An Introduction to Singularity:
Containers for Scientific and High-Performance Computing

Marty Kandes, Ph.D.

High-Performance Computing User Services Group
San Diego Supercomputer Center
University of California, San Diego

SDSC Summer Institute 2020
Friday, August 7th, 2020
8:30AM - 9:00PM PDT

Today's Session

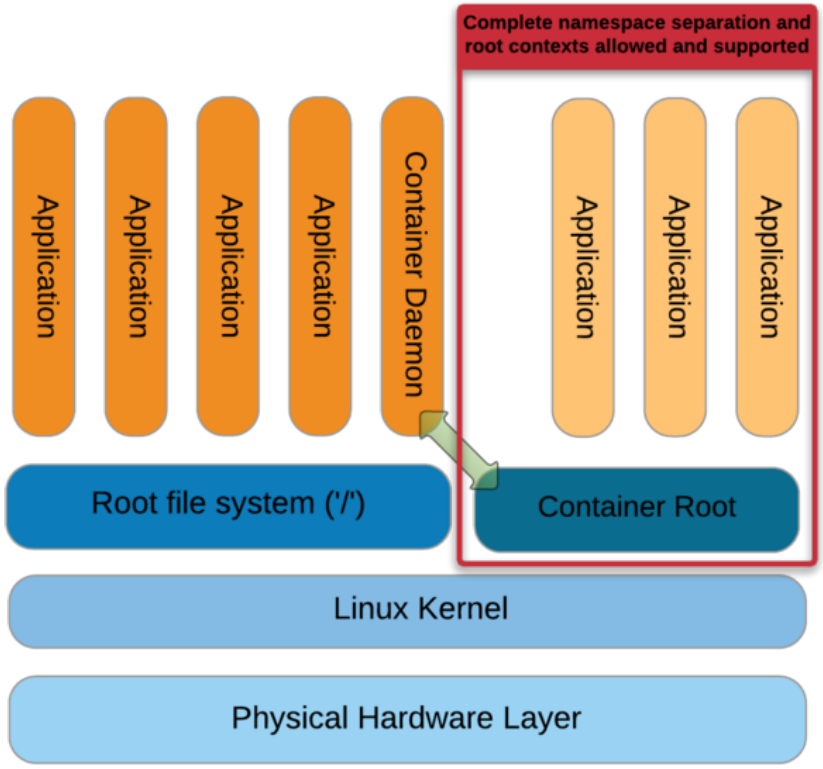
- What are containers? And why you should use them.
- What is Singularity and what does it have to do with containers?
- How to build and run Singularity containers.
- Q&A?

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.

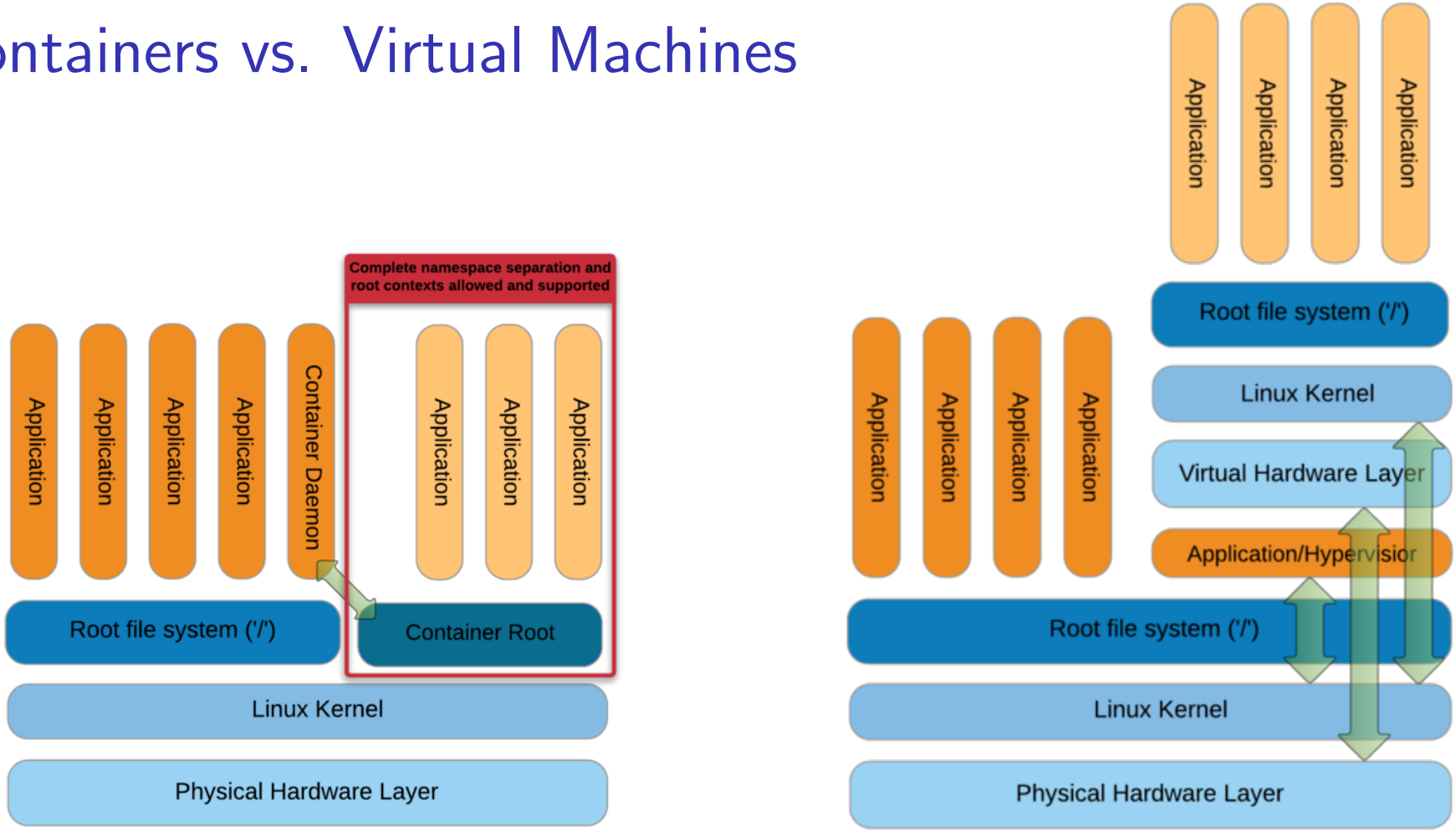


Container : Supercomputer :: Construct : Matrix

“ ... 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, *similar to 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

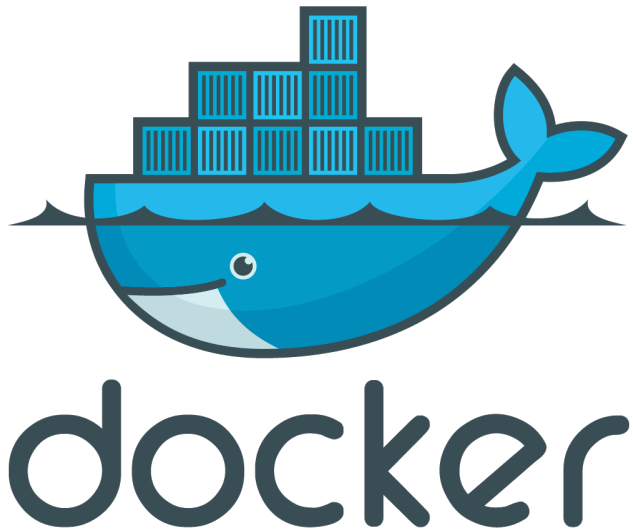
- **Performance**: Near-native application performance
- **Freedom**: Bring your own software environment
- **Reproducibility**: Package complex software applications into easy to manage, verifiable software units
- **Compatibility**: Built on open standards available in all major Linux distributions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

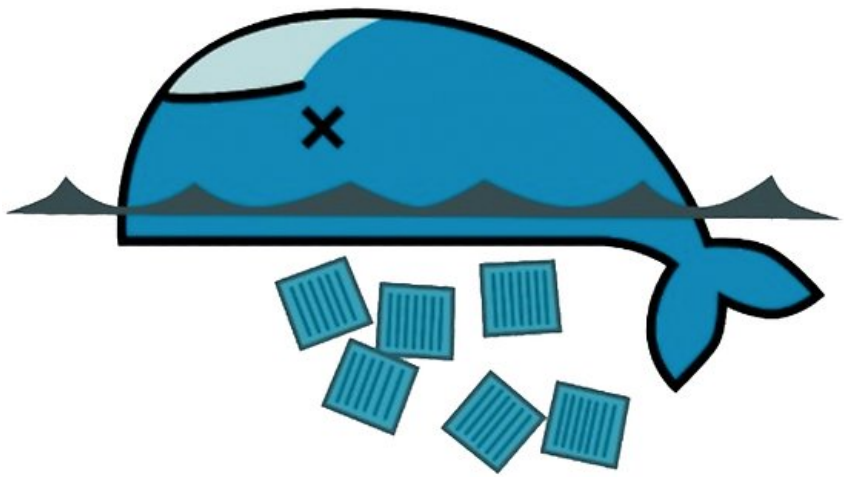
- Most common **container engine** in use today
- Provides tools and utilities to create, maintain, distribute, and run containers images
- Designed to accommodate network-centric services (web servers, databases, etc)
- Easy to install, well-documentated, and large, well-developed user community and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



