Running Singularity Containers on Comet

Background

What is Singularity?*

"Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. This means that you don't have to ask your cluster admin to install anything for you - you can put it in a Singularity container and run."

[*from the Singularity web site at http://singularity.lbl.gov/ [http://singularity.lbl.gov/]]

There are numerous good tutorials on how to install and run Singularity on Linux, OS X, or Windows so we won't go into much detail on that process here. In this tutorial you will learn how to run Singularity on Comet. First we will review how to access a compute node on Comet and provide a simple example to help get you started. There are numerous tutorial on how to get started with Singularity, but there are some details specific to running Singularity on Comet which are not covered in those tutorials. This tutorial assumes you already have an account on Comet. You will also need access to a basic set of example files to get started. SDSC hosts a Github repository containing a 'Hello world!" example which you may clone with the following command:

> git clone https://github.com/hpcdevops/singularity-hello-world.git

Tutorial Contents

- Why Singularity?
- Downloading & Installing Singularity
- Building Singularity Containers
- Running Singularity Containers on Comet
- Running Tensorflow on Comet Using Singularity

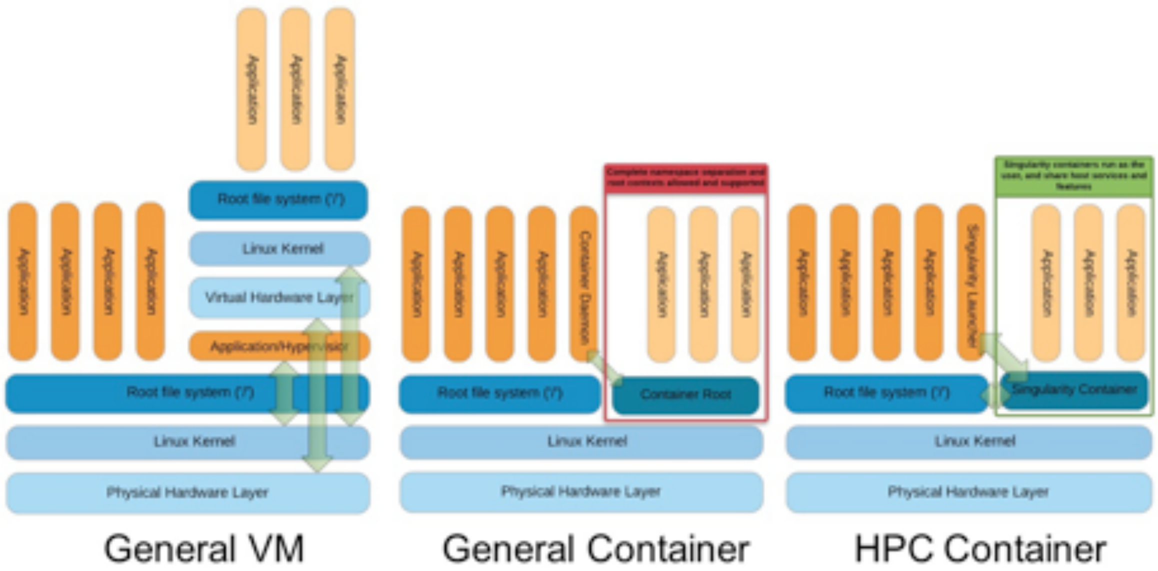
Why Singularity?

Below is a typical list of commands you would need to issue in order to implement a functional Python installation for scientific research:

```
COMMAND=apt-get -y install libx11-dev
COMMAND=apt-get install build-essential python-
libdev
COMMAND=apt-get install build-essential openmpi-
dev
COMMAND=apt-get install cmake
COMMAND=apt-get install g++
COMMAND=apt-get install git-lfs
COMMAND=apt-get install libXss.so.1
COMMAND=apt-get install libgdal1-dev libproj-dev
COMMAND=apt-get install libjsoncpp-dev
libjsoncpp0
COMMAND=apt-get install libmpich-dev --user
COMMAND=apt-get install libpthread-stubs0
libpthread-stubs0-dev libx11-dev libx11-d
COMMAND=apt-get install libudev0:i386
COMMAND=apt-get install numpy
COMMAND=apt-get install python-matplotlib
COMMAND=apt-get install python3
```

Singularity allows you to avoid this time-consuming series of steps by packaging these commands in a re-usable and editable script, allowing you to quickly, easily, and repeatedly implement a custom container designed specifically for your analytical needs.

