Singularity in HPC

Grigory Shamov– Jun 2020

Westgrid Research Computing SummerSchool

Outline of the Workshop

(Quick?)start

- Logistics: how to connect to Cedar and submit jobs
- Get a simple Singularity container and run it!

Background info about Containers in HPC

- What are software containers , and what are their use cases?

Basic Singularity usage

- Running serial, GPU and MPI jobs as containers
- Getting containers from existing software repositories
- Building containers from recipes

Advanced Singularity usage

- More on Building containers from recipes
    - (environment, keying/encryption, remote building services)
- Overlays and ephemeral temporary directories
- Running Singularity Containers as services

Materials I have used

Official Sylabs Singularity Documentation

1. https://sylabs.io/guides/3.5/user-guide/
2. https://cloud.sylabs.io/
3. https://sylabs.io/guides/3.5/admin-guide/

SC19 Singularity Tutorial (very good!) from Pawsey HPC centre, Australia

1. https://pawseysc.github.io/sc19-containers

NVIDIAs NGC cloud repository and its documentation

1. https://ngc.nvidia.com/catalog/all
2. https://www.nvidia.com/en-us/gpu-cloud/

DockerHub repository

1. https://hub.docker.com/

Getting started with the course

Accounts, systems and reservations

- Reservations are prepared on Cedar (HPC)
- On Cedar users w.o Computecanada accounts can use wg-guestNN.
- UManitoba users with WG accounts can use Grex as well.

Connecting to Cedar and submitting jobs

- **ssh -Y your_username@cedar.computecanada.ca**
- **salloc --account=def-training-wa --mem-per-cpu=8Gb **
  --time=0-2:00:00 --reservation=wgsummer-wr_cpu

Course materials

- The workshop involves pulling a lot of data from Internet!
- If fails, try a local copy of images under ***/scratch/gshamov/wg-sing-ws***
- Some Examples and scripts are also there.

Singularity quick start

Get and run a simple "hello world" container

- The "Lolcow" container by Singularity developers.

module load singularity/3.5

singularity run docker://godlovedc/lolcow

Get familiar with HPC environment and

- Submit an interactive job, login nodes might be busy and/or resource limited.

**salloc --mem-per-cpu=4gb --time=0-2:00:00 **

--cpus-per-task=2 --account= def-training-wa

- Get access to the singularity command **module load singularity/3.5** and see its options with "**singularity help**"
- Try printing something else than the fortune and cow. How about "Hello world" from the Lolcow container?

Dealing with complexity of the software

- Controlling software environments
  - Porting effort is often needed between development and production environments
  - Want to distribute software (in a portable way).
- Building software packages predictably
  - “infrastructure as a code”
- High Performance Computing:
  - Reproducibility, Mobility of computing?
- One of the approaches is to isolate/containerize software/apps together with their dependencies

Glossary: Operating systems

Operating system - All the software that let you interact with a computer, run applications, UI, etc. ; consists of the "kernel" and "userland" parts.

Kernel - Central piece of software that manages hardware and provides resources (CPU, IO, memory, devices, filesystems) to the processes it is running.

Users - Linux separates access for end users, system service accounts and root. Fine grained control (sudo, capabilities).

Program and Process - process is an instance of a running program with resources allocated by kernel . Processes associated with users.

Daemon process - a process that runs long time, "in the background"

Filesystem – an organized collection of files. Under UNIX/Linux, single / hierarchy exists and filesystems on different devices are "mounted" somewhere.

What a software task needs to run ?

- Hardware resources (as provided by OS kernel)
- Kernel/Systems functions and C runtime library (libc)
- "Userland" operation system
  - systems libraries , scripts , services
- Application libraries it depends on
  - linear algebra, file formats, parallel computing and accelerators computing libraries
- Dynamic languages are the worst because they are dynamic
  - python, R, perl, Java, might depend on each other as part of a single research pipeline

- Can we encapsulate the above dependencies for the task?

