Apptainer / Singularity
Containers
On the Clusters

W00t! UC Santa Barbara!
Ye Olde People Introductions

Paul Weakliem, PhD  
Co-Director  
Center for Scientific Computing & California Nanosystems Institute  
Eling 3231  
weakliem@cnsi.ucsb.edu

Fuzzy Rogers  
That guy in the MRL  
Center for Scientific Computing & Materials Research Laboratory  
MRL 2066B  
fuz@ucsb.edu

Yu-Chieh “Jay” Chi, PhD  
Research Computing Consultant  
Center for Scientific Computing & Enterprise Technology Services  
Elings 3229  
jaychi@ucsb.edu
Ack!

• Acknowledgements - https://csc.cnsi.ucsb.edu/publications

Please acknowledge the CSC in publications and presentations if you are using our facility’s computational resources (including staff involvement) in your research.

“We acknowledge support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1720256) and NSF CNS- 1725797.”
Caveat Emptor

Here is where I absolve myself from all blame by stating that the soon to be aforementioned was to the best of my knowledge.
What is this thing you call a container?

- Containers are Linux software environments where the user can have control over everything but the kernel.

- Apptainer / Singularity containers can be used to package entire scientific workflows, software and libraries, and even data, in an immutable format. This means that you don’t have to ask your cluster admin to install anything for you - you can create a software workflow in an Apptainer / Singularity container and run it on the clusters.

- With Docker integration, one can utilize proven shared containers as if they were applications (that can contain multiple applications).
Apptainer / Singularity on Pod

- [https://apptainer.org/docs/user/latest/](https://apptainer.org/docs/user/latest/) ➔ docs and info
- module load apptainer (or singularity/3.5.2 or singularity/2.6)
- Binaries of apptainer and singularity (and their builds) are in /sw/singularity
- Images are created by ‘Definition’ (.def) files and are very bare bones - you need to ask for the packages you want installed
- The resulting image files (.sif) are immutable

The one command to execute.

Such an immutable kitty!
Apptainer & Docker

- Apptainer can pull and transmogrify docker containers to create a .sif (singularity image format) file/image

```
apptainer pull docker://ghcr.io/apptainer/lolcow
...exciting things happen ...
```

INFO: Converting OCI blobs to SIF format
INFO: Starting build...
Getting image source signatures
Copying blob 5ca731fc36c2 done
Copying blob 16ec32c2132b done
Copying config fd0daa4d89 done
Writing manifest to image destination
Storing signatures
2023/02/08 14:37:49  info unpack layer:
  sha256:16ec32c2132b43494832a05f2b02f7a822479f8250c173d0ab27b3de78b2f058
2023/02/08 14:37:50  info unpack layer:
  sha256:5ca731fc36c28789c5ddc3216563e8bfca2ab3ea10347e07554ebba1c953242e
INFO: Creating SIF file...
Apptainer & Docker

- Look at the SIF
  -bash-4.2$ ls -lh
  ...
  -rw-r-xr-x 1 fuz seshadri 72M Feb 8 14:37 lolcow_latest.sif
- Run the container with input from the outside and then exit back to CentOS 7
  Pod:
  -bash-4.2$ apptainer exec lolcow_latest.sif cowsay moo

- Clarus the Dogcow
  1983, Apple (not created by the container)
Apptainer & Docker

- `bash-4.2$ apptainer exec lolcow_latest.sif cowsay moo`
- `bash-4.2$ apptainer run lolcow_latest.sif cowsay moo`

 Runs the commands under `%runscript` in the `.def` (Definition) file - a bit of a black box if you cannot see how the container was defined.

