An Introduction to Singularity:
A Container Engine for High-Performance Computing

Marty Kandes, Ph.D.

HPC User Services Group
San Diego Supercomputer Center
University of California, San Diego

XSEDE Webinar
Thursday, June 14th, 2017
9:00AM - 11:00AM PDT

A Quick Outline of Today's Webinar

- **Comet**: An Overview of SDSC's Newest Supercomputer
- **Containers**: What are they? And what are they good for?
- **Singularity**: How to Build and Run HPC Containers
- **Q&A**

About Me

- **I am not a container expert.**
- I am an HPC user who got into HPC infrastructure.
- Joined HPC User Services Group @ SDSC in April 2017
- Previously worked for Distributed High-Throughput Computing Group @ SDSC and the Open Science Grid

A not-so long time ago in a data center not that far, far away ...

- In 2012, 99% of all computational jobs run on NSF-funded HPC resources utilized fewer than 2048 CPU-cores, while accounting for approximately 50% of the total core-hours consumed across these resources.
- Nearly 70% of all jobs actually ran on only a single compute node (16 CPU-cores) or less.

Comet: A Supercomputer Built to Serve the 99%

Design Goals for Comet

- **Flexability** - Ability to run a wide range of scientific and engineering applications to support complex, dynamic and multidisciplinary computational workflows.
- **Scalability** - Ability to support a large, diverse community of modestly-sized research projects that in aggregate represent a significant amount of research activity.

Comet By the Numbers

- **1944 compute nodes**: Dual-socket; 2.5 GHz Intel Xeon E5-2680v3 processors; 12 cores per processor; 128 GB DDR4 DRAM; 120 GB/s memory bandwidth; 320 GB SSD (210 GB Avail)
- **4 large-shared memory nodes**: Quad-socket; 2.2 GHz Intel Xeon E7-8860v3 processors; 16 cores per processor; 1.5 TB DDR4 DRAM; 400 GB SSD (260 GB Avail)
- **36 k80 gpu nodes**: Same as standard *compute* node, but with 2 PCIe-based NVIDIA Tesla K80 dual-GPU accelerators per node
- **36 p100 gpu nodes**: Dual-socket; 2.4 GHz Intel Xeon E5-2680v4 processors; 14 cores per processor; 128 GB DDR4 DRAM; 150 GB/s memory bandwidth; 400 GB SSD (260 GB Avail); 4 PCIe-based NVIDIA Tesla P100 GPU accelerators per node

2.76 Pflop/s

Comet By the Numbers

- **Interconnect**: Mellanox FDR (56Gbps) InfiniBand; hybrid fat-tree topology; rack-level (72 node) full bisection bandwidth; 4:1 oversubscription cross-rack bandwidth
- **Storage**: NSF-based $HOME storage (100 GB per user; weekly backups); 6.4 PB 200 GB/s Lustre-based parallel filesystem storage (intermediate-term use: at least 500 GB per group allocation in /oasis/projects; short-term use: up to 10 TB per user in /oasis/scratch; *2M inodes limit*; NO BACKUP!)
- **Applications**: More than 173 software applications and libraries maintained and deployed via Rocks (Linux) cluster distribution; accessible to users via software modules; span a wide range of scientific diciplines, including, but not limited to, bioinformatics, chemistry, data analytics, engineering, fluid dynamics, mathematics, molecular dynamics, neuroscience, and statistics
- **Scientific Impact**: 1755 PIs; 358 institutions; 1144 research allocations; 4709 direct-access users; 33000+ gateway users; 997 publications

How do I get time on Comet?

1. Go to `https://portal.xsede.org`
2. Sign up for an XSEDE User Portal (XUP) account
3. If your are an eligible Principal Investigator (PI), apply for an XSEDE allocation (on Comet)
4. Once awarded an allocation, PIs can grant access to the allocation to other users who have an XUP account

Trial Allocations on Comet

- Designed to give new, prospective users rapid, but limited access to Comet and its resources
- Simply send us a request via XSEDE ticketing system
- Limited to 1,000 service units (SUs) (core-hours) on Comet's standard compute nodes and 100 SUs (GPU-hours*) on its GPU-accelerated compute nodes
- Requests reviewed and awarded witin one business day; allocations expire after 6 months
- May not be renewed or extended

`https://portal.xsede.org/allocations/startup#trial`

* Comet's P100 GPUs are more than twice as fast as its K80 GPUs for many applications. As such, users are charged a premium of 1.5 SUs per GPU-hour when running on the P100s.