`https://www.docker.com`

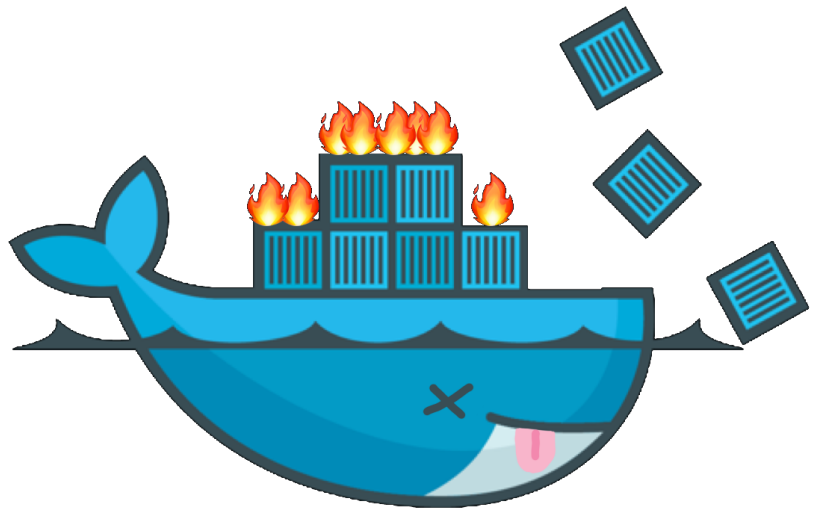
Docker on HPC Systems

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



Docker on the Cloud (CVE-2019-5736)

- "The Open Containers Initiative (OCI) recently discovered a new security vulnerability CVE-2019-5736 in runc, allowing container escape to obtain root privileges on the host node."
- " Amazon employee here: we have released a security bulletin covering how to update to the latest patched Docker on Amazon Linux, Amazon ECS, Amazon EKS, AWS Fargate, AWS IoT Greengrass, AWS Batch, AWS Elastic Beanstalk, AWS Cloud9, AWS SageMaker, AWS RoboMaker, and AWS Deep Learning AMI."



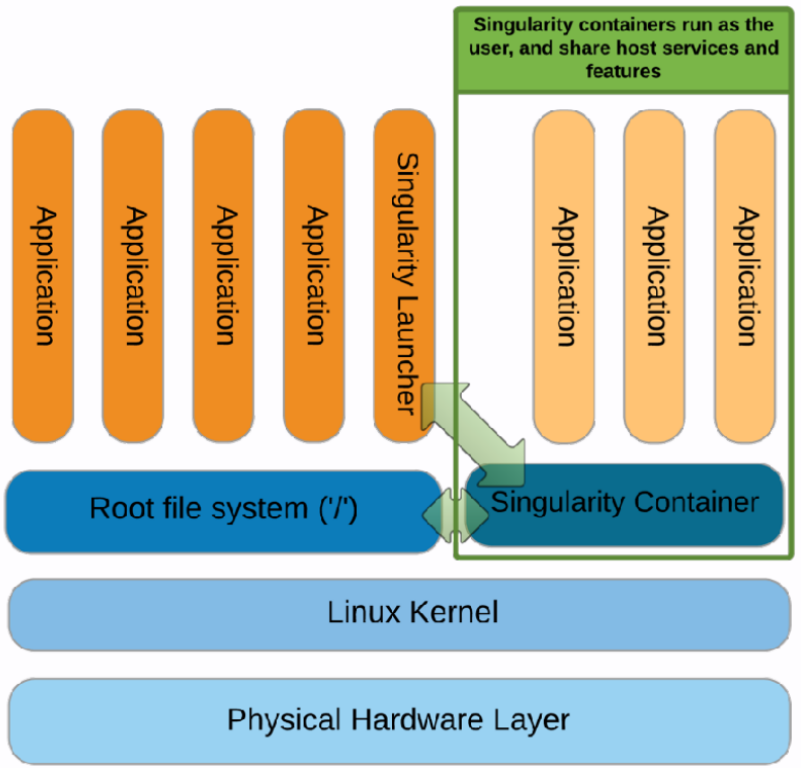
Singularity: A Container Engine for HPC

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

`http://singularity.lbl.gov`
`https://www.sylabs.io`

Features of Singularity

- 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

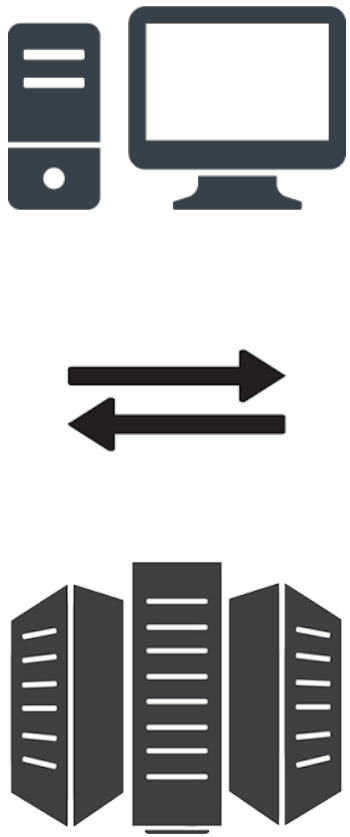
- Building and running applications that require newer system libraries than are available on host system
- Running commercial applications binaries that have specific OS requirements not met by host system
- Converting Docker containers to Singularity containers

The Singularity Workflow

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

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

3. **Run** your Singularity containers on that HPC system



Running Singularity on Mac OS X** or Windows

1. Install VirtualBox on your personal computer:
   `https://www.virtualbox.org`

2. Create either an Ubuntu or Fedora-based virtual machine, where you will build and test your Singularity containers

3. Install Singularity* on that virtual machine:
   `https://sylabs.io/guides/3.6/admin-guide/installation.html`

* 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.

** Beta Release: `https://www.sylabs.io/singularity-desktop-macos`

Essential Singularity

The main Singularity command

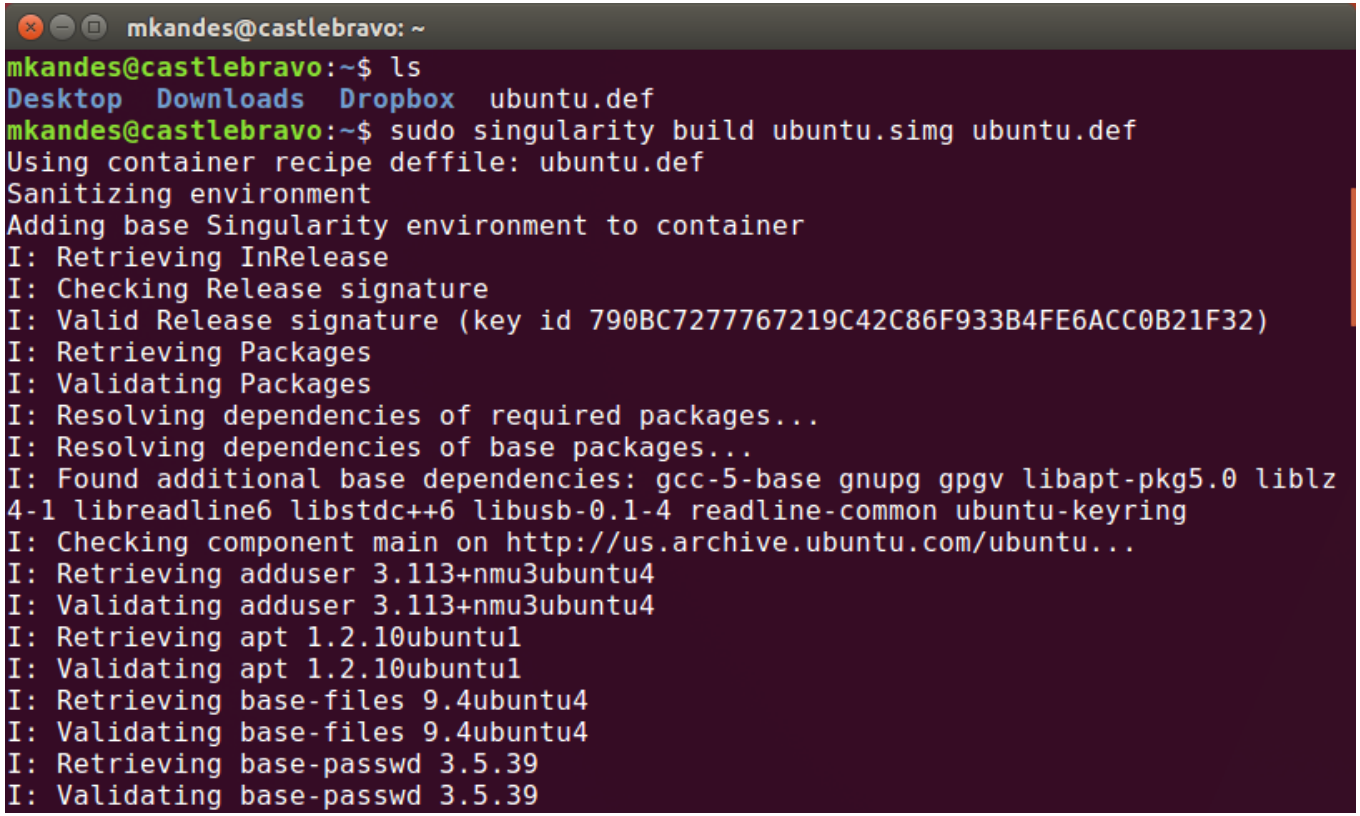
```
singularity [options] <subcommand> [subcommand options] ...
```

has three essential subcommands:

- `build`: Build your own container from scratch using a Singularity definition (or recipe) file; download and assemble any existing Singularity container; or convert your containers from one format to another (e.g., from Docker to Singularity)
- `shell`: Spawn an interactive shell session in your container.
- `exec`: Execute an arbitrary command within your container.

How to *build* a Singularity container ...

