

Using Containerized Software at Argonne's Leadership Computing Facility

J. Taylor Childers (Argonne) on behalf of the Data Science Group

jchilders@anl.gov

datascience@alcf.anl.gov



Corey Adams



Prasanna
Balaprakash



Taylor Childers



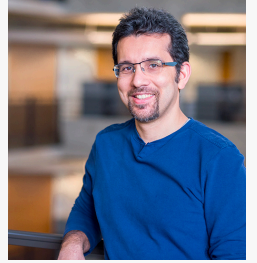
Murali Emani



Elise Jennings



Xiao-Yong Jin



Murat Keceli



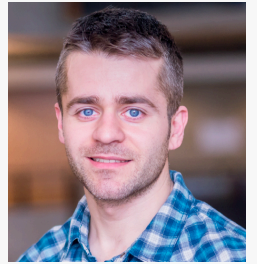
Bethany Lusch



Alvaro Vazquez
Mayagoitia



Adrian Pope



Misha Salim



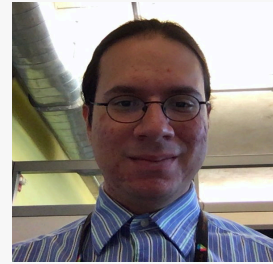
William Scullin



Ganesh Sivaraman



Tom Uram



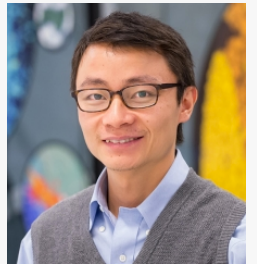
Antonio Villarreal



Venkat Vishwanath



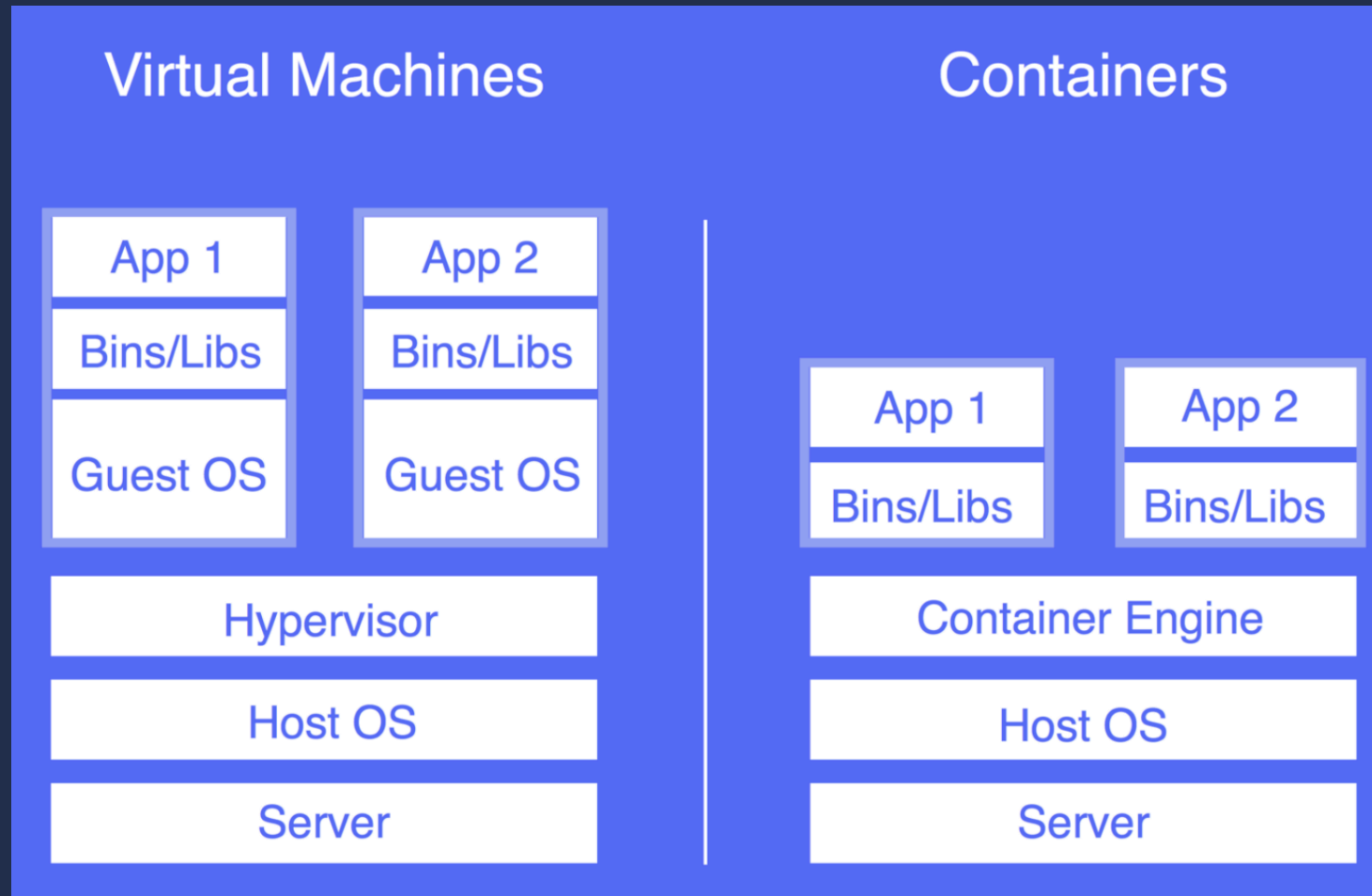
Richard Zamora



Huihuo Zheng

Introduction

- Containerization and Virtualization have been around for some time



Introduction

- Containerization and Virtualization have been around for some time
- Recently gained popularity on Cloud platforms
- Platforms include Docker, CharlieCloud, Singularity, Shifter, Rocket
- ALCF supports Singularity instead of Docker for security reasons.
- <https://www.sylabs.io/singularity/>
 - Singularity documentation pages
- <https://www.singularity-hub.org/>
 - Singularity Hub connects with a github repo and builds containers for you
- <https://www.alcf.anl.gov/user-guides/singularity>
 - Singularity documentation specific for ALCF
- Singularity is installed on:
 - Theta (4,400 Intel KNL nodes x 64 cores)
 - Cooley (126 nodes x 2 NVIDIA Teslas)



Building Singularity Containers

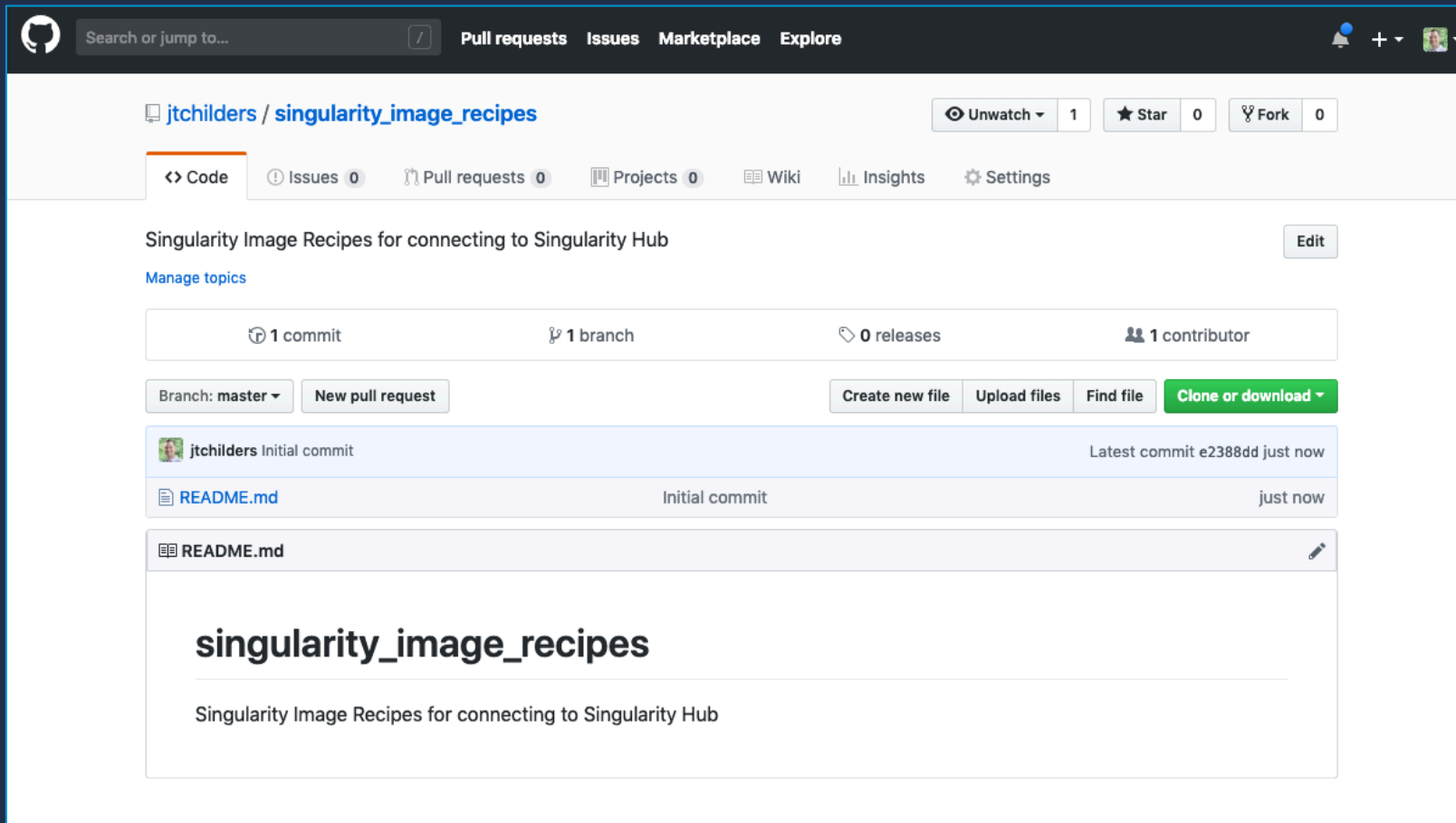
- If your container already exists on Docker hub or Singularity hub, the follow commands can download them on theta logins:

```
thetalogin5:~> singularity build myubuntu.img docker://ubuntu  
thetalogin5:~> singularity build myubuntu.img shub://singularityhub/ubuntu
```

