# Things to do while you are waiting

- Course slides are available at: hprc.tamu.edu/training/aces_containers_techlab.html
- Log into TAMU VPN (if you're off campus)
- Get ready to launch a terminal on the FASTER cluster for interactive exercises (ask if you don't know how).

# HIGH PERFORMANCE RESEARCH COMPUTING

## Introduction to Containers Tech Lab

featuring Charliecloud on the FASTER cluster

an HPRC + LANL Training Collaboration

February 14, 2023

High Performance
Research Computing
DIVISION OF RESEARCH

**Spring 2023**

# Outline

- Connecting to the FASTER Cluster
- Machine Learning with TensorFlow
- Genomics with Clara Parabricks on GPUs
- Molecular Dynamics with LAMMPS on GPUs

# Course Objectives

The researcher should be able to:

- Investigate container repositories
- Build scientific software containers
- Work with data and HPC Resources

# Learning Resources

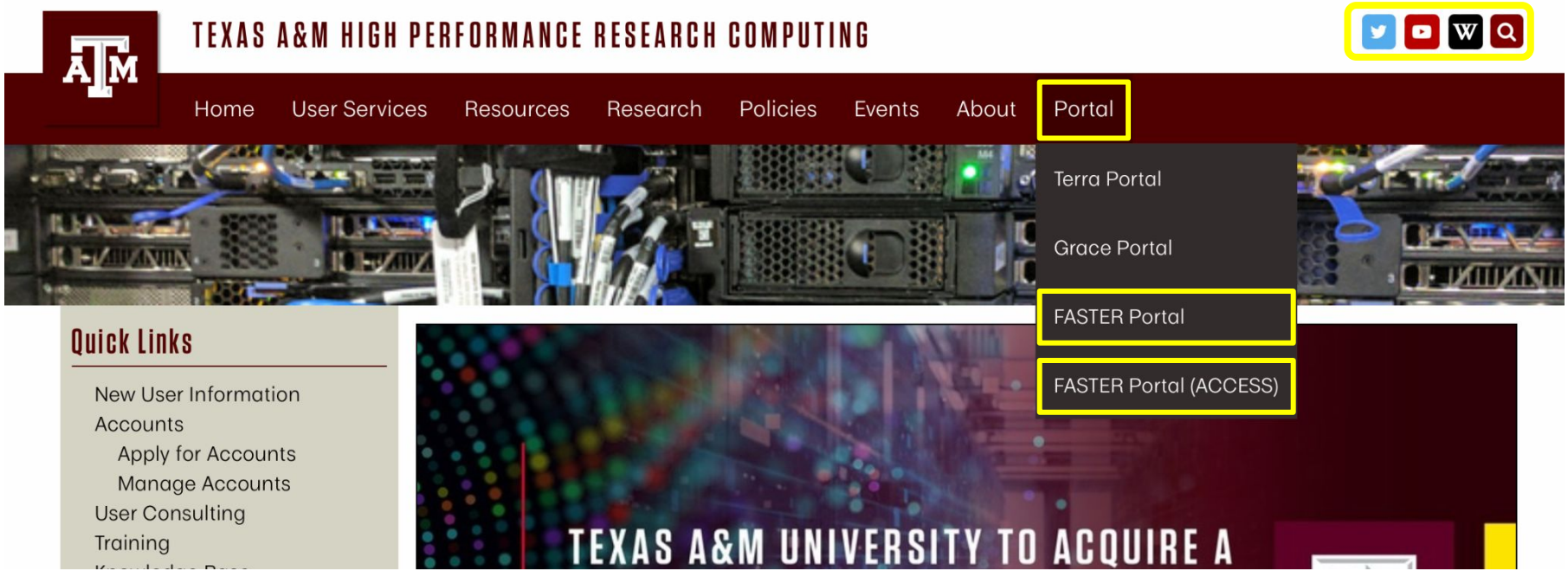
- HPRC Wiki https://hprc.tamu.edu/wiki/SW:Charliecloud
- HPRC on Youtube https://www.youtube.com/c/TexasAMHPRC
- Charliecloud Manual https://hpc.github.io/charliecloud/
- Docker Manual https://docs.docker.com/
- Other container courses:
  NBIS https://nbis-reproducible-research.readthedocs.io/en/latest/singularity/
  Arizona https://learning.cyverse.org/projects/Container-camp-2020/
  TACC https://learn.tacc.utexas.edu/mod/page/view.php?id=95

