# ALCF Data and Learning Frameworks

J. Taylor Childers
ALCF

# Data & Learning at the ALCF

Prasanna Balaprakash

Taylor Childers

Elise Jennings

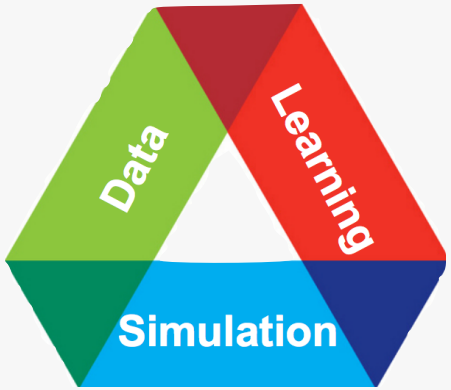
Xiao-Yong Jin

Murat Keceli

Jin Li

Alvaro Vazquez Mayagoitia

Adrian Pope




Misha Salim

William Scullin

Ganesh Sivaraman

Francois Tessier

Tom Uram

Venkat Vishwanath

Richard Zamora

# Data & Learning Frameworks for Theta

- Deep Learning:
  - Tensorflow+Keras
  - Horovod
  - Cray ML Plugin for Deep Learning
- Data Handling:
  - Spark
  - Singularity
  - Globus
  - RAM-disk (/tmp)
  - SSDs (Rick's talk)

# Deep Learning on Theta

- Support at ALCF has been focused on Tensorflow with the optional Keras API
- Intel offers an optimized Tensorflow wheel
- Using Horovod for scaling across nodes using data parallelism

- We have two supported Tensorflow installs:
  - Conda environment using Intel Tensorflow Wheel
  - Cray optimized ML plugin
- Both options have methods for running data-parallel training on Theta
  - Data-parallel means each node has a full ML model and trains on mini-batches of input data
  - After gradients are calculated locally, an ALLREDUCE is performed to compute a global gradient and synchronize the model parameters across nodes

# Tensorflow Installations: Conda Environment

- A Conda environment is available and can be loaded using the **`module load`** command
  - `module load miniconda-3.6/conda-4.4.10` (python 3.6)
  - `module load miniconda-2.7/conda-4.4.10` (python 2.7)
  - Can query the local packages installed using **`conda list`**
  - Uses Intel optimized backends such as numpy and scipy to provide better performance
  - Tensorflow installed via Intel Wheel
  - Documented here: https://www.alcf.anl.gov/user-guides/conda
- Use it this way:

```
#!/bin/bash
#COBALT -n <num-nodes>
#COBALT -t <wall-time>
#COBALT -q <queue>
#COBALT -A <project>

module load miniconda-3.6/conda-4.4.10

aprun -n <num-ranks> -N <mpi-ranks-per-node> python script.py
```

# Tensorflow Installations: Conda Environment

- If you need to install custom modules you can clone the installation
- Be aware that the Conda installations have the Cray MPI libs copied into their **`./lib`** areas to ensure compatibility with Theta

```
conda create -p /path/to/new/env --clone /soft/datascience/conda/miniconda3/4.4.10
```

- This will clone the installation to your own area.

```
source activate /path/to/new/env
```

- Then you can install other python modules using
  - `conda install`
  - `pip install`
- Removing the **`--clone`** would provide you with a clean environment with nothing installed.

# Tensorflow Installations: Conda Environment

- Using this in a submit script

```
#!/bin/bash
#COBALT -n <num-nodes>
#COBALT -t <wall-time>
#COBALT -q <queue>
#COBALT -A <project>

module load miniconda-3.6/conda-4.4.10
source activate /path/to/new/env

aprun -n <num-ranks> -N <mpi-ranks-per-node> python script.py
```

# Tensorflow Installations: Cray Plugin

- More details in Peter Mendygral's Slides
- Communication plugin with Python and C APIs
- Optimized for TensorFlow but also portable to other frameworks
  - Callable from C/C++ source
  - Called from Python if data stored in NumPy arrays or Tensors
- Like Horovod does not require modification to TensorFlow source
  - User modifies training script
- Uses custom ALLREDUCE specifically optimized for DL workloads
  - Optimized for Cray Aries interconnect and IB for Cray clusters
- Tunable through API and environment variables
- Supports multiple gradient aggregations at once with thread teams
  - Useful for Generative Adversarial Networks (GAN), for example
- Example submit scripts here:

```
/lus/theta-fs0/projects/SDL_Workshop/mendygra/cpe_plugin_py2.batch
/lus/theta-fs0/projects/SDL_Workshop/mendygra/cpe_plugin_py3.batch
```

# Tensorflow Installations: Cray Plugin

- The Cray Python environment can be loaded via
- Environment setup for Python 2.7:

```
module load cray-python
export PYTHONUSERBASE=/lus/theta-fs0/projects/SDL_Workshop/mendygra/pylibs
module load /lus/theta-fs0/projects/SDL_Workshop/mendygra/tmp_inst/modulefiles/craype-ml-plugin-py2/1.1.0
```

- Environment setup for Python 3.6

```
module load cray-python/3.6.1.1
export PYTHONUSERBASE=/lus/theta-fs0/projects/SDL_Workshop/mendygra/pylibs
module load /lus/theta-fs0/projects/SDL_Workshop/mendygra/tmp_inst/modulefiles/craype-ml-plugin-py3/1.1.0
```

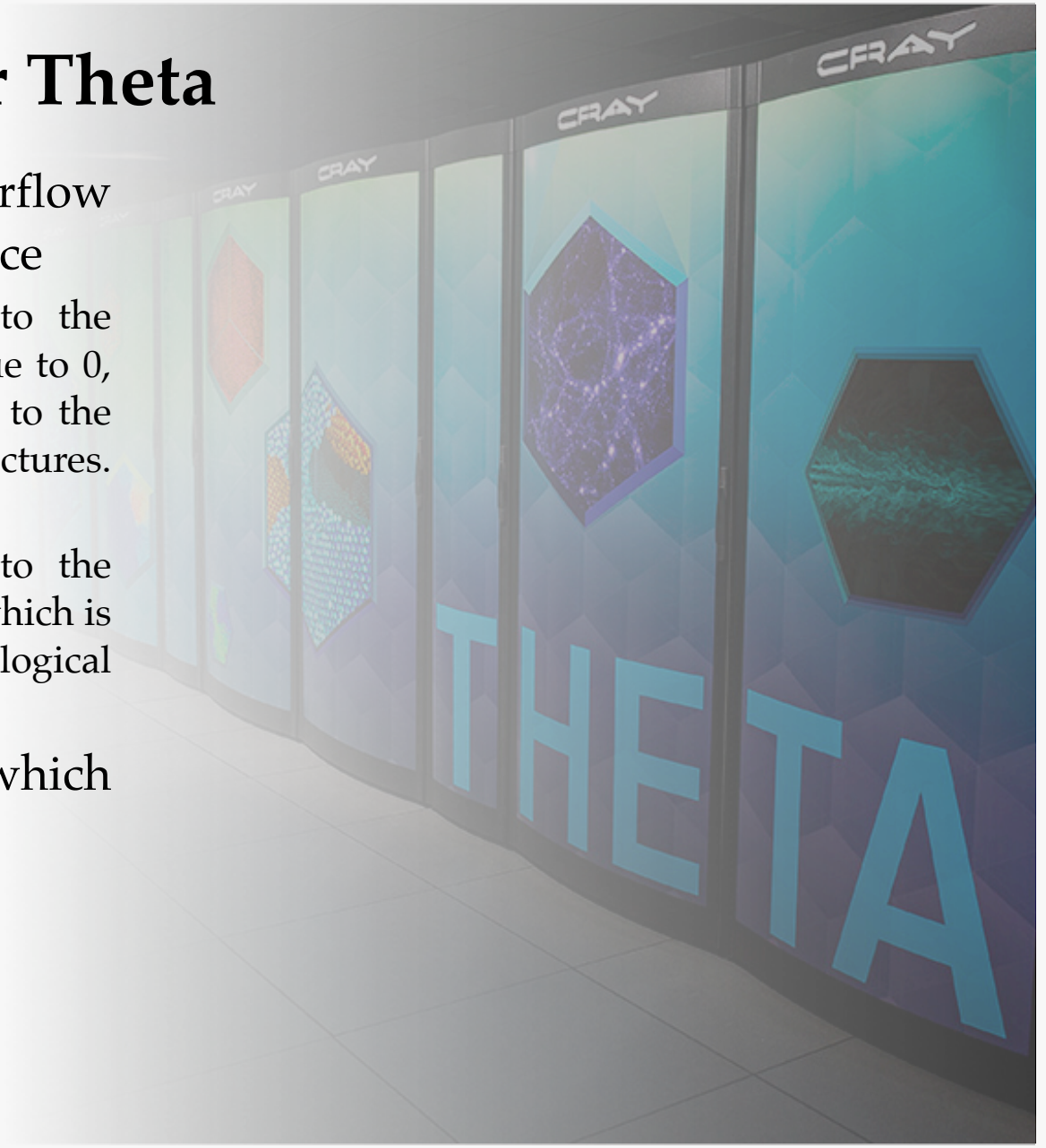
# Environment Customizations for Theta

```
#!/bin/bash
#COBALT -n <num-nodes>
#COBALT -t <wall-time>
#COBALT -q <queue>
#COBALT -A <project>

# load your environment
module load ...

# from Peter Mendygral
# Specifies the number of threads to use.
OMP_NUM_THREADS=62
# milliseconds a thread waits after completing
# the execution of a parallel region, before sleeping.
KMP_BLOCKTIME=0 # 30 sometimes good too
# Enables the run-time library to bind threads to physical processing units.
KMP_AFFINITY="granularity=fine,compact,1,0"

aprun -n <num-ranks> -N <mpi-ranks-per-node> python script.py
```