- The Singularity build command requires 'sudo' rights when making local changes to images, which is not allowed for users on ALCF systems.
- Building custom images must be done elsewhere:
 - On your laptop
 - On your own system
 - On Singularity hub
- Then the image is moved to an ALCF resource
- First you'll need a singularity hub account
- Second you'll typically want to install Singularity on your laptop
- Third you can start building images

Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipes'



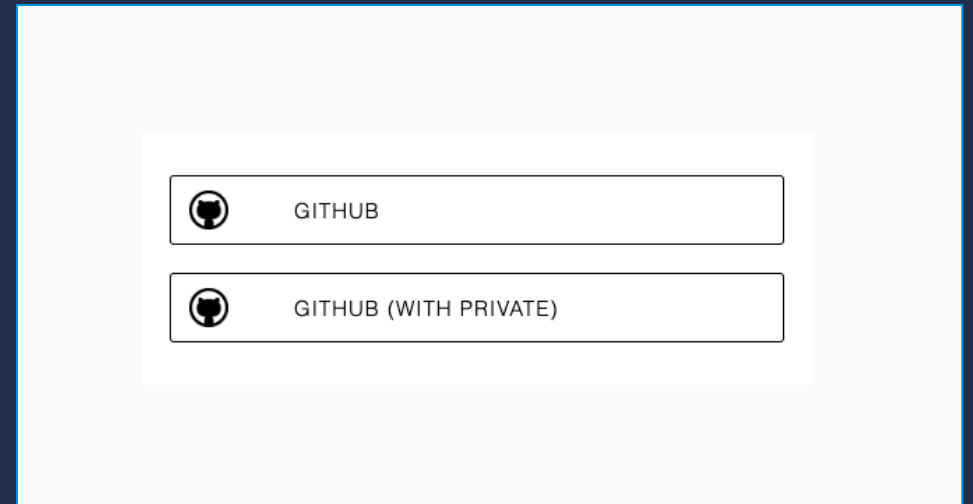
The screenshot shows a GitHub repository page for 'jtchillers / singularity_image_recipes'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The main branch is 'master'. The repository contains a single file, 'README.md', which was committed by 'jtchillers' as an 'Initial commit' just now. The README content is as follows:

```
singularity_image_recipes

Singularity Image Recipes for connecting to Singularity Hub
```

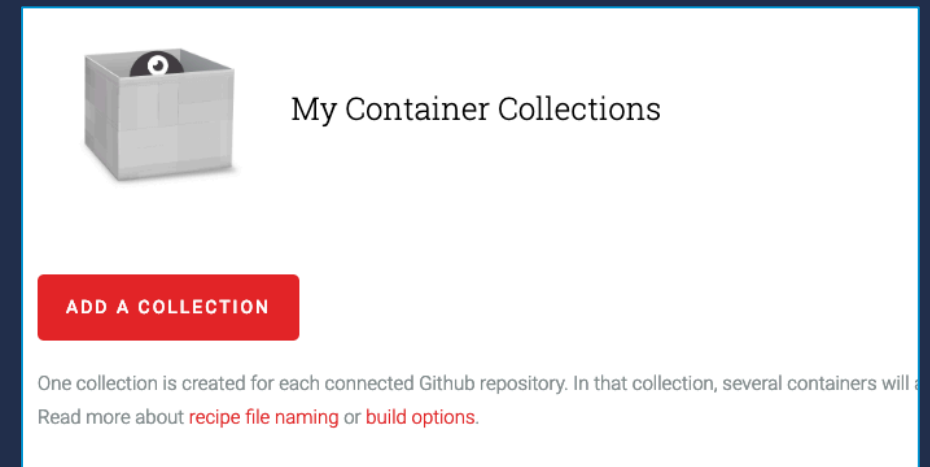
Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipes'
- Goto <https://www.singularity-hub.org/login>
- Choose to sign in with your 'github' account



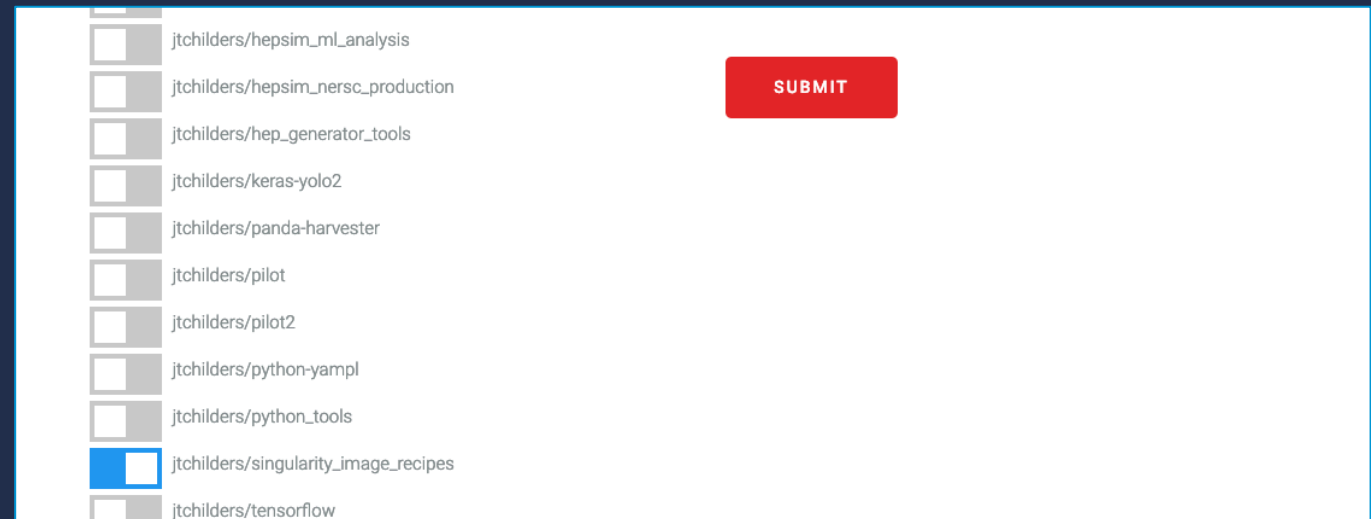
Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipes'
- Goto <https://www.singularity-hub.org/login>
- Choose to sign in with your 'github' account
- Go to the 'My Collections' part of the singularity hub website
- Click the red button 'ADD A COLLECTION'
- All your github repos will be listed, select the one you created above
- click 'SUBMIT'



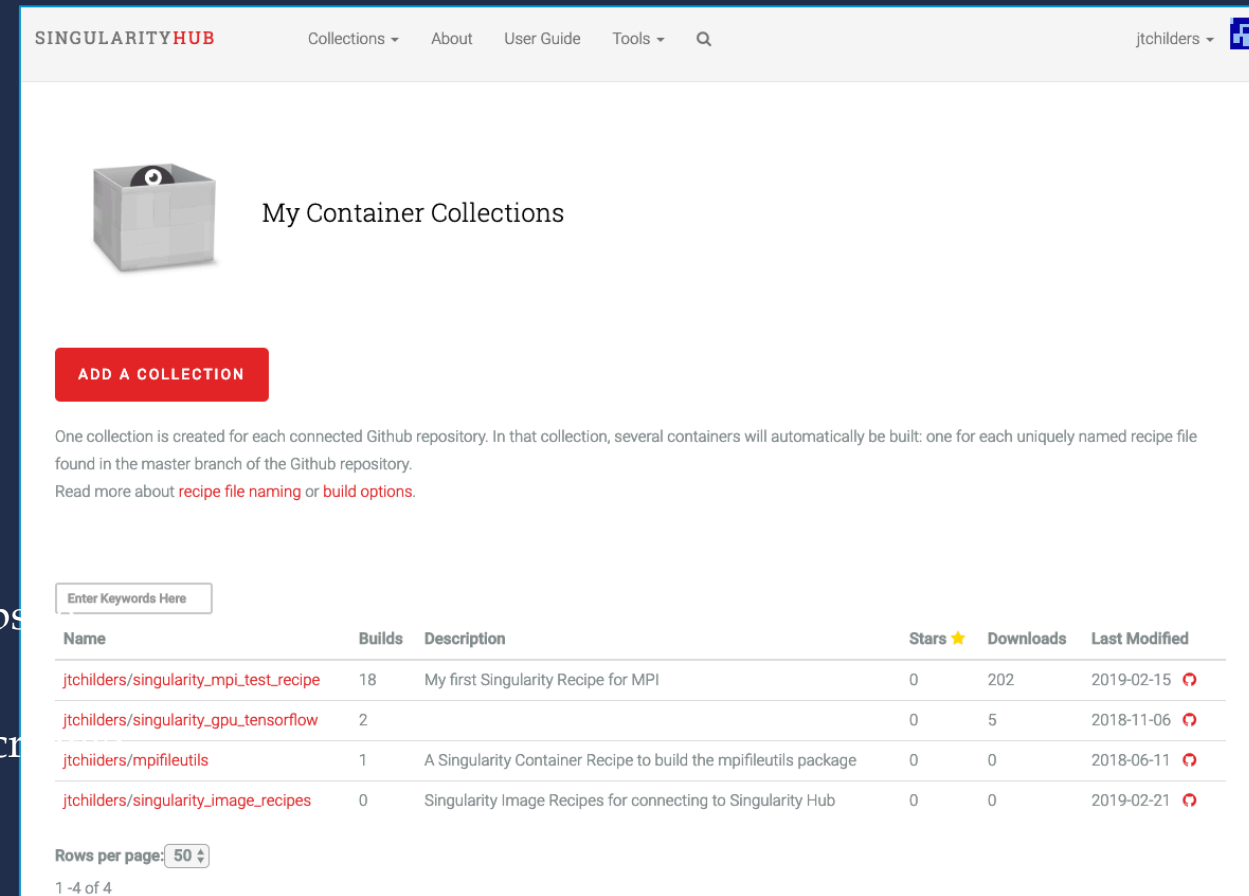
Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipes'
- Goto <https://www.singularity-hub.org/login>
- Choose to sign in with your 'github' account
- Go to the 'My Collections' part of the singularity hub website
- Click the red button 'ADD A COLLECTION'
- All your github repos will be listed, select the one you created above
- click 'SUBMIT'



Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipes'
- Goto <https://www.singularity-hub.org/login>
- Choose to sign in with your 'github' account
- Go to the 'My Collections' part of the singularity hub website
- Click the red button 'ADD A COLLECTION'
- All your github repos will be listed, select the one you created above
- click 'SUBMIT'
- Now this repo will be listed in your 'My Collections' page



SINGULARITYHUB Collections About User Guide Tools Q jtchilders

My Container Collections

ADD A COLLECTION

One collection is created for each connected Github repository. In that collection, several containers will automatically be built: one for each uniquely named recipe file found in the master branch of the Github repository.
Read more about [recipe file naming](#) or [build options](#).

