Using Containers on the Cannon Cluster : Singularity/Apptainer

RCHelp: rchelp@rc.fas.harvard.edu

Objectives

- Why containers?
- Singularity/Apptainer container system
- How to run Singularity containers on Cannon:
  - Running simple containers on cpus and GPUs
  - Serial, multicore and MPI applications
  - Running graphics with hardware rendering
- How to build your own containers
  - building local images
  - remote builds
- Bind mounts

Why Containers?

Deploying Applications:

Building software is often a complicated business, particularly on a shared and multi-tenant systems:

- HPC clusters have typically very specialized software stacks which might not adapt well to general purpose applications.
- OS installations are streamlined.
  Some applications might need dependencies that are not readily available and complex to build from source.
- End users use Ubuntu or Arch, cluster typically use RHEL, or SLES, or other specialized OS.
  (... "`>$ sudo apt-get install`" will not work)
- Researcher's code often tends to comes from some old repos.

What problems are we trying to solve?

Portability and Reproducibility:

- Running applications on multiple systems typically needs replicating the installations multiple times making it hard to keep consistency.

- It would be useful to publish the exact application used to run a calculation for reproducibility or documentation purpose.

- As a user can I minimize the part of the software stack I have no control on, to maximize reproducibility without sacrificing performance too much?

Containers: easi”er” software deployment

Containers provide a potential solution…. or at the very least can help.

- Easier software deployment:
  Users can leverage on installation tools that do not need to be available natively on the runtime host
  (e.g. package managers of various linux distributions).
- Software can be built on a platform different from the exec hosts.
- they package in one single object all necessary dependencies.
- easy to publish and sign
- they are portable **
  - … provided you run on a compatible architecture
  - access to special hardware needs special libraries also inside the container , which at the moment limits portability

Types of Containers

docker

Linux LXC

OpenVZ
Linux Containers

podman

rkt

Charliecloud

SHIFTER

apptainer.org

General purpose / microservice Oriented.

HPC Oriented :

- Compatible with WLM
- No privilege escalation needed

Singularity vocabulary

- Singularity or Apptainer – the software
  - As in “Singularity 3.8” or “Apptainer 1.0”
- Image – a compressed, usually read-only file
  - Example: “Build a Matlab 2021a image”
  - Writable image: use `--sandbox` option
- Container
  - The technology: “containers vs. virtual machines”
  - An instance of an image
    - Example: “process my data in a Singularity container of Matlab
- Host – computer/supercomputer where the image is run

Adapted from LSU Singularity training slides:
http://www.hpc.lsu.edu/training/weekly-materials/2022-Spring/HPC_Singularity_Spring2022.pdf

Singularity and Apptainer

Singularity/Apptainer provides a container runtime and an ecosystem for managing images that is suitable for multi-tenant systems and HPC environments.

Important aspects :

- no need to have elevated privileges at runtime, although root privileges are needed to build the images.
- each applications will have its own container
- containers are not fully isolated (e.g. host network is available)
- users have the same uid and gid when running an application
- containers can be executed from local image files, or pulling images from a docker registry

For basic usage refer to:

https://docs.rc.fas.harvard.edu/kb/singularity-on-the-cluster/
https://www.sylabs.io/docs/
https://apptainer.org/

some examples

/n/holyscratch01/shared/simple-examples-singularity

```
[simple-examples-singularity]$ tree .
├── build-simple-image
│   ├── lolcow.def
│   └── lolcow.sif
├── example-gpu
│   ├── sbatch-run-cuda.sh
├── example-mpi
│   ├── example-mpi.sif
│   ├── mpi.def
│   ├── mpi_pi.c
│   └── sbatch_run.sh
├── example-openmp
│   ├── example-openmp.sif
│   ├── omp_dot2.c
│   ├── omp_pi.c
│   ├── openmp.def
│   ├── sbatch_run_dot2.sh
│   └── sbatch_run_omp_pi.sh
├── localtime
└── remote-build
    ├── example-omp.sif
    └── openmp-remote.def
```

Subcommands and options

The command structure is

`singularity` `[global option]` **`subcommand`** `[subcommand option]` **`image`** `[args]`

(see “singularity help” for all available subcommands and specific options)

The most frequent subcommands are

- `build` : to build an image local or remote
- `shell` : to start an interactive shell in a container
- `exec` (or `run`) : to execute code in a container

most frequent options for “`shell`” and “`exec`” are to control storage mounts (`--bind`) at runtime.