This command will launch an Apptainer container and execute a runscript if one is defined for that container. The runscript is a metadata file within the container that contains shell commands. If the file is present (and executable) then this command will execute that file within the container automatically. All arguments following the container name will be passed directly to the runscript.

exec just executes the program in the container with the input from the end

That's a cat.
That's a cow.
Let’s look a bit at our .sif image - we can shell into it:

-bash-4.2$ `apptainer shell lolcow_latest.sif`

Apptainer>

Apptainer> cat /etc/debian_version
bullseye/sid

Apptainer> df -h

Apptainer> which cowsay
/usr/games/cowsay

Apptainer> set | grep games ←—— ‘set’ shows your environment variables
PATH=/usr/games:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Apptainer> exit

```
Filesystem  Size  Used  Avail  Use%  Mounted on
overlay      16M   12K   16M   1%   /
devtmpfs     94G    0    94G    0%  /dev
tmpfs        94G   19M    94G   1%  /dev/shm
/dev/md126   437G  128G  309G  30%  /tmp
beegfs_nodev 655T  569T   87T  87%  /home/fuz
tmpfs        16M   12K   16M   1%  /etc/group
```
Apptainer & Docker

- **Can mount other filesystems with the `--bind` flags:**
  
apptainer shell --bind /scratch,/sw lolcow_latest.sif
  
  Apptainer> df -h

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay</td>
<td>16M</td>
<td>12K</td>
<td>16M</td>
<td>1%</td>
<td>/</td>
</tr>
<tr>
<td>devtmpfs</td>
<td>94G</td>
<td>0</td>
<td>94G</td>
<td>0%</td>
<td>/dev</td>
</tr>
<tr>
<td>tmpfs</td>
<td>94G</td>
<td>19M</td>
<td>94G</td>
<td>1%</td>
<td>/dev/shm</td>
</tr>
<tr>
<td>/dev/md126</td>
<td>437G</td>
<td>128G</td>
<td>309G</td>
<td>30%</td>
<td>/tmp</td>
</tr>
<tr>
<td>beegfs_nodev</td>
<td>655T</td>
<td>569T</td>
<td>87T</td>
<td>87%</td>
<td>/home/fuz</td>
</tr>
<tr>
<td>tmpfs</td>
<td>16M</td>
<td>12K</td>
<td>16M</td>
<td>1%</td>
<td>/etc/group</td>
</tr>
<tr>
<td>10.0.50.249:/scratch</td>
<td>19T</td>
<td>8.4T</td>
<td>9.9T</td>
<td>46%</td>
<td>/scratch</td>
</tr>
<tr>
<td>10.0.50.254:/sw</td>
<td>3.5T</td>
<td>1.8T</td>
<td>1.6T</td>
<td>53%</td>
<td>/sw</td>
</tr>
<tr>
<td>/dev/loop0</td>
<td>72M</td>
<td>72M</td>
<td>0</td>
<td>100%</td>
<td>/sw/singularity/apptainer/var/apptainer/mnt/session/rootfs</td>
</tr>
</tbody>
</table>

- **Not everything you find in Docker easily turns into a .sif** - just because you find a docker website with what you want does not mean it will be ‘easy’ to make an apptainer out of it. If you have docker installed, then you can try your hand at making an image and porting it over.
  
  For instance: [https://hub.docker.com/r/nvaitc/ai-lab](https://hub.docker.com/r/nvaitc/ai-lab)
Building an Apptainer

- You can build your very own .sif container using .def files with all the packages you want that exist in base repositories (maybe non-base too, haven’t looked much)

- Bash-4.2$ more fuzcontainer.def
  BootStrap: docker
  From: ubuntu:20.04
  %post
    apt-get -y update
    apt-get -y install cowsay vim bc python3
  %environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH
  %runscript
    date | cowsay
  %labels
    Author Fuz

Let’s try vim in lolcow_latest.sif
- Bash-4.2$ apptainer shell --bind /scratch,/sw lolcow_latest.sif
  Apptainer> vim
  bash: vim: command not found

Now with fuzcontainer.sif
- Bash-4.2$ apptainer shell --bind /scratch,/sw fuzcontainer.sif
  Apptainer> vim

Build the .def with:
apptainer build fuzcontainer.sif fuzcontainer.def
Building an Apptainer

Let's play with the %runscript … what if I go… (modifying my .def file)

