Using Containers on Teton

Andrew Monaghan
Andrew.Monaghan@colorado.edu

Slides and exercises available for download at
https://github.com/researchcomputing/uwyo_2019

Outline

- Introduction to containers
- Overview: Singularity commands and running containers
- Hands-on: Running containers on Teton
- Notes: Running containers for MPI and GPU jobs

/\/\ Break /\/\

- Overview: Building containers
- Hands-on: Building containers from recipes (SingularityHub)
- Demo: Building containers from recipes (Sylabs Remote Bldr)
- Troubleshooting, caveats and summary

Introduction to Containers

What is a container?

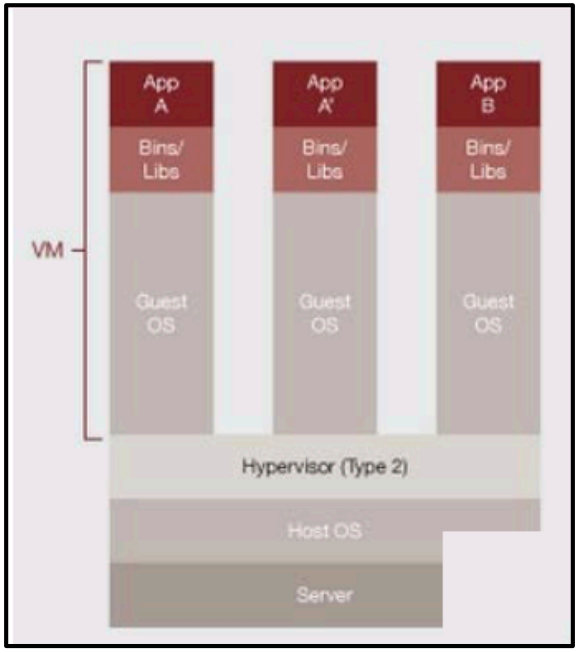
A container is a portable environment that packages some or all of the following: an operating system, software, libraries, compilers, data and workflows. Containers enable:

