Tutorial: Fundamentals of Containers

Singularity and **Charliecloud** on **ACES**

Richard Lawrence

3/19/2024

ACES
ACCELERATING COMPUTING FOR EMERGING SCIENCES

High Performance Research Computing
DIVISION OF RESEARCH

CC* CYBERTEAM SWEETER RESEARCH | ENABLEMENT | TRAINING

developed for

ACCESS Support

Outline

- Overview of Containers
- Overview of Charliecloud
- Overview of Singularity
- Getting Started
- Working with Images
- Container Recipes

Learning Resources

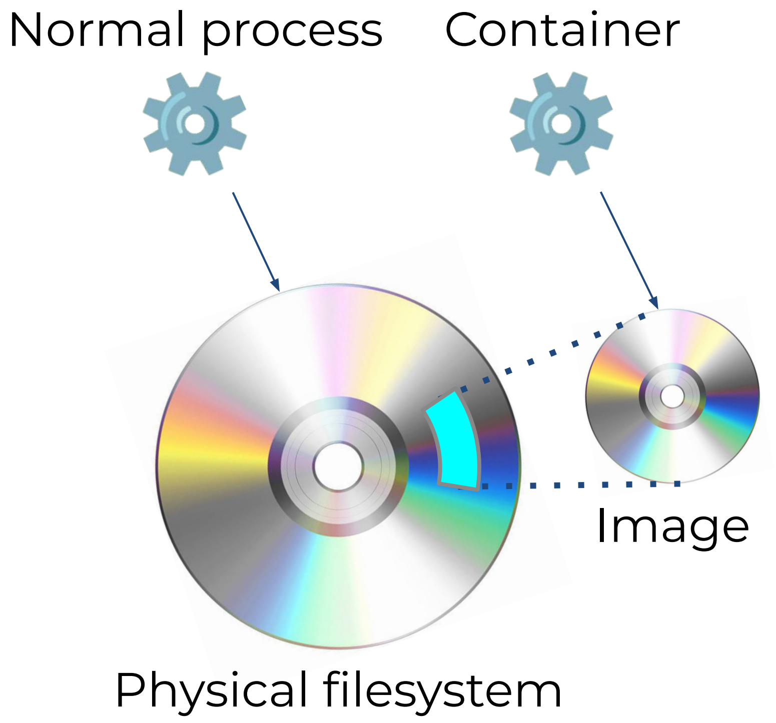
- Slides on the course web page
  https://hprc.tamu.edu/training/aces_container_fundamentals.html
  (highly recommended for working along)
- HPRC's Knowledge Base
  https://hprc.tamu.edu/kb/Software/CharlieCloud/
  https://hprc.tamu.edu/kb/Software/Singularity/
- HPRC on YouTube
  https://www.youtube.com/c/TexasAMHPRC
- ACCESS Links
  https://support.access-ci.org/ci-links

Overview of Containers

What Are Containers?

- A container is a process (⚙️) that has its own **view** of local resources:
    - Filesystem
    - User IDs
    - Network

      etc.
- Example: this container (⚙️on the right) sees the **image** instead of the physical filesystem

