Singularity on Wexac

Introduction to Singularity, basic usage.

By Igor Chebotar

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

Introduction to Containers

DOCKER	Singularity
• Daemon-based • Requires administrator privileges • Long-running services (web services, databases)	• No background daemon • • No special privileges • User-space applications (scientific software)

Introduction to Singularity

- Compatible with most Docker images
- Containers can be built on local machine and copy to cluster
- Recognize directories (mounts) and devices such as networks, work directories, GPU’s, etc.
- Supports LSF, MPI, GPU’s

Basic Singularity Commands

- **$ singularity pull** - get container images from repo's – including docker repo
- **$ singularity exec** - run command inside the container
- **$ singularity shell** – "login to" the container
- **$ singularity build** - create container from recipe

Pull and Run example

(pull singularity image to Wexac)

$ singularity pull <hub>://<image>[:<tag>]

```
(base) [igorc@access4 Singularity-test]$ singularity pull library://sylabs-jms/testing/lolcow
INFO:    Downloading library image
87.9MiB / 87.9MiB [============================================================================
 % 25.2 MiB/s 0s
WARNING: integrity: signature not found for object group 1
WARNING: Skipping container verification
(base) [igorc@access4 Singularity-test]$
```

```
(base) [igorc@access4 Singularity-test]$ singularity run lolcow_latest.sif
 ______________________________________
/ Q: Do you know what the death rate \
\ around here is? A: One per person. /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
(base) [igorc@access4 Singularity-test]$
```

Run example – BSUB

```
[igorc@access4 Singularity-test]$ bsub -q new-short -oo lolcow.out singularity run lolcow.simg
Memory reservation is (MB): 1024
Memory Limit is (MB): 1024
Job <612381> is submitted to queue <new-short>.
[igorc@access4 Singularity-test]$ cat lolcow.out
 ________________________________________
/ Beauty and harmony are as necessary to \
\ you as the very breath of life.        /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Run example – BSUB: GPU

```
[igorc@access4 Singularity-test]$ bsub -q gpu-short -gpu num=3 -Is singularity run --nv tensorflow_latest-gpu.sif
Memory reservation is (MB): 1024
Memory Limit is (MB): 1024
Job <612691> is submitted to queue <gpu-short>.
<<Waiting for dispatch ...>>
<<Starting on hgn12>>
Singularity> python
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from tensorflow.python.client import device_lib
2021-07-08 09:26:29.475560: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
>>> print(device_lib.list_local_devices())
2021-07-08 09:26:37.013889: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with properties:
pciBusID: 0000:15:00.0 name: Quadro RTX 6000 computeCapability: 7.5
coreClock: 1.77GHz coreCount: 72 deviceMemorySize: 23.65GiB deviceMemoryBandwidth: 625.94GiB/s
2021-07-08 09:26:37.015755: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 1 with properties:
pciBusID: 0000:39:00.0 name: Quadro RTX 6000 computeCapability: 7.5
coreClock: 1.77GHz coreCount: 72 deviceMemorySize: 23.65GiB deviceMemoryBandwidth: 625.94GiB/s
2021-07-08 09:26:37.017551: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 2 with properties:
pciBusID: 0000:3a:00.0 name: Quadro RTX 6000 computeCapability: 7.5
coreClock: 1.77GHz coreCount: 72 deviceMemorySize: 23.65GiB deviceMemoryBandwidth: 625.94GiB/s
2021-07-08 09:26:37.091610: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu devices: 0, 1, 2
```

Pull and Run docker image

(pull native docker image from docker hub and convert to singularity)

$ singularity pull docker://<image>[:<tag>]

```
(base) [igorc@access4 Singularity-test]$ singularity pull docker://godlovedc/lolcow
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob 9fb6c798fa41 done
Copying blob 3b61febd4aef done
Copying blob 9d99b9777eb0 done
Copying blob d010c8cf75d7 done
Copying blob 7fac07fb303e done
Copying blob 8e860504ff1e done
Copying config 73d5b1025f done
Writing manifest to image destination
Storing signatures
INFO:    Creating SIF file...
(base) [igorc@access4 Singularity-test]$ ls
cactus_latest.sif  julia-base_latest.sif  julia.sif_latest.sif  lolcow_latest.sif
(base) [igorc@access4 Singularity-test]$ singularity run lolcow_latest.sif
 ______________________________
< Many pages make a thick book. >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Singularity exec

(good for running batch jobs)

- exec command is the recommended way to run singularity containers as a batch jobs on HPC (bsub)
- Usefull options
  - --nv : Leverage GPUs – Required if you run gpu app
  - --bind /mount:/mountName : Bind mount directories to the containers
  - --env var1=value : Set your environment variables
  - --cleanenv : Clean the environment
  - --no-home : Do not mount your homefolder
  - --pwd: Initial working directory within the container
- Usage:
  - **$ singularity exec [options] image.sif command [command-args]**

```
(base) [igorc@access4 Singularity-test]$ singularity exec julia-base_latest.sif julia --version
julia version 1.6.1
```

