FALL 2021

TEXAS
The University of Texas at Austin

# CSE380: TOOLS AND TECHNIQUES OF COMPUTATIONAL SCIENCE

[ VMs, Containerization and OpenHPC ]

**DR. KARL W. SCHULZ**
Oden Institute for Computational Engineering and Sciences
The University of Texas at Austin

# Outline

- Virtualization
  - Hypervisors
  - Containerization
  - Docker
- OpenHPC
  - Overview
  - Current software
- Docker + OpenHPC
- Other containerization tools
  - Example on TACC system

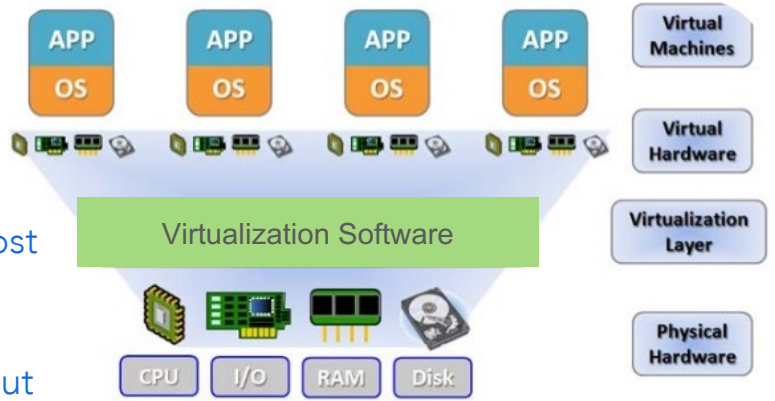
# Motivation

- One challenge we face in scientific computing is the use and interaction of a wide variety of software and revisions
  - tools like "modules" help us on HPC systems
  - hopefully all the s/w we need is available in advance, but what if it's not exactly what we need?
- Moving from system to system can be a challenge when we have large software dependencies
  - catch22: want to leverage as much existing (high-quality) software as possible, but this means we may have lot's of dependencies
  - can impact speed of porting code, restrict choices
  - impacts reproducibility
    - if you want someone else to be able to run your code, they need all the dependencies as well
- Virtualization and containerization is a way we can help manage development on multiple platforms via consistent s/w environment

# Virtualization

- Virtualization (in computing) is the creation of a virtual operating system, server, storage device, network resource, etc
  - we refer to this as a "virtual machine" (VM)

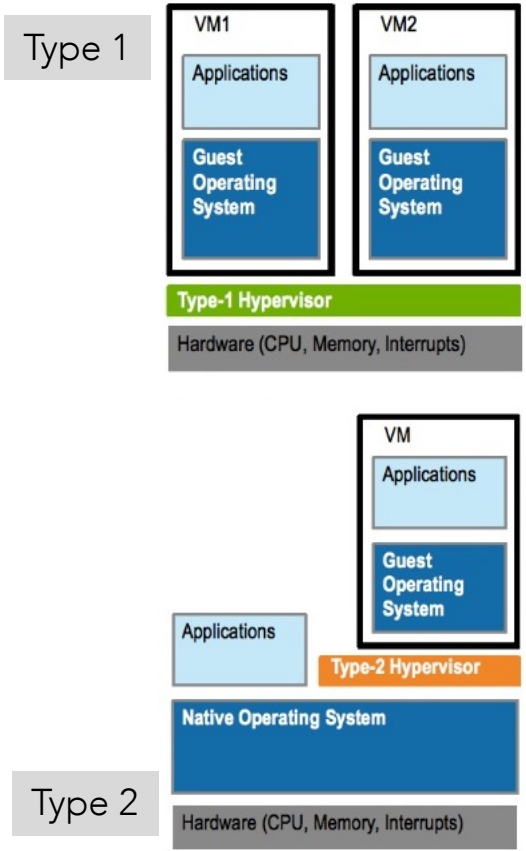
- Hypervisors: combination of software, firmware, or hardware that creates and runs virtual machines
  - allows you to host several VMs on the same physical host
    - each VM can have its own OS, run programs and appear to have its own processors, memory and other physical resources
  - hypervisors have been around for a long time (1960s) but popularized in 2000's via Linux projects (e.g. Xen) and hardware additions for virtualization (e.g. Intel's VT-x)
    - VT-x added 10 virtual machine extension instructions
    - most all modern processors have hardware virtualization features



# Classification of Hypervisors

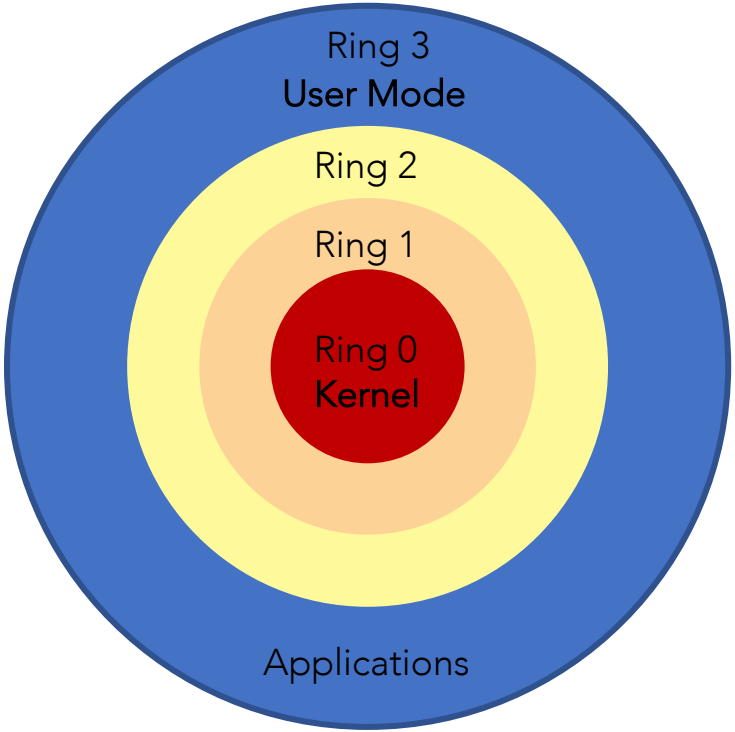
- Type 1: native or bare metal hypervisor
  - runs directly on host hardware to manage guest OS
  - direct access to h/w resources
  - high performance; h/w support may be limited
- Type 2: hosted hypervisor
  - s/w hosted by main OS
  - hypervisor asks OS to make hardware calls
  - can incur performance overheads

