Linux Containers for HPC Code Mobility

Ketan (@pitt.edu) M.
Center for Research Computing (CRC)
University of Pittsburgh

Outline

part 1: Overview

part 2: Technical Details

part 3: Use Case: Container Solution to an HPC Problem

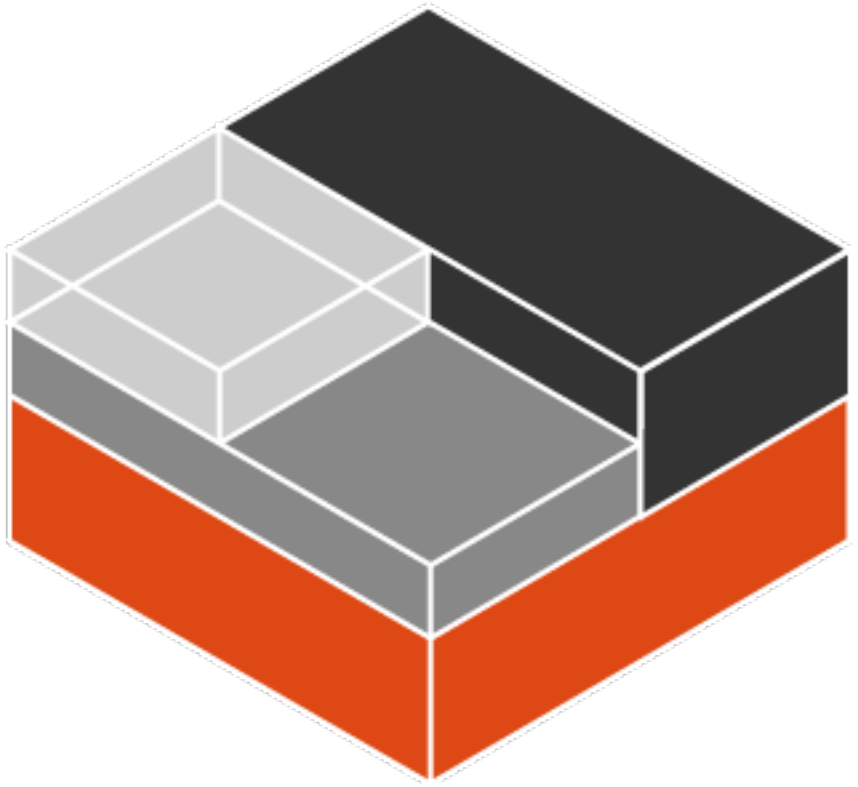
part 4: Challenges, Opportunities and Conclusions

part 1: Overview

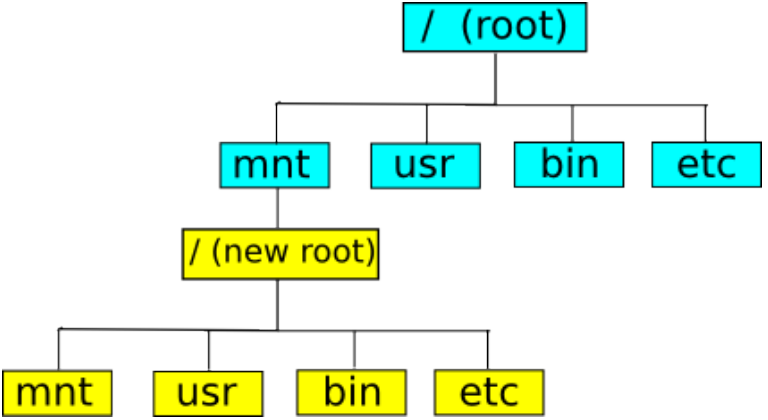
back to toc

What are Linux Containers?

