- Bootstrapping a Singularity Image
- Using docker2singularity Container
- Running a Singularity Image on the Cluster
- Building a Singularity Image with GPUs Enabled

Singularity is a tool that allows containers (including those converted from Docker) to be run within a shared high-performance computing environment. This enables users to control the OS environment (including software libraries) that their jobs run within. For example, if a user wishes to run within an Ubuntu 16.04 environment he or she may do so despite the fact that the OS on the ACCRE cluster is a completely different Linux distribution (i.e. CentOS)! Docker containers themselves cannot be run in a shared environment like the ACCRE cluster for security reasons. However, Singularity enables a user to convert a Docker container image into a Singularity container image, which can then be run on the cluster. When running within a Singularity container, a user has the same permissions and privileges that he or she would have outside the container. A Singularity image generally must first be developed and built from a Linux machine where you have administrative access (i.e. a personal machine), although ACCRE makes standard images available to all cluster users at `/scratch/singularity-images` . If you do not have administrative access to a Linux machine, you can create a virtual Linux machine using a free tool like VirtualBox . A user's cluster storage may be accessed from within the Singularity container, but no operations (e.g. the installation of system software) that require root/sudo privileges are allowed within the context of the Singularity container when run from the cluster. If you are interested in using Singularity but need assistance creating a custom image to run on the cluster, please schedule an appointment via our Helpdesk. Below are some basic instructions for running Singularity on the cluster. The Singularity documentation is very helpful, so we suggest you invest some time reading through it as well.

Bootstrapping a Singularity Image

Once you have installed Singularity on your own Linux machine or virtual machine , you are ready to create your image. First, create a spec file called ubuntu14-accre.def that looks like the following:

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

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

    %post
        apt-get -y install python3 python3-numpy python3-scipy
```

```
    # install any other software you need here...
      mkdir /scratch /data /gpfs20 /gpfs21 /gpfs22 /gpfs23
```

In this file, we are telling Singularity we want to build an image based on the latest version of Ubuntu Trusty (version 14.04). The ***%runscript*** section is for defining commands or tasks that you want to run each time you run a container of this image. The ***%post*** section contains one-time setup steps that you want to be inside the image. This is where you install your custom software needs. In this case, we use apt-get to install Python 3 with NumPy and SciPy included. Finally, it's important if you want to access all your cluster space from within the container to create the following directories from within the container: /scratch, /data, /gpfs20, /gpfs21, /gpfs22, and /gpfs23. If you don't make these directories within the container (inside the ***%post*** section), you will get a warning message when you run a container of this image on the cluster. To bootstrap this image, run the following command on your Linux machine where you have admin rights:

```
# on your personal Linux machine
sudo singularity create ubuntu14-accre.im
sudo singularity bootstrap ubuntu14-accre.im ubuntu-accre.def
```

The first command will create an empty image with a default size (768 MiB, at the time this page was written). If you need a larger image containing lots of custom software, it is prudent to pass the –size option and specify the size in MiB (e.g. *sudo singularity create –size 2048 ubuntu14-accre.im*). The second command creates your custom image based on the spec file you created above.

Using docker2singularity Container

Singularityware provides a Docker image expressly for converting Docker images to singularity images. This is especially useful for non-Linux users who do have Docker installed on their local machine. To convert the `ubuntu:14.04` image from DockerHub to a singularity image, simply run

```
docker run \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v path/to/my/singularity/images:/output \
  --privileged -t --rm \
  singularityware/docker2singularity \
  ubuntu:14.04
```

The resulting `.img` file can now be copied to the cluster for use.

Running a Singularity Image on the Cluster

Once you have successfully created your image, you can shell into it with the following command:

```
# From the cluster
setpkgs -a singularity
singularity shell ubuntu14-accre.im
```

If you plan on producing output that you want to persist outside the context of your container, run the following instead:

```
# From the cluster
setpkgs -a singularity
singularity shell --writable ubuntu14-accre.im
```

The shell subcommand is useful for interactive work. For batch processing (e.g. within a SLURM job), you should use the *exec* subcommand instead. For example:

```
setpkgs -a singularity
singularity exec ubuntu14-accre.im /path/to/my/script.sh
```

Where script.sh script contains the processing steps you want to run from within the batch job. You can also pass a generic Linux command to the *exec* subcommand. Pipes and redirected output are supported with the *exec* subcommand. Below is a quick demonstration showing the change in Linux distribution:

```
[jill@vmps10 ~]$ cat /etc/*release
CentOS release 6.8 (Final)
LSB_VERSION=base-4.0-amd64:base-4.0-noarch:core-4.0-amd64:core-4.0-noarch
CentOS release 6.8 (Final)
CentOS release 6.8 (Final)

[jill@vmps10 ~]$ setpkgs -a singularity

[jill@vmps10 ~]$ singularity shell ubuntu14-accre.img
Singularity: Invoking an interactive shell within container...

Singularity.ubuntu14-accre.img> $ cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
NAME="Ubuntu"
VERSION="14.04, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
```

```
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

Notice that the command prompt changes when you are inside the container.

```
Singularity.ubuntu14-accre.img> $ exit
[jill@vmps10 ~]$
```

Building a Singularity Image with GPUs Enabled

From NIVDIA's GitHub :

> Docker containers are often used to seamlessly deploy CPU-based applications on multiple machines. With this use case, Docker containers are both hardware- agnostic and platform-agnostic. This is obviously not the case when using NVIDIA GPUs since it is using specialized hardware and it requires the installation of the NVIDIA driver. As a result, Docker does not natively support NVIDIA GPUs with containers.

To get around this problem with Singularity, we need to install the necessary NVIDIA packages during bootstrap of our Singularity image, *taking care to match the CUDA driver version that is installed on the cluster* . At the time of writing, this is version 375.26. To install these drivers, we need to download the files to our local machine and run the normal installation process during the `setup` phase of our image.

Let's walk through the bootstrap file for tensorflow section by section, which has been adapted slightly from Singularity's definition file examples

First, we define the ubuntu:16.10 docker image as our base layer:

```
BootStrap: docker
From: ubuntu:16.10
```

Next, we need to define `%setup` , which runs outside container, prior to bootstrapping. What's important here is that we have the `NV_CUDA_FILE` , `NV_CUDNN_FILE` , and `NV_DRIVER_FILE` files downloaded and present in our working directory. The setup phase unpacks all the necessary files, creates symlinks for them, installs them in predictable locations, and finally sets some environment variables to be present inside the Singularity container.

```
%setup
    # Runs from outside the container during Bootstrap

    NV_DRIVER_VERSION=375.26
    NV_CUDA_FILE=cuda-linux64-rel-8.0.61-21551265.run
    NV_CUDNN_FILE=cudnn-8.0-linux-x64-v6.0.tgz
    NV_DRIVER_FILE=NVIDIA-Linux-x86_64-${NV_DRIVER_VERSION}.run
```

```
    working_dir=$(pwd)

    echo "Unpacking NVIDIA driver into container..."
    cd ${SINGULARITY_ROOTFS}/usr/local/
    sh ${working_dir}/${NV_DRIVER_FILE} -x
   mv NVIDIA-Linux-x86_64-${NV_DRIVER_VERSION} NVIDIA-Linux-x86_64
    cd NVIDIA-Linux-x86_64/
    for n in *.$NV_DRIVER_VERSION; do
        ln -v -s $n ${n%.$NV_DRIVER_VERSION}
    done
    ln -v -s libnvidia-ml.so.$NV_DRIVER_VERSION libnvidia-ml.so.1
    ln -v -s libcuda.so.$NV_DRIVER_VERSION libcuda.so.1
    cd $working_dir

    echo "Running NVIDIA CUDA installer..."
  sh $NV_CUDA_FILE -noprompt -nosymlink -prefix=${SINGULARITY_ROOTFS}/usr/local/cuda-8.0
  ln -r -s ${SINGULARITY_ROOTFS}/usr/local/cuda-8.0 ${SINGULARITY_ROOTFS}/usr/local/cuda

    echo "Unpacking cuDNN..."
    tar xvf $NV_CUDNN_FILE -C ${SINGULARITY_ROOTFS}/usr/local/

    ln -s /usr/local/libnvidia-ml
   ln -s /usr/lib64/libcuda.so.1 ${SINGULARITY_ROOTFS}/usr/lib64/.

    echo "Adding NVIDIA PATHs to /environment..."
    NV_DRIVER_PATH=/usr/local/NVIDIA-Linux-x86_64
    echo "

LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64:$NV_DRIVER_PATH:\$L
PATH=$NV_DRIVER_PATH:\$PATH
export PATH LD_LIBRARY_PATH

" >> $SINGULARITY_ROOTFS/environment
```

