Using Containers on the Cannon Cluster: Singularity

Plamen Krastev, PhD - plamenkrastev@fas.harvard.edu
Computational Scientist and Research Consultant

Objectives

- To advise you on the best practices for running *Singularity* containers on the FASRC cluster
- To provide the basic knowledge required for building your own (*Singularity)* containers

Overview

- What are containers and why we care? (overview)
- Singularity container system
- How to run Singularity containers on Cannon:
  - Pulling from *docker* registry or *sylab* library
  - Using GPUs
  - Running local images
- How to build your own containers
- Bind mounts
- Running parallel multicore (OpenMP) and distributed (MPI) applications

What problems are we are trying to solve?

Deploying Applications:

Building software is often a complicated business, particularly on HPC and other multi-tenant systems:

- HPC clusters have typically very specialized software stacks which might not adapt well to general purpose applications.

- OS installations are streamlined

- Some applications might need dependencies that are not readily available and complex to build from source.

- End users use Ubuntu or Arch, cluster typically use RHEL, or SLES, or other specialized OS.
(... "`$ sudo apt-get install` " will not work)

What problems are we are trying to solve?

Portability and Reproducibility:

- Running applications on multiple systems typically needs replicating the installations multiple times making it hard to keep consistency.
- It would be useful to publish the exact application used to run a calculation for reproducibility or documentation purpose.
- As a user can I minimize the part of the software stack I have no control on, to maximize reproducibility without sacrificing performance too much?

What problems are we are trying to solve?

Resource Contention and Security:

- Tasks on a normal OS float between cores and memory space.
- Want to set a cap on usage for multiple tenants.
- Ensure users cannot see other users’ applications and stacks

Containers: easi"er" software deployment

Containers provide a potential solution…. or at the very least can help.

- ❑ Easier software deployment:
  Users can leverage on installation tools that do not need to be available natively on the runtime host
  (e.g., package managers of various Linux distributions).

- ❑ Software can be built on a platform different from the execution hosts.

- ❑ They package in one single object all necessary dependencies.

- ❑ Easy to publish and sign

- ❑ They are portable **
  - … provided you run on a compatible architecture)
  - access to special hardware needs special libraries also inside the container, which at the moment limits portability

Types of Containers

- cgroups (control groups)
- python/conda environment
- Docker-like containers (e.g., rkt, podman, Linux LXC)
- Virtual Machines (VM)

Types of Containers

docker
Linux LXC
podman
OpenVZ Linux Containers
rkt

Charliecloud
SHIFTER

General purpose / Microservice Oriented

HPC oriented:

- Compatible with WLM
- No privilege escalation needed

Singularity (https://sylabs.io)

Singularity provides a container runtime and an ecosystem for managing images that is suitable for multi-tenant systems and HPC environments.

Important aspects:

- no need to have elevated privileges at runtime, although root privileges are needed to build the images.
- each applications will have its own container
- containers are not fully isolated (e.g., host network is available)
- users have the same *uid* and *gid* when running an application
- containers can be executed from local image files, or pulling images from a docker registry, a singularity hub or from *sylab* libraries (see https://cloud.sylabs.io … N.B. service is still in alpha)

For basic usage refer to:

https://www.rc.fas.harvard.edu/resources/documentation/software/singularity-on-odyssey/
https://www.sylabs.io/docs/

Example: running from a docker registry

Running tensorflow on a CPU node:

```
# --- Start an interactive session ---
[login-node  ]$ salloc -p test --mem=4G -N 1 -t 60
# --- cd to your SCRATCH folder ---
[compute-node]$ cd $SCRATCH/your_lab/your_user/
# --- Pull the latest TF version from the Docker registry ---
[compute-node]$ singularity pull --name tf27_cpu.simg   \
> docker://tensorflow/tensorflow:latest
# --- Launch Python and print the TF version ---
[compute-node]$ singularity exec tf27_cpu.simg python
… (omitted output)
>>> import tensorflow as tf
>>> print(tf.__version__)
2.7.0
# --- Get examples from keras.io ---
[compute-node]$ git clone https://github.com/keras-team/keras-io.git
# --- Execute the code ---
[compute-node]$ singularity exec tf27_cpu.simg python  \
./keras-io/examples/vision/mnist_convnet.py
… (omitted output)
Test loss: 0.026334384456276894
Test accuracy: 0.9904999732971191
```