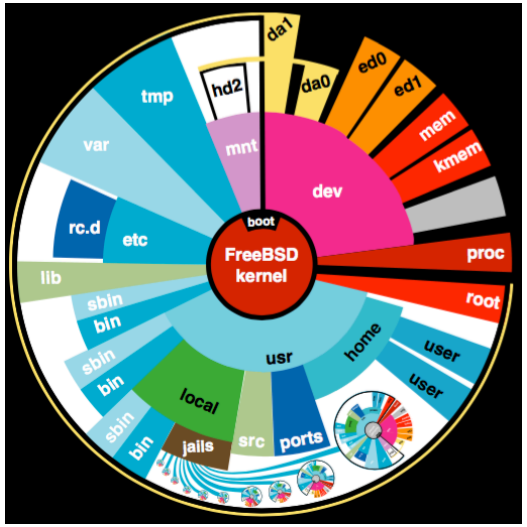
- Linux containers (containers from now) are light-weight packages that run full application stack in a self-contained environment.
- Multiple containers may coexist on a single OS, **live as files** and run in isolation.
- Users have higher control over how they want to run application by customizing container for dependencies.
- Gained popularity due to maturing kernel features, industrial acceptance and emergence of cloud.



pic. courtsey: https://linuxcontainers.org



History



1972 IBM VM

1979 chroot

1982 FreeBSD Jails

linux openVZ

2001 Linux VServers

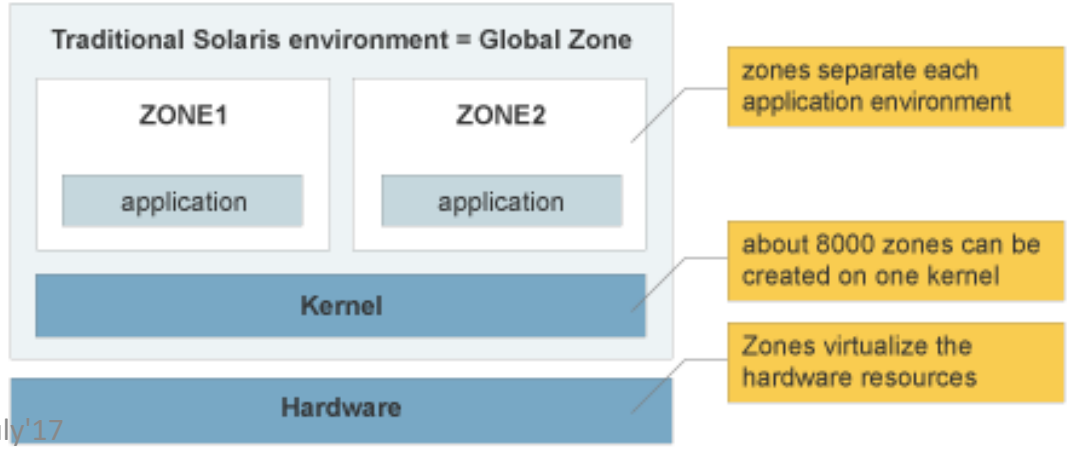
User Mode Linux (UML)

