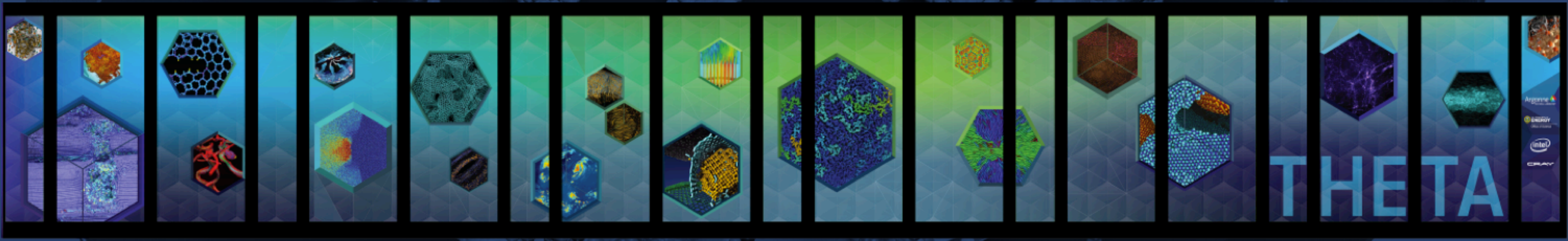
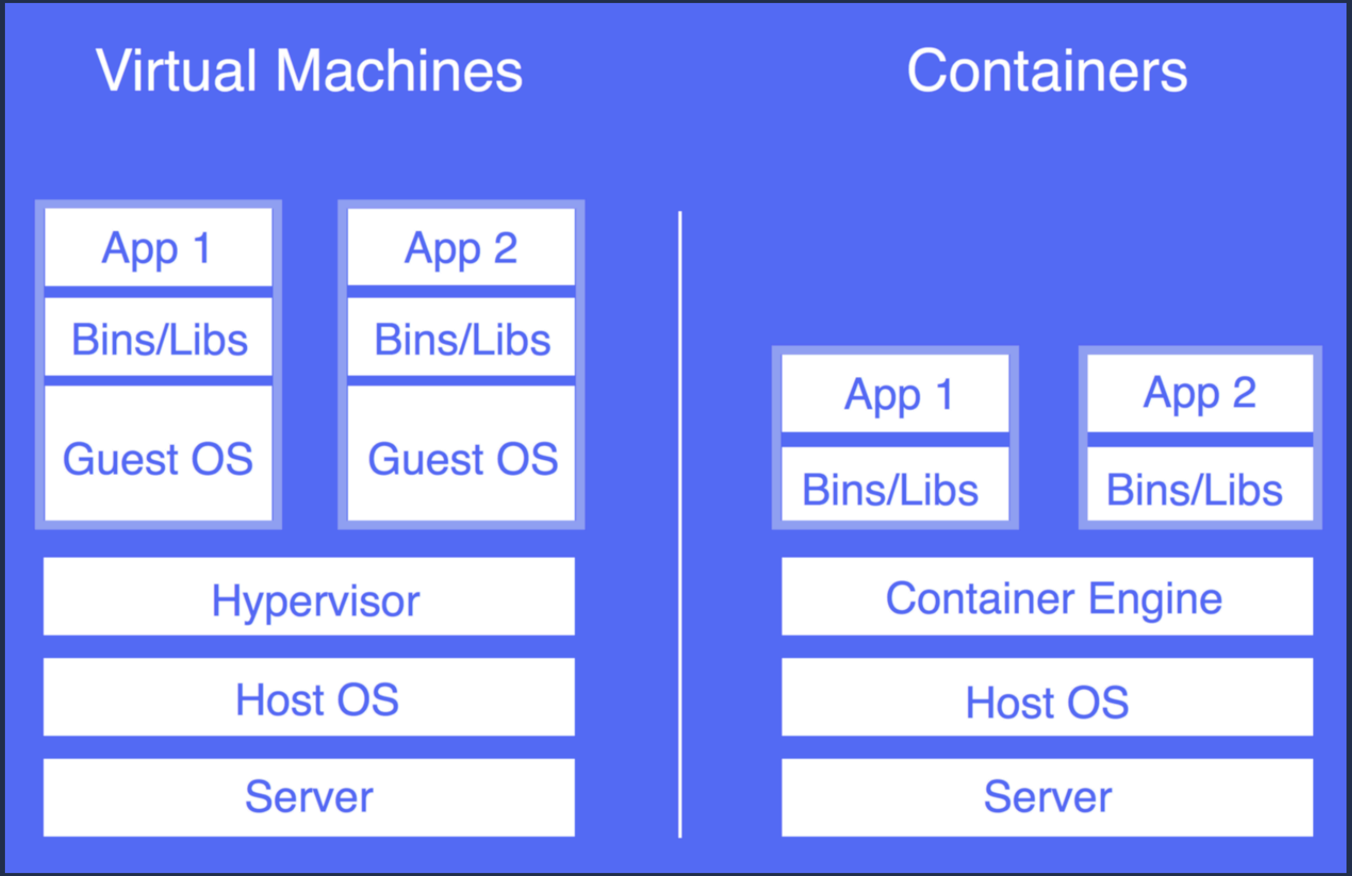
Using Containers on Theta

