Building Containers for HPC

Charles Peterson

Overview

Welcome!

In this workshop, we will go over using containers on HPC resources, like UCLA's Hoffman2. This is a follow-up to a previous workshop "Using Containers on HPC Resources"

- We will go over more advance container topics
- We will build container for use on HPC resources

Any suggestions for upcoming workshops, email me

- cpeterson@oarc.ucla.edu

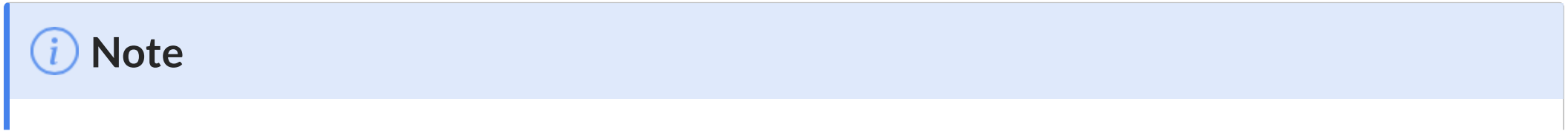
Files for this Presentation

This presentation can be found on our UCLA OARC's github repo under **MakingContainers_07062022** folder

https://github.com/ucla-oarc-hpc/hpc_workshops

The `slides` folder has this slides.

- PDF format: `MakingContainers.pdf`
- html format: `html` directory
  - You can open the `MakingContainers.html` file in your web browser

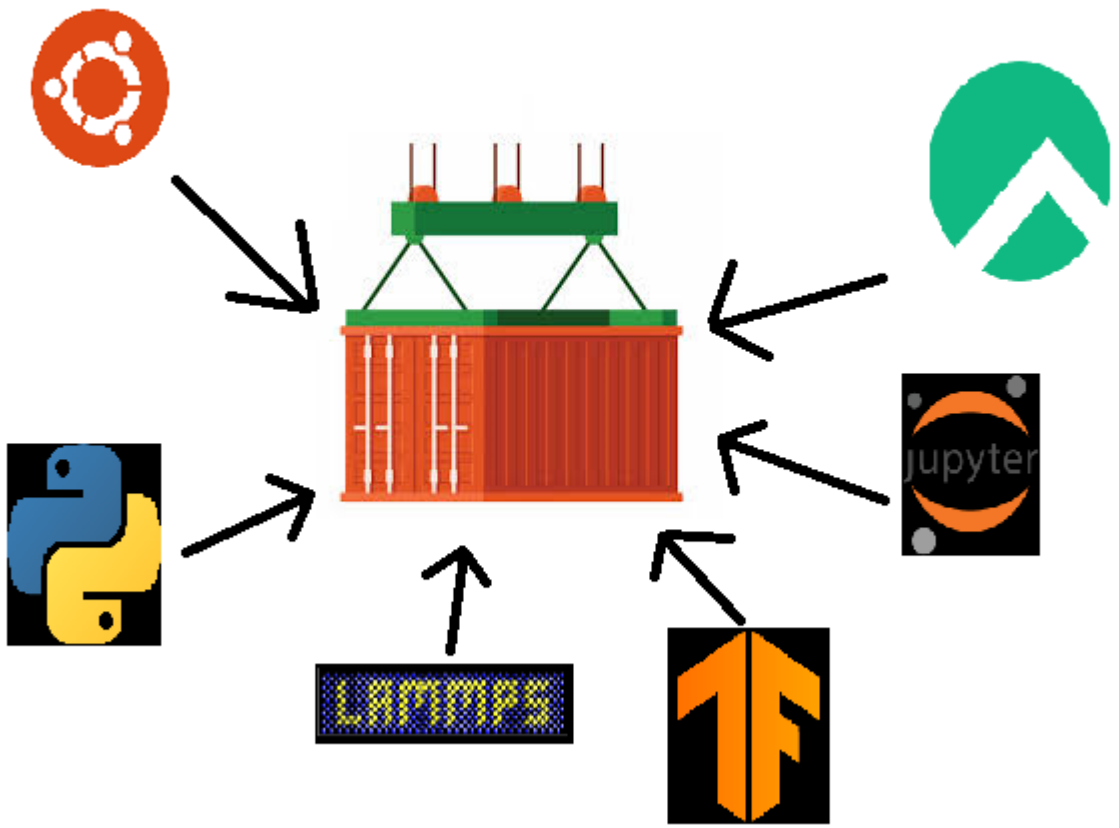


This presentation was build with Quarto and RStudio.