%runscript
    date | cowsay
df -h | cowsay
bc < bcinput

Build it … apptainer build fuzcontainer-bc.sif fuzcontainer.def

-bash-4.2$ cat bcinput
7 * 3.000482469859387459872934923

-bash-4.2$ apptainer run fuzcontainer-bc.sif

< Mon Feb 13 10:11:36 PST 2023 >
-----------
\  ^__^  \_______
(oo) \       )\_\|  \----w |  ||  ||
\_____/
-----------

/ Filesystem Size Used Avail Use% Mounted /
| on overlay 16M 12K 16M 1% / devtmpfs   |
| 94G 0 94G 0% /dev tmpfs 94G 19M 94G 1%  |
| /dev/shm /dev/md126 437G 139G 298G 32%  |
| /tmp beegfs_nodev 655T 571T 85T 88%    |
| /home/fuz tmpfs 16M 12K 16M 1%          |
| /etc/group                              |
-----------------------------------------
\  ^__^  \_______
(oo) \       )\_\|  \----w |  ||  ||
\_____/

21.003377289015712219110544461
Building an Apptainer

- You can build from Dockerfiles - but you gotta translate into apptainer syntax

  [https://apptainer.org/docs/user/1.0/docker_and_oci.html#apptainer-definition-file-vs-dockerfile](https://apptainer.org/docs/user/1.0/docker_and_oci.html#apptainer-definition-file-vs-dockerfile)

Here’s the Dockerfile for BioPython

```
FROM ubuntu:16.04
MAINTAINER Tiago Antao <tra@popgen.net>
ENV DEBIAN_FRONTEND noninteractive
# We need this for phylip
  build-essential 
  git 
  python3-numpy 
  wget 
  gcc 
  g++ 
  python3-dev 
  unzip 
  make 
  python3-matplotlib 
  python3-reportlab 
  python3-pip r-base 
  clustalw 
  fasttree 
  t-coffee python3-pil 
  bwa 
  ncbi-blast+ 
  emboss 
  clustalo 
  phylip 
  mafft 
  muscle 
  samtools 
  phyml 
  wise 
  raxml 
  language-pack-en 
  pam1 
  probcons 
  python3-pandas 
  python3.5-dev 
  libxft-dev 
  && apt-get clean

# for Phylo_CDAO
# RUN pip3 install pip --upgrade
RUN pip3 install rdflib --upgrade && pip3 install cython --upgrade && pip3 install numpy --upgrade && pip3 install Pillow --upgrade && pip3 install matplotlib --upgrade && pip3 install pandas --upgrade

# Manual software
RUN echo "export DIALIGN2_DIR=/tmp" >> .bashrc

... and it goes on and on
```

Turning this, by hand, into a modified BioPython .def would be a long process
Nicer to … apptainer pull docker://biopython/biopython
But then you don’t have any customization
Apptainer & GPUs

- Apptainer plays nicely with Pod’s GPUs - use the development node pod-gpu for testing
- –nv (2 hyphens)
- Remember to send SLURM job file to gpu: sbatch -p gpu mygpustuff.job
- Apptainer is better than Singularity for interaction with GPUs

Sidebar: So why ever use Singularity? My suggestion is to not use it. Singularity can give you a writable container, in a relatively easy fashion, that you can manipulate to your liking. Apptainer can do that to - with the Sandbox function, but once you learn how to do something, it’s sometimes easier to stick with it. Apptainer creates an entire subdirectory root filesystem of the container, whereas Singularity keeps it all inside its container.

```bash
-bash-4.2$ apptainer build --sandbox ubuntu/ docker://ubuntu
-bash-4.2$ cd ubuntu
-bash-4.2$ ls
bin  boot  dev  environment  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin
singularity  srv  sys  tmp  usr  var
```

nv - the UCSB Jayich diamond research -
https://www.10-9lab.com/spin-coherence/ (actually it stands for nvidia, not nitrogen vacancy)
As long as we’re sidebarring:
If you already have a container saved locally, you can use it as a
target to build a new container. This allows you to convert containers
from one format to another. For example if you had a sandbox container called ubuntu/
and you wanted to convert it to a SIF container called myubuntu.sif you could:

```
$ apptainer build myubuntu.sif ubuntu/
```

Use care when converting a sandbox directory to the default SIF format. If changes were
made to the writable container before conversion, there is no record of those changes in
the Apptainer definition file rendering your container non-reproducible. It is a best
practice to build your immutable production containers directly from an Apptainer
definition file instead.