- Submit script should include the environment variables below
- Some insight into these settings is here: https://www.tensorflow.org/performance/performance_guide

# Environment Customizations for Theta

- In general, you can play with the Tensorflow configuration for threading to optimize performance
  - **`intra_op_parallelism_threads`**: Setting this equal to the number of physical cores is recommended. Setting the value to 0, which is the default and will result in the value being set to the number of logical cores, is an option to try for some architectures. **This value and OMP_NUM_THREADS should be equal.**
  - **`inter_op_parallelism_threads`**: Setting this equal to the number of sockets is recommended. Setting the value to 0, which is the default, results in the value being set to the number of logical cores.
- There is an example TF CNN implementation which implements these via command line flags here:
  - https://github.com/tensorflow/benchmarks/blob/mkl_experiment/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py



# Environment Customizations for Theta

- In general, you
  configuration for th
  - **intra_op_paralle**
    number of physical c
    which is the default
    number of logical co
    **This value and OMP**
  - **inter_op_paralle**
    number of sockets is recommended. Setting the value to 0, which is
    the default, results in
    cores.
- There is an examp
  implements these via command line flags here:
  - https://github.com/tensorflow/benchmarks/blob/mkl_experiment/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py

```
def create_config_proto():
  config = tf.ConfigProto()
  config.allow_soft_placement = True
  config.intra_op_parallelism_threads = FLAGS.num_intra_threads
  config.inter_op_parallelism_threads = FLAGS.num_inter_threads
  config.gpu_options.force_gpu_compatible = FLAGS.force_gpu_compatible
  #config.graph_options.rewrite_options.disable_model_pruning = True
  return config
```

```
self.server = tf.train.Server(self.cluster, job_name=self.job_name,
                              task_index=self.task_index,
                              config=create_config_proto(),
                              protocol=FLAGS.server_protocol)
```

# Filesystem Customizations for Theta

- Use Lustre striping to improve filesystem performance during training
- First create a directory that will be striped across multiple Lustre sources

```
lfs setstripe –c 16 [samples directory]
```

- Then copy the input files into this directory

```
cp [dataset files] [samples directory]
```

# Scaling Tensorflow on Theta with Horovod

- https://github.com/uber/horovod
- Horovod is part of the Conda environment when setup
- Horovod is a simple wrapper using MPI to synchronize gradients prior to updating model parameters
- It has support for native Tensorflow or Keras with Tensorflow as the backend

# Scaling Tensorflow on Theta with Horovod

```
import keras
# ...
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
#... data loading, etc. ....
# create model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
# ...
# create optimizer
opt = keras.optimizers.Adadelta()
# wrap with Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)
# pass horovod optimizer instead of keras optimizer to model compilation step
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

- Easiest implementation using Keras + Tensorflow
- For Keras one can simply add the code above

# Scaling Tensorflow on Theta with Horovod

- In the case of Keras, one can set the Tensorflow threading options in this way.

```
import keras,os
import tensorflow as tf
config = tf.ConfigProto(intra_op_parallelism_threads=os.environ('OMP_NUM_THREADS'), \
                        inter_op_parallelism_threads=2, \
                        allow_soft_placement=True, \
                        device_count = {'CPU': args.jobs})
session = tf.Session(config=config)
keras.backend.set_session(session)
```

# Scaling Tensorflow on Theta with Horovod

- Horovod can also be added to a native Tensorflow training script
- https://github.com/uber/horovod/blob/master/examples/tensorflow_mnist.py
- This requires a few more edits

```
import tensorflow as tf
import horovod.tensorflow as hvd
# . . . helper functions . . .
def main(_):
    # Horovod: initialize Horovod.
    hvd.init()
    # Download and load MNIST dataset.
    mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())
    # . . . build model . . .
    # Horovod: adjust learning rate based on number of GPUs.
    opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
    # Horovod: add Horovod Distributed Optimizer.
    opt = hvd.DistributedOptimizer(opt)
    # . . . build train_op . . .
```

# Scaling Tensorflow on Theta with Horovod

- Horo
- https nist.py
- This

```
hooks = [
    # Horovod: BroadcastGlobalVariablesHook broadcasts initial variable states
    # from rank 0 to all other processes. This is necessary to ensure consistent
    # initialization of all workers when training is started with random weights
    # or restored from a checkpoint.
    hvd.BroadcastGlobalVariablesHook(0),

    # Horovod: adjust number of steps based on number of nodes.
    tf.train.StopAtStepHook(last_step=20000 // hvd.size()),

    tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss},
                               every_n_iter=10),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from
# corrupting them.
checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None


# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                       hooks=hooks,
                                       config=config) as mon_sess:
    while not mon_sess.should_stop():
        # Run a training step synchronously.
        image_, label_ = mnist.train.next_batch(100)
        mon_sess.run(train_op, feed_dict={image: image_, label: label_})
```

# Cray Plugin Example

- After **`module load`** setup from Slide 9
- See some example scripts:

```
less $CRAYPE_ML_PLUGIN_BASEDIR/examples/tf_mnist/mnist.py
```

- Import the Cray plugin in your code:

```
import tensorflow as tf
# load Cray plugin
import ml_comm as mc
# ...
```

# Cray Plugin Example

- Must tell the Cray plugin the number of trainable parameters in your model for memory alloc
- This is the initialization step

```
# CRAY ADDED
if FLAGS.enable_ml_comm:
  # initialize the Cray PE ML Plugin (assume 20M variables max)
  mc.init(1, 1, 20*1024*1024, "tensorflow")
  # config the thread team (correcting the number of epochs for the effective batch size))
  FLAGS.train_epochs = int(FLAGS.train_epochs / mc.get_nranks())
  max_steps = int(math.ceil(FLAGS.train_epochs *
                  (_NUM_IMAGES['train'] + _NUM_IMAGES['validation']) / FLAGS.batch_size))
  mc.config_team(0, 0, 100, max_steps, 2, 200)
  # give each rank its own directory to save in
  FLAGS.model_dir = FLAGS.model_dir + '/rank' + str(mc.get_rank())
```

- This is the finalization step

```
# CRAY ADDED
if FLAGS.enable_ml_comm:
  mc.finalize()
# END CRAY ADDED
```

# Cray Plugin Example

- Update Optimizer to synchronize gradients and apply

```
# CRAY ADDED
if FLAGS.enable_ml_comm:
  # we need to split out the minimize call below so we can modify gradients
  grads_and_vars = optimizer.compute_gradients(loss)
  grads     = mc.gradients([gv[0] for gv in grads_and_vars], 0)
  gs_and_vs = [(g,v) for (_,v), g in zip(grads_and_vars, grads)]
  train_op = optimizer.apply_gradients(gs_and_vs,
                                       global_step=tf.train.get_or_create_global_step())
# END CRAY ADDED
```

# Cray Plugin Example

- Additional initialization:

```
# CRAY ADDED
# since this script uses a monitored session, we need to create a hook to initialize
# variables after the session is generated
class BcastTensors(tf.train.SessionRunHook):

  def __init__(self):
    self.bcast = None

  def begin(self):
    if not self.bcast:
      new_vars   = mc.broadcast(tf.trainable_variables(),0)
      self.bcast = tf.group(*[tf.assign(v,new_vars[k]) for k,v in enumerate(tf.trainable_variables())])

  def after_create_session(self, session, coord):
     session.run(self.bcast)

     if FLAGS.ml_comm_validate_init:
       py_all_vars = [session.run(v) for v in tf.trainable_variables()]
       if (mc.check_buffers_match(py_all_vars,1) != 0):
         print("ERROR: not all processes have the same initial model!")
       else:
         print("Initial model is consistent on all ranks")

# END CRAY ADDED
```

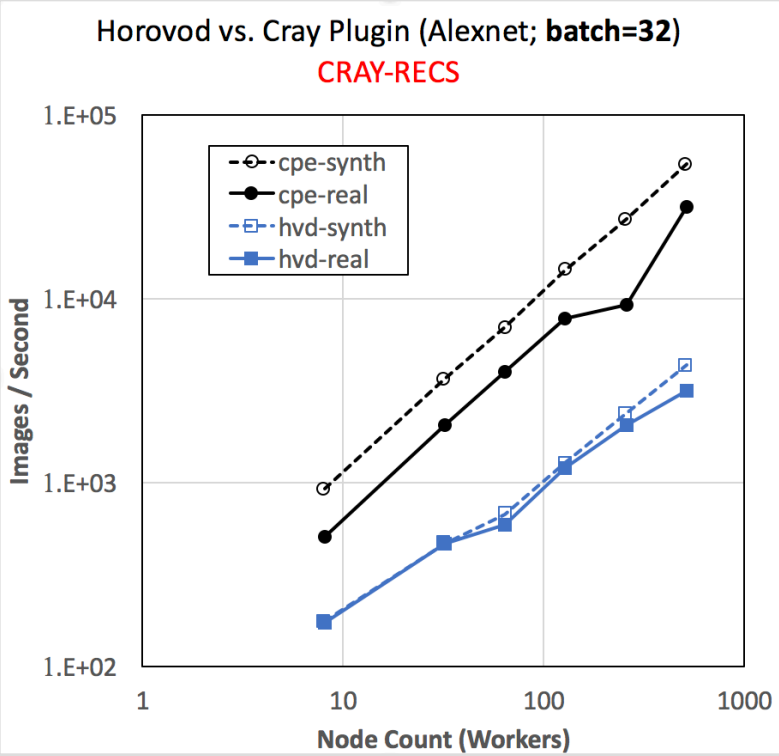
# Cray Plugin Example

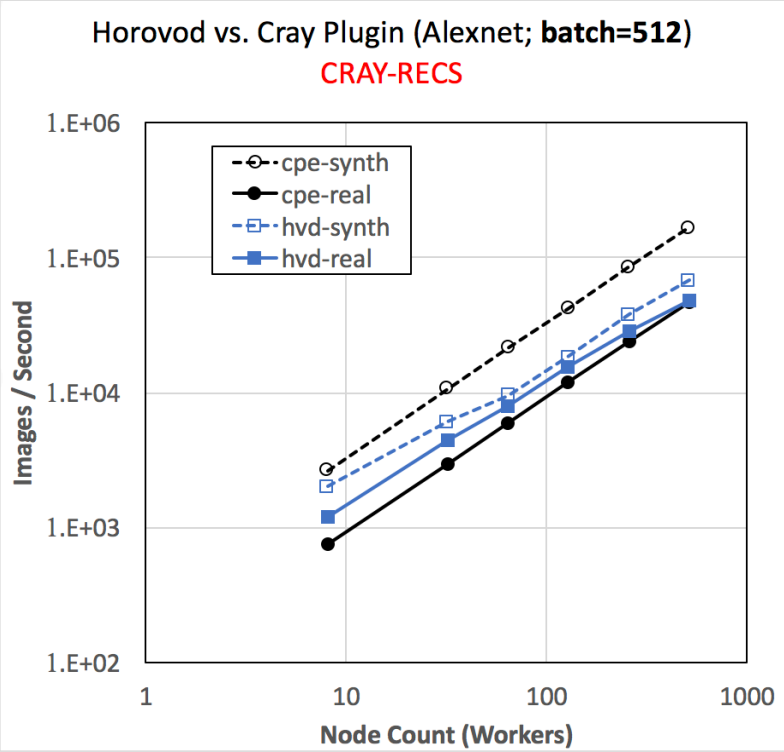
- Additional initialization:

```
# CRAY ADDED
# since this script uses a monitored session, we need to create a hook to initialize
# variables after the session is generated
class BcastTensors(tf.train.SessionRunHook)

  def __init__(self):
    self.bcast = None

  def begin(self):
    if not self.bcast:
      new_vars    = mc.broadcast(tf.trainabl
      self.bcast = tf.group(*[tf.assign(v,n

  def after_create_session(self, session, c
     session.run(self.bcast)

     if FLAGS.ml_comm_validate_init:
       py_all_vars = [session.run(v) for v in tf.trainable_variables()]
       if (mc.check_buffers_match(py_all_vars,1) != 0):
         print("ERROR: not all processes have the same initial model!")
       else:
         print("Initial model is consistent on all ranks")

# END CRAY ADDED
```

```
# CRAY ADDED
# add to our list of session hooks for the initial bcast of the model
sess_hooks = []
if FLAGS.enable_ml_comm:
  sess_hooks = [BcastTensors()]
# END CRAY ADDED
# ...
tf.estimator.EstimatorSpec(
      mode=mode,
      predictions=predictions,
      loss=loss,
      train_op=train_op,
      training_hooks=sess_hooks,
      eval_metric_ops=metrics)
```

# Cray Vs. Horovod Performance

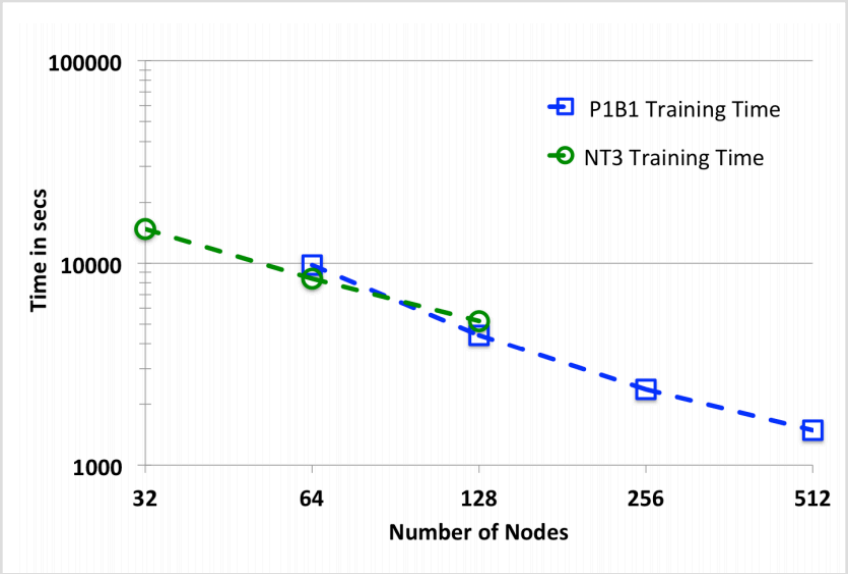


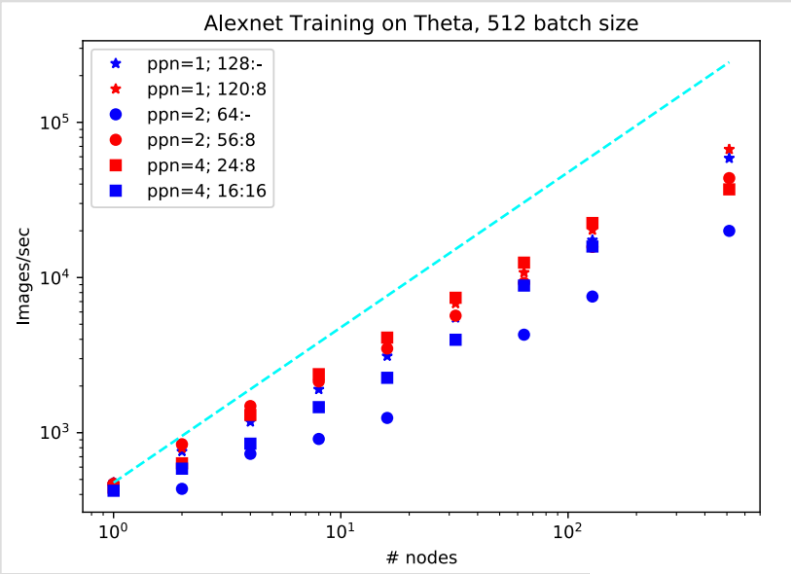




- Scaling results comparing Horovod+TF in Conda vs Cray ML Plugin
- Images processed per second
- **Left** uses local mini-batch size of 32
- **Right** uses local mini-batch size of 512

- Cray plugin outperforms Horovod in the high-communication region.

# Some Horovod Performance Measures




- **A nice example script is located here which abstracts all the features described and more:**