Example: running from a docker registry

Running tensorflow on a GPU node:

```
# --- Start an interactive session on a partition with GPUs, e.g.,
[login-node  ]$ salloc -p gpu_test --gres=gpu:1 --mem=4G -N 1 -t 60
# --- cd to your SCRATCH folder ---
[compute-node]$ cd $SCRATCH/your_lab/your_user/
# --- Pull the latest TF GPU version from the Docker registry ---
[compute-node]$ singularity pull --name tf27_gpu.simg   \
> docker://tensorflow/tensorflow:latest-gpu
# --- Get examples from keras.io ---
[compute-node]$ git clone https://github.com/keras-team/keras-io.git
# --- Execute the code ---
[compute-node]$ singularity exec --nv tf27_gpu.simg python  \
./keras-io/examples/vision/mnist_convnet.py
… (omitted output)
Test loss: 0.024948162958025932
Test accuracy: 0.9915000200271606
```

Example: pulling images from repositories

Preparation (start an interactive session and cd to $SCRATCH directory):

```
[login-node  ]$ salloc -p test --mem=4G -N 1 -t 60
[compute-node]$ cd $SCRATCH/your_lab/your_user/
```

Pulling from Docker:

```
[compute-node]$ singularity pull docker://tensorflow/tensorflow:latest
```

Pulling from shub:

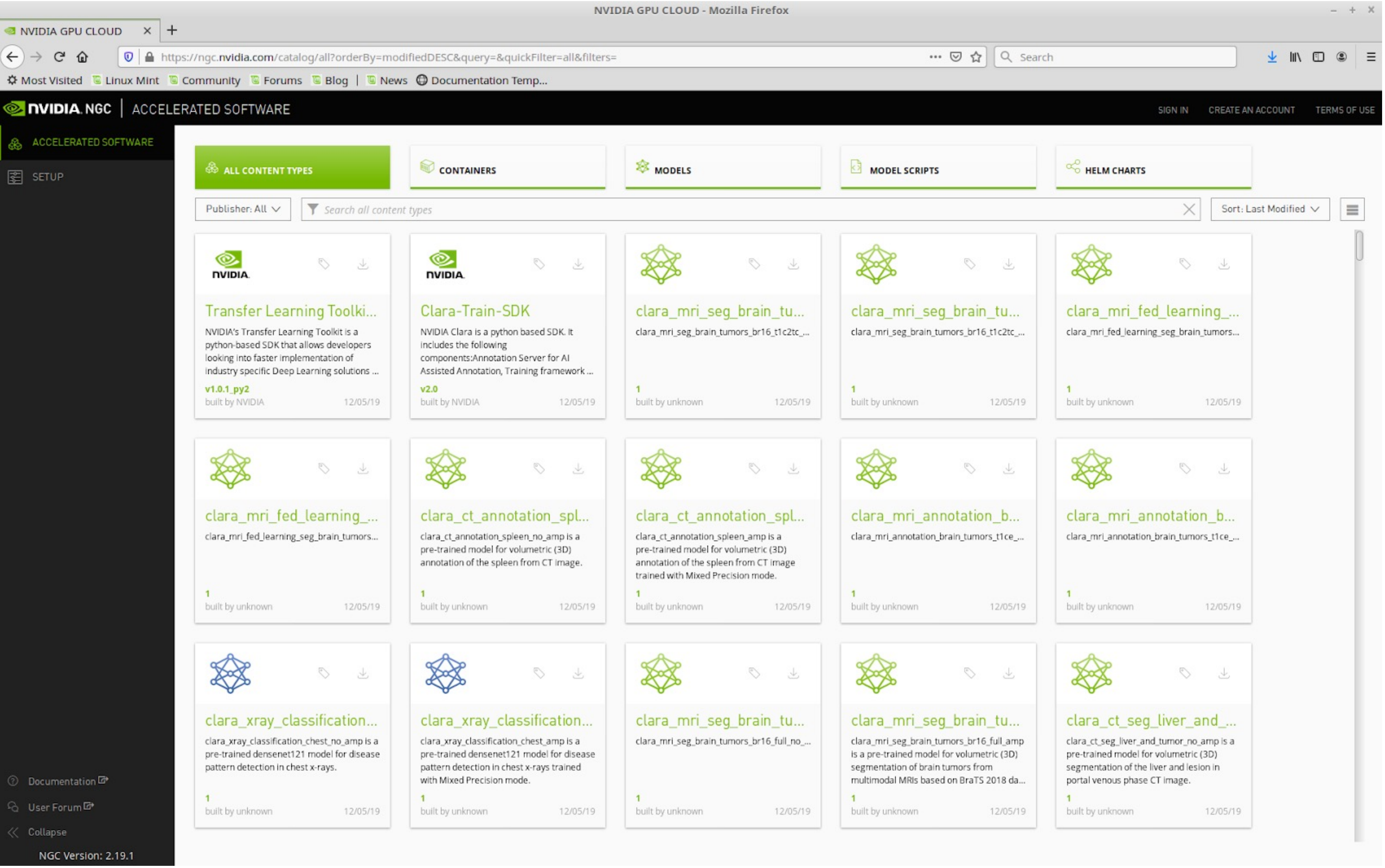
```
[compute-node]$ singularity pull shub://vsoch/hello-world
```