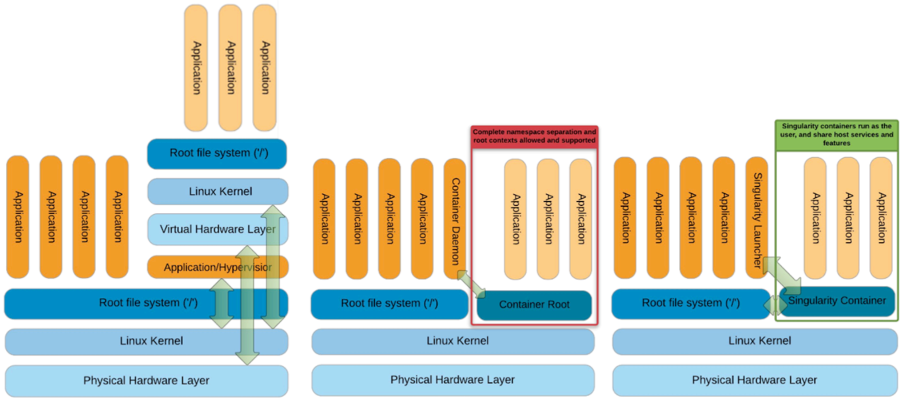
Containers vs Virtual Machines

VMs offer true isolation via virtualized Hardware; maximal flexibility at some performance and space cost

- Can run any combination of host and guest OS
- You can improve performance at cost of losing flexibility and isolation

Containers are an OS-level mechanism of isolating userland parts of OS together with a given application. Tied to the OS, less flexible.

- "*Chroot* on steroids"; ***namespaces*** for processes provided by the kernel.
- Security issues of sharing the same kernel, privilege escalation
- Almost no performance overhead



General VM eg ESXi | General Container eg Docker | HPC Container Singularity

source: Greg Kurtzer keynote at HPC Advisory Council 2017 @ Stanford

WESTGRID | University of Manitoba

Docker. https://docker.com

Developed as Containers platform for services/daemons.

- Enterprise computing, Microservices approach
- Isolation of the (micro)services
- Composable containers, version control
- Load balancers
- Orchestration tools (Google Kubernetes, Docker Swarm, ..)
- Many containers per node, oversubscription

- Runs as root or a service user; cgroups for resource management

- Uses commodity, Internet network stack extensively

Very popular with software developers, DevOps; thanks to Recipes and the huge registry at DockerHub (https://hub.docker.com)
Docker quickly made its way to Research Computing software dev.!

HPC use case for containers

- In the HPC world jobs are ran in the Batch Mode
  - The "queue, start, run, end" lifecycle of a computing task
  - Maximal utilization of hardware
- Often long running compute tasks with large state (memory, data on disk).
- Often a whole node or many nodes per job, statically allocated.
  - "Worst case" scenarios, often no resource oversubscription possible.
- Close access to hardware for speed; specialized interconnects, RDMA, direct GPU access, zero-copy IPC
- Shared systems:
  - Pretend to be a single large machine with a Scheduler, shared network FS
  - The HPC users are connecting directly on the machine. Privilege escalation is a concern.

Can we use Docker on HPC systems?

The short answer is often no.

- Security model: no *root* or *sudo* is given on shared machines
- Cgroups resource management will conflict with HPCs RM

But what if users, or developers, really want it? Containers for HPC:

- **Shifter**; **Singularity**; **CharlieCloud; Sarus**
  - Either based on Docker or can convert from Docker image format
  - running containers in user space, as a user.
  - Zero performance overhead for either of them (an SC19 paper).

Singularity

- Created first at LBNL, now developed by a company (SyLabs) https://sylabs.io/
- The goals: mobility of compute and reproducible research with Containers.
- Developed for HPC use case: runs as a regular user, can access shared filesystems. Interoperable with Docker; can run services as well.
- As of now likely the most popular container engine for HPC that is supported
  - on ComputeCanada's HPC machines.
  - by NVIDIA GPU software environment
  - on CVMFs collaborative environments (ATLAS, OSG)
  - on some public clouds (MS Azure batch etc.)

Back to the Lolcow demo!

