Containers

https://bit.ly/2DkV9Uz

Breakout Session
Statewide Users Group Conference
April 18, 2019
Zhi-Qiang (Wesley) You
Scientific Applications Engineer for OSC

- I am not a container expert
- Joined Scientific Application Group @ OSC in August 2018
- Previously worked as Research Scientist @ Q-Chem, Inc.
A Overview of Today’s Breakout

• What are containers?
• Using Singularity
What are containers
Virtualization

• A virtualization solution which can provide dense, scalable, and secure user environments

• Two major classes: container-based virtualization and hypervisor-based virtualization (virtual machine)

Containers

- Container-based virtualization is a lightweight virtualization approach using the host kernel (operating-system) to run multiple virtual environments.

Containers

- Containers are instances or standard Linux processes launched from *container images* via *container engine*
- A container image is a series of data layers on top of a base image

Advantages of Containers

• **Performance**: Near-native application performance
• **Freedom**: Bring your own software environment
• **Reproducibility**: Package complex software applications into easy to manage, verifiable software units
• **Compatibility**: Built on open standards available in all major Linux distributions
• **Portability**: Build once, run (almost) anywhere

https://www.sdsc.edu/assets/docs/events/introduction-to-singularity.pdf
Limitations of Containers

• **Architecture-dependent**: Always limited by CPU architecture (x86 64, ARM) and binary format (ELF)

• **Portability**: Requires glibc and kernel compatibility between host and container; also requires any other kernel-user space API compatibility (e.g., OFED/IB, NVIDIA/GPUs)

• **Filesystem isolation**: filesystem paths are (mostly) different when viewed inside and outside container

https://www.sdsc.edu/assets/docs/events/introduction-to-singularity.pdf
Containers for Research Computing

• Effective at distributing applications and their dependencies

• **Docker** is common container engine in use today; commonly used to containerize enterprise applications on cloud services

• **Singularity** is an emerging leader in research computing containers, and has a different approach to *security model* from Docker

[https://www.docker.com](https://www.docker.com)

[https://www.sylabs.io](https://www.sylabs.io)
Singularity for HPC

- “Untrusted users can run untrusted containers safely”
- Singularity’s design forces a user to have the same UID and GID context inside and outside of the container
- Singularity mounts the container file system with the nosuid flag
- Supports HPC hardware: Infiniband, GPUs
- Supports HPC applications: MPI
Docker for HPC

• **Shifter** at NERSC allows an HPC system to efficiently and safely allow end-users to run a docker image

• In late 2018, Docker v19.03 emerged with support for Docker daemon running as a non-root user

• Charliecloud?
Container Workshop

- 5/14 1-4pm @ BALE theater
- Container overview
- Build your own software environments
- Reproducible data and publication
Using Singularity
Common Singularity Use Cases

- Running applications that require newer system libraries or have specific OS requirements
- Running applications that have complicated dependencies
- Running pre-built container images
Download Pre-built Images

- Docker
  - Docker Hub
  - BioContainer: Bioinformatics containers

singularity pull docker://ubuntu:18.04
Download Images

- Singularity
  - Stanford University Singularity Hub: Singularity 2.5
  - Sylabs Container Library: Singularity 3.x only; support container search, signing and validation via Singularity client

```
# Singularity Hub
singularity pull shub://vsoch/hello-world

# Container Library
singularity search lolcow
singularity pull library://sylabsed/examples/lolcow
```
Manage Images

- Singularity container image is a **single image file**
  - The default file name is “application_tag.sif”
  - If the image file is large and you need to run it multiple times, keep it on /fs/scratch or /fs/project or copy it to $TMPDIR
Manage Images

• **Cache folders**: make downloading images for build and pull faster and less redundant

```bash
-- using cache command
$ singularity cache list -a

-- the default cache is in your home directory
$ ls ~/.singularity/cache/*

-- clean cached images
$ singularity cache clean -a
```

```bash
#PBS -l nodes=1:ppn=1
cd $TMPDIR
singularity pull library://sylabsed/examples/lolcow
(what if the image from source is modified: singularity checks sha256 checksum)
```
Inspect Images

• View the labels within your container

```bash
$singularity inspect lolcow_latest.sif
{
    "org.label-schema.build-date": "Tuesday_5_March_2019_7:55:21_-05",
    "org.label-schema.schema-version": "1.0",
    "org.label-schema.usage.singularity.deffile.bootstrap": "library",
    "org.label-schema.usage.singularity.deffile.from": "ubuntu:latest",
    "org.label-schema.usage.singularity.version": "3.1.0-rc4"
}
```
Interact with Images