Pulling from sylab / library -- https://cloud.sylabs.io/library

```
[compute-node]$ $ singularity pull library://library/default/ubuntu:21.04
```

Pulling from NVIDIA's NGC registry - https://catalog.ngc.nvidia.com

```
[compute-node]$ singularity pull docker://nvcr.io/nvidia/tensorflow:21.10-tf2-py3
[compute-node]$ singularity exec tensorflow_21.10-tf2-py3.sif python
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> print(tf.__version__)
2.6.0
```


NVIDIA GPU CLOUD - Mozilla Firefox
NVIDIA GPU CLOUD
https://ngc.nvidia.com/catalog/all?orderBy=modifiedDESC&query=&quickFilter=all&filters=
Most Visited
Linux Mint
Community
Forums
Blog
News
Documentation Temp...
NVIDIA NGC | ACCELERATED SOFTWARE
SIGN IN
CREATE AN ACCOUNT
TERMS OF USE
ACCELERATED SOFTWARE
SETUP
ALL CONTENT TYPES
CONTAINERS
MODELS
MODEL SCRIPTS
HELM CHARTS
Publisher: All
Search all content types
Sort: Last Modified
Transfer Learning Toolki...
NVIDIA's Transfer Learning Toolkit is a python-based SDK that allows developers looking into faster implementation of industry specific Deep Learning solutions ...
v1.0.1_py2
built by NVIDIA 12/05/19
Clara-Train-SDK
NVIDIA Clara is a python based SDK. It includes the following components:Annotation Server for AI Assisted Annotation, Training framework ...
v2.0
built by NVIDIA 12/05/19
clara_mri_seg_brain_tu...
clara_mri_seg_brain_tumors_br16_t1c2tc_...
clara_mri_fed_learning_...
clara_mri_fed_learning_seg_brain_tumors...
clara_ct_annotation_spl...
clara_ct_annotation_spleen_no_amp is a pre-trained model for volumetric (3D) annotation of the spleen from CT image.
clara_ct_annotation_spleen_amp is a pre-trained model for volumetric (3D) annotation of the spleen from CT image trained with Mixed Precision mode.
clara_mri_annotation_b...
clara_mri_annotation_brain_tumors_t1ce_...
clara_xray_classification...
clara_xray_classification_chest_no_amp is a pre-trained densenet121 model for disease pattern detection in chest x-rays.
clara_xray_classification_chest_amp is a pre-trained densenet121 model for disease pattern detection in chest x-rays trained with Mixed Precision mode.
clara_mri_seg_brain_tumors_br16_full_no_...
clara_mri_seg_brain_tumors_br16_full_amp is a pre-trained model for volumetric (3D) segmentation of brain tumors from multimodal MRIs based on BraTS 2018 da...
clara_ct_seg_liver_and_...
clara_ct_seg_liver_and_tumor_no_amp is a pre-trained model for volumetric (3D) segmentation of the liver and lesion in portal venous phase CT image.
built by unknown 12/05/19
Documentation
User Forum
Collapse
NGC Version: 2.19.1

