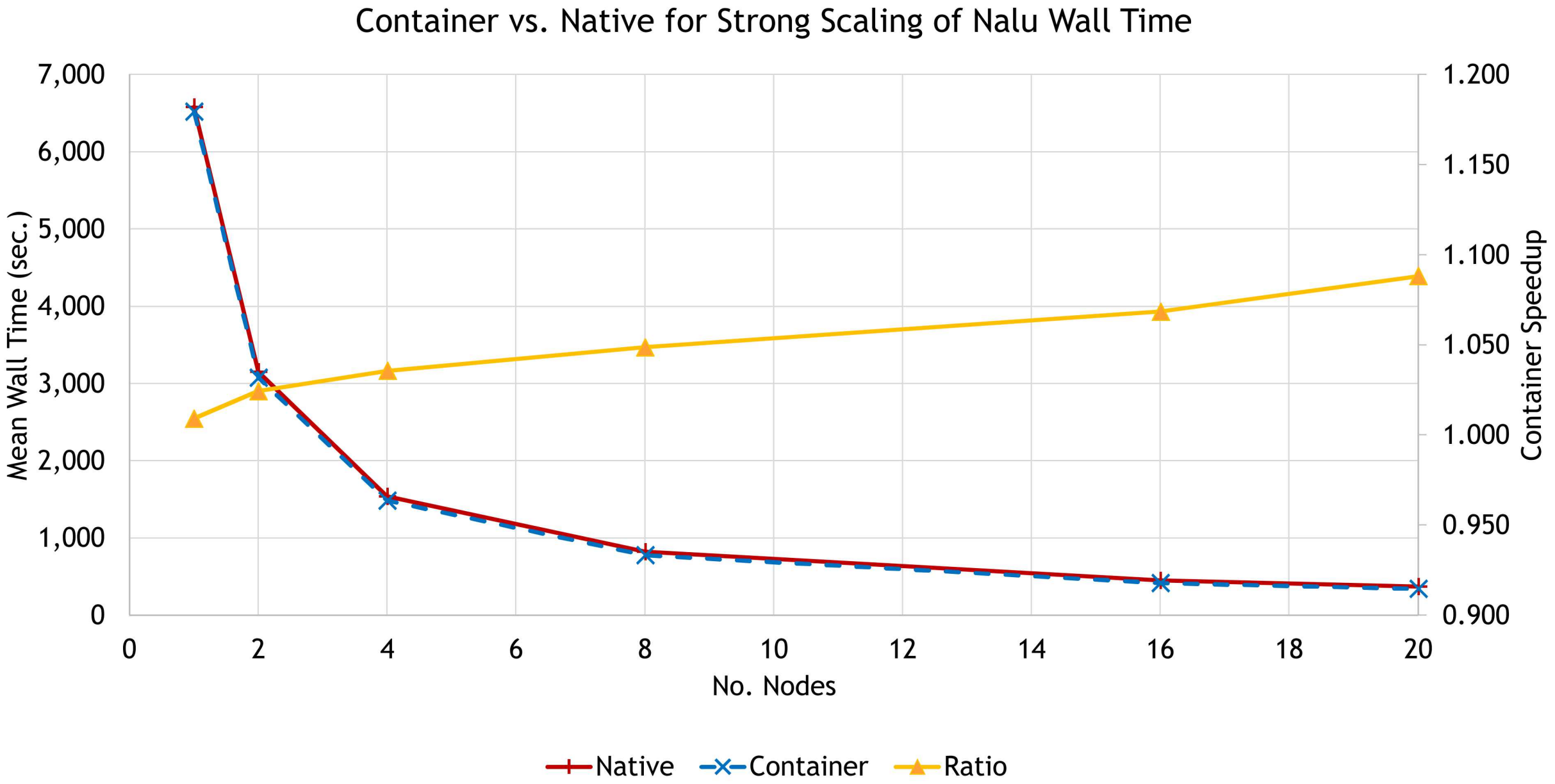
Performance Testing Description

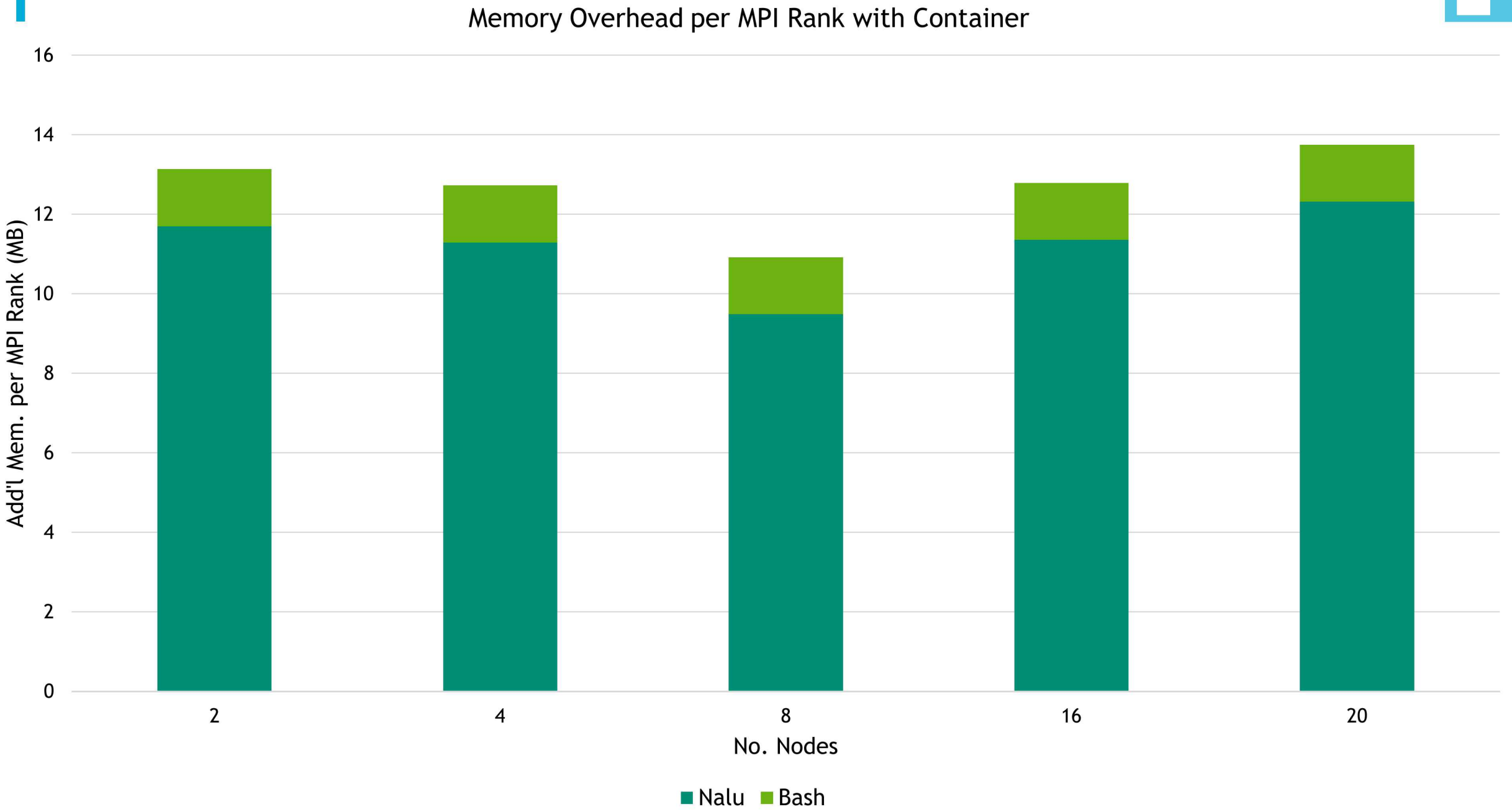
- **Test Characteristics:**
  - Strong scale problem on 1, 2, 4, 8, 16, and 20 nodes
    - 36 MPI ranks per node
    - No threading
    - Bind ranks to socket
  - Measure the the `mpiexec` process wall time for both native and container simulations
  - Extract the maximum resident set size (MaxRSS) for the `mpiexec` process and all of its sub-processes on the head node for both native and container simulations
    - We want to gather all overhead the Singularity container runtime imposes over a native simulation
    - The tested methodology for the Singularity container simulation is:

```
mpiexec -> singularity exec -> bash -> nalu
```