J. Taylor Childers

SIMULATION.DATA.LEARNING WORKSHOP

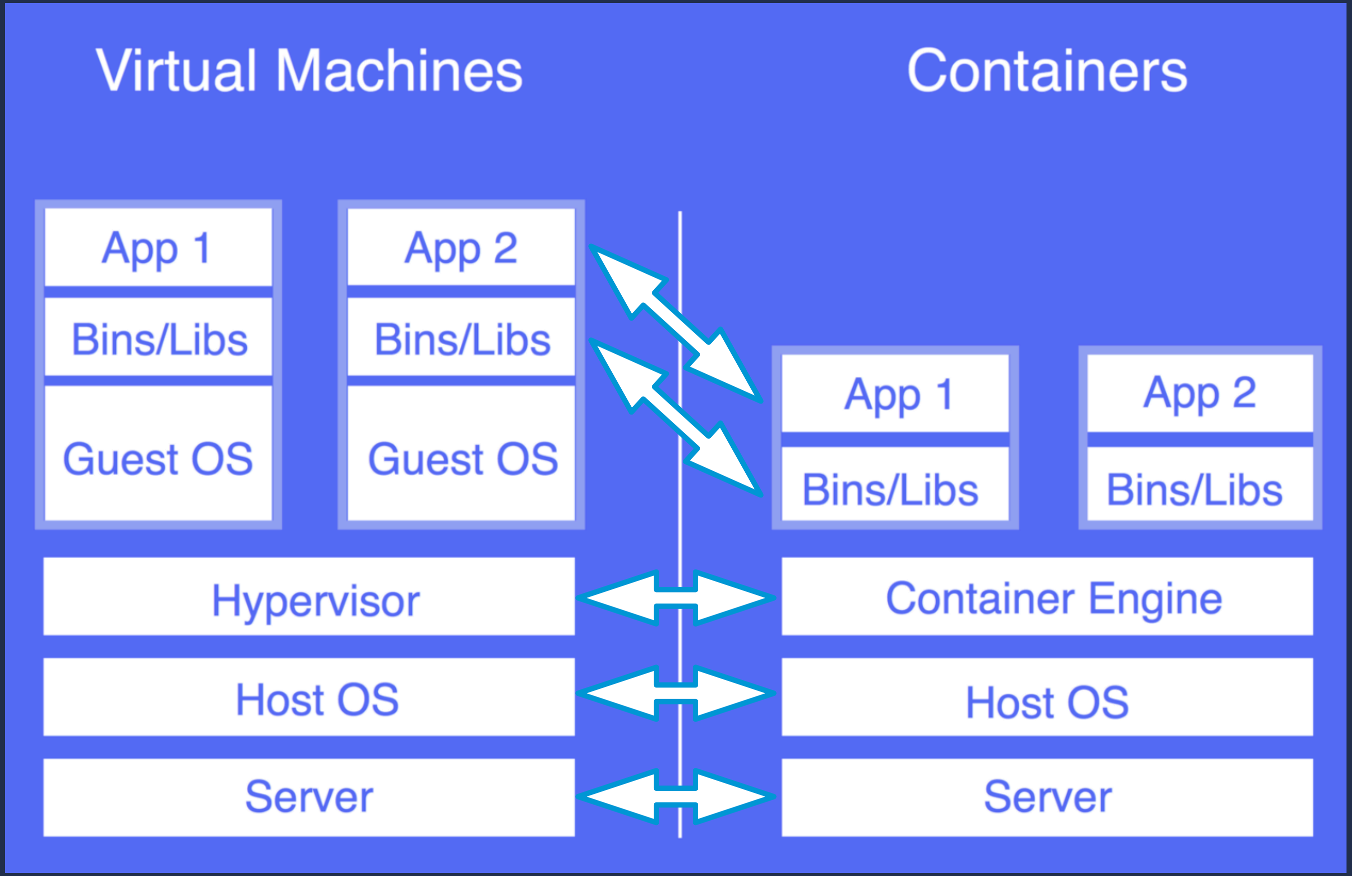


Quick Introduction



Quick Introduction

