Introduction to Containers

Martin Čuma
Center for High Performance Computing
University of Utah
m.cuma@utah.edu

Overview

- Why do we want to use containers?
- Containers basics
- Run a pre-made container
- Build and deploy a container
- Containers for complex software
- https://www.surveymonkey.com/r/RDMBHMS

Hands on setup

1. Download the talk slides

   http://home.chpc.utah.edu/~mcuma/chpc/Containers18f.pdf

   https://tinyurl.com/yc6uyzf9

2. Using FastX or Putty, ssh to any CHPC Linux machine, e.g.

   ```
   $ ssh uxxxxxx@frisco.chpc.utah.edu
   ```

3. Load the Singularity and udocker modules

   ```
   $ module load singularity udocker
   ```

4. Clone the tutorial files and cd to its directory (courtesy Andy Monaghan, CU Boulder)

   ```
   $ git clone https://github.com/rctraining/rmacc_2018_container_tutorial
   $ cd rmacc_2018_container_tutorial
   $ ls
   ```

Hands on setup for building containers

1. Create a Github account if you don't have one
   https://github.com/join
   {Remember your username and password!}

OR

1. If you have CHPC account, using terminal application (Mac terminal, PuTTY, GIT Shell)
   - `ssh uxxxxxx@singularity.chpc.utah.edu`
2. Make sure you can see singularity
   - `which singularity`
3. Make sure you can sudo singularity command
   - `sudo singularity –version`

Why to use containers?

Software dependencies

- Some programs require complex software environments
  - OS type and versions
  - Drivers
  - Compiler type and versions
  - Software dependencies
    - Python/R/MATLAB versions
    - glibc, stdlibc++ versions
    - Other libraries and executables
    - Python/R libraries

Reproducible research

- Research outputs include software and data
- Software reproducibility
  - Software repositories (svn, git)
  - Good but often software has dependencies
- Data reproducibility
  - Data as publication supplementary info, centralized repositories (NCBI), …
  - Disconnected from the production environment
- Package data AND code AND compute environment in one file

Scalable research

- Develop a program / pipeline locally, run globally
- Scale to parallel resources
  - Run many times
  - Use local or national HPC resources
- Automate the process
  - Container/software building and deployment
  - Parallel pipeline

