Singularity Tutorials

Simple Cases

You can use the pull or build commands to download pre-built images from resources like Singularity Hub or Docker.

This example pulls a container from Docker

```
$  singularity pull docker://godlovedc/lolcow
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
[...]
Writing manifest to image destination
Storing signatures
INFO:    Creating SIF file...
INFO:    Build complete: /home/u23/ric/.singularity/cache/oci-tmp
/a692b57abc43035b197b10390ea2c12855d21649f2ea2cc28094d18b93360eeb/lolcow_latest.sif


$ singularity run lolcow_latest.sif
 ______________________________________
/ Perilous to all of us are the devices \
| of an art deeper than we ourselves    |
| possess.                              |
|                                       |
| -- Gandalf the Grey [J.R.R. Tolkien,  |
\ "Lord of the Rings"]                  /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Singularity Build

CentOS with Tensorflow Example

This is an example of creating a singularity image to run code that is not supported on HPC. This example uses Tensorflow but any application could be installed in its place. It also uses CentOS but it could just as easily be Ubuntu.

1. Install Singularity on linux workstation - https://sylabs.io/docs/
2. Create the recipe file (which used to be called a definition file) on a workstation with root authority. Let's call it centosTflow.def

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum
# best to build up container using kickstart mentality.
# ie, to add more packages to image,
# re-run bootstrap command again.
# bootstrap on existing image will build on top of it, not overwriting it/restarting from scratch
# singularity .def file is like kickstart file
# unix commands can be run, but if there is any error, the bootstrap process ends
%setup
   # commands to be executed on host outside container during bootstrap
%post
    # commands to be executed inside container during bootstrap
    # add python and install some packages
    yum -y install vim wget python3 epel-release
     # install tensorflow
    pip3 install --upgrade pip
    pip3 install tensorflow-gpu==2.0.0-rc1
     # create bind points for storage.
     mkdir /extra
     mkdir /xdisk
     exit 0
# %runscript
   # commands to be executed when the container runs
# %test
   # commands to be executed within container at close of bootstrap process
   python --version
```

3. Create the Singularity container using the recipe file. This new image is about 1GB in size.

> This must be done on a workstation where you have root authority. This capability is not permitted on the Ocelote login or compute nodes

```
singularity build centosTflow.sif centosTflow.def
```

4. Copy the new image file to your space on HPC. /extra might be a good location as the image might use up your remaining home. There is a line in the definition file that will create the mount for /extra.
5. Test with a simple command

```
$module load singularity
$singularity exec centosTFlow.simg python3 --version
Python 3.6.8
```

6. Or slightly more complex create a simple python script called hello.py:

```
$singularity exec centosTFlow.simg python hello.py
Hello World: The Python version is 2.7.5
$
```

```
$singularity shell centosTFlow.simg
Singularity.centosTFlow.img>python hello.py
Hello World: The Python version is 2.7.5
Singularity.centosTFlow.simg>
```

7. And now test tensorflow on a GPU node with this Linear Regression example TFlow_example.py:

```
$singularity exec --nv centosTFlow.simg python3 /extra/netid/TFlow_example.py
step: 50, loss: 0.078236, W: 0.272136, b: 0.653456
step: 100, loss: 0.078087, W: 0.270928, b: 0.662019
step: 150, loss: 0.077954, W: 0.269792, b: 0.670079
step: 200, loss: 0.077837, W: 0.268722, b: 0.677663
step: 250, loss: 0.077733, W: 0.267715, b: 0.684801
step: 300, loss: 0.077641, W: 0.266768, b: 0.691518
step: 350, loss: 0.077560, W: 0.265876, b: 0.697839
step: 400, loss: 0.077488, W: 0.265037, b: 0.703788
step: 450, loss: 0.077424, W: 0.264247, b: 0.709386
step: 500, loss: 0.077367, W: 0.263504, b: 0.714655
step: 550, loss: 0.077317, W: 0.262805, b: 0.719612
step: 600, loss: 0.077273, W: 0.262147, b: 0.724278
step: 650, loss: 0.077233, W: 0.261527, b: 0.728669
step: 700, loss: 0.077198, W: 0.260944, b: 0.732801
step: 750, loss: 0.077168, W: 0.260396, b: 0.736690
step: 800, loss: 0.077140, W: 0.259880, b: 0.740350
step: 850, loss: 0.077116, W: 0.259394, b: 0.743794
step: 900, loss: 0.077095, W: 0.258937, b: 0.747035
step: 950, loss: 0.077076, W: 0.258506, b: 0.750085
step: 1000, loss: 0.077059, W: 0.258102, b: 0.752956
```

This example has been created and copied to /unsupported/singularity which is available for anyone to copy to their personal space and run. Look for centosTflow.sif and the recipe file if you want is centosTflow.recipe

Tensorflow as a Module on Ocelote

Tensorflow can be run using modules to make the process easier.