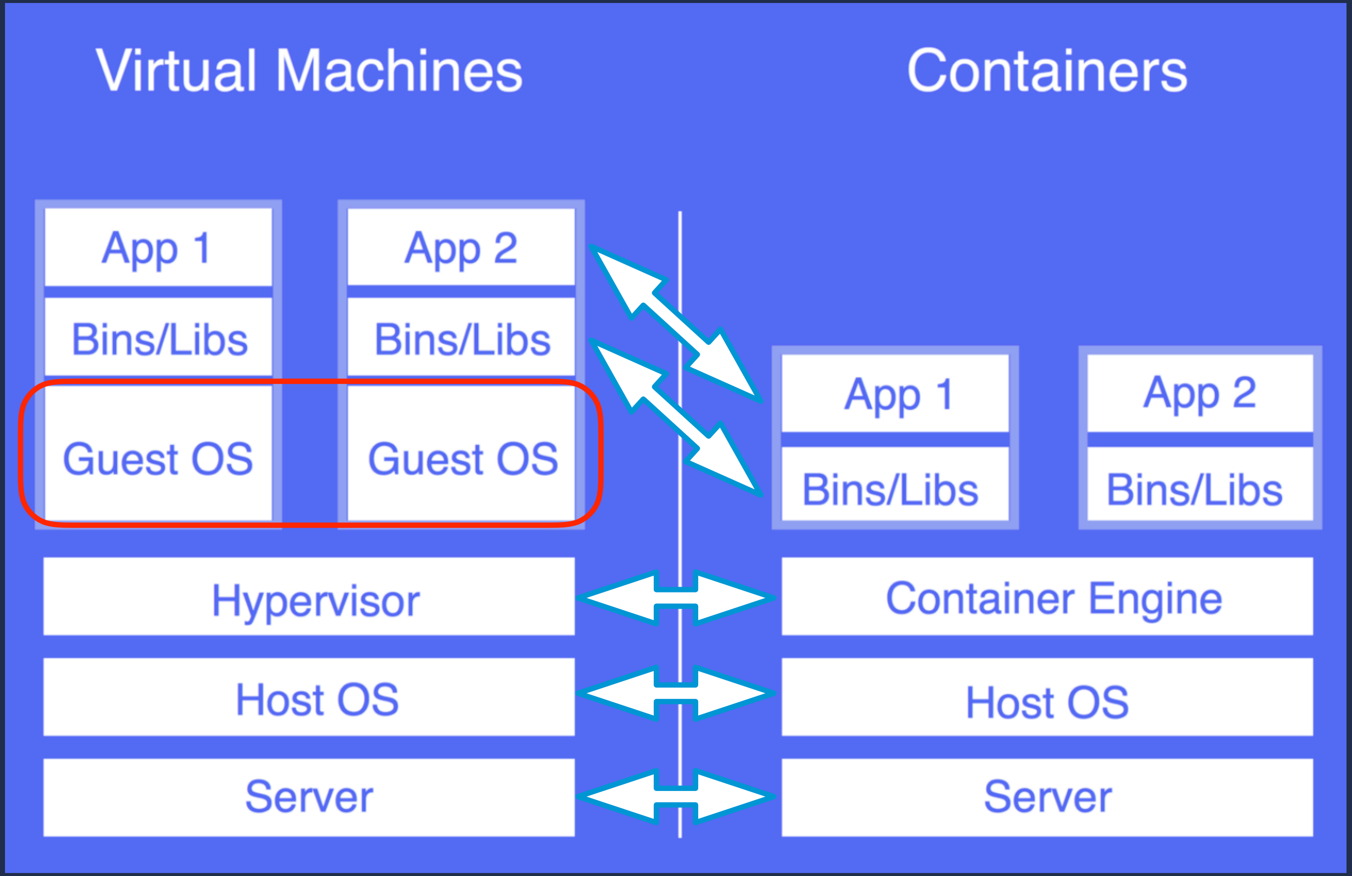
Both Require:
- Hardware
- Host Operating System
- Hypervisor or Engine
- System libraries
- Target Application