- 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



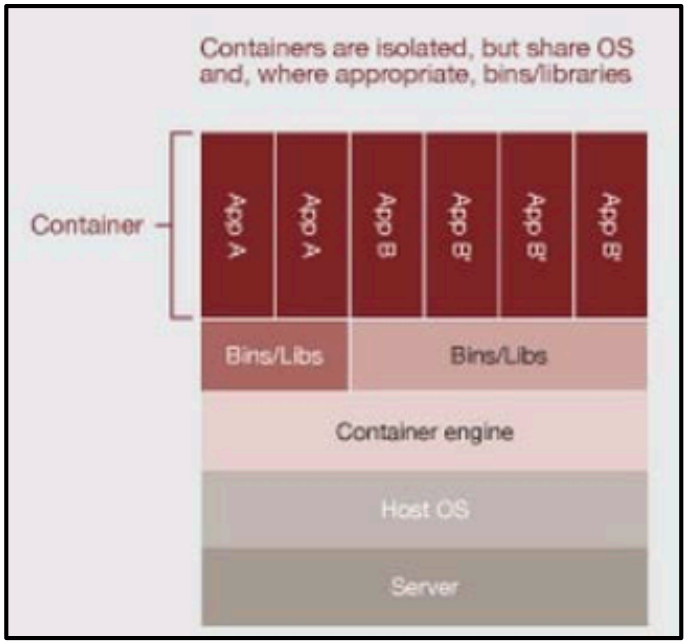
Material courtesy: M. Cuma, U. Utah

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

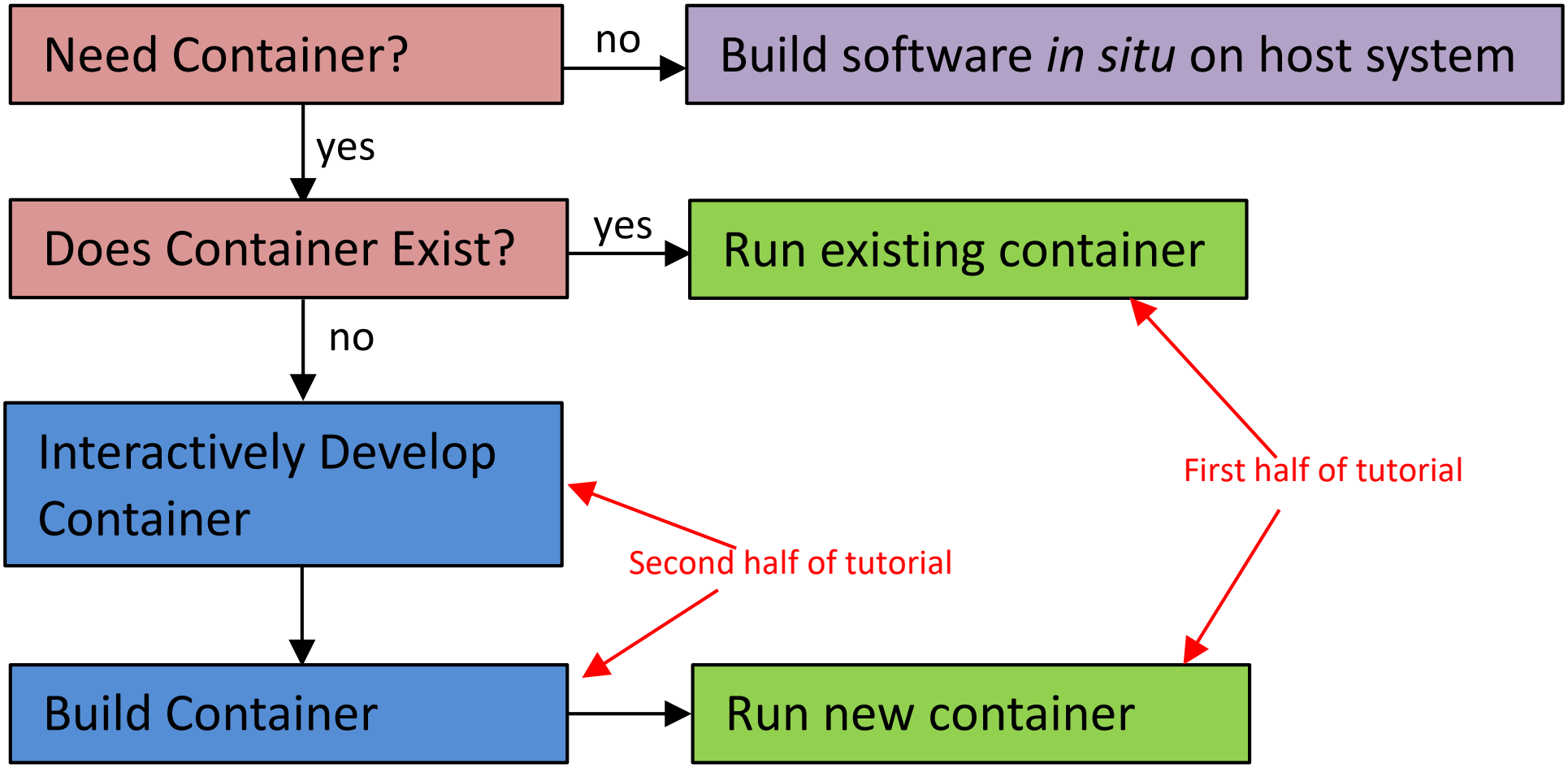
Singularity Overview

Singularity Workflow

Key Commands

Running Containers

The Singularity Workflow



Where do I find existing containers?

Docker Hub: https://hub.docker.com

Singularity Hub: https://singularity-hub.org

Sylabs Library: https://cloud.sylabs.io/library

Tutorial on finding a running an existing Docker container:
https://www.chpc.utah.edu/documentation/software/singularity.php#exd

Key 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

More: https://www.sylabs.io/guides/3.2/user-guide/cli.html

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_repo/some_image`
  - `singularity pull library://some_repo/some_image`
  - `singularity pull docker://some_repo/some_image`
- Run a container
  - `singularity run mycont.sif`
- Execute a specific program within a container
  - `singularity exec mycont.sif python myscript.py`
- "Shell" into a container to use or look around
  - `singularity shell mycont.sif`
- Inspect an image
  - `singularity inspect --runscript mycont.sif`

Hands-on Examples

- Running containers with Singularity

Logging in

If you are using a temporary account:

1. Note the username on the side of your Yubikey ("`arcc-tNN`")
2. Insert your Yubikey into the usb port on your machine
3. Open MobaXterm
4. Go to "**Session**"→"**SSH**"
5. Under "**Remote host**" type `teton.uwyo.edu`
6. Under "**Specify username**" type `arcc-tNN` ("`NN`" is your #)
7. Click "**OK**"
8. Enter your provided password at the prompt (don't save)
9. Yubikey will provide 2-factor authentication

If you are using your regular account, just log in as you normally would.