Example: running from a local image

Running IDL:

```
[login-node  ]$ salloc -p test --mem=4G -N 1 -t 60
[compute-node]$ cd $SCRATCH/your_lab/your_user/
[compute-node]$ myimage=/n/helmod/apps/centos7/Singularity/IDL/idl-8.7.2.sif
[compute-node]$ singularity exec $myimage idl
IDL 8.7.2 (linux x86_64 m64).
(c) 2019, Harris Geospatial Solutions, Inc.

Licensed for use by: Harvard University (MAIN)
License: 216887
A new version is available: IDL 8.8.1
https://harrisgeospatial.flexnetoperations.com

IDL>
```

RC Portal - Mozilla Firefox

RC Portal

https://portal.rc.fas.harvard.edu/p3/build-reports/

Most Visited | Linux Mint | Community | Forums | Blog | News | Documentation Temp...

Portal

Applications

Search

Search

Select from the available application types

Singularity 3

Cactus

Cactus is a reference-free whole-genome multiple alignment program

cactus 2019-11-29

Please see detailed instructions for the use of this cactus image on the FAS Informatics website.

To activate this build:

```
singularity exec --cleanenv /n/singularity_images/informatics/cactus/cactus:2019-11-29.sif --binariesMode local jobStore "${SEQFILE}" "${OUTPUTHAL}"
```

Cell Ranger ATAC

Cell Ranger ATAC is a set of analysis pipelines that process Chromium Single Cell ATAC data

Cache folder

When using images generated from remote sources singularity will cache layers and converted images under `~/.singularity`

```
[pkrastev@holygpu2c0703 Examples]$ singularity cache list
There are 5 container file(s) using 3.70 GiB and 53 oci blob file(s) using 3.67 GiB of
space
Total space used: 7.36 GiB

[pkrastev@holygpu2c0703 Examples]$ ls -lh ~/.singularity/cache/oci-tmp/
total 4.1G
-rwxr-xr-x 1 pkrastev rc_admin 380M Nov  7 14:13
31e09cf438a41f12c759cc8cc79c6b0fbb0db5abfc3de8169e916c8c9ac38dc5
-rwxr-xr-x 1 pkrastev rc_admin 716M Nov  7 23:13
a85971e31b430779c8fd19496c08f84122a9ebbcbe89ce32ddd729d37cdb1def
-rwxr-xr-x 1 pkrastev rc_admin 2.6G Nov  7 21:22
fc5eb0604722c7bef7b499bb007b3050c4beec5859c2e0d4409d2cca5c14d442

[pkrastev@holygpu2c0703 Examples]$ singularity cache clean
This will delete everything in your cache (containers from all sources and OCI blobs).
Hint: You can see exactly what would be deleted by canceling and using the --dry-run
option.
Do you want to continue? [N/y]
```

You can control the location of the variable `SINGULARITY_CACHEDIR`
https://sylabs.io/guides/3.7/user-guide/build_env.html

Running cluster jobs

```
#!/bin/bash
#SBATCH -J singularity_test
#SBATCH -n 1
#SBATCH -p test
#SBATCH --mem=4G
#SBATCH -t 0-08:00

singularity run my_image.sif

## OR

singularity exec my_image.sif my_command
```