- The **shell** command allows you to spawn a new shell within your container

```bash
$singularity shell lolcow_latest.sif

Singularity lolcow_latest.sif:~> cat /etc/os-release
Singularity lolcow_latest.sif:~> ls /fs
Singularity lolcow_latest.sif:~> ls -d $TMPDIR
```
Interact with Images

• The **exec** command allows you to execute an arbitrary command inside/outside a container

```bash
$ singularity exec lolcow_latest.sif cowsay moo
```

_____
< moo >
-----
\ ^___^ \
(oo)\_______ \
(__)
\-\-\-\-
\ | \-w | \\
\ | \ | \\

Interact with Images

- The **run** command allows you to run a container as an application through **runscripts** within the container.

  
  ```
  $ singularity run lolcow_latest.sif
  $ ./lolcow_latest.sif
  ```

- The **inspect** command with the flag **--runscript** or **-r** will show you the runscript.

  ```
  $ singularity inspect -r lolcow_latest.sif
  ```
Interact with Images (cont.)

```bash
#!/bin/sh

fortune | cowsay | lolcat
```

-- shell equivalent

```
$ singularity shell lolcow_latest.sif
Singularity lolcow_latest.sif:~> fortune | cowsay | lolcat
```

-- exec equivalent

```
$ singularity exec lolcow_latest.sif sh -c 'fortune | cowsay | lolcat'
```
User-defined Bind Paths

- When Singularity ‘swaps’ the host operating system for the one inside your container, the host file systems becomes inaccessible except for the system default bind points e.g. $HOME, tmp

```
$ singularity pull library://zqyou/default/sug_bind_example
$ ./sug_bind_example_latest.sif
  tee: /tmp2/sug_bind_example: No such file or directory
  Hello Word 27405

$ mkdir -p $HOME/tmp
$ singularity --bind $HOME/tmp:/tmp2 sug_bind_example_latest.sif
  Hello Word 27631

$ cat $HOME/tmp/sug_bind_example
  Hello Word 27631
```
MPI-based Singularity Containers

• Use same Message Passing Interface (MPI) distribution and version within container as would be used outside the container

• If using Infiniband (IB), install same OFED drivers and libraries inside the container as used on underlying HPC hardware

https://www.sdsc.edu/assets/docs/events/introduction-to-singularity.pdf
MPI Development Example

- Code development with openmpi-2.1.6 on CentOS 7

BootStrap: library
From: centos:7

%post
yum groupinstall -y "Development Tools"
yum install -y wget
wget https://www.open-mpi.org/software/ompi/v2.1/downloads/openmpi-2.1.6.tar.bz2
tar xf openmpi-2.1.6.tar.bz2
cd openmpi-2.1.6
./configure --prefix=/usr/local (change path in build time? use SINGULARITYENV_XXX)
make
make install
make clean
/usr/local/bin/mpicc examples/ring.c.c -o ring
cp ./ring /usr/bin/ring

%runscript
exec /usr/bin/ring
MPI Development Example

- Run the MPI program within the container by calling the ‘mpiexec’ on the host

```bash
$ singularity pull library://zqyou/default/sug_openmpi_ring
$ module load openmpi/2.1.6
$ mpiexec -np 2 ./sug_openmpi_ring_latest.sif
Process 0 sending 10 to 1, tag 201 (2 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
...
Process 0 exiting
Process 1 exiting
```

GPU Computing

- In Singularity v3.0+ the **--nv** option will look for NVIDIA libraries on the host system and automatically bind mount them to the container so that GPUs work seamlessly.

```bash
$ module load cuda
$ singularity exec --nv tensorflow-gpu.sif python mnist_deep.py
```
Code Development w/o Write Permission

- C++17 sample code

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>

int main()
{
    std::vector<int> a{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    std::vector<int> b(5);

    std::sample(a.begin(), a.end(),
                b.begin(), b.size(),
                std::mt19937{std::random_device{}});

    std::cout << "Support C++17: " << b.at(2) << std::endl;
    return 0;
}
```
Code Development w/o Write Permission

• Compile and run the sample with g++ support C++17

```
$ singularity pull docker://gcc:7.3
$ singularity exec ./gcc_7.3.sif g++ -std=c++17 cxx17_sample.C
$ module load gnu/4.8.5
$ ./a.out
./a.out: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.21' not found (required by ./a.out)
./a.out: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.20' not found (required by ./a.out)

$ singularity exec ./gcc_7.3.sif ./a.out
Support C++17: 7
```
Question?

- OSC Help