Enabling Reproducibility with Docker

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

Enabling Reproducibility with Docker

- Daniel Trahan
- *Email: Daniel.Trahan@Colorado.edu*
- *RC Homepage: https://www.colorado.edu/rc*
- *RC Email: rc-help@colorado.edu*

- Slides available for download at: https://github.com/ResearchComputing/Containers_Fall_2021

Outline

- Part 1: Container fundamentals and Docker (9/22/21)
  - Reproducibility and the Case for Containers
  - Containers
  - Docker
    - Images and Containers
    - Commands
    - File Access
    - Building Docker Images
    - Dockerhub
- Part 2: Containers for HPC w/ Singularity (9/29/21)

Demo Files:

- Before we begin I will note that this tutorial will have interactive components.
- If you would like to participate in the demos provided for this tutorial then go ahead and clone the test files from the Github.
  1. Navigate to a desired directory
  2. Clone the repository

     ```
     git clone https://github.com/ResearchComputing/Containers_Fall_2021
     ```
  3. Navagate into the directory and store the path into a variable.

     ```
     cd Containers_Fall_2021
     export CONTAINER_ROOT=$(pwd)
     ```

Reproducibility and Research

- Scientific Software is often challenging to work with
  - Difficult installation
  - Low support from the developers
  - Very outdated
  - Complex Dependency trees
- Because of this its often desired for a software to be repeatable and accurate.
- *But installs are only done once. Why should I care about reproducible applications.*

Reproducibility and Research

- Research is Collaborative
  - Team members work together to get projects done.
  - Reproducibility ensures all members of a team can provide productivity towards a project.
- Research is Correcting
  - Research is hard
  - Academic reviews are commonplace
  - Someone may wish to accurately reproduce your work
- Research is Continuous
  - You may be working on a single project for a long period of time
  - What happens in you move, but bring your work to another system?

Options for reproducibility

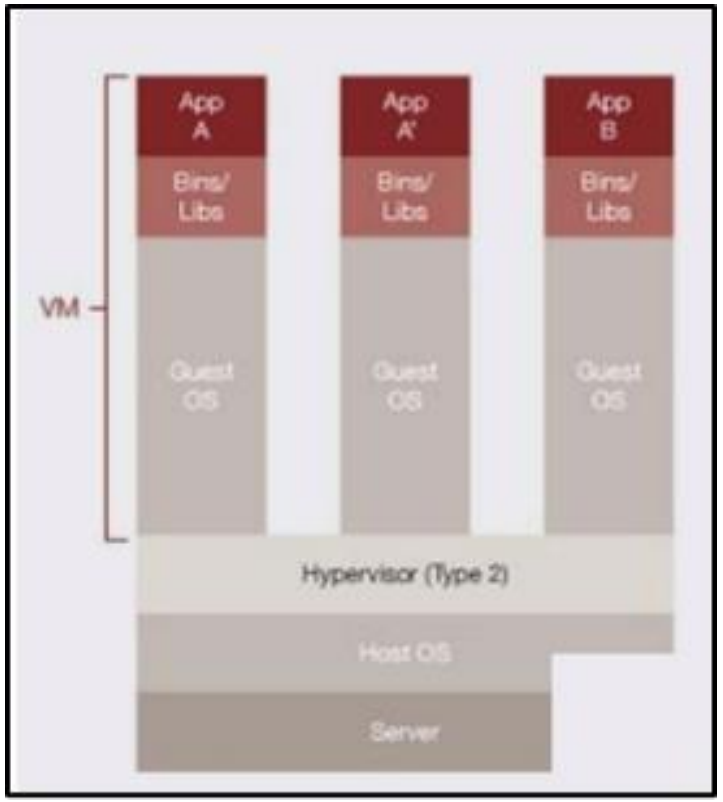
- Lots of options!
  - Detailed instructions
  - Software bundles
  - Virtual Environments
    - Python, Anaconda, Spack
- But do they really enable accurate reproducibility?
  - Incorrect installs?
  - Hardware or OS?
  - Performance?