Build your first container

Container images can be built using a (definition) file that specifies the recipe, e.g.,

```
$ cat Singularity.def
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-get -y update
    apt-get -y install vim
    apt-get clean
```

https://sylabs.io/guides/3.8/user-guide/definition_files.html

Build your first container

Once you have your singularity definition file you have 3 options to build your image:

(1) Build locally
To do this you need to be on your own development environment where you have admin / root privileges, e.g., personal PC (you will need to install singularity first)

```
[my_computer]$ singularity build some_image_name.sif  Singularity.def
```

(2) Build remotely
You can do it on Cannon, but you need to have an account on https://cloud.sylabs.io get a token and store it in `$HOME/.singularity/sylabs-token`

```
[login-node  ]$ salloc -p test --mem=4G -N 1 -t 60
[compute-node]$ cd $SCRATCH/your_lab/your_user/
[compute_node]$ singularity build --remote \
> some_image_name.sif  Singularity.def
```

This will create your def file, build the image and download it to the local folder.

Build your first container

Once you have your singularity definition file you have 3 options to build your image:

(3) Build in Docker (locally)

You can build an image in docker locally on your machine. This has the advantage of faster iteration.

Export to dockerhub or use `docker2singularity`
https://github.com/singularityhub/docker2singularity

Pull image to cluster in singularity, or scp it and use.

Bind Mount

- By default, all directories in the Singularity image are read only.
  - **Note:** When building from Docker, sometimes Docker expects something to be writable that may not be in Singularity.

- In addition, system directories are not available, only those defined in the Singularity image.

- You can bind external mounts into singularity using the `-B/--bind` option
  - `-B hostdir:containerdir`
  - `-B hostdir  # maps it to same path inside the container`

Example:

```
$ ls /data # on the host machine
bar  foo

# inside the container
$ singularity exec --bind /data:/mnt my_container.sif ls /mnt
bar  foo
```

On Cannon, we automatically map `/n`, `/net`, and `/scratch` into the image using bind mount.

https://sylabs.io/guides/3.8/user-guide/bind_paths_and_mounts.html

OpenMP applications

```
Bootstrap: docker
From: ubuntu:18.04

%setup
    mkdir ${SINGULARITY_ROOTFS}/opt/bin

%files
    omp_dot.c /opt/bin

%environment
    export PATH="/opt/bin:$PATH"

%post
    echo "Installing required packages..."
    apt-get update && apt-get install -y bash gcc gfortran

    echo "Compiling the application..."
    cd /opt/bin
    gcc -fopenmp -o omp_dot.x omp_dot.c
```

https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers/OMP_Apps

OpenMP applications

```
#!/bin/bash
#SBATCH -J omp_dot
#SBATCH -o omp_dot.out
#SBATCH -e omp_dot.err
#SBATCH -t 0-00:30
#SBATCH -p test
#SBATCH -N 1
#SBATCH -c 4
#SBATCH --mem=4000

PRO=omp_dot

# Run program
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK singularity exec omp_dot.simg \
/opt/bin/omp_dot.x | sort > ${PRO}.dat
```

MPI applications

There are several ways of running MPI applications with singularity
https://sylabs.io/guides/3.8/user-guide/mpi.html

We recommend **hybrid** mode (application from container, `mpirun` from host)

> **Note:** MPI flavors & versions and compile options on the host and in the container need to **match exactly** for best performance.

```
#!/bin/bash
#SBATCH -p test
#SBATCH -n 8
#SBATCH -J mpi_test
#SBATCH -o mpi_test.out
#SBATCH -e mpi_test.err
#SBATCH -t 30
#SBATCH --mem-per-cpu=1000

# --- Set up environment ---
module load gcc/10.2.0-fasrc01
module load openmpi/4.1.0-fasrc01

# --- Run the MPI application in the container ---
srun -n 8 --mpi=pmix singularity exec openmpi_test.simg /home/mpitest.x
```

https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers/MPI_Apps

BIG THANKS TO ^^^

Plamen Krastev – Computational Scientist and Research Consultant