Quick Introduction

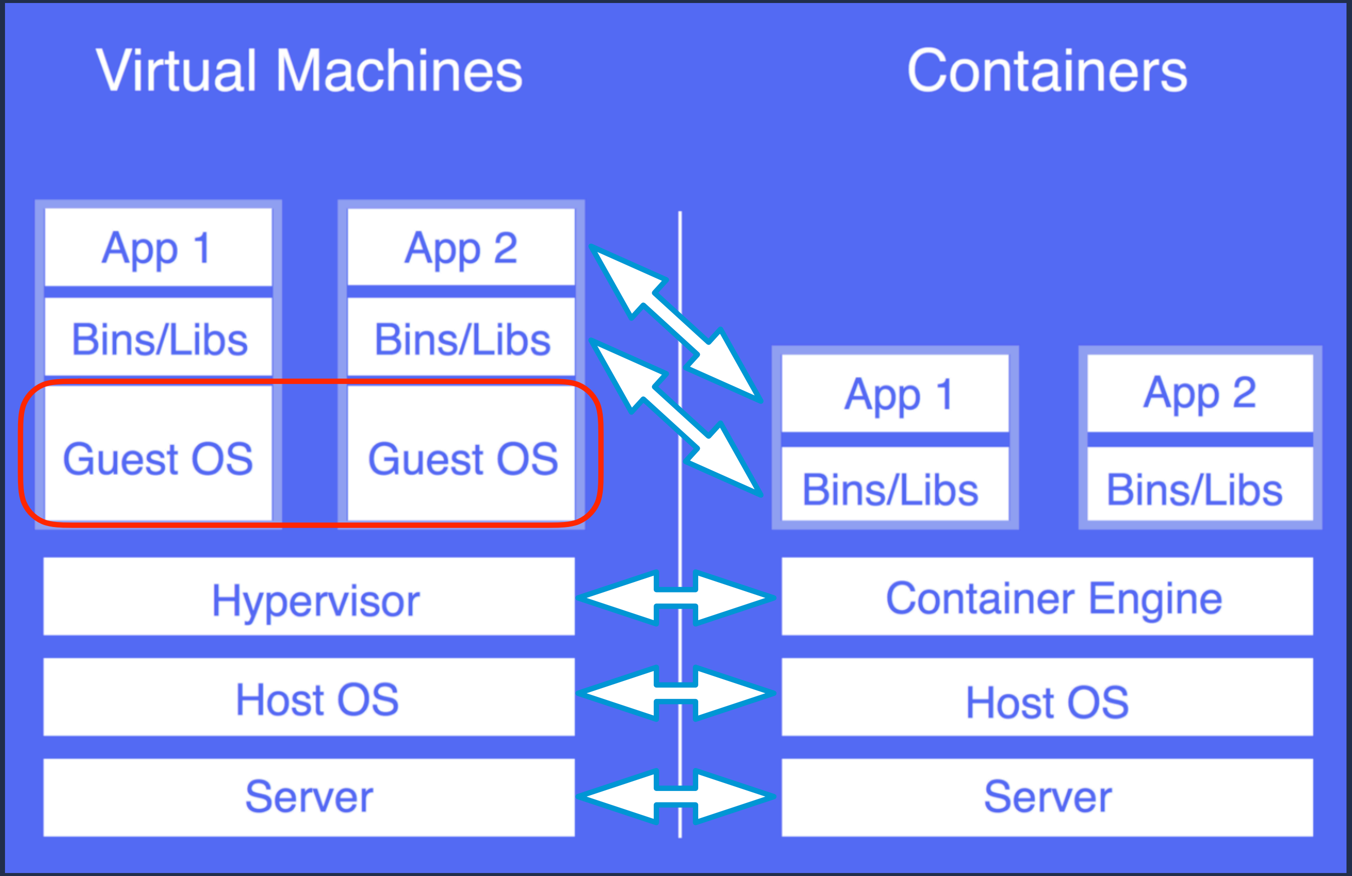
Both Require:
- Hardware
- Host Operating System
- Hypervisor or Engine
- System libraries
- Target Application

