Singularity

What?

Why?

How?

Predrag Lazic
University of Missouri
10/2020
Singularity

Introduction

Basic Singularity image - sandbox

Usage on cluster

Parallelization – MPI

Other aspects GPU, overlay

Conclusion
Introduction

VIRTUALIZATION

What is it?
Introduction

VIRTUALIZATION

What is it?

VIRTUAL MACHINE

Virtualizes hardware

CONTAINER (singularity, docker, etc.)

Virtualizes operating system (OS)
Introduction

VIRTUALIZATION

What is it?

VIRTUAL MACHINE

Virtualizes hardware

CONTAINER (singularity, docker, etc.)

Virtualizes operating system (OS)

Operating System – OS
(kernel)

Hardware
VIRTUAL MACHINE

- App1
- Bins/libs

Guest OS

kernel

VM hypervisor (Vbox)

Virtual hardware

HOST MACHINE

- App1
- Bins/libs

Operating System - OS

HOST MACHINE

- App1
- Bins/libs

- App2
- Bins/libs

- App3
- Bins/libs

Hardware

CONTAINER

- App1
- Bins/libs

- App2
- Bins/libs

Container engine
VIRTUALIZATION

- Transferability
- Flexibility
- Reproducibility
- Simplicity (for users and sysadmins)
- Scalability
VIRTUALIZATION

- Transferability
- Flexibility
- Reproducibility
- Simplicity *(for users and sysadmins)*
- Scalability

VIRTUAL MACHINE

+ **flexibility**
  *Combine different OS*

- **Overhead**
  *(CPU, memory, startup time – guest OS)*

- **No native HW support**
  *(GPU, Infiniband, Lustre)*
VIRTUALIZATION

- Transferability
- Flexibility
- Reproducibility
- Simplicity (for users and sysadmins)
- Scalability

VIRTUAL MACHINE

+ flexibility

   Combine different OS

- Overhead

   (CPU, memory, startup time – guest OS)

- No native HW support

   (GPU, Infiniband, Lustre)

CONTAINER

+ portability

+ modular

+ lightweight

+ native HW support

HPC

VM hypervisor (Vbox)

Multiple containers can work together
Different containers *(container images)*

process file

Docker

Must be run as *root*

Still popular in IT industry
Different containers *(container images)*

<table>
<thead>
<tr>
<th>Docker</th>
<th>Singularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must be run as <em>root</em></td>
<td>Still popular in IT industry</td>
</tr>
<tr>
<td></td>
<td>The <em>user</em> running singularity image on the host is also the <em>user</em> in the container</td>
</tr>
<tr>
<td></td>
<td>Designed for traditional HPC</td>
</tr>
</tbody>
</table>

No privilege escalation
Different containers *(container images)*

**Docker**
- Must be run as *root*
- Still popular in IT industry

**Singularity**
- The *user* running singularity image on the host is also the *user* in the container
- Designed for traditional **HPC**
- No privilege escalation

Images can be *easily* converted!
Real world examples of container usage
(in u nutshell avoiding (library) dependency hell)

1) User needs some software that is not available on the cluster but is a standard part of (some) Linux distro

2) There is a popular commercial software – installation for CentOS 7 but your cluster is Running on Ubuntu

3) The scientific software that you need is already available as a container!
   https://github.com/sestaton/Transposome

4) A good theoretical physicist kindly made his source code available – it is simple to compile you JUST need additional 20 libraries and it should compile without a glitch

5) The list is not exhaustive

RECOMMENDED USAGE
With Docker, you can create a container to run Transposome with the following command:

docker run -it --name transposome-con sestaton/transposome

Singularity exec docker://sestaton/transposome transposome --config config_file.yml

Geant4
Enough talk!

Let's get our hands dirty ......
Enough talk!

Let's get our hands dirty ...... in the sandbox!
Once we have everything we want we will make a *recipe file* for automatic singularity image build.

For some reason ... we want to have *inkscape* app on cluster.
Create a basic container – **CentOS7**

```
singularity build --sandbox sing_inkscape docker://centos:7
```

Enter the shell in newly created

```
singularity shell --writable sing_inkscape
```

Try to install Inkscape – you must be root!

```
sudo singularity shell --writable sing_inkscape
```

Now install Inkscape

Build a singularity image file

```
sudo singularity build sing_inkscape.sif sing_inkscape
```

Copy the image file to cluster and test it!
Singularity and OpenMPI via Infiniband

CONTAINER

OpenMPI + compiler

HOST

OpenMPI + compiler

For compiling our code

For running our compiled code
Singularity and OpenMPI via Infiniband

Application Binary Interface (ABI) matching is required

More or less it means you need exact same versions of the OpenMPI and Compiler in the container and on the HOST
Singularity and OpenMPI via Infiniband