2005 Solaris Zones

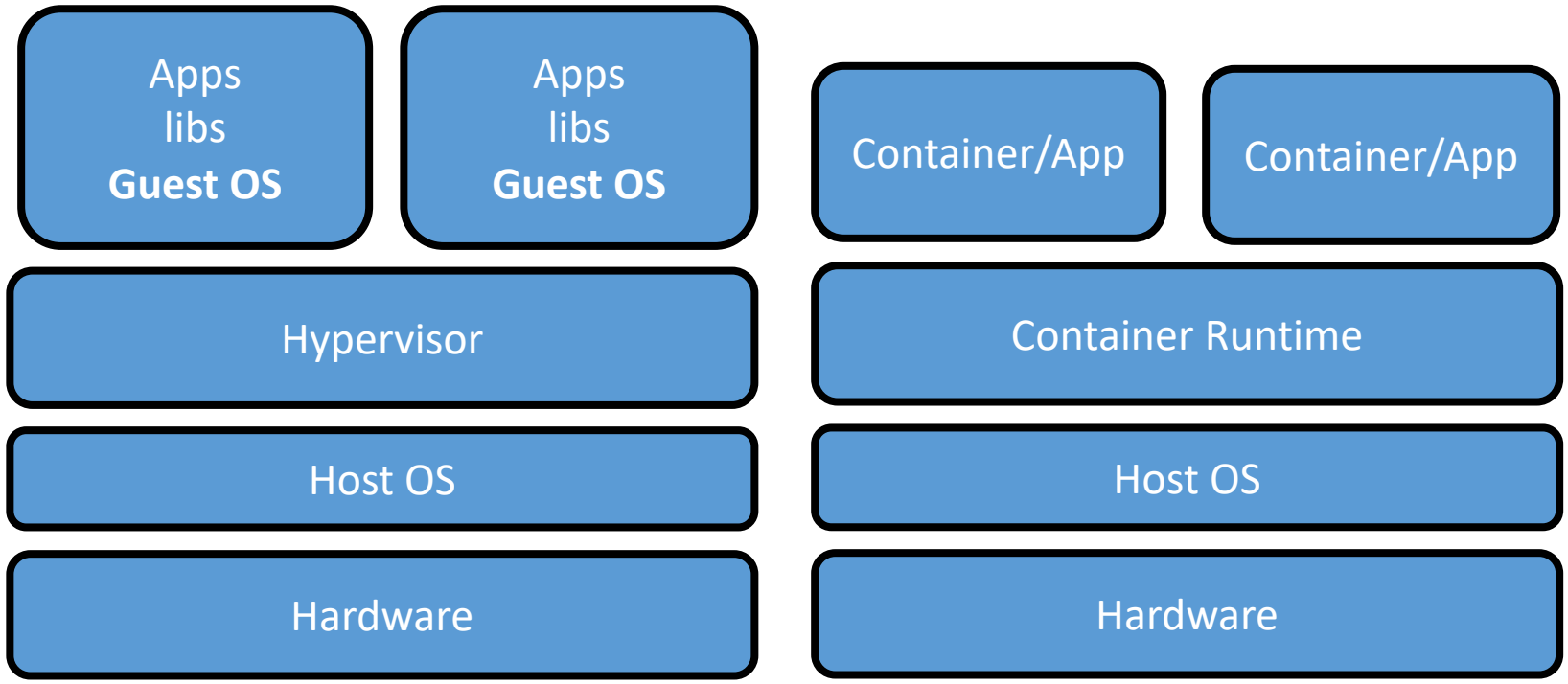
AIX Workload partitions

2007 Py Virtual env

Today LXC, Docker, Singularity



How containers differ from VMs



- Lightweight: no hypervisor
- Shared kernel: no guest OS
- software based virtualization
- Easier to spin up & tear down
- fast and low-cost
- Live migration possible

What challenges do containers address?

- Deploying a consistent production environment is easier said than done. Containers bring uniformity between development and production.
- Application portability is a constantly moving target--pack application in a self contained box && **escape the "dependency matrix"**!
- Fast and lightweight--no need to stand up a whole VM just to run one application.
- Scalable and cheap--overhead almost similar to Linux processes.

part 2: Technical Details

back to toc

Kernel Features that enable Containers

- **Control groups**: limits host hardware resources--CPU, memory, I/O
- **Namespaces**: process-ification, isolation, security
- **Pivot_root (2)**: create new environment by changing the root file system.
- **Capabilities** enforce fine tuning of access control and namespaces.
- **Mandatory Access Controls (MAC)**: security enabler via a tag based architecture eg. SELinux.

Control Groups (cgroups)

- Cgroups let kernel control the amount of resources used by a container--similar to **ulimit.**
- Each resource has a usage tree, eg. CPU, memory, block I/O
  - each resource type is root and process is a node
- Kernel keeps track of how much cpu, memory, I/O is used by each branch and sets hard-limits.
- If a process within container tries using resources beyond a hard-limit they will be killed by kernel.
- A soft-limit may be used as a guide to migrate container to another machine.