Main Difference:
- VMs require entire internal operating system
- VMs virtualize system hardware



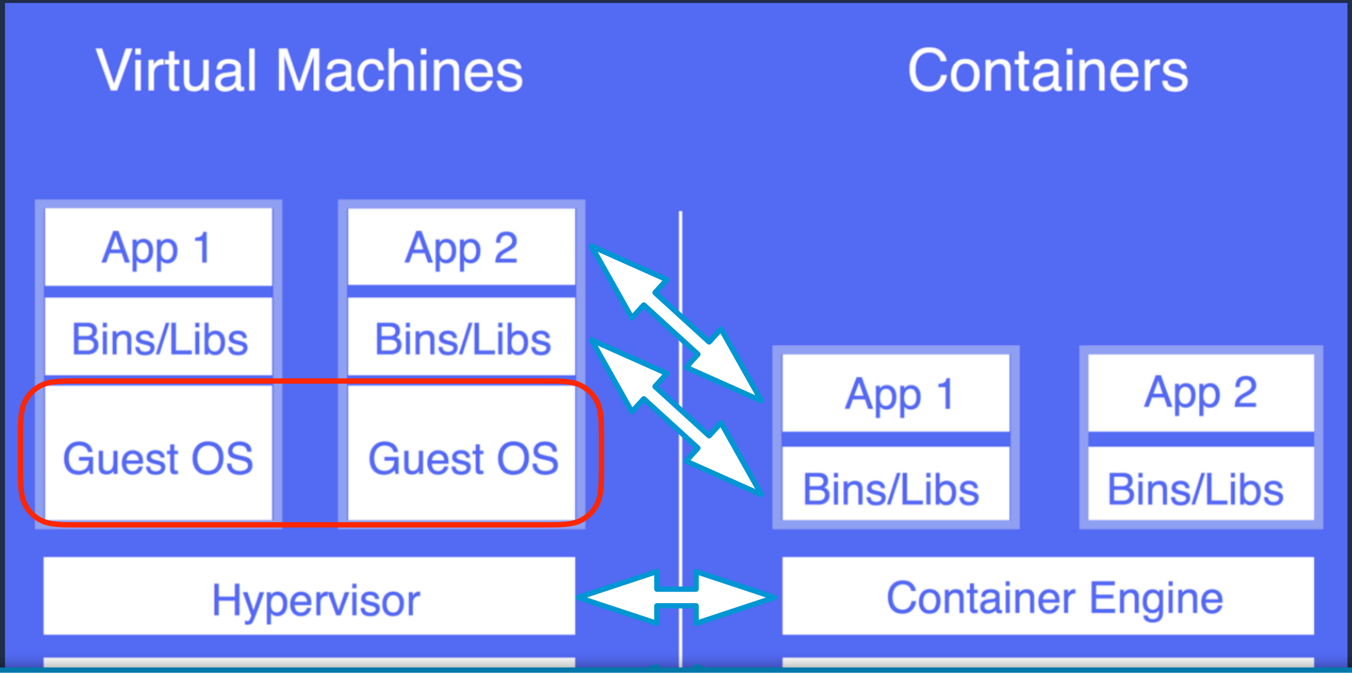
Quick Introduction

Containers use host kernel making them lighter weight, quicker to deploy.



Quick Introduction

Containers use host kernel making them lighter weight, quicker to deploy.

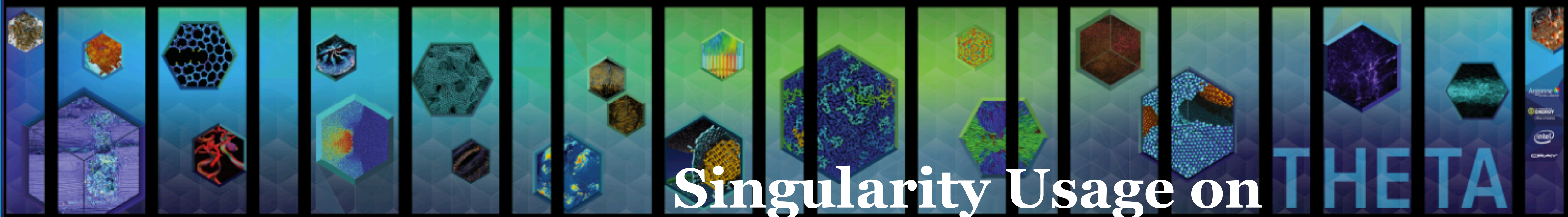


[IBM Performance Tests]

"In general, Docker equals or exceeds KVM performance in every case we tested. Our results show that both KVM and Docker introduce negligible overhead for CPU and memory performance (except in extreme cases). For I/O intensive workloads, both forms of virtualization should be used carefully."