- Distinction not always clear
  - KVM (kernel based virtual machine) is a popular open-source solution for Linux that effectively converts the host operating system to a type-1 hypervisor



# OS Privilege Rings and Virtualization

- Operating systems typically provide different levels of access to resources in diminishing priority
  - ring 0 is most privileged (all kernel code runs here)
  - ring 3 is least privileged (all user code runs here)
- Resources being protected
  - memory
  - I/O ports
  - ability to execute certain machine instructions
- Code that runs in ring 3 should be able to fail at any time without impact to the rest of the computer system
- Special gates between rings are provided to allow an outer ring to access an inner ring's resources in a predefined manner
  - hardware severely restricts the ways in which control can be passed from one ring to another
  - also enforces restrictions on the types of memory access that can be performed across rings.



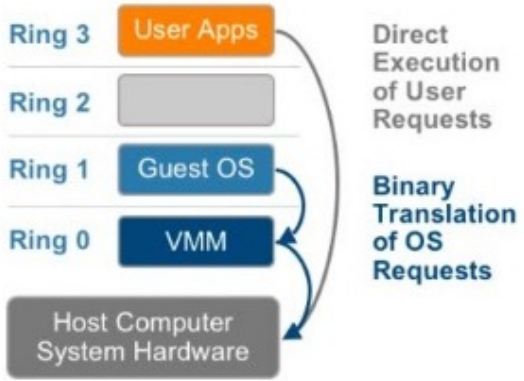

Linux/Windows has ring 0/ring 3

# Virtualization Techniques

Multiple approaches to support a guest operating system:

- full virtualization
  - completely abstracted by virtualization layer
  - guest OS doesn't know it's a guest
  - hypervisor translates all OS calls on the fly
  - no hardware assistance
- paravirtualization
- hardware-assisted virtualization (HVM)

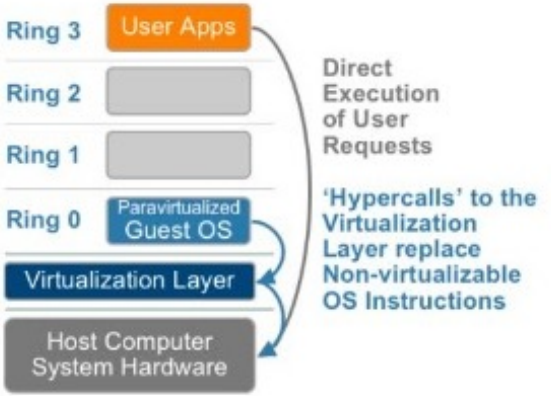
performance



# Virtualization Techniques

Multiple approaches to support a guest operating system:

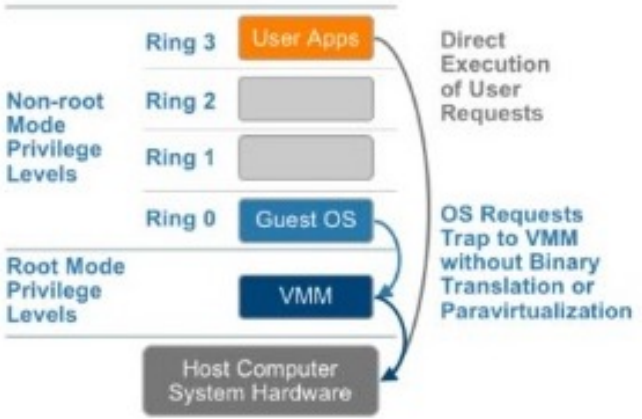
- full virtualization
- paravirtualization
  - lightweight virtualization technique
  - hypervisor provides an API that Guest OS calls (requires **modification** to guest OS)
  - does not require virtualization extensions from host CPU
  - near-native performance
- hardware-assisted virtualization (HVM)



# Virtualization Techniques

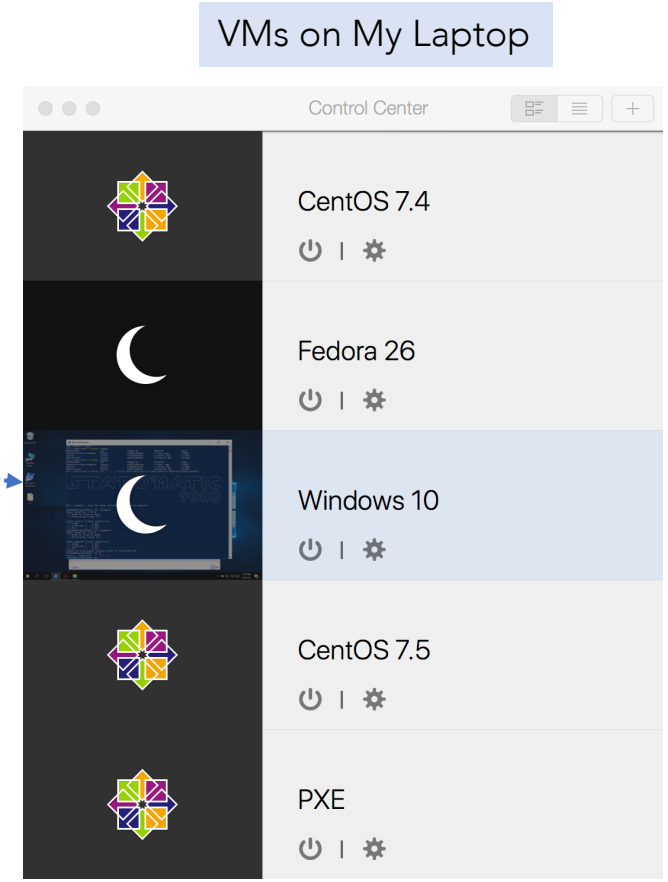
Multiple approaches to support a guest operating system:

- full virtualization
- paravirtualization
- hardware-assisted virtualization (HVM)
  - efficient full virtualization via additional hardware capabilities
    - e.g. Intel VT-X or AMD-V instructions
  - unmodified guest OS
  - hypervisor traps sensitive calls; no binary translation required
  - x86 hardware creates a "Ring -1" so that guest OS can run Ring 0 operations natively without affecting other guests or host OS

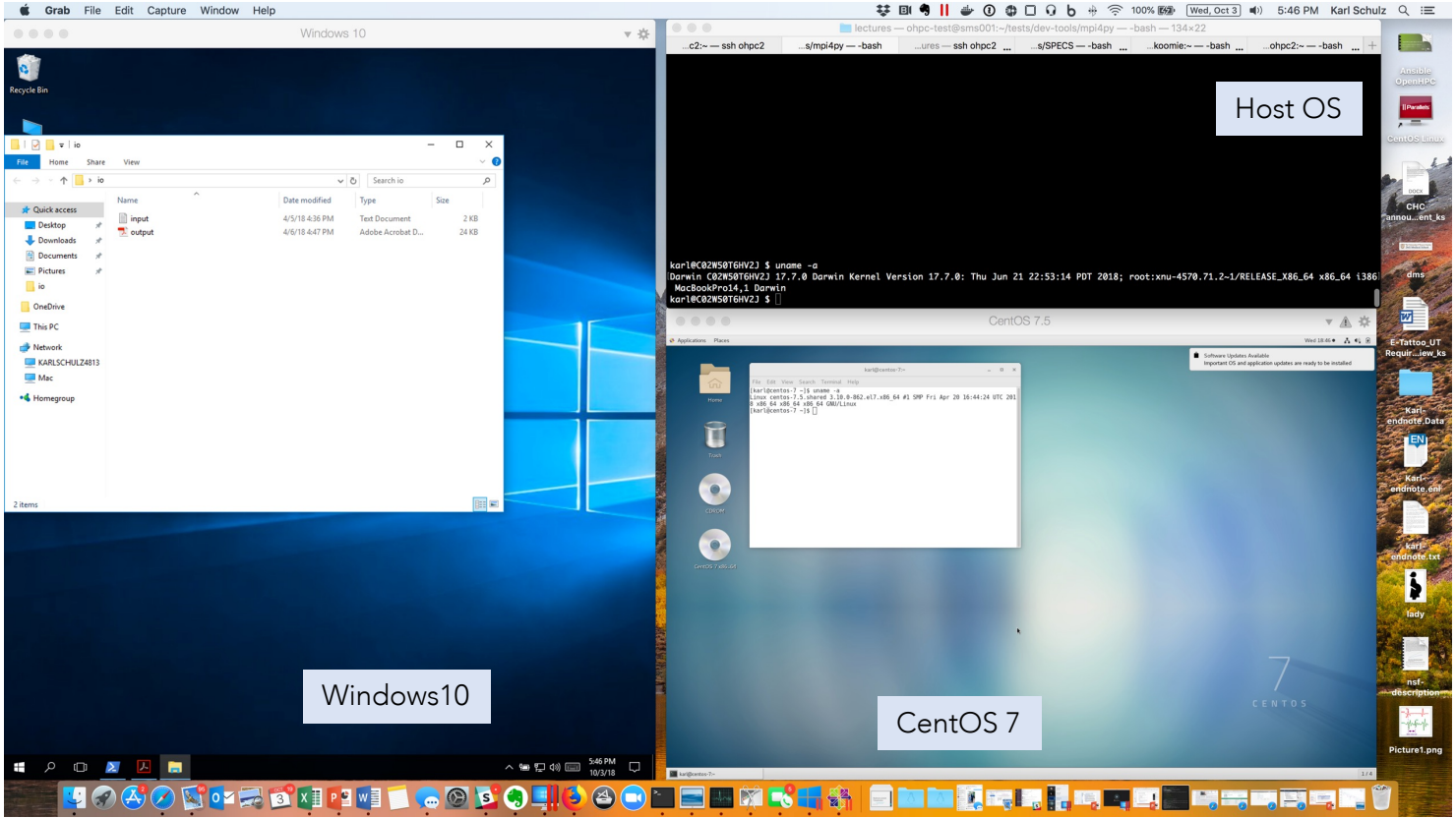


# Some Example Hypervisors

- VMware ESX, VMware Workstation
- Microsoft Hyper-V
- Citrix XenServer
- Oracle VirtualBox (open source, windows, OS X, Linux hosts)
- Parallels (OS X, what I use)
- QEMU
- KVM
- Recall: each guest OS instance is independent:
  - can power them on and off (or suspend them)
  - can snapshot them
  - use h/w resources only when powered on; do consume some disk resources permanently from host OS to define VM image
  - full guest OS, so we can have GUIs, etc



# Example: 2 VMs with Parallels



# What about Cloud Computing?

- Cloud computing is built on top of virtualization
  - built around ease of dynamically spinning up various VMs
  - cloning and consolidation
  - migration: VMs can be seamlessly migrated from one physical host to another