The diagram below compares a VM vs. Docker vs. Singularity.



Source:Â Greg Kurtzer keynote at HPC Advisory Council 2017 @ Stanford [http://www.hpcadvisorycouncil.com/events/2017/stanford-workshop/pdf/GMKurtzer_Singularity_Keynote_Tuesday_02072017.pdf#43]

Hands-On Tutorials

Next, let's get some hands-on experience with Singularity. The following tutorial includes links to asciinema video tutorials created by SDSC HPC Systems Manager, Trevor Cooper (Thanks, Trevor!) which allow you to see the console interactivity and output in detail. Look for the video icon like the one shown to the right corresponding to the task you are currently working on.

Downloading & Installing Singularity

- Download & Unpack Singularity
- Configure & Build Singularity
- Install & Test Singularity

Download & Unpack Singularity

First we download and unpack the source using the following

First we download and upack the source using the following commands (assuming your user name is 'test_user' and you are working on your local computer with super user privileges):

```
[test_user@localhost ~]$ wget
https://github.com/singularityware/singularity/
releases/download/2.5.1/singularity-2.5.1.tar.gz
tar -zxf singularity-2.5.1.tar.gz
```

[https://asciinema.org/a/129866]

If the file is successfully extracted, you should be able to view the results:

```
[test_user@localhost ~]$ cd singularity-2.5.1/
[test_user@localhost singularity-2.5.1]$ ls
```

Configure & Build Singularity

[https://asciinema.org/a/129867]Next we configure and build the package. To configure, enter the following command (we'll leave out the command prompts):

```
./configure
```

To build, issue the following command:

```
make
```

This may take several seconds depending on your computer.

Install & Test Singularity

[https://asciinema.org/a/129868]To complete the installation enter:

```
sudo make install
```

You should be prompted to enter your admin password.

Once the installation is completed, you can check to see if it succeeded in a few different ways:

```
which singularity singularity -version
```

You can also run a selftest with the following command:

```
singularity selftest
```

The output should look something like:

```
+ sh -c test -f
/usr/local/etc/singularity/singularity.conf
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/action-suid
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/create-suid
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/expand-suid
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/export-suid
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/import-suid
(retval=0) OK
+ test -u
/usr/local/libexec/singularity/bin/mount-suid
(retval=0) OK
```

Building Singularity Containers

The process of building a Singularity container consists of a few distinct steps as follows.

- Upgrading Singularity (if needed)
- Create an Empty Container
- Import into Container
- Shell into Container
- Write into Container
- Bootstrap Container

We will go through each of these steps in detail.

Upgrading Singularity

We recommend building containers using the same version of Singularity, 2.5.1, as exists on Comet. This is a 2 step process.

Step 1: run the script below to remove your existing Singularity:

```
#!/bin/bash
#
# A cleanup script to remove Singularity

sudo rm -rf /usr/local/libexec/singularity
sudo rm -rf /usr/local/etc/singularity
sudo rm -rf /usr/local/include/singularity
sudo rm -rf /usr/local/lib/singularity
sudo rm -rf /usr/local/var/lib/singularity/
sudo rm /usr/local/bin/singularity
sudo rm /usr/local/bin/run-singularity
sudo rm /usr/local/etc/bash_completion.d/singularity
sudo rm /usr/local/man/man1/singularity.1
```