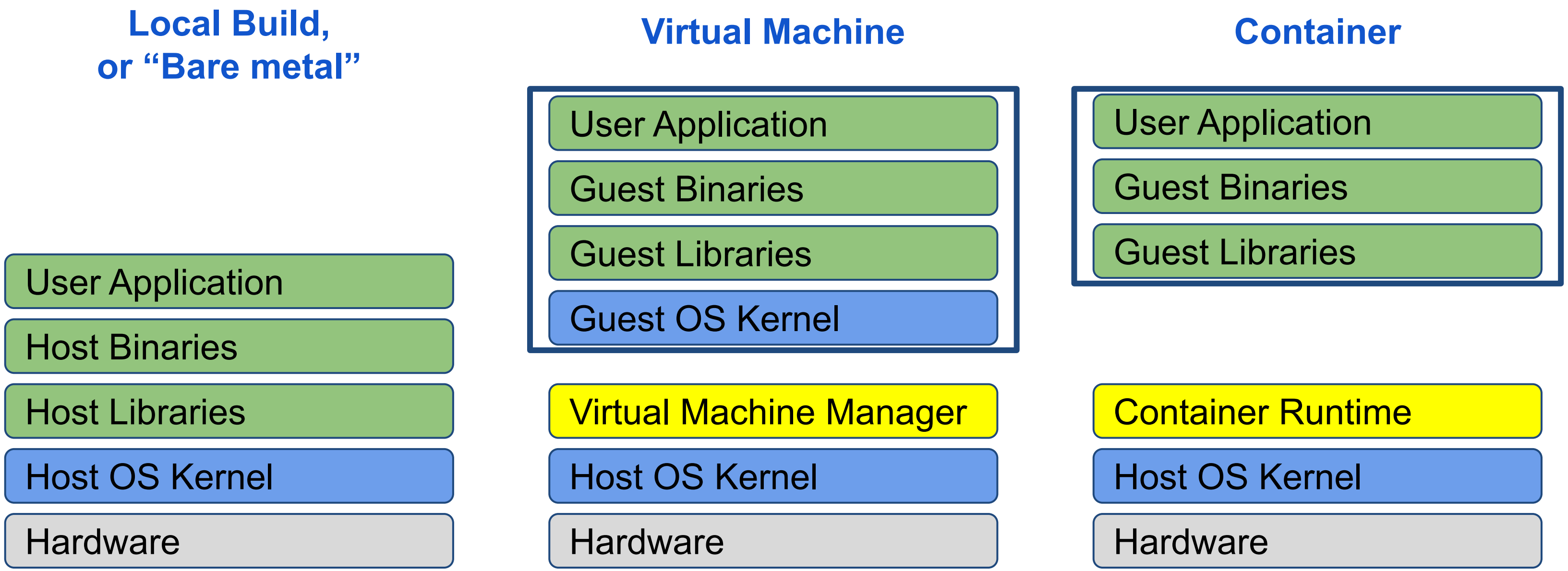


Why Use Containers?

- **Shareability**:
  - Share your container image file by uploading to a public repository
  - Use images shared by others
- **Portability**:
  - Use images on any computer with the same architecture (x84-64)
- **Reproducibility**:
  - Container users are largely unaffected by changes to the cluster environments

What Goes In Container Images?

- Unlike in VMs, the OS Kernel is not duplicated
- Container images are smaller than VM images



Popular Container Runtimes

Instant deployment to users on different devices!

LXC	Docker	Singularity	Shifter 2016	Charliecloud	Podman
2008	2013	2015		2017	2018

Overview of Charliecloud

Charliecloud

- A lightweight, fully-unprivileged container solution

Charliecloud

SMALL SOFTWARE DELIVERING BIG FLEXIBILITY TO HPC

Presented by

Los Alamos NATIONAL LABORATORY

Charliecloud Features

- Charliecloud is a container runtime and an image builder
- Charliecloud can read and convert Docker images
- Filesystem inside container is isolated
- User inside container is isolated
- Works with high-performance cluster technologies

Read more in the Charliecloud manual on github
https://hpc.github.io/charliecloud/

Charliecloud on ACES

- Charliecloud is available from our module system
  - `module load charliecloud`
- Charliecloud images can be large on disk. Be aware of your storage quota. (`/scratch` > `/home`)
- Some container activities may be too cpu-intense for the *shared* login node. Be courteous to others and use a compute node for large image operations.
- Some container activities may be too I/O-intense for the *shared* network filesystem. Be courteous to others and use a local filesystem for large image operations.

Overview of Singularity

Singularity

- An easy-to-use, high-performance container solution

Deploying Secure Container

Solutions from Edge to Exascale

Presented by

Sylabs

Singularity is Apptainer

Sylabs

THE LINUX FOUNDATION

Singularity Features

- Singularity is a container runtime and an image builder
- Singularity can read and convert Docker images
- Filesystem inside container is isolated
- User inside container is the same as the user outside
- Works with high-performance cluster technologies

