Using Containers on RMACC Summit

Jared Baker
Jared.Baker@uwyo.edu

Martin Cuma
Martin.Cuma@utah.edu

Andrew Monaghan
Andrew.Monaghan@colorado.edu

Slides and exercises available for download at
https://github.com/rctraining/rmacc_2018_container_tutorial

Outline

- Introduction to containers
- Singularity commands and options
- Hands-on: Running containers on RMACC Summit
- Special cases: Running containers for MPI and GPU jobs

/\/\ Break /\/\

- Container recipes
- Hands-on: Building containers from recipes (SingularityHub)
- Troubleshooting and caveats
- Summary

Introduction to Containers

Seacontainersales.com

What is a container?

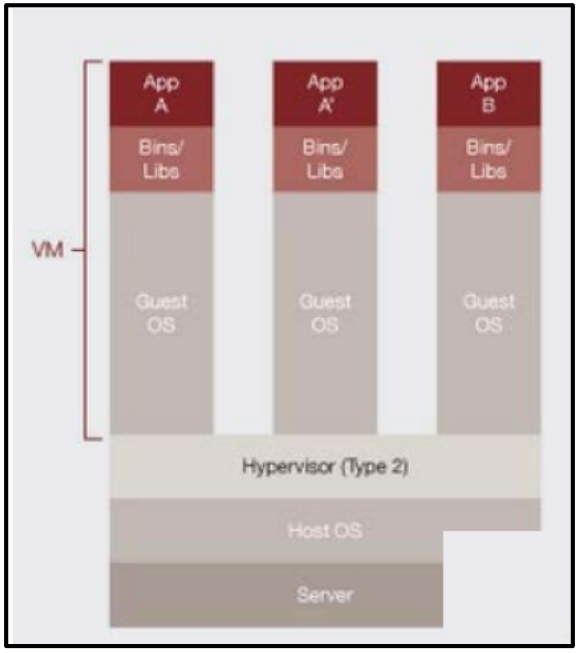
A container is a portable environment that packages some or all of the following: an operating system, software, libraries, compilers, data and workflows. Containers enable:

- Mobility of Compute
- Reproducibility (software and data)
- User Freedom

Virtualization (1)

Hardware virtualization (not used by containers!)

- Can run many OS's on same hardware (machine)
- E.g., VirtualBox, VMWare

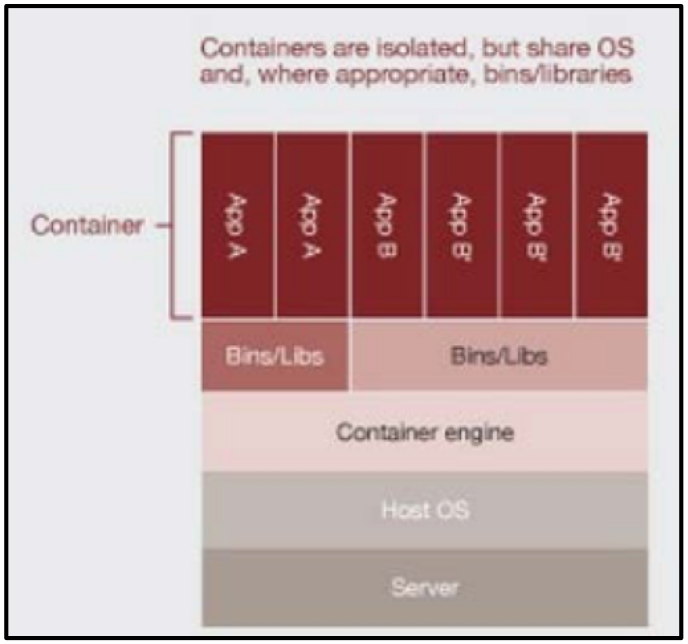


Material courtesy: M. Cuma, U. Utah

Virtualization (2)

OS-level virtualization (used by containers!)

- Can run many isolated OS instances (guests) under a server OS (host)
- Also called containers
- E.g., Docker, Singularity



Best of both worlds: isolated environment that user wants, but can leverage host OS resources (network, I/O partitions, etc.)