Exercises coming up next

# Log into FASTER via HPRC Portal

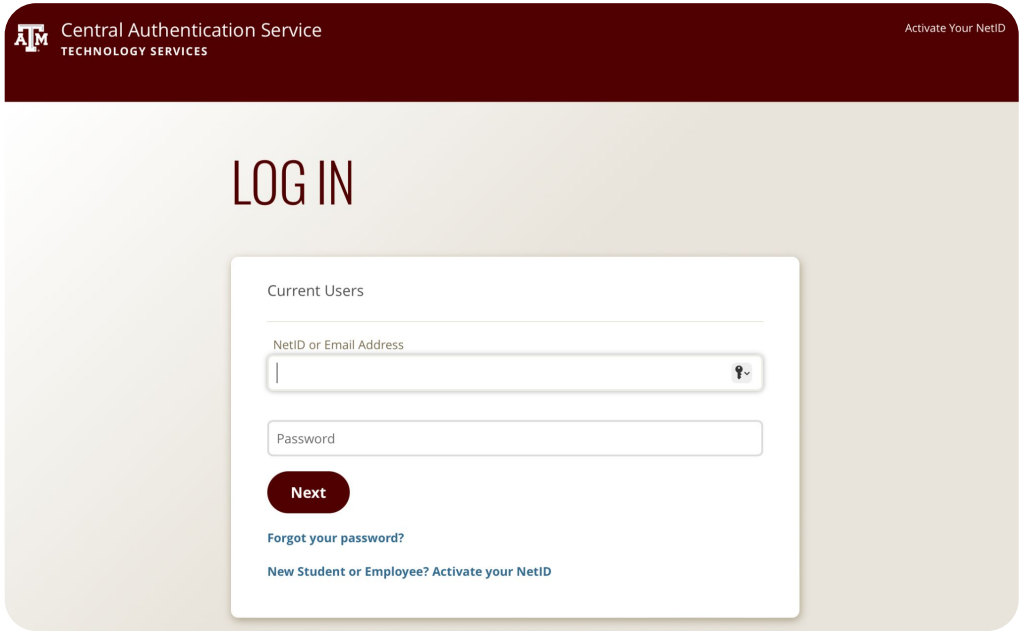
# Accessing the HPRC Portal

- HPRC webpage: hprc.tamu.edu, Portal dropdown menu



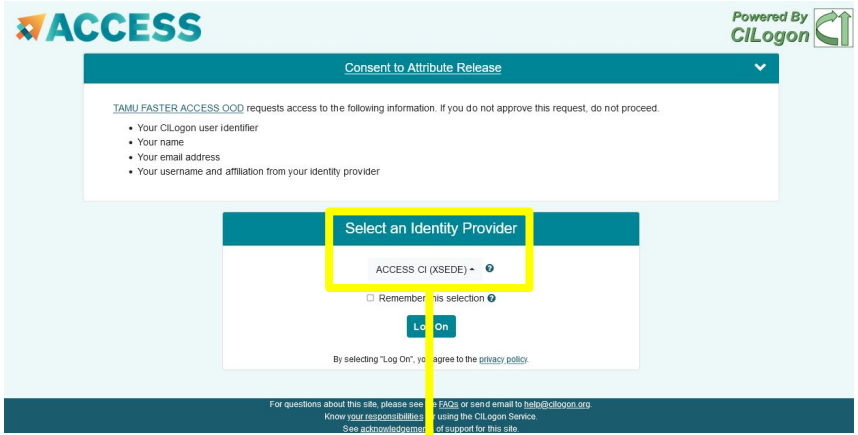
# Accessing FASTER via the HPRC Portal (TAMU)

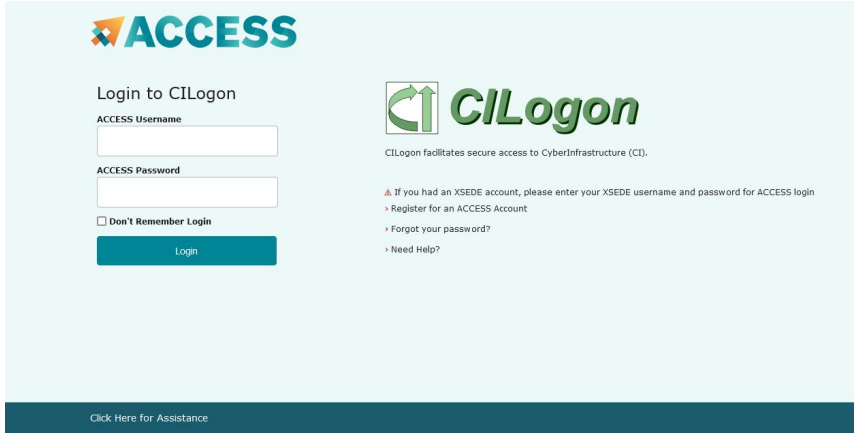
Log-in using your TAMU NetID credentials.

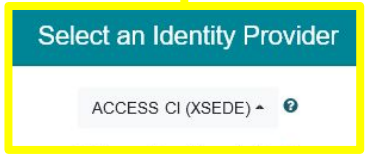


# Accessing FASTER via the HPRC Portal (ACCESS)

Log-in using your ACCESS credentials.



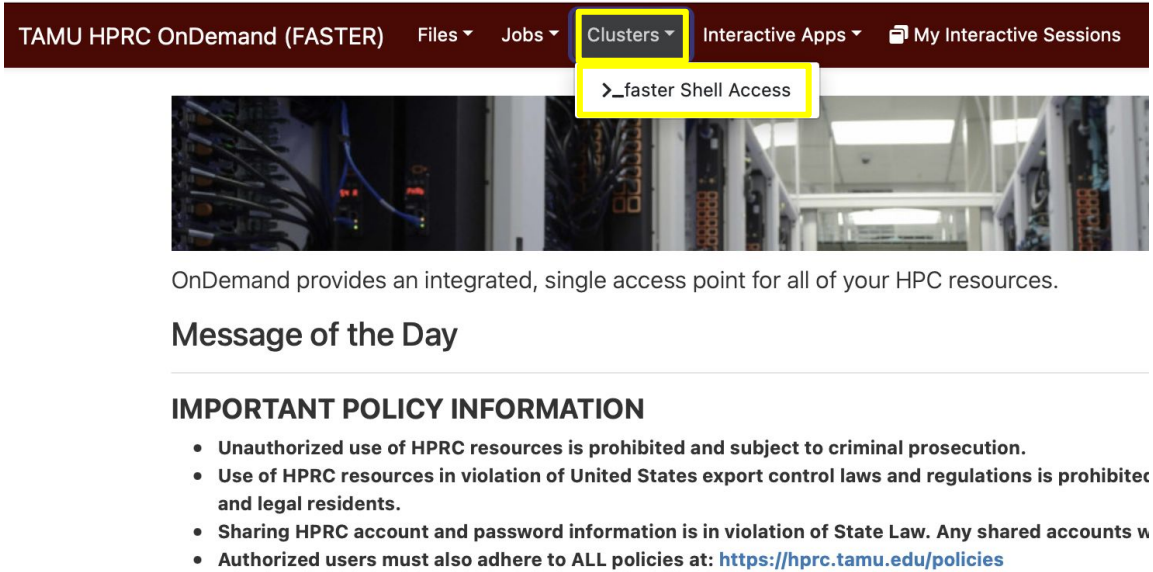




Select the Identity Provider appropriate for your account.

