Singularity (v2.2.1)

CESGA

Table of Contents

Introduction . . . 1
- The Singularity Process Flow . . . 1
- Images . . . 1
- Resources sharing . . . 2

Installation . . . 4

Usage . . . 4
- Workflow . . . 5
- Container management commands . . . 6
- Container usage commands . . . 9

Singularity in FinisTerrae II . . . 11
- Load Singularity . . . 11
- Run Singularity . . . 11
- Run Singularity with OpenMPI . . . 12

References . . . 12

Introduction

The goal of Singularity is to run an application within a contained environment such as it was not contained. Thus there is a balance between what to separate and what not to separate. At present the virtualized namespaces are process, mount points, and certain parts of the contained file system.

On the other hand, Singularity does not support user escalation or context changes, nor does it have a root owned daemon process managing the container namespaces. It also exec's the process work-flow inside the container and seamlessly redirects all IO in and out of the container directly between the environments. This makes doing things like MPI, X11 forwarding, and other kinds of work tasks trivial for Singularity.

The Singularity Process Flow

When executing container commands, the Singularity process flow can be generalized as follows:

1. Singularity application is invoked
2. Global options are parsed and activated
3. The Singularity command (subcommand) process is activated
4. Subcommand options are parsed
5. The appropriate sanity checks are made
6. Environment variables are set
7. The Singularity Execution binary is called (sexec)
8. Sexec determines if it is running privileged and calls the SUID code if necessary
9. Namespaces are created depending on configuration and process requirements
10. The Singularity image is checked, parsed, and mounted in the CLONE_NEWNS namespace
11. Bind mount points are setup
12. The namespace CLONE_FS is used to virtualize a new root file system
13. Singularity calls execvp() and Singularity process itself is replaced by the process inside the container
14. When the process inside the container exists, all namespaces collapse with that process, leaving a clean system

All of the above steps take approximately 15-25 thousandths of a second to run, which is fast enough to seem instantaneous.

Images

Singularity **images** are single files which physically contains the **container**. The effect of all files existing virtually within a single image greatly simplifies sharing, copying, branching, and other management tasks.

You do not need admin/sudo to use Singularity containers. You do however need admin/root access to install Singularity and to build/manage your containers and images, but to use the containers you do not need any additional privileges to run programs within it.

Because Singularity is based on container principals, when an application is run from within a Singularity container its default view of the file system is different from how it is on the host system. This is what allows the environment to be portable. This means that root (`/`) inside the container is different from the host!

Singularity automatically tries to resolve directory mounts such that things will just work and be portable with whatever environment you are running on. This means that `/tmp` and `/var/tmp` are automatically shared into the container as is `/home`. Additionally, if you are in a current directory that is not a system directory, Singularity will also try to bind that to your container.

NOTE	There is a caveat in that a directory must already exist within your container to serve as a mount point. If that directory does not exist, Singularity will not create it for you! You must do that.

Resources sharing

Singularity does no network isolation because it is designed to run like any other application on the system. It has all of the same networking privileges as any program running as that user

Singularity also allows you to leverage the resources of whatever host you are on. This includes HPC interconnects, resource managers, file systems, GPUs and/or accelerators, etc. Singularity does this by enabling several key facets:

- Encapsulation of the environment
- Containers are image based
- No user contextual changes or root escalation allowed
- No root owned daemon processes

MPI

Singularity utilizes a hybrid MPI container approach, this means that MPI must exist both inside and outside the container.