- Quarto file: `MakingContainers.qmd`

Container Review

Containers



Software for Containers

Apptainer

- Formerly Singularity
- Designed and developed for HPC systems
- Mostly likely installed on most HPC systems
- Supports Infiniband, GPUs, MPI, and other devices on the Host
- Can run Docker containers

Software for Containers

Docker

- Very popular
- Many popular cloud container registries
  - DockerHub, GitHub, Nvidia NGC
- MPI not well supported
- Most likely NOT available on many HPC systems

Software for Containers

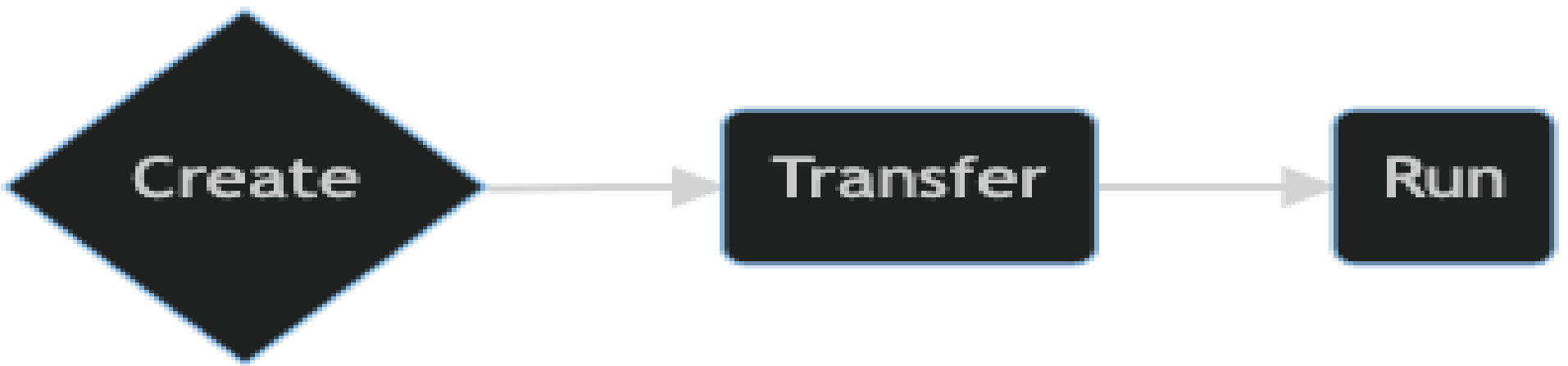
Podman

- Similar syntax as with Docker
  - Can use to ‘replace’ Docker
- Doesn’t have a root daemon process
- On some HPC resources (not on Hoffman2, yet)

Apptainer workflow

Apptainer workflow (Create)

- Build a container by installing Appainer on your computer (where you have root/sudo access) to create a container
- Use a pre-built container
  - Search Container Registries for container
  - DockerHub, GitHub packages, Nvidia NGC
- A SIF file contains the image for the container
- A sandbox container is a directory format used for writable containers



Apptainer workflow (Transfer)

Bring your container to Hoffman2

- Copy your container to Hoffman2

```
scp test.sif H2USERNAME@hoffman2.idre.ucla.edu
```

- Pull a container from online Container Register

```
apptainer pull docker://ubuntu:20.04
```

- Use a container pre-built on Hoffman2

```
#Pre-built container location on Hoffman2
ls $H2_CONTAINER_LOC
```


Create
Transfer
Run

Apptainer workflow (Run)

interactive (qrsh) session

```
qrsh -l h_data=5G
module load apptainer/1.0.0
apptainer exec mypython.sif python3 test.py
```

Batch (qsub) job

```
cat << EOF >> myjob.job
module load apptainer/1.0.0
apptainer exec mypython.sif python3 test.py
EOF
qsub -l h_data=5G myjob.job
```

Common Usage

On Hoffman2, to use apptainer, all you need to do is load the module

```
1 module load apptainer/1.0.0
```

- Only module you need to load!
  - No need to load tons of modules to run a single application
  - Expect MPI module if running parallel
    - `module load intel/2022.1.1`

Common Apptainer commands:

- Getting a container from somewhere

```
apptainer pull [options]
apptainer pull docker://ubuntu:22.04
```

- Build a container

```
apptainer build [options]
apptainer build myapp.sif myapp.def
```

- Run a single command inside of a container

```
apptainer exec [options]
apptainer exec myapp.sif MYCOMMAND
```

- Run the container with a predefined script

```
apptainer run [options]
apptainer run myapp.sif "SCRIPT ARGRMENTS"
```

Example 1: Basic TensorFlow

Running simple container with Apptainer

Tensorflow Container

This example will use Tensorflow

A great library for develop Machine Learning models

- Go to EX1 directory
- Look at `tf-example.py`
  - Simple example to train MNIST dataset

To run this job, we will run

```
1 python3 tf-example.py
```

Need tensorflow!!!

