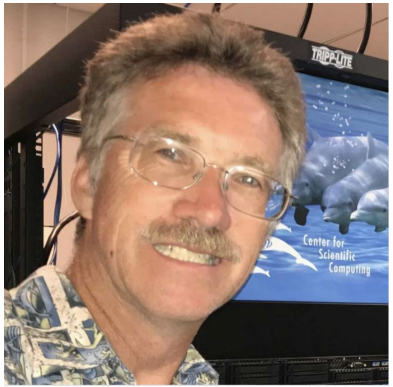
Apptainer / Singularity Containers On the Clusters

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

CSC Center for Scientific Computing

CNSI

UC SANTA BARBARA MRL Materials Research Laboratory
NSF Materials Research Science and Engineering Center [DMR 1720256]

ENTERPRISE TECHNOLOGY SERVICES

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. Acknowledgements in publications are one powerful metric the NSF uses to determine funding.

"We acknowledge support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-2308708) 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 a 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/ ←- docs and info
- module load apptainer ←loads v1.2.5 (or apptainer/1.1.5) (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

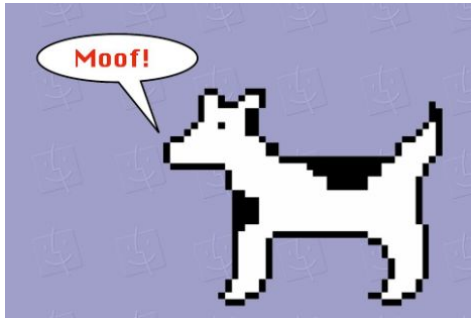
- Look at the SIF
  -bash-4.2$ ls -lh
  …
  -rwxr-xr-x 1 fuz seshadri 72M Feb 8 14:37 lolcow_latest.sif
- Run the container with input from the outside and then exit back to CentOS 7 Pod:
  -bash-4.2$ apptainer exec lolcow_latest.sif cowsay moo

Program to run in container

Input for that program

```
 _____
< moo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

←– the output



<- 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

```
-bash-4.2$ apptainer run lolcow_latest.sif cowsay
 _____________________________
< Sat Feb 11 13:22:51 PST 2023 >
 -----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
               ||----w |
               ||     ||
-bash-4.2$ apptainer exec lolcow_latest.sif cowsay moo
 _____
< moo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
               ||----w |
               ||     ||
```

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.

That's a cat.
That's a cow.

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

Apptainer & Docker

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

```
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> which cowsay

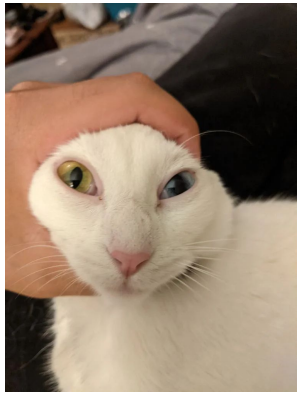
/usr/games/cowsay

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

Apptainer & Docker



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

```
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
10.0.50.249:/scratch  19T  8.4T  9.9T  46% /scratch
10.0.50.254:/sw       3.5T  1.8T  1.6T  53% /sw
/dev/loop0          72M   72M    0 100% /sw/singularity/apptainer/var/apptainer/mnt/session/rootfs
```

