THE UNIVERSITY OF UTAH™

TOGETHER WE REACH

Center for High-Performance Computing

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
- Prepare your computer for containers
- 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/Containers18s.pdf

   https://tinyurl.com/y8v44z95

2. If you have CHPC account, using terminal application (Mac terminal, PuTTY, GIT Shell)
   – `ssh uxxxxxx@singularity.chpc.utah.edu`

3. Make sure you can see singularity
   – `which singularity`

4. Make sure you can sudo singularity command
   – `sudo singularity –version`

OR – if you don't have CHPC account, use Singularity on your laptop

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

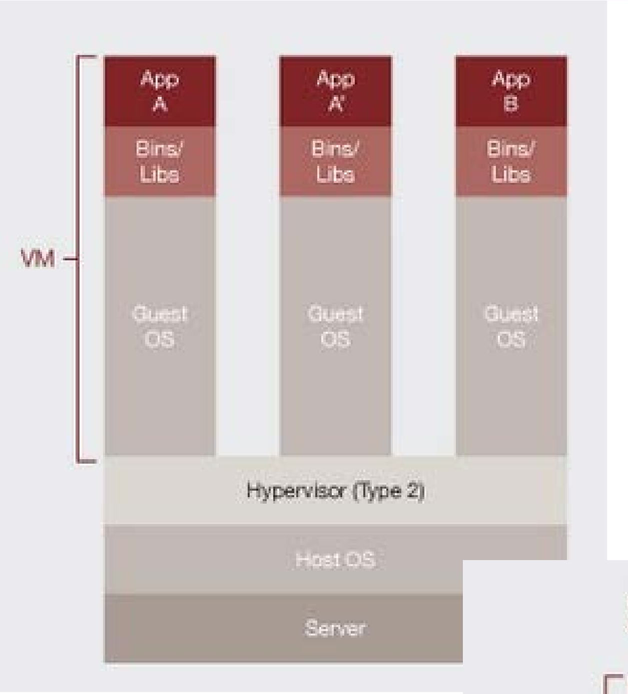
Additional bonus

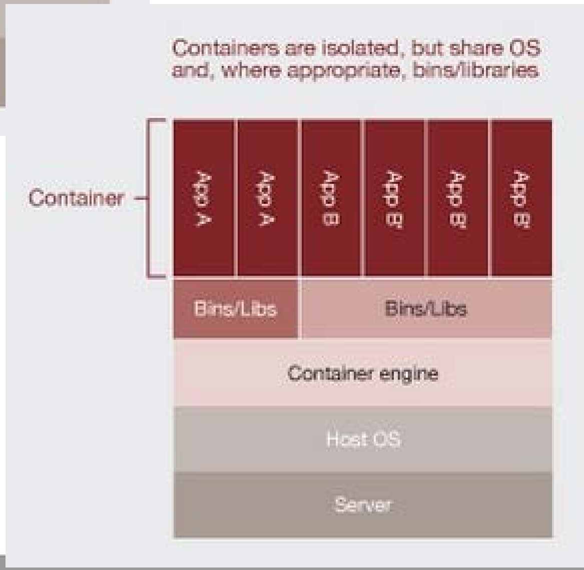
- Old applications built on old Linux versions can run on newer Linux host

Container basics

Virtualization basics

- Hardware virtualization
  - Running multiple OSes on the same hardware
  - VMWare, VirtualBox
- OS level virtualization
  - run multiple isolated OS instances (guests) 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
  - 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

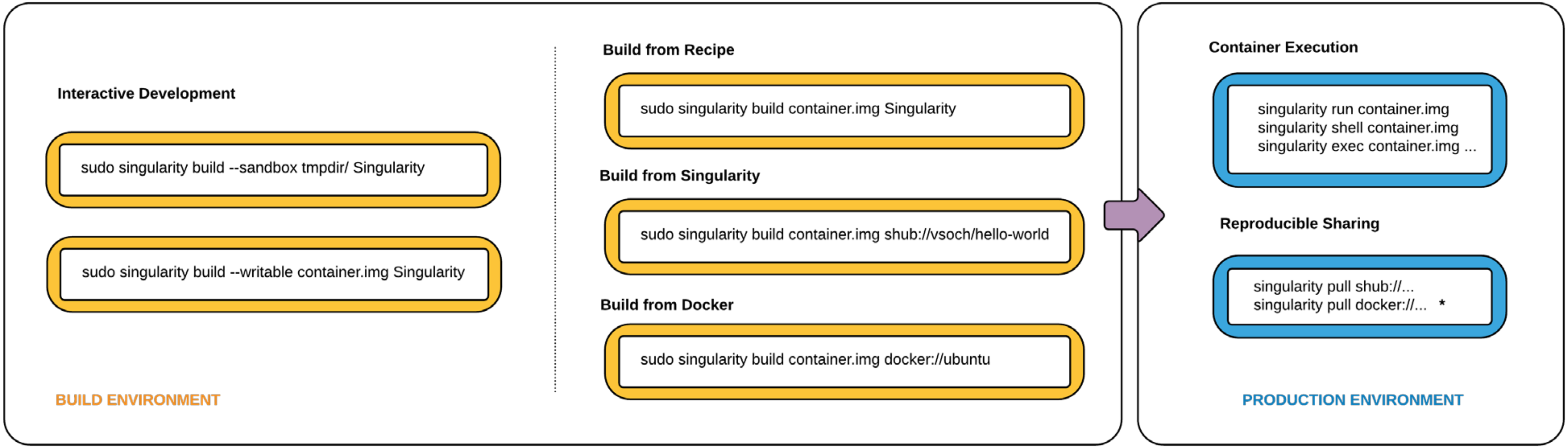
An aside into security

- 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)
  - 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
- Are the only options for our new Protected Environment cluster