Containers

- A Container is a packaged bundle of OS, libraries, software and files that runs as a process under a host OS
- Containers use an application on the host operating system called a Container Manager
  - Manages operating system and libraries run as containers
  - Like virtual machines, but does not need dedicated CPUs memory or storage

Virtualization (1)

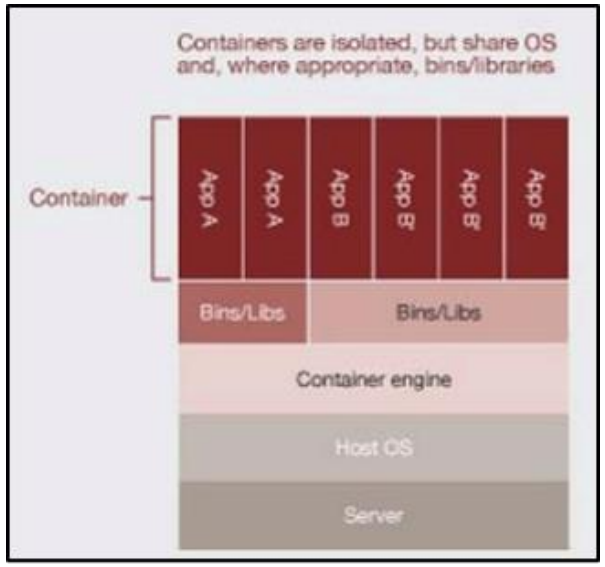
- Virtualization is a technology that utilizes software to abstract components of a technology
- The most common application is in Hardware Virtualization
  - Virtual Machines
    - Partitions off Memory, CPU, GPU, and Storage
    - Runs a virtual OS
    - Runs software on the virtualized machine
  - Examples: VMware, Virtualbox



Material courtesy: M. Cuma, U. Utah

Virtualization (2)

- Another use of virtualization is in OS Level Virtualization
  - Can run many isolated guest OS instances under a host OS
  - This virtualization is what is used by Docker and other container software.
  - Best of both worlds!
    - Isolated environments
    - No hardware partitioning



Material courtesy: M. Cuma, U. Utah

Containerization Software

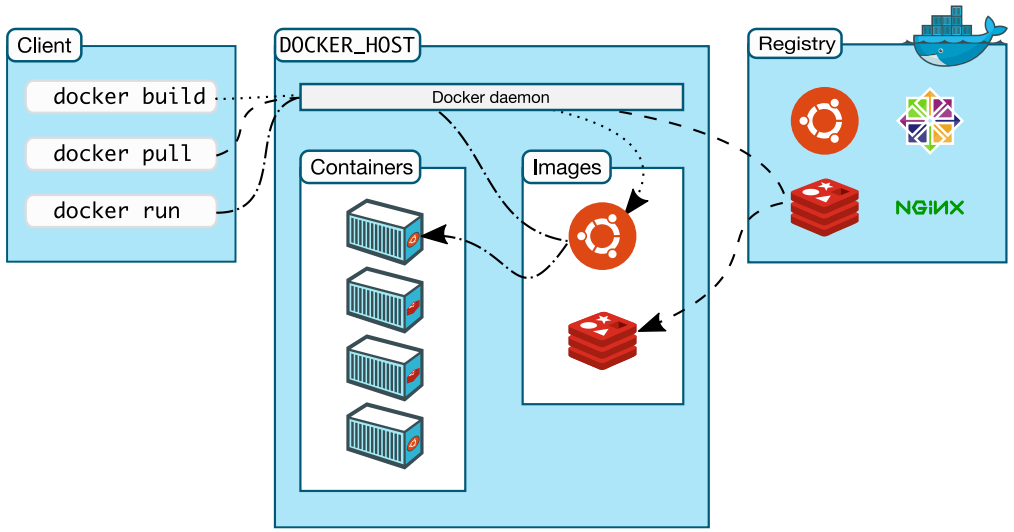
- Docker
  - Well established – largest user base
  - Has Docker Hub for container sharing
  - Problematic with HPC (Fix incoming!)
- Singularity
  - Designed for HPC
  - Second largest userbase
  - Developed for scientific use
- Charliecloud; Shifter
  - Designed for HPC
  - Based on Docker
  - Less user-friendly