```
/projects/datascience/elise/helper_scripts/tf_wrapper.py
```

- **Example batch script using this is here:**

```
/projects/datascience/elise/TF_alexnet.sh
```

- **On the Left**
- Testing with Horovod+TF using data parallel training
- Scaled data-parallel training for two Candle benchmarks to 512 nodes on Theta

- **On the Right**
- Alexnet training example using different numbers of processes per node (ppn) and total node count
- Inter/Intra Op Thread settings varied as well.
- Shows near linear strong scaling

# Monitoring With Tensorboard

- You can monitor training variables using Tensorboard on Theta

```
module load miniconda-3.6/conda-4.4.10
tensorboard --logdir </path/to/checkpointdir>
```

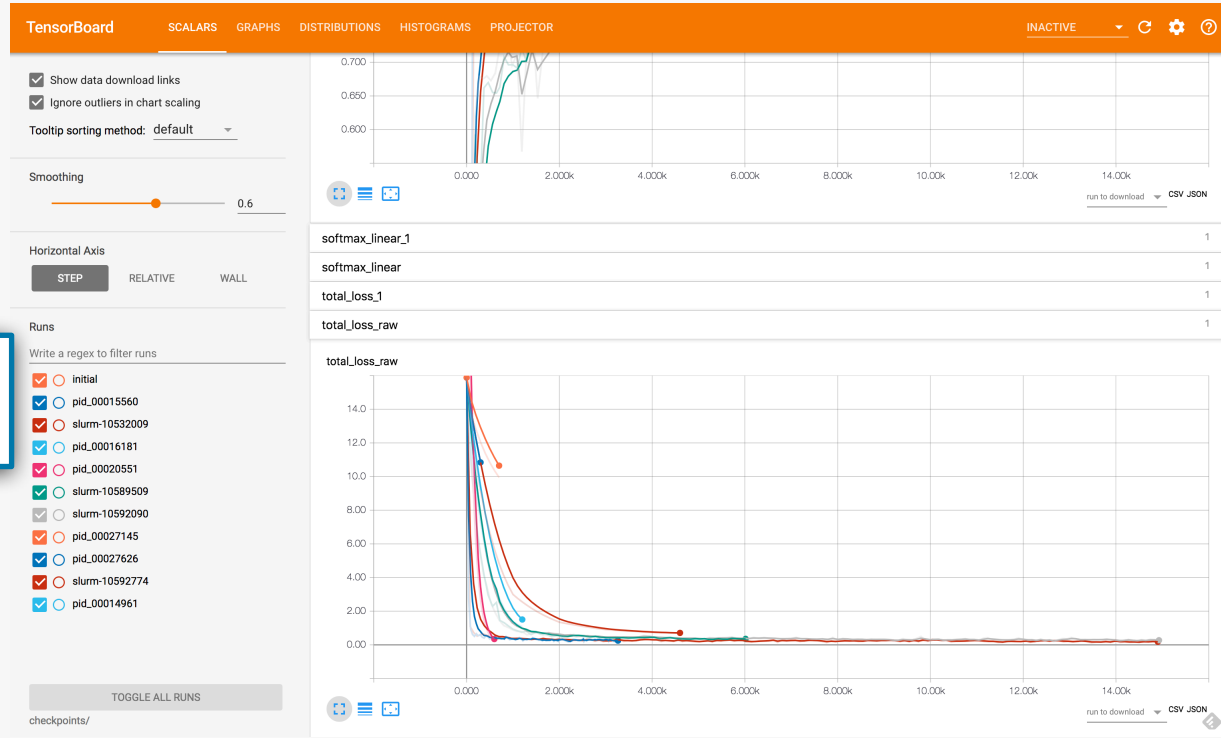
- After it starts you will see something like this

```
TensorBoard 1.6.0 at http://thetalogin5:6006
(Press CTRL+C to quit)
```

- You can connect by port forwarding when you login to Theta:

```
ssh -D <some-high-port-number> theta.alcf.anl.gov
```

- On your laptop, in Firefox, you can set the browser to use a socks5 proxy 'localhost' with the same port number you used above
- Then enter **`thetalogin5:6006`** as the url



# Running Spark on Theta

- What is Spark?
  - Method for data-parallel applications to scale easily on HPCs
- Installed on Theta, can run your Spark-enabled applications using this recipe:

```
/soft/datascience/Spark_Job/submit-spark.sh -A <project> -t 10 \
                -n <wall-time> -q <queue> run-example SparkPi
```

- Still working on documentation on website and standardizing the installation on Theta
- Currently benchmarking to understand proper configurations and use-cases

# Containers on Theta with Singularity

- We use Singularity due to the rights escalation issue in Docker
- https://www.alcf.anl.gov/user-guides/singularity
- Available on Theta login nodes for downloading images
- Images can be built using

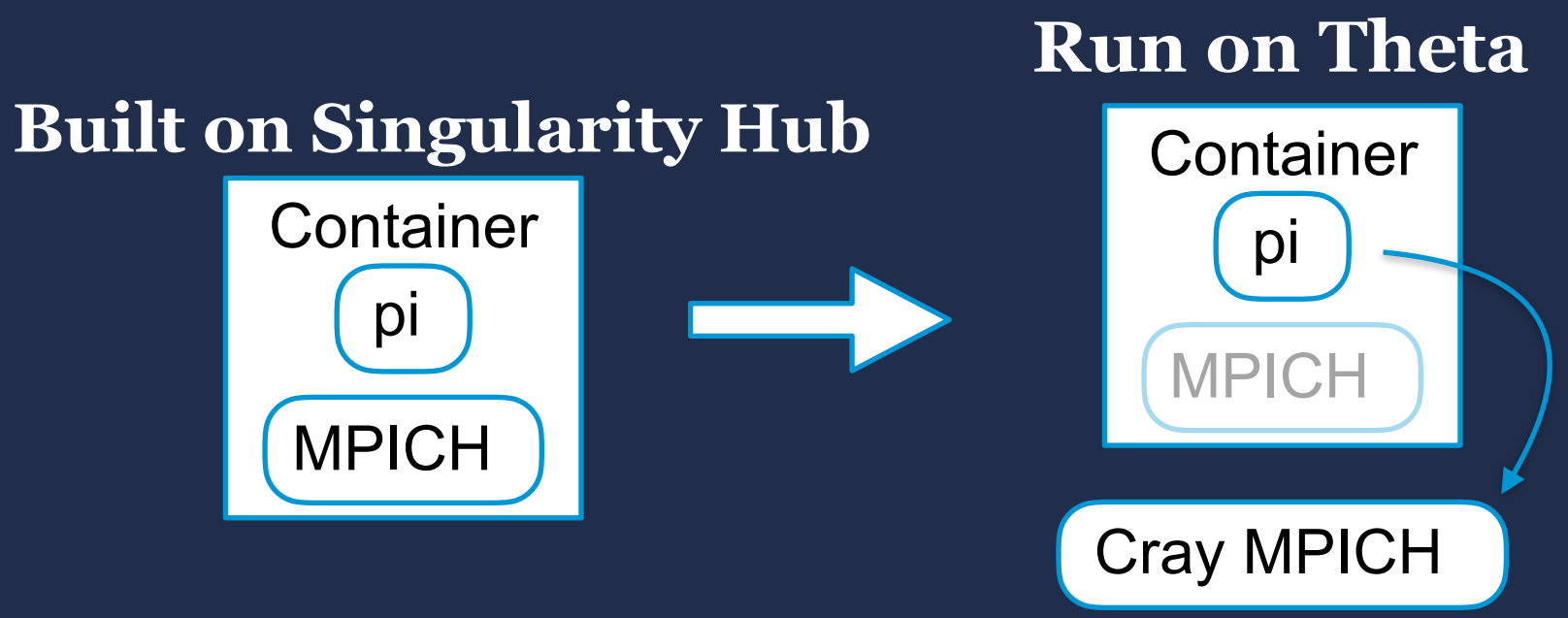
```
singularity build myubuntu.img docker://ubuntu
singularity build myubuntu.img shub://singularityhub/ubuntu
singularity build myubuntu.img docker://jtchilders/mpitest:latest
```

- Generally the Singularity build command requires 'sudo' rights to run except in these cases where you have an image already on a HUB
- The following instructions show how to build an Singularity container on the Singularity Hub

http://singularity.lbl.gov/

# Overview of the Workflow in Five Easy Steps!

1. Create SingularityFile recipe in github
2. Link repo to Singularity Hub
3. Wait for build
4. Build on Theta
5. Run on Theta



# Singularity Usage on Theta

- Building containers from Scratch
- Create a Singularity recipe file

```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

# Source of base image

```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

Source of base image

Make working directory.
Copy files from into image.

During the ‘setup’ phase, the image does not yet exist and is still on the host filesystem at the path `SINGULARITY_ROOTFS`
This creates app directory at ‘/myapp’ in the image

```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

Source of base image

Make working directory. Copy files from into image.