- Instead of installing it yourself, let is find a container

Visit DockerHub

TensorFlow (interactive)

Running on Hoffman2

- Start an interactive session

```
qrsh -l h_data=10G
```

- load the apptainer module

```
module load apptainer/1.0.0
```

- pull the TensorFlow container from DockerHub

```
apptainer pull docker://tensorflow/tensorflow:2.7.1
```

- We see a SIF file named, tensorflow_2.7.1.sif
- Start an interactive shell **INSIDE** the container

```
apptainer shell tensorflow_2.7.1.sif
python3 tf-example.py
```

TensorFlow (batch)

Run a command inside the container

```
qrsh -l h_data=10G
module load apptainer/1.0.0
apptainer pull docker://tensorflow/tensorflow:2.7.1
apptainer exec tensorflow_2.7.1.sif python3 tf-example.py
```

Alternatively, you can submit this as a batch job

- Example job script: `tf-example.job`

```
qsub tf-example.job
```

NOTE:

- See that we didn't need to load any python module!
- We didn't need to install any TF packages ourselves!!

Example 2: Create writable containers

Pytorch

This example uses PyTorch which is a great Machine Learning framework

This example will create a container by installing software inside of a container interactively

For this example, you will need Apptainer installed on a machine that you have admin/sudo access.

You can find information on installing Apptainer under their Admin Guide

You can also use the Container.ova image that can be opened on VirtualBox

Creating the container

We will first need to start building the container from an existing container on our local machine

```
sudo apptainer build --sandbox ubuntu_22.04_SB/ docker://ubuntu:22.04
```

This will create a **sandbox** image of the base ubuntu 22.04 image. The sandbox image is a writable directory that we can modify and install software.

Next, we will start an **WRIATABLE** interactive shell session in the sandbox image

```
sudo apptainer shell --writable ubuntu_22.04_SB/
```

From here, we can run any commands we need to install PyTorch.

```
apt update
apt install -y python3 python3-pip
pip3 install torch torchvision torchaudio --extra-index-url https://downloa
```

Convert sandbox container to SIF file

```
sudo apptainer build pytorch.sif ubuntu_22.04_SB/
```

Running our container

Run our SIF container

```
apptainer exec pytorch.sif python3 pytorch.py
```

We do not need to be root since we are just running python3.

Transferring our container

```
scp pytorch.sif hoffman2.idre.ucla.edu:
```

Example job script to run on Hoffman2

```
qsub pytorch.job
```

Example 3: Building from definition files

Look at EX3

Creating containers

I coded a chemistry app located on github.

- This was a very early attempt for me to learn Hartree-Fock.
- https://github.com/charliecpeterson/QUILL

We need:

- Python with the PySCF package
- Eigen3, a Linear Algebra library

Instead of installing these dependencies on H2 (or looking for modules), lets build a container!!

We will build this container by:

- Using a definition file (.def)
- Using Docker/Podman (Dockerfile)

Apptainer Definition file

The `quill.def` file has all steps needed to build the QUILL container.

```
Bootstrap: docker
From: ubuntu:20.04

%labels
Author Charles Peterson <cpeterson@oarc.ucla.edu>

%post
apt-get update
DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
   git python3 python3-dev python3-pip \
   libeigen3-dev ca-certificates cmake make gcc g++
rm -rf /var/lib/apt/lists/*

pip3 install pyscf
ln -s /usr/bin/python3 /usr/bin/python
mkdir -pv /apps
cd /apps
git clone https://github.com/charliecpeterson/QUILL
cd QUILL
mkdir build ; cd build
cmake ..
make

%environment
export PATH=/apps/QUILL/build:$PATH
```

Sections:

- Bootstrap/From - Location of starting container
- `%labels` - adds metadata
- `%post` - This section runs commands to setup the final container
- `%environment` - define environment variables inside container

Create container

The `quill.sif` container is created

```
1 sudo apptainer build quill.sif quill.def
```

Let us test the container

- We want to run the command `QUILL.x test.inp` inside the container

```
1 apptainer exec quill.sif  QUILL.x test.inp
```

Move container to Hoffman2

```
1 scp quill.sif H2USERNAME@hoffman2.idre.ucla.edu:
```

Building using Docker/Podman

You can also use Docker or Podman to create containers for apptainer.