Installing Docker

- Docker Community Edition
  - Comfy GUI to help keep track of containers and images!
  - Available on all operating systems
  - Windows users can enable WSL2 support following the instructions here: https://docs.docker.com/docker-for-windows/install/
- Docker toolbox
  - Legacy solution for Windows and Mac for versions that do not meet the version requirements.
  - Utilizes the Virtual Box hypervisor for virtualization

Docker Nuts and Bolts

- Docker runs on a concept of images and containers.
  - Images: Saved snapshots of a container environment.
    - Made from a Dockerfile or pulled from Docker Hub
    - Stored in the Docker cache on your disk
    - Immutable (mostly…)
  - Containers: Instances of images that are generated by Docker when an image is 'run'
    - Instance of image running in memory
    - Ephemeral and state cannot be saved
    - Can be run interactively



Docker ‘Hello World’

- Let's start with something simple:
  - Docker "Hello, World!"
  - Relatively small image
  - No dependencies
  - Built as a general test case
- Command we will run:

```
docker run hello-world
```

Docker Commands

- Docker Commands are usually in the form of:

```
docker <sub-command> <flags> <target/command>
```

- Examples:

```
docker run -it myimage
docker container ls
docker image prune
```

Launching a Docker Container

- Launch docker image as a container:

```
docker run <image-name>
```

- Run a docker image interactively:

```
docker run -it <image-name>
```

- If an image is not on the system, then Docker will search Dockerhub to see if the image exists.
- Specify commands after your image to execute specific software in your container.

```
docker run <image-name> <program>
```

Exploring a Docker Container

- Docker containers are running tiny operating systems!
- We can explore the operating system by invoking a shell

```
docker run -it ubuntu bash
```

- This command **launches the ubuntu Docker container with the command 'bash'**

Demo 1: Running a Container

Demo 1: GROMACS

- GROMACS is a molecular dynamics application that can often be a complex and challenging installation for the average user. Linux and Mac only
  - Dense Documentation
  - Software requires compilation
- Luckily, this can be trivialized with Docker!
  - Run the command:

```
docker run gromacs/gromacs gmx help commands
docker run -it gromacs/gromacs
```

Docker Image/Container Commands

Container Commands	
`docker container ls`	List docker containers currently running:
`docker container rm <container>` `docker rm <container>`	Remove (an) container(s):
`docker container prune`	Remove all stopped containers

Image Commands	
`docker image ls`	List docker images stored in cache:
`docker image rm <image>` `docker rmi <image>`	Remove (an) image(s):
`docker image prune`	Remove unused images

Docker Image/Container Commands

Commands	
`docker info`	Shows Docker system-wide information
`docker inspect <docker-object>`	Shows low-level information about an object
`docker config <sub-command>`	Manage docker configurations
`docker stats <container>`	Shows container resource usage
`docker top <container>`	Shows running processes of a container
`docker version`	Shows docker version information

- More details and commands can be found *on the docker documentation page*

Dockerhub

- The place where containers live!
- Dockerhub is a Docker hosted library of public and private Docker images.
  - Free and unlimited public images
  - 1 free private repository
- Great for hosting images for fellow researchers
- Commands like git

Building a Docker Container

- To build a docker container, we need a set of instructions Docker can use to set up the environment.
  - Dockerfile
- Once we set up our Dockerfile we can use the command

```
docker build –t <image-name> .
```

- Then we can run the image with our docker run command

```
docker run <image-name>
```

Whats in a Dockerfile

- A Dockerfile is simply a text file that contains instructions to build and setup a default Image
  - Commands to build
  - Setting commands
- Requires a source Image

```
mtrahan41@MTrahanRazor15:[ ubuntu-gcc ]$ cat Dockerfile
FROM ubuntu:18.04

RUN apt-get update; \
    apt-get install nano -y; \
    apt-get install gcc -y; \
    mkdir target;

WORKDIR /target
```