- Example: Amazon EC2
  - if you want to use cloud resources, you first choose an instance type
    - comprised of varying combinations of CPU, memory, storage, and networking capacity
    - amount you pay is based on the instance properties
    - instance types are just different VM configurations
  - AWS uses Xen and KVM hypervisors on bare metal
    - choose from pre-existing Linux images (AMIs)
      - both hardware virtual machine (HVM) and paravirtual (PV) AMIs are available

| Model | vCPU | CPU Credits / hour | Mem (GiB) |
|---|---|---|---|
| t2.nano | 1 | 3 | 0.5 |
| t2.micro | 1 | 6 | 1 |
| t2.small | 1 | 12 | 2 |
| t2.medium | 2 | 24 | 4 |
| t2.large | 2 | 36 | 8 |
| t2.xlarge | 4 | 54 | 16 |
| t2.2xlarge | 8 | 81 | 32 |

| | vCPU | ECU | Memory (GiB) | Instance Storage (GB) | Linux/UNIX Usage |
|---|---|---|---|---|---|
| **General Purpose - Current Generation** | | | | | |
| t3.nano | 2 | Variable | 0.5 GiB | EBS Only | $0.0052 per Hour |
| t3.micro | 2 | Variable | 1 GiB | EBS Only | $0.0104 per Hour |
| t3.small | 2 | Variable | 2 GiB | EBS Only | $0.0208 per Hour |
| t3.medium | 2 | Variable | 4 GiB | EBS Only | $0.0416 per Hour |
| t3.large | 2 | Variable | 8 GiB | EBS Only | $0.0832 per Hour |
| t3.xlarge | 4 | Variable | 16 GiB | EBS Only | $0.1664 per Hour |
| t3.2xlarge | 8 | Variable | 32 GiB | EBS Only | $0.3328 per Hour |
| t2.nano | 1 | Variable | 0.5 GiB | EBS Only | $0.0058 per Hour |
| t2.micro | 1 | Variable | 1 GiB | EBS Only | $0.0116 per Hour |
| t2.small | 1 | Variable | 2 GiB | EBS Only | $0.023 per Hour |
| t2.medium | 2 | Variable | 4 GiB | EBS Only | $0.0464 per Hour |

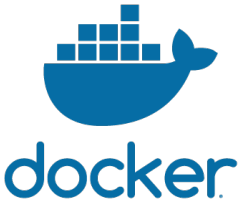
example fall 2018 pricing (on-demand)

# Virtualization

- VMs are great when you want/need full OS
- Performance degradations have reduced dramatically, can be nearly the same as bare-metal
- But…
  - does require elevated credentials to install and manage (we don't generally have that on HPC machines)
  - full OS images can eat up significant disk space over time
  - takes some time to spin up (since we are booting a full OS)
- Q: For our purposes, are there other virtualized solutions we can leverage to have fine-grained control over the software stack?
  - indeed; Linux containers have emerged as another lightweight mechanism for packing, shipping, and running your application in a controlled software environment

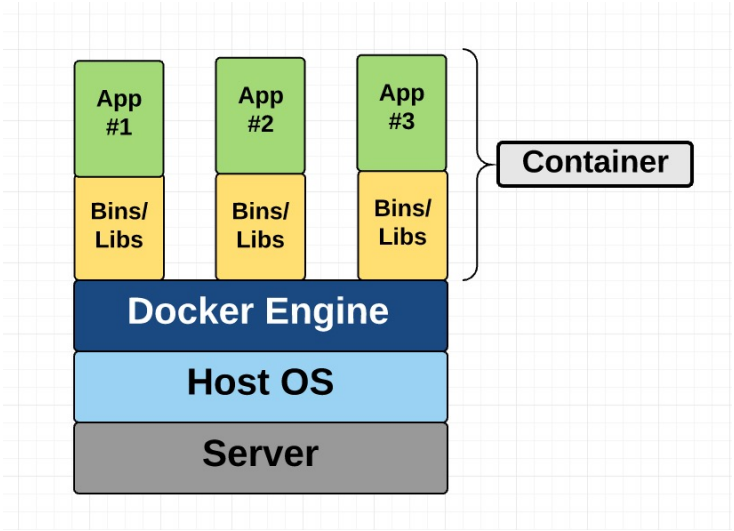
# Containers

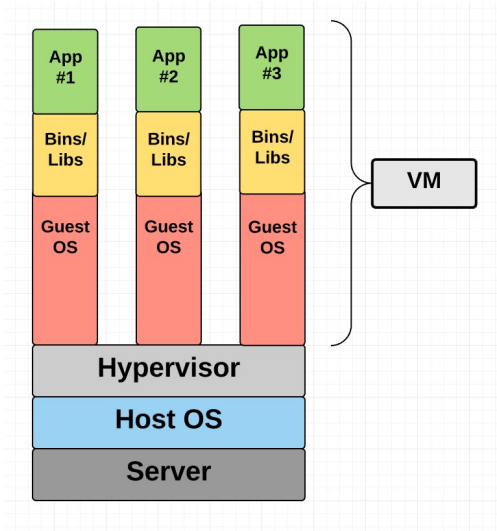
- Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the “user space”
- They *"look and feel"* like a VM
  - have private space for processing
  - can execute commands as root
  - have a private network interface and IP
  - can mount file systems, BUT…
  - containers *share* the host system's kernel with other containers
- One very popular container in use today is Docker
  - open source
  - uses Linux kernel features like namespaces and control groups to create containers on top of OS
  - rich community ecosystem for sharing container environments (Docker Hub)
  - frequent s/w engineering use for automation and continuous integration testing




https://www.docker.com

# Containers vs. VMs




- *Container Goal: to isolate an **application** and its dependencies into a self-contained unit that can run anywhere*
  - *isolates user-space*
  - *OS shared across containers*
  - *take fewer resources and can spin up faster than VMs (don't have to boot up a full OS every time)*