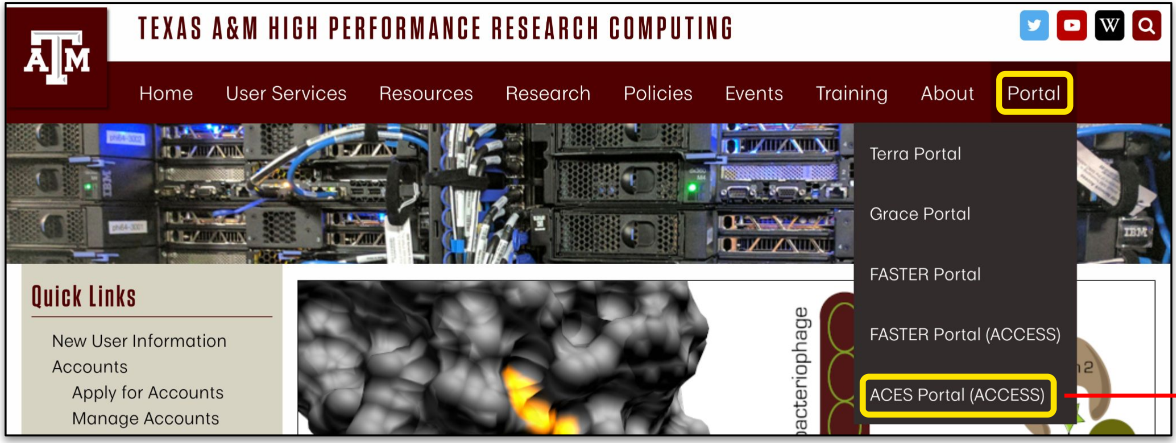
Read more in the Apptainer manual
https://apptainer.org/user-docs/3.8/

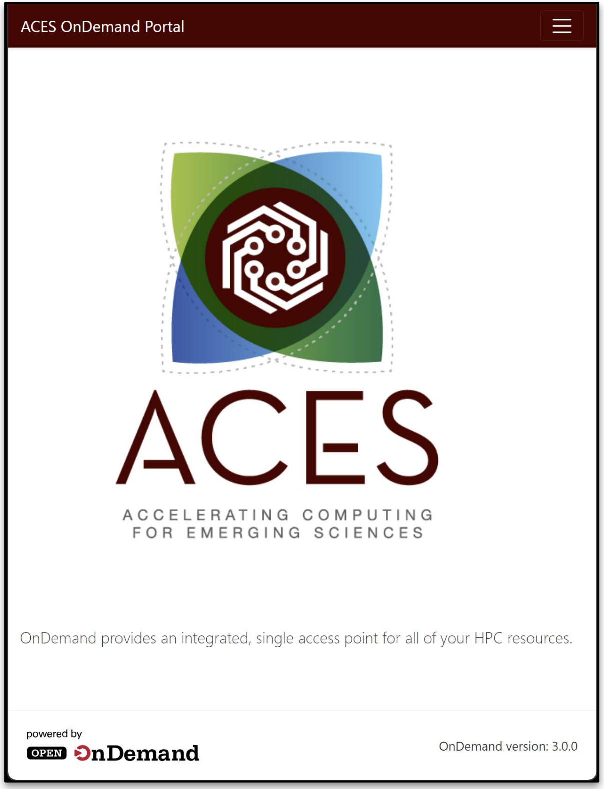
Singularity on ACES

- Singularity is available on Compute nodes
  - Singularity activities are too cpu-intensive for login nodes.
- Singularity images can be large on disk. Be aware of your storage quota. (`/scratch` > `/home`)
- Some container activities may be too I/O-intense for the shared network filesystem. Be courteous to others and use a local filesystem for large image operations.