I will be using Podman. BUT if you want to use Docker instead, just replace the command `podman` with `docker`. The syntax is exactly the same!!

```
1 docker build
2 docker images
3 docker pull
4 docker run
```

```
1 podman build
2 podman images
3 podman pull
4 podman run
```

Dockerfile

The `Dockerfile-quill` file is used by Docker to create the container

```
FROM ubuntu:20.04

## Author Charles Peterson <cpeterson@arc.ucla.edu>

RUN apt-get update \
      && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-reco
    git python3 python3-dev python3-pip \
    libeigen3-dev ca-certificates cmake make gcc g++ \
    && rm -rf /var/lib/apt/lists/*

RUN pip3 install pyscf ; ln -s /usr/bin/python3 /usr/bin/python

RUN mkdir -pv /apps \
     && cd /apps \
     && git clone https://github.com/charliecpeterson/QUILL \
     && cd QUILL \
     && mkdir build ; cd build \
     && cmake .. ; make
```

Create container

Create container

```
podman build . -t quill:1.0 -f Dockerfile-quill
```

See built docker (podman) container

```
podman image list
```

Save docker (podman) image to apptainer container

```
podman save quill:1.0 > quill.tar
apptainer build quill.sif docker-archive://quill.tar
scp quill.sif H2USERNAME@hoffman2.idre.ucla.edu:
```

Container Registry

In the previous slides, we created a SIF file (quill.sif), then transfer it to Hoffman2.

Instead of this, we can upload our container to a **Container Registry**.

- These **Registries** are used store our containers on a remote, cloud server that can then be pulled/download anywhere that has apptainer.
  - DockerHub
  - GitHub Packages

DockerHub

Lets create a repo on DockerHub

Then push our docker/podman container to DockerHub

- Registry location `docker.io`

```
# Named our container to DockerHub locaton
podman tag quill:1.0 docker.io/charliecpeterson/quill:1.0
# Save DockerHub login info
podman login docker.io
# Push our final container to DockerHub
podman push docker.io/charliecpeterson/quill:1.0
```

Then pull the container on Hoffman2

```
apptainer pull docker://docker.io/charliecpeterson/quill:1.0
```

GitHub Packages

Lets create a repo on GitHub - Look for the Packages tab

- Pretty much the same, expect registry location is `ghcr.io`

```
# Named our container to GitHub locaton
podman tag quill:1.0 ghcr.io/charliecpeterson/quill:1.0
# Save GitHub login info
podman login ghcr.io
# Push our final container to GitHub
podman push ghcr.io/charliecpeterson/quill:1.0
```

Then pull the container on Hoffman2

```
apptainer pull docker://ghcr.io/charliecpeterson/quill:1.0
```

DockerHub and GitHub Packages are popular cloud registries. You can create and deploy a local container

Running Container

Once the container is on Hoffman2, submit job.

```
#!/bin/bash
#$ -cwd
#$ -o quill.$JOB_ID
#$ -j y
#$ -l h_rt=1:00:00,h_data=5G
#$ -pe shared 1
#$ -l arch=intel-gold*

# load the job environment:
. /u/local/Modules/default/init/modules.sh
module load apptainer/1.0.0

# Container part: apptainer exec QUILL.sif
# Command: QUILL.x /apps/QUILL/input.inp
time apptainer exec quill.sif QUILL.x test.inp
```

Submit job script

```
qsub test.job
```

More information on using Definition files

- https://apptainer.org/docs/user/1.0/definition_files.html

More information on using Dockerfiles

- https://docs.docker.com/engine/reference/builder/

Example 4: Anaconda

Anaconda is a very popular python and R distributaion for simplifying package installation

- Check out my Hoffman2 Happy Hour on Anaconda on our GitHub workshop page

Though Anaconda can be tricky installing in a container due to environment setup.

We will have an example using Anaconda to install an application in a container.

Building H2O

We will go over creating a definition file for a example with Anaconda.

We will install the software h2o.ai. This is a great machine learning platform that has Python and R libraries.

In this example, we will use Anaconda to install h2o packages inside python and R.

