Singularity Containers

Containers for ...
HPC, analytics, machine learning, reproducible and trusted computing

http://sylabs.io
GREGORY M. KURTZER

• CEO, SyLabs Inc.
• Senior Architect, RStor Inc.

• Previous:
  • HPC Systems Architect, LBNL

• Open Source Work:
  • Founder and project lead: Warewulf (2001)
  • Founder: Centos Linux (2003)
  • Founder and project lead: Singularity (2015)
“Singularity + Warewulf + Centos: Winning combo!”
DAVE GODLOVE

• Senior Software Engineer, Rstor Inc.

• Previous:
  • Computational Biologist, NIH High Performance Computing Center
  • Postdoctoral Fellow, National Institute of Mental Health
Introduction To Containers and Singularity
CONTAINERS 101

- Containers are encapsulations of operating system environments
- Includes all applications, libraries, configs and dependencies
- Workflows are completely self contained and portable
- Easy to share, distribute, validate and reproduce

- Multiple implementations exist (e.g. Docker and Singularity)
- Singularity differentiates itself in its architecture and container format

- Containers share the host kernel
- Isolation is managed by the host kernel
- As efficient and performant as running applications directly on host
• Applications running within a container will always be “closer” to the physical hardware
  • Notice how close to native a container behaves
• Applications running through a virtual machine will always have multiple levels of indirection
• The container’s proximity to the physical hardware equates to less overhead, higher performance and lower latency
WHAT MAKES SINGULARITY SO SPECIAL

• Built specifically to support HPC/Science: Singularity was built by demand, requests, threats and bribes by researchers, scientists, and computational users

• Single file based container format: verifiable via checksum and cryptographic signatures enabling reproducible and validated software environments during runtime

• Extreme mobility: using standard tools (rsync, scp, GridFTP, NFS, etc.)

• Controls compliant: images can be easily archived and managed as any other data

• Compatible: with complicated architectures (e.g. HPC, Machine Learning, Cloud, etc.)

• Security model: designed to support untrusted users running untrusted containers
“Singularity is the best option among the big three considerations for HPC”
Reproducibility and Portability
Without having access to an identical source, environments must be recreated.

Documentation, recipes, source code and data might be able to get pretty close, but is it identical?

Is it close enough?
IS THIS “GOOD ENOUGH”?

WHAT IS YOUR REPRODUCIBILITY PLAN?
BIT FOR BIT SOFTWARE REPRODUCIBILITY

SHA: 5f09a35a642a68c467bf230f5e5ea3218e4177a0

SHA: 5f09a35a642a68c467bf230f5e5ea3218e4177a0
One must **squash layers**, breaking upstream validation, and/or restore to a compatible registry service to reassemble and use the layers. **Not ideal for many scientific use cases!**

Container is a single file, easy to deal with and use.
SCP, SFTP, GridFTP/Globus, Rsync, NFS, Lustre, Object Stores, etc..

Single file containers means that containers are easy to manage

Single file containers means that containers are easy to manage

Singularity enables extreme mobility
“Singularity is a fabulous tool for providing forward and backward software compatibility on clusters and for reproducibility”
• **Cryptographic signatures and verification**: The Singularity image format supports both region checksums as well as signing for your verification pleasures.

• **Faster access to container meta-data**: Container meta-data is now part of the image file but outside the container.

• **Support for multiple system partitions**: A single container image can contain multiple container regions and/or a writable overlay.

• **Support for Checkpoint Restart**: Internal support for checkpoint-restarting for mobility of state.
SIF EXAMPLE: TRUSTED, SIGNED CONTAINERS

Cryptographically Signed

Global Header
Descriptors
Recipe Definition
Labels
Environment
Immutable Runtime Container Image

Signature Block
HPC COMPATIBLE SECURITY MODEL

• **Base security problem:** Untrusted users running untrusted containers

• **Limit user’s potential security contexts:** We can not allow users to escalate to root, even in containers that they control (and know the root password to)

• **Allow user’s access to data they own:** And limit access to data that they don’t own

• **Rules, limitations, boundaries:** When running in default privilege mode, the Singularity config file must be root owned

• **Strive for transparency:** On production supported kernels (e.g. Red Hat), container related system calls require privilege so we must be transparent and mitigate risks openly.
HPC COMPATIBLE ARCHITECTURE

- **Contained process execution just like native:** You can execute a contained program as if it were installed on the host and as a child process with direct lineage to it’s caller.

- **MPI support:** MPI jobs are also easily supported using a hybrid model

- **GPU:** Users are themselves within a container, and thanks to the Singularity security model, they can not escalate. This means we can share the GPU device into the container.

