Singularity: Containers for High-Performance Computing

Grigory Shamov
Nov 21, 2017
Outline

● Software and High Performance Computing:
  ○ Installation/Maintenance of the HPC Software stack
  ○ Why containers and what are they

● What is Singularity and how it can be useful
  ○ Version history
  ○ Workflows and use cases

● Using Singularity
  ○ On Selected use cases and examples.
  ○ New features of Singularity 2.4
Resources

Everything is on a Hub: Github, DockerHub, SingularityHub.

- **Singularity examples:**
  [https://github.com/singularityware/singularity/tree/master/examples](https://github.com/singularityware/singularity/tree/master/examples)
- **Its documentation:** [http://singularity.lbl.gov/user-guide](http://singularity.lbl.gov/user-guide)
- **DockerHub:** [https://hub.docker.com/explore/](https://hub.docker.com/explore/)
- **SingularityHub:** [https://www.singularity-hub.org/](https://www.singularity-hub.org/)

Access to Singularity:

- **Westgrid users:** use Grex
  - `ssh your_wg_user@grex.westgrid.ca`
  - `module load singularity/2.3.2` or `module load singularity/2.4.0`
- **ComputeCanada new systems:** use Cedar or Graham
  - `ssh your_cc_user@cedar.computecanada.ca` or `ssh your_cc_user@graham.computecanada.ca`
  - Singularity will be updated soon; `module load singularity` gives 2.3.1
- **Some things need S. on a local machine or VM:** install it from GitHub.
HPC usage model / architecture

- Bare-metal performance, fast interconnect
- Clusters, under control of Resource Manager/ Scheduler
- Scalable, Parallel POSIX FileSystems (Lustre, GPFS, etc.)
- Centralized HPC software stack
- “Untrusted” users directly access the system; flat UID space.

Some of the issues with HPC:

- “Stable” (that is, old, and server-centered) OS Linux distribution
- Centralized HPC Software stack is great, but less than flexible;
- Little provision for multi-tenancy (FileSystems, user access)
HPC Software Stack

Operating system
• kernel
• glibc
• OS packages (Linux distro-specific: CentOS RPMs on CC machines; APT/DEBs on Ubuntu, Pacman on Arch etc.)
• Sysadmin-installed

HPC Applications
• MPI, BLAS/LAPACK, HDF5/NetCDF: many dependencies there!
• Trying to support software for each and every scientific area and community
• Many versions, Modules/LMOD to manage it
• ComputeCanada has a distributed one (CVMFS)
• Installed (mainly) by HPC Analysts (On CC machines, Nix can provide OS-level packages)

HPC applications have dynamic dependencies on the other stack components
• ldd
• On most Linux distros, at least glibc dependency is present.

Dynamic languages (Python, Perl, R) have their own package management systems
Scientific Software, problems

Dependencies on particular Linux distribution
- Developers like recent desktop Linux (Ubuntu)
- HEPhysics likes recent CentOS / ScientificLinux

“Release early, release often”
- Quick access to new software
- “Raw” software depending on errors in particular dev. env.

Dynamic languages have complex dependencies
- Python, R, Julia: rPython, rpy2, rJava
- Anaconda/Miniconda will bring its own everything (MPICH)

Installation and maintenance of large HPC software stack is a lot of work!
- Modules work, but to a point
- Virtualenv/Wheels works, to a point
- Software pipelines / workflows is a nightmare

Reproducibility: is a concern

Mobility: Would be nice to move software easier between the HPC systems

Empower users, less wait for HPC analysts doing things.
We’d want to be able to pack the software environment, and freeze it, and move it, and perhaps to automate/define/document via scripts. At the same time, staying in “traditional” HPC context.

**Virtual Machines:**

- Overhead; harder to use in HPC context
- Root access to global/parallel filesystems
- Better isolation/security; but less control for HPC

**Containers:**

- “Chroot on steroids”
- OS-level isolation; uses the Hosts kernel
- Namespaces (PID, Devices, Network, etc.)
- Cgroups for resource control
- More of security risk, root access escalation
Docker

Docker is the most popular container product

• Good for (micro)services; Enterprise/Developer workflow
• Efficient layering model, minimal images.
• Network “virtualization”
• Uses cgroups for resource management
• Popular with developers, including scientific ones
• Very many images present on DockerHUB.

Docker-based HPC projects, some of
• Shifter (docker frontend, HPC backend)
• Adaprive Docker support in Torque 6 / Moab 9
• Limiting user privileges: Udocker (Guillimin/CQ, but not only!), Socker (UOslo).

Somehow, Docker is not yet an easy fit for HPC!
Again, HPC conditions

HPC usage model / architecture

- Bare-metal performance, fast interconnect
- Clusters, under control of Resource Manager/ Scheduler ↔ (Docker’s cgroups)
- Scalable, Parallel POSIX FileSystems (Lustre, GPFS, etc.)
- Centralized HPC software stack
- “Untrusted” users directly access the system; flat UID space. ↔ (Docker’s root escalation)

Some of the issues with HPC:

- “Stable” (that is, old, and server's) OS Linux distribution ↔ (Docker’s need for of recent kernel)
- Centralized HPC Software stack is great, but less than flexible;
- Little provision for multi-tenancy (FileSystems, user access) ↔ (Docker’s root escalation)

While root escalation for Docker is just a potential possibility, it still needs recent kernel/namespace and proper configuration to avoid in, which limits the Mobility.
Singularity

Singularity!