Material courtesy: M. Cuma, U. Utah

Containerization software

- Docker 
  - Well established – largest user base
  - Has Docker Hub for container sharing
  - Problematic with HPC
- Charliecloud; Shifter
  - Designed for HPC
  - Based on Docker
  - Less user-friendly
- Singularity 
  - Primary focus of today's tutorial
  - More info on next slide…

Why Singularity?

- Singularity is a comparably safe container solution for HPC
  - User is same inside/outside container
  - User cannot escalate permissions without administrative privilege
- Can support MPI and GPU resources on HPC (scaling)
- Can use HPC filesystems
- Supports the use of Docker containers
- Container is seen as a file, and can be operated on as a file

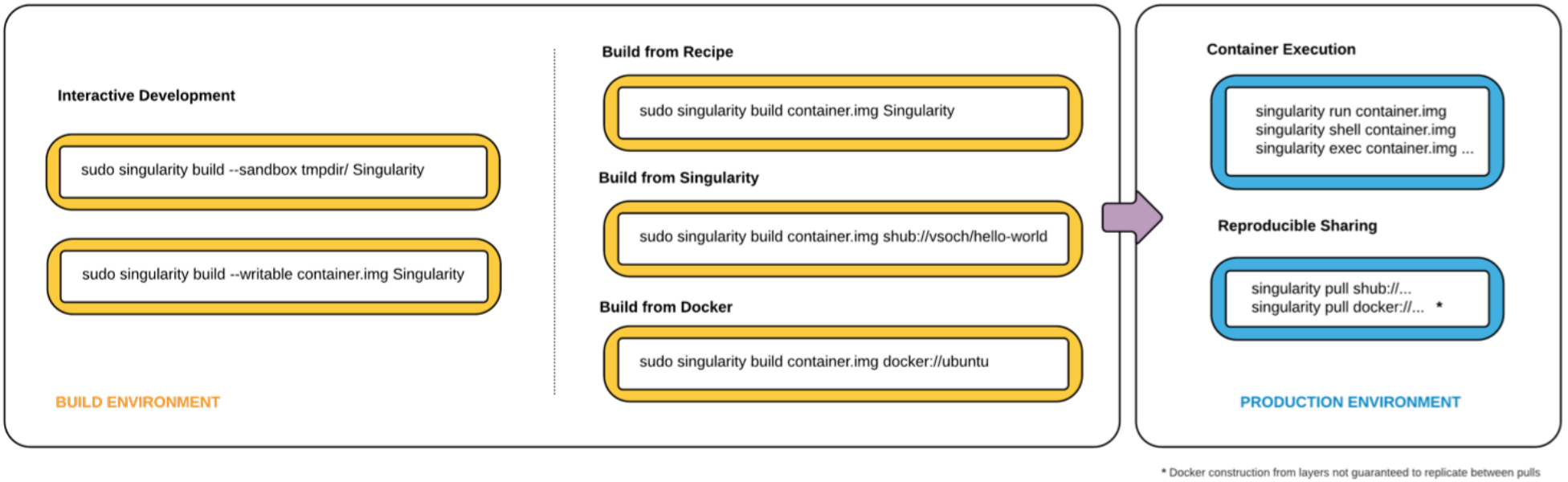
Singularity commands and options

Interactive development

Building

Running

The Singularity workflow



Second half of tutorial

First half of tutorial

https://singularity.lbl.gov

Interactive development

Within your build environment (a laptop, workstation, or a server that you control): develop and test containers

- using --sandbox (build into a writable *directory*)
  - `sudo singularity build --sandbox tmpdir/ mycont.def`

- or using --writable (build into a writable *ext3* image)
  - `sudo singularity build --writable tmp.img mycont.def`

Building containers

After development, within your build environment (a laptop, workstation, or a server that you control): build your production containers with a *squashfs* filesystem...

