DS 7347
High-Performance Computing (HPC) and Data Science
Session 7

Robert Kalescky
Adjunct Professor of Data Science
HPC Research Scientist

May 17, 2022

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

Outline

Lab Peer Review

Lab Peer Review

Group Discussion

- Assigned pairs will go to breakout rooms
- Discuss:
  - Progress
  - Problems
  - Ideas

Comments

- Provide a summary of discussion concerning your lab progress and report in `assignments/lab_01.md`; note your peer reviewers name
- Commit `assignments/lab_01.{yaml,md}` to your class repo
- Due 12:00 AM Central, Thursday, May 19, 2022

Session Question

Session Question

Describe the differences between containers and virtual machines?

Containers

Importance of Containers

Before Containerization

- Goods had to be loaded and unloaded individually
- Inefficient - it was not uncommon to spend more time loading and loading goods than transporting them
- Insecure - goods had be handled by many people, increasing the chance for loss and theft
- Inaccessible - Long distance shipping only available to the wealthy

After Containerization

- Standardized - containers are all the same size and weight allowances
- Efficient - containers are easy to load and unload and transfer to other modes of transportation
- Secure - goods may be secured in containers from source to final destination
- Available - cost effective to ship goods across the world

Common Issues with Software Stacks

- My software doesn't build on this system...
- I'm missing dependencies...
- I need version 1.3.2 but this system has version 1.0.2..
- I need to re-run the exact same thing 12 months from now...
- I want to run this exact same thing somewhere else...
- I want my collaborators to have the same exact software as me...
- I've heard about these Containers, can I just run that?
- Can I run docker on this HPC system?

What about computing?

- It's common to run on multiple systems with different requirements
- We would like to avoid installing the same sets of software again and again
- We would like other people to run our software without our help
- We would like to preserve a known configuration that our software works in

Possible Solution: Containers

- What are Containers?
- Uses a combination of Kernel “cgroups” and “namespaces” to create isolated environments
- Long history of containers Solaris Zones (2005), LXC(2008), LMCTFY/Google and then Docker(2013).
- Entire ecosystem has grown around containers including open standards and governance.

Possible Solution: Containers

- A lightweight collection of executable software that encapsulates everything needed to run an application
    - Minus the OS kernel
    - Based on Linux only
- Processes and all user-level software is isolated
- Creates a portable* software ecosystem
- Think **`chroot`** on steroids
- Docker is the most common tool today
    - Available on all major platforms
    - Widely used in industry
    - Integrated container registry via Dockerhub

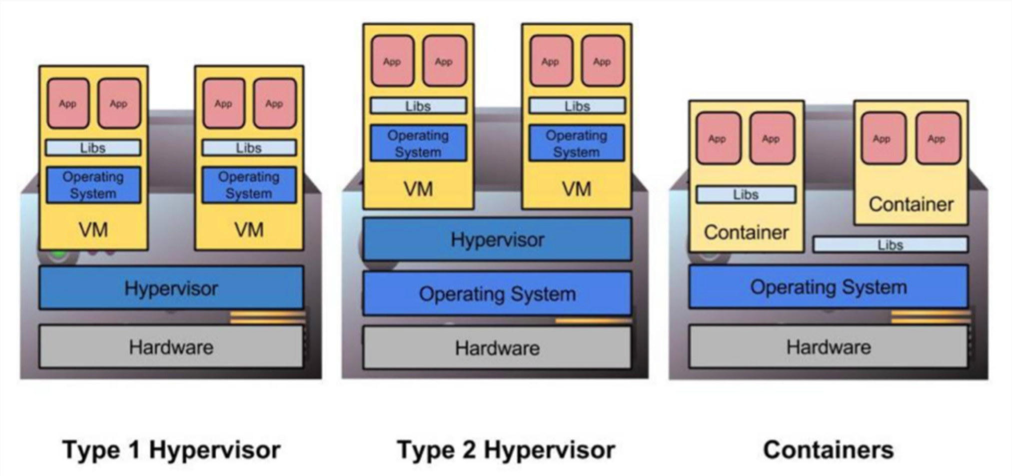
Containers Overview

- Containers offer the ability to run fully customized software stacks, *e.g.* based on different Linux distributions and versions
- Containers are not virtual machines, where an entire hardware platform is virtualized, rather containers share a common kernel and access to physical hardware resources

Hypervisors and Containers

- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment using a common kernel
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware

Hypervisors and Containers



Container Benefits

Performant Containers can perform at near native performance.

Flexible Install (almost) any software you need.

Reproducible Define complex software environments that are verifiable.

Compatible Built on open standards that works on all major Linux distributions.

Portable Build once and run (almost) anywhere.

Container Limitations

Hardware Containers are (currently) limited to the same CPU architecture (x86_64, ARM, Power, etc.) and binary formats

Software Requires glibc and kernel compatibility between host and container. Other kernel level APIs may also need to be compatible (e.g. CUDA/GPU drivers, network drivers, etc.)

Filesystem Paths can be different when viewed from inside or outside of a container

Nomenclature

Image A read-only template that defines how to create a container

Container An instantiation of an image, a running instance

Container Runtime Tool or service to execute and manage containers

Registry A service that is used to store and distribute images

Docker and HPC

- We don’t allow direct Docker use on M2
- Docker’s security model is designed to support users ”trusted” users running ”trusted” containers (e.g. users who can escalate to root access)
- Docker is not designed to support scripted / batch based workflows
- Docker is not designed to support parallel applications

Singularity Features

- Containers are a single image file
- No root owned daemon processes
- User inside containers are the same as users outside the container (no contextual changes)
- Supports shared, multi-user environments
- Supports HPC hardware such as GPUs and Infiniband networks
- Supports HPC applications like MPI

Common Use Cases

- Converting Docker containers to Singularity
- Building and running software that require newer systems and libraries
- Running commercial software binaries that have specific requirements

Singularity Workflow

- Build your Singularity containers on a local system you have root or sudo access. Alternatively build a Docker container
- Transfer your container to M2 or other HPC system. If you used Docker, you will need to convert the image
- Run your Singularity containers

Dockerfile

- Is a “recipe” for how to construct an image.
- Starts `FROM` a defined base image.
- Several basic commands (`ADD`, `COPY`, `RUN`, etc.) can be applied to mutate the image to the desired state.
- Metadata labels can also be added to provide information about the image.
- In addition to the file system changes, the Dockerfile can also control settings like the environment, the starting directory, and default commands.

Docker Build Script Example

```
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"]

```

Building and Converting Docker Container Images

```
# Build container image
docker build --platform linux/amd64 -t python3:20.04 -f python3.dockerfile .

# Run default entry point
docker run -it python3:20.04

# See who the default user
docker run --entrypoint /usr/bin/whoami -it python3:20.04

# Running arbitrary commands
docker run --entrypoint /bin/bash -it python3:20.04

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

Readings and Assignments

Readings and Assignments

Readings

None

Assignment

- Write Dockerfile that defines a container that includes and runs the script from Assignment 4, see Dockerfile reference.
- Commit `assignments/assignment_04.dockerfile` to your class repo.
- Due 12:00 AM Central, Tuesday, May 24, 2022