Containerization for R&D Applications (Part 2): Singularity

Andrew Monaghan
Andrew.Monaghan@colorado.edu

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

Please fill out the course evaluation survey:
http://tinyurl.com/curc-survey18

Outline

Part 1: Container fundamentals and Docker (this past Tue)

- Introduction to containers
- Docker commands and options
- Hands-on: Running Docker containers on your personal Machine
- Hands-on: Building Docker images
- Hands-on: Practical Application

Part 2: Containers for HPC with Singularity (12/2/21) – Today!

- Singularity commands and options
- Hands-on: Running containers
- Building containers
- Special cases: Running containers for MPI and GPU jobs

What is a container?

A container is a portable environment that packages some or all the following: an operating system, software, libraries, compilers, data and workflows. Containers enable:

- Mobility of Compute
- Reproducibility (software and data)
- User Freedom

Containerization software

- Docker
    - Part 1 - focus of tutorial given previously
    - Well established – largest user base
    - Has Docker Hub for container sharing
    - Problematic with HPC
- Singularity/Apptainer*
    - Part 2 - focus of this tutorial
    - Designed for HPC

***Note: Sylabs spun off the open-source version of Singularity to the Linux Foundation in November 2021. This open-source version has been renamed “Apptainer” to distinguish it from the commercial product, “Singularity” that will still be marketed by Sylabs.*

Why Singularity?

- Singularity is a comparably safe container solution for HPC
    - User is same inside/outside container
    - User cannot escalate permissions without administrative privilege
- Can support MPI and GPU resources on HPC (scaling)
- Can use HPC filesystems
- Supports the use of Docker containers
- Container is seen as a file, and can be operated on as a file

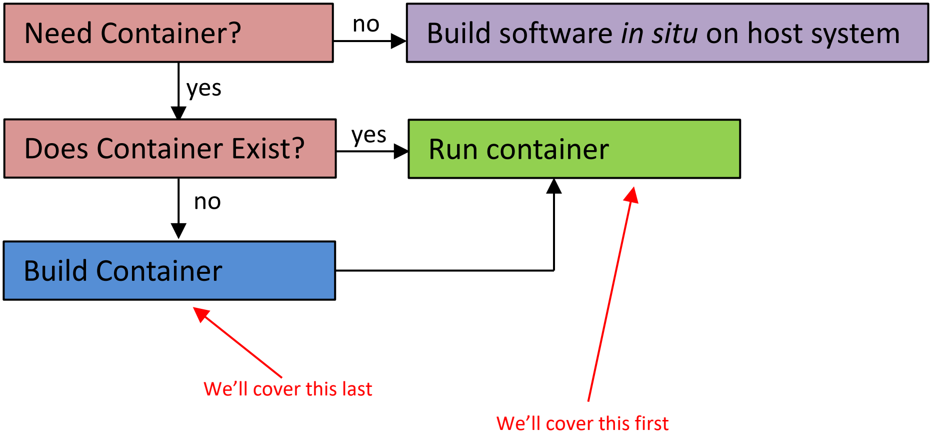
Singularity Overview

Singularity Workflow

Key Commands

Running Containers

The Singularity Workflow



Where do I find existing containers?

Docker Hub: https://hub.docker.com

Sylabs Library: https://cloud.sylabs.io/library

Tutorial on finding a running an existing Docker container:
https://www.chpc.utah.edu/documentation/software/singularity.php#exd

Key Singularity Commands

build: Build a container on your user endpoint or build environment

exec: Execute a command to your container

inspect: See labels, run and test scripts, and environment variables

pull: pull an image from Docker or Singularity Hub

run: Run your image as an executable

shell: Shell into your image

More: https://www.sylabs.io/guides/3.2/user-guide/cli.html

Running containers

Now, on ***any system*** with Singularity, even without administrative privilege, you can retrieve and use containers:

- Download a container from Singularity Hub or Docker Hub
  - `singularity pull shub://some_repo/some_image`
  - `singularity pull library://some_repo/some_image`
  - `singularity pull docker://some_repo/some_image`
- Run a container
  - `singularity run mycont.sif`
- Execute a specific program within a container
  - `singularity exec mycont.sif python myscript.py`
- “Shell” into a container to use or look around
  - `singularity shell mycont.sif`
- Inspect an image
  - `singularity inspect --runscript mycont.sif`

Running Containers

Navigate to the tutorial:

https://bit.ly/3lvwO4p

The tutorial can also be found here:
https://github.com/ResearchComputing/Containers_Fall_2021

Break!