Getting on a compute node

Start an interactive job:

```
salloc -A arccanetrain -t 150 -n 2
```

Load singularity and set up environment

```
module load singularity/3.1.1
export SINGULARITY_TMPDIR=/gscratch/$USER
```

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

Navigating

Now type the following commands:

```
cd /gscratch/$USER

git clone https://github.com/researchcomputing/uwyo_2019

cd uwyo_2019

ls
```

You should see some directories. We will use these for the hands-on examples.

Go to the example directory

Now go to the Singularity example directory

```
cd run_singularity_basic
```

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

```
ls
```

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

```
cat tutorial_steps.txt
```

Running a container from Singularity Hub

Pull an existing container image that someone else posted:

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

…And run it

```
singularity run hello.sif
```

…And look at the script inside

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

Running a container from Docker Hub

Let's grab the stock docker python container:

```
singularity pull --name pythond.sif docker://tatsushid/tinycore-python
```

…And run python

```
singularity exec pythond.sif python
```

…And shell into the container and look around

```
singularity shell pythond.sif
```

…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.sif 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.

Running an MPI container

MPI-enabled Singularity containers can be deployed on Teton, with the caveat that the MPI software within the container must be consistent with MPI software available on the system. This requirement diminishes the portability of MPI-enabled containers, as they may not run on other systems without compatible MPI software. Regardless, MPI-enabled containers can still be a very useful option in many cases.

Here we provide an example of that uses a gcc compiler with OpenMPI. Let's pull it from Dockerhub first:

```
singularity pull hello_openmpi.sif shub://monaghaa/hello_openmpi_teton
```

In order to use it, we load the gcc and openmpi modules on Teton (these are consistent with the gcc/openmpi versions installed in the container)

```
module load gcc/7.3.0
module load swset/2018.05
module load openmpi/3.1.0
```

To run it, simply preface the 'singularity exec <stuff>' command with 'mpirun –n <numprocs>':

```
mpirun -n 2 singularity exec hello_openmpi.sif mpi_hello_world
```

Binding directories

Depending on how Singularity is configured on a given host system, host directories may be “bound” (mounted) by default. But sometimes you may want to access a directory that is not already mounted. Here’s how you can do it:

```
singularity shell -bind /outsidedir:/insidedir pythond.sif
```

…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)

Notes: MPI and GPUs

Notes: MPI-enabled containers

Teton uses an the Infiniband low-latency interconnect (“fabric”) to enable MPI to be efficiently implemented across nodes). In order to use a Singularity container with OpenMPI (or any MPI) on Teton, there are two requirements:

1. The Singularity container needs to have the Infiniband libraries installed inside.
2. OpenMPI needs to be installed both inside and outside of the Singularity container. More specifically, the SAME version of OpenMPI needs to be installed inside and outside (at least very similar, you can sometimes have two different minor versions, ex: 2.1 and 2.0).

An example Singularity recipe in the tutorial Github repository named “Singularity_openmpi.def” can be used as a template to build MPI-enabled container images for Teton. It has the Infiniband libraries and ensures that OpenMPI 3.1.0 is installed in the image, which matches the openmpi/3.1.0 module available on Teton.

Running containers on GPUs

Syntax (after loading Singularity on a gpu node):

```
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., Teton) 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 NVIDIA Driver version on a GPU node can be queried (when on that node) with the command `nvidia-smi.` See *https://docs.nvidia.com/deploy/cuda-compatibility/*

You can build a container by bootstrapping a base NVIDIA CUDA image (with a compatible CUDA version) from: *https://hub.docker.com/r/nvidia/cuda/*

NVIDIA has a tool for building gpu-enabled containers for both Singularity and Docker. See: *https://github.com/NVIDIA/hpc-container-maker*

/\/\ 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

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_repo/some_image`
  - `Singularity pull library://some_repo/some_image`
  - `singularity pull docker://some_repo/some_image`
- Run a container
  - `singularity run mycont.sif`
- Execute a specific program within a container
  - `singularity exec mycont.sif python myscript.py`
- "Shell" into a container to use or look around
  - `singularity shell mycont.sif`
- Inspect an image
  - `singularity inspect --runscript mycont.sif`

Building containers

There are 3 ways to build a Singularity 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 using recipe, Github
    - **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

3. Build a container on Sylabs remote builder
    - **Pros:** Essentially zero disk space required on your system, doesn't require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, final container is placed on local machine
    - **Cons:** Cannot interactively develop the container, 'Freemium' version limited

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
```