Startup Allocations on Comet

- Designed for users to perform limited application development, **benchmarking**, and evaluation of XSEDE's computational resources for future research use
- Require minimal documentation: an abstract and PI's CV
- May request time on multiple XSEDE resources
- Limited to 50,000 SUs (core-hours) on Comet's standard compute nodes and 2,500 SUs (GPU-hours) on its GPU-accelerated compute nodes
- Requests reviewed and awared within 2 weeks; allocations expire after 1 year*, but may be extended up to 18 months

`https://portal.xsede.org/allocations/startup`

* See new Multi-Year Startup Allocations

Research Allocations on Comet

- Designed for users to perform full-scale, production-level research computing
- Awarded based on peer-reviewed proposal
- May request time on multiple XSEDE resources
- Limited to 10M SUs (core-hours) on Comet's standard compute nodes and 50,000 SUs (GPU-hours) on Comet's GPU-accelerated compute nodes
- Proposals are accepted and review quarterly; allocations expire after 1 year, but may be extended up to 18 months

`https://portal.xsede.org/allocations/research`

Who is eligible to apply for an XSEDE allocation?

- Principal Investigators (PIs) who apply for an XSEDE resource allocation are usually faculty members or researchers, including postdoctoral researchers, at U.S.-based academic or research institutions
- Eligible institutions include federal research labs, state and local governments, both commercial and non-profit organizations; note, however, special rules may apply if your institution is not a university or a two- or four-year college
- Undergraduate and graduate students* may not be a PI

`https://portal.xsede.org/allocations/policies`

* NSF Graduate Student Fellows and Honorable Mention recipients may serve as a PI

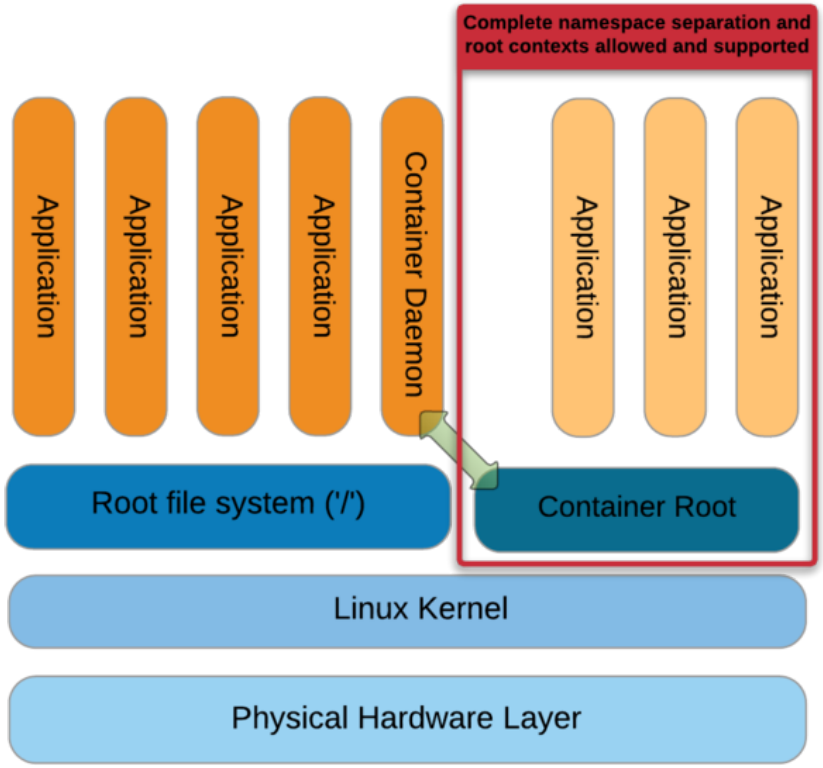
Any Questions?

Containers

What is a Container?

- At rest, a container or **container image** is simply a file (or collection of files) saved on disk that stores everything you need to run a target application or applications: code, runtime, system tools, libraries and settings, etc.