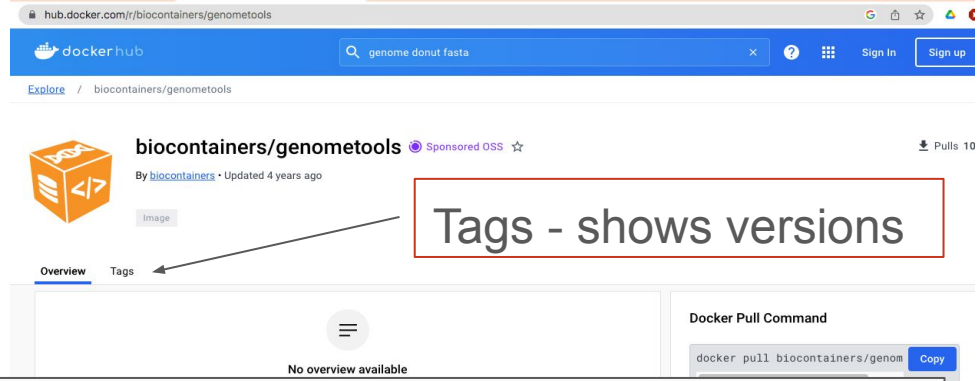
- Not everything you find in Docker easily turns into a .sif - just because you find a docker website with what you want does not mean it will be ‘easy’ to make an apptainer out of it. If you have docker installed, then you can try your hand at making an image and porting it over.
  For instance: https://hub.docker.com/r/nvaitc/ai-lab

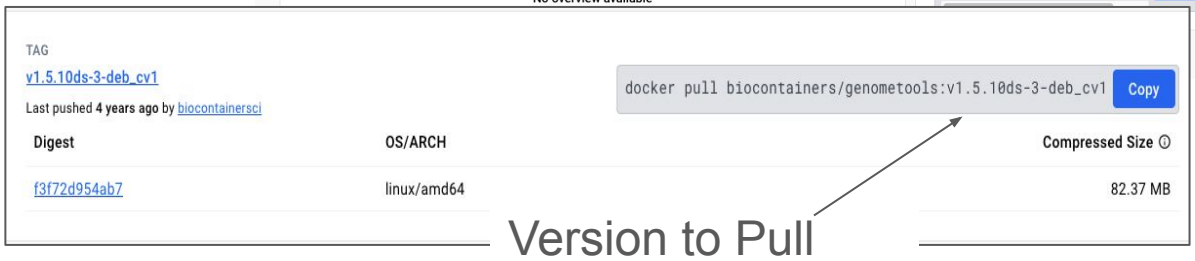
Apptainer & Docker

Another Docker Hub Pull - https://hub.docker.com/r/biocontainers/genometools

-bash-4.2$ apptainer pull docker://biocontainers/genometools
FATAL: While making image from oci registry: error fetching image to cache: failed to get checksum for docker://biocontainers/genometools: reading manifest latest in docker.io/biocontainers/genometools: manifest unknown

-bash-4.2$ apptainer pull docker://biocontainers/genometools:v1.5.10ds-3-deb_cv1
INFO: Converting OCI blobs to SIF format
INFO: Starting build...





May have to add version number in order to pull the Docker image.

← Obsequious cat

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
```

I've chosen to install vim. Is an editor necessary? Probably not. But, you'll be surprised what isn't installed by default.

Pipe 'date' into cowsay - when run'd, this is what it runs

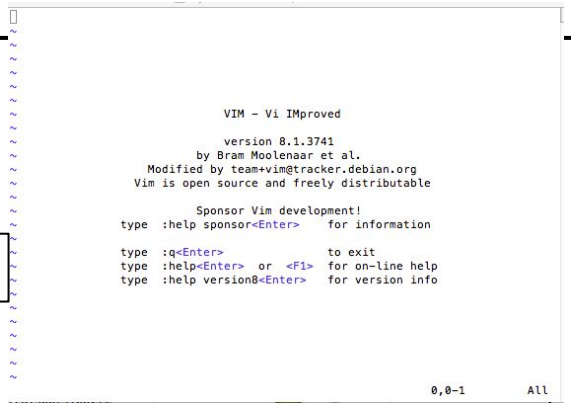
Build the .def with:
apptainer build fuzcontainer.sif fuzcontainer.def

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
```



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
```

```
-bash-4.2$ cat bcinput
7 * 3.00048246985938745987293492 3
```

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