H2o definition file

The `h2o.def` file

- Definition file for Apptainer
- Install Anaconda from an env.yml
- Use of `%runscript` to setup Anaconda env for `apptainer run`
  - the `$@` take arguments as a string from the command line
  - `apptainer run h2o.sif "R CMD BATCH h2o.R"`

```
Bootstrap: docker
From: ubuntu:22.04


%labels
Author Charles Peterson <cpeterson@oarc.ucla.edu>


%post
export DEBIAN_FRONTEND=noninteractive
apt -y update ; apt -y upgrade
apt install -y  libxml2-dev libbz2-dev wget git gcc  libreadline-dev zlib1g-dev default-jre default-jdk

#Install anaconda
cd /tmp
wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh
```

```
bash Anaconda3-2022.05-Linux-x86_64.sh -b -p /opt/anaconda
cat << EOF1 > environment.yml
name: h2oai
channels:
  - h2oai
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=conda_forge
```

Building container

Create h2o.sif

```
sudo apptainer build h2o.sif h2o.def
```

Run h2o.R inside the container

```
apptainer run h2o.sif "Rscript h2o.R"
```

> **Note**
>
> The command `apptainer exec foo.sif COMMAND` will run a single COMMAND inside the container. The runscript will NOT be ran with the COMMAND.
>
> The command `apptainer run foo.sif` will run the runscript inside the container

Example 5: RStudio and Jupyter

Jupyter

This example will create a Jupyter session on Hoffman.

You can create a container with Jupyter and have any package you want.

We will use a container already build with python 3.8 and install jupyter, seaborn and pandas packages.

```
sudo apptainer build jupyter.sif jupyter.def
scp jupyter.sif hoffman2.idre.ucla.edu:
```

Start Jupyter on Hoffman2

```
qrsh -l h_data=10G
module load apptainer
hostname
apptainer exec jupyter.sif jupyter lab --ip 0.0.0.0
```

Keep note of the hostname of the node you landed on. Then you your local machine

```
ssh  -L 8888:nXXX:8888 username@hoffman2.idre.ucla.edu
#Where nXXXX is the compute node you landed on.

# Then open a web browser and type
http://localhost:8888 #or whatever port number that was displayed
#Note that you may need to change port 8888 if jupyter used a different port
```

> **Note**

We started jupyter inside the container by running

```
apptainer exec jupyter.sif jupyter lab --ip 0.0.0.0
```

Rstudio

Like Jupyter, you can use Rstudio Server to open a Rstudio session on your web browser.

I hosted a H2HH session on using Rstudio Server on Hoffman2 previously.

Create this container using Docker/Podman

```
podman build -f Dockerfile-rstudio -t rstudio:4.1.0 .
podman save rstudio:4.1.0 > rstudio.tar
apptainer build rstudio.sif docker-archive://rstudio.tar
scp rstudio.sif hoffman2.idre.ucla.edu
```

```
# get an interactive job
qrsh -l h_data=10G
# Create tmp directories
mkdir -pv $SCRATCH/rstudiotmp/var/lib
mkdir -pv $SCRATCH/rstudiotmp/var/run
mkdir -pv $SCRATCH/rstudiotmp/tmp
#Setup apptainer
module load apptainer/1.0.0
#Run rstudio
apptainer run -B $SCRATCH/rstudiotmp/var/lib:/var/lib/rstudio-server -B $SCRATCH/rstudiotmp/var/run:/var/run/
# This command will display some information and a `ssh -L ...` command for you to run on a separate terminal
```

```
ssh  -L 8787:nXXX:8787 username@hoffman2.idre.ucla.edu # Or whatever command was displayed earlier
# Then open a web browser and type
http://localhost:8787 #or whatever port number that was displayed
```

Note

We started RStudio by running

```
apptainer run -B {tmp_dirs} rstudio.sif
```

In the DockerFile, the runscript had `rserver`

Example 6: Build/run env

Overview

You may want to build a container with complex code required certain compilers MPI, and math libraries.

This example will build a code with Intel OneAPI, which includes.

- Intel compilers
- MKL (math libraries)
- IntelMPI

We will build the final container using `Multi-Stage` builds, where we will use two pre-built containers to compile and run code.

- `build` container
  - This container will have all the compilers, libraries, headers, etc needed to compile come
  - Used to compile software and maybe very large in size
- `run` container
  - This container will **ONLY** have the libraries and execuatables needed to run the code
  - Much smaller used to build the final container.