Commands to install my image with the application.

Install via 'yum' any packages needed to build application inside the container. Build MPICH by hand, then builds application.

```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

Source of base image

Make working directory.
Copy files from into image.

Commands to install my image with the application.

Typically containers are built to run one executable.

```
singularity run myapp.img
```

Specify the executable to run with container is called

```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

pi.c source is here: https://www.alcf.anl.gov/user-guides/example-program-and-makefile-bgq
It's a straightforward MPI application that calculates pi with MPI_REDUCE.

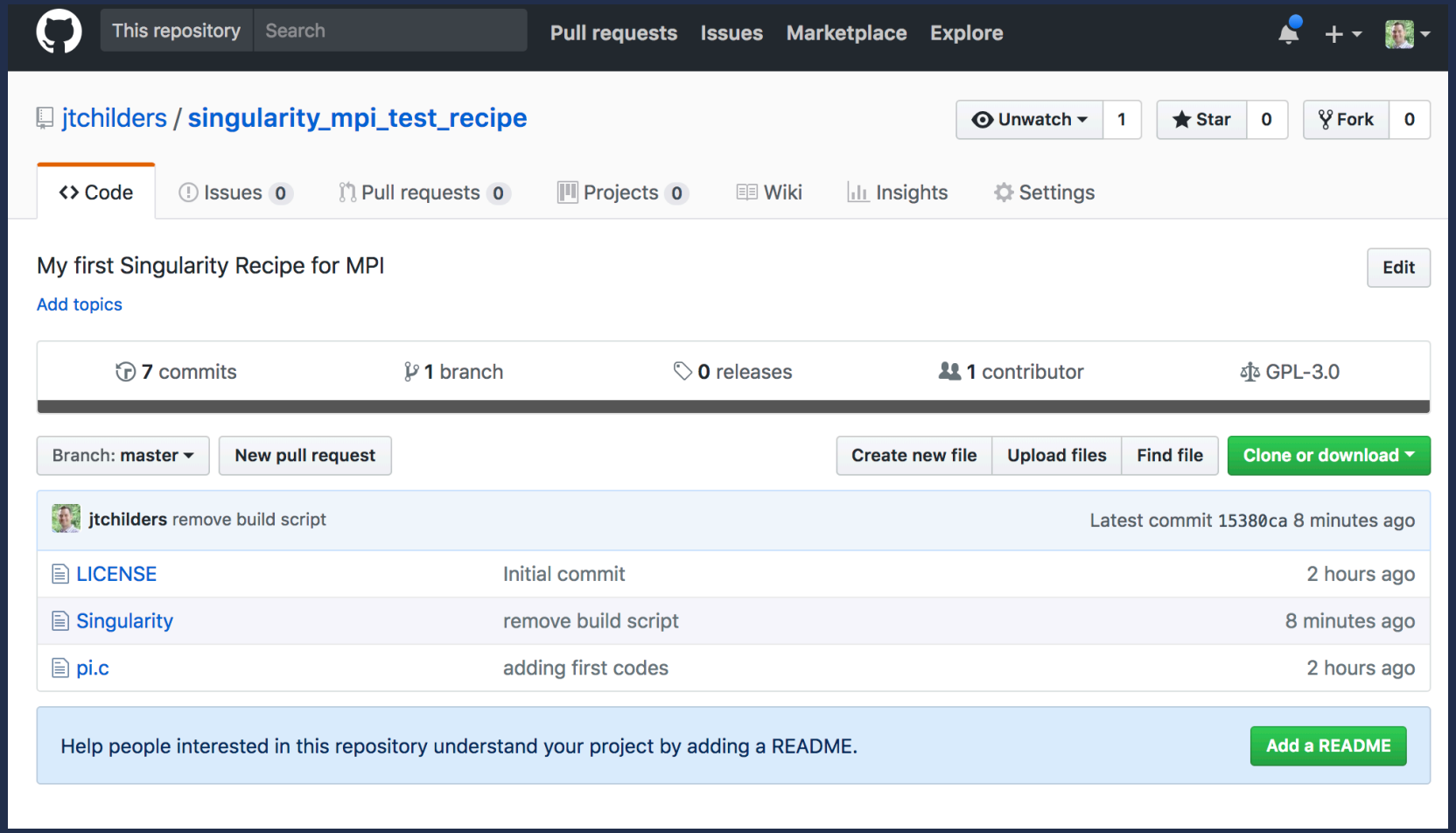
```
Bootstrap: docker
From: centos

%setup
   echo ${SINGULARITY_ROOTFS}
   mkdir ${SINGULARITY_ROOTFS}/myapp
   cp pi.c ${SINGULARITY_ROOTFS}/myapp/

%post
   yum update -y
   yum groupinstall -y "Development Tools"
   yum install -y gcc
   yum install -y gcc-c++
   yum install -y wget
   cd /myapp
   # install MPICH
   wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
   tar xf mpich-3.2.1.tar.gz
   cd mpich-3.2.1
   # disable the addition of the RPATH to compiled executables
   # this allows us to override the MPI libraries to use those
   # found via LD_LIBRARY_PATH
   ./configure --prefix=$PWD/install --disable-wrapper-rpath
   make -j 4 install
   # add to local environment to build pi.c
   export PATH=$PATH:$PWD/install/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
   cd ..
   mpicc -o pi -fPIC pi.c

%runscript
   /myapp/pi
```

```
wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
tar xf mpich-3.2.1.tar.gz
cd mpich-3.2.1
./configure --prefix=$PWD/install --disable-wrapper-rpath
make -j 4 install
export PATH=$PATH:$PWD/install/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/install/lib
cd ..
mpicc -o pi -fPIC pi.c
```

- Notice manual installation of MPICH into container.
- The configure command disables the setting of RPATH during linking of the shared MPI libraries.
- After installation of MPICH, PATH & LD_LIBRARY_PATH are set to include MPICH
- Then pi is built
- IMPORTANT: ensure it dynamically (not statically) links against MPICH

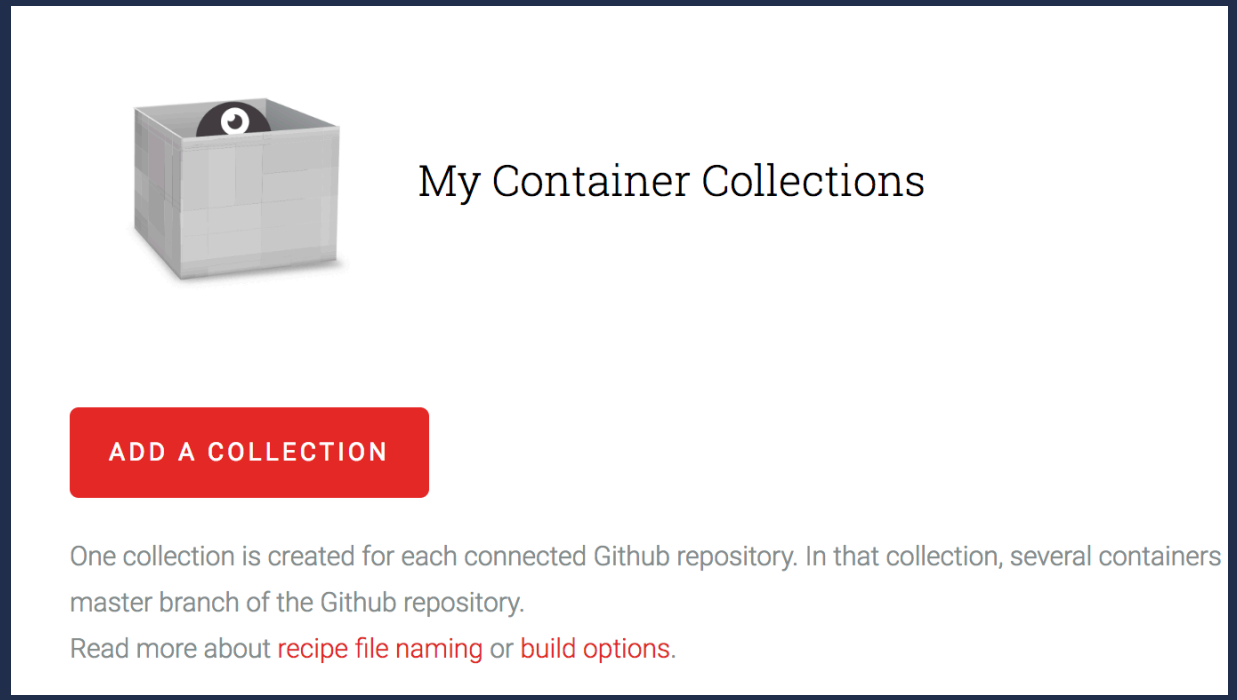
# Create new Github Repository

- https://github.com/jtchilders/singularity_mpi_test_recipe
- Need to add recipe file inside with filename 'Singularity'
- Add file pi.c from previous link