There are some important considerations to work with MPI:

- OpenMPI must be newer of equal to the version inside the container.
- To support infiniband, the container must support it.
- To support PMI, the container must support it.
- Very little (if any) performance penalty has been observed.
    - MPI/Singularity invocation pathway: **
        1. From shell (or resource manager) mpirun gets called

2. mpirun forks and exec orte daemon
3. Orted process creates PMI
4. Orted forks == to the number of process per node requested
5. Orted children exec to original command passed to mpirun (Singularity)
6. Each Singularity execs the command passed inside the given container
7. Each MPI program links in the dynamic Open MPI libraries (ldd)
8. Open MPI libraries continue to open the non-ldd shared libraries (dlopen)
9. Open MPI libraries connect back to original orted via PMI
10. All non-shared memory communication occurs through the PMI and then to local interfaces (e.g. InfiniBand)

- OpenMPI compatibility **

What are supported Open MPI Version(s)? To achieve proper containerized Open MPI support, you should use Open MPI version 2.1. There are however three caveats:

- Open MPI 1.10.x may work but we expect you will need exactly matching version of PMI and Open MPI on both host and container (the 2.1 series should relax this requirement)
- Open MPI 2.1.0 has a bug affecting compilation of libraries for some interfaces (particularly Mellanox interfaces using libmxm are known to fail). If your in this situation you should use the master branch of Open MPI rather than the release.
- Using Open MPI 2.1 does not magically allow your container to connect to networking fabric libraries in the host. If your cluster has, for example, an infiniband network you still need to install OFED libraries into the container. Alternatively you could bind mount both Open MPI and networking libraries into the container, but this could run afoul of glib compatibility issues (its generally OK if the container glibc is more recent than the host, but not the other way around)

GPUs

Device nodes are passed through into container, but Cuda libraries must be aligned with kernel drivers (similar to OFED).

To avoid the issue with drivers alignment, workarounds exists:

1. The host installs Cuda/Nvidia libraries to a directory.
2. That directory is configured as a `bind point`.
3. The library path is added to all container's environments using the environment variable (`LD_LIBRARY_PATH`).

Glibc

There is also some level of `glibc` forward compatibility that must be taken into consideration for any container system. For example, I can take a Centos-5 container and run it on Centos-7, but I can not take a Centos-7 container and run it on Centos-5.

WARNING	If you require kernel dependent features, a container platform is probably not the right solution for you.

Installation

```
git clone https://github.com/singularityware/singularity.git
cd singularity
./autogen.sh
./configure --prefix=/usr/local
make
sudo make install
```

Usage

```
USAGE:

    singularity [global options...] <command> [command options...] ...

GLOBAL OPTIONS:

    -d --debug    Print debugging information
    -h --help     Display usage summary
    -q --quiet    Only print errors
       --version  Show application version
    -v --verbose  Increase verbosity +1
    -x --sh-debug Print shell wrapper debugging information

GENERAL COMMANDS:

    help          Show additional help for a command

CONTAINER USAGE COMMANDS:

    exec          Execute a command within container
    run           Launch a runscript within container
    shell         Run a Bourne shell within container
    test          Execute any test code defined within container

CONTAINER MANAGEMENT COMMANDS (requires root):

    bootstrap     Bootstrap a new Singularity image from scratch
    copy          Copy files from your host into the container
    create        Create a new container image
    expand        Grow the container image
    export        Export the contents of a container via a tar pipe
    import        Import/add container contents via a tar pipe
    mount         Mount a Singularity container image
```

For any additional help or support visit the Singularity website: http://singularity.lbl.gov/

Workflow

```
 Root/Superuser                Regular user
+--------------+              +-----------+
| * Create     | Share image  | * Shell   |
| * Bootstrap  | ===========> | * Exec    |
| * Modify     |              | * Run     |
+--------------+              +-----------+
```

Container management commands

Singularity container images must be built and configured on a host where you have root access. Once the container image has been configured it can be used on a system where you do not have root access as long as Singularity has been installed there.

NOTE	You can create an image in a physical system (like your PC or laptop) or on a VM or a Docker image, where you have root privileges and then upload it to the cluster.

In the next sections, some important commands related with the workflow of the image creation are shown (like `create`, `expand`, `bootstrap` and `import`).

Create

Create a new Singularity formatted blank image.

The Singularity `create` command builds a new and empty file system with the specified `size` into a single file `container path`.