1. *module load tensorflow*
   This will automatically load singularity.
2. Example command
   tensorflow python TFlow_example.py
   This actually runs an alias of "singularity exec --nv /unsupported/singularity/tensorflow_1.13.1.sif python TFlow_example.py"
   This example assumes that the script *TFlow_example.py* is in your current working directory.
3. You can include the two commands in a PBS script which runs a job on a GPU node.

Tensorflow as a Python Package on ElGato

ElGato has a new operating system that meets the requirements of packages like Tensorflow without needing to use containers

```
$module load python/3.5
$python3
Python 3.5.5 (default, May 20 2019, 16:32:30)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
```

Running Jobs

Singularity is not to be run on login nodes. That is a general policy for any application.

To run a Singularity container image on ElGato or Ocelote interactively, you need to either allocate an interactive session or submit a job using a script. In this sample session, the Tensorflow Singularity container from above is started, and python is run. Note that in this example, you would be running the version of python that is installed within the Singularity container, not the version on the cluster.

ElGato Interactive Example

```
 $ qsub -I -N jobname -W group_list=hpcteam -q windfall -l select=1:ncpus=16:mem=250gb:ngpus=1 -l walltime=1:
0:0


[netid@gpu44 singularity]$ module load singularity
[netid@gpu44 singularity]$ singularity exec dockerTF.img python\ /extra/chrisreidy/singularity/TFlow_example.
py
Instructions for updating:
Use `tf.global_variables_initializer` instead.
(0, array([ 0.12366909], dtype=float32), array([ 0.3937912], dtype=float32))
(20, array([ 0.0952933], dtype=float32), array([ 0.30251619], dtype=float32))
...
(200, array([ 0.0999999], dtype=float32), array([ 0.30000007], dtype=float32))
[netid@gpu44 singularity]$ exit
```

Ocelote Interactive Example

The process is the same except that the command to initiate the interactive session will look more like:

```
 $ qsub -I -N jobname -W group_list=hpcteam -q windfall -l select=1:ncpus=28:mem=168gb:ngpus=1 -l walltime=1:
0:0
```

ElGato Job Submission

Running a job with Singularity is as easy as running other jobs. The PBS script might look like this:

```
#!/bin/bash
#PBS -N singularity-job
#PBS -W group_list=PI
#PBS -q windfall
#PBS -j oe
#PBS -l select=1:ncpus=1:mem=6gb
#PBS -l walltime=01:00:00
#PBS -l cput=12:00:00
module load singularity
cd /extra/netid/singularity
date
singularity exec --nv dockerTF.img python TFlow_example.py
date
```

Ocelote Job Submission

The PBS script might look like this, and the results will be found in singularity-job.ojobid.

```
#!/bin/bash
#PBS -N singularity-job
#PBS -W group_list=PI
#PBS -q windfall
#PBS -j oe
#PBS -l select=1:ncpus=1:mem=6gb
#PBS -l walltime=01:00:00
#PBS -l cput=12:00:00
module load singularity
cd /extra/netid/singularity
date
singularity exec --nv dockerTF.img python TFlow_example.py
date
```

GPU Computing

The nodes on Ocelote are required to keep the operating system at CentOS 6 for reasons of consistency. All of the Machine Learning workflows require CentOS 7. By using Singularity containers that are built with CentOS 7 (or Ubuntu 16) you bypass this constraint.