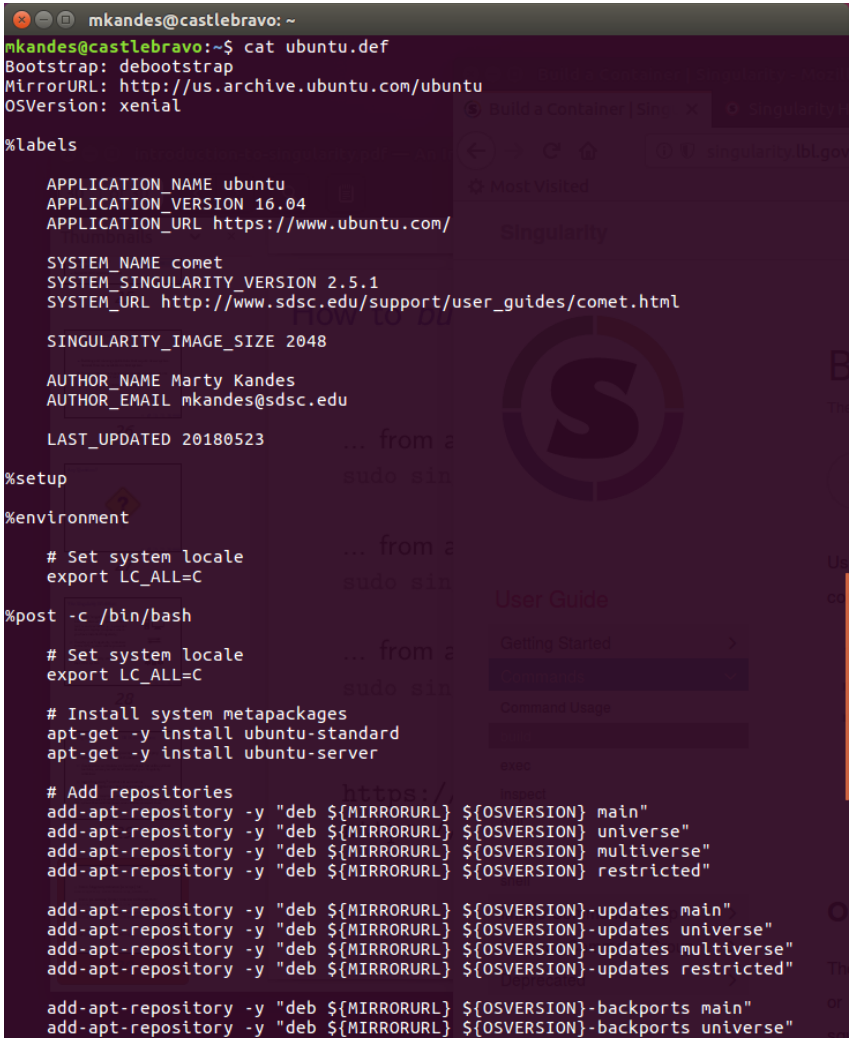
... from a Singularity definition (or recipe) file:




```
sudo singularity build ubuntu.sif ubuntu.def
```

Singularity Recipe File

- A Singularity definition (or recipe) file is the starting point for designing any custom container.
- It is a manifest of all software to be installed within the container, environment variables to be set, files to be added, directories to be mounted, container metadata, etc.
- 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.

- Aim of the project is to:

  1. Version control the Singularity containers we're building, maintaining, and deploying for you;

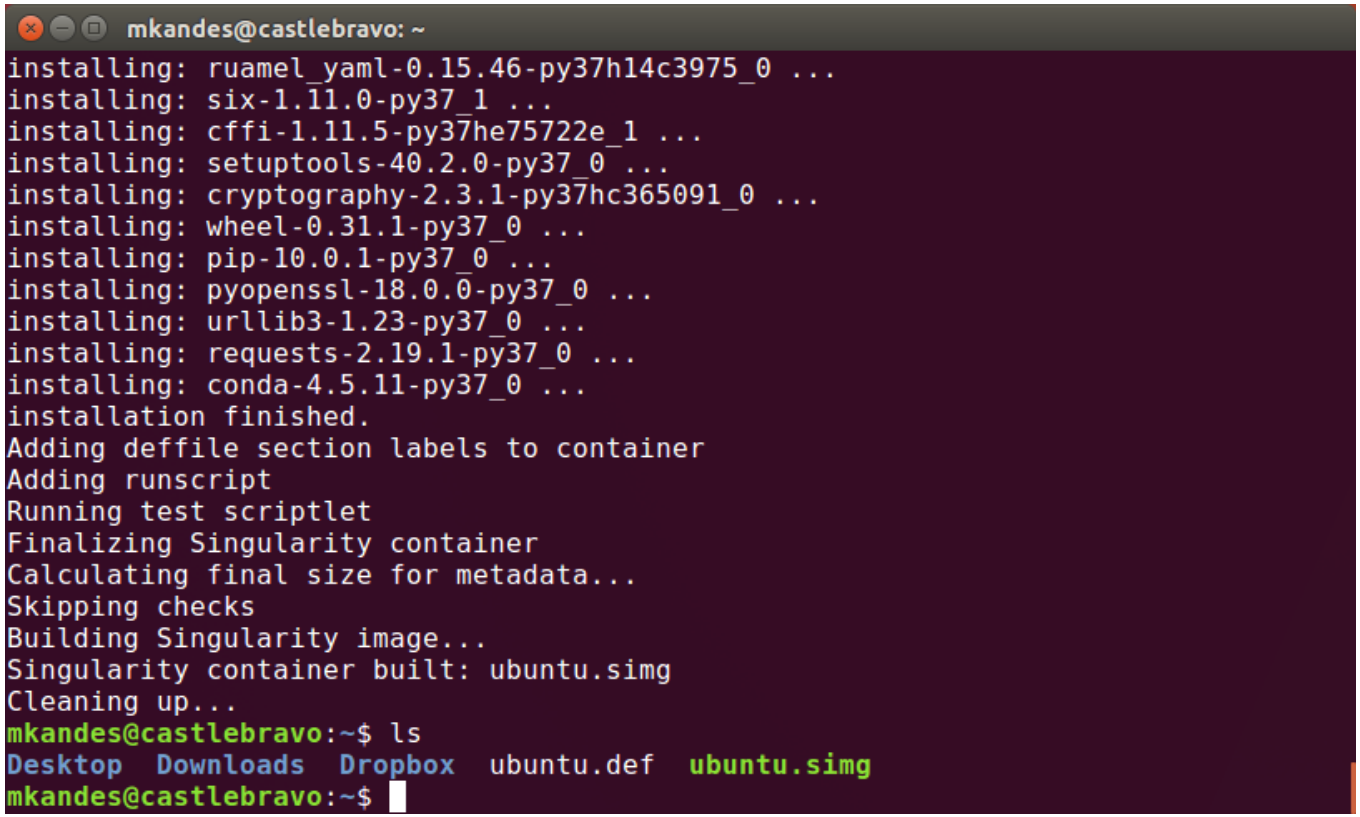
  2. Make it easy for you to see what is installed within these Singularity containers; and

  3. Make available to you the same base definition files we use to build our Singularity containers, which can serve as a starting point for your own custom Singularity containers.

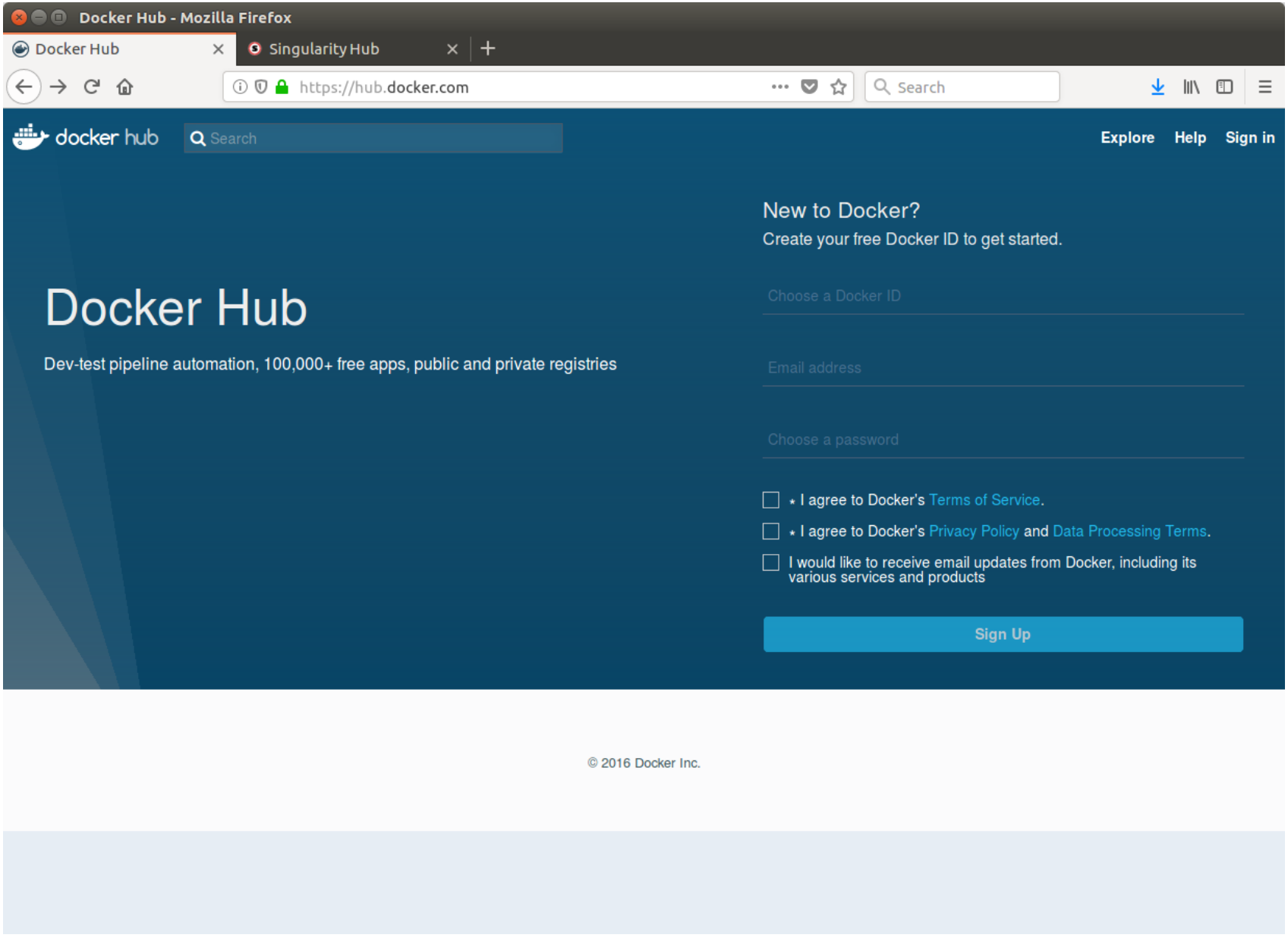
- `https://github.com/mkandes/naked-singularity`

How to *build* a Singularity container ...