# Docker Installation (Docker Desktop)

- Standard clients available for macOS, Windows, and popular Linux distributions

- Note: Docker for Windows requires 64bit Windows 10 Pro, Enterprise, or Education
  - if you have older Windows variant, can use Docker Toolbox which uses Oracle VirtualBox instead of Microsoft Hyper-V to create virtual machines
  - More info: https://docs.docker.com/docker-for-windows/install/



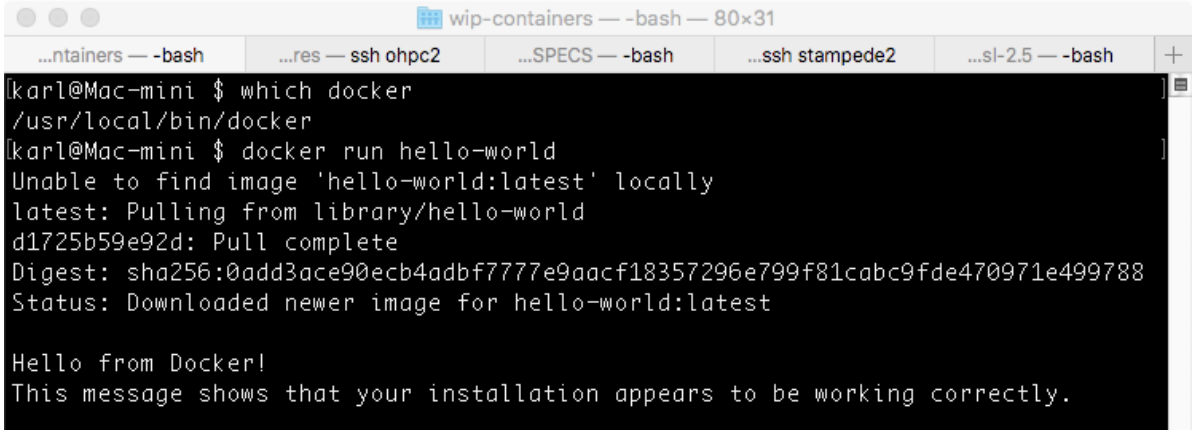

https://docs.docker.com/get-docker/

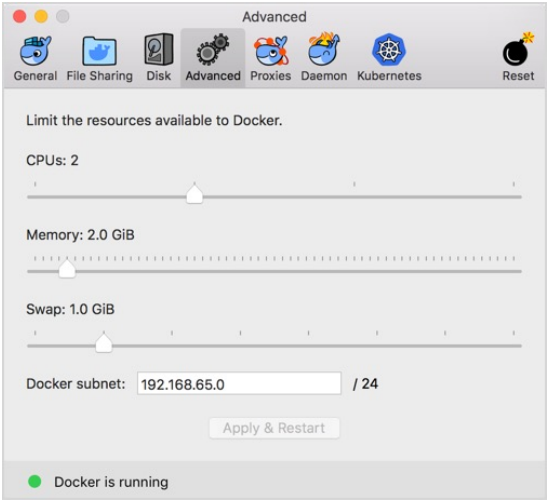
# Docker Terminology

- Image
  - the basis (input) for a Docker container; the image at rest (not running)
- Container
  - the image when its running; exercise your app in the running container
- Engine
  - s/w that executes commands for containers
  - manages networking and file system volumes
  - host daemon + command line interface
- Registry
  - stores, distributes, and manages Docker images
- Dockerfile
  - defines what goes into the environment of a given container
  - a recipe for the container, behaves the same wherever it is run

# Docker Hello World

- Once installed and running, you should have access to controls for setting resources assigned to Docker and a CLI utility called "`docker`" that communications with the daemon




OSX Client Example

# Common options for the Docker client

```
docker <subcommand> [options]
```

| subcommand | description |
|---|---|
| build | create an image from a Dockerfile |
| images | list all images on host |
| create | create a container from an image |
| start | start an stopped container |
| stop | stop an active container |
| exec | run a command in an active container (e.g. 'bash' to start a bash shell) |
| run | run a command in a new container (replaces create, start then exec) |
| ps | list all running and stopped instances |
| rm | remove a container |
| rmi | remove an image |

- Number of these commands must be followed by the ID or name of a container.
- Plan on forgetting the above? "`docker help`" will show available subcommands

# Custom Dockerfile

- Dockerfile is a text-based file that contains all the commands a user needs to automate assembly of a desired image
- Previous "hello-world" example used the Docker daemon to pull down an image from Docker Hub (ie. a shared recipe)
- We can create our own images tailored for our own specific application need(s)
  - this is where containers really shine
  - let's work thru creating a containerized environment akin to the type of environment we see on TACC systems

# Custom Dockerfile

## Step1:

- setup a common Linux OS environment
- Stampede2 is running CentOS7, let's do the same
  - create Dockerfile, request CentOS7
  - use **`docker`** to build the image

Dockerfile

```
# Pull latest image of CentOS from Docker repo
FROM centos:7

# Creator
MAINTAINER Karl W. Schulz <karl@oden.utexas.edu>
```

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              4ab4c602aa5e        3 weeks ago         1.84kB

$ docker build -t step1 .