There is more detail in the section on GPU's

Cuda / Tensorflow Example

This is a complex example that details the use of Singularity. We already maintain a library of Machine Learning workflows that are completely built as Singularity containers

If you want to build a singularity container image that can run applications on Ocelote or ElGato GPU nodes, you must prepare your container:

- Download the .run installer for the same NVIDIA driver that is currently running on our GPU nodes
- Extract the contents (don't actually need to install the driver)
- Move all of the libraries to a central location like /usr/local/nvidia and make all necessary symbolic links
- Download and install the .run file for the same CUDA libraries that are currently running on the GPU node
- Download, extract, and copy the correct cuDNN libraries
- Edit and export $PATH, $LD_LIBRARY_PATH, and $CUDA_HOME to point to the correct libraries and binaries

For your convenience, the staff at the National Institute of Health maintains an installation script (called cuda4singularity) that automates this process. It has been tested with Ubuntu 16.4 and CentOS 7. You can either copy or download cuda4singularity into an existing container and execute it with root privileges, or you can add the lines of code into your .def file and install the NVIDIA/CUDA libraries during the bootstrap procedure. (Note that your container will probably need ~10GB of empty space for the CUDA libraries to install properly. This is because the installer checks for minimum space requirements before running.)

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum wget

%setup
    # commands to be executed on host outside container during bootstrap

%post
    # commands to be executed inside container during bootstrap

    # yum needs some tlc to work properly in container
    RELEASEVER=7
    ARCH=x86_64
    echo $RELEASEVER > /etc/yum/vars/releasever
    echo $ARCH > /etc/yum/vars/arch
    wget http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-8.noarch.rpm
    rpm -ivh --nodeps epel-release-7-8.noarch.rpm
    # yum -d 10 check-update  # this line caused problems in testing

    # install other needed packages
    yum -y install man which tar gzip vim-minimal perl python python-dev python-pip

    # create bind points for NIH HPC environment
    mkdir -p /extra /rsgrps

    # download and run NIH HPC cuda for singularity installer
    wget ftp://helix.nih.gov/CUDA/cuda4singularity
    chmod 755 cuda4singularity
    ./cuda4singularity
    rm cuda4singularity

    # install tensorflow
    pip install --upgrade pip
    pip install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.10.0-cp27-none-
linux_x86_64.whl

%runscript
    # commands to be executed when the container runs

%test
    # commands to be executed within container at close of bootstrap process
```

Be patient. This bootstrap procedure will take a long time to run and may look like it has stalled at several points. If you watch the output you may see CUDA issuing warnings and complaining about an incomplete installation. Don't worry. The drivers are running on the GPU nodes so they don't need to be installed within the container.

After creating a container with one of these files, you can copy it to ElGato and test it. On Ocelote, the same test will access a GPU node with this interactive command:

```
qsub -I -N jobname -W group_list=YOUR_GROUP_HERE -q windfall -l select=1:ncpus=28:mem=168gb:ngpus=1 -l
cput=3:0:0 -l walltime=1:0:0
```

```
[netid@elgato]$ bsub -R gpu -Is bash
Job <633858> is submitted to default queue <windfall>.
<<Waiting for dispatch ...>>
<<Starting on gpu64>>
[netid@gpu64]$ module load singularity
[netid@gpu64]$ singularity shell cuda4tf.img
WARNING: Not mounting current directory: user bind control is disabled by system administrator
Singularity: Invoking an interactive shell within container...

Singularity.cuda4tf.img> nvidia-smi
Mon Dec 12 14:17:55 2016
+------------------------------------------------------+
| NVIDIA-SMI 352.39     Driver Version: 352.39         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K20Xm         Off  | 0000:20:00.0     Off |                    0 |
| N/A   24C    P8    30W / 235W |     12MiB /  5759MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K20Xm         Off  | 0000:8B:00.0     Off |                    0 |
| N/A   24C    P8    30W / 235W |     12MiB /  5759MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Singularity.cuda4tf.img> python -m tensorflow.models.image.mnist.convolutional
I 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
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
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
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: Tesla K20Xm
major: 3 minor: 5 memoryClockRate (GHz) 0.732
pciBusID 0000:20:00.0
Total memory: 5.62GiB
Free memory: 5.54GiB
W tensorflow/stream_executor/cuda/cuda_driver.cc:572] creating context when one is currently active;
existing: 0x4c25cb0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 1 with properties:
name: Tesla K20Xm
major: 3 minor: 5 memoryClockRate (GHz) 0.732
pciBusID 0000:8b:00.0
Total memory: 5.62GiB
Free memory: 5.54GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:59] cannot enable peer access from device ordinal 0 to
```