Install OpenMPI in container – the same version as on the destination HPC cluster (Lewis)

We go back into our singularity image
sudo singularity shell --writable sing_inkscape

wget openmpi 4.0.2
yum install gcc...
yum install ucx....
yum install gcc-gfortran
untar openmpi
configure with ucx and fortran
./configure --prefix=/usr/local --enable-mpi-fortran=yes --with-ucx
make
make install

Compile your MPI code
mpifort ring.f Producing a.out binary file
Singularity and OpenMPI via Infiniband

Move container image on the cluster and test it

Before that 2 important options

Binding directories from the HOST to the CONTAINER

```bash
export SINGULARITY_BINDPATH="/host_dir1,/host_dir2:/container_dir1"
```

Telling container about LD_LIBRARY_PATH

```bash
export SINGULARITYENV_LD_LIBRARY_PATH="/host_dir1/lib,container_dir1/lib"
```
1) MPI-run the code in the container (with MPI from the container!)

HOST (Lewis)

Other_libs

CONTAINER

Mpirun

a.out

Other_libs
1) MPI-run the code in the container (with MPI from the container!)

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

2) Copy the code (a.out) from the singularity image and run it **completely** out of the Container – i.e. on the host!

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

copy

Other_libs

1) MPI-run the code in the container (with MPI from the container!)

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

2) Copy the code (a.out) from the singularity image and run it **completely** out of the Container – i.e. on the host!

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

3) Bind libraries from the host to the container and run host MPI with the code from singularity container

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs
1) MPI-run the code in the container (with MPI from the container!)

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

2) Copy the code (a.out) from the singularity image and run it **completely** out of the Container – i.e. on the host!

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs

3) Bind libraries from the host to the container and run host MPI with the code from singularity container

HOST (Lewis)
Mpirun
Other_libs

CONTAINER
Mpirun
a.out
Other_libs
Making a recipe file

```
BootStrap: docker
From: ubuntu:18.04

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

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

%labels
Author Pawsey Supercomputing Centre Version v0.0.1

%help
This is a demo container used to illustrate a def file.

%runscript
fortune | cowsay | lolcat
```

If you do not have your own Linux system (root access) to build images there is an option for remote build.

https://cloud.sylabs.io/builder

Also fakeroot on HPC system

---

Build singularity image

```
sudo singularity build lolcow.sif lolcow.def
```

Run it!

```
./lolcow.sif
```

```
singularity run lolcow.sif
```

```
singularity exec lolcow.sif command (fortune)
```

```
singularity shell lolcow.sif
```
Containers and GPU

Install CUDA libs and desired compiler in the container
compile the code - we **do not need** GPU driver in the container!

When running on the HOST, CONTAINER needs to know about GPU driver libs

```
singularity exec --nv image.sif command_running_gpu_code
```
Writable containers – benefits of overlay

```
singularity exec --writable-tmpfs image.sif command_to_run
```

small temporary amount of space to write (~MB)

```
singularity exec -B ~/my_run:/run image.sif /run/command_to_run
```

Binding HOST directory and writing into it (permanent)

```
singularity exec --overlay my_overlay image.sif command_to_run
```

permanent

*my_overlay* is a **FILE** on the HOST system produced *(with some extra steps)* by:

```
dd if=/dev/zero of=my_overlay count=200 bs=1M
```

200 MB in this case

Huge benefit if user writes large number of files in the container (they get written to the overlay file) so the HOST system deals only with a single file (overlay)

There is no metadata for large number of files which puts huge strain on HOST’s LUSTRE filesystem.
Efficiency of Containers

From:
Exploring the performance of Singularity for HPC Computing Scenarios
Conclusion

Containers make life much easier for users and sysadmins

Simple codes, GUI, GPU relatively simple to handle

However, real HPC usage MPI + Infiniband requires almost as much effort \textit{(if not more)} to make it in the container than on the HOST system.

This could be remedied if each HPC system would provide its own basic singularity recipe file for MPI usage.
Thank you!

Thanks to:

Christina Roberts
Asif Ahamed
Brian Marxkors
Ashkan Mirzaee
Nate Roettgen
Buddy Scharfenberg
Useful links:

Singularity workshop from SC19
https://pawseysc.github.io/sc19-containers/

Example of QuantumEspresso (MPI) for singularity
https://medium.com/@uniquelock/singularity-containers-at-iaus-hpc-center-quantumespresso-56e51308d221

Exploring the performance of Singularity for HPC Computing Scenarios (paper)