Docker vs Singularity

- Docker and Singularity are both container frameworks
- Both are easy to use and deploy
- Why not Docker on Theta?
  - Applications run as root inside container
  - Since containers can mount host folders, container can mount local filesystem as root with all the access privileges
  - Perhaps OK if you are Google and have no outside users running apps on your system
  - This is not OK for DOE user facilities
- Singularity containers run as the user and cannot escalate privileges
  - Otherwise come with all the benefits of Docker



Singularity Usage on THETA

http://singularity.lbl.gov/

Building Containers:

- Singularity containers should be built from base images
- Base images can be found on
  - https://hub.docker.com/
  - https://singularity-hub.org/
- Example build commands:

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

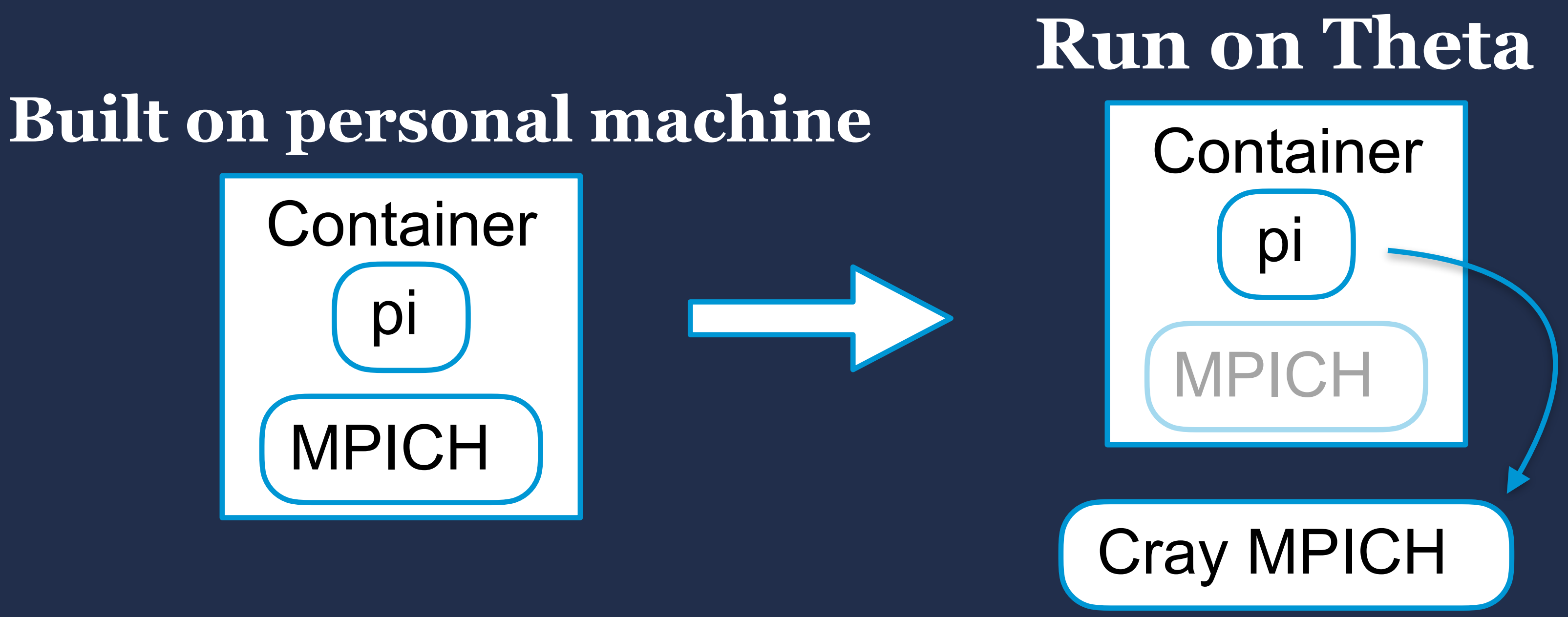
- This can be done on a Theta login node if you can use base images produced by Docker or Singularity.
- There is a known bug in Singularity which causes user uploaded images to fail with ‘permission denied’ errors:

```
thetalogin5:~> singularity build myubuntu.img docker://jtchilders/mpitest:latest
```

- This succeeds if you have ‘sudo’ rights, therefore…

Overview of the Workflow in Six Easy Steps!