Please fill out the course evaluation survey:
http://tinyurl.com/curc-survey18

Building containers

There are 3 ways to build a Singularity container

1. Build a container on a system on which you have administrative privilege (e.g., your laptop).
   - **Pros:** You can interactively develop the container.
   - **Cons:** Requires many GB of disk space, requires administrative privilege, must keep software up-to-date, container transfer speeds can be slow depending on personal network connection.

2. Build a container on Sylabs remote builder (this may change w/ “Apptainer”!)
   - **Pros:** Essentially zero disk space required on your system, doesn’t require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, final container is placed on local machine
   - **Cons:** Cannot interactively develop the container, ‘Freemium’ version limited

3. Build a docker container and “pull” it as a singularity container

What is a recipe?

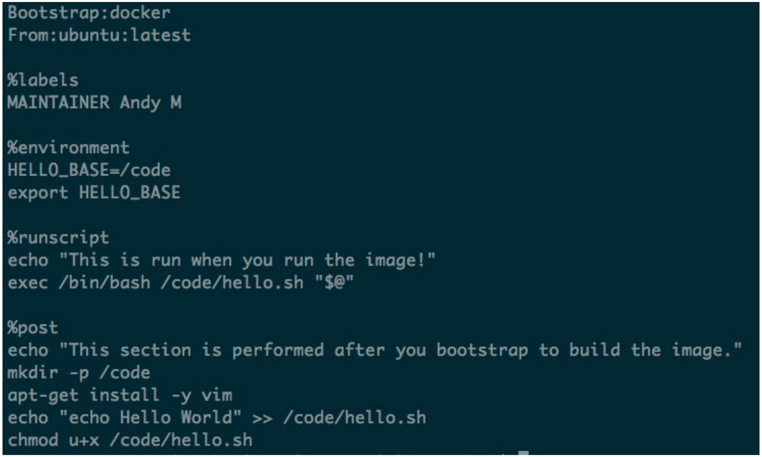
Header

Metadata

Runtime environment variables

Default program at runtime

Where software and directories are installed at buildtime

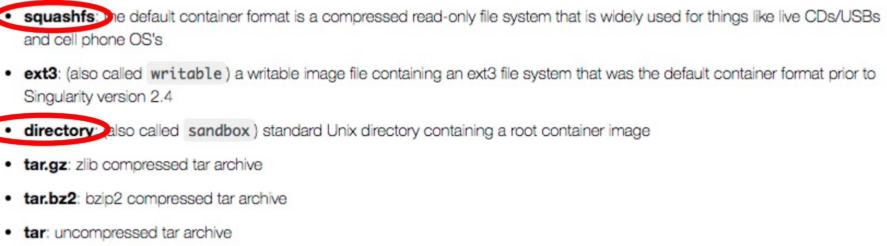


```
Bootstrap:docker
From:ubuntu:latest

%labels
MAINTAINER Andy M

%environment
HELLO_BASE=/code
export HELLO_BASE

%runscript
echo "This is run when you run the image!"
exec /bin/bash /code/hello.sh "$@"

%post
echo "This section is performed after you bootstrap to build the image."
mkdir -p /code
apt-get install -y vim
echo "echo Hello World" >> /code/hello.sh
chmod u+x /code/hello.sh
```

More: https://www.sylabs.io/guides/3.2/user-guide/definition_files.html

Container Formats

*.sif format and older *.simg format
(immutable final container format)



- **squashfs**: the default container format is a compressed read-only file system that is widely used for things like live CDs/USBs and cell phone OS's
- **ext3**: (also called `writable`) a writable image file containing an ext3 file system that was the default container format prior to Singularity version 2.4
- **directory**: (also called `sandbox`) standard Unix directory containing a root container image
- **tar.gz**: zlib compressed tar archive
- **tar.bz2**: bzip2 compressed tar archive
- **tar**: uncompressed tar archive

Writeable sandbox used for interactive container development

https://singularity.lbl.gov

1. Building a container interactively (demo)

- Bootstrap a base container (has OS you want, maybe other stuff too) into a sandbox:

```
sudo singularity build --sandbox mytest/ docker://alpine:latest
```

- Shell into the container and install what you need by trial and error:

```
sudo singularity shell --writable mytest/
```

  - [now do stuff in container; as you get it correct, add commands to Singularity recipe]

- Now finalize container.
  - You can either build squashfs image from sandbox:

    ```
    sudo singularity build mytest.sif mytest/
    ```

  - Or you can build squashfs image from recipe (best practice):

    ```
    sudo singularity build mytest.sif Singularity
    ```

2. Building a container with Sylabs Remote Builder (demo)

1. Get a token from Sylabs Cloud: https://cloud.sylabs.io/auth
    1. Login using your Github account
    2. Create a new token under "Keystore"
    3. Copy the token string (<your-token>)

2. Add the token on your host machine:
    1. `singularity remote login`
    2. Then paste your token when prompted.

3. Issue the command to build your container remotely
    1. `singularity build --remote myimage.sif Singularity`

4. …where "Singularity" is your definition file. If successful, the container will be placed in your working directory

Thank you!

Please fill out the survey:
http://tinyurl.com/curc-survey18

Contact information:
Andrew.Monaghan@Colorado.edu

Additional learning resources:

Slides and Examples from this course:
https://github.com/ResearchComputing/Containers_Fall_2021

Web resources:
https://www.sylabs.io/guides/3.6/user-guide/ *(user guide for Singularity)*
https://www.singularity-hub.org/ *(Singularity Hub)*

Extra slides

Hands-on Example

- Build a container on Singularity Hub

Notes: MPI and GPUs

Notes: MPI-enabled containers

HPC systems use low-latency interconnects ("fabric") to enable MPI to be efficiently implemented across nodes). In order to use a Singularity container with OpenMPI (or any MPI) on a Supercomputer there are two requirements:

1. The Singularity container needs to have the fabric libraries installed inside.
2. OpenMPI needs to be installed both inside and outside of the Singularity container. More specifically, the SAME version of OpenMPI needs to be installed inside and outside (at least very similar, you can sometimes have two different minor versions, ex: 2.1 and 2.0).

Running containers on GPUs

Syntax (after loading Singularity on a gpu node):

```
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu
python mytensorflow_script.py
```

With the ***--nv*** option the driver libraries do not have to be installed in the container. Instead, they are located on the host system (e.g., Teton) and then bind mounted into the container at runtime. This means you can run your container on a host with one version of the NVIDIA driver, and then move the same container to another host with a different version of the NVIDIA driver and both will work. (Assuming the CUDA version installed in your container is compatible with both drivers.)

The NVIDIA Driver version on a GPU node can be queried (when on that node) with the command `nvidia-smi`. See *https://docs.nvidia.com/deploy/cuda-compatibility/*

You can build a container by bootstrapping a base NVIDIA CUDA image (with a compatible CUDA version) from: *https://hub.docker.com/r/nvidia/cuda/*

NVIDIA has a tool for building gpu-enabled containers for both Singularity and Docker. See: *https://github.com/NVIDIA/hpc-container-maker*

Troubleshooting and Caveats

Troubleshooting (1)

Container/host environment conflicts

- Container problems are often linked with how the container "sees" the host system. Common issues:
    - The container doesn't have a bind point to a directory you need to read from / write to
    - The container will "see" python libraries installed in your home directory (and perhaps the same is true for R and other packages. If this happens, set the PYTHONPATH environment variable in your job script so that it points to the container paths first.
        - `export PYTHONPATH=<path-to-container-libs>:$PYTHONPATH`

- To diagnose the issues noted above, as well as others, "shelling in" to the container is a great way to see what's going on inside. Also, look in the singularity.conf file for system settings (can't modify).

Troubleshooting (2)

Failures during container pulls that are attributed (in the error messages) to *tar.gz files are often due to corrupt tar.gz files that are downloaded while the image is being built from layers. Removing the offending tar.gz file will often solve the problem.

When building ubuntu containers, failures during *%post* stage of container builds from a recipe file can often be remedied by starting the *%post* section with the command “`apt-get update`”. As a best practice, make sure you insert this line at the beginning of the *%post* section in all recipe files for ubuntu containers.

https://singularity.lbl.gov

Caveats (1)

We didn't cover overlays. These are additional images that are "laid" on top of existing images, enabling the user to modify a container environment without modifying the actual container. Useful because:

1. Overlay images enable users to modify a container environment even if they don't have root access (though changes disappear after session)
2. Root users can permanently modify overlay images without modifying the underlying image.
3. Overlays are a likely way to customize images for different HPC environments without changing the underlying images.
4. More on overlays: http://singularity.lbl.gov/docs-overlay

We didn't cover GPU usage with containers. See:
http://singularity.lbl.gov/archive/docs/v2-3/tutorial-gpu-drivers-open-mpi-mtls

Caveats (2): Moving containers

You've built your first container on your laptop. It is 3 Gigabytes. Now you want to move it to Summit to take advantage of the HPC resources. What's the best way?

Remember, containers are files, so you can transfer them to Summit just as you would a file:

- Command line utilities (scp, sftp)
- Globus (recommended)
- For more on data transfers to/from Summit: https://github.com/ResearchComputing/Research-Computing-User-Tutorials/wiki/Data-Transfers