Build container

This example creates a build container with GCC 10.3.0 and OneAPI 2022.1.2

`Docker-build` file

```
# Base GCC/Oneapi building container image
# Ubuntu 20.04
# Charles Peterson <cpeterson@oarc.ucla.edu>
# Created 1/2022
# Updated 2/2022
# updated 7/2022

#Bulding Cmake
FROM ubuntu:20.04 as buildcmake
RUN apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
    ca-certificates curl  \
    && rm -rf /var/lib/apt/lists/*
RUN cd /opt  \
    && curl -LO https://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2-linux-x86_64.sh \
    && mkdir -p /opt/cmake  \
    && /bin/bash cmake-3.20.2-linux-x86_64.sh --prefix=/opt/cmake --skip-license
RUN rm -rf /opt/cmake/include \
    && rm -rf /opt/cmake/lib/pkgconfig \
    && find /opt -name "*.a" -exec rm -f {} \; || echo ""
RUN rm -rf /opt/cmake/share/doc
RUN rm -rf /opt/cmake/share/man
RUN rm -rf /opt/cmake/man

FROM ubuntu:20.04
ENV gcc_VER=10.3.0
```

Build and push build container

```
podman build -t docker.io/charliecpeterson/build:gcc10.3.0_oneapi -f Dockerfile-build .
podman push docker.io/charliecpeterson/build:gcc10.3.0_oneapi
```

Run container

This container has only the RUNTIME libraries

`Docker-run` file

```
# Base GCC/Oneapi building container image
# Ubuntu 20.04
# Charles Peterson <cpeterson@oarc.ucla.edu>
# Created 1/2022
# Updated 2/2022
# Updated 7/2022

FROM ubuntu:20.04
ENV gcc_VER=10.3.0


RUN apt-get update  \
    &&  DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends binutils libc6-dev gpg-ager
    && curl -fsSL https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB | apt-key add
    && echo "deb [trusted=yes] https://apt.repos.intel.com/oneapi all main " > /etc/apt/sources.list.d/oneAPI
    && apt-get update  \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
    intel-oneapi-common-vars \
    intel-oneapi-common-licensing \
    intel-oneapi-runtime-mkl\
    intel-oneapi-compiler-dpcpp-cpp-runtime \
    intel-oneapi-compiler-fortran-runtime \
    intel-oneapi-runtime-mpi \
    && rm -rf /var/lib/apt/lists/*  \
    && apt-get remove -y curl ; apt-get purge -y --auto-remove -o APT::AutoRemove::RecommendsImportant=false
```

Build and push run container

```
podman build -t docker.io/charliecpeterson/run:gcc10.3.0_oneapi -f Dockerfile-run .
podman push docker.io/charliecpeterson/run:gcc10.3.0_oneapi
```

Build our software

Now that we have build and pushed are container to DockerHub (or GitHub), we can build our final container

```
# Compile code from build container
Bootstrap: docker
From: charliecpeterson/build:gcc10.3.0_oneapi
Stage: devel

%files
  MPI_hello.c /MPI_hello.c

%post
  mkdir -pv /apps
  cd /apps
  mv /MPI_hello.c ./
  mpiicpc -o MPI_test.x MPI_hello.c
  rm MPI_hello.c

# move binary from build env to run env
Bootstrap: docker
From: charliecpeterson/run:gcc10.3.0_oneapi
Stage: final

%files from devel
  /apps /apps

%environment
export PATH=/apps:$PATH
```

Build container

```
sudo apptainer build hello.sif hello.def
scp hello.sif hoffman2.idre.ucla.edu:
qsub hello.job
```

Note

The same Multi Stage approach can be build by Docker/Podman in a Dockerfile

Things to Think About

Tips

Size of container

- Try to keep the size of your container small and minimal
  - Only have the things necessary for your applications to run
- Large containers will need more **memory** and will take more take to start up
- Use `Multi-Stage` builds with Build/Run containers

Build from an existing container

- Look for images on DockerHub, NGC
- Build you own and upload to DockerHub/GitHub

- Good idea to build a sandbox container, then create definition file
  - test out commands in sandbox while making the def file!

More Things to Think About

- Share .sif files with your friends!
  - Save your (Docker) containers to DockerHub or GitHub Packages
  - Create Dockerfile/Def files to recreate or modify containers
- Find examples of Dockerfiles and Apptainer def files on our GitHub Page
  - https://github.com/ucla-oarc-hpc/hpc_containers

Thank you!

Questions? Comments?

Charles Peterson

- cpeterson@oarc.ucla.edu
- https://charlespeterson3.com

Look at for more Hoffman2 workshops at https://idre.ucla.edu/calendar

- Search for HPC