Example : running a simple container

```
# request interactive job
user@login-node example-1]$ salloc --mem=4G -p test -N 1 -t 60

[user@holy7c24401 example-1]$ singularity run docker://r-base R
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob 2aa31e5eaa2f done
….. omitted output
2021/05/04 14:34:03  info unpack layer:
sha256:debce101a705a0b612a556daedb6d93ca5b3ec0d8cc3b20b57b8e22ca7e64d1b
INFO:    Creating SIF file...
R version 4.0.5 (2021-03-31) -- "Shake and Throw"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
>

[user@holy7c24401 example-1]$ singularity run docker://r-base R -q
INFO:    Using cached SIF image
>
```

Cache folder

When using images generated from remote sources singularity will cache layers and converted images under ~/.singularity

```
[user@holy7c24401 example-1]$ singularity cache list
There are 5 container file(s) using 7.77 GiB and 95 oci blob file(s) using 7.93 GiB of space
Total space used: 15.70 GiB

[user@holy7c24401 example-1]$ ls -lrtha ~/.singularity/cache/oci-tmp/
total 9.0G
... omitted output
-rwxr-xr-x 1 francesco rc_admin  91M Apr 16 11:07
c16e3a79b79dbf1e825c56485ac26e22290db3ad4c69eca4e18e0cc957c02548
-rwxr-xr-x 1 francesco rc_admin 294M May  4 10:34
ff41c917e639685e53adf9a881f000c97e7f172e9bde6b1d0ab854db6ca6b593
drwx------ 3 francesco rc_admin  410 May  4 10:34.
```

r-base image just pulled

```
[user@holy7c24401 example-1]$ singularity cache clean
This will delete everything in your cache (containers from all sources and OCI blobs).
Hint: You can see exactly what would be deleted by canceling and using the --dry-run option.
Do you want to continue? [N/y] y
```

You can control location of cache with the variable `SINGULARITY_CACHEDIR`

https://docs.sylabs.io/guides/3.8/user-guide/build_env.html

Multicore applications

No difference from running natively built applications.

You can use “srun -c $SLURM_CPUS_PER_TASK”, OMP_NUM_THREDS, or other tools from your code to control multicore performance.

```
[user@holy7c24401 example-openmp]$ cat sbatch_run_dot2.sh
#!/bin/bash
#SBATCH -o omp_dot2.out
#SBATCH -t 0-00:30
#SBATCH --mem=4000
# Run program
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK singularity exec ./example-openmp.sif /opt/bin/omp_dot2.x

[user@holy7c24401 example-openmp]$ sbatch -p test -c 4 sbatch_run_dot2.sh
Submitted batch job 26602599

[user@holy7c24401 example-openmp]$ cat omp_dot2.out
Running on 4 threads.
Thread 0: partial dot product = 128300.000000
Thread 3: partial dot product = 202550.000000
Thread 2: partial dot product = 175300.000000
Thread 1: partial dot product = 150550.000000
Global dot product = 656700.000000
```

MPI applications

There are several ways of running MPI applications with singularity (https://docs.sylabs.io/guides/3.8/user-guide/mpi.html)

We recommend hybrid mode (application from container, mpirun from host)

```
[user@holy7c24401 example-mpi]$ cat sbatch_run.sh
#!/bin/bash
#SBATCH -J mpi_pi
#SBATCH -o mpi_pi.out
#SBATCH -e mpi_pi.err
#SBATCH -t 0-00:30
#SBATCH -n 16
#SBATCH -N 1
#SBATCH --mem-per-cpu=1000
#SBATCH -p test

# Run program
for i in 1 2 4 8 16
do
    echo "Number of processes: ${i}"
    mpirun -np $i singularity  exec
./example-mpi.sif  /opt/bin/mpi_pi.x 1000000000
    echo " "
done
```

```
[user@holy7c24401 example-mpi]$ sbatch sbatch_run.sh
Submitted batch job 26602885

[user@holy7c24401 example-mpi]$ cat mpi_pi.out  |
grep -A1  Number
Number of processes: 1
Elapsed time = 17.859150 seconds
--
Number of processes: 2
Elapsed time = 8.926302 seconds
--
Number of processes: 4
Elapsed time = 4.479907 seconds
--
Number of processes: 8
Elapsed time = 2.253929 seconds
--
Number of processes: 16
Elapsed time = 1.134280 seconds
```

