CONTAINERIZING JOBS ON THE ACCRE CLUSTER WITH SINGULARITY
VIRTUAL MACHINE (VM)

- Uses software to emulate an entire computer, including both hardware and software.

Host Computer

Virtual Machine

Host Resources: CPU, Disk Space, RAM, Network, etc

Host’s resources carved off for VM

VMs can be created with tools like VirtualBox, Vagrant, and VMware.
ADVANTAGES OF VMs

• Testing/development outside of production context
• Overcome software portability issues (e.g. OS- or processor architecture-related)
• Multiple computer systems can be run within a single “bare metal” computer
• Consolidation and isolation of applications/services running on a single server
• VM recovery very simple through snapshotting
• Decreased hardware maintenance
Containers

A container is similar to a VM but with the following important differences:

- Emulates just an operating system, not the entire system of underlying hardware
- Shares the underlying kernel of the host operating system
- Requires fewer resources (e.g. generally requires less CPU cycles, disk space, and RAM than a VM)
- Generally faster boot and deployment times

The most widely used containerization tool in use today is Docker

- Allows Linux containers to be created that are isolated from the rest of the system
- Uses an image-based deployment model
- Allows access to the host’s root filesystem
- By default, users can gain root access to a host’s root filesystem from inside a Docker container
CONTAINERS IN SCIENTIFIC AND HIGH-PERFORMANCE COMPUTING

• Allowing root access in a shared environment is a non-starter

• **Singularity** is designed to overcome this problem
  
  • Permissions inside a container are the same as those outside of the container

• Allows users to access their files that are stored outside the container

• Designed with performance in mind for HPC applications

• Supports conversion of Docker images to Singularity images

• Installed on the ACCRE cluster!
**Terminology: Image vs. Container**

- **Image**
  - A file that includes a virtual filesystem and software installed onto that filesystem
  - Typically build a single time (on personal machine with root access) and use multiple times (on cluster)
  - Immutable (i.e. does not change)

- **Container**
  - A running instance of an image
  - Possible to access files outside of container and to write new files to locations outside of container

---

**Diagram:**

1. Create & bootstrap image
2. Copy image to cluster
3. Launch container

**Personal Linux Machine**

- **Image**
  - `/home`
  - `/scratch`
  - `/data`

**Cluster**

- **Image**
  - `/home`
  - `/scratch`
  - `/data`

- **Container**
  - `/home`
  - `/scratch`
  - `/data`
Typical Singularity Workflows

- Singularity workflow 1: Build your own image
- Singularity workflow 2: Use existing image
- Singularity workflow 3: Convert Docker image
Typical Singularity Workflows

• Singularity workflow 1: Build your own image
• Singularity workflow 2: Use existing image
• Singularity workflow 3: Convert Docker image
**Singularity Workflow 1: Build it Yourself**

1. Create image
   - Images are relatively large (on the order of 1GB or more depending on amount of software needed).
   - Root access *not* required for this step.

2. Bootstrap image
   - Build image based on user-defined recipe in a .def file. Root access required!

3. Run container from image
   - Use image (created in steps above) to launch container
   - This can be done anywhere Singularity is installed (e.g. ACCRE cluster)
   - The container can be run as many times as you’d like
Singularity is currently only supported on Linux platforms. Installation is very straightforward but you need root access to a Linux box!

**Creating a Singularity Image**

```
singularity create ubuntu16.img
```

- Creates a blank image with a virtual filesystem
- Default image size is 768 MiB
- Requires root privileges to run (hence the `sudo` command)

```
singularity create --size 2048 ubuntu16.img
```

- Creates image of size 2048 MiB (2 GiB or ~2.1 GB)
- Appropriate size depends on amount of software to be installed
- Can also use the `--force` option to overwrite an image with a new blank one
Bootstrapping an Image (1/3) – Definition File

**sudo singularity bootstrap ubuntu16.img Singularity**

- Uses recipe in text definition file (called Singularity) to install OS and software into image
- Requires root privileges to run (hence the `sudo` command)
- Creating a definition file is the majority of the work when building a new image
- A definition file contains a header and multiple sections

**Header**

- Specifies information about the Linux OS distribution you want installed
- Supports pulling from Docker Hub! For example:

  Bootstrap: docker
  From: ubuntu:latest
  .
  .
  .