Step 2: run the following script to install Singularity 2.5.1:

```
#!/bin/bash
#
# A build script for Singularity (http://singularity.lbl.gov/)

declare -r SINGULARITY_NAME='singularity'
declare -r SINGULARITY_VERSION='2.5.1'
declare -r SINGULARITY_PREFIX='/usr/local'
declare -r SINGULARITY_CONFIG_DIR='/etc'

sudo apt update
sudo apt install python dh-autoreconf build-essential debootstrap

cd ../
tar -xzvf "${PWD}/tarballs/${SINGULARITY_NAME}-
```

tar -xzvf "${PWD}/tarballs/${SINGULARITY_NAME}-
${SINGULARITY_VERSION}.tar.gz"
cd "${SINGULARITY_NAME}-${SINGULARITY_VERSION}"
./configure --prefix="${SINGULARITY_PREFIX}" --
sysconfdir="${SINGULARITY_CONFIG_DIR}"
make
sudo make install

Create an Empty Container

[https://asciinema.org/a/130106]To create an empty Singularity container, you simply issue the following command:

```
singularity create centos7.img
```

This will create a CentOS 7 container with a default size of ~805 Mb. Depending on what additional configurations you plan to make to the container, this size may or may not be big enough. To specify a particular size, such as ~4 Gb, include the -s parameter, as shown in the following command:

```
singularity create -s 4096 centos7.img
```

To view the resulting image in a directory listing, enter the following:

```
ls
```

Import Into a Singularity Container

[https://asciinema.org/a/130107]Next, we will import a Docker image into our empty Singularity container:

```
singularity import centos7.img
docker://centos:7
```

Shell Into a Singularity Container

[https://asciinema.org/a/130109]Once the container actually contains a CentOS 7 installation, you can ‘shell’ into it with the installation, you can “shell” into it with the following:

```
singularity shell centos7.img
```

Once you enter the container you should see a different command prompt. At this new prompt, try typing:

```
whoami
```

Your user id should be identical to your user id outside the container. However, the operating system will probably be different. Try issuing the following command from inside the container to see what the OS version is:

```
cat /etc/*-release
```

Write Into a Singularity Container

[https://asciinema.org/a/130110]Next, let’s trying writing into the container (as root):

```
sudo /usr/local/bin/singularity shell
-w centos7.img
```

You should be prompted for your password, and then you should see something like the following:

```
Invoking an interactive shell within the
container...
```

Next, let’s create a script within the container so we can use it to test the ability of the container to execute shell scripts:

```
vi hello_world.sh
```

The above command assumes you know the vi editor. Enter the following text into the script, save it, and quit the vi editor:

```
#!/bin/bash
echo “Hello, World!”
```

You may need to change the permissions on the script so it can be executable:

```
chmod +x hello_world.sh
```

Try running the script manually:

```
./hello_world.sh
```

The output should be:

The output should be:

```
Hello, World!
```

Bootstrapping a Singularity Container

[https://asciinema.org/a/130111]Bootstrapping a Singularity container allows you to use what is called a ‘definitions file’ so you can reproduce the resulting container configurations on demand.

Let’s say you want to create a container with Ubuntu, but you may want to create variations on the configurations without having to repeat a long list of commands manually. First, we need our definitions file. Below is the contents of a definitions file which should suffice for our purposes.

```
Bootstrap: docker
From: ubuntu:latest
%runscript
exec echo “The runscript is the containers
default runtime command!”

%files
/home/testuser/ubuntu.def /data/ubuntu.def

%environment
VARIABLE=HELLOWORLD
Export VARIABLE
%labels
AUTHOR testuser@sdsc.edu

%post
apt-get update && apt-get -y install python3 git
wget
mkdir /data
echo “The post section is where you can install
and configure your container.”
```

To bootstrap your container, first we need to create an empty container.

```
singularity create -s 4096 ubuntu.img
```

Now, we simply need to issue the following command to configure our container with Ubuntu:

```
sudo /usr/local/bin/singularity bootstrap
./ubuntu.img ./ubuntu.def
```

This may take a while to complete. In principle, you can accomplish the same result by manually issuing each of the commands contained in the script file, but why do that when you can use bootstrapping to save time and avoid errors.

If all goes according to plan, you should then be able to shell into your new Ubuntu container.