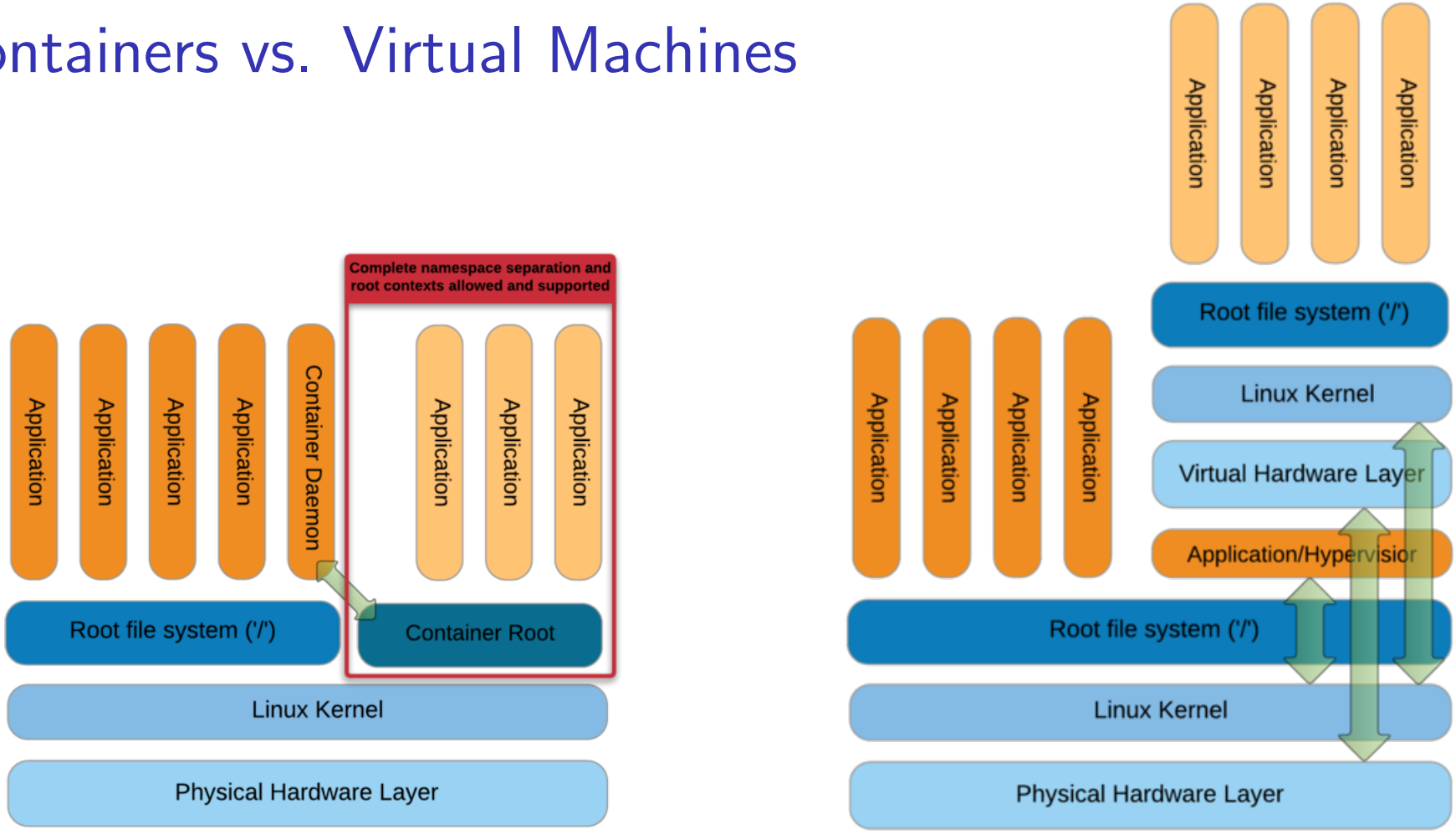
- *In motion*, a container or **container process** is simply a standard (Linux) process running on top of the underlying host's operating system and kernel, but whose software environment is defined by the contents of the container image.



Container : Supercomputer :: Construct : Matrix

“ ... it’s our loading program. We can load anything ... anything we need.”

Containers vs. Virtual Machines



Container-based applications have **direct access** to the host kernel and hardware, *just like native applications*. In contrast, VM-based applications only have **indirect access** via the guest OS and hypervisor, which creates a significant performance overhead.

Advantages of Containers

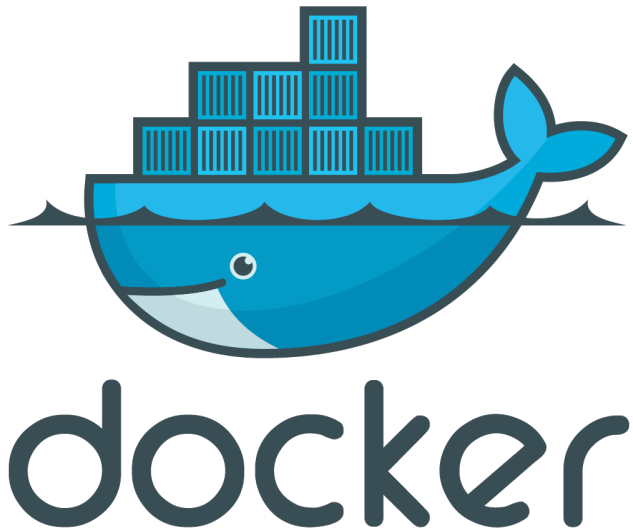
- **Performance**: Near-native application performance
- **Freedom**: Bring your own software environment
- **Reproducibility**: Package complex software applications into easy to manage, verifiable software units
- **Compatibility**: Built on open standards available in all major Linux distributions
- **Portability**: Build once, run (almost) anywhere

Limitations of Containers

- **Architecture-dependent**: Always limited by CPU architecture (x86_64, ARM) and binary format (ELF)
- **Portability**: Requires glibc and kernel compatibility between host and container; also requires any other kernel-user space API compatibility (e.g., OFED/IB, NVIDIA/GPUs)
- **Filesystem isolation**: filesystem paths are (mostly) different when viewed inside and outside container