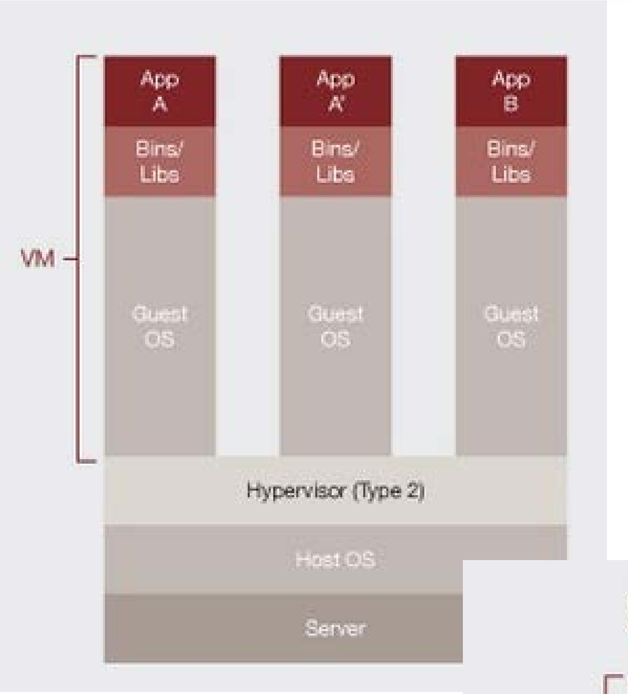
Additional bonus

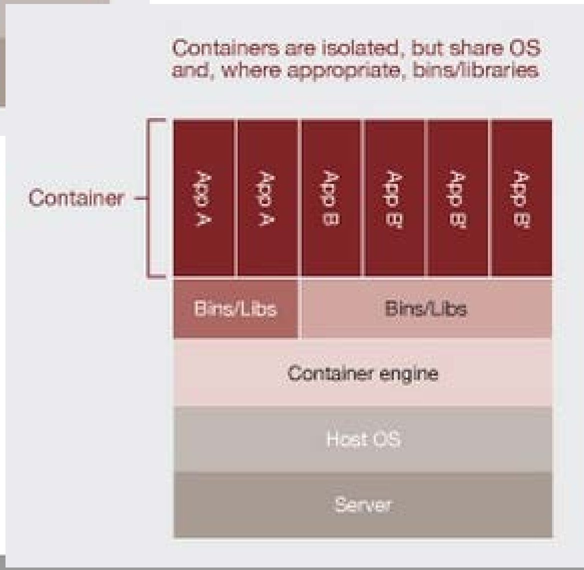
- Old applications built on old Linux versions can run on newer Linux host
- May be able to run Windows programs on Linux

Container basics

Virtualization basics

- Hardware virtualization
  - Running multiple OSes on the same hardware
  - VMWare, VirtualBox
- OS level virtualization
  - run isolated OS instance (guest) under a server OS (host)
  - Also called containers; user defined software stack (UDSS)
  - Docker, Singularity





Containers

- Isolate computing environments
  - And allow for regenerating computing environments
- Guest OS running over host OS
  - Guest’s OS can be different that host’s
  - Low level operations (kernel, network, I/O) run through the host
- From user standpoint guest OS behaves like standard OS

Container solutions

- Docker
  - Well established
  - Has docker hub for container sharing
  - Problematic with HPC
- Singularity
  - Designed for HPC, user friendly
  - Support for MPI, GPUs
- Charliecloud; Shifter, udocker
  - Also HPC designed, built on top of Docker
  - Simple but less user friendly

Singularity containers

- Integrate with traditional HPC
  - Same user inside and outside of the container
  - Same file systems (home, scratch), environment
  - Can integrate with existing software (CHPC sys branch)
- Portable and sharable
  - A container is a file
  - It can be built on one OS and run on another
- Only Linux support right now
- Not completely secure due to use of setUID executables
  - Hacker can exploit potential flaws in setUID programs to gain root
  - http://singularity.lbl.gov/docs-security

An aside into security

- Containers need privilege escalation to run
  - Give sudo
  - Run root owned daemon process (Docker)
  - Use setUID programs (programs which parts can run in privileged mode) (Singularity now, udocker)
  - User namespaces – new Linux kernel feature to further isolate users (Charliecloud)
  - Linux capability set – fine grained privilege isolation (Singularity future)
- In HPC environment
  - setUID if you have some trust in your users, user namepaces if you don't (and have newer Linux distribution – e.g. CentOS >= 7.4)