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
step1               latest              b97391818f48        2 minutes ago       200MB
hello-world         latest              4ab4c602aa5e        3 weeks ago         1.84kB
centos              latest              5182e96772bf        8 weeks ago         200MB
```

# Custom Dockerfile

## Using the image:

- with the image built, we can now spin up a container from the image

```
$ docker run -it step1 /bin/bash
[root@0735060a3074 /]#

[root@0735060a3074 /]# exit
```

Dockerfile

```
# Pull latest image of CentOS from Docker repo
FROM centos:7

# Creator
MAINTAINER Karl W. Schulz <karl@oden.utexas.edu>
```

# Custom Dockerfile

Dockerfile

```
# Pull latest image of CentOS from Docker repo
FROM centos:7

# Creator
MAINTAINER Karl W. Schulz karl@oden.utexas.edu

# Define a user
RUN useradd -m karl

# Add some packages
RUN yum -y install make which git

# Set starting dir and user to run as
USER karl
```

Update #2:

- let's add a user to the container
- also add packages from the distro to provide "which", "make", and "git"
- mount host file system in container (run option)

```
$ docker build -t step2 .
$ docker run -it -v /Users/karl:/home/karl step2 /bin/bash

karl@3eb94e009b59 $ whoami
karl

karl@3eb94e009b59 $ which git
/usr/bin/git
```

# Custom Dockerfile

- Sharing our image: if we have a Docker Hub account, can push images to central hub
  - convenient for running across multiple systems
  - can also package entire application
- General approach:
  - tag a desired image version and register the tag with Docker Hub
  - push the image (you made need to "`docker login`" first)

```
$ docker tag step2 koomietx/step2:v1.0
$ docker push koomietx/step2:v1.0
```

[ koomietx == my Docker Hub userid ]

```
The push refers to repository [docker.io/koomietx/step2]
4f50ff2e459e: Pushed
c12eb6326e2e: Pushed
1d31b5806ba4: Pushed
v1.0: digest:
 sha256:e75a1f38ce134a984df76149ed1db259a6b28cd18693c3883a44ded5bf01f8f7 size: 949
```

# Creating a Development Environment

- With the Dockerfile, we have control over exactly which s/w is instantiated in a running container
- To support local development, it would be nice if we could replicate an environment similar to what we see on HPC systems
- Q: where do we get the s/w to do this?
  - we could build it all ourselves
  - pull from existing repositories
    - we are already pulling from the CentOS repository via the yum package manager
  - leverage builds from other community repositories
    - e.g. OpenHPC

/etc/yum.repos.d/CentOS-Base.repo file from running container