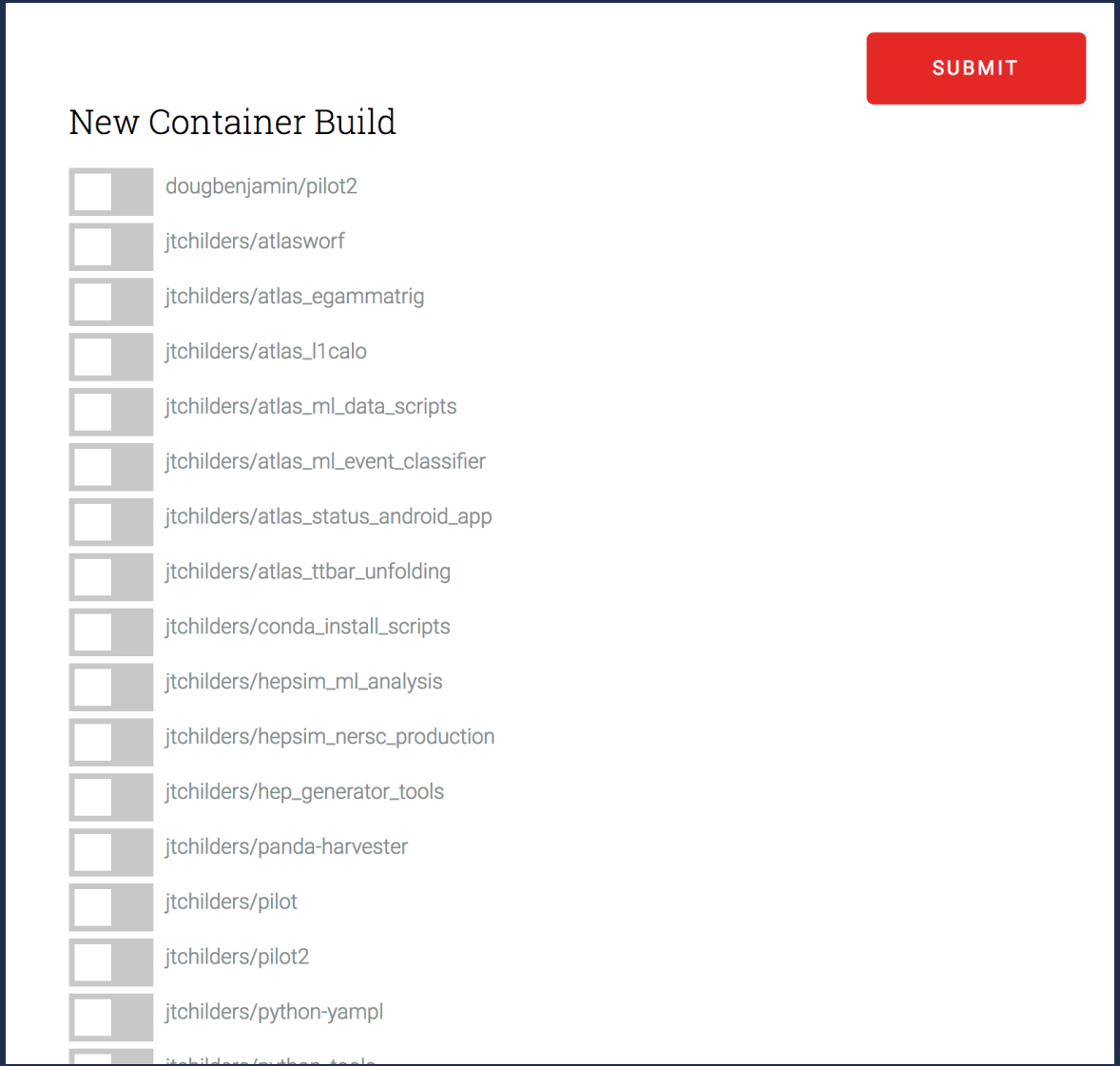
# Create Singularity Hub Account

- Goto: https://www.singularity-hub.org/login/
- Authenticate using your Github account
- You can then add github repositories to your container collection.
- Click the big red button
-



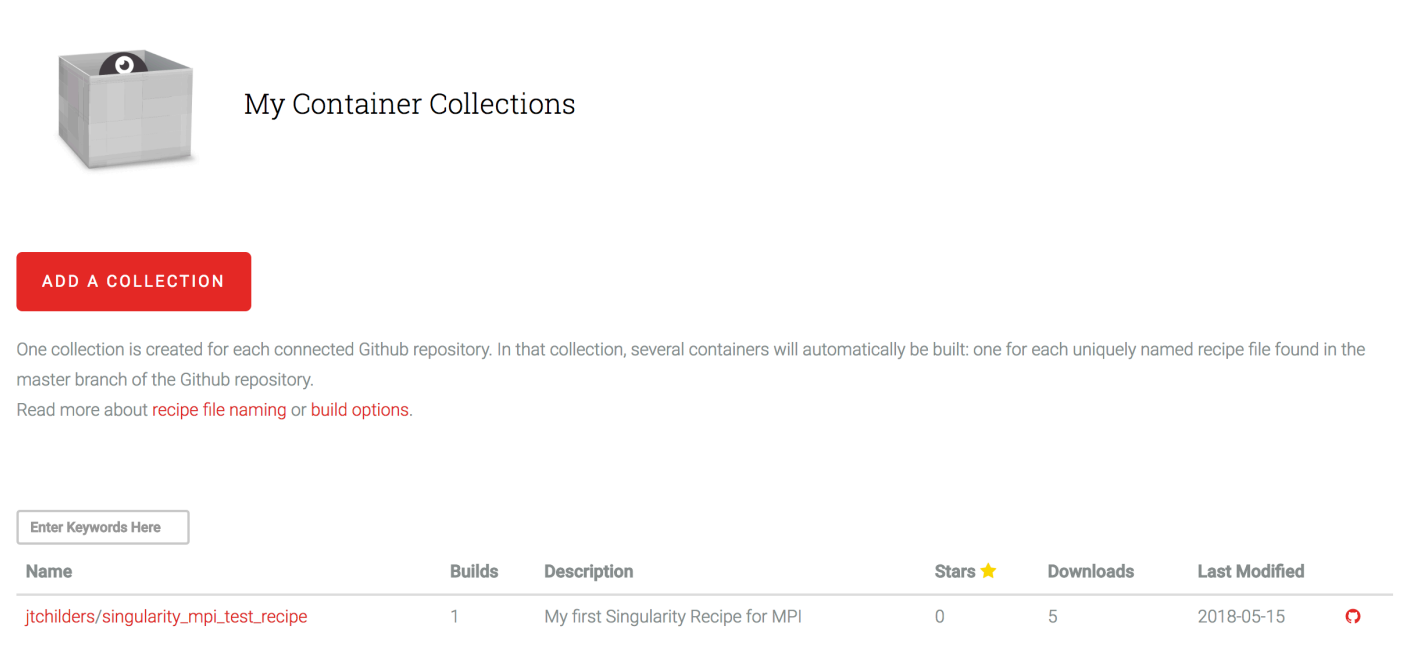
# Create Singularity Hub Account

- Goto: https://www.singularity-hub.org/login/
- Authenticate using your Github account
- You can then add github repositories to your container collection.
- Click the big red button
- Select your new repository and click the big red button
-



