Singularity containers

Please note: This instruction has been written for Singularity 3.8.x.

Please note: If you would like to fully manage your singularity container images directly on the cluster, including build and/or modify actions, please contact us and ask for the permission “singularity fakeroot” to be added to your account (because you will need it).

Singularity containers on the cluster

Singularity enables users to execute containers on High-Performance Computing (HPC) cluster like they are native programs or scripts on a host computer. For example, if the cluster system is running CentOS Linux, but your application runs in Ubuntu, you can create an Ubuntu container image, install your application into that image, copy the image to an approved location on the cluster and run your application using Singularity in its native Ubuntu environment.

The main advantage of Singularity is that containers are executed as an unprivileged user on the cluster system and, besides the local storage `TMPDIR`, they can access the network storage systems like `HOME`, `BIGWORK` and `PROJECT`, as well as GPUs that the host machine is equipped with.

Additionally, Singularity properly integrates with the Message Passing Interface (MPI), and utilizes communication fabrics such as InfiniBand and Intel Omni-Path.

If you want to create a container and set up an environment for your jobs, we recommend that you start by reading the Singularity documentation. The basic steps to get started are described below.

Building Singularity container using a recipe file

If you already have a pre-build container ready for use, you can simply upload the container image to the cluster and execute it. See the section below about running container images.

Below we will describe how to build a new or modify an existing container directly on the cluster. A container image can be created from scratch using a recipe file, or fetched from some remote container repository. In this sub-section, we will illustrate a recipe file method. In the next one, we will take a glance at remote container repositories.

Using a Singularity recipe file is the recommended way to create containers if you want to build reproducible container images. This example recipe file builds a RockyLinux 8 container:

rocky8.def

```
BootStrap: yum
OSVersion: 8
MirrorURL: https://ftp.uni-
hannover.de/rocky/%{OSVERSION}/BaseOS/$basearch/os
Include: yum wget
```

```
%setup
  echo "This section runs on the host outside the container during
bootstrap"

%post
  echo "This section runs inside the container during bootstrap"

  # install packages in the container
  yum -y groupinstall "Development Tools"
  yum -y install vim python38 epel-release
  yum -y install python38-pip

  # install tensorflow
  pip3 install --upgrade tensorflow

  # enable access to BIGWORK and PROJECT storage on the cluster system
  mkdir -p /bigwork /project

%runscript
  echo "This is what happens when you run the container"

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

%test
  echo "This test will be run at the very end of the bootstrapping
process"

  /bin/python3 --version
```

This recipe file uses the `yum` bootstrap module to bootstrap the core operation system, RockyLinux 8, within the container. For other bootstrap modules (e.g.. `docker`) and details on singularity recipe files, refer to the online documentation.

The next step is to build a container image on one of the cluster login servers.

Note: your account must be authorized to use the `--fakeroot` option. Please contact us at cluster-help@luis.uni-hannover.de.

Note: Currently, the `--fakeroot` option is enabled only on the cluster login nodes.

```
username@login01$ singularity build --fakeroot rocky8.sif rocky8.def
```

This creates an image file named `rocky8.sif`. By default, singularity containers are built as read-only SIF(Singularity Image Format) image files. Having a container in the form of a file makes it easier to transfer it to other locations both within the cluster and outside of it. Additionally, a SIF file can be signed and verified.

Note that a container as the SIF file can be built on any storage of the cluster you have a write access to. However, it is recommended to build containers either in your `$BIGWORK` or in some directory

under `/tmp` (or use the variable `$MY_SINGULARITY`) on the login nodes.

Note: Containers located only under the paths `$BIGWORK`, `$SOFTWARE` and `/tmp` are allowed to be executed using `shell`, `run` or `exec` commands, see the section below,

The latest version of the `singularity` command can be used directly on any cluster node without prior activation. Older versions are installed as loadable modules. Execute `module spider singularity` to list available versions.

Downloading containers from external repositories

Another easy way to obtain and use a Singularity container is to retrieve pre-build images directly from external repositories. Popular repositories are Docker Hub or Singularity Library. You can go there and search if they have a container that meets your needs. The `search` sub-command also lets you search for images at the Singularity Library. For docker images, use the search bock at Docker Hub instead.

In the following example we will pull the latest python container from Docker Hub and save it in a file named `python_latest.sif`:

```
username@login01$ singularity pull docker://python:latest
```

The `build` sub-command can also be used to download images, where you can additionally specify your preferred container file name:

```
 username@login01$ singularity build my-ubuntu20.04.sif
library://library/default/ubuntu:20.04
```

How to modify existing Singularity images

First you should check if you really need to modify the container image. For example, if you are using Python in an image and simply need to add new packages via `pip` you can do that without modifying the image by running `pip` in the container with the `--user` option.