GPU computing

The option "**--nv**" allows to automatically import in the container all you need to run your cuda based application.

```
[user@boslogin04 example-gpu]$ cat sbatch-run-cuda.sh
#!/bin/bash
#SBATCH -p gpu_test
#SBATCH -t 30
#SBATCH -o tf_example.out
#SBATCH --gres=gpu:1
#SBATCH --gpu-freq=high
#SBATCH --mem=8G

export
myimage=/n/singularity_images/FAS/nvidia-ngc/tensorflow/tensor
flow_19.10-py3.sif

[ ! -d ./benchmarks ] &&  git clone --branch
cnn_tf_v1.14_compatible
https://github.com/tensorflow/benchmarks.git

cd  benchmarks/scripts/tf_cnn_benchmarks
singularity exec --nv $myimage python tf_cnn_benchmarks.py
--num_gpus=1 --batch_size=32 --model=resnet50
--variable_update=parameter_server

[user@boslogin04 example-gpu]$ sbatch sbatch-run-cuda.sh
Submitted batch job 26603987
```

```
[user@boslogin04 example-gpu]$ tail -20  tf_example.out
==========
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sectotal_loss
1       images/sec: 310.9 +/- 0.0 (jitter = 0.0)   8.169
10      images/sec: 310.1 +/- 0.2 (jitter = 0.4)   7.593
20      images/sec: 310.4 +/- 0.2 (jitter = 0.6)   7.696
30      images/sec: 310.4 +/- 0.1 (jitter = 0.6)   7.753
40      images/sec: 310.4 +/- 0.1 (jitter = 0.4)   8.007
50      images/sec: 309.2 +/- 1.0 (jitter = 0.7)   7.520
60      images/sec: 309.4 +/- 0.9 (jitter = 0.6)   7.989
70      images/sec: 309.5 +/- 0.7 (jitter = 0.5)   8.027
80      images/sec: 309.6 +/- 0.6 (jitter = 0.6)   7.932
90      images/sec: 309.5 +/- 0.6 (jitter = 0.6)   7.851
100     images/sec: 309.2 +/- 0.6 (jitter = 0.6)   7.798
----------------------------------------------------------------
total images/sec: 308.94
----------------------------------------------------------------
```

Example : running with accelerated graphics

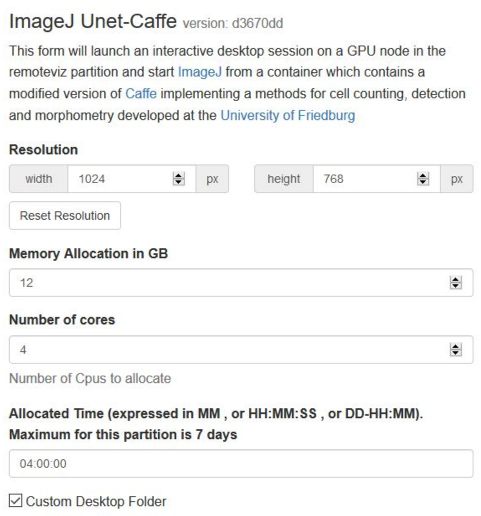
This needs to be run on a gpu node where gpu accepts graphics tasks and with a graphics server running

