COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Lecture 21
Containers in HPC

Michael Messina
Research Computing Specialist LSA Technology Services

11/18/2019
NERS 590-004

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Outline

- Containers
  - Docker
  - Singularity
  - Shifter
- Building Singularity containers
- How we use Singularity on Great Lakes
- Special Cases
  - MPI
  - GPU
- Evolving Features of Singularity
- Singularity container creation demo

Today's Learning Objectives

- Define containers and why we use them
- Learn how to build a singularity container
- Learn the specifics of our usage model in LSA
- Learn about methods for dealing with special cases
  - MPI
  - GPU computing
- Learn about the evolving features
  - Encrypted containers and container signing
  - Executable containers
  - Container distribution
- Show a demo of using Singularity

Motivation

Building large, complex software packages with many dependencies is difficult in an HPC environment. Unlike your local system, you have to be concerned with how software and dependencies you install impact others. You also need to create a package that others can build, test, and use.

Containers offer a way to wrangle these challenges while introducing a few of their own.

Containers in HPC

This presentation borrows profusely from the Singularity website at
https://sylabs.io
and the user guide in particular at:
https://sylabs.io/guides/3.5/user-guide/

What are Containers? Let’s ask google.

- Meh
  CIO.com: Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another.
- Good
  AWS: Containers provide a standard way to package your application's code, configurations, and dependencies into a single object.
- Better
  Docker: A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- Winner
  IBM.com: Containers are an executable unit of software in which application code is packaged, along with its libraries and dependencies, in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud.

Let’s break that down

Containers are an executable unit of software

- Containers are not virtual machines, vm’s contain their own kernel
- They are simply an application, containing almost everything but uses the hosting systems kernel

in which application code is packaged, along with its libraries and dependencies,

- The payload software
- Any dependencies up to and possibly including the OS in which to run it

in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud.

- With limitations

Container types

- **Docker**
    - **Used extensively with Kubernetes and other launcher**
    - **Used as a collection of layers of micro-services, especially in web applications**
    - **Left permission escalation protection up to the host implementation**
    - **Does have some major players distributing their applications as docker images**
    - **Docker hub**
- **Singularity**
    - **Designed for HPC at Lawrence Berkeley National Laboratory**
    - **Offers sandboxes**
    - **Permission escalation protection**
    - **Can convert docker images**
    - **Singularity hub**
- **Shifter**
    - **New type with many similarities to Singularity**

Why use containers?

- **Reproducibility -** your container can be locked away from system changes
- **Portability -** your container is running the same libraries regardless of the hosting system
- **Alternate OS -** you can build your software in whatever (kernel compatible) OS you choose
- **Obsolete OS -** you can run an outdated application in an appropriate OS
- **Distributable -** you can send someone a run ready package
- **Ease of building -** you can use standard packaging tools like yum and apt-get for dependencies. https://seissol.readthedocs.io/en/latest/compilation.html
- **Isolation** - install any dependencies your package needs without impacting other packages that are already installed

How to build a singularity container

Simple definition file (formerly called a recipe file)

```
BootStrap: library
From: ubuntu:16.04

%post
    apt-get -y update
    apt-get -y install fortune cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    fortune | cowsay | lolcat

%labels
     Author GodloveD
```

Complete definition file

```
Bootstrap: library
From: ubuntu:18.04
Stage: build

%setup
    touch /file1
    touch ${SINGULARITY_ROOTFS}/file2

%files
    /file1
    /file1 /opt

%environment
    export LISTEN_PORT=12345
    export LC_ALL=C

%post
    apt-get update && apt-get install -y netcat
    NOW=`date`
    echo "export NOW=\"${NOW}\"" >> $SINGULARITY_ENVIRONMENT

%runscript
    echo "Container was created $NOW"
    echo "Arguments received: $*"
    exec echo "$@"

%startscript
    nc -lp $LISTEN_PORT

%test
    grep -q NAME=\"Ubuntu\" /etc/os-release
    if [ $? -eq 0 ]; then
        echo "Container base is Ubuntu as expected."
    else
        echo "Container base is not Ubuntu."
    fi

%labels
    Author d@sylabs.io
    Version v0.0.1

%help
    Demo container using all supported sections.
```