# Shell access via the HPRC Portal

Access through (most) web browsers

–Top Banner Menu “Clusters” -> “Shell Access”



# Training Materials for Charliecloud Tech Lab

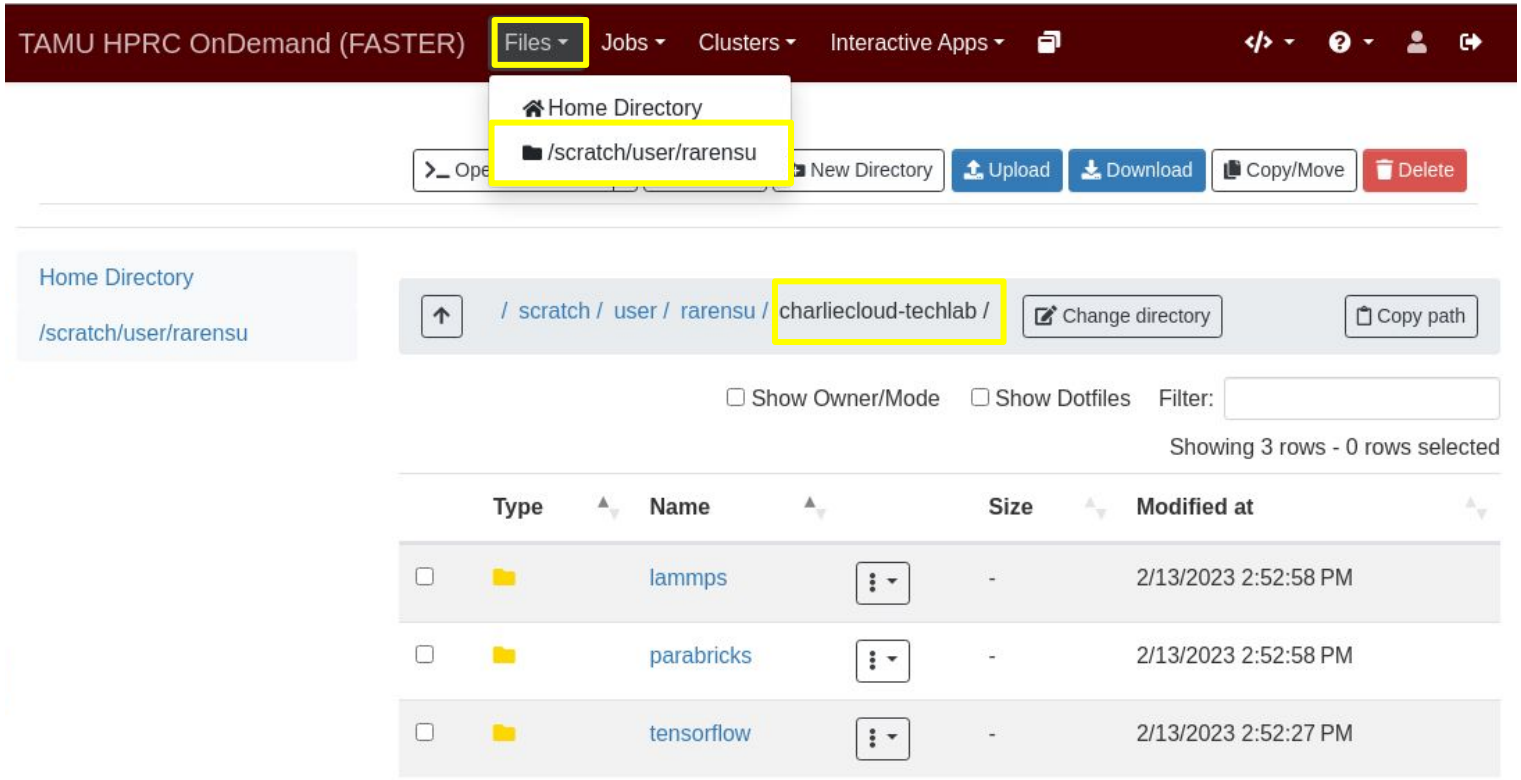
- Copy the exercise materials to your scratch directory:

```
cp -r /scratch/training/charliecloud-techlab $SCRATCH
```

- Navigate to the new exercise directory:

```
cd $SCRATCH/charliecloud-techlab
```