Namespaces

- Fundamental building block for isolation.
- Multiple namespaces--pid, user, net, mnt, uts, ipc
- Namespaces let containers have their own--processes, hostname, network stack, mount points, and more.
- Implemented using **`clone(2)`** flags (from man page, emphasis mine) :

  `CLONE_NEWPID (since Linux 2.6.24)`

  `If CLONE_NEWPID is set, then create the process in a` **`new PID namespace.`** `…` **`This flag is intended for the implementation of containers.`**

- Everything is not namespaced yet -- eg. devices, syslog, procfs
  - security implication: shared namespace between kernel and container

pivot_root

- System call that creates the new environment for container by changing the root filesystem.
- From the man page (emphasis mine):

```
int pivot_root(const char *new_root, const char *put_old);
pivot_root moves the root file system of the calling process to
the directory put_old and makes new_root the new root file
system of the calling process.
```

- Pivot_root implementation is evolving (from the man page):

```
The paragraph above is intentionally vague because the
implementation of pivot_root() may change in the
future.
```

Container security issues

- Container escape: system call parameters may let container user escalate privileges to host.
- DoS attacks: user may jam the host by running resource intensive processes if controls are not in place.
- Functions for container creation are suid'ed -- a user's privileges are elevated while running those functions which may be taken advantage of.
- Poisoned images may cause kernel panic killing hosts.

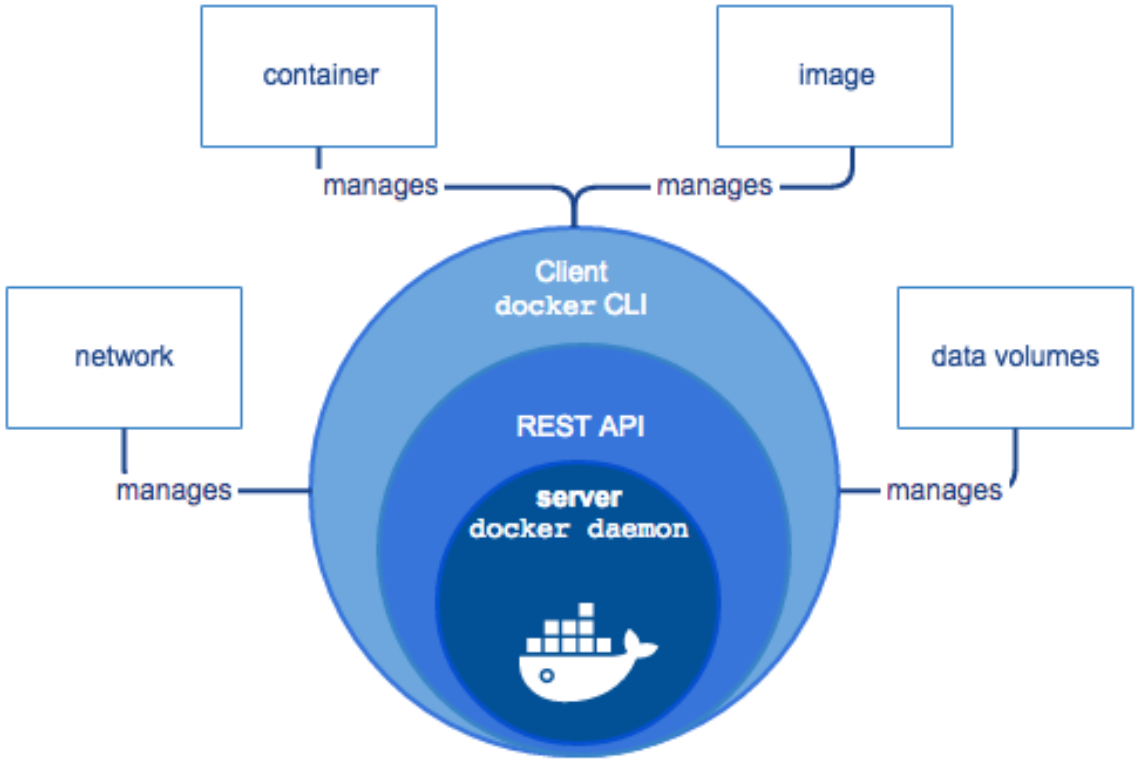
- **Summary**: main cause for security issues is a shared kernel! Likewise hardening the kernel is main solution.