Bootstrap Agents

Preferred

- library (images hosted on the Container Library)
- docker (images hosted on Docker Hub)
- shub (images hosted on Singularity Hub)
- oras (images from supporting OCI registries)
- scratch (a flexible option for building a container from scratch)

Other options

- localimage (images saved on your machine)
- yum (yum based systems such as CentOS and Scientific Linux)
- debootstrap (apt based systems such as Debian and Ubuntu)
- oci (bundle compliant with OCI Image Specification)
- oci-archive (tar files obeying the OCI Image Layout Specification)
- docker-daemon (images managed by the locally running docker daemon)
- docker-archive (archived docker images)
- arch (Arch Linux)
- busybox (BusyBox)
- zypper (zypper based systems such as Suse and OpenSuse)

LSA Containers on Great Lakes

- Singularity (3.4.1)
- Build host (lsa -sing-ubuntu.lsa.umich.edu)
- Ubuntu (16.04, 18.04, and 19.04)
- Bind mapped storage (/home and /scratch)
- Extensive use of sandboxes
- Modules (environment setup system)
- Bin scripts (scripts that call into the container)

Sandboxes

Singularity has a very useful feature for HPC and other complex software builds. This feature is called a sandbox. It is a fully writable version of the container that you can shell into and run commands.

You can build a sandbox from any singularity build type:

```
singularity build --sandbox simple.sb simple.def
```

Even another container or docker image

```
singularity build --sandbox trimtest.sb trimtest.simg
```

You can shell in make changes then turn the sandbox back into a squashed image

```
singularity shell -w trimtest.sb --login
singularity build trimtest2.simg trimtest.sb
```

Sandboxes

We use sandboxes in a way that would make container purists angry.

Here is an example of a HPC software installation instructions
https://seissol.readthedocs.io/en/latest/compilation.html

Trying to script up that list of commands into the post install script and sort out errors during a container build would be an arduous process.

Sandboxes allow us to install them by hand as if we were installing to our own private machine.

The /.singularity/env/90-environment.sh file.

Environment modules

Lmod environment modules

https://lmod.readthedocs.io/en/latest/

Lmod is a Lua based module system that easily handles the MODULEPATH Hierarchical problem. Environment Modules provide a convenient way to dynamically change the users' environment through modulefiles. This includes easily adding or removing directories to the PATH environment variable. Modulefiles for Library packages provide environment variables that specify where the library and header files can be found.

Example commands

module load intel/18.0.5

module list

module purge

module save

Bin scripts

Short bash scripts which define the singularity call for the user rather than making the user learn the command needed to run in the container. The scripts appear as standard executables to the user and they have no idea whether they are running a standard executable or a container. This simplifies commands and allows users to seamlessly follow standard tutorials and practices.

Rather than the user typing:

singularity exec <path to container> command args

They type:

command args

```
#!/bin/bash
exec /opt/singularity/3.4.1/bin/singularity exec
/sw/lsa/centos7/gubbins/2.3.1/gubbins.simg run_gubbins "$@"
```

Special case MPI

MPI is a special case container, since it is tied to hardware features, network interfaces and protocols, that require different handling than other software. This is one place where portability is not guaranteed.

For MPI you need to install a compatible version of MPI both inside the container and on the hosting system. Usually you will be matching the MPI for your build to whatever you expect on the hosting system.

You not only need to have compatible versions, you also may need to know the build options to insure that MPI goes over the appropriate interface.

Special case GPU computing

GPU/Cuda/ROCm is also special case container, since it is tied to hardware features and protocols of the graphics cards. It requires different handling than other software. This is another place where portability is not guaranteed.

For Nvidia Cuda you used to need to install a compatible version inside the container and on the hosting system. This has changed in singularity 3.x. You can invoke the singularity --nv flag. This mounts the hosting systems cuda install into the container and uses it.

For AMD ROCm there is a similar option --gpu=rocm.

Evolving features of Singularity

- Encrypted containers
- Container signing
- Cloud libraries
- Executable containers
- Container distribution

Singularity Demo

"Hold on to your butts…"
- Arnold