# Training Materials in FASTER Portal



# Machine Learning with TensorFlow

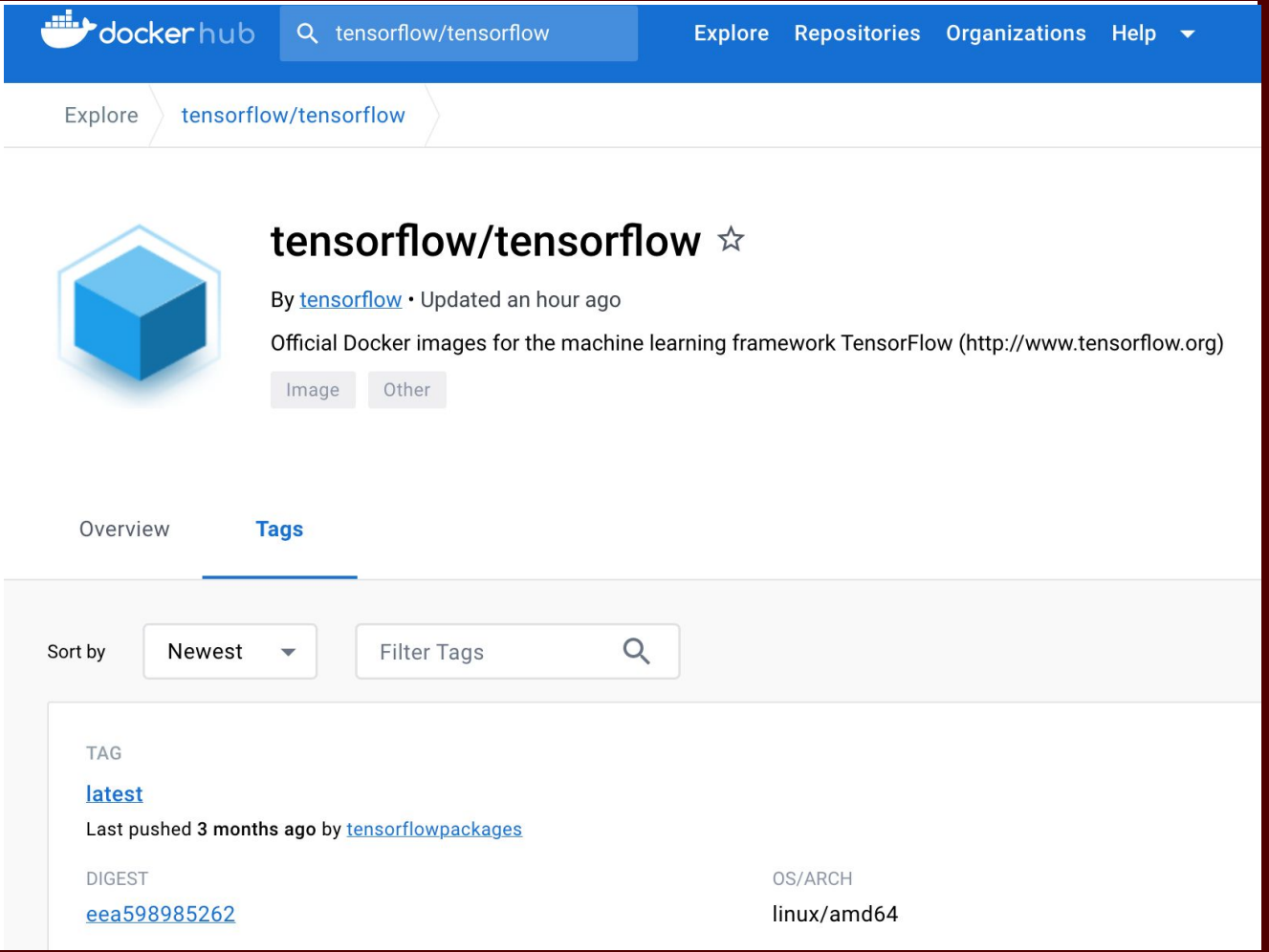
With exercises


# Introduction to TensorFlow


TensorFlow is one of the most popular program frameworks for building machine learning applications.

- Google Brain built **DistBelief** in 2011 for internal usage.
- TensorFlow 1.0.0 was released on Feb 11, 2017
- TensorFlow 2.0 was released in Jan 2018.
- The latest stable version of TensorFlow is 2.10 as of Nov 2022.

# TensorFlow in Docker Hub



# Navigate to the TensorFlow Training Exercises

From the FASTER shell accessed through the HPRC Open OnDemand Portal:

Navigate to the tensorflow exercise directory:

```
module purge
module load charliecloud
cd $SCRATCH/charliecloud-techlab/tensorflow/exercise
```

# Pull a TensorFlow Image

```
$ ch-image pull tensorflow/tensorflow:latest
initializing empty build cache
pulling image:    tensorflow/tensorflow:latest
...
```

```
$ ch-image list
tensorflow/tensorflow:latest
```

```
$ ch-convert tensorflow/tensorflow:latest tensorflow.sqfs
input:   ch-image  tensorflow/tensorflow:latest
output:  squash    tensorflow.sqfs
packing ...
...
```

# Verify to be in the Container

```
$ ch-run tensorflow.sqfs -- python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
>>>
>>> import tensorflow as tf
>>> tf
<module 'tensorflow' from '/usr/local/lib/python3.8/dist-packages/tensorflow/__init__.py'>
```

Press CTRL + D to exit the container

*Explore: Does it work outside the container?*

```
$ python
Python 3.9.7 (default, Sep 16 2021, 13:09:58)
>>>
```

# Verify to be in the Container

*Does it work outside the container? Result:*

```
>>> import tensorflow as tf
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'tensorflow'
```

**So, we were running TensorFlow in container not on the host!**

# Try a Simple TensorFlow Program in Charliecloud

```
$ ch-run tensorflow.sqfs -- python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
>>> import tensorflow as tf
>>> a = tf.constant(2)
>>> b = tf.constant(3)
>>> c = a + b
>>> print('a + b =', c)
a + b = tf.Tensor(5, shape=(), dtype=int32)
```

# 2. Build image from a Dockerfile

```
$ cd tf-simple/
```

```
# Use the official TensorFlow image as the base image
FROM tensorflow/tensorflow:latest

# Copy the current directory to the container
COPY ./tf-example.py /

# Make that file executable
RUN chmod 755 /tf-example.py
```

# Build and Convert

```
$ ch-image build -t tf-example -f Dockerfile .
initializing empty build cache
...
$ ch-image list
tensorflow/tensorflow:latest
tf-example

$ ch-convert tf-example tf-example.sqfs
input:   ch-image  tf-example
output:  squash    tf-example.sqfs
packing ...
```

# Run the script in container

```
$ ch-run tf-example.sqfs -- python ./tf-example.py
```

a + b = tf.Tensor(5, shape=(), dtype=int32)

# Charliecloud Pull Batch Example

```
$ cd ..
$ sbatch tf-job.slurm
```

```
#!/bin/bash

## JOB SPECIFICATIONS
#SBATCH --job-name=cc_pull          #Set the job name to "cc_pull"
#SBATCH --time=01:00:00             #Set the wall clock limit to 1hr
#SBATCH --ntasks=4                  #Request 4 task
#SBATCH --mem=2560M                 #Request 2560MB (2.5GB) per node
#SBATCH --output=cc_pull.%j         #Send stdout/err to "cc_pull.[jobID]"

cd $SCRATCH/charliecloud-techlab/tensorflow/exercise

module load charliecloud
module load WebProxy

# Pull the TF image
ch-image pull tensorflow/tensorflow:latest
```