Getting Started

ACES Portal



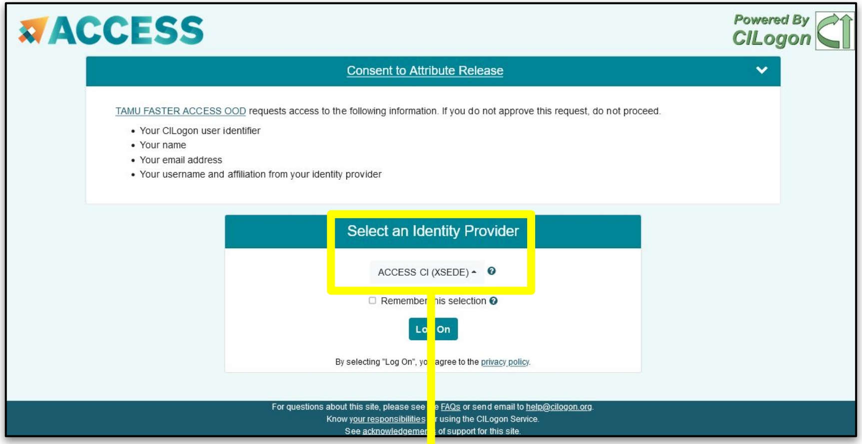


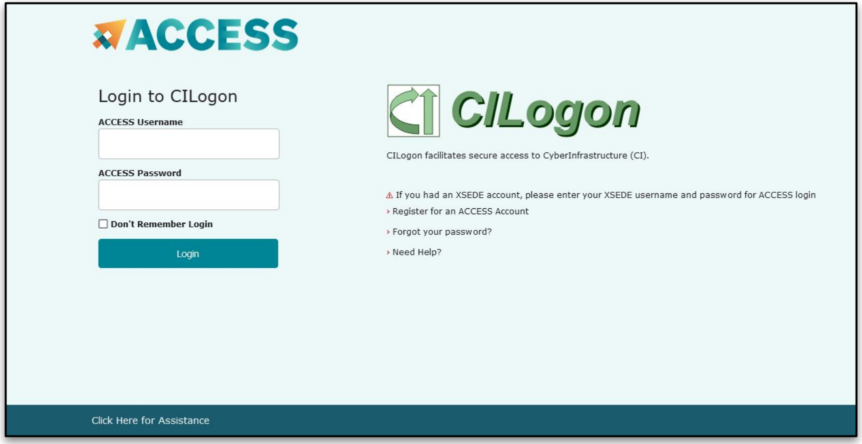
ACES Portal portal-aces.hprc.tamu.edu
is the web-based user interface for the ACES cluster

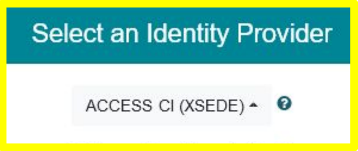
Open OnDemand (OOD) is an advanced web-based graphical interface framework for HPC users

Authentication via CILogin

Log-in using your ACCESS CI credentials.



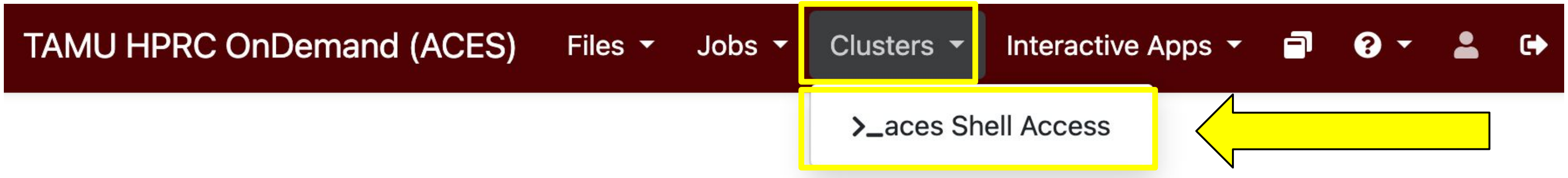




Select the Identity Provider appropriate for your account.