Docker

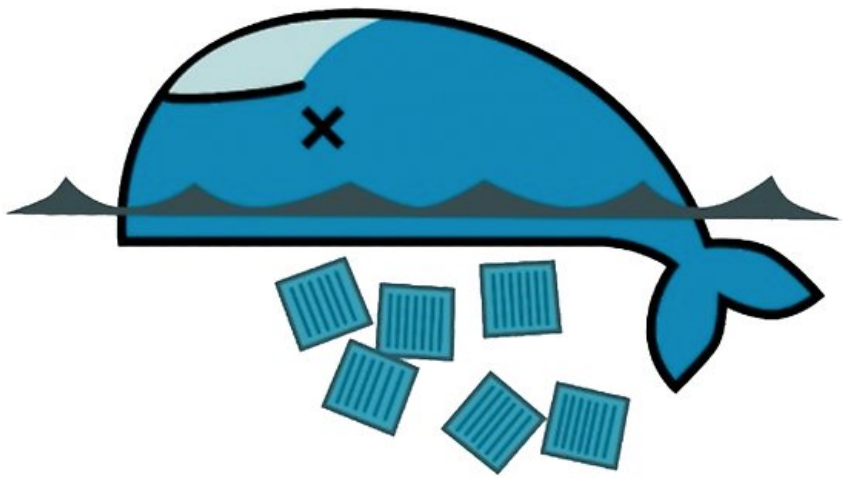
- Most common **container engine** in use today
- Provides tools and utilities to create, maintain, distribute, and run containers images
- Desgined to accomodate network-centric services (web servers, databases, etc)
- Easy to install, well-documentated, and large, well-developed user community and container ecosystem (DockerHub)



`https://www.docker.com/`

Docker on HPC Systems

- HPC systems are shared resources, mostly batch-based jobs
- Docker's security model is desgined to support trusted users running trusted containers; e.g., users can esclate to root
- Docker not designed to support batch-based workflows
- Docker not designed to support tightly-coupled, highly distributed parallel applications (MPI).



Singularity: A Container Engine for HPC

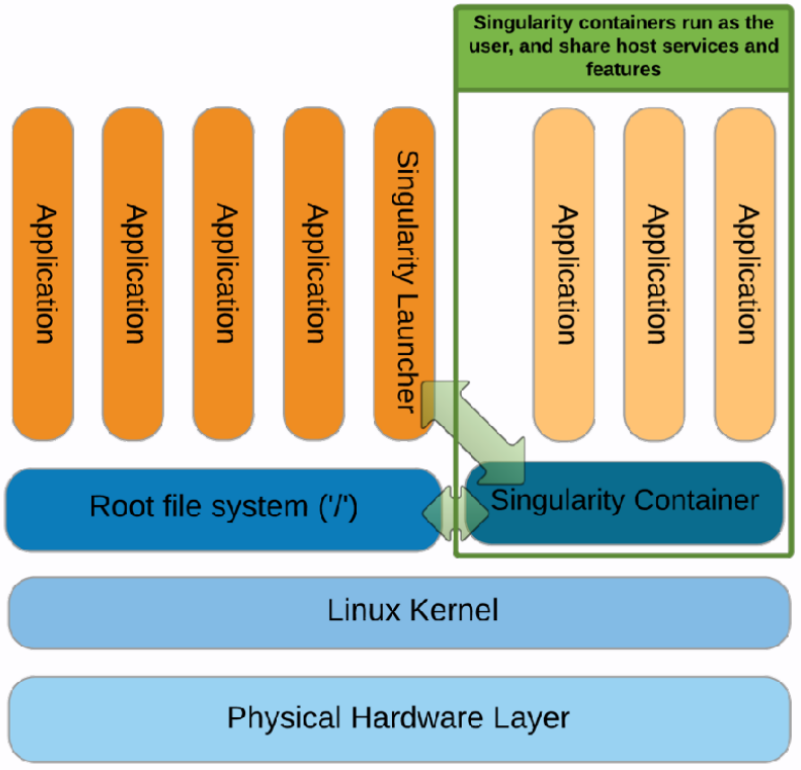
http://singularity.lbl.gov/

Singularity Design Goals

- Reproducible, portable, sharable, and distributable containers
- No trust security model: untrusted users running untrusted containers
- Support HPC hardware and scientific applications

Singularity Features

- Each container is a single image file
- No root owned daemon processes
- No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
- Supports shared/multi-tenant resource environments
- Supports HPC hardware: Infiniband, GPUs
- Supports HPC applications: MPI



Common Singularity Use Cases

- Building and running applications that require newer system libraries than are available on host system
- Running commercial applications binaries that have specific OS requirements not met by host system
- Converting Docker containers to Singularity containers

Any Questions?