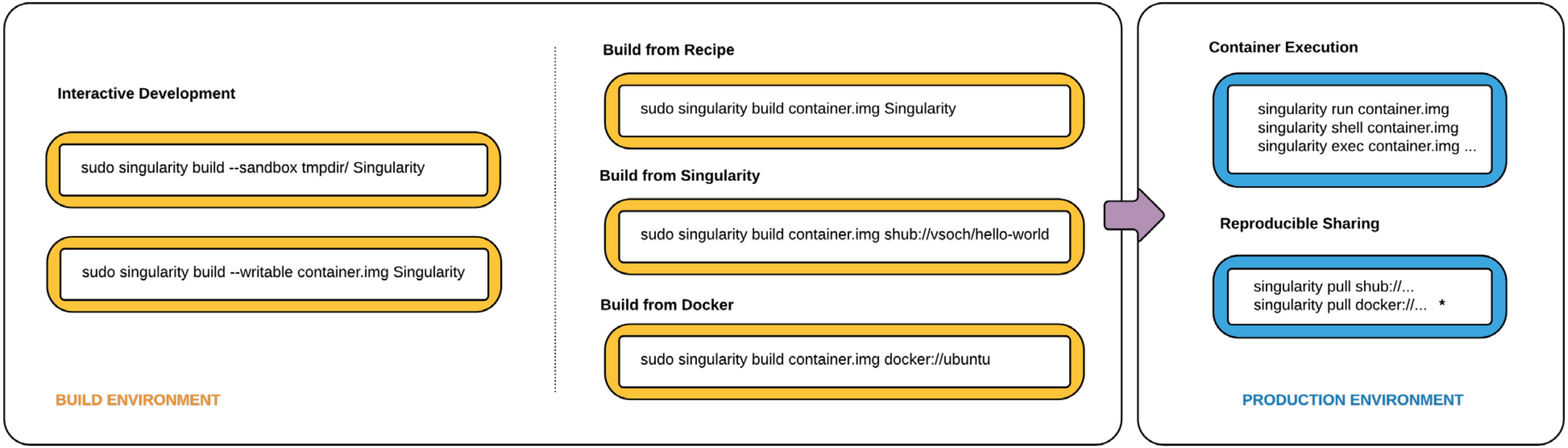
Charliecloud containers

- Uses user namespaces for isolation
  - More secure
  - Limited to CentOS 7 and other recent Linux distributions (not supported in older CentOS or other Linux releases)
- Uses Docker containers
  - Needs Docker to build containers
  - Extracts and repackages Docker containers
- Singularity has an –userns option to force User namespace
  - But capabilities limited to directory (sandbox) based containers

udocker

- a lightweight runtime for Docker containers
- No actual root escalation
  - Only does chroot to mount base OS to a different path
  - Uses different mechanisms to achieve this
  - The default, PRoot, uses ptrace system call
- A good choice to run Dockerhub containers
- Allows to change container files without privileges

Singularity workflow



* Docker construction from layers not guaranteed to replicate between pulls

Run a pre-made container

A few pre-requisites

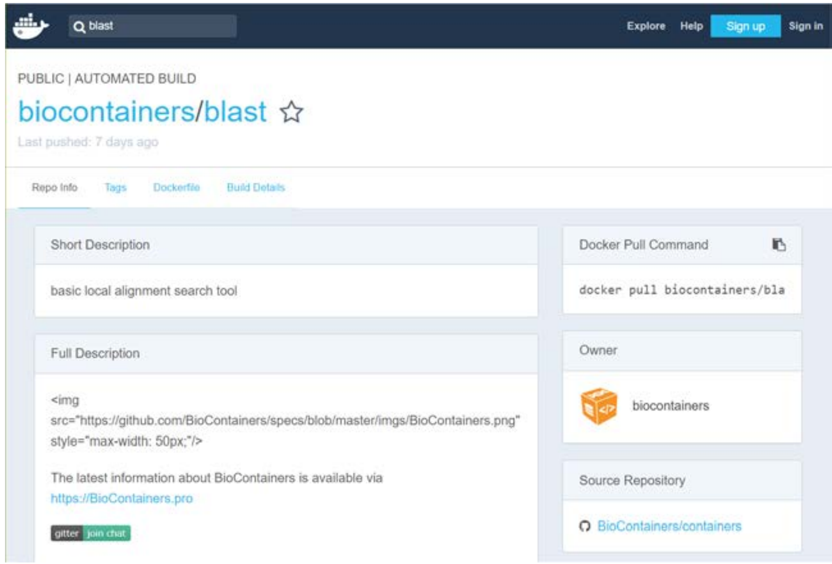
- Building a container requires a root, or sudo
  - You can do that on your own machine
  - You can't do that at CHPC clusters
  - > build your containers locally, or on container hubs
- You can run a container as an user
  - You can run your own containers at CHPC
  - You can run CHPC provided containers at CHPC
  - You can run containers obtained on the internet at CHPC