# Charliecloud CPU Job Batch Example

```
$ cd tf-nn/
$ sbatch tf-nn.slurm
```

```
#!/bin/bash

## JOB SPECIFICATIONS
#SBATCH --job-name=cc_cpu_job          #Set the job name to "cc_cpu_job"
#SBATCH --time=01:00:00                #Set the wall clock limit to 1hr
#SBATCH --ntasks=4                     #Request 4 task
#SBATCH --mem=2560M                    #Request 2560MB (2.5GB) per node
#SBATCH --output=cc_cpu_job.%j         #Send stdout/err to "cc_cpu_job.[jobID]"

cd $SCRATCH/charliecloud-techlab/tensorflow/exercise/tf-nn

module load charliecloud
module load WebProxy

# Build the image from Dockerfile
echo "building the image"
ch-image build -t tf-nn -f Dockerfile  .

# Convert the image to SquashFS format
echo "converting the image to SquashFS format"
ch-convert tf-nn tf-nn.sqfs

# Run the TensorFlow image
echo "Running the TensorFlow image"
ch-run tf-nn.sqfs -- python ./train.py
```

# GPU Jobs with TensorFlow

An optional activity follows.

I will submit a GPU job and observe that it runs. You will not need to understand how the job works.

In the following section, we will learn how to use containers with GPU.

# Charliecloud GPU Job Batch Example

```
$ cd ../tf-gpu/
$ sbatch tf-gpu.slurm
```

```
#!/bin/bash

## JOB SPECIFICATIONS
#SBATCH --job-name=cc_gpu          #Set the job name to "cc_gpu"
#SBATCH --time=01:00:00            #Set the wall clock limit to 1hr
#SBATCH --mem=180G                 #Request 180GB per node
#SBATCH --output=cc_gpu.%j         #Send stdout/err to "cc_gpu.[jobID]"
#SBATCH --gres=gpu:1
#SBATCH --partition=gpu
#SBATCH --cpus-per-task=24

cd $SCRATCH/charliecloud-techlab/tensorflow/exercise/tf-gpu


module load charliecloud
module load nvidia-container-cli/1.11.0-hprc
module load WebProxy
```

# Charliecloud GPU Job Batch Example

```
echo "building the image"
ch-image build -t tf-gpu -f Dockerfile .

echo "converting the image to a directory"
ch-convert tf-gpu $TMPDIR/tf-gpu-dir

echo "Injecting the necessary NVIDIA libraries"
ch-fromhost --nvidia $TMPDIR/tf-gpu-dir

echo "converting the image to SquashFS format"
ch-convert $TMPDIR/tf-gpu-dir tf-gpu.sqfs

echo "Running the TensorFlow image"
ch-run tf-gpu.sqfs -- python /app/train-gpu.py
```

# Monitor GPU Usage

```
Every 1.0s: squeue -u happidence1                                   login1: Sat Feb 11 08:48:26 2023

JOBID        NAME                 USER                 PARTITION            NODES  CPUS  STATE
TIME         TIME_LEFT    START_TIME         REASON                 NODELIST
120734       cc_gpu               happidence1          gpu                  1      24    RUNNING
1:25         58:35        2023-02-11T08:47:0 None                   fc172
```

```
$ ssh <compute-node>
$ watch -n 1 nvidia-smi
```

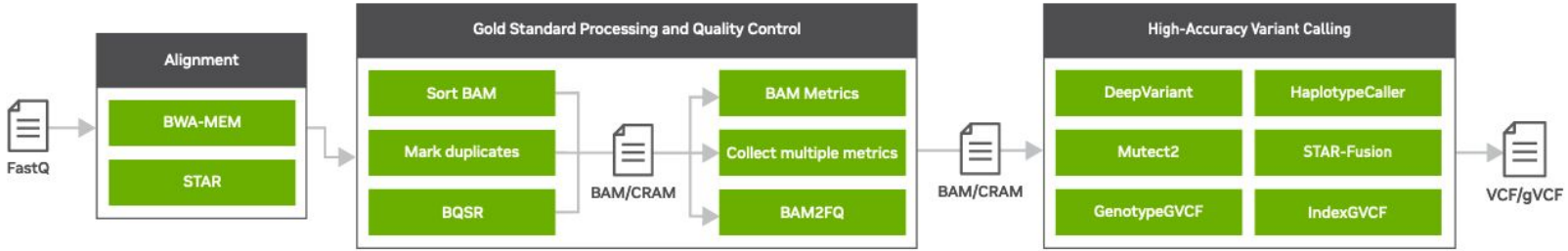
```
Every 1.0s: nvidia-smi

Sat Feb 11 08:50:26 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:1D:00.0 Off |                    Off |
| N/A   29C    P0    46W /  70W |  15651MiB / 16384MiB |     53%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:1E:00.0 Off |                    Off |
| N/A   21C    P8     9W /  70W |      2MiB / 16384MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla T4            On   | 00000000:1F:00.0 Off |                    Off |
| N/A   21C    P8    11W /  70W |      2MiB / 16384MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla T4            On   | 00000000:20:00.0 Off |                    Off |
| N/A   21C    P8     9W /  70W |      2MiB / 16384MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    771345      C   python                           15646MiB |
+-----------------------------------------------------------------------------+
```

# Genomics with Clara Parabricks on GPUs



# Clara Parabricks

- GPU-accelerated version of common bioinformatics pipeline
- Works with both RNA-seq and WGS data
- NVIDIA provides images that containers easily integrate with Charliecloud
- Today's exercise will focus on completing the first portion of the pipeline