Singularity workflow



* Docker construction from layers not guaranteed to replicate between pulls

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
```

Linux - Install Singularity

- In Ubuntu VM, or standalone Linux

```
$ VERSION=2.4
$ wget
https://github.com/singularityware/singularity/releases/download/$VERSION/
singularity-$VERSION.tar.gz
$ tar xvf singularity-$VERSION.tar.gz
$ cd singularity-$VERSION
$ ./configure --prefix=/usr/local
$ make
$ sudo make install
```

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

Now we're ready to use singularity

Or use our singularity box

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`

Build and run Singularity containers

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
- You can run a container as an user
  - You can run your own containers at CHPC
  - You can run CHPC provided containers at CHPC

Run someone else's container

- Singularity allows to run images from Docker hub (and Singularity hub)

```
$ singularity shell docker://ubuntu:latest
$ whoami
$ env | grep SINGULARITY
$ exit
```

- Other ways to run

$ singularity exec *image* program

$ singularity run *image*

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

- Create a production container

```
$ sudo singularity build ubuntu16.simg 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:16.04
```

  - Linux distro based

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

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 bootstrap
  - 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

Let’s get a definition file

- Download CHPC containers GIT repo

```
$ git clone https://github.com/CHPC-UofU/Singularity-ubuntu-python
```

- Go to the `Singularity-ubuntu-python` directory and view what’s in there

```
$ cd Singularity-ubuntu-python
$ ls
$ cat build_container.sh # this script builds the container
$ more Singularity # this is the definition file
```

Build the container

- Simply type the build script

```
$ ./build_container.sh
```

- CHPC specific caveats
  - In order to see your home directory and scratches, file server mount points need to be created in the container

```
$ mkdir /uufs /scratch
```

Run the container

- Locally

```
$ singularity shell ubuntu_python.simg
$ /usr/bin/python -c "import numpy as np;np.__config__.show()"
```

- At CHPC cluster

```
$ scp ubuntu_python.simg myUNID@ember.chpc.utah.edu:~/
$ ssh myUNID@ember.chpc.utah.edu
$ ml singularity/2.4
$ singularity shell ubuntu_python.img
$ /usr/bin/python -c "import numpy as np;np.__config__.show()"
```

Run the container

- In the protected environment
  - Pack the container directory to archive and copy to cluster

```
$ sudo tar cfz ubuntu_python.dir.tar.gz ubuntu_python.dir
$ scp ubuntu_python.dir myUNID@ember.chpc.utah.edu:~/
```

  - ssh to the cluster, unpack archive and run

```
$ ssh myUNID@ember.chpc.utah.edu
$ ml singularity/2.4
$ tar xfz ubuntu_python.dir.tar.gz –C /scratch/local
$ singularity shell --userns /scratch/local/ubuntu_python.dir
$ /usr/bin/python -c "import numpy as np;np.__config__.show()"
```

Importing Docker container

- Need to have `/uufs` mount point for mounting home
- Build your own Docker container with `/uufs` and `/scratch`
- Build Singularity container based on Docker container and add `/uufs` and `/scratch`
- Use persistent overlay
  - an image that “sits on top” of compressed, immutable squashfs container
  - New in Singularity 2.4

Importing Docker container

- Use of persistent overlay
  - On a machine with sudo singularity (singularity.chpc.utah.edu)

```
$ singularity pull docker://ubuntu:latest
$ singularity image.create --size 2 chpc-overlay.img
$ sudo singularity shell --overlay chpc-overlay.img ubuntu-latest.img
$ mkdir /uufs /scratch ; exit
```

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

```
$ ... scp chpc-overlay.img and ubuntu-latest.img
$ ml singularity/2.4
$ singularity shell --overlay chpc-overlay.img ubuntu-latest.img
```

Importing Docker container in the PE

- `--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/2.4
$ 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
```

Other Singularity features

- Container checks
  - Tags or scripts to check on things in the container
- Labels and metadata
- Scientific Filesystem (SCIF)
  - Multiple programs and dependencies in one container
- Image group commands
  - Create, export, import, resize containers
- Container instantiation
  - Run containers in the background (databases, web servers)

Some useful tips

- Binding mount points

```
$ 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 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

Using GPUs

- Need to bring in the Nvidia driver stack
  - Pre Singularity 2.3 – explicitly install – make sure to have the same driver version on the host and in the container
  - Singularity 2.3+ --nv runtime flag

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

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
  - 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

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
```

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

Containers for complex software

When to use 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

- Bootstrap the basic container
- 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

Example - bioBakery

bioBakery
A virtual environment for meta'omic analysis

- 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

bioBakery
A virtual environment for meta'omic analysis

- 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

- 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

Questions?

Survey

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