Basic principles

- Containers are run in user space (no root required)
- An appropriate environment module has to be loaded
- User inside of the container
  - Current user
  - If specified, other user, including root
  - Root inside container is contained = can't be root outside
- Containers can't be modified by non-root user
  - One small exception with udocker

Container registries and repositories

- Most containers reside in registries
  - Content delivery and storage systems for container images
- Docker Hub is the most common registry
  - https://hub.docker.com
  - Contains many channels, e.g. Biocontainers (http://biocontainers.pro/)
- There are a few other registries
- Singularity also has a hub
  - We'll use it to build a container



How to find a container

- Google or hub.docker.com search
  - E.g. “blast docker”
- Use udocker or singularity search

```
$ ml udocker ; ml singularity
$ udocker search blast
NAME                                      OFFICIAL DESCRIPTION
biocontainers/blast                           ---- basic local alignment search tool
comics/blast                                  ---- blast
simonalpha/ncbi-blast-docker                  ----
```

Run container with udocker

- Will pull the container to user directory (default ~/.udocker)

```
$ udocker run --user=u0101881 biocontainers/blast blastp -help
Warning: non-existing user will be created
************************************************************
*          STARTING 62b84d21-9c78-3b89-aa68-01d24520ff62
*
************************************************************
 executing: blastp
USAGE
  blastp [-h] [-help] [-import_search_strategy filename]
```

Run container with Singularity

- To get a shell in a container – `singularity shell`

```
$ singularity shell docker://ubuntu:latest
```

- To run a command in a container – `singularity exec`

```
$ singularity exec docker://ubuntu:latest cat /etc/os-release
```

- To run default command in a container – `singularity run`

```
$ singularity run mycontainer.sif
$ ./mycontainer.sif
```

- To inspect a container – `singularity inspect`

```
$ singularity inspect --runscript docker://python
```

Run vs. pull

- pull will only download the container
- run/exec will download if needed and run

```
$ udocker pull biocontainers/blast
```

  – Downloads container to local repository (default ~/.udocker)

```
$ udocker run biocontainers/blast blastp –help
```

  – Runs the local container

```
$ singularity pull docker://ubuntu:latest
$ ls
$ singularity exec ubuntu_latest.sif /bin/bash
$ singularity shell ubuntu_latest.sif
```

Files in and out of the container

- Let’s create a Python script

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

- Now shell into a stock Python container and ls to look around

```
singularity shell docker://python
```

- Now exit the container, and run

```
singularity exec docker://python 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.

Bind mounting file systems

- Needed for group and scratch file systems
- Container has to recognize `/uufs` and `/scratch`
  - Singularity now does this automatically (as of 3.0.0)
  - udocker needs to explicitly list this

```
udocker run --user=u0101881 --bindhome --volume=/scratch
--volume=/uufs ubuntu:latest /bin/bash
```

Modify and use container – only with udocker

- cd to a directory with udocker example

```
$ cd run_container_w_udocker
```

- 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
```

- 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
```

Modify and use container – only with udocker cont

- …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 (= use FastX)

```
gs Rplots.pdf
```

Importing Docker container in the PE

- Udocker – do the same thing as on general clusters

```
udocker run --user=u0101881 --bindhome --volume=/scratch
--volume=/uufs ubuntu:latest /bin/bash
```

- Singularity - `--userns` must use expanded file system
  - On a machine with sudo singularity (singularity.chpc.utah.edu)

```
$ singularity pull docker://ubuntu:latest
$ sudo singularity image.export ubuntu-latest.img | gzip -9 > ubuntu-
latest.tar.gz
```

  - ssh to the PE cluster, get, unpack archive and run

```
$ ml singularity
$ tar xfz ubuntu-latest.tar.gz –C /scratch/local/ubuntu-latest
$ mkdir /scratch/local/ubuntu-latest/uufs
$ mkdir /scratch/local/ubuntu-latest/scratch
$ singularity shell --userns /scratch/local/ubuntu-latest
```

Some useful tips

- Binding mount points (singularity < 3.0)