- Developed from scratch for HPC by **Greg Kurtzer, LBNL**.
- Container is ran as user, can mount cluster’s local and parallel FS
- Does not require special plumbing, can run under any Scheduler/RM.
- Can import Docker containers into own format(s).

**History**: exactly two years now!

Version 1 of Nov 2015: detect and pack application dependencies.
- A single file / chroot; small setuid routine to run it as the user. (runC, Flatpak, Snappy : not so HPC!)

Version 2 to 2.3: packing Linux OS images
- Supports bind mounts to cluster
- Requires root access to build the images; but not to run them.
- Converts Docker containers to Singularity ones

Version 2.4 (most recent, Oct. 2017 ): Lots of new features! (Getting more like Docker).
- We will be using 2.3, and then covering 2.4 new features.
Singularity workflow (2.3-ish)

Demo: run containers; demo: Tensorflow.
Singularity 2.x definition/recipe

Examples at: Git!  https://github.com/singularityware/singularity/tree/master/examples
From legacy/2.2/centos.def:

=========

BootStrap: yum  ← The header; specifies OS bootstrap source and method
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum

%runscript  ← The %runscript determines what happens on ‘singularity run’
   echo "This is what happens when you run the container..."

%post  ← The %post is just BASH to execute after bootstrap within
   echo "Hello from inside the container"
   yum -y install vim-minimal
   mkdir -p /global/scratch

=======

$ singularity create –size 1024 centos-7.img ; sudo singularity bootstrap centos-7.img centos.def
$ singularity run centos-7.img ; singularity shell centos-7.img ; singularity exec centos-7.img bash
Singularity 2.x definition/recipe

Examples at: Git! [https://github.com/singularityware/singularity/tree/master/examples](https://github.com/singularityware/singularity/tree/master/examples)
From legacy/2.2/ubuntu.def:

=========
BootStrap: debootstrap ← The header; specifies OS bootstrap source and method
OSVersion: trusty
MirrorURL: [http://us.archive.ubuntu.com/ubuntu/](http://us.archive.ubuntu.com/ubuntu/)

%runscript ← The %runscript determines what happens on ‘singularity run’
echo "This is what happens when you run the container..."

%post ← The %post is just BASH to execute after bootstrap within
   echo "Hello from inside the container"
sed -i 's/$/ universe/' /etc/apt/sources.list
   apt-get –y update; apt-get –y –force-yes install vim
=========

$ singularity create –size 1024 ubuntu.img ; sudo singularity bootstrap ubuntu.img ubuntu.def
$ singularity run ubuntu.img ; singularity shell ubuntu.img ; singularity exec ubuntu.img bash
Singularity workflow / Hubs

- An example of Tensorflow: pulling containers from DockerHUB. Uses a cache directory.

- "Do not try in on $HOME!": Grex $HOME quota is small, so images can easily exhaust it.

```bash
export SINGULARITY_CACHEDIR=/global/scratch/$USER/.cache/ or
export SINGULARITY_CACHEDIR=/dev/shm/$USER
```

Pulling Images from DockerHub do not require root, nor they need to be create’d! Bootstrap the image with a DockerHub URL with

```bash
$ singularity pull docker://tensorflow/tensorflow:latest
```

Use cases: Python/Tensorflow:
- To Exec or Run the container?
- Do we need to pull or can Exec/Run right away?
- Containers from SingularityHUB can be accessed same way: `shub://`
Single-node, serial or SMP/threaded jobs are straightforward. Can we do MPI jobs within one node? Can we go across-the-nodes parallel?

Choices for where MPI is:
• Within the container; needs a way to kickstart
• Outside, MPI running containerized processes

The outside way expected to work in OpenMPI 2.1.x; also MPICH based ones (IntelMPI, MVAPICH2).

$ mpirun -np 4 singularity exec … [various options] … container.img ./a.out

Containers might need access to interconnect which limits mobility and is difficult to average users.

https://github.com/singularityware/singularity/issues/876
Singularity 2.4, new features

• SquashFS immutable images and/or sandboxes instead of ext3 chroot

• Layered image format, “docker/style” is possible

• Bootstrap is “depreciated” (but still there) in favor of the new build command

• Breaks image format compatibility with 2.3.

• New container standard proposed: %app definitions, per-app dependencies

• Easier to run services within the container, network virtualization
  %startscript
  singularity instance.start
  singularity instance.list

• Automated build service on SingularityHub 2.0, integrated with Git:
  https://www.singularity-hub.org/collections/280
  https://github.com/gshamov/shub-test/blob/master/Singularity
Singularity 2.4, the new build

`singularity build [flags] target source`

- Can bootstrap ext3 images as before with `--writable`
- Can pull dockerHub and singularity hub as before.
- Also build can be used to convert between different formats of local containers as well (squashfs to/from ext3, etc.)
Singularity workflow, 2.4

Interactive Development
- `sudo singularity build --sandbox tmpdir/ Singularity`
- `sudo singularity build --writable container.img Singularity`

BUILD ENVIRONMENT

Build from Recipe
- `sudo singularity build container.img Singularity`

Build from Singularity
- `sudo singularity build container.img shub://vsoch/hello-world`

Build from Docker
- `sudo singularity build container.img docker://ubuntu`

Container Execution
- `singularity run container.img`
- `singularity shell container.img`
- `singularity exec container.img ...

Reproducible Sharing
- `singularity pull shub://...
- `singularity pull docker://...

PRODUCTION ENVIRONMENT

* Docker construction from layers not guaranteed to replicate between pulls

Source: Singularity 2.4 documentation
Thank you for your attention!

Questions?