NVIDIA Product Sheet:
https://resources.nvidia.com/en-us-genomics-ug-ep/healthcare-genomics-?lx=M-s96I&ncid=em-nurt-521116&mkt_tok=MTU2LU9GTi03NDIAAAGG5gQCuzMHKWvhCg5ODJ9NTi9KCxm57Lxjd5DcahRJvhUUc-g_yTLDcNVB3HBmOyWbGWigpg4yq1h3SK9QNOLnbLU6cm8VhMCHmup4BGcunnUvwRCy#cid=ix09_em-nurt_en-us

# Clara Parabricks

- Massive speed-up versus CPU-only pipelines


Data was generated using publicly available data (https://precision.fda.gov/challenges/truth) for NA12878, deprecating the data to 30X coverage. For the 22-minute runtime, DGX A100 with 320G memory was used. The native GATK4.1 numbers were generated using 32 vCPU (3.1 GHz Intel Xeon® Platinum 8175M) using 320Gb RAM.

NVIDIA Product Sheet:
https://resources.nvidia.com/en-us-genomics-ug-ep/healthcare-genomics-?lx=M-s96I&ncid=em-nurt-521116&mkt_tok=MTU2LU9GTi03NDIAAAGG5gQCuzMHKWvhCg5ODJ9NTi9KCxm57Lxjd5DcahRJvhUUc-q_yTLDcNVB3HBmOyWbGWigpg4yq1h3SK9QNOLnbLU6cm8VhMCHmup4BGcunnUvwRCy#cid=ix09_em-nurt_en-us

# GPUs with Charliecloud

https://hpc.github.io/charliecloud/install.html#running-containers
https://hpc.github.io/charliecloud/ch-fromhost.html#examples
Says “to inject nVidia GPU libraries”:

- nvidia-container-cli ≥ 1.0.0
- nvidia libraries & executables present
- Use `ch-fromhost --nvidia <image in directory format>`

On FASTER cluster:

- nvidia-container-cli is provided as a module.
- Compute nodes with GPUs have matching libraries present.

# Using Charliecloud to run NVIDIA Clara Parabricks

- Containers need to be created on a node with GPUs
- Request an interactive session on a compute node equipped with a GPU:

```
srun --mem=128G --time=01:00:00 --gres=gpu:1 \
    --partition=gpu --reservation=training \
    --cpus-per-task=24 --pty bash -i
```

```
cd $SCRATCH/charliecloud-techlab/parabricks
```

# Using Charliecloud to run NVIDIA Clara Parabricks

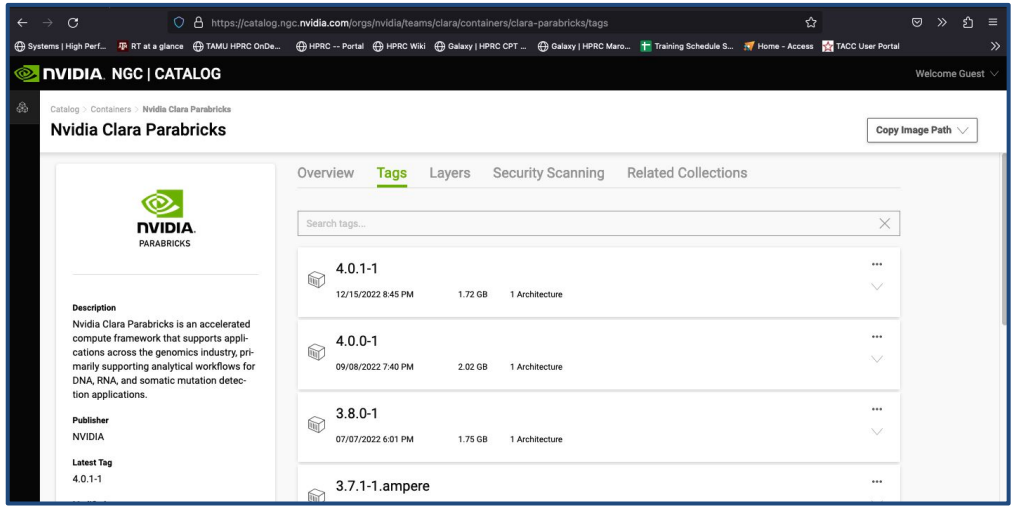
- Load the required modules:

```
# Load the module for Charliecloud
module load charliecloud/0.31

# Load the module we'll need for the NVIDIA libraries
module load nvidia-container-cli/1.11.0-hprc

# Load a module to allow for internet access
module load WebProxy
```

# Using Charliecloud to run NVIDIA Clara Parabricks

- Grab the image from NVIDIA using Charliecloud:

```
ch-image pull nvcr.io/nvidia/clara/clara-parabricks:4.0.1-1 parabricks-4.0.1-1
```



# Using Charliecloud to run NVIDIA Clara Parabricks

- Check for the image that we just pulled:

```
ch-image list
```

- Convert the image to a directory stored on $TMPDIR:

```
ch-convert parabricks-4.0.1-1 $TMPDIR/parabricks4
```

- Inject the necessary NVIDIA libraries (to be able to run on the GPUs):

```
ch-fromhost --nvidia $TMPDIR/parabricks4
```

# Using Charliecloud to run NVIDIA Clara Parabricks

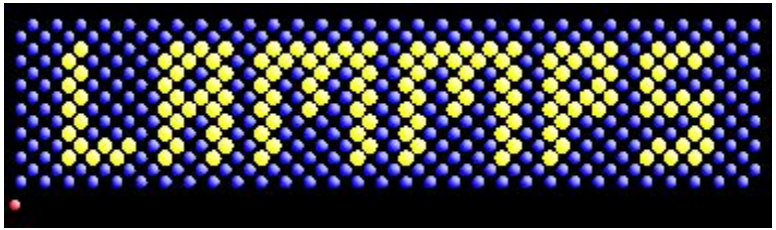
- Convert the container to a SquashFS file

```
ch-convert $TMPDIR/parabricks4 parabricks4.sqfs
```

- We're now ready to run Parabricks!

```
ch-run -b "$PWD:/mnt/1" -c "mnt/1" parabricks4.sqfs pbrun \
    fq2bam -- --ref Homo_sapiens_assembly38.fasta \
    --in-fq sample_1.fastq.gz sample_2.fastq.gz \
    --out-bam test.bam
```

# Molecular Dynamics with LAMMPS on GPUs



# Container Concepts You Need To Know

Some containers set Environment variables at build time.