1. Install Singularity on machine with ‘sudo’ access
2. Create SingularityFile recipe
3. Run Build command with ‘sudo’
4. Copy to Theta
5. Create Cobalt submission script
6. ‘qsub’ script



Singularity Usage on Theta

Building containers from Scratch:

- **Need a machine with Singularity installed and 'sudo' rights**
  - **Your laptop will work**
- **Create a Singularity recipe file**

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /vagrant_data/pi.c /myapp/
   /vagrant_data/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

Source of base image

Similar to docker://centos

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

Create a working directory for my app

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
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

Copy files from into image

Left-hand side is host file system path, Right-hand side is image path

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

Install via ‘yum’ any packages need to build application inside the container.

Commands to install my image with the application.

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

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
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

Source of base image

Create a working directory for my app

Copy files from into image

Commands to install my image with the application.

Specify the executable to run with container is called

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%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
   mkdir ${SINGULARITY_ROOTFS}/myapp

%files
   /host/path/to/myapp/pi.c /myapp/
   /host/path/to/myapp/build.sh /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
   ./build.sh

%runscript
   /myapp/pi
```

```
Bootstrap: docker
From: centos

%setup
   mkdir ${SINGULARITY_ROOTFS}/myapp
```

```
#!/bin/bash
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
```

```
.c /myapp/
ld.sh /myapp/

evelopment Tools"

   ./build.sh

%runscript
   /myapp/pi
```

```
#!/bin/bash
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**

Actual Build Command

```
> sudo singularity build myapp.img SingularityFile
```

Running Singularity Container on Theta

- Copying container to Theta (my image was 225MB)
- Run the following

```
> 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 1
#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 1
#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 1
#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

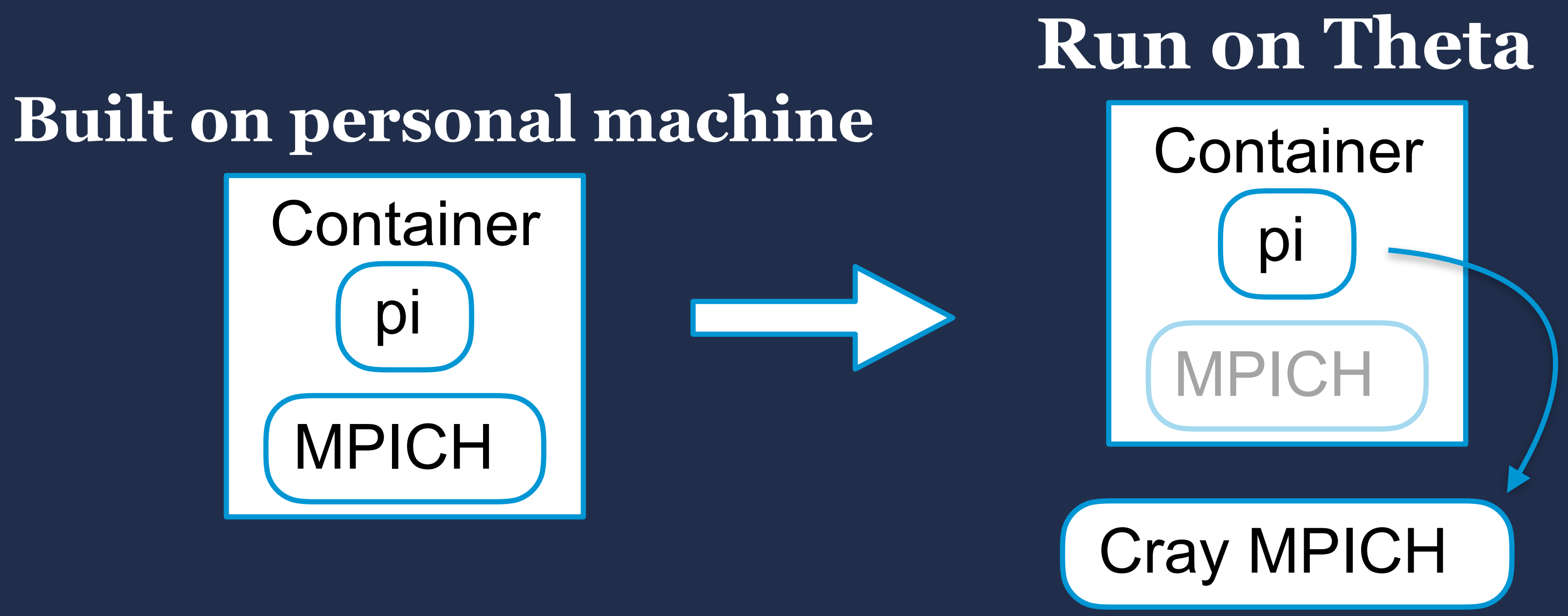
Running Singularity Container on Theta