```
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release
=$releasever&arch=$basearch&repo=os&infra=$infra
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-
CentOS-7
```

# Quick overview of OpenHPC

# OpenHPC is a Linux Foundation community project

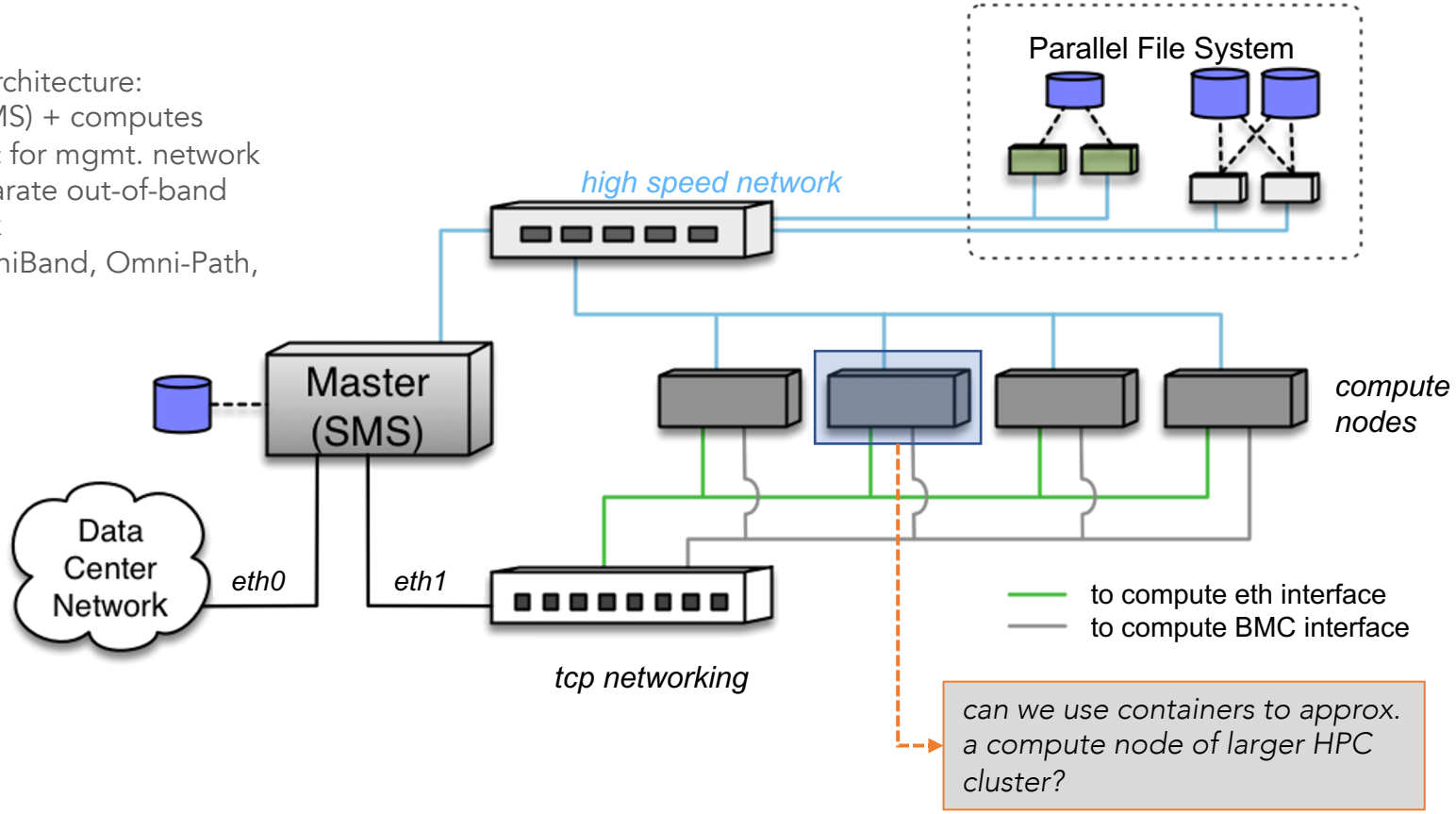
Mission: to provide a reference collection of open-source HPC software components and best practices, lowering barriers to deployment, advancement, and use of modern HPC methods and tools.

Vision: OpenHPC components and best practices will enable and accelerate innovation and discoveries by broadening access to state-of-the-art, open-source HPC methods and tools in a consistent environment, supported by a collaborative, worldwide community of HPC users, developers, researchers, administrators, and vendors.

# OpenHPC: Institutional Members

# Target System Architecture

- Basic cluster architecture: head node (SMS) + computes
- Ethernet fabric for mgmt. network
- Shared or separate out-of-band (BMC) network
- MPI fabric (InfiniBand, Omni-Path, or eth-only)



# OpenHPC: a s/w building block repository

[ Key takeaway ]

- OpenHPC provides a collection of pre-built ingredients common in HPC environments; fundamentally it is a software repository
- The repository is published for use with Linux distro package managers:
  - yum (CentOS/RHEL)
  - zypper (SLES)


- You can pick relevant bits of interest for your site
  - if you prefer a resource manager that is not included, you can build that locally and still leverage the scientific libraries and development environment
  - similarly, you might prefer to utilize a different provisioning system

*We can choose items of interest from the development environment*

# OpenHPC v1.3.9 - S/W components

Released Nov. 12, 2019

| Functional Areas | Components (components available 89 | updates 36%) |
|---|---|
| Base OS | CentOS 7.6, SLES12 SP4 |
| Architecture | aarch64, x86_64 |
| Administrative Tools | Conman, Ganglia, Lmod, LosF, Nagios, NHC, pdsh, pdsh-mod-slurm, prun, EasyBuild, ClusterShell, mrsh, Genders, Shine, Spack, test-suite |
| Provisioning | Warewulf, xCAT |
| Resource Mgmt. | SLURM, Munge, PBS Professional, PMIx |
| Runtimes | Charliecloud, OpenMP, OCR, Singularity |
| I/O Services | Lustre client, BeeGFS client* |
| Numerical/Scientific Libraries | Boost, GSL, FFTW, Hypre, Metis, MFEM, Mumps, OpenBLAS, OpenCoarrays, PETSc, PLASMA, Scalapack, Scotch, SLEPc, SuperLU, SuperLU_Dist, Trilinos |
| I/O Libraries | HDF5 (pHDF5), NetCDF/pNetCDF (including C++ and Fortran interfaces), Adios |
| Compiler Families | GNU (gcc, g++, gfortran), Clang/LLVM, Intel Parallel Studio* |
| MPI Families | MVAPICH2, OpenMPI, MPICH, Intel MPI* |
| Development Tools | Autotools, cmake, hwloc, mpi4py, R, SciPy/NumPy, Valgrind |
| Performance Tools | PAPI, IMB, Likwid, mpiP, pdtoolkit TAU, Scalasca, ScoreP, SIONLib, GeoPM, msr-safe, Dimemas, Extrae, Paraver, OSU Microbenchmarks |

dev items

- 3rd Party libraries are built for each compiler/MPI family
- Resulting repositories currently comprised of ~700 RPMs

Let’s get back to Docker and add some items from OpenHPC

# Custom Dockerfile

Update #3:

- add packages from OpenHPC
  - enable ohpc repo
  - pull in compiler, MPI, modules
  - other dev tools: autotools, valgrind
- we now have similar modules env!

```
FROM centos:7
MAINTAINER Karl W. Schulz karl@oden.utexas.edu
RUN useradd -m karl
# enable OpenHPC repository
RUN yum install -y
http://build.openhpc.community/OpenHPC:/1.3/CentOS_7
/x86_64/ohpc-release-1.3-1.el7.x86_64.rpm

# Add some packages (from distro & OpenHPC)
RUN yum -y install make which git
RUN yum -y install ohpc-autotools
RUN yum -y install valgrind-ohpc
RUN yum -y install gnu8-compilers-ohpc
RUN yum -y install gsl-gnu8-ohpc
RUN yum -y install openmpi3-gnu8-ohpc
RUN yum -y install lmod-defaults-gnu8-openmpi3-ohpc
RUN yum -y install examples-ohpc

RUN mkdir /home1
WORKDIR /home/karl
```

```
$ docker build -t step3 .
$ docker run -it step3 /bin/bash -l

[karl@ 614bc293286f ~]$ module avail

---------------------- /opt/ohpc/pub/moduledeps/gnu7 ----------------------
   gsl/2.5    openmpi3/3.1.4 (L)

------------------------ /opt/ohpc/pub/modulefiles ------------------------
   autotools (L)    gnu8/8.3.0 (L)    ohpc (L)    prun/1.3 (L)    valgrind/3.15.0
```

# Custom Dockerfile

## Update #3:

- now we can compile and run MPI binaries locally

```
FROM centos:7
MAINTAINER Karl W. Schulz karl@oden.utexas.edu
RUN useradd -m karl
# enable OpenHPC repository
RUN yum install -y
http://build.openhpc.community/OpenHPC:/1.3/CentOS_7
/x86_64/ohpc-release-1.3-1.el7.x86_64.rpm