- What is the relation/difference between ”container” and “image”?
- Singularity command offers the following actions commands
  - **singularity run**
  - **singularity exec**
  - **singularity shell**
  - (also, “**singularity instance**” group of commands)
- What is the difference between “**run**” and “**exec**”?
- We can inspect images with “**singularity inspect**”
  - By default, to check the metadata, tags etc.
  - Can also see the recipe (for native Sing. Images) and run script with
    - **singularity inspect --deffile** and singularity **inspect --runscript**.
- Docker images are portable “layers” while Singularity image is a single file

Basic usage : R containers

- R is a popular dynamic scientific language with many packages and several repositories (CRAN, Bioconductor, etc.). Some packages are hard to maintain and install so a natural target for containerization.
- The Rocker Project maintains a number of docker://rocker/ images . On dockerHub:
    - docker://rocker/r:latest
    - docker://rocker/tidyland:latest
    - docker://rocker/rstudio:latest
- It supports singularity: https://www.rocker-project.org/use/singularity/
- DEMO: lets pull some containers and run R examples.

  session-info.R and **R-benchmark-25.R**
- DEMO: lets try R INLA tutorial? **isbaspde.R** from :
    - http://www.r-inla.org/examples/tutorials/spde-from-the-isba-bulletin

Pulling the containers

Lets start with pulling an R image. (Things to consider on CC systems: network and FS performance; memory and threads to pack/unpack the container). **singularity pull** is the command. https://hub.docker.org is the Registry.

singularity pull docker://rocker/tidyland:latest

- The Images are cached , under $HOME/singularity

- Docker layers are cached too.

- We can control the cache location with SINGULARITY_CACHE environment (and SINGULARITY_TMP); /scratch/$USER might be better if $HOME is full.

- **singularity cache {list|clean}** commands are used to manage the cache.

- **singularity inspect** shows the image's metadata ; **--runscript**, **--deffile** options for SIF images are useful.

Repositories to pull containers from:

Public containers repositories from where to "*pull*" or "*build*" containers use the following URI

- First, the DockerHub public registry. **docker://** ; a RedHat repo quay.io/ has some science stuff; NVIDIA NGC has docker images.
- SyLabs cloud library, native SIF images: l**ibrary://**
- SingularityHub, native SIF images: **shub://**

Private Docker repositorie qequire authentication. **singularity pull --docker-login docker://your-private-repo/container:tag**

CVMFS distributions! They distribute container images in an unpacked directory format. CVMFS handling various optimizations and caching of this format.

- OpenScienceGrid: https://opensciencegrid.org/docs/worker-node/install-singularity/

Running containers, access to FS

If you have *a)* Singularity container image, and *b)* Singularity runtime installed, you can run your app in the container in them.

./my-app [options] my-input.dat

[singularity command] [singularity-options] ./container.sif [options] input.dat

For example, **singularity run ./lolcow.sif** (or just **./lolcow.sif**)

Or **singularity exec lolcow.sif echo "Hello, World!"**

Access to filesystems using --bind | -B options to action commands:

- Bind-mount is a Linux kernel mechanism. **-B outside:inside**
- Some paths are mounted by default. */home , /tmp, $(pwd)*
- Sysadmin can configure more/less paths by default
- You can prevent mounting the default paths by **--containall**

singularity exec -B /scratch:/scratch tidyverse-latest.sif Rscript session-info.R

Running containers, access to environment variables

Environment variables are key-value pairs that are passed to a running process by OS. Some of them are very influential system-wide (PATH, LD_LIBRARY_PATH, CPATH, etc.). Some are used by a particular code only (PETSC_DIR).

- Singularity inherits environment from the build/pull time
- Building recipe might define some env vars explicitly in the %environment section
- At run time, passing variables to container can be done by prefixing their names with ***SINGULARITYENV_***
- A flag **--cleanenv** prevents form inheriting the environment.

Exercise: do “singularity exec ” for the command “env” with and without the --cleanenv flag.   More information:

https://sylabs.io/guides/3.5/user-guide/environment_and_metadata.html

Building your own containers

In case of R-INLA we could not find a suitable Docker or Singularity image, so we had to build it. The process is like for Docker, based on a text file "recipe" that will define the container image.

The command is "**singularity build {target} {source}**". Note that source might be:

- a Singularity recipe, which is a text file like for Docker
- a container repository URI (then "build" is like "pull")
- another Singularity container image.

Singularity 3.x has two main image formats: the compressed image (SIF) and the Sandbox directory. Building a new container can be interactive process with **shell –writable and sandbox format** very useful to fix things; however, it is a good practice to capture everything in the (final) recipe.