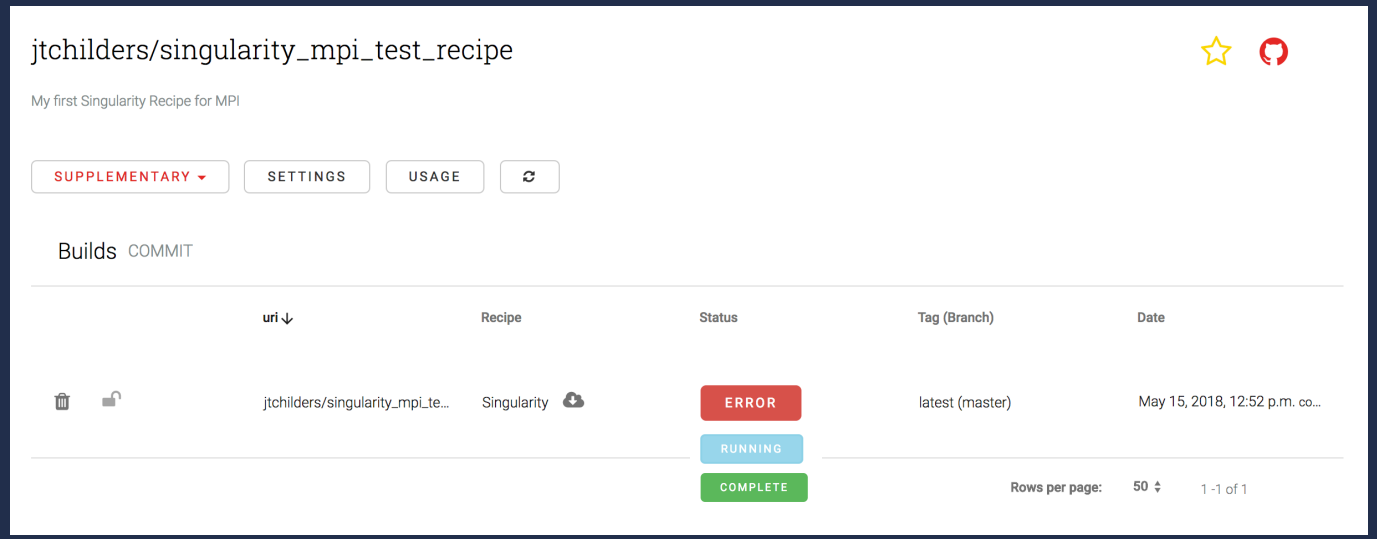
# Create Singularity Hub Account

- Goto: https://www.singularity-hub.org/login/
- Authenticate using your Github account
- You can then add github repositories to your container collection.
- Click the big red button
- Select your new repository and click the big red button
- Now you have your recipe listed and Singularity Hub will begin recursively searching the repo for any files named 'Singularity' and building those recipes
- Our example only has 1 recipe
- Click on the recipe



# Create Singularity Hub Account

- Goto: https://www.singularity-hub.org/login/
- Authenticate using your Github account
- You can then add github repositories to your container collection.
- Click the big red button
- Select your new repository and click the big red button
- Now you have your recipe listed and Singularity Hub will begin recursively searching the repo for any files named 'Singularity' and building those recipes
- Our example only has 1 recipe
- Click on the recipe to see it's build status
- Error messages during build can be seen by clicking the big red button
- Otherwise it will list the container as COMPLETE



# Retrieving Container

- Run the following on Theta to download and create an image:

```
singularity build myapp.img shub://jchilders/singularity_mpi_test_recipe
```

# Running Singularity Container on Theta

```
qsub submit.sh
```

# Running Singularity Container on Theta

```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 2
#COBALT -A EnergyFEC_3

# app build with GNU not Intel
module swap PrgEnv-intel PrgEnv-gnu
# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# prints to log file the list of modules loaded (just a check)
module list

# include CRAY_LD_LIBRARY_PATH in to the system library path
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
# also need this additional library
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64/:$LD_LIBRARY_PATH
# in order to pass environment variables to a Singularity container create the variable
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
# print to log file for debug
echo $SINGULARITYENV_LD_LIBRARY_PATH

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running agains the host machines Cray libmpi.so not the one inside the container
aprun -n 1 -N 1 singularity exec -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img ldd /myapp/pi
# run my contianer like an application, which will run '/myapp/pi'
aprun -n 8 -N 4 singularity run -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img
```

Standard Cobalt parameters

# Running Singularity Container on Theta

```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 2
#COBALT -A EnergyFEC_3

# app build with GNU not Intel
module swap PrgEnv-intel PrgEnv-gnu
# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# prints to log file the list of modules loaded (just a check)
module list

# include CRAY_LD_LIBRARY_PATH in to the system library path
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
# also need this additional library
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64/:$LD_LIBRARY_PATH
# in order to pass environment variables to a Singularity container create the variable
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
# print to log file for debug
echo $SINGULARITYENV_LD_LIBRARY_PATH

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running agains the host machines Cray libmpi.so not the one inside the container
aprun -n 1 -N 1 singularity exec -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img ldd /myapp/pi
# run my contianer like an application, which will run '/myapp/pi'
aprun -n 8 -N 4 singularity run -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img
```

Swap module for app

# Running Singularity Container on Theta

```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 2
#COBALT -A EnergyFEC_3

# app build with GNU not Intel
module swap PrgEnv-intel PrgEnv-gnu
# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# prints to log file the list of modules loaded (just a check)
module list

# include CRAY_LD_LIBRARY_PATH in to the system library path
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
# also need this additional library
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64/:$LD_LIBRARY_PATH
# in order to pass environment variables to a Singularity container create the variable
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
# print to log file for debug
echo $SINGULARITYENV_LD_LIBRARY_PATH

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running agains the host machines Cray libmpi.so not the one inside the container
aprun -n 1 -N 1 singularity exec -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img ldd /myapp/pi
# run my contianer like an application, which will run '/myapp/pi'
aprun -n 8 -N 4 singularity run -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img
```

Module changes updated CRAY_LD_LIBRARY_PATH, append it to local LD_LIBRARY_PATH
Also need to add addition library path.

# Running Singularity Container on Theta

```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 2
#COBALT -A EnergyFEC_3

# app build with GNU not Intel
module swap PrgEnv-intel PrgEnv-gnu
# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# prints to log file the list of modules loaded (just a check)
module list

# include CRAY_LD_LIBRARY_PATH in to the system library path
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
# also need this additional library
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.2.1-6.0.4.0_22.1__gd26a3dc.ari/lib64/:$LD_LIBRARY_PATH
# in order to pass environment variables to a Singularity container create the variable
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PATH
# print to log file for debug
echo $SINGULARITYENV_LD_LIBRARY_PATH

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running agains the host machines Cray libmpi.so not the one inside the container
aprun -n 1 -N 1 singularity exec -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img ldd /myapp/pi
# run my contianer like an application, which will run '/myapp/pi'
aprun -n 8 -N 4 singularity run -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img
```