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



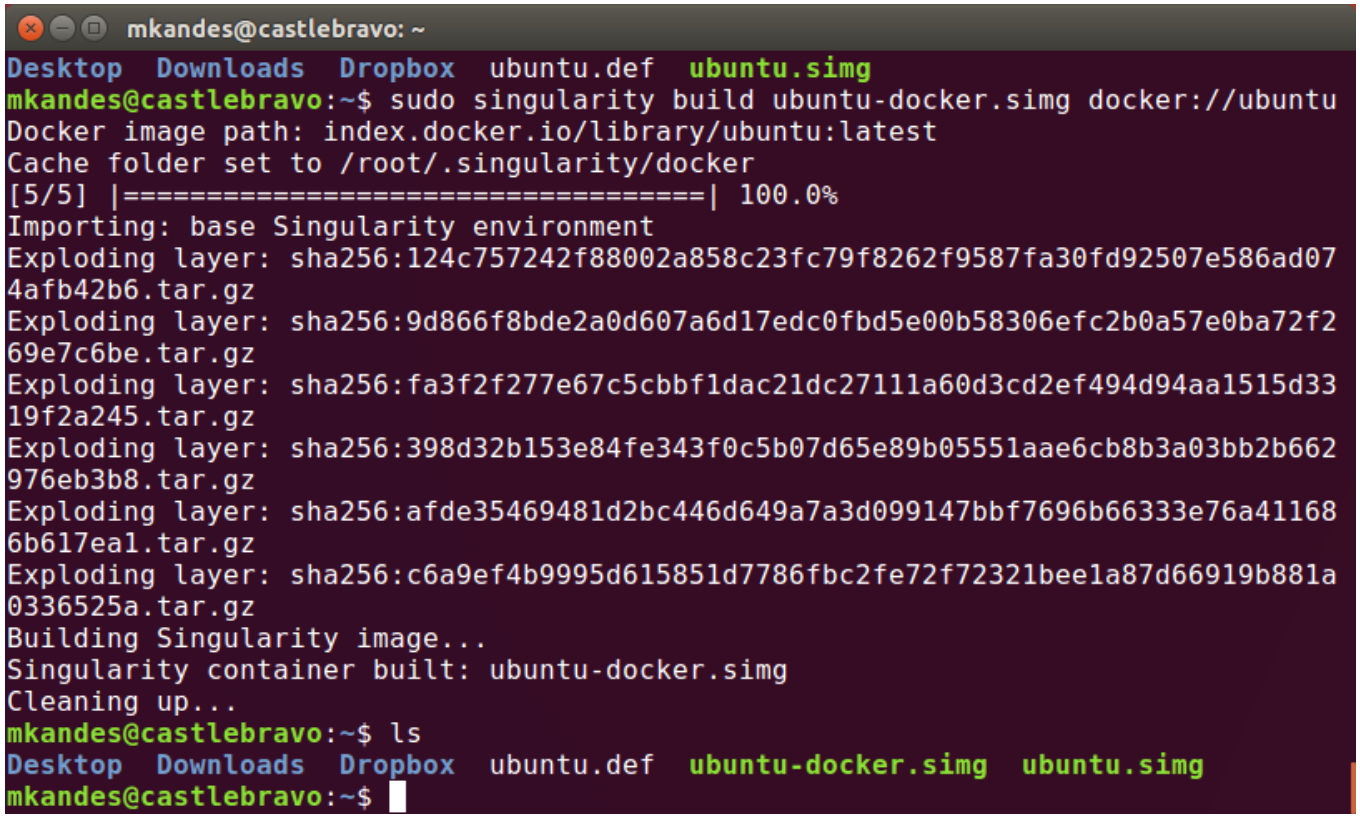

```
sudo singularity build ubuntu.sif ubuntu.def
```



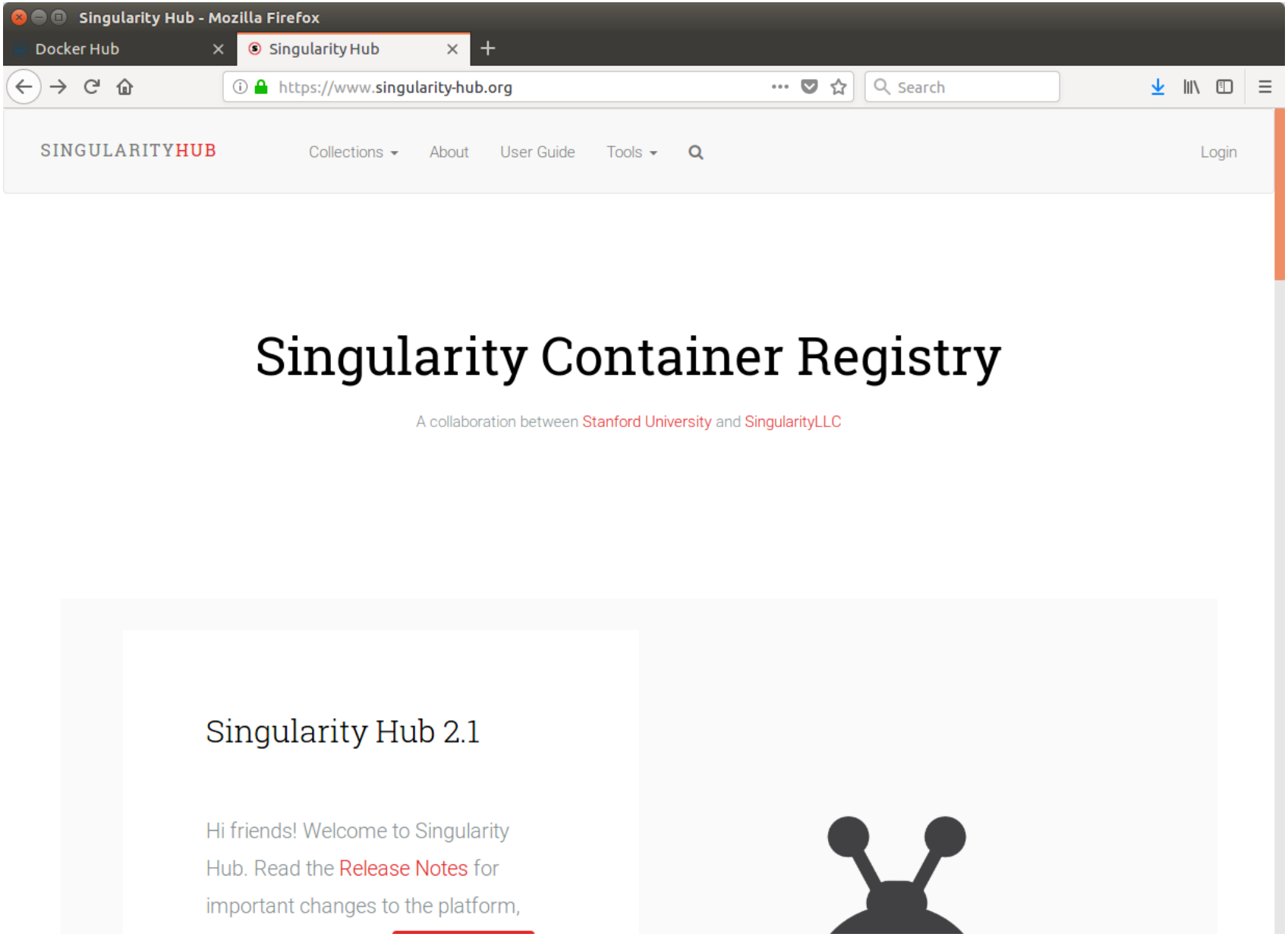
https://hub.docker.com

How to *build* a Singularity container ...