```
-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 |
                ||     ||
21.0033772890157122191105444 61
```

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

  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
RUN echo 'deb http://archive.ubuntu.com/ubuntu xenial multiverse' >> /etc/apt/sources.list \
                    && apt-get update \
                    && apt-get upgrade -y --force-yes \
                    && apt-get install -y --force-yes \
    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 \
        paml \
        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 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 & Docker

Docker Pull to Mac, Push to DockerHub, Pull to Pod for Apptainer

- Install Docker on Mac (https://docs.docker.com/desktop/install/mac-install/)
- Github CROCO with Dockerfile (https://github.com/AndresSepulveda/docker-croco-public)

  Coastal and Regional Ocean COmmunity model
- Create a docker desktop login, login, push your image and get the dockerhub link.

```
docker build --platform linux/amd64 -t croco-amd64 .
docker tag croco fuzzrence/croco-amd64:version1
docker push fuzzrence/croco-amd64:version1
The push refers to repository [docker.io/fuzzrence/croco-amd64]
5f70bf18a086: Pushed
4d8229e8583b: Pushed ….
version1: digest:
sha256:bdf28077d1849391ab939e470da2344a9861862e27c4a
3a50bfce38c89c9f591 size: 4083
```

```
bash-4.2$ apptainer build croco-amd64-fuzzrence.sif
docker://fuzzrence/croco-amd64:version1
INFO:    Starting build...
Getting image source signatures
Copying blob cbc92d9d523f done  …..
INFO:    Creating SIF file...
INFO:    Build complete: croco-amd64-fuzzrence.sif
-bash-4.2$ apptainer shell croco-amd64-fuzzrence.sif
Apptainer> cd /home/croco/
```

Docker may need configuration, which should be done on docker computer.

Apptainer Sandboxes

Sandbox - Writable Containers for Modifying Known Images

```
-bash-4.2$ apptainer build --sandbox croco-sandbox croco_oceanv1.2.1b_latest.sif
```

Target (i.e. new directory)

Source (.sif .def etc.)

--fakeroot required!

```
-bash-4.2$ apptainer shell --writable --fakeroot croco-sandbox
WARNING: Skipping mount /etc/localtime [binds]: /etc/localtime doesn't exist in container
Apptainer> apt-get install r-base
Reading package lists... Done
…
Processing triggers for install-info (6.8-4build1) ...
Apptainer> R
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
…
```

Sandbox creates a writable directory structure. You can modify files or copy data in/out without being in Apptainer. But to do things that the OS knows about (like installing packages), hop into Apptainer as above.

Build the sandbox: apptainer build croco-sandbox.sif croco-dock-sanbox

Building an Apptainer from GitHub Dockerfile

- pip install spython ← python program to ‘kind of’ convert Dockerfiles into .def files.
- spython recipe dockerfile > dockerfile.def and now you can edit .def
  - Some things in the .def file that prevented dockerfile.def from being built:

    useradd -m croco && echo "croco:croco" | chpasswd && adduser croco sudo <- requires superuser permissions
    && apt-get install git -y \ <- git wants to install openssl which requires superuser permissions
    && apt-get install octave -y \ <- octave wants to install dbus which requires superuser permission
  - Removing those lines allowed a sandbox to be built:

    apptainer build --fakeroot --sandbox croco-dock-sanbox croco-dock.def
  - Some things in the Dockerfile I still need to do —-->

    ln -s /home/fuz/singuularity/apptainer/examples/croco/
    Do that to install large datasets/tools outside of the container- but still visible inside the container.

wget https://data-croco.ifremer.fr/CODE_ARCHIVE/croco-v1.3.tar.gz
gzip -d croco-v1.3.tar.gz
tar -xvf croco-v1.3.tar
rm croco-v1.3.tar
wget https://data-croco.ifremer.fr/CODE_ARCHIVE/croco_tools-v1.3.tar.gz
gzip -d croco_tools-v1.3.tar.gz
tar -xvf croco_tools-v1.3.tar
rm croco_tools-v1.3.tar
wget https://data-croco.ifremer.fr/DATASETS/DATASETS_CROCOTOOLS.tar.gz

spython isn’t perfect - but it’s a way to get going!