(shamelessly stolen from: https://apptainer.org/docs/user/main/build_a_container.html )
Apptainer & GPUs

apptainer pull docker://tensorflow/tensorflow:latest-gpu
apptainer run --nv tensorflow_latest-gpu.sif

Apptainer> python
Python 3.6.8 [Anaconda, Inc.] (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>from tensorflow.python.client import device_lib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'tensorflow'

Ohh? What’s this? Aha - it found the wrong python - it found my anaconda python....

Apptainer> which python
/home/fuz/anaconda3/bin/python

So - let’s use the container’s python that has TF

Apptainer> /bin/python3
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
from tensorflow.python.client import device_lib
.....stuff............

print(device_lib.list_local_devices())

.....stuff.....

Important! Your .bashrc may affect what the container sees. When you submit a job, you’d need /bin/python3 mypython.py

Let’s see what the container sees for the GPUs

Or: Let’s pull a GPU thingy!
Apptainer & GPUs

Continuing the output, seeing the GPUs.....

This is pod-gpu login node, FYI

I know! I was surprised too.
Apptainer & GPUs

- Okay- this is all well and good, but let’s do something SLURM-y

In my frantic preparation for this talk I decided to grab the TF “1st grader” example - it’s all Paul’s fault

My SLURM job script:

```bash
#!/bin/bash
# ask for 1 core on one node and 1 GPU
#SBATCH -N 1 --ntasks-per-node=1
#SBATCH --time=01:00:00
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1

cd $SLURM_SUBMIT_DIR
module load apptainer
hostname
apptainer exec --nv tensorflow_latest-gpu.sif /bin/python3 teras-example.py
```

Umm, yeah- this is your friend if you use GPUs, you’ll need that flag

My python file (teras-example.py) to run:

```python
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
import tensorflow as tf
print("TensorFlow version:", tf.__version__)

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])
predictions = model(x_train[:1]).numpy()
nn.softmax(predictions).numpy())
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions).numpy()
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.evaluate(x_test, y_test, verbose=2)
probability_model = tf.keras.Sequential([model,
                                          tf.keras.layers.Softmax()])
probability_model(x_test[:5])
```

Agitated cat sees the problem. Fuzzy did not.
Apptainer & GPUs

OUTPUT -
...stuff from finding NV devices ....
TensorFlow version: 2.11.0

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step

Epoch ⅕
2023-02-09 16:19:27.502094: I tensorflow/compiler/xla/service/service.cc:173] XLA service 0x7f3e84022ba0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
2023-02-09 16:19:27.502181: I tensorflow/compiler/xla/service/service.cc:181]   StreamExecutor device (0): Tesla V100-PCIE-32GB, Compute Capability 7.0
2023-02-09 16:19:29.277989: I tensorflow/compiler/jit/xla_compilation_cache.cc:477] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
1875/1875 [==============================] - 8s 3ms/step - loss: 0.3024 - accuracy: 0.9112
Epoch %
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1465 - accuracy: 0.9573
Epoch %
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1113 - accuracy: 0.9662
Epoch %
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0901 - accuracy: 0.9718
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0759 - accuracy: 0.9766
313/313 - 1s - loss: 0.0813 - accuracy: 0.9760 - 645ms/epoch - 2ms/step

Looks good - oh wait … none of the function evaluations show up in the output.
Apptainer & GPUs

- Python evaluations are NOT standard out. You want your results? Be sure to write them (or verify that they go to standard out)

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                tf.keras.layers.Dense(128, activation='relu'),
                                tf.keras.layers.Dropout(0.2),
                                tf.keras.layers.Dense(10)])
predictions = model(x_train[:1]).numpy()
f = open('results.txt','w')
f.write(str(predictions) + '
')
f.write(str(tf.nn.softmax(predictions).numpy()) + '
')
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
f.write(str(loss_fn(y_train[:1], predictions).numpy()) + '
')
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
f.write(str(model.evaluate(x_test, y_test, verbose=2)) + '
')
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
f.write(str(probability_model(x_test[:5])) + '
')