Running Singularity Containers on Comet

Of course, the purpose of this tutorial is to enable you to use the San Diego Supercomputer Center's Comet supercomputer to run your jobs. This assumes you have an account on Comet already. If you do not have an account on Comet and you feel you can justify the need for such an account (i.e. your research is limited by the limited compute power you have in your government-funded research lab), you can request a 'Startup Allocation' through the XSEDE User Portal:

https://portal.xsede.org/allocations-overview#types-trial [https://portal.xsede.org/allocations-overview#types-trial]

You may create a free account on the XUP if you do not already have one and then proceed to submit an allocation request at the above link.

[NOTE: SDSC provides a Comet User Guide (http://www.sdsc.edu/support/user_guides/comet.html [http://www.sdsc.edu/support/user_guides/comet.html]) to help get you started with Comet. Learn more about The San Diego Supercomputer Center at http://www.sdsc.edu [http://www.sdsc.edu] .]

This tutorial walks you through the following four steps towards running your first Singularity container on Comet:

- Transfer the Container to Comet
- Run the Container on Comet
- Allocate Resources to Run the Container
- Integrate the Container with Slurm
- Use existing Comet Containers

Transfer the Container to Comet

[https://asciinema.org/a/130195]Once you have created your container on your local system, you will need to transfer it to Comet. There are multiple ways to do this and it can take a varying amount of time depending on its size and your network connection speeds.

To do this, we will use scp (secure copy). If you have a Globus account and your containers are more than 4 Gb you will probably want to use that file transfer method instead of scp.

Browse to the directory containing the container. Copy the container to your scratch directory on Comet. By issuing the following command:

```
scp ./centos7.img
comet.sdsc.edu:/oasis/scratch/comet/test_user/temp_project/
```

The container is ~805 Mb so it should not take too long, hopefully.

Run the Container on Comet

[https://asciinema.org/a/130196]Once the file is transferred, login to Comet (assuming your Comet user is named 'test_user'):

```
ssh test_user@comet.sdsc.edu
```

Navigate to your scratch directory on Comet, which should be something like:

```
[test_user@comet-ln3 ~]$ cd
/oasis/scratch/comet/test_user/temp_project/
```

Next, you should submit a request for an interactive session on one of Comet's compute, debug, or shared nodes.

```
[test_user@comet-ln3 ~]$ srun --pty --nodes=1 --
ntasks-per-node=24 -p compute -t 01:00:00 --wait
0 /bin/bash
```

Once your request is approved your command prompt should reflect the new node id.

Before you can run your container you will need to load the Singularity module (if you are unfamiliar with modules on Comet, you may want to review the Comet User Guide). The command to load Singularity on Comet is:

```
[test_user@comet-ln3 ~]$ module load singularity
```

You may issue the above command from any directory on Comet. Recall that we added a hello_world.sh script to our centos7.img container. Let's try executing that script with the following command:

```
[test_user@comet-ln3 ~]$ singularity exec
/oasis/scratch/comet/test_user/temp_project/singularity/centos7.img
/hello_world.sh
```

If all goes well,Â you should see "Hello, World!" in the console output. You might also see some warnings pertaining to non-existent bind points. You can resolve this by adding some additional lines to your definitions file before you build your container. We did not do that for this tutorial, but you would use a command like the following in your definitions file:

```
# create bind points for SDSC HPC environment
mkdir /oasis /scratch/ /comet /temp_project
```

You will find additional examples located in the following locations on Comet:

```
/share/apps/examples/SI2017/Singularity
```

and

```
/share/apps/examples/SINGULARITY
```

Allocate Resources to Run the Container

[https://asciinema.org/a/130197]It is best to avoid working on Comet's login nodes since they can become a performance bottleneck not only for you but for all other users. You should rather allocate resources specific for computationally-intensive jobs. To allocate a 'compute node' for your user on Comet, issue the following command:

```
[test_user@comet-ln3 ~]$ salloc -N 1 -t 00:10:00
```

This allocation requests a single node (-N 1) for a total time of 10 minutes (-t 00:10:00). Once your request has been approved, your computer node name should be displayed, e.g. comet-17-12.