- Build from a “recipe” that you developed.
  - `sudo singularity build mycont.img mycont.def`
- Build from an existing container on Singularity Hub
  - `sudo singularity build mycont.img shub://somewhere`
- Build from an existing container on Docker Hub
  - `sudo singularity build mycont.img docker://somewhere`

Container Formats

- **squashfs**: the default container format is a compressed read-only file system that is widely used for things like live CDs/USBs and cell phone OS's
- **ext3**: (also called `writable`) a writable image file containing an ext3 file system that was the default container format prior to Singularity version 2.4
- **directory**: (also called `sandbox`) standard Unix directory containing a root container image
- **tar.gz**: zlib compressed tar archive
- **tar.bz2**: bzip2 compressed tar archive
- **tar**: uncompressed tar archive

https://singularity.lbl.gov

Running containers

Now, on ***any system*** with Singularity, even without administrative privilege, you can retrieve and use containers:

- Download a container from Singularity Hub or Docker Hub
  - `singularity pull shub://some_image`
  - `singularity pull docker://some_image`
- Run a container
  - `singularity run mycont.img`
- Execute a specific program within a container
  - `singularity exec mycont.img python myscript.py`
- "Shell" into a container to use or look around
  - `singularity shell mycont.img`
- Inspect an image
  - `singularity inspect --runscript mycont.img`

Summary of Singularity Commands

build: Build a container on your user endpoint or build environment

exec: Execute a command to your container

inspect: See labels, run and test scripts, and environment variables

pull: pull an image from Docker or Singularity Hub

run: Run your image as an executable

shell: Shell into your image

https://singularity.lbl.gov

Alternative to Singularity on HPC systems: Udocker

Python-based tool for executing Docker containers without requiring administrative privilege (i.e., can be used on RMACC Summit).

Alternative to using Singularity for Docker containers.

User can install in seconds without needing root access.

Main advantages: Provides more flexibility to modify Docker containers on RMACC Summit (e.g., user can install software in existing container as root, and can install python/R packages).

Main disadvantages: Does not work for Singularity containers; Cannot create containers from scratch.

Easiest way to teach is by example….

Hands-on Examples

- Running containers with Singularity
- Running containers with UDocker

Logging in

If you are using a temporary account: In a terminal or Git Bash window, type the following:

```
ssh -X userNNNN@tutorial-login.rc.colorado.edu
```

```
Password:
```

***Note that `userNNNN` is a temporary account that I have assigned to you. If you don't have one, let me know.*

If you are using your regular RC account (e.g., 'monaghaa' for me), log in as you normally would.

Navigating

Now type the following commands:

```
ssh -X scompile

cd /scratch/summit/$USER

git clone https://github.com/rctraining/rmacc_2018_container_tutorial

cd rmacc_2018_container_tutorial

ls
```

You should see three directories, one for each of the examples we will do today.

Singularity Examples

Go to the example directory

Now go to the Singularity example directory

```
cd /projects/$USER/rmacc_2018_container_tutorial/run_container_w_singularity
```

List the contents of the directory and see a description of each file:

```
ls
cat README.md
```

If you get behind or have issues, look in '`tutorial_steps.txt`'

Running a container from Singularity Hub

Pull an existing container image that someone else posted:

```
singularity pull --name hello.img
shub://monaghaa/singularity_git_tutorial
```

…And run it

```
singularity run hello.img
```

…And look at the script inside

```
singularity exec hello.img cat /code/hello.sh
```

Running a container from Docker Hub

Let's grab the stock docker python container:

```
singularity pull --name pythond.img docker://python
```

…And run python

```
singularity exec pythond.img python
```

…And shell into the container and look around

```
singularity shell pythond.img
```

…try "`ls /`" What directories do you see?

Running a container from Docker Hub (2)

Let's run an external python script using the containerized version of python:

First create a script called *"myscript.py"* as follows:

```
echo 'print("hello world from the outside")' >myscript.py
```

…And now let's run the script using the containerized python

```
singularity exec pythond.img python ./myscript.py
```

…Conclusion: Scripts and data can be kept inside or outside the container. In some instances (e.g., large datasets or scripts that will change frequently) it is easier to containerize the software and keep everything else outside.

Binding directories

On Summit, most host directories are “bound” (mounted) by default. But on other systems, or in some instances on Summit, you may want to access a directory that is not already mounted. Let’s try it.