The Singularity Workflow

1. Build your Singularity containers on a local system where you have have root or sudo access; e.g., a personal desktop or laptop computer where you have installed Singularity

2. Transfer your Singularity containers to the HPC system where you want to run them

3. Run your Singualrity containers on that HPC system

Running Singularity on Mac OS X or Windows

1. Install VirtualBox on your personal desktop or laptop computer: `https://www.virtualbox.org/`
2. Create either an Ubuntu- or CentOS-based VirtualBox virtual machine, where you will build and test your Singularity containers
3. Install Singularity* on that virtual machine: `http://singularity.lbl.gov/install-linux`

* Recommendation: Install the same version of Singularity used on the HPC system where you plan to run your containers; if you plan to run on multiple HPC systems, then install the lowest version number you expect to use

How to *build* a Singularity container ...

... from a Singularity definition (or recipe) file:

```
sudo singularity build ubuntu.simg ubuntu.def
```

... from a pre-exisiting Docker container on Docker Hub:

```
sudo singularity build ubuntu.simg docker://ubuntu
```

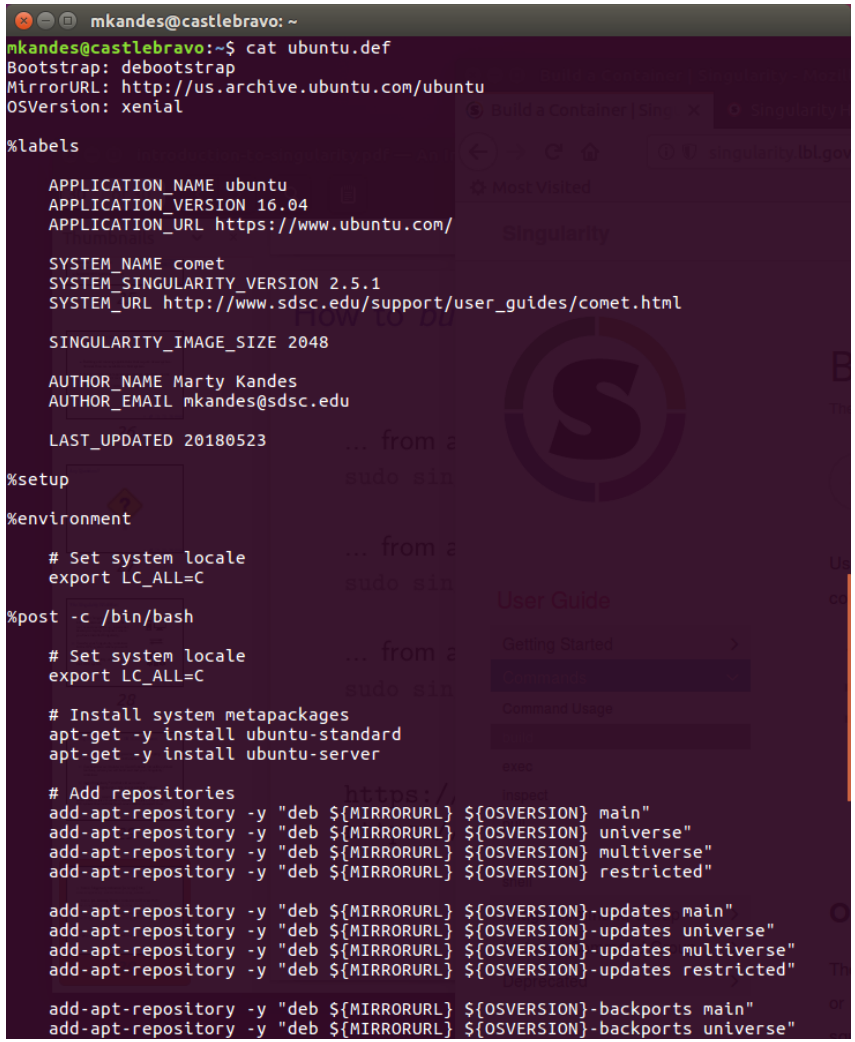
... from a pre-exisiting Singularity container on Singularity Hub:

```
sudo singularity build ubuntu.simg shub://singularityhub/ubuntu
```

`https://hub.docker.com/`
`https://www.singularity-hub.org/`

Singularity Recipe File

- A Singularity definition (or recipe) file is the starting point for designing any custom container.
- It includes specifics about installation of software, environment variables, files to add, and container metadata.
- You can even write a help section, or define modular components in the container.



naked-singularity

- A repository of definition (or recipe) files for building Singularity containers around the software applications, frameworks, and libraries you need to run on high-performance computing systems.
- `https://github.com/mkandes/naked-singularity`

Singularity Recipe File: Header

```
mkandes@castlebravo: ~
mkandes@castlebravo:~$ ls
Desktop  Downloads  Dropbox  Software  ubuntu.def
mkandes@castlebravo:~$ cat ubuntu.def
Bootstrap: debootstrap
MirrorURL: http://us.archive.ubuntu.com/ubuntu
OSVersion: xenial

%labels

    APPLICATION_NAME ubuntu
    APPLICATION_VERSION 16.04
    APPLICATION_URL https://www.ubuntu.com/
```

The header is at the top of the file, and tells Singularity the base operating system that it should use to build the container. `Bootstrap:` references the kind of base you want to use (e.g., docker, debootstrap, shub). Depending on the value assigned to `Bootstrap:`, other keywords may also be valid in the header.