  - Extract the maximum resident set size (MaxRSS) for all of the Nalu MPI processes across all nodes and compute the “average” MaxRSS for Nalu
    - This value is computed for both the native and container simulations so that the former can be subtracted from the latter to compute the container overhead
    - This was extracted via LDPXI using LD_PRELOAD to attach to the native and containerized Nalu processes; LDPXI extracts this via `ru_maxrss` from `getrusage()` at the end of the simulation

Wall Time and Memory Are Key Performance Parameters for Production Workloads


Container vs. Native for Strong Scaling of Nalu Wall Time
Mean Wall Time (sec.)
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Container Speedup
1.200
1.150
1.100
1.050
1.000
0.950
0.900
0
2
4
6
8
10
12
14
16
18
20
No. Nodes
Native
Container
Ratio


Memory Overhead per MPI Rank with Container
Add'l Mem. per MPI Rank (MB)
16
14
12
10
8
6
4
2
0
2
4
8
16
20
No. Nodes
Nalu
Bash

Container vs Native Analysis

- The container was faster, but used more memory?!

- Dynamic linking of GCC 7.2 in container vs system GCC 4.8 Memory usage
  - Memory differences: gfortran & stdlibc++ libraries
    - GCC 7.2 libs much larger, ~18MB total
  - Performance differences: OpenMPI libs
    - Container's OpenMPI w/ GCC7 provides usempif08 in OpenMPI
    - usempif08 iincludes MPI3 optimizations vs MPI2 with usempi
- Position Independent Code (-fPIC) used throughout container compiles
  - Provides larger .GOT in memory, but often slightly improved performance on x86_64
- Overhead with using bash in container to load LD_LIBRARY_PATH before exec
  - Constant but small, depends on .bashrc file

Demonstrates both the power and pitfalls of building your own HPC application environment in containers

Container support at Sandia

- Containers are primarily built on unclassified networks then moved to classified networks via automated transfers.
- Cybersecurity approvals necessary to run containers on unclassified and classified networks.
- Security controls used in running containers on HPC systems.
- Automated Transfer Services to Classified Networks
- Challenges of automated transfers
  - Size – 5GB-10GB are ideal
  - Integrity – md5 is enough
  - Availability – who are you competing against?
  - Transfer policies – executables, code, etc.
- Takeaway: Containers will fully work with automated transfers.

Conclusions

- Containers are a powerful tool for developing & deploying NNSA HPC applications
- Singularity used to deploy containers on production CTS resources
- Nalu exhibited near-native performance within a Singularity container
  - Performance was faster (1-9%) in container due to newer libs
- The Singularity runtime added minimal memory usage overhead
  - The overhead was, approx. ~10MB per rank
- There is a path forward for using containers on restricted and classified networks

Future Work

- Deployment of Singularity on Astra ARM supercomputer
  - Develop/integrate remote builder for ARM platforms
  - Evaluate scaling requirements
  - Investigate new Singularity 3.X features
  - Build ATSE software stack in container, with TOSS base image
- Improve DevOps model
  - Continuous Integration (CI) coordination with containers in Gitlab
  - Integration with Spack packaging, site-specific configs
- Foster interoperability across deployments and container runtimes

Questions?