Enter Keywords Here

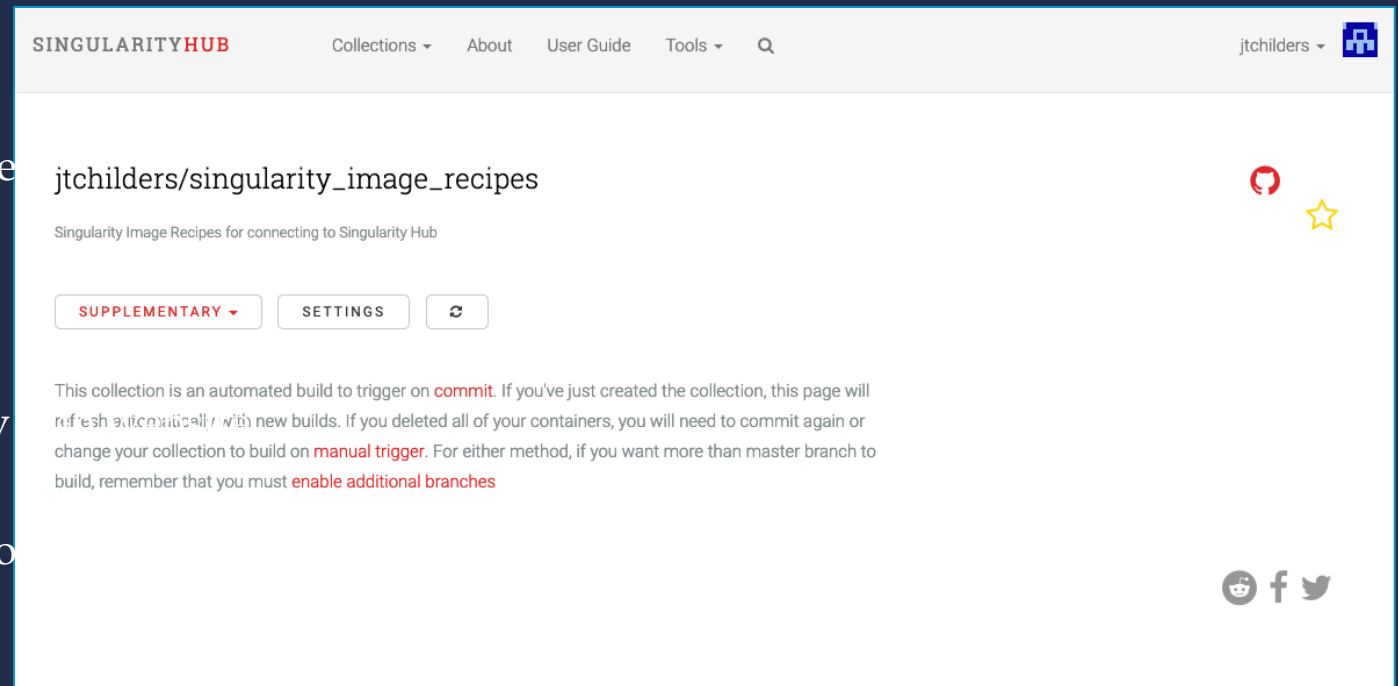
Name	Builds	Description	Stars ★	Downloads	Last Modified
jtchilders/singularity_mpi_test_recipe	18	My first Singularity Recipe for MPI	0	202	2019-02-15
jtchilders/singularity_gpu_tensorflow	2		0	5	2018-11-06
jtchilders/mpifileutils	1	A Singularity Container Recipe to build the mpifileutils package	0	0	2018-06-11
jtchilders/singularity_image_recipes	0	Singularity Image Recipes for connecting to Singularity Hub	0	0	2019-02-21

Rows per page: 50

1 - 4 of 4

Singularity Hub

- Goto or create your github account
- Create a new repo like 'singularity_image_recipe'
- Goto <https://www.singularity-hub.org/login>
- Choose to sign in with your 'github' account
- Go to the 'My Collections' part of the singularity
- Click the red button 'ADD A COLLECTION'
- All your github repos will be listed, select the one above
- click 'SUBMIT'
- Now this repo will be listed in your 'My Collections' page
- If you click on the repo it will have no images listed
- Now we need to build an image recipe file in our github repo



The screenshot shows the Singularity Hub interface for a collection named 'jtchilders/singularity_image_recipes'. The page title is 'SINGULARITY HUB' and the user is logged in as 'jtchilders'. The collection is described as 'Singularity Image Recipes for connecting to Singularity Hub'. There are three buttons: 'SUPPLEMENTARY', 'SETTINGS', and a refresh icon. Below the buttons, there is a paragraph of text: 'This collection is an automated build to trigger on **commit**. If you've just created the collection, this page will refresh automatically with new builds. If you deleted all of your containers, you will need to commit again or change your collection to build on **manual trigger**. For either method, if you want more than master branch to build, remember that you must **enable additional branches**'. There are social media icons for GitHub, Facebook, and Twitter.

Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # setup is run after the base 'centos' image is
6     # downloaded and unpacked but before entering the
7     # container environment
8
9     # this is the path on the local system to
10    # what will become your container's root directory
11    echo ${SINGULARITY_ROOTFS}
12    # create a directory for your application
13    mkdir ${SINGULARITY_ROOTFS}/myapp
14    # copy the hello world example from the github to
15    # the app directory
16    cp example_codes/hello_world.cpp ${SINGULARITY_ROOTFS}/myapp/
17
18 %post
19     # post is run after entering the container env.
20
21     # need to install some development tools to
22     # build our code
23     yum update -y
24     yum groupinstall -y "Development Tools"
25     yum install -y gcc g++
26
27     # enter directory where source file was copied
28     cd /myapp
29
30     # build
31     gcc -o hello_world hello_world.cpp
32
33 %runscript
34     # run script
35     /myapp/hello_world
36
37 %environment
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```

Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`
- This image uses the centos image from docker hub as a base image
 - This CENTOS image is compatible with most of the current DOE HPC resources

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # setup is run after the base 'centos' image is
6     # downloaded and unpacked but before entering the
7     # container environment
8
9     # this is the path on the local system to
10    # what will become your container's root directory
11    echo ${SINGULARITY_ROOTFS}
12    # create a directory for your application
13    mkdir ${SINGULARITY_ROOTFS}/myapp
14    # copy the hello world example from the github to
15    # the app directory
16    cp example_codes/hello_world.cpp ${SINGULARITY_ROOTFS}/myapp/
17
18 %post
19     # post is run after entering the container env.
```

```
1 Bootstrap: docker
2 From: centos
3
```

```
25 yum install -y gcc g++
26
27     # enter directory where source file was copied
28     cd /myapp
29
30     # build
31     gcc -o hello_world hello_world.cpp
32
33 %runscript
34     # run script
35     /myapp/hello_world
36
37 %environment
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```

Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`
- This image uses the centos image from docker hub as a base
 - This CENTOS image is compatible with most of the current HPC resources
- The 'setup' portion is run after the CENTOS image is unpacked on to the local file system. If you have source code to copy into the image you can do that now.

```
4 %setup
5 # setup is run after the base 'centos' image is
6 # downloaded and unpacked but before entering the
7 # container environment
8
9 # this is the path on the local system to
10 # what will become your container's root directory
11 echo ${SINGULARITY_ROOTFS}
12 # create a directory for your application
13 mkdir ${SINGULARITY_ROOTFS}/myapp
14 # copy the hello world example from the github to
15 # the app directory
16 cp example_codes/hello_world.cpp ${SINGULARITY_ROOTFS}/myapp/
```

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5 # setup is run after the base 'centos' image is
```