- **Resource managers:** Container processes are decedents of the RM (rather then a root owned container daemon which the RM has no direct control of)
DEEPER LOOK INTO RESOURCE MANAGEMENT

- RM does not speak to container daemon directly thus the user must control the container daemon
- Container daemon runs the jobs on the user’s behalf
- This is obviously dangerous and circumvents RM control and context
- Better to keep "program / workflow" within RM context
- This is the Singularity design

HPC Resource Manager
Execution daemon
(on compute node)

User’s shell / Batch Script

Singularity

Program / Workflow

User Privilege and PID Context

Container Daemon runs as root

Contained Program / Workflow
HPC COMPATIBLE ARCHITECTURE (CONT)

• **Contained process execution just like native:** You can execute a contained program as if it were installed on the host and as a child process with direct lineage to it’s caller.

• **MPI support:** MPI jobs are also easily supported using a hybrid model

• **GPU:** Users are themselves within a container, and thanks to the Singularity security model, they can not escalate. This means we can share the GPU device into the container.

• **Resource managers:** Container processes are decedents of the RM (rather then a root owned container daemon which the RM has no direct control of)

• **Single file container images:** This image format is highly optimized for parallel FS
Objectives:
1. Measure scaling of python startup and import speed with increasing numbers of concurrent python interpreters
2. Compare scaling of a standard python installation with an identical containerized installation (Singularity).

Note: Underlying file system is NFS, max jobs was 5120 over 320 nodes, graph is logarithmic on both axis.
INVOCATION EXAMPLE

1. Singularity is invoked
2. Parses the user’s environment and evaluates options at the command line
3. Checks the container image (SIF) and host’s capabilities
4. Creates necessary namespaces and sets up isolation
5. Mounts container image and sets up virtual root FS and binds within namespaces
6. Singularity exec()s itself out of existence, and attaches namespaces to foreground process
“Singularity allowed us to use software that was otherwise impossible to install under SL6, such as TensorFlow”
Use Cases
- Nextflow is a workflow management language for data-driven computational pipelines
- Nextflow uses Singularity to deploy large-scale distributed scientific workflows
- Commonly used in genomics pipelines
- Supports both HPC cluster and cloud based resources in a portable manner
- Used by:
  - Center for Genomic Regulation (CRG)
  - Pasteur Institute (France)
  - SciLifeLab (Sweden)
  - Sanger Institute (UK)
The NIH uses Singularity to provide programs like TensorFlow and OpenCV3 which are difficult or impossible to run with their current operating system. With Singularity they can create "portable reproducible data analysis pipelines". Singularity allows the NIH to provide this functionality to their users in a secure environment. The systemadmins found it easy and intuitive to use Singularity. Some applications have been installed into Singularity containers and used as standalone programs via environment modules for the users (the users don’t even know they are running within a container!)
• Among standard HPC use cases…
• Researchers are using iPython Notebooks via JupyterHub
• iPython JupyterHub kernels were deployed in Singularity containers
• Once the container is deployed via JupyterHub, the job runs within the container while maintaining access to local node resources
• This is a multi-user environment so Docker is a non-starter
• ALICE jobs are packaged into Singularity containers
• Jobs are executed via Singularity through a modified SLURM script
• At any given moment in time, there are about 2000 Singularity containers active on the system

GSI Green Cube
6 stories tall
30,000 sqft
12 MegaWatts
PUE = 1.07 (world record)
The OSG uses Singularity to provide a consistent runtime environment across heterogeneous resources worldwide.

- Container images are distributed via CVMFS to all sites.
- About half a million jobs are run through Singularity per day.
Titan @ Oak Ridge Leadership Computing Facility

Adam Simpson (front) and Matt Belhorn (back), high-performance computing user support specialists at the OLCF, use the Singularity application to develop containers that will allow newer systems to run deep learning packages.

https://www.olcf.ornl.gov/2017/05/09/containers-provide-access-to-deep-learning-frameworks/
Tutorial:
https://singularity-tutorial.github.io/
• Virtual container boot within instances
• Evolving signed containers
• Optional non-SetUID execution modes (Linux capabilities)
• OCI (Open Container Initiative) compliance
• Native Kubernetes support
• Performance profiling of contained applications
• Checkpoint / Restart of containers
• OS X and Windows support
SINGULARITY: CONTRIBUTORS AND THANKS!
Singularity Containers

Containers for ...
HPC, analytics, machine learning, reproducible and trusted computing

http://sylabs.io