Building a new container often would require encapsulation of OS userland parts that require root access and root ownership: so in many cases the *local build needs sudo!*

Singularity recipe examples

1. Specifies from where to **Bootstrap from** something (OS repo, docker, etc.)

2. Modifies the container in **%post**, copies **%files**

3. Sets the **%environment**

4. Defines entry point in **%runscript**

Sylabs documentation:

https://sylabs.io/guides/3.5/user-guide/definition_files.html

More examples on Github
https://github.com/sylabs/singularity/blob/master/examples

```
Bootstrap: docker
From:  rocker/r-ver:latest
%post
	apt-get update -y
	apt-get install -y libssl-dev libsasl2-dev\
jags   autoconf automake curl wget\
 libudunits2-dev bash libicu-dev libeigen3-dev \
gcc-multilib g++-multilib
	# generic R packages
	R -e "install.packages('ggplot2')"
	# skipped a few packages #
	R -e "install.packages('R2jags')"
	#R2OpenBUGS
	wget http://pj.freefaculty.org/Ubuntu/15.04
	tar xzf openbugs_3.2.3.orig.tar.gz
	cd openbugs-3.2.3
	./configure
	make && make check && make install
	R -e "install.packages('R2OpenBUGS')"
```

Remote building services: no sudo!

The original build service: V. Sochat's **SingularityHUB**

- Link your Github repo with recipe to https://singularity-hub.org ; wait for it to build; setup auto rebuild hooks.

- Pull the container from anywhere like so (putting your URI of course): **singularity pull shub://vsoch/hello-world**

- Right now locked down due to an abuse by a malicious user; limits are set for the number of downloads per client.

The new **SyLabs Cloud** service fro Singularity 3.x

- Register with an identity provider (Google, FB, MS, Github) at https://cloud.sylabs.io ; Get an access token and do "**singularity remote login**" to enter it.

- Use **singularity build --remote** CLI option from a local Singularity installation or deposit a recipe using the SyLabs Cloud web interface. EXERCISE: try remote building the *Lolcow*.

- Documentation: https://sylabs.io/guides/3.5/user-guide/endpoint.html

Running containers on GPUs

Singularity supports Nvidia GPUs through bind-mounding the GPU drivers and base CUDA libraries. The **--nv** flag does it transparently to the user. For example,

singularity exec --nv -B /scratch:/mnt tensorflow.sif python my-tf.py

•NVidia NGC provides readily made containers for a large number of HPC apps.

- https://ngc.nvidia.com/catalog/containers

EXERCISE1 : lets run GAMESS-US binary from Nvidia NGC in an interactive SLURM job.

- Do salloc command to get a GPU compute node (--gres=gpu:p100:1 or –gres=gpu:v100:1 as described here: https://docs.computecanada.ca/wiki/Using_GPUs_with_Slurm#On_Cedar

**salloc --gres=gpu:p100:1 --cpus-per-task=8 --mem=40Gb --time=0-2:00:00 **
--account=def-training-wa --reservation=wgsummer-wr_gpu

- Do pull the GAMESS-US container and run an example following instructions here: https://ngc.nvidia.com/catalog/containers/hpc:gamess

Running containers on GPUs

•NVidia NGC provides readily made containers for a large number of HPC apps.

- https://ngc.nvidia.com/catalog/containers

EXERCISE 2a : lets run the single node NAMD binary from Nvidia NGC in an interactive SLURM job.

1. Do the salloc command to get a GPU compute node (--gres=gpu:p100:1 or –gres=gpu:v100:1 as described here: https://docs.computecanada.ca/wiki/Using_GPUs_with_Slurm#On_Cedar

**salloc --gres=gpu:p100:1 --cpus-per-task=8 --mem=40Gb --time=0-2:00:00 **
--account=def-training-wa --reservation=wgsummer-wr_gpu

Pull the NAMD container and run an example following instructions here: https://ngc.nvidia.com/catalog/containers/hpc:namd

EXERCISE 2b: Lets run a multimode NAMD binary as a batch SLURM job, using the multimode NAMD image and the example SLURM job script as provided.

Singularity and MPI applications

MPI is a standard; for message passing interface. MPI comes with several implementations (OpebnMPI, MPICH, IntelMPI, PlatformMPI, Cray MPI, ..)

MPI libraries on HPC systems usually are using a high-performance interconnect, RDMA etc. which rely on variety of kernel device drivers and low level userland libraries, so they are hard to containerize. Thus no generic --mpi flag is easy to implement for the containers.