```
bash-4.2$ apptainer build croco-sandbox.sif croco-dock-sanbox
-rwxr-xr-x.  1 fuz seshadri 364M Mar  5 21:00 croco-sandbox.sif
```

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

nv - the UCSB Jayich diamond research - https://www.10-9lab.com/spin-coherence/ (actually it stands for nvidia, not nitrogen vacancy)

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-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
```

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

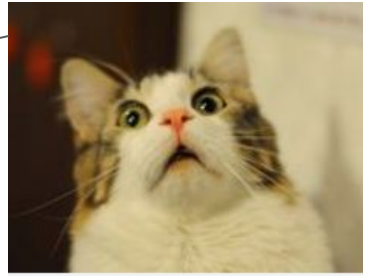
Let’s see what the container sees for the GPUs

Apptainer & GPUs

Continuing the output, seeing the GPUs…..

```
-> device: 0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:17:00.0, compute capability: 7.0
2023-02-09 12:24:25.069041: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created device /device:GPU:1 with 13779 MB memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:65:00.0, compute capability: 7.5
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 8336112430592221785
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 32473874432
locality {
bus_id: 1
links {
}
}
incarnation: 8868894900487111325
physical_device_desc: "device: 0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:17:00.0, compute capability: 7.0"
xla_global_id: 416903419
, name: "/device:GPU:1"
device_type: "GPU"
memory_limit: 14449115136
locality {
bus_id: 1
links {
}
}
incarnation: 14753446127920954602
physical_device_desc: "device: 1, name: Tesla T4, pci bus id: 0000:65:00.0, compute capability: 7.5"
xla_global_id: 2144165316
]
```

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 SLURMy

  I decided to grab the TF "1st grader" example

My SLURM job script:

```
#!/bin/bash
# ask for 2 cores on one node and 1 GPU
#SBATCH -N 1 --ntasks-per-node=2
#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
```

2 cores

Umm, yeah- this is your friend
If you use GPUs, you'll need that flag

Agitated cat sees the problem.
Fuzzy did not.

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

```
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()
predictions) + '\n')
f.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])
```

Apptainer & GPUs

GPU Device Choice

My SLURM job script:

```
#!/bin/bash
# ask for 2 core on one node and 1 GPU
#SBATCH -N 1 --ntasks-per-node=2
#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
```

NO SRUN. SRUN BAD FOR GPUs.

(unless you include the gres text in your srun, but that's just more work)

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

```
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
export CUDA_VISIBLE_DEVICES=$gpu_devices   or
export CUDA_VISIBLE_DEVICES=0

(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()
predictions) + '\n')
f.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])
```

On Pod, the GPU nodes have 4 GPUs per node (V100s, 32GB RAM). Usually the user should just let SLURM via the #SBATCH –gres=gpu:1 line choose *which* of the GPUs on the node your code will use. However! There is a lot of code out there that 'hard codes' the environment variable CUDA_VISIBLE_DEVICES
You do not want hard coded GPU device choice, SLURM will do it for you.

GPU_DEVICE_ORDINAL environment variable tells you what GPU you're using (0-3).

Apptainer & GPUs

GPU Device Choice

When your job is running on a GPU node, you can ssh to it and see what's going on.

```
ssh node123
nvidia-smi
Mon Mar  4 17:32:01 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.27.04    Driver Version: 460.27.04    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
….stuf….

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    226879      C   python                           309MiB |
|    1   N/A  N/A    226883      C   python                           309MiB |
|    2   N/A  N/A    226870      C   python                           309MiB |
|    3   N/A  N/A    183605      C   ...uPSS/bin/modelGravity_Phi     2957MiB |
+-----------------------------------------------------------------------------+
```

Driver version (latest CUDAs may require us to update)