Now you may login to this node:

```
[test_user@comet-ln3 ~]$ ssh comet-17-12
```

Notice that the command prompt has now changed to reflect the fact that you are on a compute node and not a login node.

```
[test_user@comet-06-04 ~]$
```

Next, load the Singularity module, shell into the container, and execute the hello_world.sh script:

```
[test_user@comet-06-04 ~]$ module load
singularity
[test_user@comet-06-04 ~]$ singularity shell
centos7.img
[test_user@comet-06-04 ~]$ ./hello_world.sh
```

If all goes well, you should see "Hello, World!" in the console output.

Integrate the Container with Slurm

[https://asciinema.org/a/130218]Of course, most users simply want to submit their jobs to the

Comet queue and let it run to completion and go on to other things while waiting. Slurm is the job manager for Comet.

Below is a job script (which we will name singularity_mvapich2_hellow.run) which will submit your Singularity container to the Comet queue and run a program, hellow.c (written in C using MPI and provided as part of the examples with the mvapich2 default installation).

```
#!/bin/bash
#SBATCH --job-name="singularity_mvapich2_hellow"
#SBATCH --
output="singularity_mvapich2_hellow.%j.out"
#SBATCH --
error="singularity_mvapich2_hellow.%j.err"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --time=00:10:00
#SBATCH --export=all module load mvapich2_ib
singularity

CONTAINER=/oasis/scratch/comet/$USER/temp_project/singularity/centos7-
mvapich2.img

mpirun singularity exec ${CONTAINER}
/usr/bin/hellow
```

The above script requests 2 nodes and 24 tasks per node with a wall time of 10 minutes. Notice that two modules are loaded (see the line beginning with ‘module’), one for Singularity and one for MPI. An environment variable ‘CONTAINER’ is also defined to make it a little easier to manage long reusable text strings such as file paths.

You may need to add a line specifying with allocation to be used for this job. When you are ready to submit the job to the Comet queue, issue the following command:

```
[test_user@comet-06-04 ~]$ sbatch -p debug
./singularity_mvapich2_hellow.run
```

To view the status of your job in the Comet queue, issue the following:

```
[test_user@comet-06-04 ~]$ squeue -u test_user
```

When the job is complete, view the output which should be written to the output file singularity_mvapich2_hellow.%j.out where %j is the job ID (let's say the job ID is 1000001):

```
[test_user@comet-06-04 ~]$ more
singularity_mvapich2_hellow.1000001.out
```

The output should look something like the following:

```
.
.
.
Hello world from process 28 of 48
Hello world from process 29 of 48
Hello world from process 30 of 48
Hello world from process 31 of 48
Hello world from process 32 of 48
Hello world from process 33 of 48
Hello world from process 34 of 48
Hello world from process 35 of 48
Hello world from process 36 of 48
Hello world from process 37 of 48
Hello world from process 38 of 48
.
.
.
```

Use Existing Comet Containers

SDSC User Support staff, Marty Kandes, has built several custom Singularity containers designed specifically for the Comet environment.

Learn more about these containers for Comet [about_comet_singularity_containers.html].

News Flash!

Now there's an easier way to run a Singularity container on Comet...

PULL IT!

[https://asciinema.org/a/129906]Comet now supports the capability to pull a container directly from any properly configured remote singularity hub. For example, the following command can pull a container from the hpcdevops singularity hub straight to an empty container located on Comet:

```
[test_user@comet-06-04 ~]$ singularity pull
shub://hpcdevops/singularity-hello-world:master
```

The resulting container should be named something like singularity-hello-world.img.

Learn more about Singularity Hubs and container collections at:

https://singularity-hub.org/collections [https://singularity-hub.org/collections]

That's it! Congratulations! You should now be able to run Singularity containers on Comet either interactively or through the job queue. We hope you found this tutorial useful. Please contact support@xsede.org [mailto:support@xsede.org] with any questions you might have. Your Comet-related questions will be routed to the amazing SDSC Support Team.