```
24 yum groupinstall -y "Development Tools"
25 yum install -y gcc g++
26
27 # enter directory where source file was copied
28 cd /myapp
29
30 # build
31 gcc -o hello_world hello_world.cpp
32
33 %runscript
34 # run script
35 /myapp/hello_world
36
37 %environment
38 # can define runtime environment variables here
39 # these vars will be set during calls to 'shell'
40 # or 'exec' or 'run' but will not be set during
41 # the previous 'post' section of the recipe file
42 # so, if you need them, define them there as well
43 export PATH=$PATH:/myapp
```

Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`
- This image uses the centos image from docker hub as a base image
 - This CENTOS image is compatible with most of the current DOE HPC resources
- The 'setup' portion is run after the CENTOS image is unpacked on to the local file system. If you have source code to copy into the image you can do that now. Uses BASH syntax.
- The 'post' portion is run after entering the CENTOS image environment. Uses BASH syntax.

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # setup is run after the base 'centos' image is
6     # downloaded and unpacked but before entering the
7     # container environment
8
9     # this is the path on the local system to
10    # what will become your container's root directory
11    echo ${SINGULARITY_ROOTFS}
12    # create a directory for your application
13    mkdir ${SINGULARITY_ROOTFS}/myapp
```

```
18 %post
19     # post is run after entering the container env.
20
21     # need to install some development tools to
22     # build our code
23     yum update -y
24     yum groupinstall -y "Development Tools"
25     yum install -y gcc g++
26
27     # enter directory where source file was copied
28     cd /myapp
29
30     # build
31     gcc -o hello_world hello_world.cpp
```

```
34     # run script
35     /myapp/hello_world
36
37 %environment
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```

Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`
- This image uses the centos image from docker hub as a base image
 - This CENTOS image is compatible with most of the current DOE HPC resources
- The 'setup' portion is run after the CENTOS image is unpacked on to the local file system. If you have source code to copy into the image you can do that now. Uses BASH syntax.
- The 'post' portion is run after entering the CENTOS image environment. Uses BASH syntax.
- The 'runscript' portion identifies a script to run when a user calls
 - `singularity run <image_filename>`

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # setup is run after the base 'centos' image is
6     # downloaded and unpacked but before entering the
7     # container environment
8
9     # this is the path on the local system to
10    # what will become your container's root directory
11    echo ${SINGULARITY_ROOTFS}
12    # create a directory for your application
13    mkdir ${SINGULARITY_ROOTFS}/myapp
14    # copy the hello world example from the github to
15    # the app directory
16    cp example_codes/hello_world.cpp ${SINGULARITY_ROOTFS}/myapp/
17
18 %post
19     # post is run after entering the container env.
20
21     # need to install some development tools to
22     # build our code
23     yum update -y
24     yum groupinstall -y "Development Tools"
25     yum install -y gcc g++
26
27     # enter directory where source file was copied
28     cd /myapp
29
30     # build
31
32
33 %runscript
34     # run script
35     /myapp/hello_world
36
37 %environment
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```


Your First Singularity Recipe

- Checkout your github repo on your laptop:
 - `git clone git@github.com:jtchilders/singularity_image_recipes.git`
 - `cp singularity_image_recipes`
- Now create a new recipe in this repo:
 - filename must start with `Singularity`
 - Can have extensions like this: `Singularity.hello_world`
- This image uses the centos image from docker hub as a base image
 - This CENTOS image is compatible with most of the current DOE HPC resources
- The 'setup' portion is run after the CENTOS image is unpacked on to the local file system. If you have source code to copy into the image you can do that now. Uses BASH syntax.
- The 'post' portion is run after entering the CENTOS image environment. Uses BASH syntax.
- The 'runscript' portion identifies a script to run when a user calls
 - `singularity run <image_filename>`
- The 'environment' portion defines environment variables that exist while the container is running.

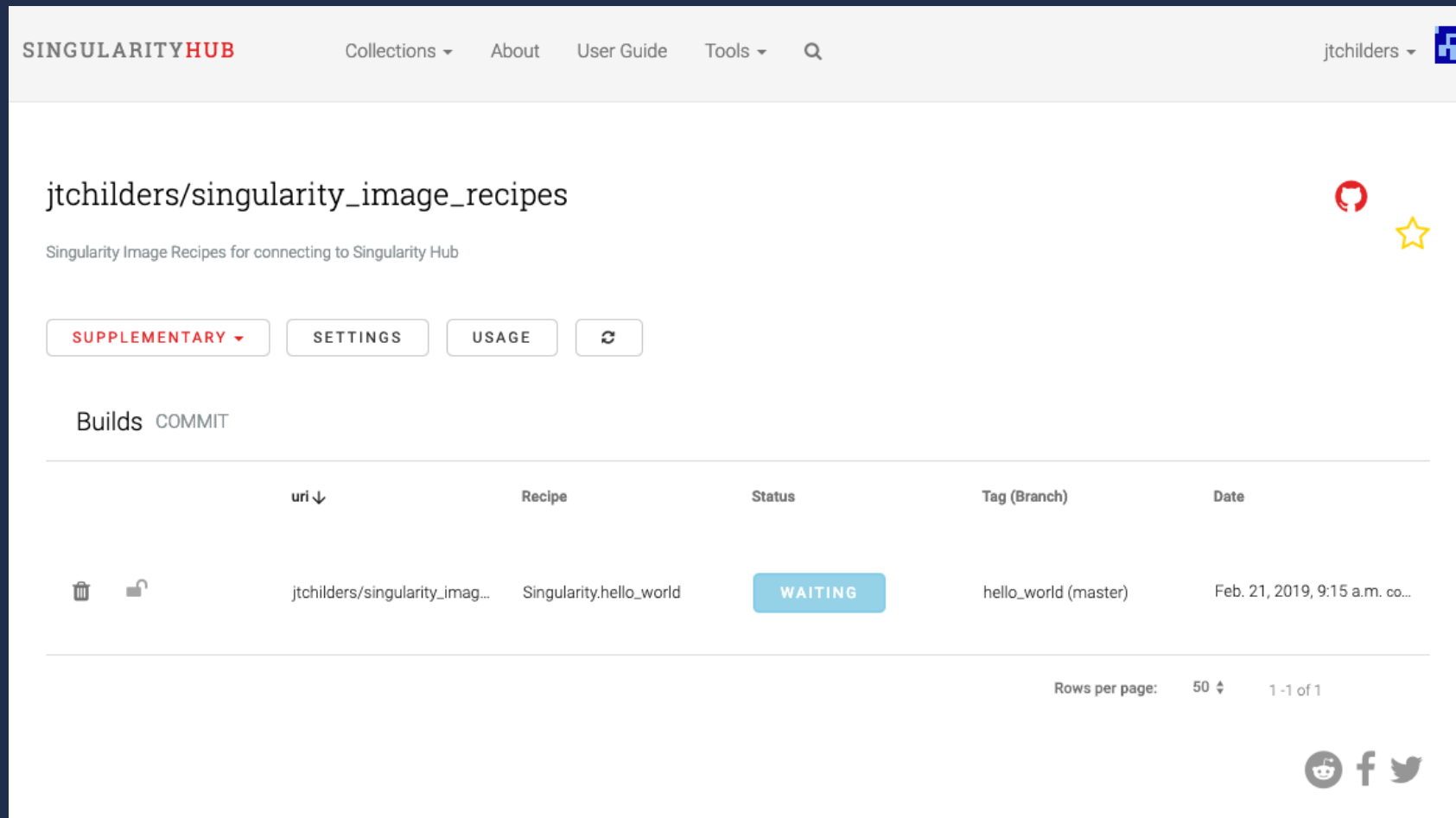
```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # setup is run after the base 'centos' image is
6     # downloaded and unpacked but before entering the
7     # container environment
8
9     # this is the path on the local system to
10    # what will become your container's root directory
11    echo ${SINGULARITY_ROOTFS}
12    # create a directory for your application
13    mkdir ${SINGULARITY_ROOTFS}/myapp
14    # copy the hello world example from the github to
15    # the app directory
16    cp example_codes/hello_world.cpp ${SINGULARITY_ROOTFS}/myapp/
17
18 %post
19     # post is run after entering the container env.
20
21     # need to install some development tools to
22     # build our code
23     yum update -y
24     yum groupinstall -y "Development Tools"
25     yum install -y gcc g++
26
```

```
37 %environment
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```



```
38     # can define runtime environment variables here
39     # these vars will be set during calls to 'shell'
40     # or 'exec' or 'run' but will not be set during
41     # the previous 'post' section of the recipe file
42     # so, if you need them, define them there as well
43     export PATH=$PATH:/myapp
```

Your First Singularity Recipe

- After a 'git add', 'git commit' & 'git push' you should see the status on your singularity hub page should report that the container is 'waiting', which means it is queued for building.



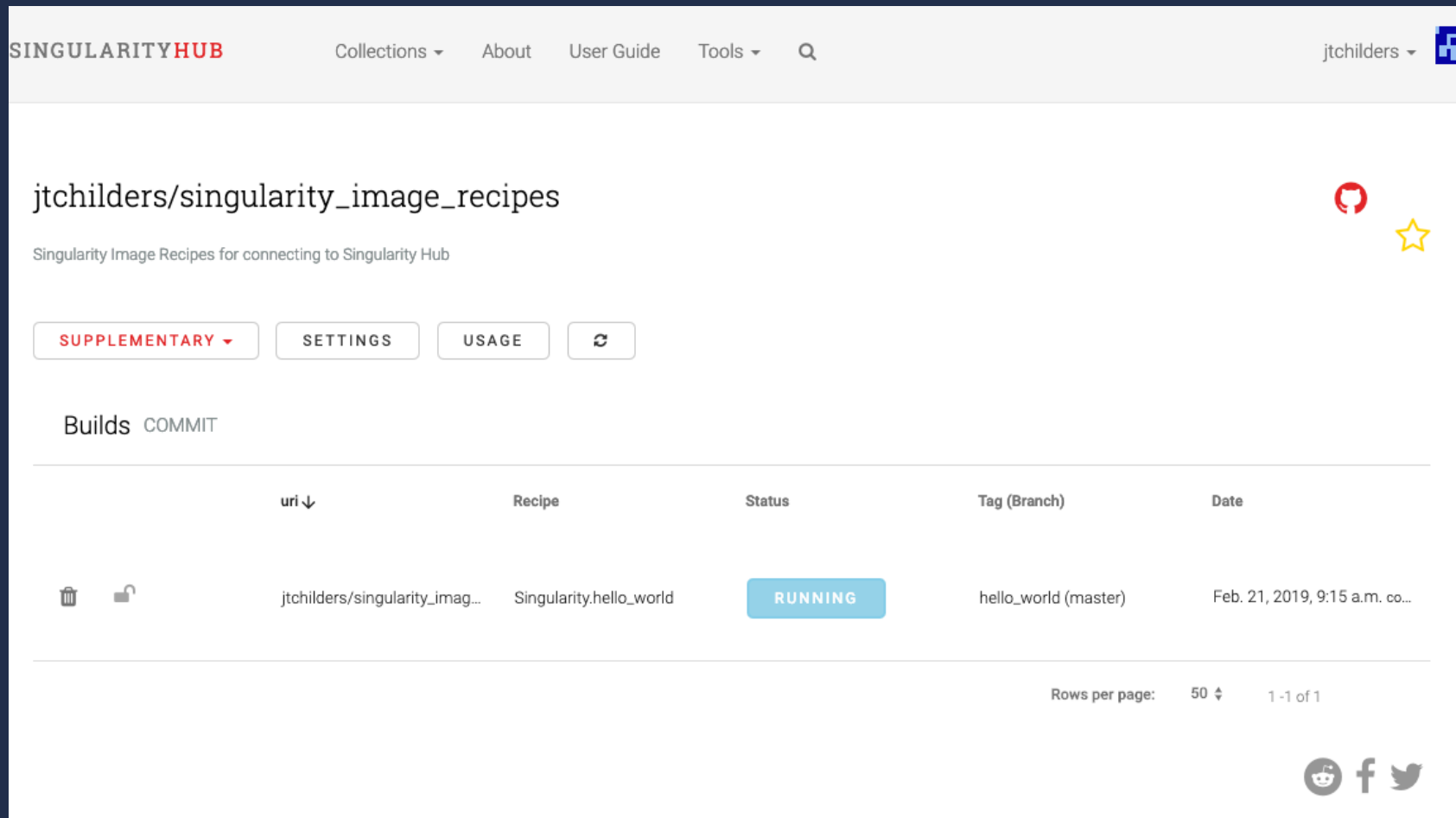
The screenshot shows the Singularity Hub interface for a user named 'jtchilders'. The page title is 'jtchilders/singularity_image_recipes'. Below the title, there is a description: 'Singularity Image Recipes for connecting to Singularity Hub'. There are four buttons: 'SUPPLEMENTARY', 'SETTINGS', 'USAGE', and a refresh icon. Below these buttons, there is a section for 'Builds' with a 'COMMIT' link. A table lists the build details:

uri ↓	Recipe	Status	Tag (Branch)	Date
  jtchilders/singularity_imag...	Singularity.hello_world	WAITING	hello_world (master)	Feb. 21, 2019, 9:15 a.m. co...



At the bottom right of the table, there is a 'Rows per page: 50' dropdown and '1 -1 of 1'. At the very bottom right, there are social media icons for GitHub, Facebook, and Twitter.