Header section for Singularity file. Here, we are installing the latest version of Linux Ubuntu.
**%runscript**

- Commands that run each time a container is launched using the “singularity run” subcommand
- Often includes handling of command line arguments you want to pass to container

**%post**

- Commands that run once inside container during bootstrap process
- Software installation commands go here

**%test**

- Commands that test the proper function of the container
- Section runs at the end of bootstrapping; disable w/ “singularity bootstrap --notest”

---

Example Singularity definition file.

```
Bootstrap: docker
From: ubuntu:latest

%runscript

# Pass all args from “singularity run” to python3 interpreter
exec python3 "$@

%post

# Enables access to ACCRE storage
mkdir /scratch /data /gpfs21 /gpfs22 /gpfs23 /dors

# Update apt-get’s (Ubuntu package manager) package list
apt-get -y update

# Install Python3, Numpy, Nose, and GCC compiler
apt-get -y install python3 python3-numpy python3-nose gcc vim

%test

# Run NumPy tests to verify it works
python3 -c "import numpy as np; np.test()"
```
%files
• Used for copying files from outside of image to inside the container
• Runs once immediately after %post section
• <source> <destination>

%setup
• Commands performed once outside container prior to %post section
• Useful if %post section needs files that live outside container

%labels
• Define custom labels/attributes for your container in this section if desired

%environment
• Define environment variables that should exist within running container
• <VARIABLE_NAME> <VALUE>
WORKING WITH LINUX PACKAGE MANAGERS

- Debian (e.g. Ubuntu) Linux distributions : **apt-get**
- Red Hat (e.g. CentOS) Linux distributions : **yum**

**apt-get install pkg1 pkg2**
- Install pkg1 and pkg2, as well as their dependencies
- Package names must be an exact match (case sensitive)
- Use –y option to automate

**apt-get update**
- Downloads references to the latest versions of all packages and their dependencies
- Generally should be run prior to installing a new package

**apt-cache search keyword**
- Determine whether package is available to install
- Returns any package name containing the keyword

**apt-get upgrade**
- Updates the versions of *installed* packages
- Need to run **apt-get update** first!

**apt-get dist-upgrade**
- More intelligent version of apt-get upgrade
- Resolves complex dependency issues
**RUNNING A CONTAINER (1/3)**

```bash
singularity run ubuntu16.img
```

- Runs recipe defined in `%runscript` section of definition file
- Often `%runscript` section written to accept command line arguments from the `singularity run` subcommand
- No root privilege required to run

Excerpt from example Singularity definition file.

```
%runscript
python3 "$@
```

Running `bmark.py` through `python3` interpreter in a Ubuntu 16 container.

```bash
$ module load GCC Singularity
$ singularity run ubuntu16.img bmark.py
```

It’s good practice to define your most common workflows through a `%runscript` section for reproducibility.
RUNNING A CONTAINER (2/3)

singularity exec ubuntu16.img foo.sh

- Runs an arbitrary command within the context of the container
- Unlike the “run” subcommand, you do not need to predefine a
  command within a section of your definition file to use “exec”
- No root privilege required to run
- More than a single command can be bundled up in a bash script

The “run” subcommand is generally preferred to “exec” because the former provides
better documentation & reproducibility.

$ module load GCC Singularity
$ singularity exec ubuntu16.img python3 --version
$ singularity exec ubuntu16.img which python3
$ singularity exec ubuntu16.img python3 bmark.py
$ singularity exec ubuntu16.img bash commands.sh
**Running a Container (3/3)**

Launch interactive shell session (REPL) within container

- Good for initial testing and environment verification
- No root privilege required to run, unless used with the --writable option, which allows you to write new data (e.g. install software) that will persist to future instances of container image

Avoid using the --writable flag for tools intended for production. The %post section provides better documentation and reproducibility.

```
$ module load GCC Singularity
$ singularity shell ubuntu16.img
$ sudo singularity shell --writable ubuntu16.img
```
Typical Singularity Workflows

• Singularity workflow 1: Build your own image
• Singularity workflow 2: Use existing image
• Singularity workflow 3: Convert Docker image
**Singularity Workflow 2: Use Existing Image**

1. Import or pull image from the web
   - Image can be hosted on Docker Hub or Singularity Hub
   - No need to install Linux on your personal machine w/ root access!
   - Less flexible than bootstrapping, must use image as-is

2. Run container from image
   - Use image (created in steps above) to launch container
   - This can be done anywhere Singularity is installed (e.g. ACCRE cluster)
   - The container can be run as many times as you’d like
**DIRECT IMPORT/PULL OF EXISTING CONTAINERS**