```
$ export SINGULARITY_BINDPATH="/scratch,/uufs/chpc.utah.edu"
$ singularity shell -B /scratch,/uufs/chpc.utah.edu
ubuntu_python.img
```

- Specifying shell

```
$ export SINGULARITY_SHELL=/bin/bash
$ singularity shell –s /bin/bash ubuntu_python.img
```

- More specialized topics – ask us
  - Using environment modules from the host
  - Using GPUs, MPI over InfiniBand

Using GPUs

- Need to bring in the Nvidia driver stack
  - Pre Singularity 2.3, udocker – explicitly install – make sure to have the same driver version on the host and in the container
  - Singularity 2.3+ --nv runtime flag brings the drivers from the host
    - Still need to have a compatible CUDA installed in the container (older than or the same as the driver)
    - The present NVIDIA Driver at CHPC 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/

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

Using MPI and InfiniBand

- Need to bring the IB stack in the container
  - Some people bring the needed IB libraries from the host
  - For Ubuntu we prefer to install the Ubuntu stack
  - https://github.com/CHPC-UofU/Singularity-ubuntu-mpi
- MPI
  - Build inside the container with IB, or use CHPC's modules
  - Prefer MPICH and derivatives, OpenMPI is very picky with versions
  - If using OS stock MPI, then make sure to LD_PRELOAD or LD_LIBRARY_PATH ABI compatible libmpi.so with InfiniBand
  - https://github.com/CHPC-UofU/Singularity-meep-mpi

Using Lmod from containers

- Many Linux programs are binary compatible between distros
  - Most installed binaries are (Intel, PGI tools, DDT, …)
- No need to install these in the container – use our NFS mounted software stack through Lmod
  - Need to have separate Lmod installation for Ubuntu due to some files having different location
- In the container
  - Install Lmod dependencies
  - Modify /etc/bash.bashrc to source our Lmod

https://github.com/CHPC-UofU/Singularity-ubuntu-python/blob/master/Singularity

Prompt change

- It can be confusing to know if one in in a container or not
  – Singularity changes prompt by default
  – Or redefine prompt in ~/.bashrc:

```
if [ -n "$SINGULARITY_CONTAINER" ] || [ -n "$CHARLIECLOUD_CONTAINER" ]; then
 if [ -x "$(command -v lsb_release)" ]; then
    OSREL=`lsb_release -i | awk '{ print $3; }'`
  else
    OSREL=`head -n 1 /etc/os-release | cut -d = -f 2 | tr -d \"`
  fi
  PS1="$OSREL[\u@\h:\W]\$ "
else
  PS1="[\u@\h:\W]\$ "
fi
```

Building Singularity containers

Hands on setup for building containers

1. Create a Github account if you don't have one
   https://github.com/join
   {Remember your username and password!}

OR

1. If you have CHPC account, using terminal application (Mac terminal, PuTTY, GIT Shell)
   - `ssh uxxxxxx@singularity.chpc.utah.edu`
2. Make sure you can see singularity
   - `which singularity`
3. Make sure you can sudo singularity command
   - `sudo singularity –version`

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
```

When to build own containers

- Complex software dependencies
  - Especially Python and R packages
    - bioBakery – intricate dependencies of Python and R which did not build on CentOS
    - SEQLinkage – instructions to build on Ubuntu using its packages
- Quick deployment
  - Some Linux distros provide program packages while others don't
    - paraview-python on Ubuntu via apt-get
- Deploying your own code or pipeline

Container build strategy

- Start with a the basic container (e.g. ubuntu:latest from Docker)
- 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
- Build the container again
  - Now with the additional commands in the `%post`
  - If something fails, fix it, build container again
- Iterate until all needed programs are installed

Two ways of building containers

- Build a container on a system on which you have administrative privilege (e.g., your laptop, singularity.chpc.utah.edu).
  - 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.
- 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