Singularity Recipe File: %labels

```
mkandes@castlebravo: ~
Bootstrap: debootstrap
MirrorURL: http://us.archive.ubuntu.com/ubuntu
OSVersion: xenial

%labels

    APPLICATION_NAME ubuntu
    APPLICATION_VERSION 16.04
    APPLICATION_URL https://www.ubuntu.com/

    SYSTEM_NAME comet
    SYSTEM_SINGULARITY_VERSION 2.5.1
    SYSTEM_URL http://www.sdsc.edu/support/user_guides/comet.html

    SINGULARITY_IMAGE_SIZE 2048

    AUTHOR_NAME Marty Kandes
    AUTHOR_EMAIL mkandes@sdsc.edu

    LAST_UPDATED 20180523

%setup

                                                          1,1           Top
```

You can add metadata labels to your container in the `%labels` section of the recipe file. They will be stored in the file `/.singularity.d/labels.json` within the container and may be viewed with the `singularity inspect` command. The general format is a `LABELNAME` followed by a `LABELVALUE`.

Singularity Recipe File: %environment

```
mkandes@castlebravo: ~
    SINGULARITY_IMAGE_SIZE 2048

    AUTHOR_NAME Marty Kandes
    AUTHOR_EMAIL mkandes@sdsc.edu

    LAST_UPDATED 20180523

%setup

%environment

    # Set system locale
    export LC_ALL=C

%post -c /bin/bash

    # Set system locale
    export LC_ALL=C

    # Install system metapackages
    apt-get -y install ubuntu-standard
    apt-get -y install ubuntu-server
                                                    37,0-1        17%
```

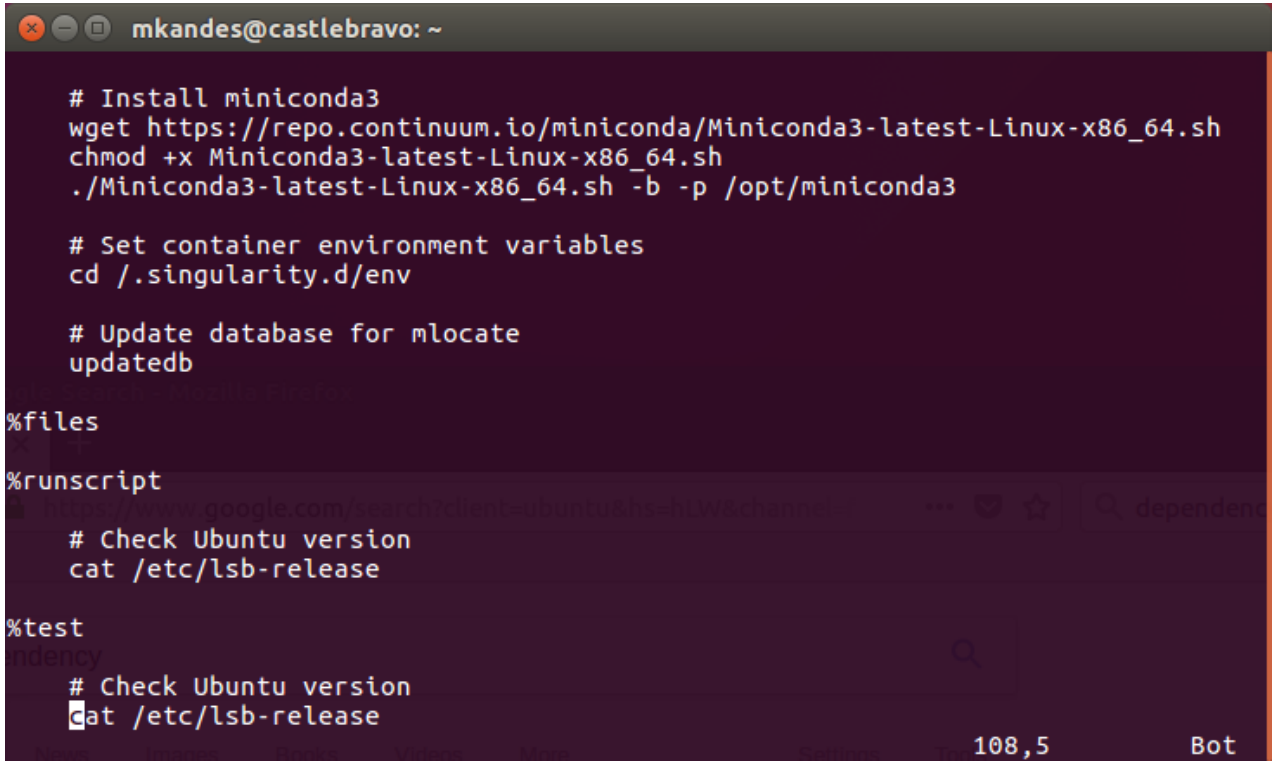
You can add environment variables to your container in the `%environment` section of the recipe file. Note, these environment variables are sourced at runtime and *not* at build time. Therefore, if you need the same variables during build time, you should also define them in the `%post` section.