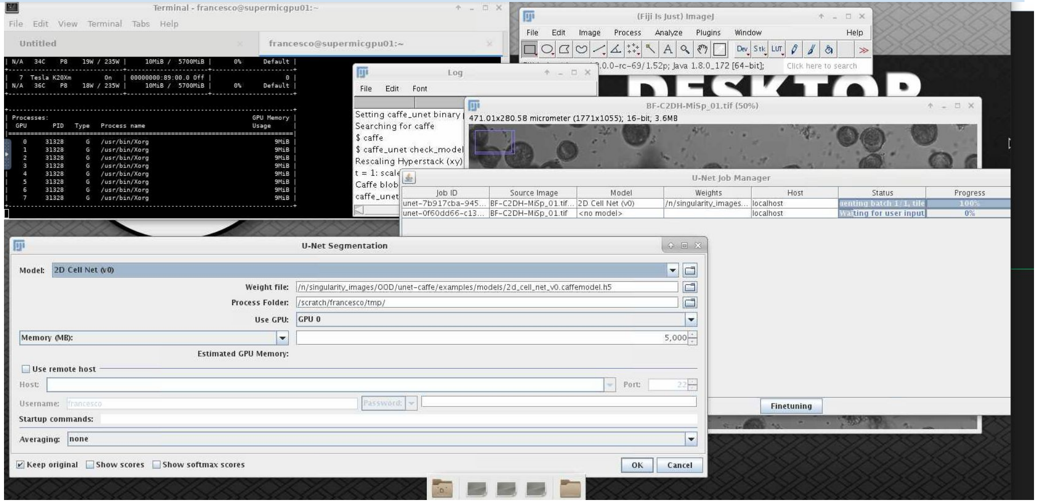
(remoteviz partition)

>$ singularity exec **--nv** /n/singularity_images/OOD/unet-caffe/unet-caffe-fiji_latest-2020-03-23.sif /bin/bash -c "**vglrun** ImageJ-linux64"

vglrun needs to be installed **inside** the container.

example of vdi app : https://gitlab-int.rc.fas.harvard.edu/openondemand/imagej-unet-caffe





Building images locally (i): the Singularity way

1. Build Locally in your own laptop/computer

To do this you need to be on your own development environment where you have elevated privileges **and** Singularity installed

```
[myLinuxSystem ]> sudo /usr/local/bin/singularity build some_imagename.sif
some_definition_file.def
```

2. Build locally using Docker to build singularity. (only if you understand the options)

If you don't have a linux system available (or can't run VMs on your laptop), you can try to use docker to run singularity.

```
$> docker run --privileged  --rm -ti -v $PWD/examples:/mnt  -v $PWD/localtime:/etc/localtime --entrypoint /bin/bash
quay.io/singularity/docker2singularity
bash-5.0# cd /mnt/
bash-5.0# singularity build lolcow.sif lolcow.def
INFO:    Starting build...
... omitted output
INFO:    Build complete: lolcow.sif
bash-5.0# singularity run lolcow.sif
 _______________________________________
/ Your true value depends entirely on \
\ what you are compared with.          /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Singularity definition file example

```
$ cat build-simple-image/lolcow.def
Bootstrap: docker
From: ubuntu:16.04

%post
    apt-get -y update
    apt-get -y install fortune cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    fortune | cowsay | lolcat
```

https://docs.sylabs.io/guides/3.8/user-guide/definition_files.html

Building images locally: (i) the Singularity way

Build in your own laptop/computer

To do this you need to be on your own development environment where you have elevated privileges **and** Singularity installed

```
[myLinuxSystem]> sudo singularity build some_imagename.sif  some_definition_file.def
```

Singularity definition file example:

```
$ cat build-simple-image/lolcow.def
Bootstrap: docker
From: ubuntu:16.04

%post
    apt-get -y update
    apt-get -y install fortune cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    fortune | cowsay | lolcat
```

Building images locally: (ii) the Docker way

Use Docker to build your local image and then convert it to singularity :

- build your image in docker from a Dockerfile, or interactively
- convert the image with docker2singularity

Example :

```
[user@mylaptop ~]$ docker build -t rstudio_rockerverse_fasrc:4.0.0 .
[user@mylaptop ~]$ docker run -v /var/run/docker.sock:/var/run/docker.sock \
                                  -v /data/sing-images-tmp/:/output -v /data/tmp:/tmp \
                                  --privileged -t --rm quay.io/singularity/docker2singularity rstudio_rockerverse_fasrc:4.0.0
```

Using docker can be convenient because the tools are very mature and it’s very simple to modify images, add layers, …

https://github.com/singularityhub/docker2singularity

Building images remotely with Sylabs cloud

Build remotely directly from Cannon to singularity cloud

You can do it on Cannon, but you need to first (detailed FASRC docs):

- sign up for an account on https://cloud.sylabs.io/library
- generate a token
- use “singularity login” to login