- USAGE:

```
singularity [...] create [-s Size(MB)|-F] <container path>
```

- EXAMPLE:

```
sudo singularity create -s 1024 Ubuntu.img
```

Expand

Grow the Singularity disk image by the given amount.

- USAGE:

```
singularity [...] expand [-s Size(MB)] <container path>
```

- EXAMPLE:

```
sudo singularity expand -s 256 Ubuntu.img
```

Bootstrap

Bootstrapping is the process where we install an operating system and then configure it appropriately for a specified need. To do this we use a bootstrap definition file which is a recipe of how to specifically build the container.

The bootstrap command is useful for creating a new bootstrap or modifying an existing one using a definition file that describes how to build the container.

- USAGE:

```
singularity [...] bootstrap <container path> <definition file>
```

- EXAMPLE:

```
sudo singularity bootstrap Ubuntu.img bootstrap.def
```

Bootstrap definition file

A Bootstrap definition file contains two main parts:

- **Header**: The core operating system to bootstrap whithin the container
- **Sections**: Shell scriptlets to run during the bootstrap process.

Header fields

The `Bootstrap:` keyword Identifies the singularity module (`yum`, `debootstrap`, `arch`, `docker`) that will be used for building the core components of the OS.

Depending on the loaded module several keywords can be used to particularize the installation (`MirrorURL`, `OSVersion`, `Include`, `From`, etc.).

An example of a bootstrap definition file header to import an Ubuntu docker container is shown below:

```
BootStrap: docker
From: ubuntu:16.04
```

Boostrap sections

- `%setup`: A Bourne shell scriptlet which will be executed on the host outside the container during bootstrap.
- `%post`: A Bourne shell scriptlet executed during bootstraping from inside the container.
- `%runscript`: A persistent scriptlet in the container that will be executed via the `singularity run` command.
- `%test`: A persistent scriplet in the container that will be executed via the `singularity test` command. This section will be also executed at the end of the bootstrap process.

Boostrap definition file example

The following bootstrap definition file example install a minimal Ubuntu OS with Python.

ubuntu.def

```
BootStrap: docker
From: ubuntu:latest

%setup
    echo "Looking in directory '$SINGULARITY_ROOTFS' for /bin/sh"
    if [ ! -x "$SINGULARITY_ROOTFS/bin/sh" ]; then
        echo "Hrmm, this container does not have /bin/sh installed..."
        exit 1
    fi
    exit 0

%post
    echo "Installing Python"
    apt -y --allow-unauthenticated install python
    exit 0

%runscript
    echo "Arguments received: $*"
    exec /usr/bin/python "$@"

%test
    python --version
```

You can find more examples in the following link:

https://github.com/singularityware/singularity/tree/master/examples

Import

Import takes a URI and will populate a container with the contents of the URI.

> **NOTE** Supported URIs: http/https, docker, file

If no URI is given, import will expect an incoming tar pipe.

> **NOTE** Supported file formats: .tar, .tar.gz, .tgz, .tar.bz2

The size of the container you need to create to import a complete system may be significantly larger than the size of the tar file/stream due to overheads of the container filesystem.

- USAGE:

```
    singularity [...] import <container path> [import from URI]
```

- EXAMPLES:

```
sudo singularity import Ubuntu.img file://ubuntu.tar.gz
sudo singularity import Ubuntu.img http://foo.com/ubuntu.tar.gz
sudo singularity import Ubuntu.img docker://ubuntu:latest
```

Container usage commands

To run a container you don't need special permissions, a normal user will be able to user the container.

In the next sections, some important commands related with the workflow of the image usage are shown (like `shell`, `exec` and `run`).

NOTE	If there is a daemon process running inside the container, then subsequent container commands will all run within the same namespaces. This means that the `--writable` and `--contain` options will not be honored as the namespaces have already been configured by the `singularity start` command.