Note that the “/opt” directory in ”pythond.img” is empty. But the Summit ”/opt” directory is not. Let’s bind it:

```
singularity shell –bind /opt:/opt pythond.img
```

…It isn’t necessary to bind like-named directories like we did above. Try binding your /home/$USER directory to /opt.

***Note: If your host system does not allow binding, you will need to create the host directories you want mounted when you build the container (as root on, e.g., your laptop)

Udocker Example

Go to the example directory

Go to the Udocker example directory

```
cd /projects/$USER/rmacc_2018_container_tutorial/run_container_w_udocker/
```

List the contents of the directory and see a description of each file:

```
ls
cat README.md
```

If you get behind or have issues, look in '`tutorial_steps.txt`'

Install Udocker and create a container

Install Udocker

```
./make_udocker.sh
```

Make local image of r-base (from: https://hub.docker.com/_/r-base/)

```
./udocker pull r-base:latest
```

#Create a container from the image

```
./udocker create --name=myR r-base:latest
```

Modify and use the container

Open the container's "R" command line & install the "lme4" package.

- "lme4" is a package that creates linear mixed effects models.
- The '-v' option binds an external directory to an internal directory
- The '-w' option specifies the working directory in the container
- The '-u' option specifies the user ("docker" in this case)

```
./udocker run -v "$PWD":/home/docker -w /home/docker -u docker myR R
```

…then type at the "R" prompt (note, this may take a few minutes):

```
>install.packages("lme4")
```

Now run the script "lme_example.R" to plot some data and create a model

```
./udocker run -v "$PWD":/home/docker -w /home/docker -u docker
  myR Rscript lme_example.R
```

#View the plot of the data if you have X11-forwarding enabled

```
gs Rplots.pdf
```

Special Cases: MPI and GPUs

Running a multi-core (MPI) job with a container (1)

Syntax (after loading singularity and openmpi):

```
"mpirun –np X singularity exec container.img executable-in-container"
```

1. mpirun is called by the resource manager or the user directly from a shell
2. Open MPI then calls the process management daemon (ORTED)
3. The ORTED process launches the Singularity container requested by the mpirun command
4. Singularity builds the container and namespace environment
5. Singularity then launches the MPI application within the container
6. The MPI application launches and loads the Open MPI libraries
7. The Open MPI libraries connect back to the ORTED process via the Process Management Interface (PMI)
8. At this point the processes within the container run as they would normally directly on the host.

http://singularity.lbl.gov/docs-hpc

Running a multi-core (MPI) a container (2)

What are supported Open MPI Version(s)? To achieve proper container'ized Open MPI support, you should use Open MPI version 2.1. There are however three caveats:

Open MPI 1.10.x *may* work but we expect you will need exactly matching version of PMI and Open MPI on both host and container (the 2.1 series should relax this requirement)

Open MPI 2.1.0 has a bug affecting compilation of libraries for some interfaces (particularly Mellanox interfaces using libmxm are known to fail). If your in this situation you should use the master branch of Open MPI rather than the release.

Using Open MPI 2.1 does not magically allow your container to connect to networking fabric libraries in the host. If your cluster has, for example, an infiniband network you still need to install OFED libraries into the container. Alternatively you could bind mount both Open MPI and networking libraries into the container, but this could run afoul of glib compatibility issues (its generally OK if the container glibc is more recent than the host, but not the other way around)

Source (and additional context): http://singularity.lbl.gov/docs-hpc

Running containers on GPUs

Syntax (after loading Singularity on a gpu node ["--partition=sgpu"]):

```
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu
python mytensorflow_script.py
```

With the ***--nv*** option the driver libraries do not have to be installed in the container. Instead, they are located on the host system (e.g., RMACC Summit) and then bind mounted into the container at runtime. This means you can run your container on a host with one version of the NVIDIA driver, and then move the same container to another host with a different version of the NVIDIA driver and both will work. (Assuming the CUDA version installed in your container is compatible with both drivers.)

The present NVIDIA Driver on RMACC Summit is 390.48, which is compatible with CUDA 9.1 or lower.

