# DS 7347
# High-Performance Computing (HPC) and Data Science
# Session 8

Robert Kalescky
Adjunct Professor of Data Science
HPC Research Scientist

May 19, 2022

Research and Data Sciences Services
Office of Information Technology
Center for Research Computing
Southern Methodist University

# Outline

# Session Question

# Session Question

Why don’t HPC systems use Docker?

# Docker

# Docker Image Registries

- There are several public and private sources for Docker images.
- Images can be used as the base image for custom images.
- Already optimized images can help with reproducible and efficient development workflows.

# Docker Image Registries

| | |
|---|---|
| Docker | https://hub.docker.com |
| Quay.io | https://quay.io |
| NVIDIA | https://catalog.ngc.nvidia.com |
| Intel | https://hub.docker.com/u/intel |
| AMD | https://hub.docker.com/u/amdih |

# Pulling Images from Registries

```
#!/usr/bin/env sh

pull_and_check() {
  name=${1}
  tag=`echo ${name} | cut -d':' -f2`
  docker pull ${name}
  docker image ls | egrep "REPOSITORY|${tag}"
  docker run --rm -it ${name} bash
}

images[0]="ubuntu:jammy"
images[1]="nvcr.io/nvidia/nvhpc:22.3-devel-cuda_multi-ubuntu20.04"
images[2]="nvcr.io/nvidia/nvhpc:20.7-runtime-cuda10.1-centos7"

for image in ${images[@]}; do
  pull_and_check ${image}
done

```

# Multi-Architecture Builds

- Images are CPU-architecture specific
- Docker supports multi-architecture builds
  - Platforms: amd64, arm32v5, arm32v6, arm32v7, arm64v8, i386, ppc64le, and s390x
  - `docker build --platform` with single platform
  - `docker buildx --platform` with list of platforms
- Builds on non-native platforms will be slower as it is running through a virtual machine

# Basic Dockerfile

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update &&\
 apt-get -y install\
 python3-pip\
 python3-numpy\
 python3-pandas

RUN pip3 install\
 jupyterlab

ENTRYPOINT ["python3"]

```

# Docker Multi-Architecture Builds

```
#!/usr/bin/env sh

# Create builder to build images
docker buildx create --name builder --use

# Build images for x86_64 and ARM64
docker buildx build --platform\
 linux/amd64,linux/arm64 -t rkalescky/python3:latest\
 -f python3.dockerfile --push .

# Inspect the built images
docker buildx imagetools inspect rkalescky/python3:latest

```

# Multi-Stage Builds with Docker

- Images with build tools can be very large.
- Use the needed image for building.
- Use the smallest image for running.
- Define both the build and execution in a single Dockerfile.

# Basic Multi-Stage Dockerfile

```
1  FROM nvcr.io/nvidia/nvhpc:22.3-devel-cuda_multi-ubuntu20.04 as builder
2  WORKDIR /build
3  COPY hello_world.cpp ./
4  RUN nvc++ -Bstatic -o hello_world hello_world.cpp
5
6  FROM alpine:3.15.4
7  WORKDIR /opt/hello/bin
8  COPY --from=builder /build/hello_world ./
9  ENTRYPOINT ["/opt/hello/bin/hello_world"]
10
```

# Docker Multi-Architecture Builds

```
#!/usr/bin/env sh

# Build image using multi-stage Dockerfile
docker build -t hello:20.04 -f hello_world.dockerfile .

# Run the image
docker run hello:20.04

# Note the size difference
docker image ls | egrep "hello|22.3-devel"

```

# Singularity

# Building Singularity Images

- Singularity has it's own image definition language.
  - Requires (re)writing the definition file.
  - Requires root or "fakeroot", which is not widely available on HPC systems.
  - Can be done on a Linux system with Singularity installed and them copying the image.
  - Not generally recommended as there would be two definition files to maintain, presumably Docker and also Singularity.
- Pull from Docker registries.
  - Requires pushing and pulling of Docker images.
- Build from Docker archives.
  - Requires exporting, copying, and conversion of Docker images.

# Pulling Docker Containers

```
#!/usr/bin/env sh

# Build images for both x86_64 and ARM64
. ./docker_buildx.sh

# Consume the image on M2 via Singularity
ssh m2 'bash -l -c "module load singularity\
 && singularity run docker://rkalescky/python3 -c \"import numpy as np;
↪  print(np.pi)\""'
```

# Pulling Docker Containers

```
10  # Pull Docker image to a Singularity image
11  ssh m2 'bash -l -c "module load singularity\
12   && singularity pull -F python3_3.9.13-slim.sif docker://python:3.9.13-slim\
13   && ls -lh ./python3_3.9.13-slim.sif\
14   && singularity exec ./python3_3.9.13-slim.sif python3 -c \"import sys;
     ↪  print(sys.version)\""'
15
16  # Singularity mount points
17  ssh m2 'bash -l -c "module load singularity\
18   && echo $SINGULARITY_BIND"'
19
```

# Converting from Docker Archives

```
15  # Export, upload, convert, and run on M2 via Singularity
16  docker save python3:20.04 | ssh m2 'bash -l -c "n=python3_20_04\
17   && cat > ~/$n.tar\
18   && module load singularity\
19   && singularity build -F $n.sif docker-archive:$HOME/$n.tar\
20   && singularity exec $n.sif whoami"'
```

# Spack

# Spack Containerize

- Build images defined by Spack environments.
- Spack-based build optimizations are preserved.
- Intermediate Dockerfile uses multi-stage builds
- Currently does not work for multi-architecture builds.

# Define a Spack Environment

```yaml
spack:
  specs:
  - gromacs+mpi
  - mpich
  container:
    format: docker
    images:
      os: "ubuntu:20.04"
      spack: develop
```

# Build the Image from the Environment

```
#!/usr/bin/env sh

# Define the Spack environment
cat spack.yaml

# Build container definition file
spack containerize > Dockerfile

# Build the container image
docker build -t gromacs:latest .

```

# Readings and Assignments

# Readings and Assignments

## Readings

None

## Lab

- Use `spack containerize` to produce a Dockerfile of the Spack environment from Lab 1.
- Build and export the Docker image.
- Use Singularity on M2 to convert and run the image.
- Commit to your class repo:
  1. `assignments/lab_02.dockerfile`.
  2. `assignments/lab_02.{png,jpg}` of your terminal session showing a Python session with NumPy, Matplotlib, and Pandas loaded.
- Due 12:00 AM Central, Thursday, May 26, 2022