The following options are common to `shell`, `exec` and `run` commands.

```
-B/--bind <spec>    A user-bind path specification.  spec can either be a
                    path or a src:dest pair, specifying the bind mount to
                    perform (can be called multiple times)
-c/--contain        This option disables the automatic sharing of writable
                    filesystems on your host (e.g. $HOME and /tmp).
-C/--containall     Contain not only file systems, but also PID and IPC
-H/--home <path>    Path to a different home directory to virtualize within
                    the container
-i/--ipc            Run container in a new IPC namespace
-p/--pid            Run container in a new PID namespace
--pwd               Initial working directory for payload process inside
                    the container
-S/--scratch <path> Include a scratch directory within the container that
                    is linked to a temporary dir (use -W to force location)
-u/--user           Try to run completely unprivileged (only works on very
                    new kernels/distros)
-W/--workdir        Working directory to be used for /tmp, /var/tmp and
                    $HOME (if -c/--contain was also used)
-w/--writable       By default all Singularity containers are available as
                    read only. This option makes the file system accessible
                    as read/write.
```

Shell

Obtain a shell (`/bin/sh` by default) within the container image. You can use the `SINGULARITY_SHELL` environment variable to change the default shell.

- USAGE:

```
singularity [...] shell [options...] <container path>
```

- EXAMPLE:

```
singularity shell Ubuntu.img
```

NOTE	When invoking a shell within a container, the container image is by default writable.

Exec

This command will allow you to execute any program within the given container image.

- USAGE:

```
singularity [...] exec [options...] <container path> <command>
```

- EXAMPLE:

```
singularity exec Ubuntu.img echo hola
```

Run

This command will launch a Singularity container and execute a runscript if one is defined for that container.

The runscript is a file at `/singularity` and defined during bootstrap. If this file is present (and executable) then this command will execute that file within the container automatically. All arguments following the container name will be passed directly to the runscript.

- USAGE:

```
singularity [...] run [options...] <container path> [...]
```

- EXAMPLE:

```
singularity run Ubuntu.img
```

Test

Run any defined tests for this particular container. Tests will be run contained (e.g. no persistent writable directories will be available inside of container) and will be executed as the calling user.

- USAGE:

```
singularity [...] test  <container path>
```

- EXAMPLE:

```
singularity test Ubuntu.img
```

Singularity in FinisTerrae II

Load Singularity

Access to Singularity in FT2 is provided using the module system. Use the following command to be able to execute `singularity`:

```
module load singularity/2.2.1
```

After loading the singularity module you can run the `singularity` executable. To get help about its usage, please type:

```
singularity --help
```

Run Singularity

Singularity module provides `run_singularity.sh` executable script in order to ease the usage of singularity.

```
run_singularity.sh <container path>
```

This script launch the `singularity exec` command binding some user directories inside the container. The path of most of this directories inside the container can be mounted in `/mnt`, and are available from the container via the following environment variables:

```
$LUSTRE_SCRATCH
$HOMESVG
$STORE
$LUSTRE
```

In order to use the `/scratch` directories of the compute nodes, this script expects to find the `/scratch` directory inside the container. This directory is accessible via the following variables inside the container:

```
$TMPDIR
$LOCAL_SCRATCH
```

As it was explained before, target directories must exist whithin the container. If this directories doesn't exist inside the container the following warning message is shown:

```
WARNING: Skipping user bind, non existant bind point (directory) in container: ...
```

There also are other ad-hoc scripts for running particular applications. These scripts follows the principals explained here.

Run Singularity with OpenMPI

You need to load the required modules to have access to OpenMPI.

```
module load gcc/5.3.0 openmpi/1.10.2
```

Then, you only need to launch `mpirun` with the command of singularity you want to call.

```
mpirun singularity exec <container> <command>
```

References

- http://singularity.lbl.gov
- http://singularity.lbl.gov/faq
- https://github.com/singularityware/intel-hpc-devcon
- https://groups.google.com/a/lbl.gov/forum/#!forum/singularity