... from an existing Docker container on DockerHub:



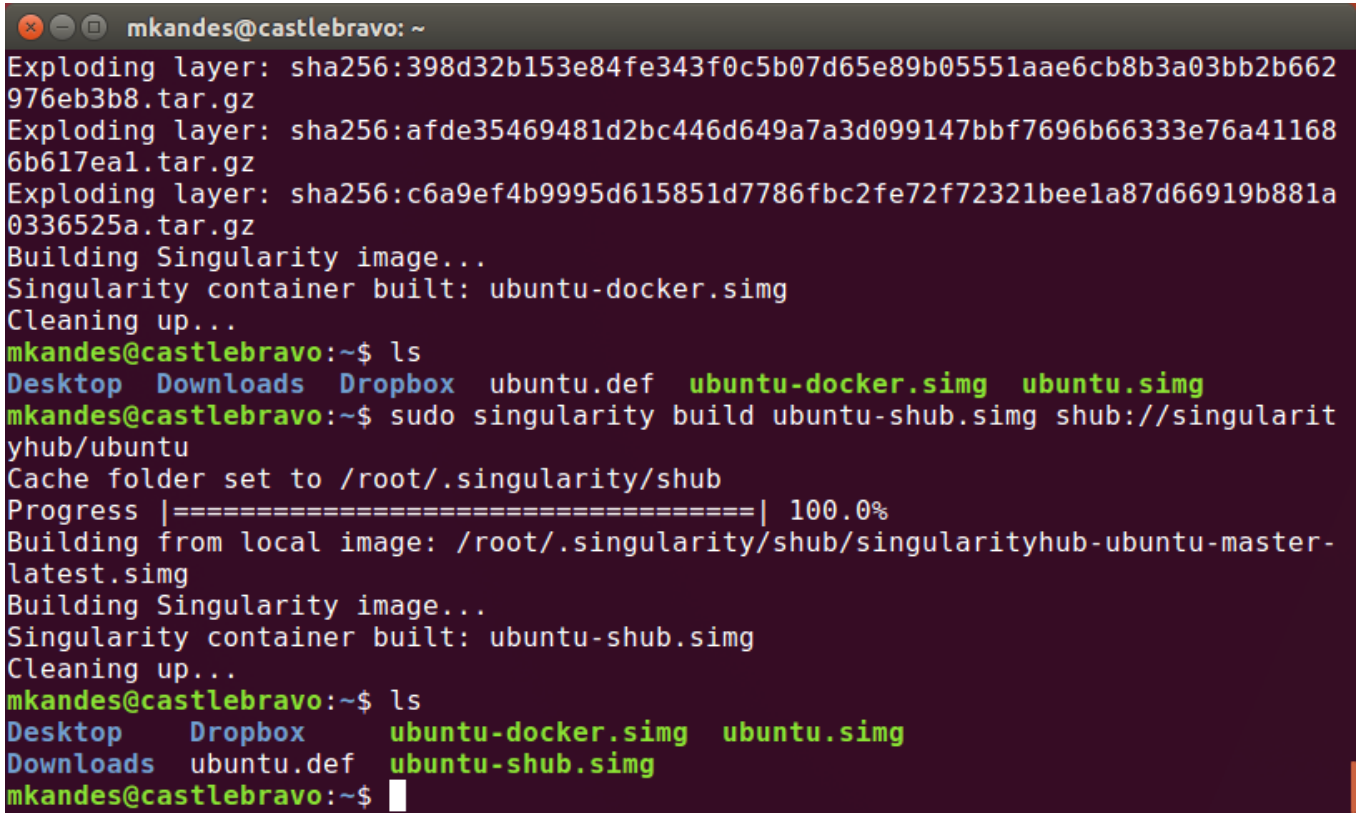

```
sudo singularity build ubuntu-docker.sif docker://ubuntu
```



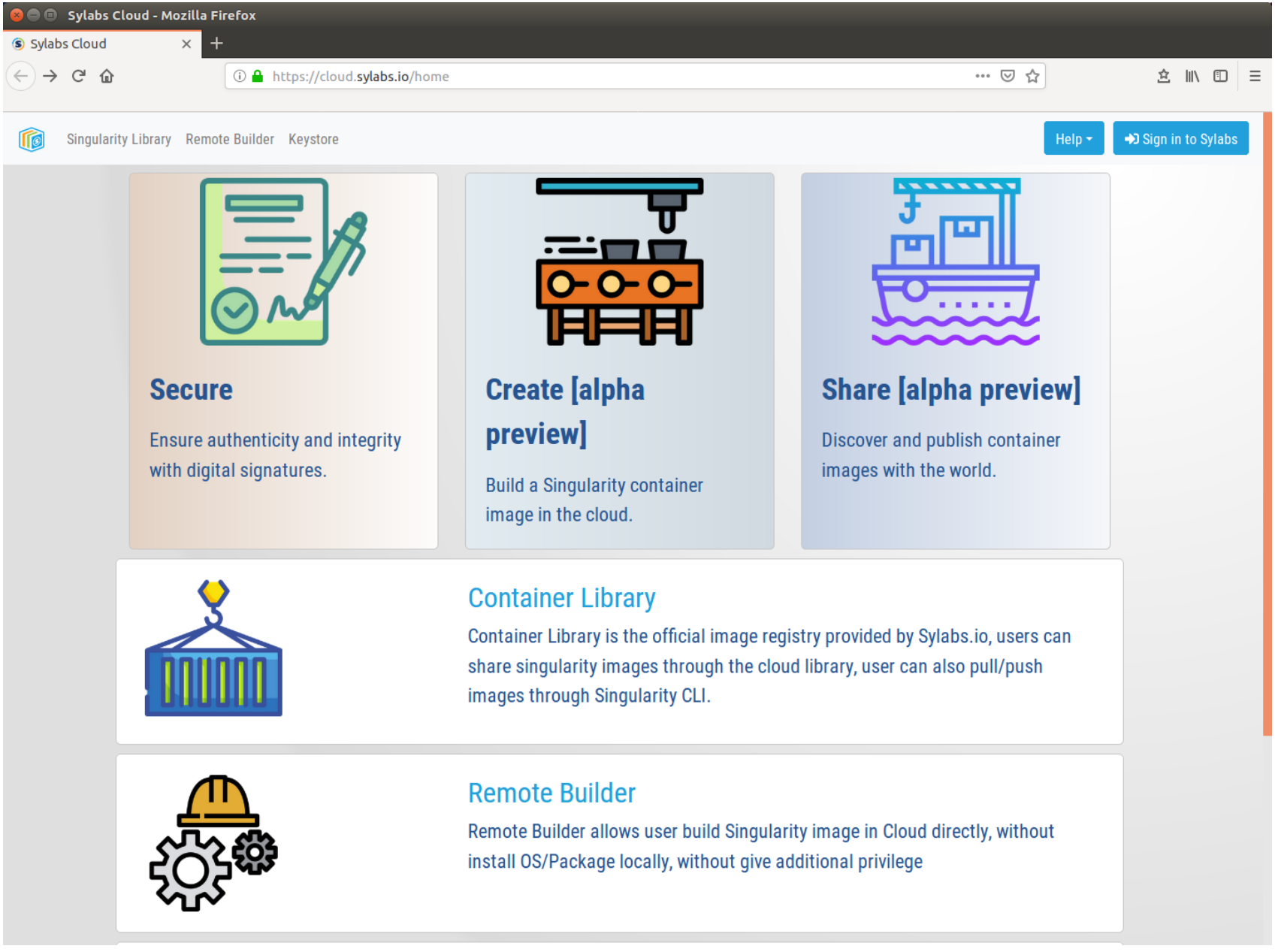
https://www.singularity-hub.org

How to *build* a Singularity container ...

... from an existing Singularity container on SingularityHub:




```
sudo singularity build ubuntu-shub.sif shub://mkandes/ubuntu
```



https://cloud.sylabs.io

If you *build* it ...

... what can you do with it?

Essential Singularity

The main Singularity command

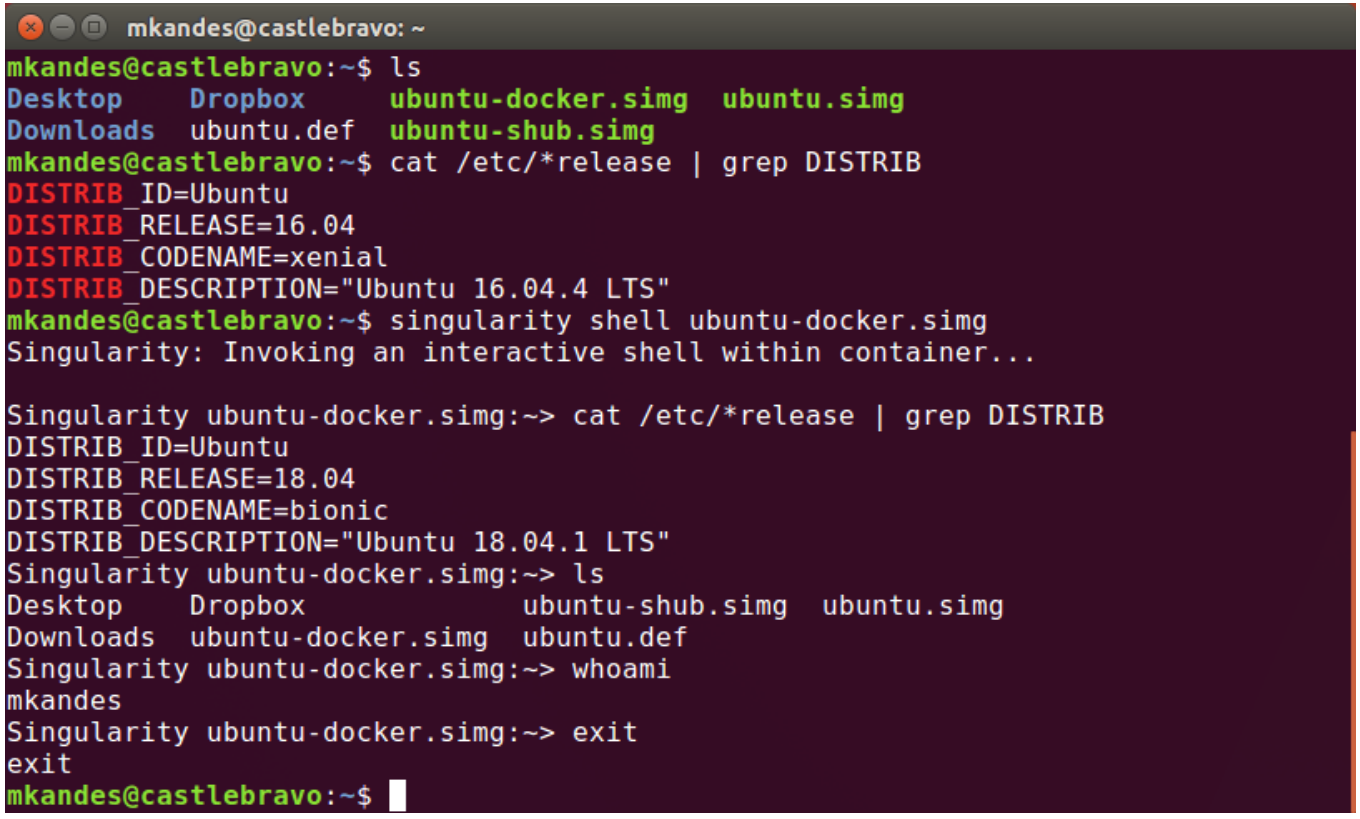
```
singularity [options] <subcommand> [subcommand options] ...
```

has three essential subcommands:

- `build`: Build your own container from scratch using a Singularity definition (or recipe) file; download and assemble any existing Singularity container; or convert your containers from one format to another (e.g., from Docker to Singularity)
- `shell`: Spawn an interactive shell session in your container.
- `exec`: Execute an arbitrary command within your container.