Container build process

- Create a writeable container (the only choice in PE)

```
$ sudo singularity build --sandbox mycont.img ubuntu16.def
```

  - This creates a container directory called `mycont.img`

- If additional installation is needed after the build
  - Shell into the container and do the install manually

```
$ sudo singularity shell -w -s /bin/bash mycont.img
```

  - Execute what’s needed, modify container definition file, repeat

- When done, create a production container

```
$ sudo singularity build ubuntu16.sif ubuntu16.def
```

Container definition file (a.k.a. recipe)

- Defines how the container is bootstrapped
  - Header – defines the core OS to bootstrap
  - Sections – scriptlets that perform additional tasks
- Header
  - Docker based (faster installation)

```
BootStrap: docker
From: ubuntu:latest
```

  - Linux distro based

```
BootStrap: debootstrap
OSVersion: xenial
MirrorURL: http://us.archive.ubuntu.com/ubuntu/
```

```
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
```

Definition file sections

- `%setup` Runs on the host
  - Install host based drivers (e.g. GPU)
- `%post` Runs in the container
  - Install additional packages, configure, etc
- `%runscript` Defines what happens when container is run
  - Execution commands
- `%test` Runs tests after the container is built
  - Basic testing

Definition file sections cont'd

- `%environment` Definition of environment variables
- `%files` Files to copy into the container
- `%labels` Container metadata
- `%help` What displays during `singularity help` command

- More details at http://singularity.lbl.gov/docs-recipes

Building a container on Singularity Hub

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

Log in and set up environment

- Log to any CHPC interactive node

```
$ ssh uxxxxxxx@frisco1.chpc.utah.edu
```

- Load singularity module

```
$ ml singularity
```

- cd to the example directory and view its files

```
$ cd rmacc_2018_container_tutorial/build_container_on_shub
$ 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 <u>2 times</u> 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)", agree
  3. Go to "My container collections" (upper right user menu)
  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 '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!

- Navigate to Singularity Hub in your browser:

```
singularity pull shub://<your-github-username>/build_container_on_shub
```

- Run your container with the ‘inside’ version of the python script:

```
singularity run build_container_on_shub_latest.sif
```

- …and run your container with the ‘outside’ version of the python script:

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

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

Troubleshooting and Caveats

Host/container 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).

Pull and build errors

- 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.

Overlays

- Overlays 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:
  - Overlay images enable users to modify a container environment even if they don't have root access (though changes disappear after session)
  - Root users can permanently modify overlay images without modifying the underlying image.
  - Overlays are a likely way to customize images for different HPC environments without changing the underlying images.
  - More on overlays: https://www.sylabs.io/guides/3.0/user-guide/persistent_overlays.html

Moving containers

- You've built your first container on your laptop. It is 3 Gigabytes. Now you want to move it to CHPC to take advantage of the HPC resources. What's the best way?
- Remember, containers are files, so you can transfer them to CHPC just as you would a file:
  - Command line utilities (scp, sftp)
  - Globus or rclone (recommended) https://www.chpc.utah.edu/documentation/software/rclone.php
  - https://www.chpc.utah.edu/documentation/software/globus.php
  - For more on data transfers to/from CHPC: https://www.chpc.utah.edu/documentation/data_services.php

Containers for complex software

Example - bioBakery

- Instructions say to
  - Install VirtualBox, Vagrant, and bioBakery from an archive
    - Great for a desktop, but, not for an HPC cluster
  - Further below they mention Google Cloud
- So we download the bioBakery archive, unpack it and look inside
  - Great, there is `google_cloud/build_biobakery.sh` script
  - In that file, Ubuntu 16.04 is mentioned

Building bioBakery container

- Build base Ubuntu 16.04 container
- sudo shell into the container
  - Start executing the lines of the build_biobakery.sh script, one after another
  - Some dependencies pop up, install them
  - Another caveat – Linuxbrew requires to be installed as non-root
  - Do some web searching and figure how to add a new user and run Linuxbrew as this user
  - In the end, add the correct paths to the container environment