This means you can build a container by bootstrapping a base NVIDIA CUDA image (9.1 or lower) from: https://hub.docker.com/r/nvidia/cuda/

Source and additional context: http://singularity.lbl.gov/docs-exec#a-gpu-example

/\/\ BREAK /\/\

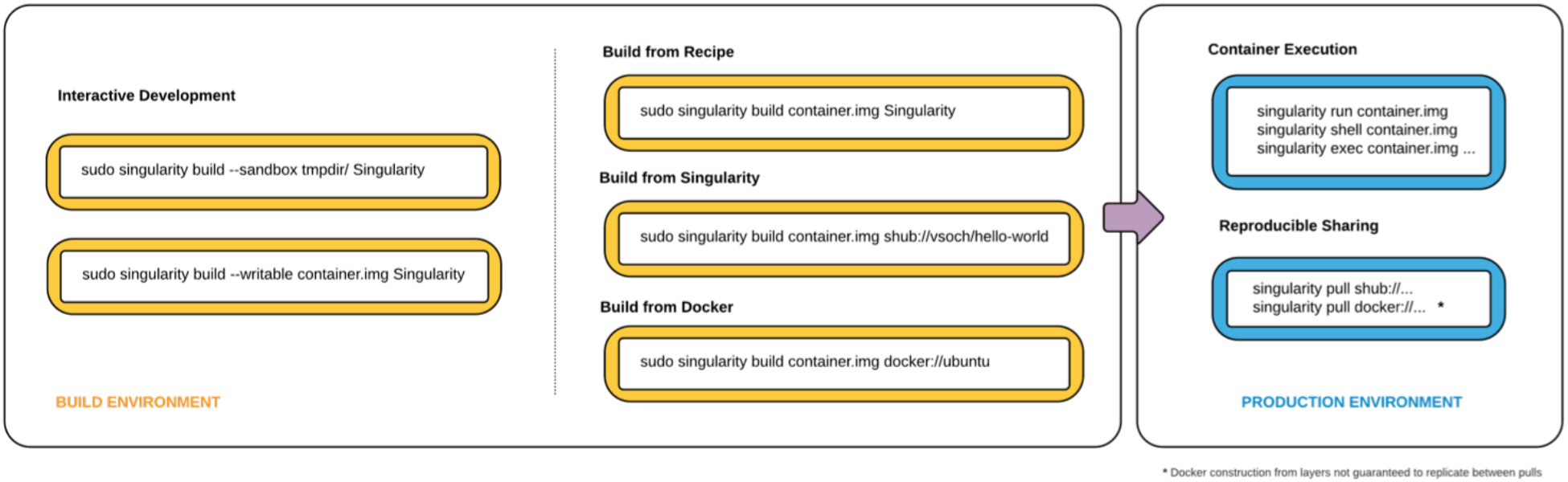
Task for next session: Create a github a account if you don't already have one.

https://github.com/join

{Remember your username and password!}

Review

The Singularity workflow



Second half of tutorial

First half of tutorial

https://singularity.lbl.gov

Recall: Container Execution

On ***any system*** with Singularity, even without administrative privilege, you can retrieve and use containers:

- Download a container from Singularity Hub or Docker Hub
  - `singularity pull shub://some_image`
  - `singularity pull docker://some_image`
- Run a container
  - `singularity run mycont.img`
- Execute a specific program within a container
  - `singularity exec mycont.img python myscript.py`
- "Shell" into a container to use or look around
  - `singularity shell mycont.img`
- Inspect an image
  - `singularity inspect --runscript mycont.img`

Building containers

Seacontainersales.com

There are 2 basic ways to build a container

1. Build a container on a system on which you have administrative privilege (e.g., your laptop).
    - **Pros:** You can interactively develop the container.
    - **Cons:** Requires many GB of disk space, requires administrative privilege, must keep software up-to-date, container transfer speeds can be slow depending on personal network connection.