How to spawn an interactive *shell* ...

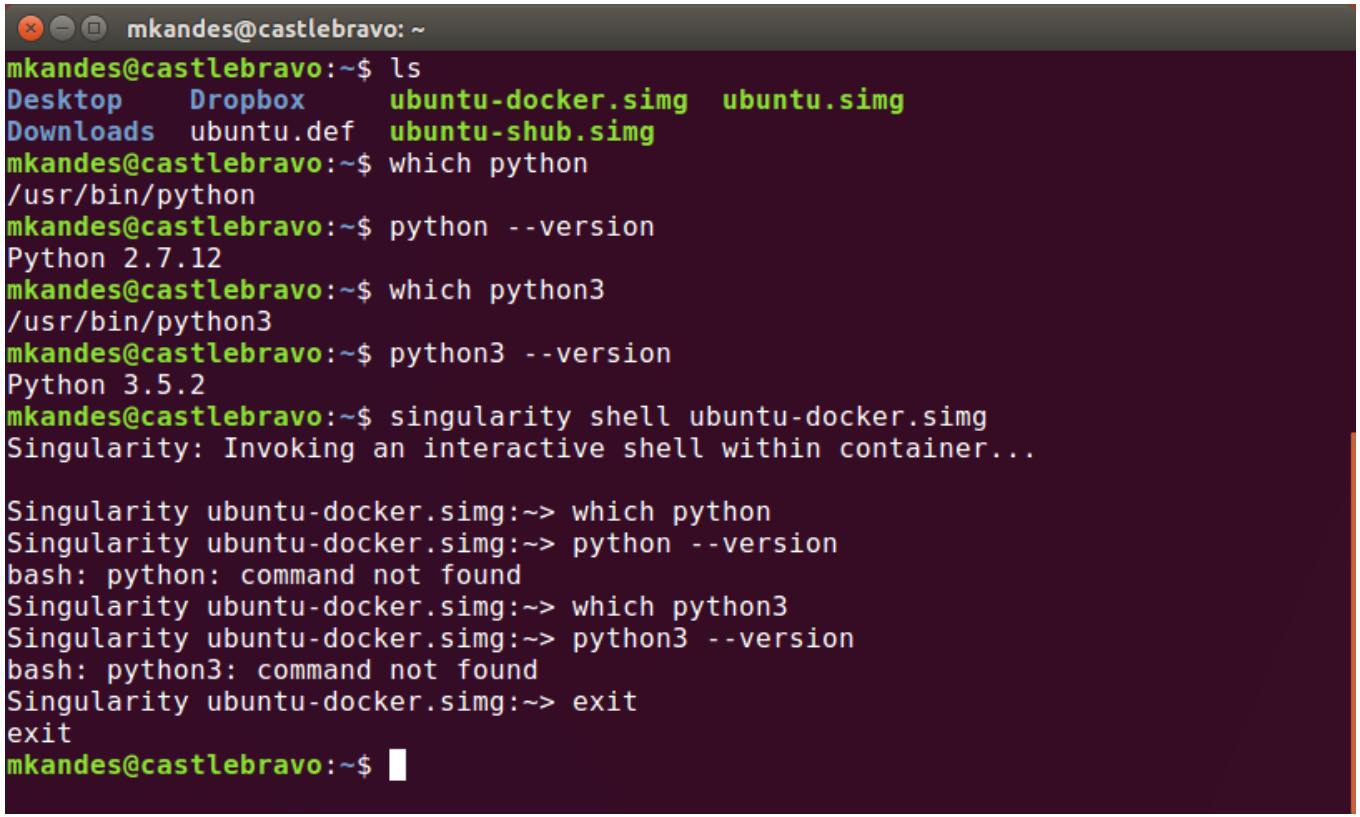
... within a Singularity container:




```
singularity shell ubuntu-docker.sif
```

Use the interactive *shell* to ...

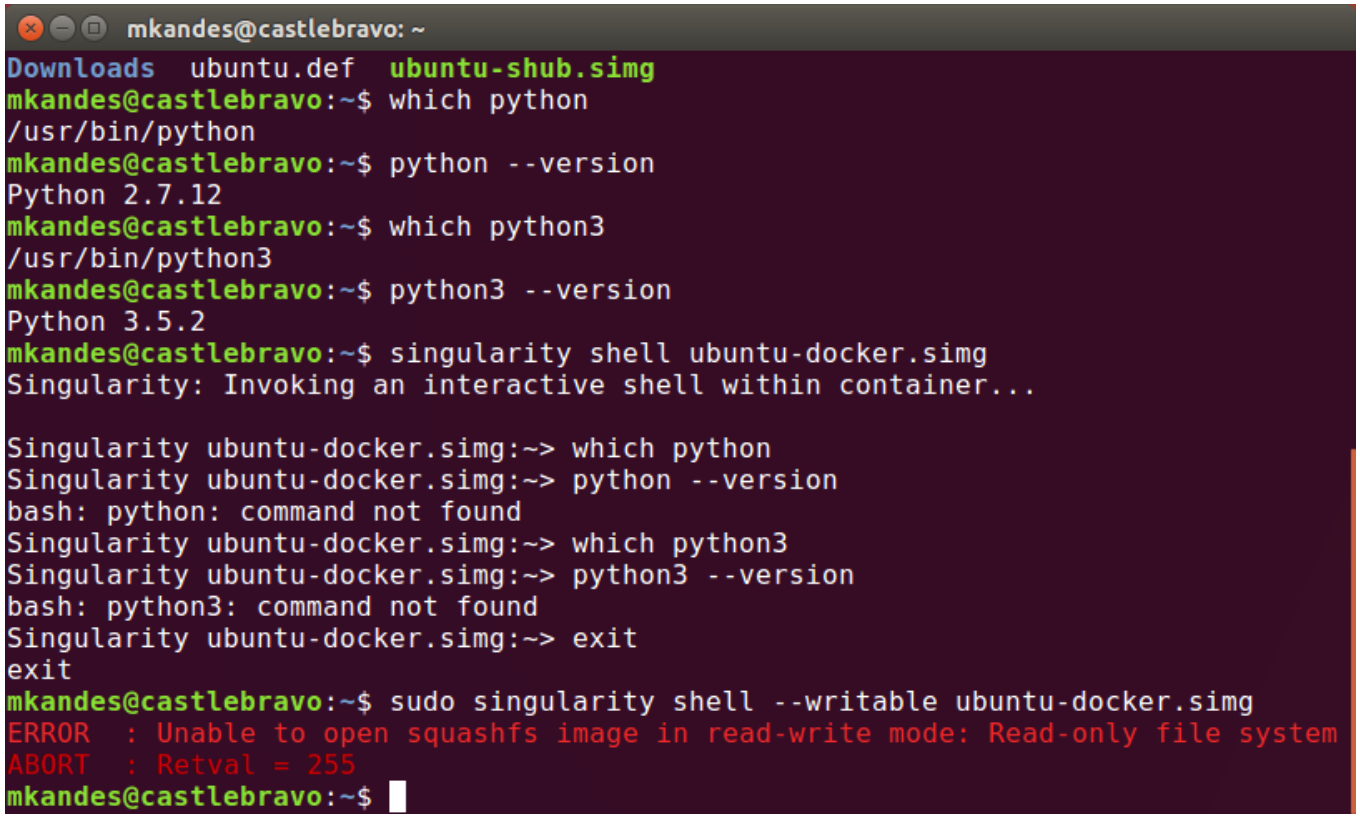
... explore the software environment of a Singularity container:



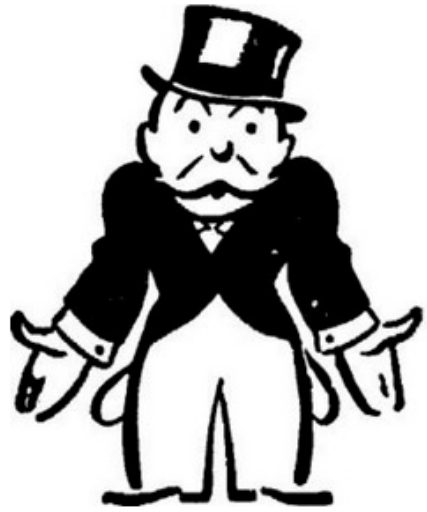

```
singularity shell ubuntu-docker.sif
```

Use the interactive *shell* to ...