Docker and Singularity

- Both **Docker** and **Singularity** are popular container management solutions.
- There are some similarities and key differences.
- We explore the features and compare them in subsequent slides.

Docker overview

- Largest and most popular provider of the container based services and products.
- Widely adopted by industry -- has become synonymous to 'container'
- Maintains and offers a repository of general purpose 'images'-- Docker hub.
- Started with LxC, later switched to runC as container runtime.



pic. courtesy: https://docker.com

Singularity overview

- Singularity enables users to create, manage and distribute Linux Containers.
- Arguably the largest initiative to offer container management in a scientific computing context.
- Allows user level access (no root access required) to the containers and **native access to filesystems, network, hardware and storage.**
- A singularity hub is under development.

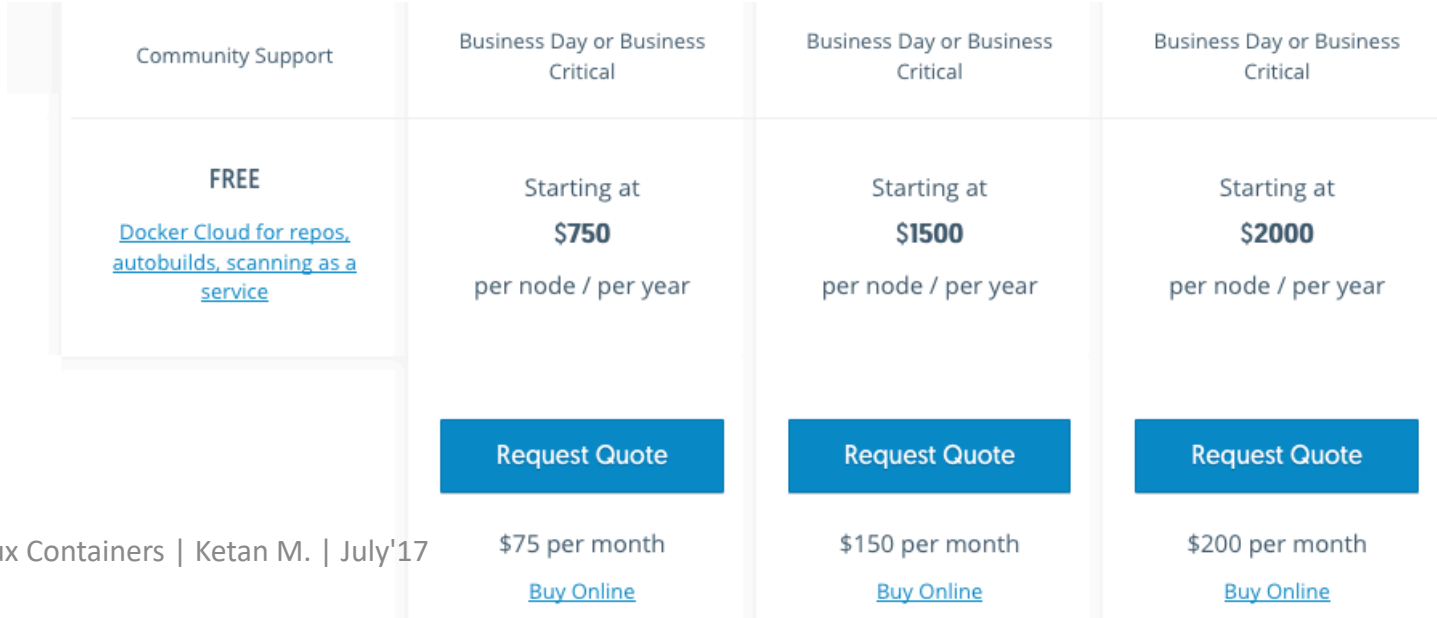
pic. courtsey: https://singularity.lbl.gov