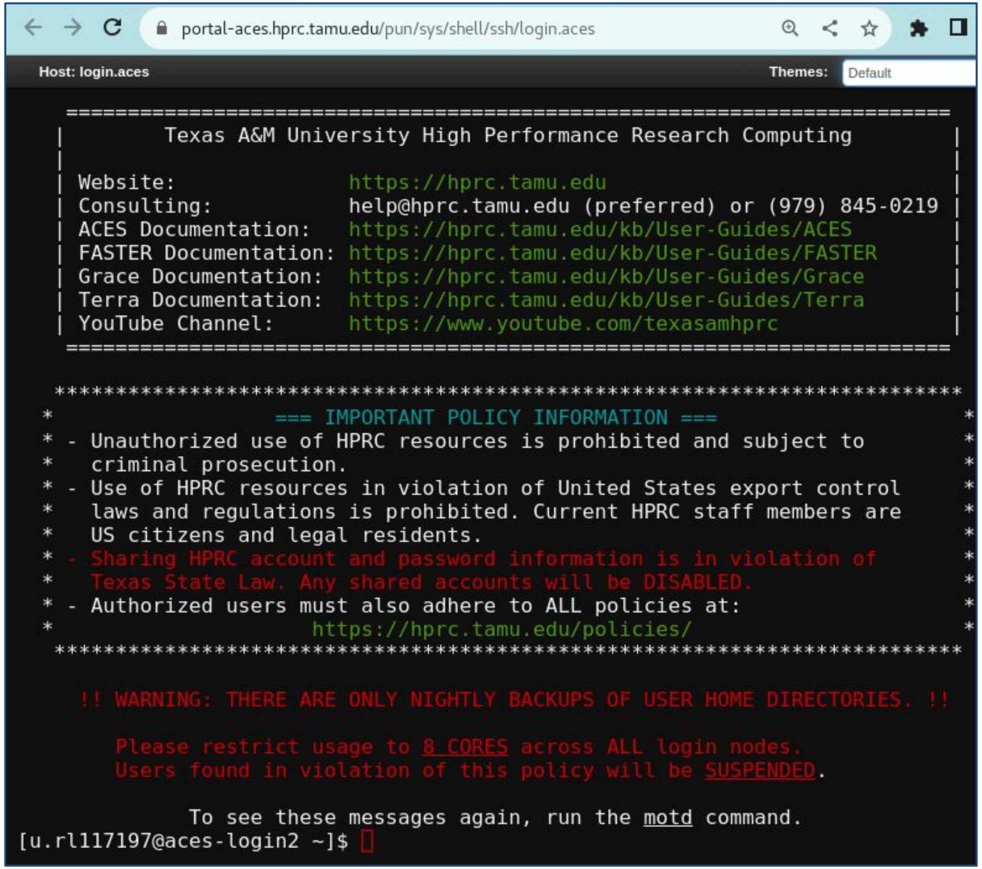
Get a Shell on ACES

Click on “Clusters” menu → _aces Shell Access



Success!

Welcome to the ACES login node.



Set Up Your Charliecloud Environment

On the login node:

```
module load charliecloud
module list
```

Note. *light green highlight* means please perform this exercise in your ACES shell terminal.

Your First Charliecloud Image

The charliecloud image tool helps you build and organize your images.

```
ch-image --help
```

Let's fetch a small, basic linux distro: Almalinux.

```
ch-image pull almalinux:8
ch-image list
```

The image is in your personal temporary local image repository.

```
echo $CH_IMAGE_STORAGE
ls $CH_IMAGE_STORAGE/img/
```

Your First Charliecloud Container

The ACES login node has Red Hat Enterprise linux installed.

```
cat /etc/redhat-release
```

The charliecloud-run tool launches containers out of existing images.

```
ch-run --help
```

Launch a bash shell, investigate linux, and stop the container.

```
ch-run almalinux:8 bash
cat /etc/redhat-release
exit
```

Set Up Your Singularity Environment

Get to a compute node (from the login node)

```
srun --time=90 --mem=4G --pty bash -i
```

The compute node should be similar to the login node

```
cat /etc/redhat-release
```

Set your singularity cache directory

```
export SINGULARITY_CACHEDIR=$TMPDIR
```

Connect to the internet

```
module load WebProxy
```

Your First Singularity Container

Singularity can fetch an image *and* launch a shell in one line.

```
singularity shell --help
```

Fetch that same image (again) and launch a shell from it (again)

```
singularity shell docker://almalinux:8
cat /etc/redhat-release
exit
```

Don't forget to return to the login node

```
exit
```


Congratulations!
Welcome to containers
WWW.FUNIMADA.COM

Working with Images

Charliecloud Image Formats

- Charliecloud container images come in two main formats:

1. Directory
2. Single file. HPRC supports the squashfs filesystem format for single file images. (more about that on a later slide)

- The `ch-convert` tool copies images into different formats

```
ch-convert --help
```

Directory Image Format

- The image name should end in `/`.
- Directory images are writable.
- Directory read/write operation are slow, so put directory images on the high-speed `/tmp` filesystem.
- Images in `$CH_IMAGE_STORAGE` are also directory images, but you refer to them by name without the trailing slash.

Convert to Directory Exercise

Create a space on the login node for yourself

```
mkdir /tmp/$USER
```

Convert our image in the cache to a directory image.
(note the order of the arguments)

```
ch-convert almalinux:8 /tmp/$USER/almalinux/
```

What did we make?

```
ls /tmp/$USER/almalinux/
```

Editing Images Exercise

Directory images can be modified by adding the

`--write`

flag to `ch-run`. Any changes you make will be saved.

```
ch-run --write /tmp/$USER/almalinux/ bash
mkdir /my_dir
exit
```

Are the changes still there?

```
ch-run /tmp/$USER/almalinux/ bash
ls /
exit
```

Squashfs Image Format

- Squashfs is an open-source file format for filesystem images
- The whole filesystem becomes one single file
- The image name should end in `.sqfs`
- Squashfs images are read-only.
- Squashfs read operations are fast, so put squashfs images on the network filesystem `/scratch`.

Set Up Your Environment

Create a workspace in your **scratch** directory.

```
cd $SCRATCH
mkdir c_tutorial
cd c_tutorial
pwd
```

Convert to Squashfs Exercise

Make sure you are still in your `c_tutorial` directory in `$SCRATCH`

```
pwd
```

Then convert *(note the order of the arguments)*

```
ch-convert /tmp/$USER/almalinux/ almalinux.sqfs
```

Are your changes still there?

```
ch-run almalinux.sqfs /bin/bash
ls
exit
```

Singularity Image Formats

- Singularity container images come in two main formats:
  1. Directory
  2. Single file. Singularity uses the `SIF` format for single file images. This is the default.
- The `singularity build` tool can convert images in both formats.

```
singularity build --help
```

- The `--sandbox` option is used to create directory-format images.

Set Up Your Singularity Environment

Get to a compute node (from the login node)

```
srun --time=90 --mem=4G --pty bash -i
```

Set your singularity cache directory

```
export SINGULARITY_CACHEDIR=$TMPDIR
```

Connect to the internet

```
module load WebProxy
```

Return to your scratch area

```
cd $SCRATCH/c_tutorial
```

Singularity Image Exercise

Singularity pull can fetch an image and write to either file format.
(note the order of the arguments)

```
singularity pull almalinux.sif docker://almalinux:8
```

Singularity can convert an image to the directory file format.
Use the --sandbox argument to specify the directory type.
(note the order of the arguments)

```
singularity build --sandbox $TMPDIR/almalinux almalinux.sif
```

Singularity Write Exercise

Directory images are writable. Simply add the `--writeable` flag to your container command.

```
singularity shell --writable $TMPDIR/almalinux
mkdir /my_dir
exit
```

Are the changes still there?

```
singularity shell $TMPDIR/almalinux
ls /
```

Are Directory Images All Compatible?

Let's try an experiment. Still on your compute node,

```
module load charliecloud
ch-run $TMPDIR/almalinux bash
ls /
```

Directory images are universal.
They can come from a variety of sources, for example:
https://github.com/alpinelinux/docker-alpine/
this github repo contains a Linux package in a directory format that's designed for containers.

Container from Tarball Exercise (1/3)

Navigate to your tmp space

```
cd $TMPDIR
```

Fetch the distro's tarball:

```
wget -O alpine.tar.xz
'https://github.com/alpinelinux/docker-alpine/blob/v3.18/x86_64/al
pine-minirootfs-3.18.6-x86_64.tar.gz?raw=true'
```

or

```
cp /scratch/training/containers/alpine.tar.xz .
```

Container from Tarball Exercise (2/3)

Oh no! the tarball is a tar *bomb* (too many files)

```
tar tf alpine.tar.xz | head -10
```

Unpack it in a new subdirectory.

```
mkdir alpine
cd alpine
tar xf ../alpine.tar.xz
ls
cd ..
```

Container from Tarball Exercise (3/3)

Now we can run a shell in the container!

```
ch-run ./alpine -- /bin/sh
cat /etc/alpine-release
exit
```

```
singularity shell ./alpine
cat /etc/alpine-release
exit
```

(alpine doesn't have bash)
(singularity knew what to do, charliecloud needed a hint)

Container Recipes

Container Recipes

- Modifying containers by hand is bad in practice. The information about what steps were taken is lost.
- Better to write down those steps in a recipe file.
- Docker uses a recipe file named Dockerfile.
- Chariecloud supports Dockerfiles.
- Singularity uses a different recipe file called a Definition file.