Singularity Recipe File: %post

```
mkandes@castlebravo: ~
    SINGULARITY_IMAGE_SIZE 2048

    AUTHOR_NAME Marty Kandes
    AUTHOR_EMAIL mkandes@sdsc.edu

    LAST_UPDATED 20180523

%setup

%environment

    # Set system locale
    export LC_ALL=C

%post -c /bin/bash

    # Set system locale
    export LC_ALL=C

    # Install system metapackages
    apt-get -y install ubuntu-standard
    apt-get -y install ubuntu-server
                                                          37,0-1        17%
```

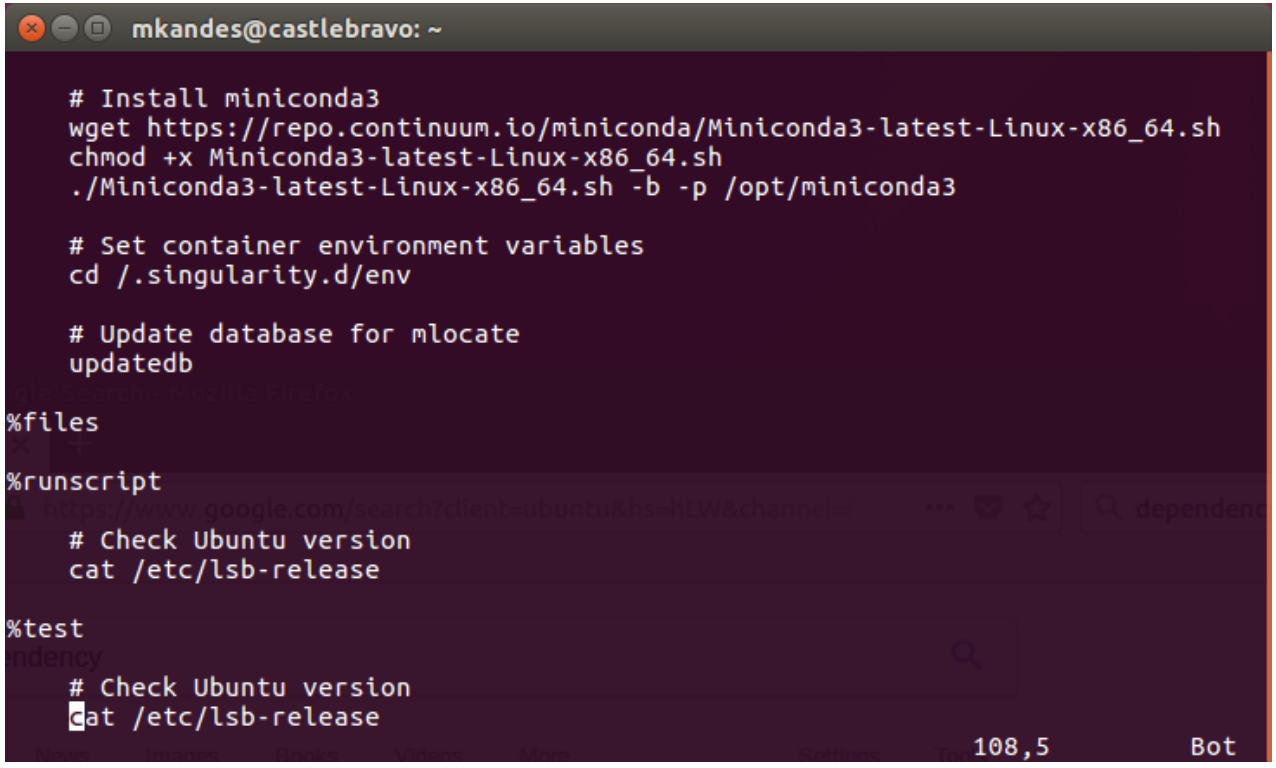
All commands in the %post section of the recipe file are executed after the base OS of the container has been installed. This is where you will customize your container, including making directories, installing software and libraries, etc.

Singularity Recipe File: %runscript



The `%runscript` section of the recipe file is a scriptlet that is excuted when the container is *run* via the `singularity run` command. Used for the most commonly issued call to the container application.

Singularity Recipe File: %test



Like `%runscript`, the `%test` section of the recipe file is a scriptlet, but one which is excuted at the end of the container *build* process. Used to validate container.