This is a happy node - 4 jobs, 1 unique GPU per job

This is a sad node - 4 jobs, 3 jobs on GPU0, and 1 on GPU3

SLOW

```
|=============================================================================
==============|
|    0   N/A  N/A    226879      C   python                           309MiB |
|    0   N/A  N/A    226883      C   python                           309MiB |
|    0   N/A  N/A    226870      C   python                           309MiB |
|    3   N/A  N/A    183605      C   ...uPSS/bin/modelGravity_Phi     2957MiB |
+-----------------------------------------------------------------------------+
```

Apptainer & GPUs

OUTPUT (since I didn't name my output file in SBATCH, it'll be something like 'slurm-3411234.out'):
…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
2023-02-09 16:19:19.821484: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 27060 MB memory: -> device: 0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:17:00.0, compute capability: 7.0
2023-02-09 16:19:19.822409: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created device /job:localhost/replica:0/task:0/device:GPU:1 with 13779 MB memory: -> device: 1, name: Tesla T4, pci bus id: 0000:65:00.0, compute capability: 7.5
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 16:19:27.502222: I tensorflow/compiler/xla/service/service.cc:181] StreamExecutor device (1): Tesla T4, Compute Capability 7.5
2023-02-09 16:19:27.655725: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
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 ⅖
1875/1875 [==========] - 6s 3ms/step - loss: 0.1465 - accuracy: 0.9573
Epoch ⅗
1875/1875 [==========] - 4s 2ms/step - loss: 0.1113 - accuracy: 0.9662
Epoch ⅘
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) + '\n')
f.write(str(tf.nn.softmax(predictions).numpy()) + '\n')
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
f.write(str(loss_fn(y_train[:1], predictions).numpy()) + '\n')
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
f.write(str(model.evaluate(x_test,  y_test, verbose=2)) + '\n')
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
f.write(str(probability_model(x_test[:5])) + '\n')
```

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\r\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
[[-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]
tf.Tensor(
[[0.05063404 0.07061377 0.17750613 0.07071802 0.09076004 0.10206965
  0.13977249 0.08153952 0.09993464 0.11645163]
 [0.04648628 0.08442134 0.07763657 0.08092945 0.08309506 0.12882507
  0.19512239 0.10946266 0.11590897 0.07811217]
 [0.09442651 0.08400892 0.11063591 0.086678   0.09030239 0.09628462
  0.10558278 0.12173646 0.0983725  0.11197192]
 [0.04697128 0.08966808 0.14622916 0.03771305 0.04190792 0.08916441
  0.11697701 0.16301493 0.11301447 0.15533967]
 [0.08090983 0.05662173 0.11566644 0.11236666 0.06174224 0.15317099
  0.16894276 0.09333923 0.08112669 0.07611344]], shape=(5, 10), dtype=float32)
```

17 sig figs… useful… if you're measuring the diameter of a proton \s

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

Apptainer & GPUs

PyG from NVIDIA -(https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pyg)

Pytorch Geometric - write and train Graph Neural Networks (GNNs)

(https://distill.pub/2021/gnn-intro/)

```
-bash-4.2$ apptainer pull docker://nvcr.io/nvidia/pyg:23.11-py3
INFO:    Converting OCI blobs to SIF format
….
done!
-bash-4.2$ ls -lh  —----->     -rwxr-xr-x 1 fuz seshadri 9.7G Mar  5 17:30 pyg_23.11-py3.sif
-bash-4.2$ cd /home/fuz/singuularity/apptainer/examples/pyg
-bash-4.2$ apptainer shell –nv pyg_23.11-py3.sif
Apptainer> cd /opt/pyg/gnn-platform/tests/
Apptainer> py.test -s /opt/pyg/gnn-platform/tests/unit/
========================================================================================== test session starts
==========================================================================================
platform linux -- Python 3.10.12, pytest-7.4.3, pluggy-1.3.0
….
================================================================================= 2 passed, 6 warnings in 17.88s
###CHECKING WHAT DEVICES WE SEE – by hand, the old fashioned way###
Apptainer> python
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux …
>>> import torch
>>> for i in range(torch.cuda.device_count()):
...     print(torch.cuda.get_device_properties(i).name)
...
Tesla V100-PCIE-32GB
Tesla T4
```