```
device ordinal 1
I tensorflow/core/common_runtime/gpu/gpu_init.cc:59] cannot enable peer access from device ordinal 1 to
device ordinal 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0 1
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y N
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 1:   N Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0,
name: Tesla K20Xm, pci bus id: 0000:20:00.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:1) -> (device: 1,
name: Tesla K20Xm, pci bus id: 0000:8b:00.0)
Initialized!
Step 0 (epoch 0.00), 205.4 ms
....
```

Building your own Container

This section presumes that you have root authority on a Linux / MAC workstation and that you have Singularity installed.

You can follow the instructions at the Sylabs web site: https://sylabs.io/docs/
Or
Use HPC Container Maker from Nvidia: https://devblogs.nvidia.com/making-containers-easier-with-hpc-container-maker/
This may be easier as it lets you build a recipe without having to know all the syntax. You just include the building blocks that you need, like Cuda and Infiniband, and it will create the Singularity recipe that you will use to build the container. Again, the build is on your workstation.

The resulting containers can be run on HPC using the instructions above.

Pulling Containers from Nvidia

You can pull containers from the Nvidia registry called NGC. You can do this on HPC because you are effectively encapsulating Docker images in Singularity and do not need root privilege. Here are the instructions:
https://docs.nvidia.com/ngc/ngc-user-guide/singularity.html#singularity
In brief, sign up for an NGC account, create an API key, browse the catalog, and run the *singularity build* command.

Writable Container

You might want to create a Singularity image from a Docker container but update it since the code is no longer current. There are other reasons for doing this but the feature is the same. The container will be made with the *--writable* option.

This must be done on a workstation where you have root authority. This capability is not permitted on the Ocelote login or compute nodes

```
$ sudo singularity build --writable ubuntu.simg  docker://ubuntu

Docker image path: index.docker.io/library/ubuntu:latest
Cache folder set to /root/.singularity/docker
Importing: base Singularity environment
Importing: /root/.singularity/docker/sha256:1be7f2b886e89a58e59c4e685fcc5905a26ddef3201f290b96f1eff7d778e122.
tar.gz
...
Creating empty Singularity writable container 120MB
Creating empty 150MiB image file: ubuntu.simg
Formatting image with ext3 file system
Image is done: ubuntu.simg
Building Singularity image...
Singularity container built: ubuntu.simg
Cleaning up...

$ singularity shell ubuntu.simg

Singularity: Invoking an interactive shell within container...

Singularity ubuntu.simg:~> apt-get update

Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
...
Fetched 23.2 MB in 4s (5569 kB/s)
Reading package lists... Done
```

You will need to have root authority to execute this function since root owns the container and you are modifying it. So for example you might have to use sudo:

```
sudo singularity shell ubuntu.simg
```

MPI

Singularity supports MPI fairly well. Since (by default) the network is the same insde and outside the container, the communication between containers usually just works. The more complicated bit is making sure that the container has the right set of MPI libraries. MPI is an open specification, but there are several implementations (OpenMPI, MVAPICH2, and Intel MPI to name three) with some non-overlapping feature sets. If the host and container are running different MPI implementations, or even different versions of the same implementation, hilarity may ensue.

The general rule is that you want the version of MPI inside the container to be the same version or newer than the host. You may be thinking that this is not good for the portability of your container, and you are right. Containerizing MPI applications is not terribly difficult with Singularity, but it comes at the cost of additional requirements for the host system.

```
BootStrap: debootstrap
OSVersion: xenial
MirrorURL: http://us.archive.ubuntu.com/ubuntu/


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


%post
    echo "Hello from inside the container"
    sed -i 's/$/ universe/' /etc/apt/sources.list
    apt update
    apt -y --allow-unauthenticated install vim build-essential wget gfortran bison libibverbs-dev libibmad-
dev libibumad-dev librdmacm-dev libmlx5-dev libmlx4-dev
    wget http://mvapich.cse.ohio-state.edu/download/mvapich/mv2/mvapich2-2.1.tar.gz
    tar xvf mvapich2-2.1.tar.gz
    cd mvapich2-2.1
    ./configure --prefix=/usr/local
    make -j4
    make install
    /usr/local/bin/mpicc examples/hellow.c -o /usr/bin/hellow
```

In the example above the infiniband pieces are installed and then the MVAPICH version of MPI. When the job is run the script will need to load the correct module with the matching version of MVAPICH

```
module load mvapich2/gcc/64
```