# Add some packages (from distro & OpenHPC)
RUN yum -y install make which git
RUN yum -y install ohpc-autotools
RUN yum -y install valgrind-ohpc
RUN yum -y install gnu7-compilers-ohpc
RUN yum -y install gsl-gnu7-ohpc
RUN yum -y install openmpi3-gnu7-ohpc
RUN yum -y install lmod-defaults-gnu7-openmpi3-ohpc
RUN yum -y install examples-ohpc
RUN mkdir /home1
WORKDIR /home/karl
```

```
$ docker run -it -v /Users/karl:/home/karl step3 /bin/bash -l

[karl@614bc293286f ~]$ mpicc /opt/ohpc/pub/examples/mpi/hello.c
[karl@614bc293286f ~]$ mpirun -np 2 ./a.out

 Hello, world (2 procs total)
    --> Process #   0 of   2 is alive. -> 614bc293286f
    --> Process #   1 of   2 is alive. -> 614bc293286f
```

*push step3 image to Registry*
*(i did this already, step3:v1.1)*

# Containers

- Docker is a great way for us to create custom development environments locally
- Very popular in cloud computing, IT deployment space (millions of dockerized apps)
- However, we cannot generally run Docker containers on HPC systems
  - the reason? security…
  - requires elevated credentials that we don't have
  - MPI interconnects can also be problematic
- Other container technologies have emerged that are targeted for HPC systems
  - Shifter (NERSC)
  - Singularity (LBNL)
  - Charliecloud (LANL)
- Given the wild popularity of Docker, these all have some mechanism to leverage existing Docker images
  - TACC supports singularity, so let's look at using our existing Docker image on Stampede2

# Singularity from Docker

Singularity images can be built from existing Docker images (and pulled from Docker Hub)

- setup singularity env
- run interactive SLURM job
- build a singularity image (step3 from before)
- this will create an image file

Example on Stampede2

```
$ module load tacc-singularity
$ singularity build step3 docker://koomietx/step3:v1.1

$ ls -lh step3
-rwxr-xr-x 1 karl G-800747 239M Oct 15 10:23 step3
```

# Singularity from Docker

## Singularity: running an image

- singularity defaults to propagating the host environment to the running container
- we can disable this so we have pristine login environment with our modules
- singularity on TACC also setup to mount your $HOME file system (and $WORK/$SCRATCH). Need the mount point for $HOME to exist

Example on Stampede2

Dockerfile snippet

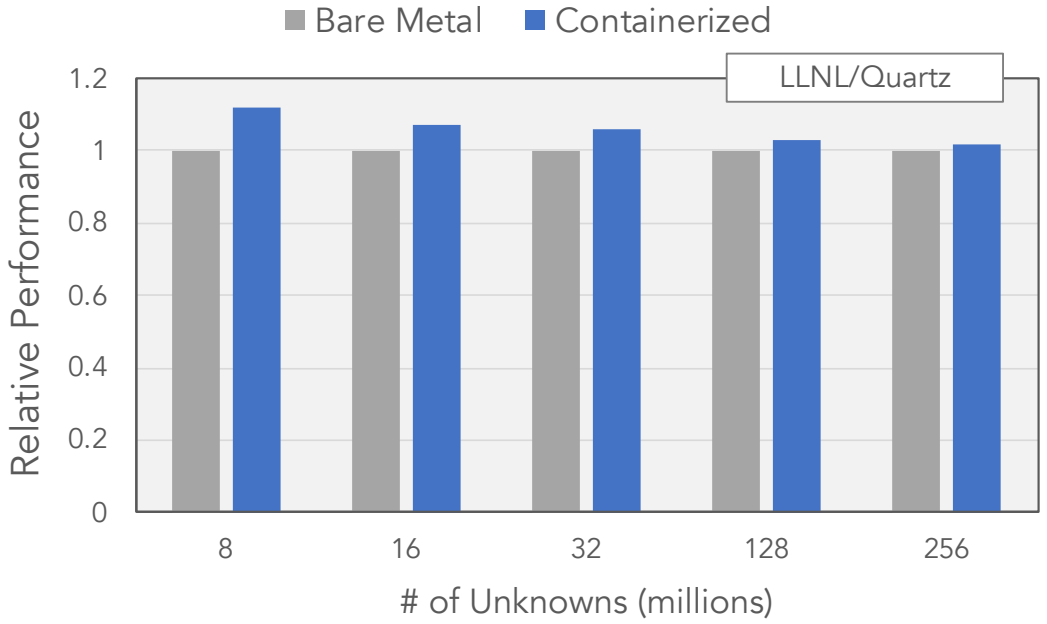
```
...
RUN mkdir /home1
...
```

```
$ singularity exec --cleanenv  step3 /bin/bash --login
WARNING: Non existent bind point (directory) in container: '/scratch'
WARNING: Non existent bind point (directory) in container: '/work'
Singularity> module list

Currently Loaded Modules:
  1) autotools   2) prun/1.2   3) gnu7/7.3.0   4) openmpi3/3.1.0   5) ohpc
```

# Singularity Performance against Bare Metal

- Example C++ finite-element benchmarks using the MFEM library
- Containerized execution compared against native bare-metal



# References

- VMs/Containers
  - Docker get started tutorial: https://docs.docker.com/get-started/
  - Docker Installing on Windows: https://medium.com/@tushar0618/installing-docker-toolbox-on-windows-1f04c794e2ae
  - https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b
  - Singularity Docs: https://www.sylabs.io/docs/

- OpenHPC - http://www.openhpc.community/
  - GitHub: https://github.com/openhpc/ohpc
  - Mailing Lists: http://www.openhpc.community/support/mail-lists/

- Acknowledgements:
  - thanks to Clark Pennie for creating/testing Dockerfiles on Windows