- In a Dockerfile, use the `ENV` statement to create variables.
- Using `ch-run`, add the `--set-env` flag to load those variables.

Some containers set Environment variables at runtime. This is called a **runscript**.

- Other Container frameworks use Dockerfile `ENTRYPOINT` statements to define this script.
- In a Charliecloud Dockerfile, copy the runscript into the container as a regular file instead (Dockerfile `COPY`).
- Using `ch-run`, execute the runscript from the command line.

# LAMMPS

LAMMPS is a classical molecular dynamics code with a focus on materials modeling. It's an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.

https://www.lammps.org/ has a cool animated logo.

NVIDIA provides GPU-ready container images for lammps.
https://catalog.ngc.nvidia.com/orgs/hpc/containers/lammps

# Inspect Container Images at Home

*Following along live? Do not attempt this.*

Docker method:

```
docker pull nvcr.io/hpc/lammps:29Sep2021up2
docker inspect nvcr.io/hpc/lammps:29Sep2021up2
```

Podman method:

```
podman pull docker://nvcr.io/hpc/lammps:29Sep2021up2
podman inspect nvcr.io/hpc/lammps:29Sep2021up2
```

Find the Entrypoint and Env variables under “Config”.

```
[rarensu@ye-olde-dell ~]$ podman inspect nvcr.io/hpc/lammps:29Sep2021up2
[
    {
      "Config": {
           "Env": [
                "PATH=/usr/local/openmpi/bin:/usr/local/ucx/bin:/usr/local/nvidia/bin:/usr/local/cuc
                "CPATH=/usr/local/knem/include:/usr/local/gdrcopy/include:",
                "LIBRARY_PATH=/usr/local/gdrcopy/lib:",
                "LD_LIBRARY_PATH=/usr/local/cuda/lib:/usr/local/cuda/lib64:/usr/local/fftw/lib:",
                "NVIDIA_DRIVER_CAPABILITIES=compute,utility",
                "NVIDIA_REQUIRE_CUDA=cuda>=11.6 brand=tesla,driver>=460,driver<461 brand=tesla,drive
                "NVIDIA_VISIBLE_DEVICES=all",
                "OMPI_ALLOW_RUN_AS_ROOT=1",
                "OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1",
                "OMPI_MCA_rmaps_base_oversubscribe=1",
                "UCX_MEMTYPE_CACHE=n"
           ],
           "Entrypoint": [
                "/usr/bin/nventry",
                "--build_base_dir=/usr/local/lammps",
                "--build_default=gpu_native"
           ],
           "WorkingDir": "/host_pwd"
      },
```

# Navigate to the LAMMPS Training Exercises

From the FASTER shell accessed through the HPRC Open OnDemand Portal:

Navigate to the lammps exercise directory:

```
cd $SCRATCH/charliecloud-techlab/lammps/
```

*Warning*: it contains the solutions to the exercises.

Alternatively, make your own empty directory and work there:

```
mkdir $SCRATCH/charliecloud-techlab/my-lammps
```

# Inspect Container Images using Singularity

*Following along live? This is optional.*

*On a compute node:*

```
srun --mem=4000m --time=01:00:00 --pty bash -i
```

*Option A (not recommended) download your own image:*

```
export SINGULARITY_CACHEDIR=$TMPDIR/.singularity
module load WebProxy
singularity pull lammps-29Sep2021up2.sif docker://nvcr.io/hpc/lammps:29Sep2021up2
SIF=lammps-29Sep2021up2.sif
```

*Option B (recommended) use the pre-downloaded image:*

```
SIF=/scratch/data/Singularity/images/lammps-29Sep2021up2.sif
```

*Copy the Runscript and Environment:*

```
singularity exec $SIF cp /.singularity.d/runscript .
singularity exec $SIF cp /.singularity.d/env/10-docker2singularity.sh .
exit
```

# Inspect the Runscript

Inspect the runscript we borrowed from Singularity. It corresponds to the Entrypoint reported by Docker/Podman.

```
#!/bin/sh
OCI_ENTRYPOINT='"/usr/bin/nventry" "--build_base_dir=/usr/local/lammps"
"--build_default=gpu_native"'

… # lots of boilerplate code

exec "$@"
```

Optional: also inspect the Environment script.

# Dockerfile

Create a regular file named Dockerfile and add the following text.

```
FROM nvcr.io/hpc/lammps:29Sep2021up2

COPY ./runscript /

RUN chmod 755 /runscript
```