Any python evaluation/function that outputs, throw it into an f.write

Here- I'm saying - open results.txt for writing

\n = newline (you probably knew that)

Skeptical cat is saying - "You're not labeling your output! It's garbage! Put in a f.write("Predictions!! \%d\n") or whatever before every python evaluation! Imbecile!"
But maybe Skeptical cat doesn't realize I want to take the 'unadulterated' output and process through another program that would prefer not to have labels like "Predictions"?
Apptainer & GPUs

-bash-4.2$ more results.txt

\[
\begin{bmatrix}
-0.6788503 & 0.08507155 & 0.7489541 & -0.3592714 & -0.4191291 & 0.3637312 & 0.15091619 & 0.44977978 & 0.41373825 & 0.18217495 \\
0.04244909 & 0.09112456 & 0.17699295 & 0.05843321 & 0.05503815 & 0.12040813 & 0.09732657 & 0.13122791 & 0.12658247 & 0.10041693 \\
2.1168683 & 2.3465797901153564 & 0.08789999783039093
\end{bmatrix}
\]

Voila. Yup. Exactly what I expected. Uh huh. Sure. Well- they are results, whether they mean anything is a different story.

17 sig figs... useful... if you're measuring the diameter of a proton 😅
Especially for GPUs, there are some prebuilt images which are easy to work with

https://catalog.ngc.nvidia.com/ and, for example, if you search on Gromacs, you get

https://catalog.ngc.nvidia.com/orgs/hpc/containers/gromacs which has a docker image you can download(!!), as well as some instructions.


and then you can do your usual with your 'gromacs-2022.3.sif' file

```
cat runfile
/usr/local/gromacs/avx2_256/bin/gmx mdrun -ntomp 8 -ntmpi 4 -s benchRIB.tpr -nsteps 400
```

```
apptainer run --nv gromacs_2022.3.sif < runfile
```

Will run your gromacs job for ‘benchRIB’.

However…..
Security (or beware!)

- Yeah- be careful about any Docker images you find on the internet. It does not take too much imagination to create a Docker image called “Generate my Physics Thesis with ChatGPT” And, when you run it, it promptly deletes all your files.
- Trusted workflows, from trusted sources - a good start
- Apptainer uses private PGP keys to create a container signature, and the corresponding public key in order to verify the container signature. Verification of signed containers can be done at any time by a user and happens automatically in apptainer pull commands against Library API registries. The prevalence of PGP key servers, (like https://keys.openpgp.org/), make sharing and obtaining public keys for container verification relatively simple. Yup, sure, you’ll all do that.
Apptainer Instances

The subtitle to this slide is “How to impress a prospective employer to hire you at 6 figures”

- Instances are running containers waiting for interaction
- IMHO - these are not suitable for the clusters
- “Instances allow you to run containers as background processes. This can be useful for running services such as web servers or databases.”
- I only mention them because they will give you a feel for a cousin cluster called Nautilus that uses Kubernetes. If you say the word “Kubernetes” in a job interview, and mention “instances”, and how much you like containers, you’ll probably get the job.
- [https://portal.nrp-nautilus.io/](https://portal.nrp-nautilus.io/)
Making your own Containers

• The Workflow – Step 1 – Build a linux VM so you can be root
  
  • Download and install a Virtual Machine application (I chose VirtualBox)
  
  • For pod.cnsi.ucsb.edu, build a CentOS 7 virtual machine
    
    • Choose your HD size so that it can accommodate your OS *and* your singularity images that you will create (i.e. 20GBs or so)
    • I chose CentOS-7-x86_64-Everything-1804.iso as the base
      
      • My favorite mirror is http://mirrors.oit.uci.edu/centos/7/isos/x86_64/
      • Remember that you want to install the Development Tools (Compute Node has it on the left)
    • Then we’ll download and build singularity as we’ll be root on our own little linux machine.
    • And then you can build singularity images to fit your exact needs. Once you’ve tested your workflow, you can copy those images to pod.cnsi.ucsb.edu and create jobs for them to run.