```
[user@login-node]$ salloc --mem=4000   -p test -N 1 -t 60
[user@compute-node]$ singularity remote login
[user@compute-node]$ singularity build --remote some_imagename.sif some_definition_file.def
```

This will use your definition file to build the image in Sylabs cloud and download it to your local folder on Cannon

Note that you likely need to adapt some of the sections of your definition file:

- `%files` section: copy files from outside of the container into the container
- `%post` section: download and install software and libraries

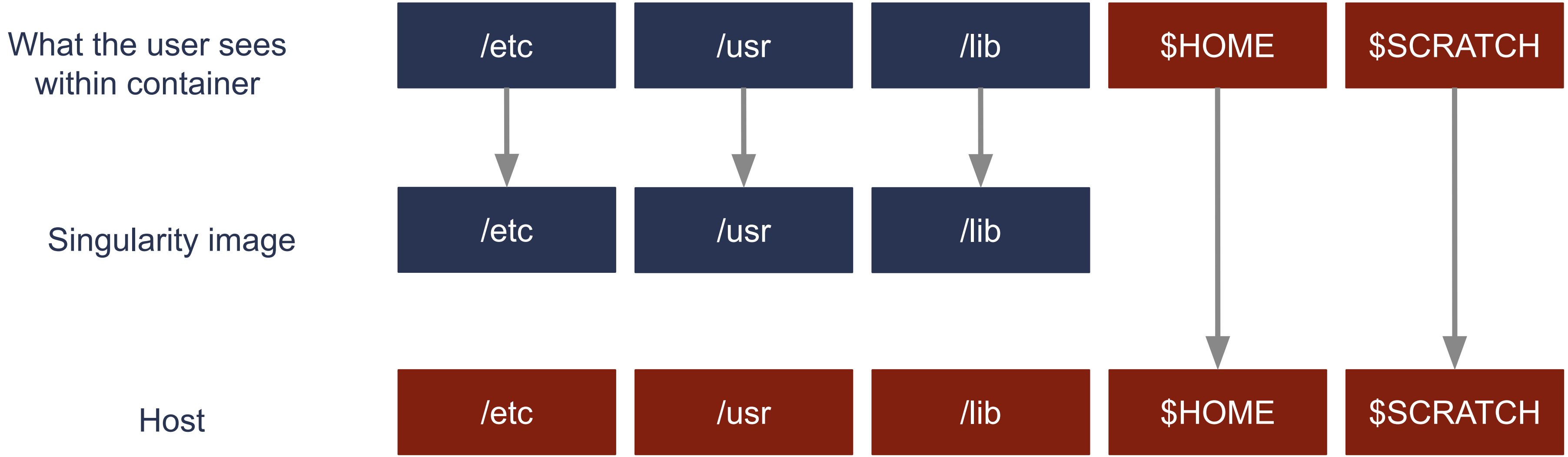
Local vs Remote build

Remote builds are very convenient but you might need to adapt your definition file

```
$ cat example-openmp/openmp.def
Bootstrap: docker
From: ubuntu:18.04

%setup
   mkdir ${SINGULARITY_ROOTFS}/opt/bin

%files
   omp_pi.c /opt/bin
   omp_dot2.c /opt/bin

%environment
   export PATH="/opt/bin:$PATH"

%post
   echo "Installing required packages..."
   apt-get update && apt-get install -y bash gcc gfortran

   echo "Compiling the application..."
   cd /opt/bin && gcc -fopenmp -o omp_pi.x omp_pi.c &&  gcc -fopenmp -o
omp_dot2.x omp_dot2.c
```

```
$ cat remote-build/openmp-remote.def
Bootstrap: docker
From: ubuntu:18.04

%setup
   mkdir ${SINGULARITY_ROOTFS}/opt/bin

%environment
   export PATH="/opt/bin:$PATH"