Using Tensorflow With Singularity

One of the more common advantages of using Singularity is the ability to use pre-built containers for specific applications which may be difficult to install and maintain by yourself, such as Tensorflow. The most common example of a Tensorflow application is character recognition using the MNIST dataset. You can learn more about this dataset at

http://yann.lecun.com/exdb/mnist/
[http://yann.lecun.com/exdb/mnist/].

XSEDE's Comet supercomputer supports Singularity and provides several pre-built container which run Tensorflow. Below is an example batch script which runs a Tensorflow job within a Singularity container on Comet. Copy this script and paste it into a shell script named "mnist_tensorflow_example.sb".

```
#!/bin/bash
#SBATCH --job-name="TensorFlow"
#SBATCH --output="TensorFlow.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:k80:1
#SBATCH -t 01:00:00

#Run the job

module load singularity
singularity exec
/share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img
lsb_release -a
singularity exec
/share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img python -m
tensorflow.models.image.mnist.convolutional
```

To submit the script to Comet, first you'll need to request a compute node with the following command (replace account with your XSEDE account number):

```
[test_user@comet-ln3 ~]$ srun --
account=your_account_code --partition=gpu-shared
--gres=gpu:1 --pty --nodes=1 --ntasks-per-node=1
-t 00:30:00 --wait=0 --export=ALL /bin/bash
```

To submit a job to the Comet queue, issue the following command:

```
[test_user@comet-06-04 ~]$ sbatch
mnist_tensorflow_example.sb
```

When the job is done you should see an output file in your output directory containing something resembling the following:

```
Distributor ID: Ubuntu
Description: Ubuntu 16.04 LTS
Release: 16.04
Codename: xenial
^[[33mWARNING: Non existent bind point
(directory) in container: '/scratch'
^[[0mI
tensorflow/stream_executor/dso_loader.cc:108]
successfully opened CUDA library libcublas.so
locally
I tensorflow/stream_executor/dso_loader.cc:108]
successfully opened CUDA library libcudnn.so
locally
I tensorflow/stream_executor/dso_loader.cc:108]
successfully opened CUDA library libcufft.so
locally
I tensorflow/stream_executor/dso_loader.cc:108]
successfully opened CUDA library libcuda.so.1
locally
I tensorflow/stream_executor/dso_loader.cc:108]
successfully opened CUDA library libcurand.so
locally
I
tensorflow/core/common_runtime/gpu/gpu_init.cc:102]
Found device 0 with properties:
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 0000:85:00.0
Total memory: 11.17GiB
Free memory: 11.11GiB
I
tensorflow/core/common_runtime/gpu/gpu_init.cc:126]
DMA: 0
I
tensorflow/core/common_runtime/gpu/gpu_init.cc:136]
0: Y
```

```
I
tensorflow/core/common_runtime/gpu/gpu_device.cc:838]
Creating TensorFlow device (/gpu:0) -> (device:
0, name: Tesla K80, pci bus id: 0000:85:00.0)
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Initialized!
Step 0 (epoch 0.00), 40.0 ms
Minibatch loss: 12.054, learning rate: 0.010000
Minibatch error: 90.6%
Validation error: 84.6%
Step 100 (epoch 0.12), 12.6 ms
Minibatch loss: 3.293, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 7.0%

.

.

.

Step 8400 (epoch 9.77), 11.5 ms
Minibatch loss: 1.596, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Step 8500 (epoch 9.89), 11.5 ms
Minibatch loss: 1.593, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.8%
Test error: 0.9%
```

Congratulations! You have successfully trained a neural network to recognize ascii numeric characters.

Contact Us

help@xsede.org [mailto:help@xsede.org]

SDSC - UC San Diego, MC 0505 | 9500 Gilman Drive | La Jolla, CA 92093-0505
Tel. (858) 534-5000 | Fax. (858) 534-5152 | info@sdsc.edu [mailto:info@sdsc.edu] | Terms of Use [../../terms_of_use.html] | Privacy Policy [../../privacy.html] | Sitemap [../../about_sdsc/sitemap.html] | Got feedback? [javascript:void(0);]