More: https://www.sylabs.io/guides/3.2/user-guide/definition_files.html

Container Formats

*.sif format and older *.simg format
(immutable final container format)

- **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

Writeable sandbox used for interactive container development

https://singularity.lbl.gov

1. Building a container interactively

- Bootstrap a base container (has OS you want, maybe other stuff too) into a sandbox:

```
sudo singularity build --sandbox mytest/ docker://alpine:latest
```

- Shell into the container and install what you need by trial and error:

```
sudo singularity shell --writable mytest/
```

  - [now do stuff in container; as you get it correct, add commands to Singularity recipe]

- Now finalize container.
  - You can either build squashfs image from sandbox:

```
sudo singularity build mytest.sif mytest/
```

  - Or you can build squashfs image from recipe (best practice):

```
sudo singularity build mytest.sif Singularity
```

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

***Note: You can build Docker containers on Docker Hub using a nearly identical process!*

Hands-on Example

- Build a container on Singularity Hub

Logging in

If you are using a temporary account:

1. Note the username on the side of your Yubikey ("`arcc-tNN`")
2. Insert your Yubikey into the usb port on your machine
3. Open MobaXterm
4. Go to "**Session**"→"**SSH**"
5. Under "**Remote host**" type `teton.uwyo.edu`
6. Under "**Specify username**" type `arcc-tNN` ("`NN`" is your #)
7. Click "**OK**"
8. Enter your provided password at the prompt (don't save)
9. Yubikey will provide 2FA

If you are using your regular account, just log in as you normally would.

Getting on a compute node

Start an interactive job:

```
salloc -A arccanetrain -t 150 -n 2
```

Load singularity and set up environment

```
module load singularity/3.1.1
export SINGULARITY_TMPDIR=/gscratch/$USER
```

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

Go to the example directory

Go to the Singularity Hub example directory

```
cd /gscratch/$USER/uwyo_2019/build_container_on_shub/
```

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

```
ls
```

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

```
cat 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:

```
chmod +x make_git_repo.sh
./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.sif shub://<your-github-username>/build_container_on_shub
```

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

```
singularity run mytranslator.sif
```

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

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

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

3. Building a container with Sylabs Remote Builder (demo)

1. Get a token from Sylabs Cloud: https://cloud.sylabs.io/auth
    1. Login using your Github account
    2. Provide a label under “Create a New Access Token”
    3. Click “Create New Token”
    4. Copy the token string (<your-token>)

2. Add the token on your host machine:
    1. `mkdir ~/.singularity`
    2. `echo “<your-token>” > ~/.singularity/sylabs-token`

3. Issue the command to build your container remotely
    1. `singularity build --remote myimage.sif myrecipe.def`

4. If successful, the container will be placed in your working directory

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.

”Pulling” containers onto Teton can require substantial disk space and may fill up the /tmp directory with temporary files. Setting an alternate temporary directory as follows can alleviate this issue:

```
export SINGULARITY_TMPDIR=/gscratch/$USER
```

When building 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`” or “`yum -y update`” (depending on which Linux version you install). As a best practice, make sure you insert this line at the beginning of the *%post* section in all recipe files for ubuntu containers.

Caveats (1)

We didn't cover the following topics

- Container security and verification
- Searching for containers
- Overlays
- Running services with containers ("instances")
- Setting environment variables at runtime
- Installing Singularity on your laptop or desktop machine
- Best practices https://github.com/ResearchComputing/container-best-practices-rmacc19

For information on these and other topics:
https://www.sylabs.io/docs/

Caveats (2): Moving containers

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

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

- Command line utilities (scp, sftp)
- Globus
- For more on data transfers to/from Teton:
  https://arccwiki.uwyo.edu/index.php/Data_Transfer

Thank you!

Please fill out the survey:
http://tinyurl.com/curc-survey18

Additional learning resources:

Slides and Examples from this course:
https://github.com/researchcomputing/uwyo_2019

Web resources:
https://curc.readthedocs.io/en/latest/software/ContainerizationonSummit.html
https://www.chpc.utah.edu/documentation/software/containers.php
https://www.sylabs.io/docs/ *(documentation for Singularity)*
https://www.singularity-hub.org/ *(Singularity Hub)*
https://cloud.sylabs.io/library *(Sylabs Library)*