Caveat: this is old for Singularity
Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 5.1 packages, see VirtualBox 5.1 builds. Consider upgrading.

VirtualBox 5.2.20 platform packages

- Windows hosts
- macOS hosts
- Linux distributions
- Solaris hosts

The binaries are released under the terms of the GPL version 2.

See the changelog for what has changed.

You might want to compare the checksums to verify the integrity of downloaded packages. The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!

- SHA256 checksums, MD5 checksums

Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

VirtualBox 5.2.20 Oracle VM VirtualBox Extension Pack

- All supported platforms

Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See this chapter from the User Manual for an introduction to this Extension Pack. The Extension Pack binaries are released under the VirtualBox Personal Use and Evaluation License (PUEL). Please install the same version extension pack as your installed version of VirtualBox.
Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

CentOS-7-x86_64-Everything-1804 (1).iso (8.0 MiB)
Making your own Containers

• The Workflow – Step 2 – Getting and Building Singularity / Apptainer
  • Start your VM from VirtualBox, login as root
  • [https://github.com/sylabs/singularity/releases](https://github.com/sylabs/singularity/releases) - the .tar.gz are fine
    • wget [https://github.com/sylabs/singularity/releases/download/2.5.2/singularity-2.5.2.tar.gz](https://github.com/sylabs/singularity/releases/download/2.5.2/singularity-2.5.2.tar.gz)
    • gunzip that file, untar that file
    • cdsingularity-2.5.2
    • ./configure –prefix=/singularity (prefix not necessary) Note if configure fails with missing packages – you might need to yum install somepackage\likegcc
    • make - if there are no errors....
    • make install
    • yum install epel-release , yum install debootstrap
  • Voila – you now have singularity in your VM and can create singularity / apptainer containers
Making your own Singularity Containers

• The Workflow – Step 3 – Creating an Ubuntu container

• Build an empty container

  ● export PATH=$PATH:/singularity/bin
  ● singularity create ubuntu.img
  ● singularity image.expand --size4000 ubuntu.img
  ● singularity build ubuntu.img createdeb.def where createdeb.def:
    
    BootStrap: debootstrap
    DistType: Debian
    MirrorURL: http://us.archive.ubuntu.com/ubuntu
    
    OSVersion: xenial
    %runscript
    apt-get install python

  ● singularity shell ubuntu.img ← you're now in the container ( --writable)
  ● apt-get install python sudo ← and anything else you want to install (might need sudo
    for other apt-gets like sudo apt-get install somepackage or another – so you need sudo)
  ● Exit ← gets you out the container back into CentOS 7
Next Steps

- Now that you have a container, customize it to work with your workflow. Install whatever packages you need.
- When you use a container on the clusters, it automatically mounts your home directory.
- The container sees all of the system’s memory and CPUs, but none of the other filesystems/directories unless you explicitly mount them – and then they’re generally readonly unless it’s /scratch.
  - singularityshell-B/scratch:/mnt/sw/singularity/SingularityImages-knot/ubuntu_croco.img Here, the /scratch directory is mounted in your container at /mnt.
- From your CentOS 7 install, scp myubuntu.img username@pod.cnsi.ucsb.edu
- Note that once your image is on the clusters, it is immutable (unless you ask us to alter something)
• Example job submission file on pod.cnsi.ucsb.edu – test-croco.job

#!/bin/bash -l
#Serial (1 core on one node) job…
#SBATCH --nodes=1 --ntasks-per-node=1
cd $SLURM_SUBMIT_DIR
source .bashrc
singularity exec -B /scratch:/mnt /sw/singularity/SingularityImages-knot/ubuntu_croco.img /home/fuz/test-croco.in

• Example run file for the container – test-croco.in

export PATH=/home/fuz/anaconda2/bin:$PATH apt list --installed
echo ""
echo "Which python am I using:"
which python
echo ""
Echo "Determine whether a number is prime or not" python primeornot.py

• Submit the job
  • sbatch test-croco.job