- You can import or pull images directly from **Docker Hub** or **Singularity Hub**
  - Avoid reinventing the wheel

```
singularity create ubuntu.img
singularity import ubuntu.img docker://ubuntu:latest
```
  - Creates a blank image with a virtual filesystem
  - Copies image from Docker Hub into your local image

```
singularity create ubuntu.img
singularity import ubuntu.img shub://singularityhub:ubuntu
```
  - Creates a blank image with a virtual filesystem
  - Copies image from Singularity Hub into your local image

```
singularity pull docker://ubuntu:latest
```
  - Pulls image directly from Docker Hub with same name as remote image
Typical Singularity Workflows

• Singularity workflow 1: Build your own image
• Singularity workflow 2: Use existing image
• Singularity workflow 3: Convert Docker image
**Singularity Workflow 3: Convert Docker Image**

1. Convert Docker image to Singularity image
   - Singularity developers provide a Docker image specifically for converting a Docker image to a Singularity image
   - Single Docker command (see next slide)

2. Run container from image
   - Use image (created in steps above) to launch container
   - This can be done anywhere Singularity is installed (e.g. ACCRE cluster)
   - The container can be run as many times as you’d like
CONVERTING DOCKER IMAGES TO SINGULARITY IMAGES

• If you have Docker installed on your machine you can directly convert a Docker image to a Singularity image using the docker2singularity image

```bash
docker run -v /var/run/docker.sock:/var/run/docker.sock \
    -v path/to/my/singularity/images:/output --privileged -t --rm \
    singularityware/docker2singularity ubuntu:16.04
```

• Creates a Singularity image from ubuntu:16.04 image on Docker Hub
• Singularity image stored at `path/to/my/singularity/images`
• You can also import directly from a Docker image living on your local machine
• docker2singularity is a Docker image that lives on Docker Hub and is programmed to convert a Docker image to a Singularity image
**TESTING A CONTAINER**

**singularity test ubuntu16.img**

- Runs recipe defined in `%test` section of definition file
- Verifies that application inside container is behaving as expected
- No root privilege required to run

Excerpt from example Singularity definition file.

```
. .
%test
  python3 -c "import numpy as np; np.test()"
. .
```
“Sniffing” an Image

**singularity inspect ubuntu16.img**

- Provides details about container
- Lists sections of definition file
- Only supported in Singularity 2.3+

Examples:

```
$ module load GCC Singularity
$ singularity inspect --test ubuntu16.img
$ singularity inspect --runscript ubuntu16.img
$ singularity inspect --deffile ubuntu16.img
$ singularity inspect --json ubuntu16.img
```
**EXPORTING AN IMAGE**

singularity export ubuntu16.img > ubuntu16.tar

- Create compressed version of image
- Advantageous for large images especially that you wish to share and distribute with collaborators
- Be patient…

Examples:

$ module load GCC Singularity
$ singularity export ubuntu16.img > ubuntu16.tar
$ singularity export ubuntu16.img | gzip -9 > ubuntu16.tar.gz
Good Use Cases for Singularity

- I need a different/newer version of a library that is not installed on the cluster (e.g. a newer GLIBC, or a super old piece of software needed for reproducibility reasons)
- I want to decouple my workflow from software available on the cluster (Singularity becomes the only dependency!) to reduce the impact of OS/software/library updates on the cluster
- I want to standardize my workflow across many different platforms
- I prefer a different “flavor” of Linux than CentOS / Red Hat
- Probably others…?
ACCRE Software Management – Best Practices

- When to use software from LMOD (module load pkg1)
  - Need results fast, relatively few dependencies, versions don’t seem to matter
  - MPI, multithreaded, and GPU apps more challenging to deploy from container
  - Using commercial software (e.g. Matlab, STATA, Intel compiler, etc.)

- When to install software locally (/home on cluster)
  - Software (or specific version) not available through LMOD, need results fast, relatively few dependencies

- When to install tools into a Singularity image
  - See reasons on previous slide
  - Need stability and reliability
Wrapping Up...

• Singularity docs are excellent: http://singularity.lbl.gov/user-guide

• Possible to wrap up MPI or GPU-based jobs in a Singularity container
  • GPU-enabled Tensorflow image built here:
    https://github.com/accre/singularity/tree/master/tensorflow-1.0-jupyter-gpu

• Containerizing your jobs on the cluster provides a number of benefits:
  • Decouples your application from software installed on cluster (less prone to failure)
  • Standard, multi-platform workflow
  • No performance penalty

• A few images (and their definition files) available at /scratch/singularity

• Follow our Singularity GitHub repo: https://github.com/accre/singularity