To modify an existing SIF container file, you need to first convert it to a writable sandbox format.

Please note: Since the `--fakeroot` option of the `shell` and `build` sub-commands does not work with container sandbox when the container is located on a shared storage such as `BIGWORK`, `PROJECT` or `HOME`, the container sandbox must be stored locally on the login nodes. We recommend using the `/tmp` directory (or variable `$MY_SINGULARITY`) which has sufficient capacity.

```
username@login01$ cd $MY_SINGULARITY
username@login01$ singularity build --sandbox rocky8-sandbox rocky8.sif
```

The `build` command above creates a sandbox directory called *rocky8-sandbox* which you can then `shell` into in writable mode and modify the container as desired:

```
username@login01$ singularity shell --writable --fakeroot rocky8-sandbox
```

```
Singularity> yum install -qy python3-matplotlib
```

After making all desired changes, you exit the container and convert the sandbox back to the SIF file using:

```
Singularity> exit
username@login01$ singularity build -F --fakeroot rocky8.sif rocky8-sandbox
```

Note: you can try to remove the sandbox directory *rocky8-sandbox* afterward but there might be a few files you can not delete due to the namespace mappings that happens. The daily `/tmp` cleaner job will eventually clean it up.

Running container images

Please note: In order to run a Singularity container, the container SIF file or sandbox directory must be located either in your `$BIGWORK`, in your group's `$SOFTWARE` or in the `/tmp` directory.

There are four ways to run a container under Singularity.

If you simple call the container image as an executable or use the Singularity `run` sub-command it will carry out instructions in the `%runscript` section of the container recipe file:

How to call the container SIF file:

```
username@login01:~$ ./rocky8.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

Use the `run` sub-command:

```
username@login01:~$ singularity run rocky8.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

The Singularity `exec` sub-command lets you execute an arbitrary command within your container instead of just the `%runscript`. For example, to get the content of file `/etc/os-release` inside the container:

```
username@login01:~$ singularity exec rocky8.sif cat /etc/os-release
NAME="Rocky Linux"
VERSION="8.4 (Green Obsidian)"
....
```

The Singularity `shell` sub-command invokes an interactive shell within a container. Note the `Singularity>` prompt within the shell in the example below:

```
username@login01:$ singularity shell rocky8.sif
```

```
Singularity>
```

Note that all three sub-commands `shell`, `exec` and `run` let you execute a container directly from remote repository without first downloading it on the cluster. For example, to run an one-liner "Hello World" ruby program:

```
 username@login01:$ singularity  exec library://sylabs/examples/ruby ruby -e
'puts "Hello World!"'
Hello World!
```

Please note: You can access (read & write mode) your HOME, BIGWORK and PROJECT (only login nodes) storage from inside your container. In addition, the `/tmp` (or TMPDIR on compute nodes) directory of a host machine is automatically mounted in a container. Additional mounts can be specified using the `--bind` option of the `exec`, `run` and `shell` sub-commands, see `singularity run --help`.

Singularity & parallel MPI applications

In order to containerize your parallel MPI application and run it properly on the cluster system you have to provide MPI library stack inside your container. In addition, the userspace driver for Mellanox InfiniBand HCAs should be installed in the container to utilize cluster InfiniBand fabric as a MPI transport layer.

This example Singularity recipe file `ubuntu-openmpi.def` retrieves an Ubuntu container from Docker Hub, and installs required MPI and InfiniBand packages:

Ubuntu 16.x

ubuntu-openmpi.def

```
BootStrap: docker
From: ubuntu:xenial

%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common libibverbs1 libmlx4-1

# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project

# enable access to /scratch dir. required by mpi jobs
mkdir -p /scratch
```

Ubuntu 18.x - 21.x

ubuntu-openmpi.def

```
BootStrap: docker
From: ubuntu:latest

%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common ibverbs-providers

# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project

# enable access to /scratch dir. required by mpi jobs
mkdir -p /scratch
```

Once you have built the image file `ubuntu-openmpi.sif` as explained in the previous sections, your MPI application can be run as follows (assuming you have already reserved a number of cluster compute nodes):

```
module load GCC/8.3.0 OpenMPI/3.1.4
mpirun singularity exec ubuntu-openmpi.sif /path/to/your/parallel-mpi-app
```

The above lines can be entered at the command line of an interactive session, or can also be inserted into a batch job script.

Further Reading

- Singularity home page
- Singularity Library
- Docker Hub