```
$ echo "export PATH=/usr/local/bin:$PATH” >> /environment
```

Building bioBakery container

bioBakery
A virtual environment for meta'omic analysis

- Once everything installs in the container
  - Run the bioBakery tests
  - Add %test section that run the bioBakery tests
  - Build the container again, now it will run the tests (will take a few hours)
- Create a module file or an alias to start the container
- See it all at

  https://github.com/CHPC-UofU/Singularity-bioBakery

Resources

- http://singularity.lbl.gov
- https://singularity-hub.org
- https://www.chpc.utah.edu/documentation/software/containers.php
- https://github.com/CHPC-UofU

Windows in a container?

Windows and HPC

- What, Windows?
  - There are programs that researchers use that only run on Windows
  - E.g. data processing that comes with an instrument
- Our current approach
  - Tell them to run on our only Windows server
    - Gets oversubscribed quickly
  - Build a specific VM
    - Resource intensive for us, not high performing
- What if we could run Windows programs on our Linux clusters

Wine

- Windows compatibility layer on Linux
  - https://www.winehq.org/
  - Not an emulator – translates Windows system calls to Linux, provides alternative Windows system libraries,…
  - Actively developed, under CodeWeavers company umbrella
  - Windows ABI completely in user space
  - Most Linux distros come with some version of Wine
  - Generally better to use recent Linux distros for more recent Wine version (https://www.winehq.org/download)

Winetricks

- While Wine provides the basic Windows support, Winetrics is a set of scripts that install additional Windows libraries
  - Like library dependencies in Linux
  - `winetricks list` – to list available libraries
  - Most commonly used libraries are DirectX, .NET, VB or C runtimes

Wine and Singularity

- Poached out of http://dolmades.org/
- Basic Singularity container
  - Recent Ubuntu or Fedora
  - Some winetricks work better on Fedora than Ubuntu, and vice versa
  - Include Wine repo from winehq to get the latest Wine version
  - Some experimentation is needed but if the Windows program is not complicated, chances of success are there

%post section

- Install Wine and Winetricks

```
dpkg --add-architecture i386
apt update
apt -y install wget less vim software-properties-common
python3-software-properties apt-transport-https winbind
wget https://dl.winehq.org/wine-builds/Release.key
apt-key add Release.key
apt-add-repository https://dl.winehq.org/wine-builds/ubuntu/
apt update
apt install -y winehq-stable winetricks
```

User space

- User application
  - Done in `%runscript` section
  - First container launch creates WINEPREFIX (Windows file space), then installs the needed applications, and tars the whole WINEPREFIX for future use
  - Subsequent container launch untars WINEPREFIX and launches program

%runscript section

```
TEMPDIR="$(mktemp -d)"
APPDIR="$HOME/WINE/Topofusion"
PROFILEDIR="$HOME/WINE/PROFILES/${USER}@${HOSTNAME}"
…
export WINEPREFIX="$TEMPDIR/wineprefix"
export WINEARCH="win32"