```
#!/bin/bash
#COBALT -t 30
#COBALT -q debug-cache-quad
#COBALT -n 1
#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 /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
```

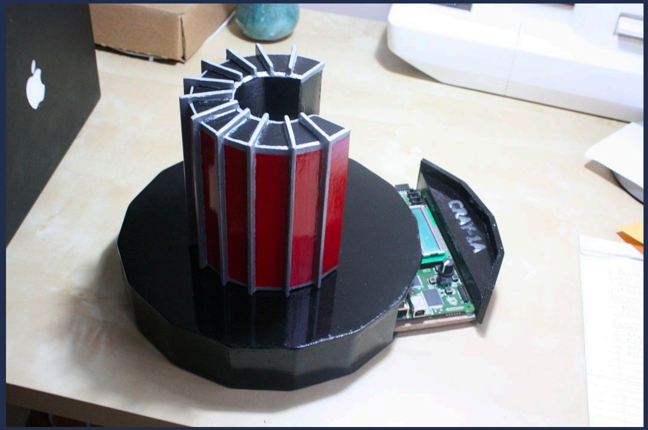
`-B /opt:/opt:ro` causes Singularity to mount the host '/opt' inside the container at '/opt' in read-only (ro) mode. This allows the use of cray libraries that are needed to take advantage of Theta's unique hardware.

Six Easy Steps!

1. Install Singularity on machine with ‘sudo’ access
2. Create SingularityFile recipe
3. Run Build command with ‘sudo’
4. Copy to Theta
5. Create Cobalt submission script
6. ‘qsub’ script



Workflow Option #2: Cray Container



- We have a second way to build containers on Theta
- Created container with entire Theta Cray Environment
  - 6GB image
- Can reach out to Derek Jensen if you would like to use it
- Can not be made publicly because Cray software is proprietary.
- Otherwise, the workflow is similar:
  - Copy image to personal machine
  - Create Singularity recipe to copy application into new container and build it against cray environment
  - Build container
  - Copy to Theta
  - Create Cobalt submission script
  - Submit Job

Workflow Option #2

```
Bootstrap: localimage
From: ./cray_base.simg


%files
   ./pi/

%labels
   Version pe_17.11-8-4

%environment
   MODULEPATH=/opt/cray/pe/perftools/6.5.2/modulefiles:/opt/cray/pe/craype/2.5.13/modulefiles:/opt/cray/pe/
modulefiles:/opt/cray/modulefiles:/opt/modulefiles:/opt/cray/pe/craype/default/modulefiles/:/opt/cray/ari/
modulefiles/:/opt/cray/ari/modulefiles

%post
   bash
   source /opt/cray/pe/modules/default/init/bash
   export MODULEPATH=$MODULEPATH:/opt/cray/pe/craype/default/modulefiles/:/opt/cray/ari/modulefiles/
   module load PrgEnv-cray
   module load craype-network-aries
   module load craype-mic-knl
   module list
   cd pi
   make

%runscript
/pi/pi
```

No need to install packages, just module load them

Summary

- Currently recommending one of two workflows:
  - Build Singularity Container on your own machine, using generic base images, import to Theta
  - Build Singularity Container based on the the Cray Container , import to Theta
- When Singularity bug is fix, could also build Docker image on your own machine and do ‘singularity build’ directly on Theta.

Argonne NATIONAL LABORATORY