How to create an inteactive *shell* within a Singularity Container ...

Singularity Container Image Formats

There are 3 different Singularity container image formats:

- **squashfs**
- **ext3**

How to *build* a **writable** Singularity container ...

... from a Singularity definition (or recipe) file:

```
singularity image.create --size 2048 ubuntu.img
sudo singularity build --writable ubuntu.img ubuntu.def
```

... from a pre-exisiting Docker container on Docker Hub:

```
sudo singularity build ubuntu.simg docker://ubuntu
sudo singularity build --writable ubuntu.img
```

... from a pre-exisiting Singularity container on Singularity Hub:

```
sudo singularity build ubuntu.simg shub://singularityhub/ubuntu
```

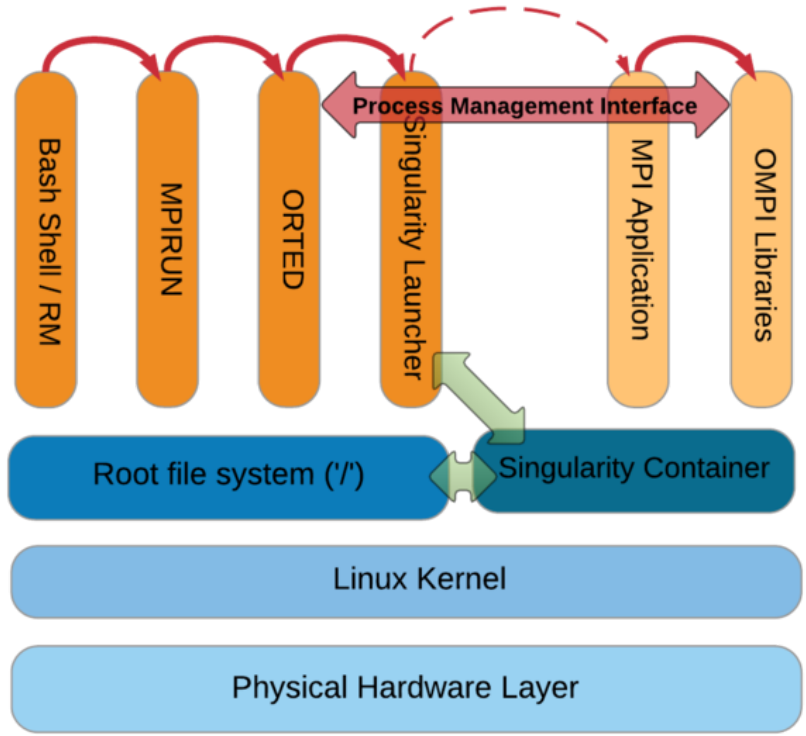
`https://hub.docker.com/`
`https://www.singularity-hub.org/`

Don't *build*, PULL!

- If you do not need to make any custom modifications to an existing Docker or Singularity Hub container, you can directly download the Singularity container to Comet using the `singularity pull` command.
- `singularity pull ubuntu.simg docker://ubuntu`

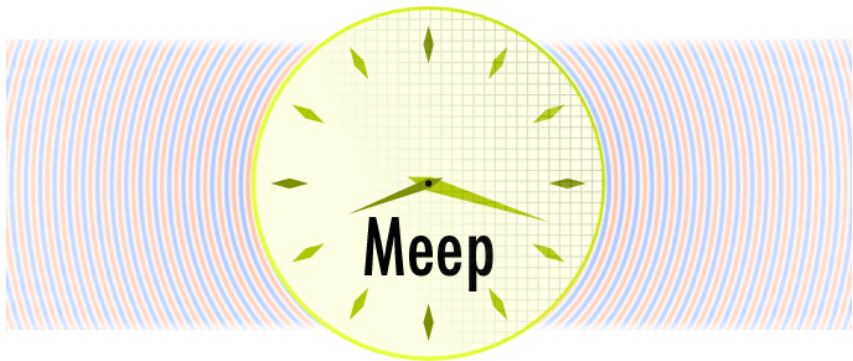
MPI-based Singularity Containers

- Use same Message Passing Interface (MPI) distribution and version within container as would be used outside the container.
- If using Infiniband, install same OFED drivers and libraries inside the container as used on underlying HPC hardware.



MPI-based Singularity Containers: MEEP

- MEEP: MIT Electromagnetic Equation Propagation is a free and open-source software package for simulating electromagnetic systems via the finite-difference time-domain (FDTD) method.
- Dependency hell: Too difficult to compile in Comet's native environment.



GPU-accelerated Singularity Containers: TensorFlow

Exoskeletal (or Dependency-Only) Singularity Containers: Julia

Singularity: A Summary

1. You can now install (almost) any software you like on your favorite HPC system without having to make a special request to the system's administrators or user support staff.
2. In many cases, this software is now portable between different HPC systems.
3. And this

Questions?