2. Build a container on Singularity Hub
    - **Pros:** Essentially zero disk space required on your system, doesn't require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, typically faster transfers from Singularity Hub to desired endpoint.
    - **Cons:** Cannot interactively develop the container

1. Building a container on your machine (interactive):

Within your build environment (a laptop, workstation, or a server that you control): develop and test ***writable*** containers in ext3 format:

- Build from an existing recipe
  - `sudo singularity build --writable tmp.img mycont.def`
- Build from an existing container on Singularity Hub
  - `sudo singularity build --writeable mycont.img shub://somewhere`
- Build from an existing container on Docker Hub
  - `sudo singularity build --writable mycont.img docker://somewhere`
- Can also build a writable directory ("sandbox") instead of image, e.g.:
  - `sudo singularity build --sandbox tmpdir/ mycont.def`

What is a recipe?

Header

Metadata

Runtime environment variables

Default program at runtime

Where software and directories are installed at buildtime

```
Bootstrap:docker
From:ubuntu:latest

%labels
MAINTAINER Andy M

%environment
HELLO_BASE=/code
export HELLO_BASE

%runscript
echo "This is run when you run the image!"
exec /bin/bash /code/hello.sh "$@"

%post
echo "This section is performed after you bootstrap to build the image."
mkdir -p /code
apt-get install -y vim
echo "echo Hello World" >> /code/hello.sh
chmod u+x /code/hello.sh
```

See http://singularity.lbl.gov/docs-recipes for more.

Basic process of building a container interactively

- Bootstrap a base container (has OS you want, maybe other stuff too)
- Shell into the container
  - Install additional needed programs
    - If they have dependencies, install the dependencies – google for the OS provided packages first and install with apt-get/yum if possible
  - Put the commands in the %post scriptlet of the recipe.
- Build the container again
  - Now with the additional commands in the recipe
  - If something fails, fix it, build container again
- Iterate until all needed programs are installed

When you are done building interactively…

After interactive development, within your build environment (a laptop, workstation, or a server that you control): build your production containers with a *squashfs* filesystem…

- Build from a "recipe" that you developed.
  - `sudo singularity build mycont.img mycont.def`
- Build from an existing container on Singularity Hub
  - `sudo singularity build mycont.img shub://somewhere`
- Build from an existing container on Docker Hub
  - `sudo singularity build mycont.img docker://somewhere`

2. Building a container on Singularity Hub (basic steps)

1. Create a recipe file for your container
2. Name it “Singularity”
3. Create a github repository for your container
4. Upload it to your github repository
5. Log into Singularity Hub using your github username/password
6. Go to “My Collections” and choose “ADD A COLLECTION”
7. Select the github repository you just uploaded.
8. The container will build automatically.
9. Revised recipes are automatically rebuilt when pushed to github.
10. Additional details at:
    https://github.com/singularityhub/singularityhub.github.io/wiki

Hands-on Example

- Build a container on Singularity Hub

Logging in

If you are using a temporary account: In a terminal or Git Bash window, type the following:

```
ssh -X userNNNN@tutorial-login.rc.colorado.edu
```

```
Password:
```

***Note that `userNNNN` is a temporary account that I have assigned to you. If you don't have one, let me know.*

If you are using your regular RC account (e.g., 'monaghaa' for me), log in as you normally would.

Getting on a compute node

Navigate to scompile

```
ssh scompile
```

Now start an interactive job:

```
sinteractive –ntasks=1 --reservation=container
```

And load singularity:

```
module load singularity
```

…We will go through the example together in the following slides.

Go to the example directory

Go to the Singularity Hub example directory

```
cd /projects/$USER/rmacc_2018_container_tutorial/build_container_on_shub/
```

List the contents of the directory and see a description of each file:

```
ls
cat README.md
```

If you get behind or have issues, look in '`tutorial_steps.txt`'

Prepare and upload your recipe

Use a text editor to explore and customize the recipe file '`Singularity`':

```
nano Singularity
```

- Can you determine the purpose of each section?
- Can you determine what this container does?
- Are there any files copied to the container?
- Now change the 'Maintainer' to your name
- To exit and save type [ctrl-x], then "y", then [enter].