%post
   echo "Installing required packages..."
   apt-get update && apt-get install -y bash gcc gfortran curl

   echo "Compiling the application..."
   cd /opt/bin && curl -O
https://raw.githubusercontent.com/fasrc/User_Codes/master/Courses/CS20
5/OpenMP/Example6/omp_pi.c && gcc -fopenmp -o omp_pi.x omp_pi.c
   cd /opt/bin && curl -O
https://raw.githubusercontent.com/fasrc/User_Codes/master/Courses/CS20
5/OpenMP/Example5/omp_dot2.c && gcc -fopenmp -o omp_dot2.x omp_dot2.c
```

Singularity and host file system



Adapted from LSU Singularity training slides:
http://www.hpc.lsu.edu/training/weekly-materials/2022-Spring/HPC_Singularity_Spring2022.pdf

Storage: Bind Mount

- By default all directories in the Singularity image are read only.
  - **Note:** When building from Docker, sometimes Docker expects something to be writable that may not be in Singularity.
- In addition system directories are not available, only those defined in the Singularity image.
- You can bind external mounts into singularity using the `-B` option
  - `-B hostdir:containerdir`
  - `-B hostdir` <- Maps it to same path inside the container
- On Cannon, we automatically map `/n`, `/net`, `/scratch` and `/cvmfs` into the image using bind mount.

Bind Mount: some interesting use cases

1. Expose extra tools from the host inside the container (for example slurm)

```
$> export SING_BINDS=" -B /etc/nsswitch.conf -B /etc/sssd/ -B /var/lib/sss -B /etc/slurm -B /slurm -B
/var/run/munge  -B `which sbatch ` -B `which srun ` -B `which sacct ` -B `which scontrol `   -B
/usr/lib64/slurm/ "

$> export SING_BINDS="$SING_BINDS  -B ${OMNIROOT}/omnisci-storage:/omnisci-storage  -B
${OMNIROOT}/Datasets:/omnisci/sample_datasets "
```

$> singularity run $SING_GPU **$SING_BINDS** --pwd /omnisci $container_image

2. Overlay folders to hide the content to applications in case there is no option to specify an alternate location. e.g.

```
$> export SING_BINDS=" $SING_BINDS -B $MYTMP:/tmp -B $RSTUDIO_CONFDIR:$HOME/.rstudio "
```

3. Sometimes you need to only allow a few specific things to be mapped in the container

```
$> export SING_BIND="  --contain -B /tmp -B /dev -B /scratch -B `dirname $HOME` -B
/n/helmod/apps -B /n/helmod/modulefiles -B /n/sw/eb -B /n/sw/intel-cluster-studio-2017 "
```

Additional use cases (i) : Legacy software

When we moved from Centos6 to Centos7 there were a few legacy software that would not work in Centos 7. We can still run those in a centos6 container.

We need to provide support to GUI legacy software that needs to run in centos6-ish environment.

```
cat > run-abaqus <<EOF
#!/bin/bash
(
  export SEND_256_COLORS_TO_REMOTE=1
  export XDG_CONFIG_HOME="<%= session.staged_root.join("config") %>"
  export XDG_DATA_HOME="<%= session.staged_root.join("share") %>"
  export XDG_CACHE_HOME="\$(mktemp -d)"
  module restore
  set -x
  xfwm4 --compositor=off --daemon --sm-client-disable
  xsetroot -solid "#D3D3D3"
  xfsettingsd --sm-client-disable
  xfce4-panel --sm-client-disable
) &