This is what would be in your SLURM file

Apptainer & GPUs

PyG from NVIDIA -(https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pyg)

Pytorch Geometric - write and train Graph Neural Networks (GNNs)

```
#!/bin/bash
# ask for 2 cores on one node
#SBATCH -N 1 --ntasks-per-node=2
###SBATCH --nodelist=node112
#SBATCH --time=01:00:00
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
cd $SLURM_SUBMIT_DIR
module load apptainer
hostname
nvidia-smi
set
echo "JUMPING INTO CONTAINER"
apptainer exec --nv pyg_23.11-py3.sif sh set.sh
apptainer exec --nv pyg_23.11-py3.sif py.test -s /opt/pyg/gnn-platform/tests/unit/
```

–nodelist=node112 - asking for node112 in particular because it has the latest NVIDIA driver on it - currently updating drivers, but have to wait for jobs to finish. So far node112,node116,node117 have NVIDIA 535.161.07 on them.

Hostname is useful if there are problems - it immediately lets us know what node the job was running on. You certainly don't need nvidia-smi or set - but I wanted to see my environment variables outside of and inside of the container.

SLURM file

Apptainer & GPUs

NVIDIA GROMACS - (https://catalog.ngc.nvidia.com/orgs/hpc/containers/gromacs)

```
apptainer pull docker://nvcr.io/nvidia/pyg:23.11-py3
-bash-4.2$ more gromacs-gpu.job
#!/bin/bash
# ask for 10 cores on one node
#SBATCH -N 1 --ntasks-per-node=20
###SBATCH --nodelist=node112
#SBATCH --time=12:00:00
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
cd $SLURM_SUBMIT_DIR
module load apptainer
apptainer exec --nv /sw/singularity/images/gromacs-2023.2.sif /usr/local/gromacs/avx2_256/bin/gmx mdrun  -ntomp 5 -ntmpi 4  -s
topol.tpr -nsteps 4000
```

We're putting prebuilt images in /sw/singularity/images/

Villin headpiece solvated - smallest known cooperatively folding protein

Output Re: GPUs

On host node112.podcluster 1 **GPU** selected for this run.
Mapping of **GPU** IDs to the 4 **GPU** tasks in the 4 ranks on this node:
PP tasks will do (non-perturbed) short-ranged and most bonded interactions on the **GPU**
PP task will update and constrain coordinates on the **GPU**
Using **GPU** 8x8 nonbonded short-range kernels

Updating coordinates and applying constraints on the **GPU**.

Launch PP **GPU** ops.	4	5	29877	1.511	72.511	21.8
Wait Bonded **GPU**	4	5	101	0.000	0.006	0.0
Wait **GPU** NB nonloc.	4	5	10001	0.001	0.071	0.0
Wait **GPU** NB local	4	5	10001	0.001	0.031	0.0
Wait **GPU** state copy	4	5	21353	0.242	11.594	3.5

GROMACS can use Multiple GPUs

Downloadable images

Especially for GPUs, there are some prebuilt images with which to work

(They can be a lesson in frustration - using pod-gpu you can install pytorch and others in your own python environments)

https://catalog.ngc.nvidia.com/ and, for example, if 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.

Singularity Cloud Library https://cloud.sylabs.io/library
https://cloud.sylabs.io/library/bioinformant/ghru/snp-phylogeny

Docker Hub - https://hub.docker.com/ Lots of AI/ML images

NVIDIA AI - https://catalog.ngc.nvidia.com/ai-foundation-models

Scaleway Docker AI Images
https://www.scaleway.com/en/docs/compute/gpu/reference-content/docker-images/

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/

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.