if [ -f "$APPDIR/wineprefix.tgz" ]; then
    echo "Found existing wineprefix - restoring it..."
    mkdir -p "$WINEPREFIX“; cd "$WINEPREFIX“; tar xzf "$APPDIR/wineprefix.tgz"
else
  wineboot --init
  echo "Installing TopoFusion and its dependencies ..."
  winetricks dlls directx9 vb6run
  wget http://topofusion.com/TopoFusion-Demo-Pro-5.43.exe
fi
wine ./TopoFusion-Demo-Pro-5.43.exe
```

Examples

- IDL 6.4 runtime + PeakSelector
  - IDL runtime under Linux crashes due to IDL bug
  - Windows runtime works fine, older IDL (ca. 2010)
  - https://github.com/CHPC-UofU/Singularity-ubuntu-wine-peakselector
- Topofusion
  - My favorite GPS mapping program, e.g. http://home.chpc.utah.edu/~mcuma/summer16/madison/wed/
  - Needs DirectX and VB runtime
  - https://github.com/CHPC-UofU/Singularity-ubuntu-wine-topofusion

Caveats (failed examples)

- Very new application (Win10 like)
  - Installer was not functional under Wine
- Complex scientific application
  - .NET – did not install on Ubuntu, worked on Fedora
  - Microsoft SQL did not install – show stopper
- Wine application compatibility
  - https://appdb.winehq.org/
  - Notice a lot of games

Outlook

- Success rate 1 out of 3 is not that great
  - Still worth trying, the chances are there
  - Singularity makes it easier to experiment
- It would be nice to have a HPC support for Windows so that
  - We would not need to have specialized Win machines
  - We would not have to build special purpose VMs
- May still need to look into the direction of reconfigurable HPC clusters like Bridges or Jetstream

Build and run Charliecloud containers

Container build and deployment process

- Build a Docker container using Charliecloud

```
$ sudo ch-build -t hello /path-to/dir-with-dockerfile
```

This creates Docker container layers wherever Docker stores them
(check that container exists with `sudo docker images`)

- Convert Docker container to tar file

```
$ ch-docker2tar hello ~/containers
```

This creates file `~/containers/hello.tar.gz`

- Copy container, unpack and run

```
$ scp /var/tmp/hello.tar.gz myhost:~/containers
$ ch-tar2dir ~/containers/hello.tar.gz /scratch/local
$ ch-run -b /uufs:/uufs -b /scratch:/scratch
/scratch/local/hello -- bash
```

Charliecloud/Docker container at CHPC

- Use standard Dockerfiles

```
FROM ubuntu1604
RUN    apt-get update
...
```

- Create mount points for CHPC file systems

```
RUN mkdir /uufs /scratch
```

- To bring in CHPC modules (and InfiniBand) see

https://github.com/CHPC-UofU/Charliecloud/blob/master/ubuntu1604openmpi3/Dockerfile.ubuntu1604openmpi3

Thanks to Andy Monaghan from CU Boulder for making the tutorial

Questions?

Survey

https://www.surveymonkey.com/r/RDMBHMS

Prepare your computer for Singularity containers

Linux

- We need to run Linux to build/run Singularity
  - If you already run Linux, make sure you have a root
  - On Windows and Mac, we need to install Linux first
- Install Linux in a VM
  - Windows – GIT Bash, Virtual Box and Vagrant
    - http://singularity.lbl.gov/install-windows
  - Mac – Homebrew with Virtual Box and Vagrant
    - http://singularity.lbl.gov/install-mac

Windows/Mac Install prerequisites

- Windows – GIT Bash, VirtualBox, Vagrant
  - GIT Bash provides a bash terminal on Windows
  - VirtualBox provides VM virtualization
  - Vagrant automates VM setup
- Mac – VirtualBox and Vagrant
  - Already have a terminal
  - Use Homebrew to install VirtualBox and Vagrant

Windows/Mac Set up VM

- Start GIT Bash or Mac terminal and there
  - Create directory where the VM will live

```
$ cd <somewhere sensible>
$ mkdir singularity-2.4
$ cd singularity-2.4
```

  - Initialize and download the Vagrant Box

```
$ vagrant init singularityware/singularity-2.4
$ vagrant up
```

http://singularity.lbl.gov/install-windows

http://singularity.lbl.gov/install-mac

Windows/Mac connect to VM

- SSH to the spun up VM

```
$ vagrant ssh
```

- Now we are in the VM

```
vagrant@vagrant:~$ which singularity
/usr/local/bin/singularity
vagrant@vagrant:~$ singularity --version
2.4-dist
```

Install Singularity

- In Ubuntu VM, or standalone Linux, follow https://www.sylabs.io/guides/3.0/user-guide/quick_start.html#quick-installation-steps to install Singularity

$