Docker vs Singularity

Docker	Singularity
Use Case: Commercial product for DevOps, micro-services	Use Case: Scientific, HPC
Runs in a client-server setup--daemon needs root; communication via REST api	Runs as a standalone binary -- simpler model
Reuse: Docker hub repository	Reuse: Purpose built for applications, works with Docker images
Works on Linux, Windows and Mac, installs from binary/package managers, source	Works on Linux distros; support for resource managers, eg. SLURM
Large scale operations in the cloud, wide industrial adoption	Well adopted by academia and government HPC centers, bio community

Choice between Docker and Singularity

- Orientation
  - Must suit HPC, Scientific computing
  - HPC scalability issues with Docker
- Community
  - Who are currently using Docker? Singularity?
  - What are the best practices?
- Support
  - Community vs commercial
  - updates and upgrades
- Cost
  - Docker is not free!



part 3: Use Case: Container Solution to an HPC Problem

back to toc

Situation

- User requests installation of GPU enabled Tensorflow.
- While trivial on modern distros (using pip), extremely tedious to install from source on older distros.
- Some of the complexities involved are:
  - glibc dependency (must have **glibc v 2.17+**)
  - involves multiple tool-chains: **gcc + cuda**, **java** (bazel build), **python** (TF is a python library)
- Build process is **fragile** involving manual edits to files.

Containers to the rescue

Approach: Install Singularity

```
$ git clone
https://github.com/singularityware/singularity.git

$ cd singularity && ./autogen.sh

$ ./configure --with-userns --with-slurm
              --prefix=$HOME/singularity-install
              --sysconfdir=$HOME/singularity-install/conf

$ make
$ sudo make install

$ export PATH=$PATH:$HOME/singularity-install/bin

# Sanity tests
$ singularity shell docker://ubuntu:latest

$ sudo singularity create container.img
$ singularity -v exec container.img /bin/date
```

Build and Test the TF Container

```
$ IMG=ubuntu_tensorflow_GPU.img
$ DEF=ubuntu.def

$ singularity create $IMG
$ singularity expand --size 5000 $IMG
$ sudo singularity bootstrap $IMG $DEF

$ sudo singularity exec -B `pwd`:/mnt -w $IMG sh
/mnt/tensorflow.sh

$ singularity exec ubuntu_tensorflow_GPU.img python
./test.py
```

The ubuntu.def File

```
BootStrap: debootstrap #use debian base
OSVersion: trusty
MirrorURL: http://us.archive.ubuntu.com/ubuntu/

%runscript
    exec "$@"

%post
    apt-get update
    apt-get -y --force-yes install vim curl libfreetype6-dev \
    libpng12-dev libzmq3-dev python-numpy python-pip python-scipy \
    && apt-get clean
```

The tensorflow.sh file

```
#variables
driver_version=375.20
cuda="cuda_8.0.44_linux.run"
cudnn="cudnn-8.0-linux-x64-v5.1.tgz"
TF_GPU_PKG=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.10.0-cp27-none-linux_x86_64.whl

#install Nvidia stack
sh /mnt/NVIDIA-Linux-x86_64-$driver_version.run -x
mv NVIDIA-Linux-x86_64-$driver_version /usr/local/
sh /mnt/links.sh $driver_version
sh /mnt/$cuda --toolkit --silent
tar xvf /mnt/$cudnn -C /usr/local

#set environment
driver_path=/usr/local/NVIDIA-Linux-x86_64-$driver_version
echo "" >> /environment
echo "LD_LIBRARY_PATH=/usr/local/cuda/lib64:$driver_path:$LD_LIBRARY_PATH" >> /environment
echo "PATH=$driver_path:\$PATH" >> /environment
echo "export PATH LD_LIBRARY_PATH" >> /environment
echo " " >> /environment

#install Tensorflow
pip install --upgrade pip
pip install matplotlib
pip install --upgrade $TF_GPU_PKG
mkdir -p /ihome/sam/ketan #create homedir location
```

Test with Slurm

```
#!/bin/bash

#SBATCH --job-name=gputf
#SBATCH --output=testslurm.out
#SBATCH --error=testslurm.err
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cluster=gpu
#SBATCH --partition=gpu
#SBATCH --gres=gpu:2

module load singularity

singularity exec ubuntu_tensorflow_GPU.img test.py
```