... modify the contents of a Singularity container:




```
sudo singularity shell --writable ubuntu-docker.sif
```



```
ERROR : Unable to open squashfs image in read-write mode:
        Read-only file system
ABORT : Retval = 255
```

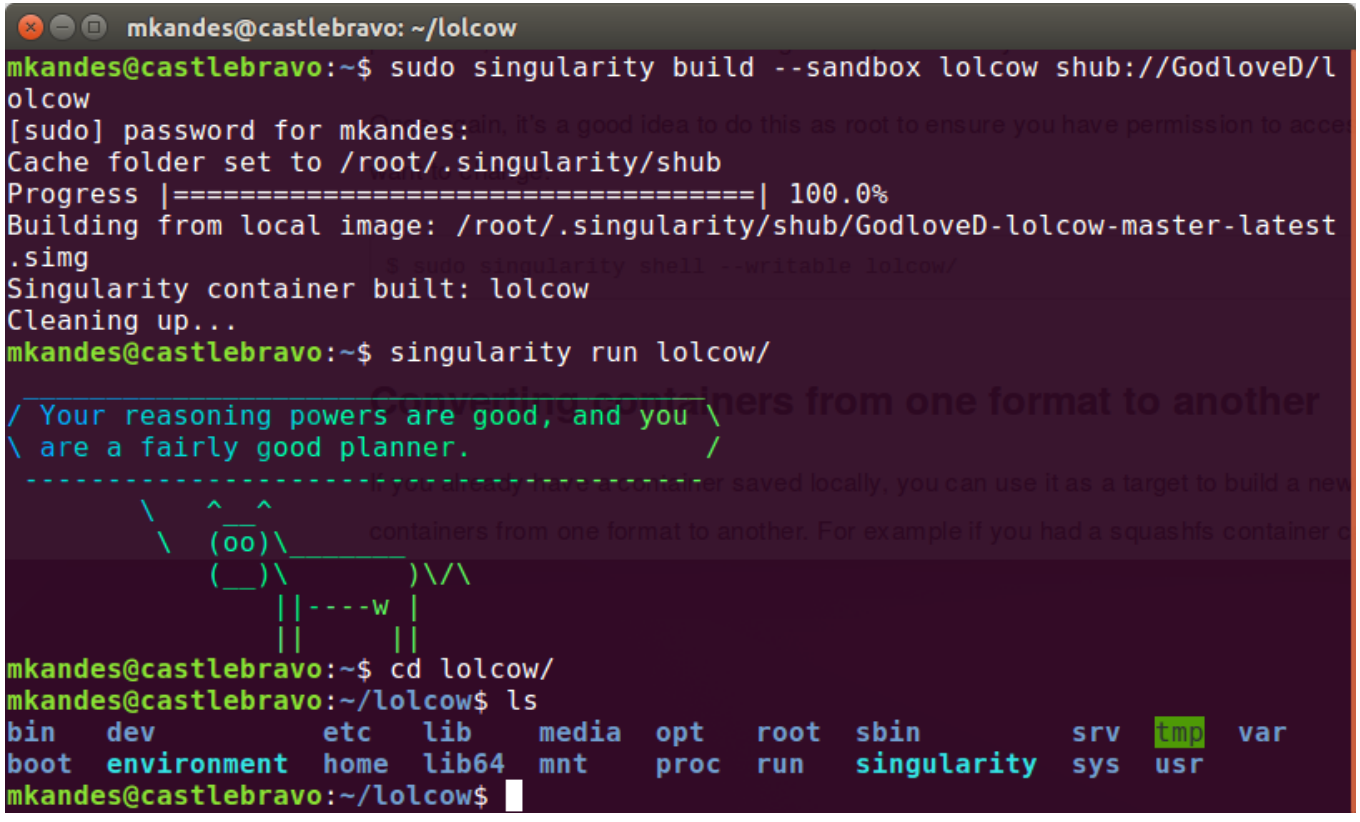
Singularity Container Image Formats

There are now (only) 2 different Singularity container image formats:

- Compressed READ-ONLY **Singularity Image File (SIF)** format suitable for production (default)
- writable **(ch)root directory** called a sandbox for interactive (`--writable`) development (–sandbox option)

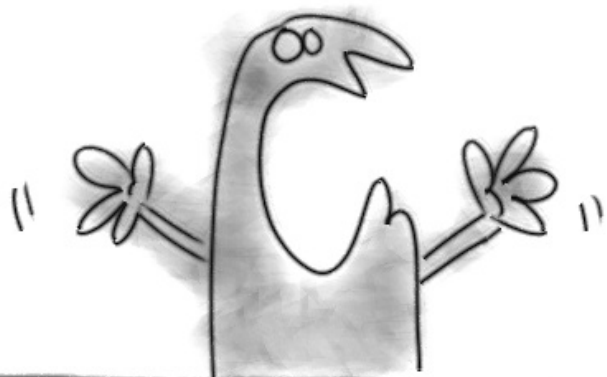
How to *build* a `--sandbox` Singularity container ...

... from an existing Singularity container on SingularityHub:




```
sudo singularity build --sandbox lolcow shub://GodloveD/lolcow
```

Now What?!!



Essential Singularity

The main Singularity command

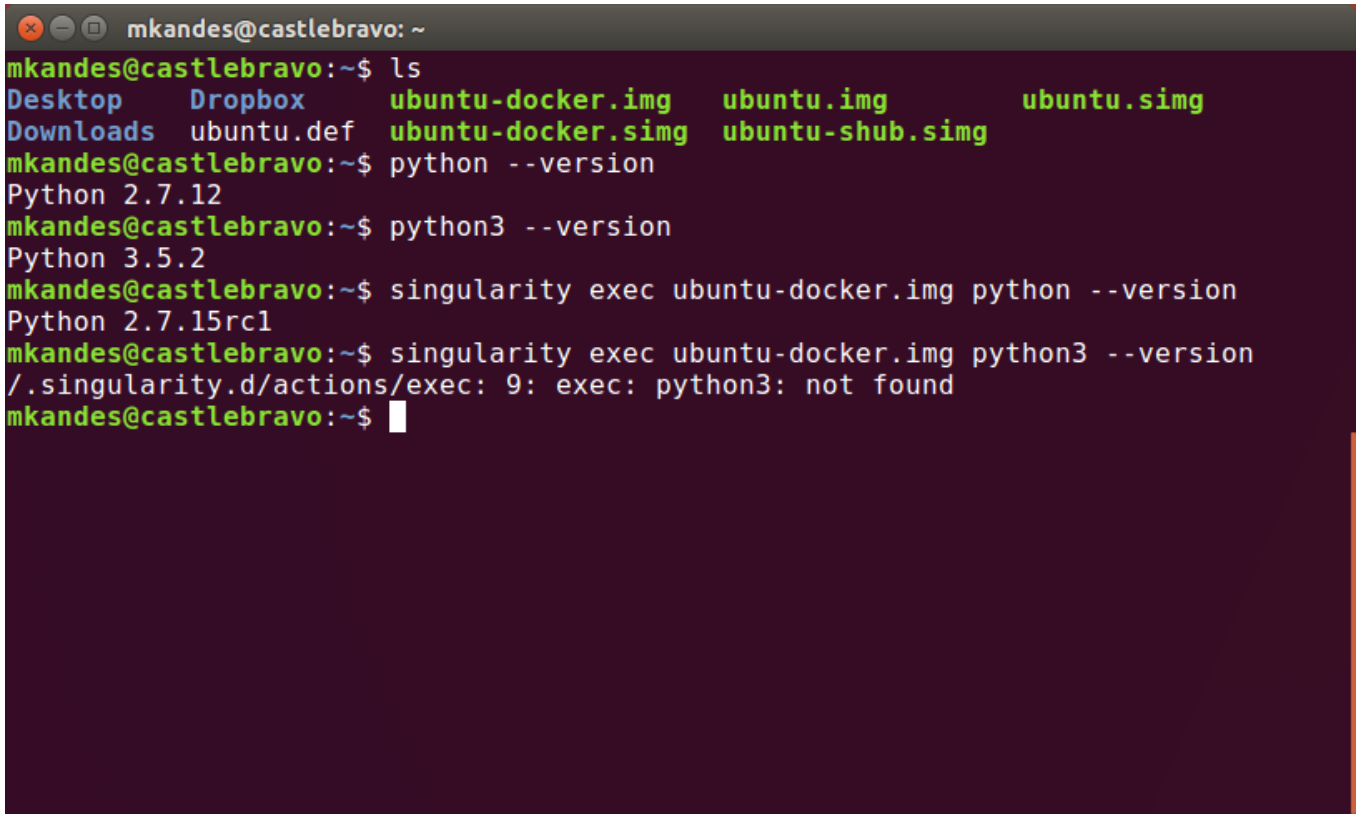
```
singularity [options] <subcommand> [subcommand options] ...
```

has three essential subcommands:

- `build`: Build your own container from scratch using a Singularity definition (or recipe) file; download and assemble any existing Singularity container; or convert your containers from one format to another (e.g., from Docker to Singularity)
- `shell`: Spawn an interactive shell session in your container.
- `exec`: Execute an arbitrary command within your container.

How to *execute* ...

... arbitrary commands within a Singularity container:




```
singularity exec ubuntu-docker.img python --version
```

Essential Singularity

The main Singularity command

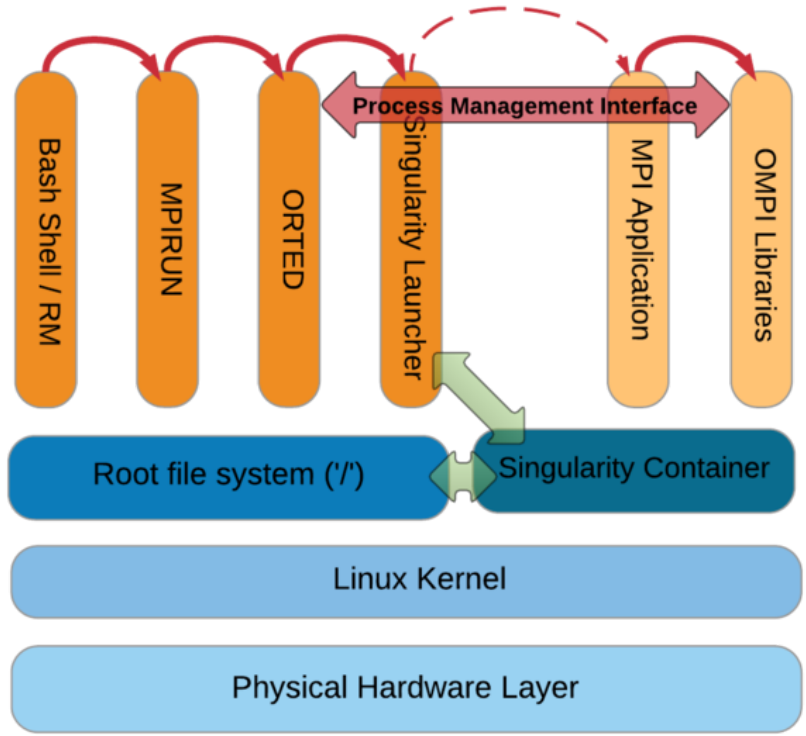
```
singularity [options] <subcommand> [subcommand options] ...
```

has three essential subcommands:

- `build`: Build your own container from scratch using a Singularity definition (or recipe) file; download and assemble any existing Singularity container; or convert your containers from one format to another (e.g., from Docker to Singularity)
- `shell`: Spawn an interactive shell session in your container.
- `exec`: Execute an arbitrary command within your container.