Now we are ready to build a container on Singularity Hub using this recipe. Recall that we do this by creating a github repository containing the 'Singularity' recipe file. For the sake of time we've created a script to facilitate this step. Type:

```
./make_git_repo.sh <your-github-username>
```

You will be prompted for your github password 2 times while the script is running; enter it each time. User your browser to confirm your repository was created:

```
https://github.com/<your-github-username>/build_container_on_shub
```

Build your container on 'shub'

Now navigate to Singularity Hub in your browser:

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

…and do the following:

1. Go to the "login" link in the upper right corner and login with your github credentials
2. Choose "GITHUB", not "GITHUB (WITH PRIVATE)"
3. Go to "My container collections"
4. Go to "Add a Collection"
5. Choose "<your-github-username>/build_container_on_shub" and click on "SUBMIT"
6. This will take you another page while your container builds. You can click "refresh" to check it's status. It may take several minutes.
7. If your container builds successfully, you will see a green 'Complete' button. If not, let us know and we'll help: you can quickly fix and re-commit the recipe to github, which will initiate another build on Singularity Hub.

Now pull your container from 'shub' and run it!

Now navigate to Singularity Hub in your browser:

```
singularity pull --name mytranslator.img shub://<your-github-username>/build_container_on_shub
```

Run your container with the 'inside' version of the python script:

```
singularity run mytranslator.img
```

…and run your container with the 'outside' version of the python script:

```
singularity exec mytranslator.img python ./text_translate.py
```

(hint: you can change the language in ./text_translate.py to confirm that the outside script is running)

Troubleshooting & Caveats

Troubleshooting (1)

Container/host environment conflicts

- Container problems are often linked with how the container "sees" the host system. Common issues:
  - The container doesn't have a bind point to a directory you need to read from / write to
  - The container will "see" python libraries installed in your home directory (and perhaps the same is true for R and other packages. If this happens, set the PYTHONPATH environment variable in your job script so that it points to the container paths first.
    - `export PYTHONPATH=<path-to-container-libs>:$PYTHONPATH`

- To diagnose the issues noted above, as well as others, "shelling in" to the container is a great way to see what's going on inside. Also, look in the singularity.conf file for system settings (can't modify).

Troubleshooting (2)

Failures during container pulls that are attributed (in the error messages) to *tar.gz files are often due to corrupt tar.gz files that are downloaded while the image is being built from layers. Removing the offending tar.gz file will often solve the problem.

When building ubuntu containers, failures during *%post* stage of container builds from a recipe file can often be remedied by starting the *%post* section with the command "`apt-get update`". As a best practice, make sure you insert this line at the beginning of the *%post* section in all recipe files for ubuntu containers.

https://singularity.lbl.gov

Caveats (1)

We didn't cover overlays. These are additional images that are "laid" on top of existing images, enabling the user to modify a container environment without modifying the actual container. Useful because:

1. Overlay images enable users to modify a container environment even if they don't have root access (though changes disappear after session)
2. Root users can permanently modify overlay images without modifying the underlying image.
3. Overlays are a likely way to customize images for different HPC environments without changing the underlying images.
4. More on overlays: http://singularity.lbl.gov/docs-overlay

We didn't cover GPU usage with containers. See:
http://singularity.lbl.gov/archive/docs/v2-3/tutorial-gpu-drivers-open-mpi-mtls

Caveats (2): Moving containers

You've built your first container on your laptop. It is 3 Gigabytes. Now you want to move it to Summit to take advantage of the HPC resources. What's the best way?

Remember, containers are files, so you can transfer them to Summit just as you would a file:

- Command line utilities (scp, sftp)
- Globus (recommended)
- For more on data transfers to/from Summit: https://github.com/ResearchComputing/Research-Computing-User-Tutorials/wiki/Data-Transfers

Thank you!

Please fill out the survey:

????

Additional learning resources:

Slides and Examples from this course:

https://github.com/rctraining/rmacc_2018_container_tutorial

Web resources:

https://www.chpc.utah.edu/documentation/software/containers.php

https://singularity.lbl.gov/user-guide *(user guide for Singularity)*

https://www.singularity-hub.org/ *(Singularity Hub)*

Extra slides

On your personal machine: How to start your Linux VM so that you can run Singularity

Once you've installed Singularity on your machine (assuming Mac or Windows):

```
$ mkdir singularity-vm
$ cd singularity-vm
$ vagrant init singularityware/singularity-2.5.2
$ vagrant up
$ vagrant ssh
```

If you are on Linux, you don't need to start a VM