Your First Singularity Recipe

- Then it will change to RUNNING



The screenshot shows the Singularity Hub interface for a user named 'jtchilders'. The main heading is 'jtchilders/singularity_image_recipes', with a subtitle 'Singularity Image Recipes for connecting to Singularity Hub'. Below this are buttons for 'SUPPLEMENTARY', 'SETTINGS', 'USAGE', and a refresh icon. A 'Builds' section is visible, with a 'COMMIT' link. A table lists the build details:



	uri ↓	Recipe	Status	Tag (Branch)	Date
 	jtchilders/singularity_imag...	Singularity,hello_world	RUNNING	hello_world (master)	Feb. 21, 2019, 9:15 a.m. co...

At the bottom right, there are social media icons for GitHub, Facebook, and Twitter, and a footer indicating 'Rows per page: 50' and '1 - 1 of 1'.

Your First Singularity Recipe

- If there is a problem it will report ERROR, you can click on ERROR and it will show you a log file.

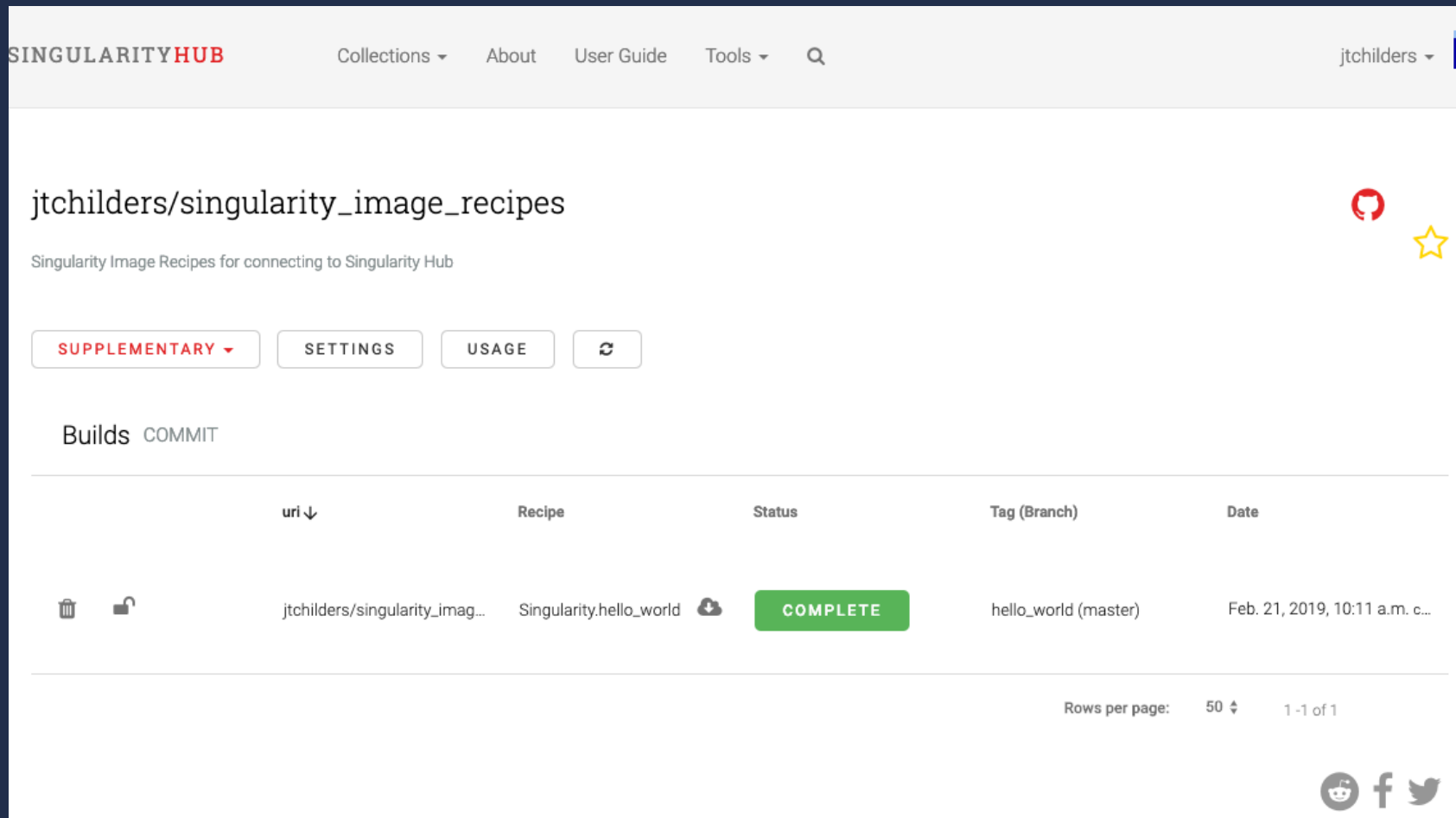
The screenshot shows the Singularity Hub interface for the user 'jtchilders'. The main heading is 'jtchilders/singularity_image_recipes' with a subtitle 'Singularity Image Recipes for connecting to Singularity Hub'. There are navigation buttons for 'SUPPLEMENTARY', 'SETTINGS', 'USAGE', and a refresh icon. Below this is a 'Builds' section with a 'COMMIT' link. A table lists the build details:

	uri ↓	Recipe	Status	Tag (Branch)	Date
 	jtchilders/singularity_imag...	Singularity.hello_world	ERROR	hello_world (master)	Feb. 21, 2019, 9:15 a.m. co...




At the bottom right, there are social media icons for GitHub, Facebook, and Twitter, and pagination information: 'Rows per page: 50' and '1 -1 of 1'.

Your First Singularity Recipe

- Once the container is completed it will show COMPLETE
- Now we can download it on Theta.



The screenshot shows the Singularity Hub interface for a user named 'jtchilders'. The page title is 'jtchilders/singularity_image_recipes'. Below the title, there is a description: 'Singularity Image Recipes for connecting to Singularity Hub'. There are four buttons: 'SUPPLEMENTARY' (with a dropdown arrow), 'SETTINGS', 'USAGE', and a refresh icon. Below these buttons, there is a section for 'Builds' with a 'COMMIT' link. A table lists the build details:

uri ↓	Recipe	Status	Tag (Branch)	Date
  jtchilders/singularity_imag...	Singularity.hello_world 	COMPLETE	hello_world (master)	Feb. 21, 2019, 10:11 a.m. c...

At the bottom right, there are social media icons for GitHub, Facebook, and Twitter. The footer shows 'Rows per page: 50' and '1 -1 of 1'.

Your First Singularity Recipe

- Once the container is completed it will show COMPLETE
- Now we can download it.
- Run on Theta login node:
- `singularity build hello_world.simg shub://jtchilders/singularity_image_recipes:hello_world`

```
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity build hello_world.simg shub://jtchilders/singularity_image_recipes:hello_world
Cache folder set to /gpfs/mira-home/parton/.singularity/shub
Progress |=====| 100.0%
Building from local image: /gpfs/mira-home/parton/.singularity/shub/jtchilders-singularity_image_recipes-master-hello_world.simg
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: hello_world.simg
Cleaning up...
```

Your First Singularity Recipe

- Once the container is completed it will show COMPLETE
- Now we can download it.
- Run on Theta login node:
- `singularity build hello_world.simg shub://jtchilders/singularity_image_recipes:hello_world`
- Now you can 'run' the container.
- Using the run command simply runs the '%runscript' specified in the container recipe

```
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity run hello_world.simg  
Hello World
```

Your First Singularity Recipe

- Once the container is completed it will show COMPLETE
- Now we can download it.
- Run on Theta login node:
- `singularity build hello_world.simg shub://jtchilders/singularity_image_recipes:hello_world`
- Now you can 'run' the container.
- Using the run command simply runs the '%runscript' specified in the container recipe
- Otherwise you can enter a shell using 'singularity shell <image_filename>'

```
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity shell hello_world.simg
Singularity: Invoking an interactive shell within container...

Singularity hello_world.simg:~/git/singularity_image_recipes> ls
example_codes hello_world.simg README.md Singularity.hello_world
Singularity hello_world.simg:~/git/singularity_image_recipes> ls /
anaconda-post.log bin dev environment etc home lib lib64 media mnt myapp opt proc run
Singularity hello_world.simg:~/git/singularity_image_recipes> ls /myapp
hello_world hello_world.cpp
Singularity hello_world.simg:~/git/singularity_image_recipes> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/myapp
```


Your First Singularity Recipe

- Once the container is completed it will show COMPLETE
- Now we can download it.
- Run on Theta login node:
- `singularity build hello_world.simg shub://jtchillers/singularity_image_recipes:hello_world`
- Now you can 'run' the container.
- Using the run command simply runs the '%runscript' specified in the container recipe
- Otherwise you can enter a shell using 'singularity shell <image_filename>'

- One can also run commands inside the container using 'singularity exec <image_filename> <command>'

```
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity exec hello_world.simg echo "Hello World"
Hello World
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity exec hello_world.simg ls /myapp
hello_world hello_world.cpp
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity exec hello_world.simg echo \${PATH}
${PATH}
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity exec hello_world.simg bash -c "echo \${PATH}"
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/myapp
```

Useful for investigating internal environment issues when building.