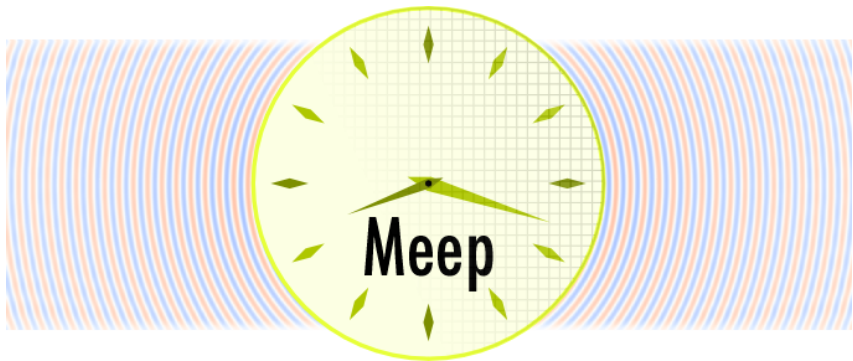
MPI-based Singularity Containers

- Use same Message Passing Interface (MPI) distribution and version within container as would be used outside the container.
- If using Infiniband (IB), 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 TSCC's native software environment.



MPI-based Singularity Containers: MEEP

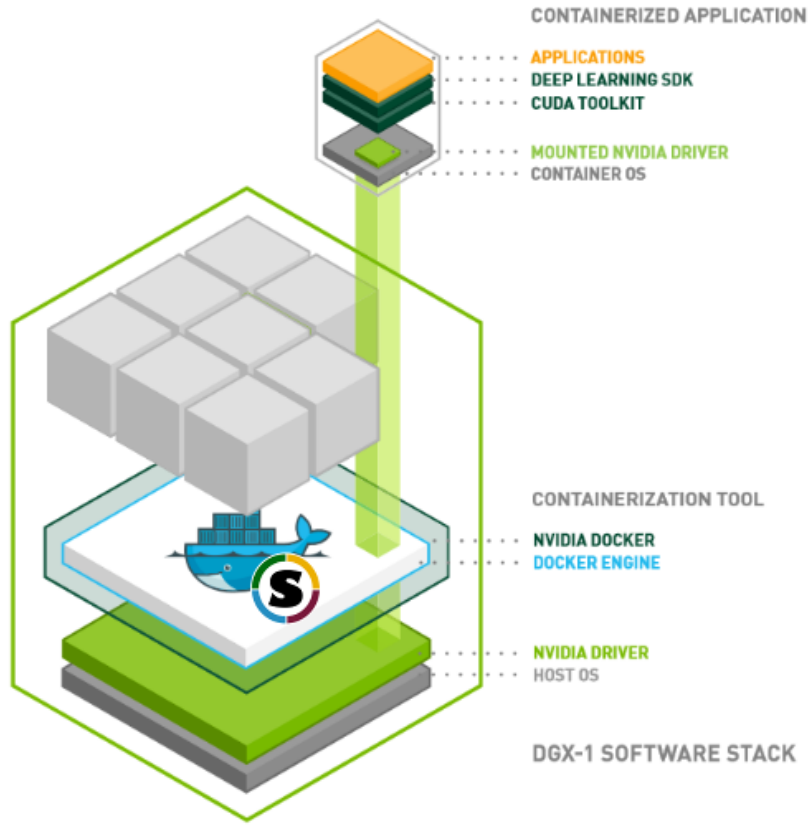
mkandes@comet-ln2:~/Software/meep

```
#!/usr/bin/env bash

#SBATCH --job-name="meep-example"
#SBATCH --account=use300
#SBATCH --partition=debug
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=24
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH --output="meep-example.o%j.%N"

declare -xr COMPILER_MODULE='gnu/4.9.2'
declare -xr MPI_MODULE='openmpi_ib/1.8.4'
declare -xr SINGULARITY_MODULE='singularity/2.6.1'

declare -xr SINGULARITY_IMAGE_DIR='/share/apps/compute/singularity/images'

module purge
module load "${COMPILER_MODULE}"
module load "${MPI_MODULE}"
module load "${SINGULARITY_MODULE}"
module list
printenv

time -p ibrun singularity exec "${SINGULARITY_IMAGE_DIR}/meep/meep.simg" \
  meep /opt/meep/scheme/examples/parallel-wvgs-force.ctl
                                                    1,2           All
```

`mpirun -np X singularity exec meep.sif meep parallel-wvgs-force.ctl`

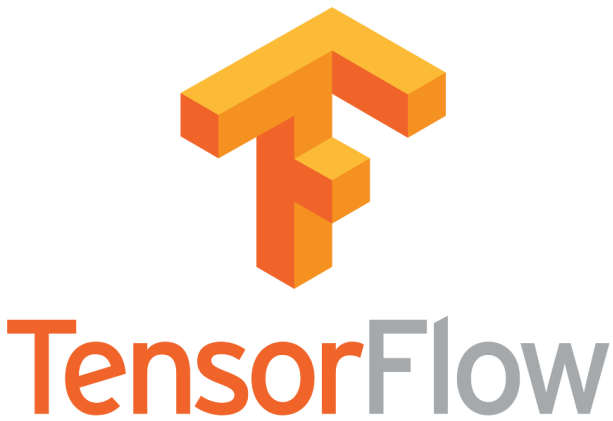
GPU-accelerated Singularity Containers

- GPU-accelerated containers also require an interface for accessing GPU drivers and libraries on the underlying host system.
- Traditionally, you would install the same driver and libraries within container that match distribution and version of them available on the host system.
- Today, Singularity actually allows you to bind mount the GPU driver and its supporting libraries at runtime with the `--nv` option.



GPU-accelerated Singularity Containers: TensorFlow

- TensorFlow is an open source software library for high performance numeric and symbolic computation, and is most popularly used today for machine learning applications such as neural networks.
- Like many of the most popular machine learning frameworks, TensorFlow continues to evolve rapidly. At present, the latest versions of TensorFlow are incompatible with the version of TSCC's `glibc` library.



GPU-accelerated Singularity Containers: TensorFlow

mkandes@comet-ln3:~/Software/tensorflow

```
#!/usr/bin/env bash

#SBATCH --job-name="tensorflow-gpu-example"
#SBATCH --account=use300
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH --output="tensorflow-gpu-example.o%j.%N"

declare -xr SINGULARITY_MODULE='singularity/2.6.1'
declare -xr SINGULARITY_IMAGE_DIR='/share/apps/gpu/singularity/images'

module purge
module "${SINGULARITY_MODULE}"
module list
printenv

time -p singularity exec "${SINGULARITY_IMAGE_DIR}/tensorflow/tensorflow-gpu.simg" \
  python2 /opt/tensorflow/tensorflow/examples/tutorials/mnist/mnist_deep.py

time -p singularity exec "${SINGULARITY_IMAGE_DIR}/tensorflow/tensorflow-gpu.simg" \
  python3 /opt/tensorflow/tensorflow/examples/tutorials/mnist/mnist_deep.py
~
                                                              14,50         All
```

```
singularity exec keras-tensorflow-gpu.sif python mnist_deep.py
```

GPU-accelerated Singularity Containers: TensorFlow

mkandes@comet-ln3:~/Software/tensorflow

```
#!/usr/bin/env bash

#SBATCH --job-name="tensorflow-gpu-example"
#SBATCH --account=use300
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH --output="tensorflow-gpu-example.o%j.%N"

declare -xr SINGULARITY_MODULE='singularity/2.6.1'
declare -xr SINGULARITY_IMAGE_DIR='/share/apps/gpu/singularity/images'

module purge
module "${SINGULARITY_MODULE}"
module list
printenv

time -p singularity exec --nv "${SINGULARITY_IMAGE_DIR}/tensorflow/tensorflow-gpu.simg" \
  python2 /opt/tensorflow/tensorflow/examples/tutorials/mnist/mnist_deep.py

time -p singularity exec --nv "${SINGULARITY_IMAGE_DIR}/tensorflow/tensorflow-gpu.simg" \
  python3 /opt/tensorflow/tensorflow/examples/tutorials/mnist/mnist_deep.py
~
"run-tensorflow-gpu.slurm" 25L, 805C written                  12,0-1        All
```

```
singularity exec --nv keras-tensorflow-gpu.sif python mnist_deep.py
```

Exoskeletal (Dependency-Only) Singularity Containers

- Some software applications may need to be installed in your $HOME directory on the underlying host system, but may require software dependencies not available and/or not supported on the host system. e.g., Julia with GPU support.
- Most often, users simply want to install additional packages in the Singularity containers we build and maintain for you.
- In both cases, the you can use any container's environment to install the software you need to in your $HOME directory. Note, however, this can create some dependency confusion down the road when you forget which packages were installed with what container.

```
singularity exec ubuntu.sif pip install --user networkx
```

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, your software is now completely portable between the different HPC systems you want to run on.

3. And finally, you now have discrete software units (containers) that you can use to help maintain science reproducibility over the lifetime of a project, independent of how the software environment on any given HPC system changes over time.

Questions?