Target Recipe

1. Start with the almalinux:8 image
2. Install Python 3.

   `yum -y install python39` *(do not attempt)*

3. Add a “hello.py” python script (and make it executable).

```
#!/usr/bin/python3
print("Hello World!")
```

```
chmod 755 hello.py
```

And we shall name the image “hello”.

Elements of a Dockerfile

```
FROM almalinux:8
RUN yum -y install python39
COPY ./hello.py /
RUN chmod 755 /hello.py
```

1. `FROM`: We are extending the almalinux:8 base image.
2. `RUN`: Install the python39 RPM package
3. `COPY`: Copy the file hello.py from outside the image into the root directory of the image.
4. `RUN`: Make that file executable.

Charliecloud Recipe

Take a moment to set up this workspace in `$SCRATCH`:

```
c_tutorial/
    hello.src.docker/
        hello.py
```

```
#!/usr/bin/python3
print("Hello World!")
```

```
        Dockerfile
```

```
FROM almalinux:8
RUN yum -y install python36
COPY ./hello.py /
RUN chmod 755 /hello.py
```

Build from Dockerfile

Make sure you have charliecloud loaded, then build the image:

```
cd $SCRATCH/c_tutorial #if necessary
cd hello.src.docker
ch-image build -t hello -f Dockerfile .
ch-image list
```

ch-image build arguments:

- (`-t hello`) Build an image named (a.k.a. tagged) “hello”.
- (`-f Dockerfile`) Use the Recipe named “Dockerfile”.
- (`.`) Use the current directory as the context directory .

Testing the Recipe Image

Convert from image cache to flat image

```
ch-convert hello hello.sqfs
```

run the container and launch hello from python

```
ch-run hello.sqfs -- /hello.py
```

run the container and check for python

```
ch-run hello.sqfs -- python3 --version
```

Does python3 exist on the local node?

```
python3 --version
```

Elements of a Definition File

```
Bootstrap: docker
From: almalinux:8
%files
    hello.py
%post
    yum -y install python39
    chmod 755 /hello.py
```

1. `BOOTSTRAP`: Base image comes from Docker Hub.
2. `FROM:` Base image is almalinux:8
3. `%files`:
   Copy the file hello.py from outside the image into the root directory of the image.
4. `%post`:
   a. Install the python39 RPM
   b. Make `/hello.py` executable.

Singularity Recipe

Take a moment to set up this workspace in `$SCRATCH`:

```
c_tutorial/
    hello.src.def/
        hello.py

        Definition
```

```
#!/usr/bin/python3
print("Hello World!")
```

```
Bootstrap: docker
From: almalinux:8
%files
    hello.py
%post
    yum -y install python36
    chmod 755 /hello.py
```

Build from Definition file

Make sure you are on the compute node, then build the image:

```
cd $SCRATCH/c_tutorial #if necessary
cd hello.src.def
singularity build --fakeroot hello.sif Definition
```

singularity build arguments:

- (`--fakeroot`) Needed for `yum` install permission.
- (`Definition`) Use the Recipe named "`Definition`".
- (`hello.sif`) save the result in an SIF image.

Testing the Recipe Image

Singularity exec is used for running non-shell containers.

```
singularity exec --help
```

run the container and launch hello from python

```
singularity exec hello.sif /hello.py
```

run the container and check for python

```
singularity exec hello.sif python3 --version
```

Does python3 exist on the local node?

```
python3 --version
```

Acknowledgements

This work was supported by

- the National Science Foundation (NSF), award numbers:
  - 2112356 - ACES - Accelerating Computing for Emerging Sciences
  - 1925764 - SWEETER - SouthWest Expertise in Expanding, Training, Education and Research
  - 2019129 - FASTER - Fostering Accelerated Scientific Transformations, Education, and Research
- Staff and students at Texas A&M High-Performance Research Computing.
- ACCESS CCEP pilot program, Tier-II

High Performance Research Computing
DIVISION OF RESEARCH

https://hprc.tamu.edu

HPRC Helpdesk:

help@hprc.tamu.edu
Phone: 979-845-0219

Help us help you. Please include details in your request for support, such as, Cluster (ACES, Faster, Grace, ViDaL), NetID (UserID), Job information (Job id(s), Location of your jobfile, input/output files, Application, Module(s) loaded, Error messages, etc), and Steps you have taken, so we can reproduce the problem.