# Getting on a GPU node

```
srun --ntasks=16 --mem=4000m --time=01:00:00 --gres=gpu:1 --partition=gpu --pty bash -i
```

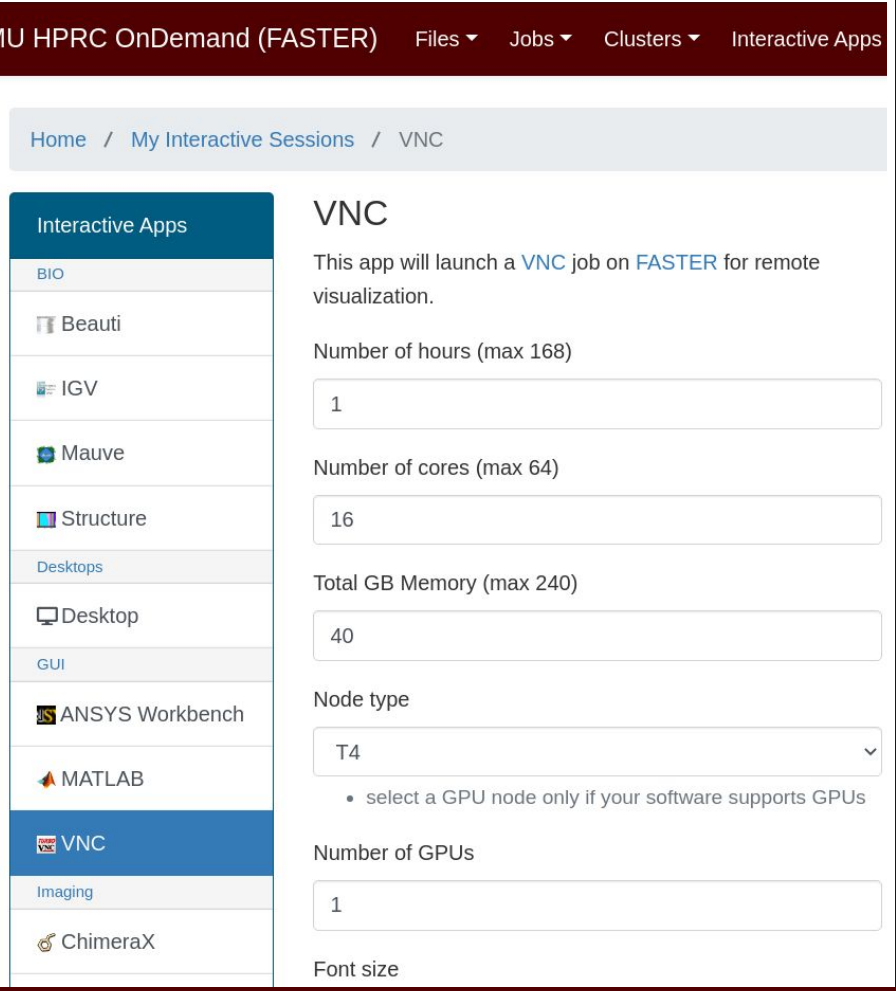
*Following along live? add --reservation=training*

```
module load charliecloud
module load WebProxy
```

```
cd $SCRATCH/charliecloud-techlab/lammps
```
*(or your workdir)*

# Getting a GPU, Alternative

```
# also need to
module load charliecloud
module load WebProxy
```



# Building on a GPU node
# *Challenge*: can you recall the steps?

Fetch image from Repository.

```
ch-image pull nvcr.io/hpc/lammps:29Sep2021up2
```

Build the Dockerfile that adds our runscript to the image.

```
ch-image build -t lammps:29Sep2021up2 .
```

Convert image to Directory format.

```
ch-convert lammps:29Sep2021up2 $TMPDIR/lammps-29Sep2021up2
```

Insert the local NVIDIA libraries.

```
module load nvidia-container-cli
ch-fromhost --nvidia $TMPDIR/lammps-29Sep2021up2
```

Convert image to SquashFS format.

```
ch-convert $TMPDIR/lammps-29Sep2021up2 lammps-29Sep2021up2.sqfs
```

# Testing if LAMMPS is installed

*(still on the GPU node, of course)*

```
ch-run --set-env lammps-29Sep2021up2.sqfs -- /runscript mpirun lmp -h
```

`mpirun` is used to execute LAMMPS to work around a problem with `srun`.
`lmp` is the LAMMPS executable.

Quiz: What does `/runscript` do?

# LAMMPS Benchmark

*These files are also found in the Training Materials you copied.*

NVIDIA provides a benchmarking script for their container. On the same page, find:

"An example Slrum [sic] batch script that may be modified for your specific cluster setup may be viewed here."

Copy the last line in a file named **benchmark.sh** and edit slightly

```
mpirun lmp -k on g 1 -sf kk -pk kokkos cuda/aware on neigh full comm device \
binsize 2.8 -var x 4 -var y 4 -var z 4 -in /host_pwd/in.lj.txt
```

*Recommended: set* `${gpus_per_node}` *to* `1` *or replace it.*
*Recommended: change* `-var xyz 8` to `-var xyz 4` *to prevent out-of-memory error.*
*Recommended: insert* `mpirun` *to workaround a problem with srun.*

Download **in.lj.txt**

```
wget https://lammps.sandia.gov/inputs/in.lj.txt
```

# LAMMPS GPU benchmark

*(still on gpu node, of course)*

Test if `/host_pwd` exists in container. *(Needed for the benchmark.sh to work correctly.)*

```
ch-run lammps-29Sep2021up2.sqfs -- ls /host_pwd
```

*(it exists, and it's empty)*

Apply the environment variables.
Bind mount `host_pwd` so we can use our local files.
Execute our benchmark script.

```
ch-run --set-env -b "$PWD:/host_pwd" -c /host_pwd lammps-29Sep2021up2.sqfs -- /runscript
bash benchmark.sh
```

# Tech Lab Complete

# Conclusion

- Run Containers on clusters! Take control of your software.
- HPRC supports Charliecloud.
- Convert Docker to Charliecloud!
- Ask for help!

# Questions

# Learning Resources

- HPRC Wiki https://hprc.tamu.edu/wiki/SW:Charliecloud
- HPRC on Youtube https://www.youtube.com/c/TexasAMHPRC
- Charliecloud Manual https://hpc.github.io/charliecloud/
- Docker Manual https://docs.docker.com/
- Other container courses:
  NBIS https://nbis-reproducible-research.readthedocs.io/en/latest/singularity/
  Arizona https://learning.cyverse.org/projects/Container-camp-2020/
  TACC https://learn.tacc.utexas.edu/mod/page/view.php?id=95

# Thank you

Contact: help@hprc.tamu.edu