SC 19 Tutorial: Getting Started with Containers on HPC

Shane Canon[1], Sameer Shende[2], Carlos Eduardo Arango[3] , Andrew J. Younge[4]

[1]Lawrence Berkeley National Lab scanon@lbl.gov	[2]University of Oregon sameer@cs.uoregon.edu
[3]Sylabs Inc eduardo@sylabs.io	[4]Sandia National Labs ajyoung@sandia.gov

Outline

- **13:30 – 13:45 Introduction to Containers in HPC (Younge)**
- 13:45 – 14:15 How to build your first Docker container (Canon)
- 14:15 – 14:45 How to deploy a container on a supercomputer (Canon)
- 14:45 – 15:00 Best Practices (Canon)
- 15:00 – 15:30 -- Break –
- 15:30 – 16:00 Running an HPC app on the E4S container (Shende)
- 16:00 - 16:30 How to build a Singularity container image (Arango)
- 16:30 - 16:50 Running Singularity on a supercomputer & adv features (Arango)
- 16:50 - 17:00 Success Stories & Summary (Canon)

Link: https://tinyurl.com/sc19tut

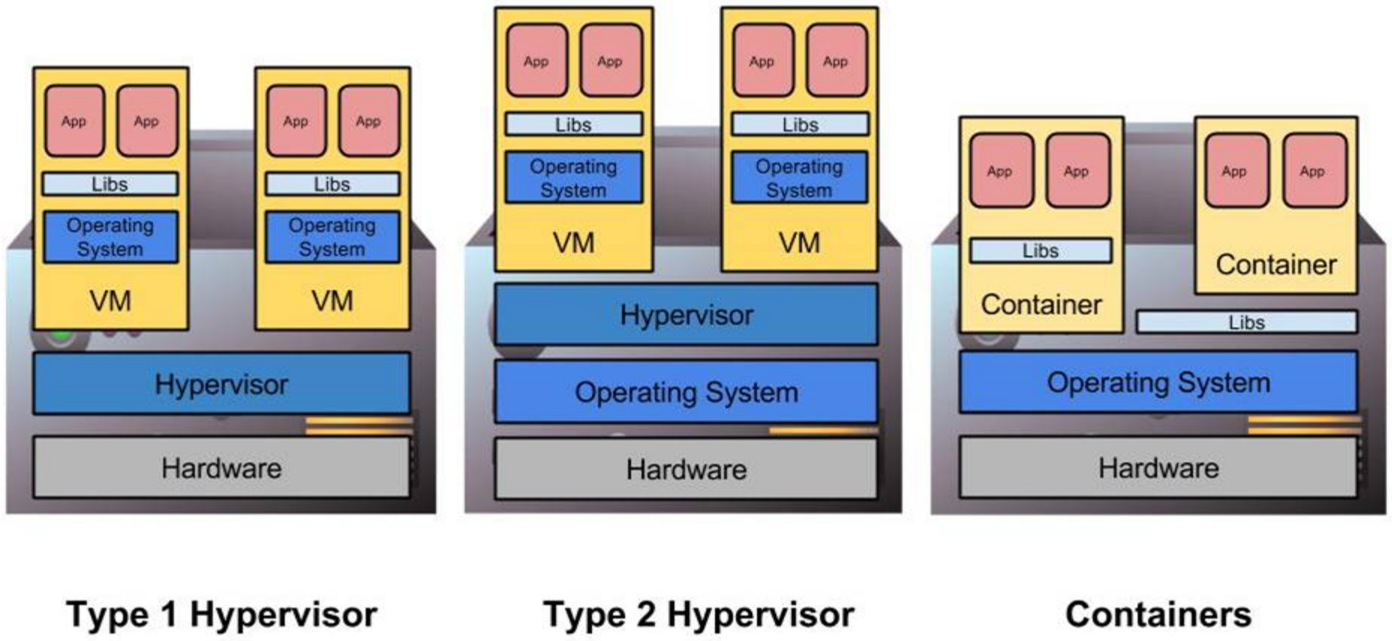
Introduction to Containers in HPC

What are containers

- A lightweight collection of executable software that encapsulates everything needed to run a single specific task
  - Minus the OS kernel
  - Based on Linux only
- Processes and all user-level software is isolated
- Creates a portable* software ecosystem
- Think `chroot` on steroids
- Docker most common tool today
  - Available on all major platforms
  - Widely used in industry
  - Integrated container registry via Dockerhub

Hypervisors and Containers

- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware



EXASCALE COMPUTING PROJECT

Background

- Abstracting hardware and software resources has had profound impact on computing
- Virtual Machines to Cloud computing in the past decade
  - Early implementations limited by performance
  - HPC on clouds: FutureGrid, Magellan, Chameleon Cloud, Hobbes, etc
  - Some initial successes, but not always straightforward
- OS-level virtualization a bit different
  - User level code packaged in container, can then be transported
  - Single OS kernel shared across containers and provides isolation
  - Cgroups traditionally multiplexes hardware resources
  - Performance is good, but OS flexibility is limited

Containers in Cloud Industry

- Containers are used to create large-scale loosely coupled services
- Each container runs just 1 user process – “micro-services”
  - 3 httpd containers, 2 DBs, 1 logger, etc
- Scaling achieved through load balancers and service provisioning
- Jam many containers on hosts for increased system utilization
- Helps with dev-ops issues
  - Same software environment for developing and deploying
  - Only images changes are pushed to production, not whole new image (CoW).
  - Develop on laptop, push to production servers
  - Interact with github similar to developer code bases
  - Upload images to ”hub” or “repository” whereby they can just be pulled and provisioned

Containers

- Containers are gaining popularity for software management of distributed systems
- Enable way for developers to specify software ecosystem
- US DOE High Performance Computing (HPC) resources need to support emerging software stacks
  - Applicable to DevOps problems seen with large HPC codes today
  - Support new frameworks & cloud platform services
- But HPC systems are very dissimilar from cloud infrastructure
  - MPI-based bulk synchronous parallel workloads are common
  - Scale-out to thousands of nodes
  - Performance is paramount

Container features 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
  - 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

HPC Containers

- Docker not good fit for running HPC workloads
  - Security issues
    - Can’t allow root on shared resources
  - Lack of HPC architecture support
    - No batch integration
    - Assumes local resources
    - Assumes commodity TCP/IP
- Many different container options in HPC

Shifter Singularity Charliecloud …

Developing Container Vision

- Support software dev and testing on laptops
  - Working builds that then can run on supercomputers
  - Dev time on supercomputers is expensive
  - May also leverage VM/binary translation
- Let developers specify how to build the env AND app
  - Import and run container on target platform
  - Many containers, but can have different code "branches"
  - Not bound to vendor and sysadmin software
- Focus on Interoperability
- Provide containerized services coupled with simulations
  - Developing mechanisms to support services
- Performance matters
  - Want to manage permutations of architectures and compilers
  - Ensure container implementations on HPC are performant
  - Keep features to support future complete workflows

Container DevOps

- 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

This tutorial will show you:

- How to build your first Docker container.
- How to run a Docker container on a supercomputer with Shifter.
- How to build your first Singularity container.
- How to run a container on a supercomputer with Singularity.
  – And work with some Sylabs cloud features
- How to use the Extreme-scale Scientific Software Stack (E4S) container image.
  – And a bit about Spack
- And maybe some best practices and lessons learned.

Tutorial Link

https://tinyurl.com/sc19tut

https://supercontainers.github.io/sc19-tutorial/

Tutorial Training Accounts

1. EC2 instance login
2. Cori training account

Questions?

Next: learn how to work with your first container!