The %post phase runs inside the Singularity container, as before:

```
%post
    # Runs within the container during Bootstrap

    # Set up some required environment defaults
    export LC_ALL=C
    export PATH=/bin:/sbin:/usr/bin:/usr/sbin:$PATH

    # Install the necessary packages (from repo)
    apt-get update && apt-get install -y --no-install-recommends \
        build-essential \
```

```
        curl \
        git \
     libcupti-dev \
        libcurl4-openssl-dev \
        libfreetype6-dev \
        libpng-dev \
        libzmq3-dev \
        python-pip \
        pkg-config \
        python-dev \
        rsync \
        software-properties-common \
        unzip \
        zip \
        zlib1g-dev

     apt-get clean

     # Update to the latest pip (newer than repo)
     pip install --no-cache-dir --upgrade pip

    # Added according to this issue https://github.com/pypa/pip/issues/1064
     pip install -U setuptools

     # Install other commonly-needed packages
     pip install --no-cache-dir --upgrade \
        future \
        matplotlib \
        scipy \
        sklearn \
        jupyter

     # TensorFlow package versions as listed here:
    #  https://www.tensorflow.org/get_started/os_setup#test_the_tensorflow_installation
     #
    # Ubuntu/Linux 64-bit, GPU enabled, Python 2.7 (Requires CUDA toolkit 8.0 and CuDNN v5)
    pip install --no-cache-dir --ignore-installed --upgrade tensorflow-gpu
```

Here, we install lots of low, level tools, Python, PIP, and some specific Python packages, including Jupyter which allows us to run notebooks from inside the container (see below). Finally, tensorflow can be installed with a single pip command.

The runscript for this singularity image is:

```
%runscript
    # When executed, the container will run a Jupyter notebook on the specified port
```

```
# Check the current environment
chk_nvidia_uvm=$(grep nvidia_uvm /proc/modules)
if [ -z "$chk_nvidia_uvm" ]; then
 echo "Problem detected on the host: the Linux kernel module nvidia_uvm is not loaded"
    exit 1
fi

if [ "$#" -ne 1 ]; then
    echo "Must pass port number"
    exit 1
fi

exec jupyter notebook --ip=0.0.0.0 --port="$@"
```

Thus, the command to launch the singularity container is:

```
singularity run singularity-images/tensorflow-1.0-jupyter-gpu.img 9999
```

Putting it all together, these commands can be rolled into a SLURM batch script:

```
#!/bin/bash

#SBATCH --mem=10G  # Total memory (RAM) required, per node
#SBATCH --time=0:03:00  # Wall Clock time (dd-hh:mm:ss) [max of 14 days]
#SBATCH --account=accre_gpu
#SBATCH --partition=maxwell    # low-latency RoCE network and 4 Titan X GPUs per node
#SBATCH --gres=gpu:2             # GPUs allocated

PORT_NUM=8888

echo "This job will run a Jupyter notebook from within a Singularity image"
echo "To view the notebook, create a new ssh connection to accre with port forwarding:"
echo "ssh -N -L 9999:$hostname:$PORTNUM $USER@login.accre.vanderbilt.edu"

setpkgs -a singularity

singularity run /scratch/singularity-images/tensorflow-1.0-jupyter-gpu.img $PORT_NUM
```

To use the Jupyter notebook on your local machine, from a separate terminal window, create a secure shell connection to accre with port forwarding:

```
ssh -N -L 8888:gpu00XX:9999.
```

This will forward the port 9999 on `gpu00XX` to your localhost, so you can then open a web browser on your client machine to `localhost:9999` . Before proceeding, you may have to enter the token for the jupyter notebook, which can be found in the standard output of the `jupyter notebook` command.