export PATH=/n/sw/abaqus-6.12/Commands:$PATH
abaqus cae -mesa
EOF

chmod +x myrun.sh
dbus-uuidgen > mymachid
image=xfce-centos6/xfce-el6fasrc.img
singularity exec -B ./mymachid:/var/lib/dbus/machine-id $image ./run-abaqus
```

Xfce centos6 Launched from the container

Abaqus is loaded from our software repo

Singularity container running xfce in centos6

Additional use cases: Storage performance

Matlab 2018b Desktop has a large startup time
(~ 3 – 5 minutes) when loaded from our module system.
This can be considered ok for long batch jobs, but not acceptable for interactive jobs.

A Matlab 2018b container stored on our Lustre filesystem can be launched in ~ 10 seconds.

HARVARD UNIVERSITY

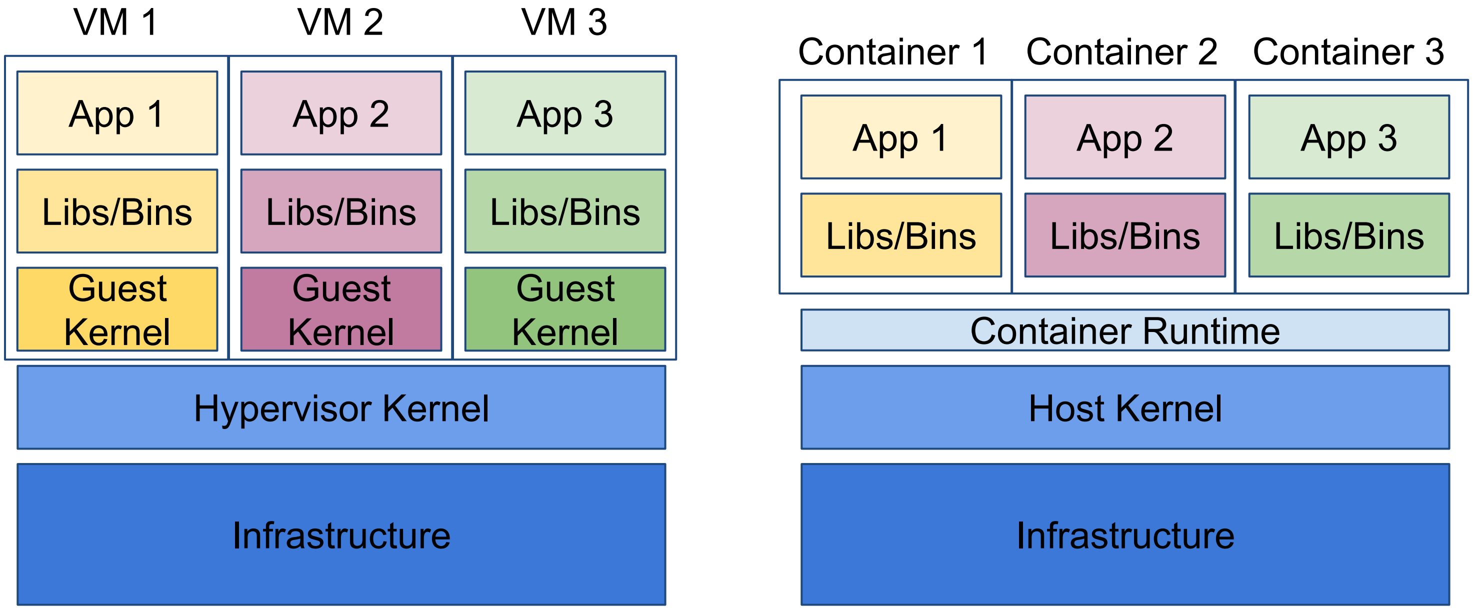
FASRC

Overlay:
useful for dealing with large number of files

from https://informatics.fas.harvard.edu/cactus-on-the-fasrc-cluster.html (courtesy of Nathan Weeks)

```
readonly JOBSTORE_IMAGE=jobStore.img
readonly CACTUS_SCRATCH=/scratch/cactus-${SLURM_JOB_ID}

 mkdir -m 777 ${CACTUS_SCRATCH}/upper ${CACTUS_SCRATCH}/work
 truncate -s 2T "${JOBSTORE_IMAGE}"
  singularity exec ${CACTUS_IMAGE} mkfs.ext3 -d ${CACTUS_SCRATCH} "${JOBSTORE_IMAGE}"

# Use empty /tmp directory in the container (to avoid, e.g., pip-installed packages in ~/.local)
mkdir -m 700 -p ${CACTUS_SCRATCH}/tmp

# the toil workDir must be on the same file system as the cactus jobStore
singularity exec --overlay ${JOBSTORE_IMAGE} ${CACTUS_IMAGE} mkdir -p /cactus/workDir
srun -n 1 singularity exec --cleanenv \
            --no-home \
            --overlay ${JOBSTORE_IMAGE} \
            --bind ${CACTUS_SCRATCH}/tmp:/tmp \
            ${CACTUS_IMAGE} \
  cactus ${CACTUS_OPTIONS-} ${restart-} --workDir=/cactus/workDir --binariesMode local /cactus/jobStore "${SEQFILE}" "${OUTPUTHAL}"
```

VMs or Containers



VMs:
hardware virtualization + OS

Containers:
User defined software stack