The Sylabs documentation page (https://sylabs.io/guides/3.5/user-guide/mpi.html) covers it in more detail.

Intel MPI provides a knowledge base page on using Singularity ()

Less of a use case? Most MPI software in HPC world comes as sources.

However, MPI+X model might be useful (try GAMESS-US with MPI with GPU, or LAMMPS+GPU, etc.).

Singularity and MPI applications

The following modes can be thought of::

1. MPI inside of the container (less interesting, won't work across the nodes); the code is built singularity exec my.simg mpiexec hello.mpi

2. The Hybrid mode: same (or similar) MPI inside and outside of the container. The software is built against the container's MPI. mpiexec singularity exec hello.mpi

3. The Bind mode: host's MPI libraries and drivers are mounted into the container; the applicaton has to be built against the . Mpiexec singularity exec –B /paths/to/mpi hello.mpi

EXERCISE: build a hybrid or bind-mode MPI application and benchmark the performance.

TODO

Advanced topics: writable overlays

If you really a writable container layer, there is a new development feature, writable overlays. The overlay “layers” on top of the (immutable) SIF image and allows for changes without rebuilding the image. The overlay can be

- A sandbox directory.
- A writable ext3 filesystem image. To be created with mkfs.ext3 first.
- A writable ext3 image embedded in the SIF file.

The command : **singularity shell --overlay name_of_overlay name_of_image.sif**

Unfortunately, there are too many limitations to use it on current ComputeCanada systems. It either needs *sudo*, or needs userIDs less than 65535 (use the **id** command to see yours) or needs a newer Linux kernel than available on CentOS 7.

•Documented at SyLabs site:

https://sylabs.io/guides/3.5/user-guide/persistent_overlays.html

Advanced topics: services

If you really need to run a daemon / service with Singularity as if it was Docker.

- The command is **singularity instance** (**singularity instance start http.sif my-web**)
- Instead of %runscript, the %startscript section is used to define the containers' action.
- It can use its own Cgroups mechanism (--cgroups flags) to manage the resources.
- It can run in a privileged mode, as root, with fine-grained capabilities. (--addcap)
- Documented at SyLabs site:
    - https://sylabs.io/guides/3.5/user-guide/running_services.html

A service example, Rstudio

- An example of service can be Viz/GUI on a compute node. It is just an example and not necessarily the recommended way to run Rstudio on Cedar!
- The workflow would more or less follow the ComputeCanada's doc here:
  - https://docs.computecanada.ca/wiki/Jupyter#Connecting_to_Jupyter_Notebook and https://www.rocker-project.org/use/singularity/
  - Start the server container instance (Rstudio server) inside a SLURM job, on a compute node
  - Set up an SSH tunnel to the compute node
  - Connect via the SSH tunnel using your browser at localhost:port using the credentials from the SLURM job
  - When work done, cancel the SLURM job on the compute node
- To start the server, we'd need a container image built and started.
  - Actually, a batch container will do as well, but our purpose is to demonstrate the services.
  - A container based on rocker/tydyr and defining the port and password as described on the Rocker pages above: I called it rocker-server.sif. It has the **%startscript** as follows:

```
%startscript
  export R_PORT=${R_PORT:-"8787" R_ADDRESS=${R_ADDRESS:-"0.0.0.0"}
   rserver --www-port $R_PORT --www-address $R_ADDRESS --auth-none=0 \
          --auth-pam-helper-path=pam-helper
```

A service example, Rstudio

- Get an interactive job, on Cedar:

salloc --mem=4gb --cpus-per-task=1 --account=def-training-wa

- Start the container instance (named myserver) defining the port (pick one above 1000) and a password. The user will be current user.

**R_PORT=8765 PASSWORD=dodo singularity instance start **

-c rstudio-server.sif myserver

Check the instance status; you can also execute commands there:

singularity instance list

singularity exec instance://myserver echo $USER

Make the tunnel, on your client machine. This one's for Mac

**ssh -L 8765:*cdr767*.int.cedar.computecanada.ca:8765 **
***gshamov*@cedar.computecanada.ca**

Point a browser to **http://localhost:8765** and use the user/password. When done, terminate the instance with **singularity instance stop myserver**

Questions?