Run application inside singularity, aprun handles the MPI

# Running Singularity Container on Theta

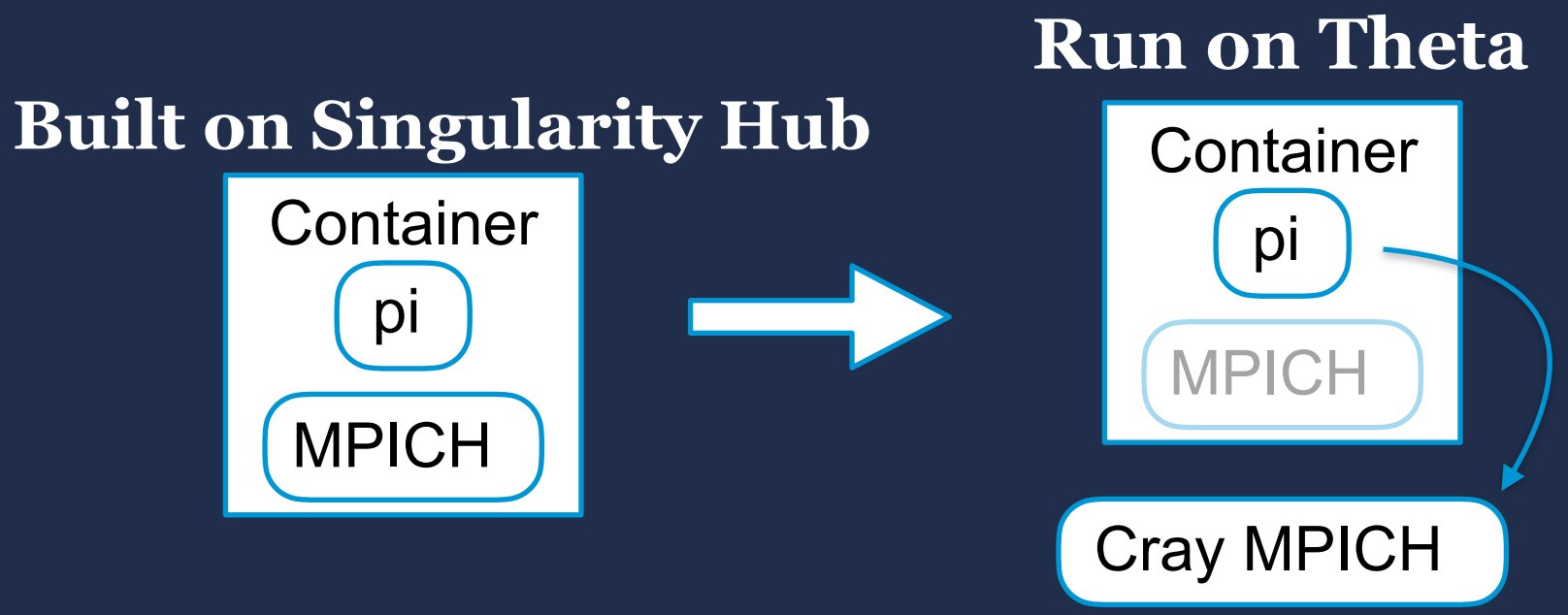
```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 2
#COBALT -A EnergyFEC_3

# app build with GNU not Intel
module swap PrgEnv-intel PrgEnv-gnu
# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# prints to log file the list of modules loaded (just a check)
module list

# include CRAY_LD_LIBRARY_PATH in to
export LD_LIBRARY_PATH=$CRAY_LD_LIBR
# also need this additional library
export LD_LIBRARY_PATH=/opt/cray/wlm
# in order to pass environment varia
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PAT
# print to log file for debug
echo $SINGULARITYENV_LD_LIBRARY_PATH

# this simply runs the command 'ldd /myap /pi' inside the container and should show that
# the app is running agains the host mac es Cray libmpi.so not the one inside the container
aprun -n 1 -N 1 singularity exec -B /opt: opt:ro -B /var/opt:/var/opt:ro mpitest.img ldd /myapp/pi
# run my contianer like an application, which will run '/myapp/pi'
aprun -n 8 -N 4 singularity run -B /opt:/opt:ro -B /var/opt:/var/opt:ro mpitest.img
```

`-B /opt:/opt:ro` causes Singularity to mount the host '/opt' inside the container at '/opt' in read-only (ro) mode. This allows the use of cray libraries that are needed to take advantage of Theta's unique hardware.

# Overview of the Workflow in Five Easy Steps!

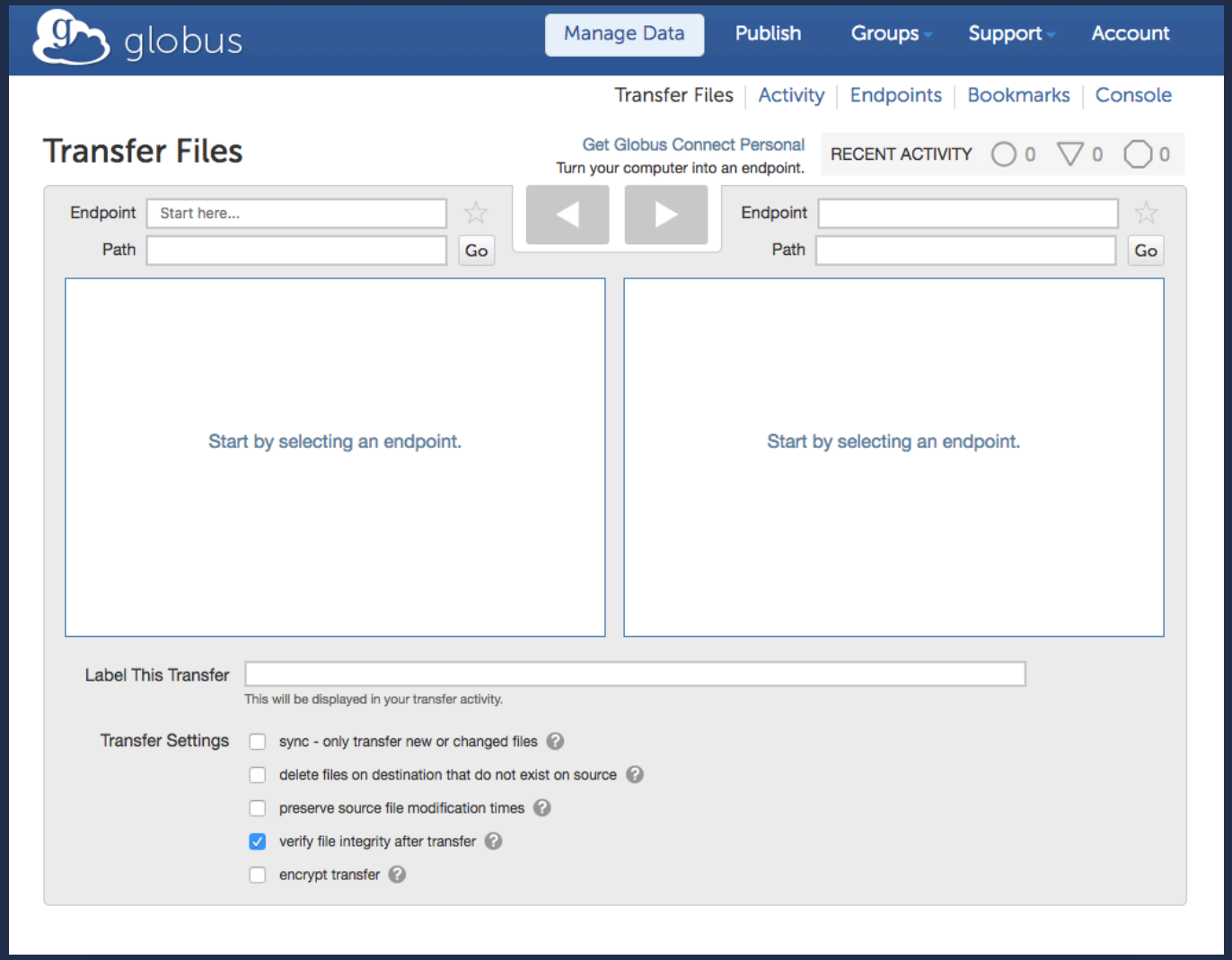
1. Create SingularityFile recipe in github
2. Link repo to Singularity Hub
3. Wait for build
4. Build on Theta
5. Run on Theta

Instructions for building on local machine:
https://www.alcf.anl.gov/user-guides/singularity



# Globus for Data Transfer

- Web Interface to transfer files between Globus Endpoints (NERSC,ALCF,OLCF,BNL,etc.)
- Login using ANL Credentials or other institutes
- Must authenticate with the myproxy server of source and destination.




https://www.globus.org/app/transfer

# Globus for Data Transfer

https://docs.globus.org/api/transfer/

- There is also a Python/Java API for doing this

https://github.com/globusonline/transfer-api-client-python

- Example Python implementation

```
from globusonline.transfer import api_client

api = api_client.TransferAPIClient(username="myusername",
                                   cert_file="/path/to/client/credential",
                                   key_file="/path/to/client/credential")
status_code, status_message, data = api.task_list()
```

- Provides effective transfer rates at the scale of 300MB/s between large facilities

# Theta Nodes RAM-disk (/tmp)

- Processes running on Theta compute nodes have access to `/tmp`
- This path maps some portion of the 192GB node DDR to a usable local filesystem
- The benefit is for low-memory applications with intermediate file-IO for non-persistent data
- Limited to 95GB
- **USE WITH CARE**: Know how much DDR your application requires, and do not write so much data to the RAM disk that your application runs out causing a crash.

# Summary

- Data Science Group is working to support Data & Learning software stacks
- Growing support for distributed learning frameworks
- Intel/Cray support of Tensorflow through custom libraries leading to scalable Deep Learning on Theta
- Singularity installed for users
- Containers offer portability and easy distribution of software though come with complications in custom hardware environments
- Globus provides high speed data transfers between supported endpoints