Demo 2: Building a Docker Image

Demo 2: Ubuntu w/ GCC

- For this first example we will be building a custom Ubuntu image that will provide a location to run the GNU Compiler Collection.
- Dockerfile provided:
- Need to build:
    1. Navigate to the directory:
       ```
       cd $CONTAINER_ROOT/Containers_Fall_2021/dockerdemo/ubuntu-gcc
       ```
    2. Build the image with:
       ```
       docker build -t test-gcc .
       ```
    3. Run the image as a container:
       ```
       docker run -it test-gcc
       ```

Mounting and File Access (1)

- So now that we have a working container, how can we access the test files we downloaded?
  - Mounting directories: Bind Mount
    - Allows the docker container to access files on the host OS
    - Choose host's *source directory*, files in the directory will be moved to the container's *target directory*
      - **Source Directory**: Directory on the host system. Never within a container.
      - **Target Directory**: Directory in the Docker Container. Never on the host system.
    - A flag set within the docker run command:

```
docker run -v <source-dir>:<target-dir> <image>
```

Mounting and File Access (2)

- Mounting directories: Volume Mount
  - Same concept, but volumes are stored within docker cache.
  - Create Docker volumes in your terminal and link your volume directory
  - Similarly linked through the docker run command.

```
docker run -v <volume-name>:<target-dir> <image>
```

Demo 2 (Cont.): Mounting

- Returning to our demo, can we give our container access to our test files?
- Let's use a bind mount!
- In the directoru where our Dockerfile lives, use this command (all on one line):

```
docker run -it -v $(pwd)/source:/target happy-gcc
```

- We can cd /target and run our test files!
- Command

```
gcc hello.c -o hello.exe
./hello.exe
```

Modifying a Docker Image

- Suppose you have an existing docker image and want to make changes…
  - Rebuild Dockerfile!
  - Usually a bit cumbersome
- No Dockerfile?
  - Use docker commit!

```
docker run -it <image-name> bash # or any shell...
```

  - Then commit it to the image

```
docker commit <image-id> <image>
```

Dockerhub Commands

- Download and upload docker images with ease.

```
docker run <image>
docker pull <image>
```

- Uploading a little more complicated...
  - Sign in with:

```
docker login
```

  - List docker images with:

```
docker image ls
```

  - Tag your image:

```
docker tag <image-id> <your-username>/<image-name>:<tag>
```

  - Push!

```
docker push <your-username>/<image-name>
```

Demo 3: NCL container

Demo 3: NCL Container

- For this next example we will be building a Docker image that will run the NCAR Command Language (NCL)
- Dockerfile provided
- Same process:
  1. Navigate to the Dockerfile found at:
     `$CONTAINER_ROOT/Containers_Fall_2021/dockerdemo/ncl`
  2. Build the Dockerfile and name the image: "ncl-demo"
  3. Run "ncl-demo"
- Can we test a sample script?

Docker Compose

- External Utility that can create and install docker images.
- Builds docker images based on a docker-compose.yml file.
- YAML: YAML Ain't Markup Language
    - Data serialization language
- Describes containers you wish to build with what features.
- Not a docker command but comes bundled with docker!

Docker Compose Commands

- Build all containers in YAML file

```
docker-compose build
```

- Build and run all containers in YAML file:

```
docker-compose up
```

- List all containers in YAML file:

```
docker-compose images
```

- Run a one-off command from a container:

```
docker-compose run <container-name> <command>
```

Questions?

Additional Resources

- Docker: https://www.docker.com/
- Docker Docs: https://docs.docker.com/
- Docker Hub: https://hub.docker.com/

Thank you!

- Please fill out the survey: http://tinyurl.com/curc-survey18
- Contact information: rc-help@Colorado.edu
- Slides:

  https://github.com/ResearchComputing/Containers_Fall_2021