A Short Demo

```
[login1.crc.pitt.edu singularity-Tensorflow]$
```

part 4: Challenges, Opportunities and Conclusions

back to toc

Challenges

- Relatively old tech in new limelight
  - User education and training
  - Awareness
  - Keep up with progress
- An additional admin layer
- Technical issues
  - security
  - bugs and features
- Efforts needed towards
  - Customization
  - Hardening the host

Opportunities

- Platform dependency problem solved for **niche applications** such as science gateways, legacyware.
- Significant improvement in HPC code mobility.
- Research projects surrounding containers
  - Improve usability, security
  - Data-only Containers
  - Reproducible research

Summary/Conclusions

- Container technology is a **game changer** for modern HPC development.
- Dramatically improves porting of applications over diverse infrastructure and vice versa.
- Docker and Singularity are two major players.
- Easy to set up--took me a couple of days to stand up a usable Singularity image from scratch.
- Not without challenges but future seems exciting.

Acknowledgements, Credits and References

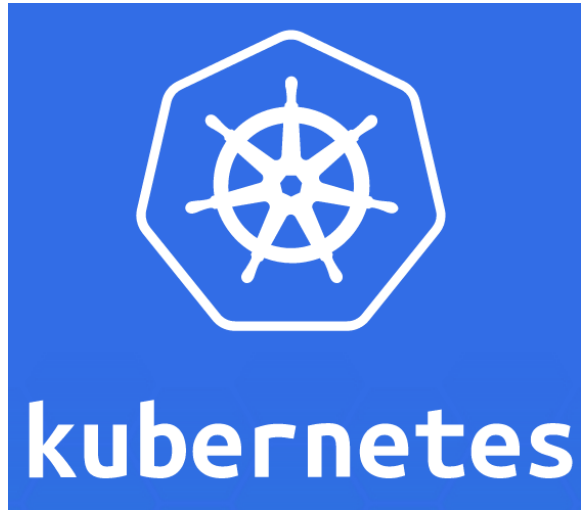
- Thanks to colleagues at Pitt CRC for supporting the initiative.

- Linux Containers: www.linuxcontainers.org
- Docker: www.docker.com
- Singularity: singularity.lbl.gov
- Docker vs Singularity: http://geekyap.blogspot.com/2016/11/docker-vs-singularity-vs-shifter-in-hpc.html
- Singularity at NIH: https://hpc.nih.gov/apps/singularity.html
- Understanding and Hardening Linux Containers: www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/april/understanding-and-hardening-linux-containers

Thank you!

Honorable Mentions

OpenVZ
Linux Containers



Are Containers a Fad or Here to Stay?

Singularity features

- Target environment is an academic HPC setup with scientific user community
- Free of cost!
- May use Docker images if required
- Nothing seems to be missing in terms of features, for instance:
  - Works well with resource managers, eg. SLURM
  - Access to homedir, /opt, /mnt
  - non-root access possible

Capabilities

- In Linux you are either root and can do everything or a non-root and can do nothing!

- Capabilities enable breaking down the root capabilities and restricting some of them within a container so that a container root may not harm the host.

VM vs Container

Virtual Machine	Container
Bulky: uses heavy "hypervisors", -- hardware-level virtualization	Lightweight: no hypervisor, reuses kernel -- OS-level virtualization
Run different OSs' at the bare metal level	Multiple containers share a single OS
Tight isolation and security	Loose isolation, security issues remain
Useful in some cases such as cluster wide host OS (eg. warewulf), and supporting distinct OS families, eg. Linux and Windows	Useful to port application level environments such as full stacks of a web app or a software test environment eg. a Jenkins suite to test all Bio software
Relatively hard to install and upgrade	Relatively easy to spin up and switch to a new container

Why are containers suddenly popular?

- Emergence of the cloud technologies--lightweight virtualization
- Application complexity--include all dependencies in a self-contained package, create a super-static package
- Infrastructure and user diversity sharing hardware
- Industrial acceptance