An MPI-enabled application

- Challenge of using MPI enabled containers on supercomputers is that MPI libraries are typically custom for each machine.
- Software in the container must link to native libraries to run properly

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # make directory for test MPI program
6     mkdir ${SINGULARITY_ROOTFS}/mpitestapp
7     cp example_codes/pi.c ${SINGULARITY_ROOTFS}/mpitestapp/
8
9 %post
10    # install development tools
11    yum update -y
12    yum groupinstall -y "Development Tools"
13    yum install -y gcc gcc-c++ wget
14
15    # install MPICH
16    MPICH_VERSION=3.3
17    mkdir /mpich
18    cd /mpich
19    wget http://www.mpich.org/static/downloads/$MPICH_VERSION/mpich-$MPICH_VERSION.tar.gz
20    tar xf mpich-$MPICH_VERSION.tar.gz --strip-components=1
21
22    # disable the addition of the RPATH to compiled executables
23    # this allows us to override the MPI libraries to use those
24    # found via LD_LIBRARY_PATH
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
33
34 %runscript
35    /mpitestapp/pi
```

<https://goo.gl/VBZAqL>

An MPI-enabled application

- Challenge of using MPI enabled containers on supercomputers is that MPI libraries are typically custom for each machine.
- Software in the container must link to native libraries to run properly
- Similar to previous except MPICH build
- Download code, untar

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # make directory for test MPI program
6     mkdir ${SINGULARITY_ROOTFS}/mpitestapp
7     cp example_codes/pi.c ${SINGULARITY_ROOTFS}/mpitestapp/
8
9 %post
10    # install development tools
11    yum update -y
12    yum groupinstall -y "Development Tools"
13    yum install -y gcc gcc-c++ wget
14
15    # install MPICH
```

<https://goo.gl/VBZAqL>

```
15    # install MPICH
16    MPICH_VERSION=3.3
17    mkdir /mpich
18    cd /mpich
19    wget http://www.mpich.org/static/downloads/$MPICH_VERSION/mpich-$MPICH_VERSION.tar.gz
20    tar xf mpich-$MPICH_VERSION.tar.gz --strip-components=1
```

```
23    # this allows us to override the MPI libraries to use those
24    # found via LD_LIBRARY_PATH
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
33
34 %runscript
35    /mpitestapp/pi
```

An MPI-enabled application

- Challenge of using MPI enabled containers on supercomputers is that MPI libraries are typically custom for each machine.
- Software in the container must link to native libraries to run properly
- Similar to previous except MPICH build
- Download code, untar
- Then build MPICH using `--disable-wrapper-rpath`
- Then build example MPI application 'pi.c'

<https://goo.gl/VBZAqL>

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # make directory for test MPI program
6     mkdir ${SINGULARITY_ROOTFS}/mpitestapp
7     cp example_codes/pi.c ${SINGULARITY_ROOTFS}/mpitestapp/
8
9 %post
10    # install development tools
11    yum update -y
12    yum groupinstall -y "Development Tools"
13    yum install -y gcc gcc-c++ rust
```

```
22    # disable the addition of the RPATH to compiled executables
23    # this allows us to override the MPI libraries to use those
24    # found via LD_LIBRARY_PATH
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
```

```
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
33
34 %runscript
35    /mpitestapp/pi
```

An MPI-enabled application

- Challenge of using MPI enabled containers on supercomputers is that MPI libraries are typically custom for each machine.
- Software in the container must link to native libraries to run properly
- Similar to previous except MPICH build
- Download code, untar
- Then build MPICH using `--disable-wrapper-rpath`
- Then build example MPI application 'pi.c'
- Build MPI applications using 'mpicc' or similar
- Without the build flag above, these binaries would build the pi.c binary such that it directly links to the local MPI libraries
- With the flag, the binary must use `LD_LIBRARY_PATH` to find the MPI libraries
- This allows us to override the containers version of MPI with the system version

```
1 Bootstrap: docker
2 From: centos
3
4 %setup
5     # make directory for test MPI program
6     mkdir ${SINGULARITY_ROOTFS}/mpitestapp
7     cp example_codes/pi.c ${SINGULARITY_ROOTFS}/mpitestapp/
8
9 %post
10    # install development tools
11    yum update -y
12    yum groupinstall -y "Development Tools"
13    yum install -y gcc gcc-c++ rust
```

```
22    # disable the addition of the RPATH to compiled executables
23    # this allows us to override the MPI libraries to use those
24    # found via LD_LIBRARY_PATH
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
```

```
25    ./configure --prefix=/mpich/install --disable-wrapper-rpath
26    make -j 4 install
27    # add to local environment to build pi.c
28    export PATH=$PATH:/mpich/install/bin
29    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
30    env | sort
31    cd /mpitestapp
32    mpicc -o pi -fPIC pi.c
33
34 %runscript
35    /mpitestapp/pi
```

Run Containerized MPI-app on Theta

- Download the image on Theta
- Now create a submit script as follows
- Running the container on Theta requires one to add the Cray MPI libraries to the LD_LIBRARY_PATH inside the container.
- This means the dynamically linked binary application uses the Cray MPI library instead of the MPICH library inside the container.

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

# pass container as first argument to script
CONTAINER=$1

# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# 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/1.3.2-6.0.6.0_3.8__g388ccd5.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

RANKS_PER_NODE=4
TOTAL_RANKS=$(( $COBALT_JOBSIZE * $RANKS_PER_NODE ))

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running against the host machines Cray libmpi.so not the one inside the container
# run my container like an application, which will run '/myapp/pi'
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

<https://goo.gl/PU2dy2>

Run Containerized MPI-app on Theta

<https://goo.gl/PU2dy2>

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

# pass container as first argument to script
CONTAINER=$1

# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# 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
LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.3.2-6.0.6.0_3.8__g388ccd5.ari/lib64/:$LD_LIBRARY_PATH

# environment variables to a Singularity container create the variable
SINGULARITYENV_prefix
SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
# file for debug
SINGULARITYENV_LD_LIBRARY_PATH

COBALT_JOBSIZE * $RANKS_PER_NODE ))

# the command 'ldd /myapp/pi' inside the container and should show that
# pointing against the host machines Cray libmpi.so not the one inside the container

# run my container like an application, which will run '/myapp/pi'
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Swap standard Cray MPICH with an ABI (Application Binary Interface) version. The ABI version ensures codes compiled with different MPI version are compatible.

Run Containerized MPI-app on Theta

<https://goo.gl/PU2dy2>

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

# pass container as first argument to script
CONTAINER=$1

# Use Cray's Application Binary Independent MPI build
module swap cray-mpich cray-mpich-abi

# 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/1.3.2-6.0.6.0_3.8__g388ccd5.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
# run my contianer like an application, which will run '/myapp/pi'
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Add Cray libraries to LD_LBIRARY_PATH

* \$RANKS_PER_NODE))

Run Containerized MPI-app on Theta

<https://goo.gl/PU2dy2>

In order to pass environment variables in to the container environment, define external variables prefixed with SINGULARITYENV_

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

```
ment to script
```

```
ly Independent MPI build
ich-abi
```

```
in to the system library path
```

```
LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

```
# also need this additional library
```

```
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.3.2-6.0.6.0_3.8_0388ccd5.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
```

```
RANKS_PER_NODE=4
```

```
TOTAL_RANKS=$(( $COBALT_JOBSIZE * $RANKS_PER_NODE ))
```

```
# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
```

```
# the app is running against the host machines Cray libmpi.so not the one inside the container
```

```
# run my container like an application, which will run '/myapp/pi'
```

```
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Run Containerized MPI-app on Theta

The aprun command first executes the singularity container then the application of interest.

<https://goo.gl/PU2dy2>

In this case, we use the 'run' command which directly executes the pi-executable.

Note the '-B' options which allow the mounting of external paths into the container environment. The syntax here is

```
-B <outside_path>:<inside_path>:<permissions>
```

ro = readonly, rw = readwrite

```
...d5.ari/lib64/:$LD_LIBRARY_PATH  
...create the variable
```

```
# this simply runs the command 'ldd /myapp/pi' inside the container and should show that  
# the app is running against the host machines Cray libmpi.so not the one inside the container  
# run my container like an application, which will run '/myapp/pi'  
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Run Containerized MPI-app on Theta

<https://goo.gl/PU2dy2>

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

```
parton::thetalogin4 { ~/git/singularity_image_recipes }-> singularity build mpich33.simg shub://jtchilders/singularity_image_recipes:mpich33
Cache folder set to /gpfs/mira-home/parton/.singularity/shub
Progress |=====| 100.0%
Building from local image: /gpfs/mira-home/parton/.singularity/shub/jtchilders-singularity_image_recipes-master-mpich33.simg
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: mpich33.simg
Cleaning up...
```

```
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.3.2-6.0.6.0_3.8__g388ccd5.ar1/11b64/:$LD_LIBRARY_PATH
# in order to pass environment variables to a Singularity container create the variable
# with the SINGULARITYENV_ prefix
export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
```

```
parton::thetalogin6 { ~/git/singularity_image_recipes }-> qsub submit.sh mpich33.simg
Job routed to queue "debug-cache-quad".
Memory mode set to cache quad for queue debug-cache-quad
315401
```

```
# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running against the host machines Cray libmpi.so not the one inside the container
# run my container like an application, which will run '/myapp/pi'
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Run Containerized MPI-app on Theta

<https://goo.gl/PU2dy2>

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

# pass container as first argument to script
CONTAINER=$1
```

```
worker 1 of 8
worker 0 of 8
worker 3 of 8
worker 2 of 8
worker 4 of 8
worker 5 of 8
worker 6 of 8
worker 7 of 8
pi is approximately 3.1417259869152532, Error is 0.0001333333254601
Application 11596238 resources: utime ~2s, stime ~5s, Rss ~12392, inblocks ~56552, outblocks ~0
```

64/:\$LD_LIBRARY_PATH
variable

```
RANKS_PER_NODE=4
TOTAL_RANKS=$(( $COBALT_JOBSIZE * $RANKS_PER_NODE ))

# this simply runs the command 'ldd /myapp/pi' inside the container and should show that
# the app is running against the host machines Cray libmpi.so not the one inside the container
# run my container like an application, which will run '/myapp/pi'
aprun -n $TOTAL_RANKS -N $RANKS_PER_NODE singularity run -B /opt:/opt:ro $CONTAINER
```

Building on Previous Images

- Highly recommended for complex multi-step builds in which each step takes a bit of time
 - This saves time when debugging builds
 - keep each step in its own recipe file, building on the last
- As an example, assume we want to build up an environment with GCC 6 and python 3.6
- We can again start with the CENTOS image which comes only with gcc 4.8.5 and python 3.6
- First we install the GCC 6
- Then we add Python 3.6 on top
- notice the change in the Bootstrap/From

```
1 Bootstrap: docker
2 From: centos https://goo.gl/KfPsYF
3
4 %post
5     # install development tools
6     yum update -y
7     yum install -y centos-release-scl
8     yum install -y devtoolset-6
9     yum install -y java-1.8.0-openjdk-devel.x86_64
```

```
1 Bootstrap: shub
2 From: jtchilders/singularity_image_recipes:devtoolset6 https://goo.gl/fBknSy
3
4 %post
5     # install python36
6     yum install -y https://centos7.iuscommunity.org/ius-release.rpm
7     yum update -y
8     yum install -y python36u python36u-libs python36u-devel python36u-pip
```

Building on Previous Image

- Next we add in MPICH again, and a python pi script

<https://goo.gl/uuh6kr>

```
1 Bootstrap: shub
2 From: jtcholders/singularity_image_recipes:dvs6_py36
3
4 %setup
5     # copy test script to container
6     mkdir $SINGULARITY_ROOTFS/myapp
7     cp example_codes/pi.py $SINGULARITY_ROOTFS/myapp
8
9 %post
10    yum install -y wget
11    echo setting up devtoolset6
12    # setup devtoolset6
13    scl enable devtoolset-6 bash
14
15    MPICH_VERSION=3.3
16    echo installing mpich $MPICH_VERSION
17    mkdir /mpich
18    cd /mpich
19    wget http://www.mpich.org/static/downloads/$MPICH_VERSION/mpich-$MPICH_VERSION.tar.gz
20    tar xf mpich-$MPICH_VERSION.tar.gz --strip-components=1
21    ./configure --prefix=/mpich/install --disable-wrapper-rpath
22    make -j 4 install
23
24    # add mpich to environment
25    export PATH=$PATH:/mpich/install/bin
26    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
27
28    # install mpi4py
29    pip3.6 install mpi4py
30
31 %runscript
32    python3.6 /myapp/pi.py
33
34
```

Miniconda installation

- There is also an example of installing a container with miniconda, using the intel channel, with Tensorflow, PyTorch, and Keras
- Used this module to measure Python Import performance on Lustre versus from within the container.

<https://goo.gl/JNjJFh>

```
1 Bootstrap: shub
2 From: jtchillers/singularity_image_recipes:mpich33
3
4 %setup
5     mkdir ${SINGULARITY_ROOTFS}/test
6     cp example_codes/keras_mnist.py $SINGULARITY_ROOTFS/test
7     cp example_codes/download_mnist.sh $SINGULARITY_ROOTFS/test
8     cp example_codes/mnist.npz $SINGULARITY_ROOTFS/test
9
10 %post
11     # install development tools
12     yum update -y
13     yum groupinstall -y "Development Tools"
14     yum install -y gcc
15     yum install -y gcc-c++
16     yum install -y wget
17
18     # setup MPICH
19     export PATH=$PATH:/mpich/install/bin
20     export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mpich/install/lib
21
22     #####
23     # miniconda install
24     #####
25
26
27
28     # change this if you want different versions
29     CONDAVER=3
30     VERSION=4.5.12
31     BASE_DIR=/miniconda$CONDAVER
32     PREFIX_PATH=$BASE_DIR/$VERSION
33     DOWNLOAD_PATH=$BASE_DIR/DOWNLOADS
34
35     # make install area
36     mkdir -p $PREFIX_PATH
37     mkdir -p $DOWNLOAD_PATH
38
39     MINICONDA_INSTALL_FILE=Miniconda$CONDAVER-$VERSION-Linux-x86_64.sh
40
41     echo Downloading miniconda installer
42     wget https://repo.continuum.io/miniconda/$MINICONDA_INSTALL_FILE -P $DOWNLOAD_PATH
43
44     chmod +x $DOWNLOAD_PATH/Miniconda$CONDAVER-$VERSION-Linux-x86_64.sh
45
46     echo Installing Miniconda
47     $DOWNLOAD_PATH/Miniconda$CONDAVER-$VERSION-Linux-x86_64.sh -b -p $PREFIX_PATH -u
```

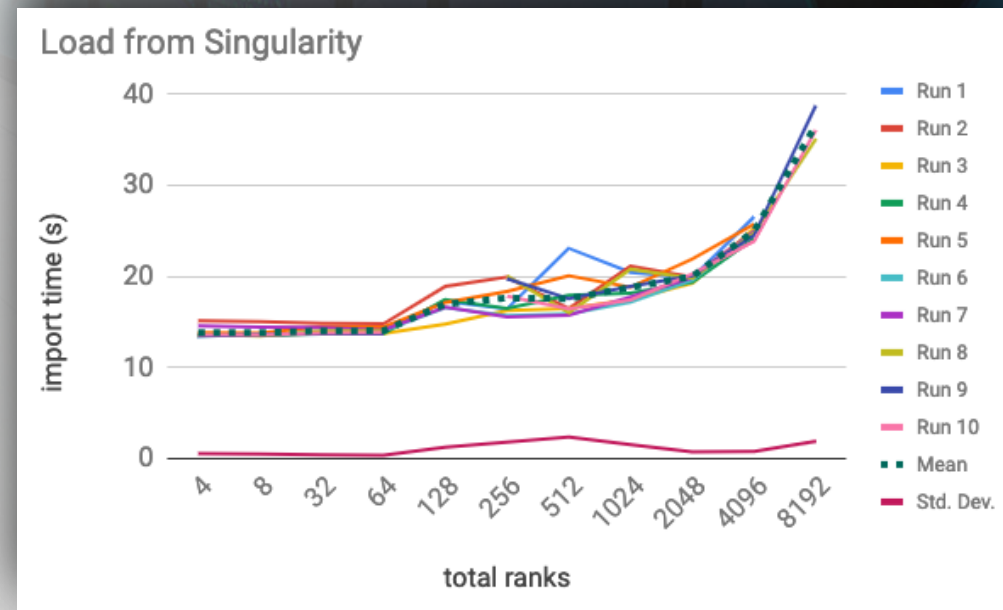
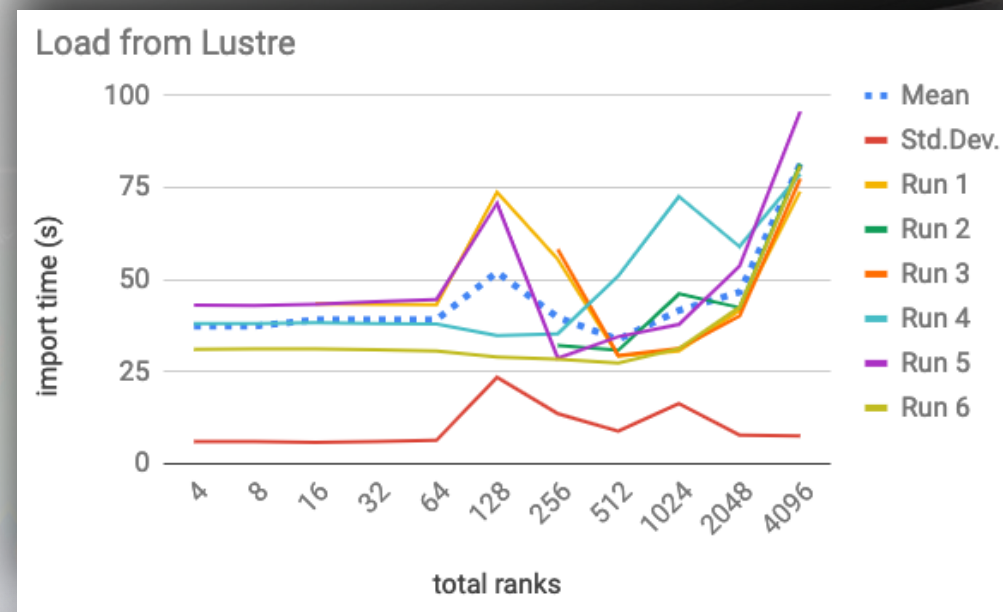
Singularity Performance on Theta

- File IO on leadership machines is always an issue.
- Python imports can be time consuming with thousands of ranks loading modules in parallel.



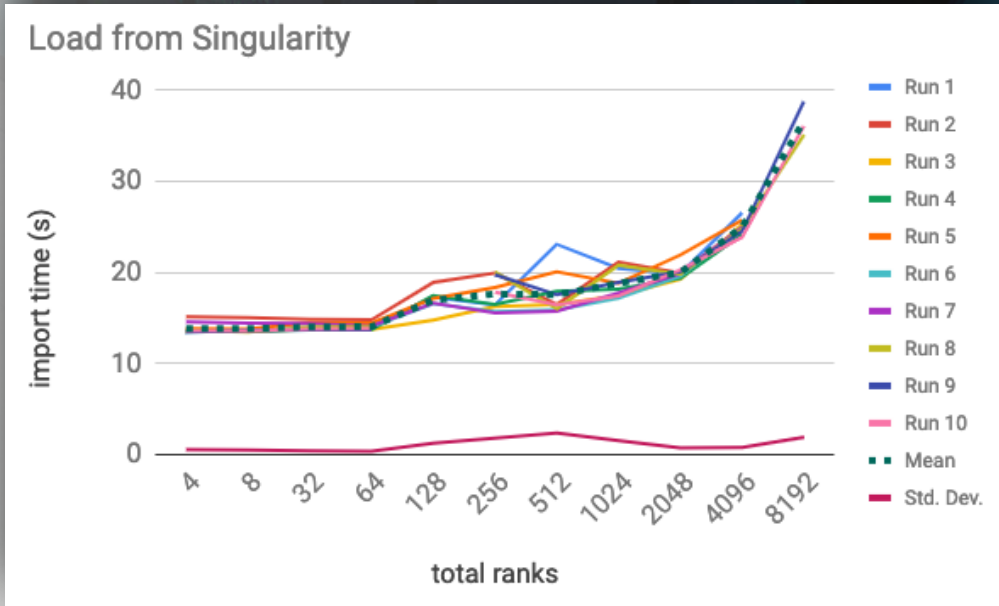
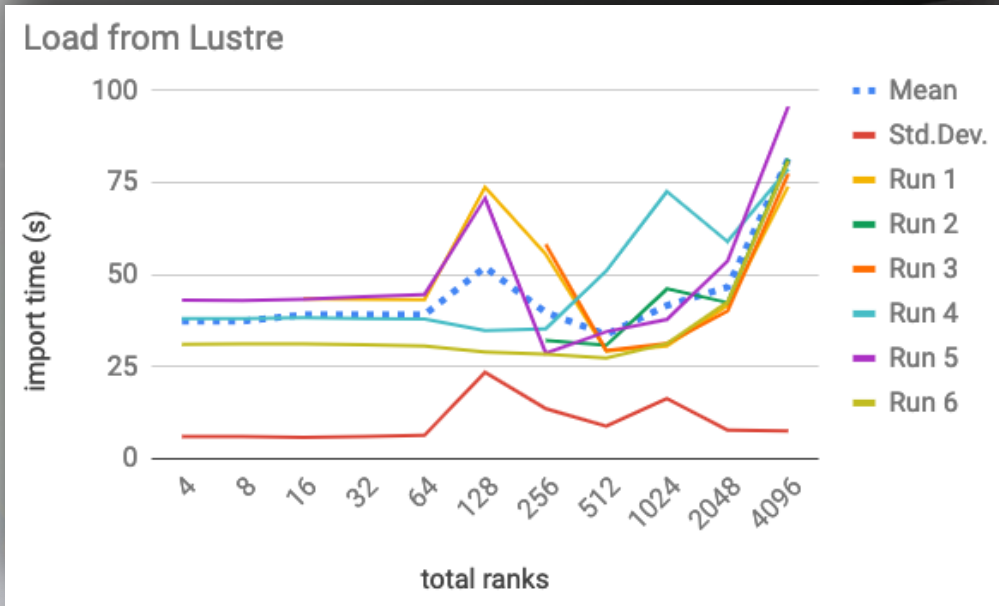
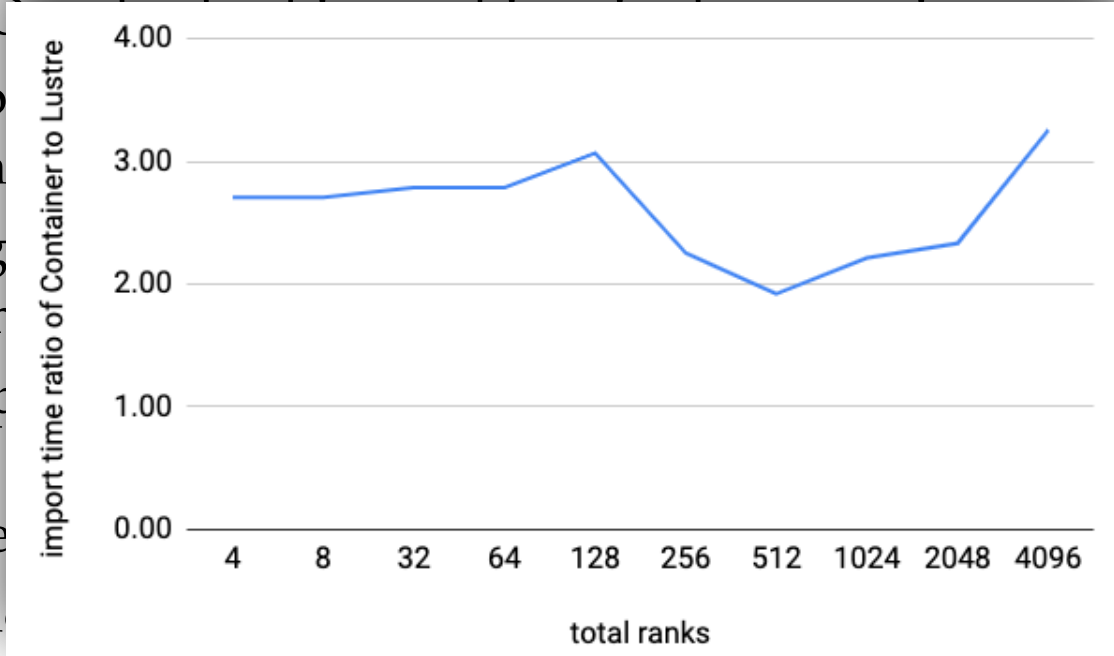
Singularity Performance on Theta

- File IO on leadership machines is always an issue.
- Python imports can be time consuming with thousands of ranks loading modules in parallel.
- Using a Miniconda install of Tensorflow/Keras, I tested the import times for the 'keras_mnist.py' test example
- The plots show the performance across a few runs when running on the Lustre filesystem versus running inside a container.
- The load time from Lustre varies unpredictably due to the varying load on the Lustre meta data server.
- The load time from within the container is generally more than 2x faster.



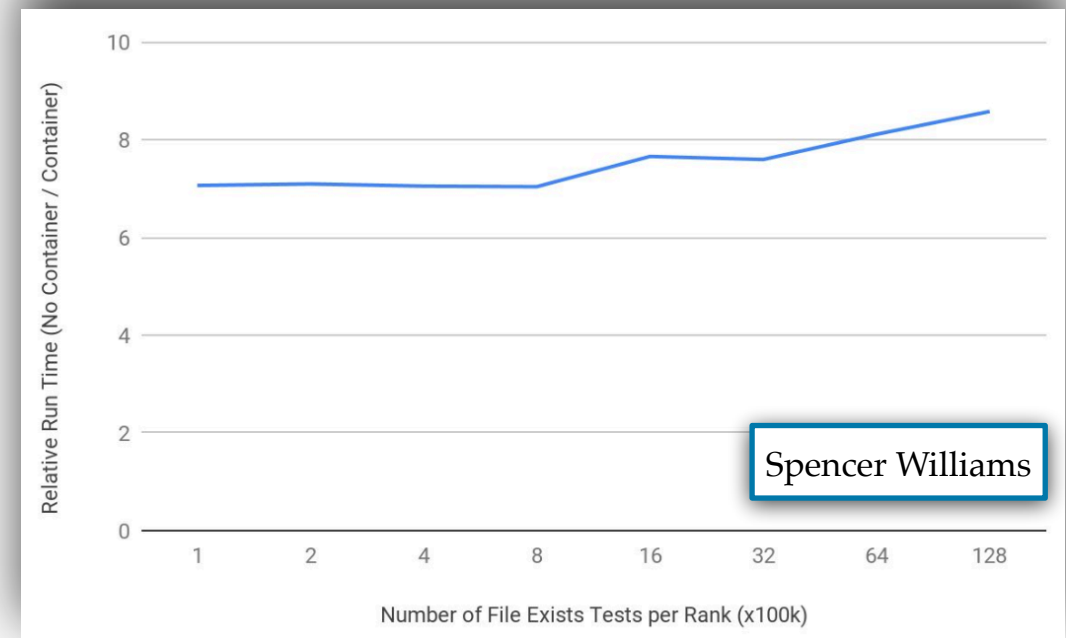
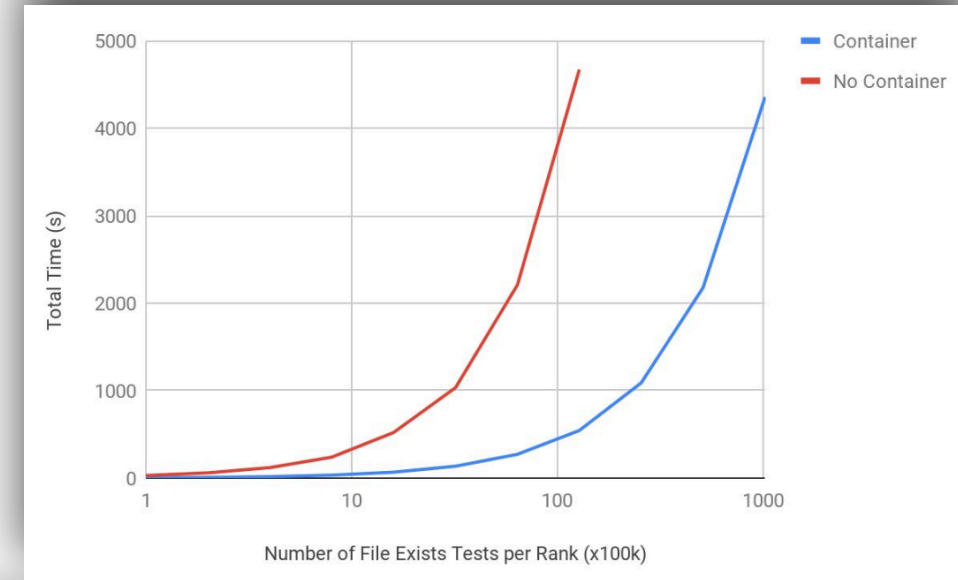
Singularity Performance on Theta

- File IO
- Python
- of ran
- Using
- the im
- The p
- when
- inside
- The 1
- the varying load on the Lustre meta data server.
- The load time from within the container is generally more than 2x faster.



Meta-data Performance in Containers

- More generally we've seen that containers help hide excessive meta-data access.
- Lustre only has one meta-data server and therefore can be a bottleneck for non-optimized codes.
- We always recommend statically linking applications for this reason.
- we performed a simple test by doing an 'os.path.exists()' call on files that are either inside a container or directly on Lustre
- The 'stat'-ing of files inside the container compared to Lustre are 7x faster.



Filesystem Customizations for Theta

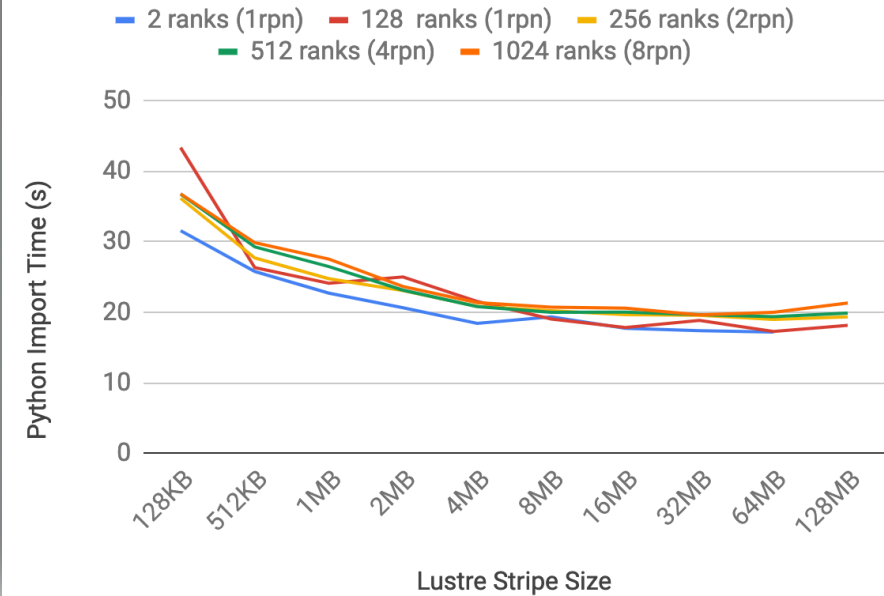
- Use Lustre striping to improve filesystem performance during training
- First create a directory that will be striped across multiple Lustre sources

```
lfs setstripe -c 50 -stripe-size 8m [samples directory]
```

- Then copy the input files into this directory

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

Container with Lustre Striping



Containers On Theta/Cooley

- On Theta and Cooley you can find pre-existing containers here:
 - `/soft/datascience/singularity/`
- On Theta, you'll find a Miniconda installation of Tensorflow/PyTorch/Horovod for scalable ML
- On Cooley, you'll find Tensorflow and PyTorch containers
- Each have example submit scripts

- More documentation is at:
 - <https://www.alcf.anl.gov/user-guides/singularity>
 - <https://www.alcf.anl.gov/user-guides/singularity-cooley>



Summary

- Singularity Containers support on Theta/Cooley
- Containers can be derived from Docker images
- Building custom containers must be done off site
- Performance benefits exist related to filesystem meta data access
- Need help? Have Questions? Email: datascience@alcf.anl.gov