Singularity build

- Recommended way is to build container on a local machine with ROOT. Another option is to use docker1 node where we allow building containers.
- It is possible to build the container without root, but the functionality may be poor
- Usage example:

$ singularity build lolcow.simg library://sylabs-jms/testing/lolcow

Singularity recipe example - TensorFlow

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
```

How to use Singularity on Wexac?

- To run container - simply load the Singularity module:

  $ module load Singularity

  and Singularity environment will be available.

- To build image – Use your local machine where you have root. Another option is to use docker1 host for building singularity with fakeroot option.

Singularity CHEATSHEET

singularity --version -> **verify installation**
which singularity -> **verify installation**

Doing changes or running root commands from Singularity image is prohibited on WEXAC cluster.
To build and change your Singularity image – use your local machine or docker1 server.

Use writeable image – sandbox (This is for changing your container contents/adding more programs/updating the containers)
singularity build --sandbox /tmp/debian.sandbox docker://debian:latest -> **Build a base sandbox from DockerHub**
singularity exec --writable /tmp/debian.sandbox apt-get install git-> **Make changes to it from executing command**
singularity shell --writable /tmp/debian.sandbox -> **Make changes to it using a shell connection to the image**

After you finish editing/changing your sandbox image – Create a singularity application container:
singularity build --fakeroot /tmp/debianV1.simg /tmp/debian.sandbox -> **Build your custom image from sandbox**
sudo singularity exec /tmp/mondebian.simg ls -> **Browse your custom image**

Work with non writeable image:
singularity build /tmp/ubuntusingular.simg ubuntu.def -> **Create a ready-to-go container using Singularity recipe file**
singularity exec /tmp/ubuntusingular.simg ls -> **Execute ls command from your container**
Work with writable image:
singularity build --fakeroot --sandbox /tmp/centossingular.simg centos.def -> **Create edit-able sandbox container**
singularity exec --writable /tmp/centossingular.simg touch test1.txt -> **Create file inside the edit-able container**

Just run a container from repository
singularity run shub://ajreling/Singularity-CentOS

Singularity CHEATSHEET #2

Make a Singularity image from Wexac local docker repo – ops:5000
$ singularity pull docker://ops:5000/tensorflow:v1

This will download and build a singularity image of tensorflow from ops:5000 repository directly to your homefolder.

Execute interactive job with Singularity image:
bsub -q gpu-interactive -J JOBNAME -gpu num=1:j_exclusive=yes **'module load Singularity ; singularity shell pytorch.sif '**
(Where pytorch.sif should be replaced with your singularity image name)

Run BATCH job with Singularity – pytorch example:
bsub -q gpu-short -J JOBNAME -o log_%J -e err_%J -gpu num=2:j_exclusive=yes **' module load Singularity ; singularity exec pytorch:v3.sif test_model_wexac.py "200" "new_model.py" "validation_lib_pos_binned_b200.pkl" "trained_models_b200_no_x_val" '**

Steps to prepare and run Sing container

1. Pull/build image from repo or build using singularity recipe file (From local machine or docker1 node):
   From docker hub: *singularity pull docker://Imagename:TAG*
   From Nvidia: *singularity pull docker://nvcr.io/nvidia/tensorflow:19.11-tf1-py3*
   From Singularity hub: *singularity pull library://Imagename*
   From recipe file: *singularity build myimage.sif recipe.def*
2. Confirm your container is running properly
3. Upload your container file to your homefolder on Wexac
4. Execute your container as a bsub job(Do not forget to add relevant flags - bind , env , nv , etc…):
   Interactive: *bsub -q gpu-interactive -J JOBNAME -gpu num=1:j_exclusive=yes 'module load Singularity ; singularity shell pytorch.sif '*
   Batch: *bsub -q gpu-short -J JOBNAME -o log_%J -e err_%J -gpu num=2:j_exclusive=yes ' module load Singularity ; singularity exec pytorch:v3.sif test_model_wexac.py "200" "new_model.py" "validation_lib_pos_binned_b200.pkl" "trained_models_b200_no_x_val" '*

Troubleshooting

1. **no space left on device when pulling an image:**
   Singularity is trying to use the default /tmp folder that may run out of space.
   To fix it – define temp dir under your homefolder instead:

```
module load Singularity
mkdir =~/tmp/singularity_cache
export SINGULARITY_TMPDIR=~/tmp/singularity_cache
singularity pull docker://IMAGE:VERSION
```

2. **Container build/run fail because of different memory/resource errors:**

You are probably trying to build container under one of the access nodes which now has a resource usage limitation to prevent overloading those nodes.

The solution is to BUILD singularity images on your local machine or docker1 node and RUN the containers as bsub jobs under compute nodes.

THANKS!

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE