Apptainer

Paul Preney, OCT, M.Sc., B.Ed., B.Sc.
preney@sharcnet.ca

School of Computer Science
University of Windsor
Windsor, Ontario, Canada

Apr. 6, 2022

Table of Contents

What is Apptainer?

Apptainer:

- is software that was **first developed** by **Lawrence Berkeley National Lab**
- was created to **run Linux applications** on HPC clusters **portably** and **reproducibly**
- was originally called **Singularity**, now called **Apptainer** (Nov. 30, 2021)
- it is installed on **many** HPC and academic clusters
- it is **open-source** and **secure**
- as a way to **enable** users to have **full control of their environment**, and,
- **URL:** `https://apptainer.org`

What is Apptainer?

Apptainer enables the use of **containers** which **virtualize** the **operating system**.

A **container** is not a **virtual machine**. Unlike virtual machines:

- containers have **very little overhead**, and,
- can **only** use the **same operating system** inside the container as the computer runs itself.
  - i.e., Linux

What is Apptainer?

Apptainer:

- was designed to enable containers to be used securely on multi-user HPC systems without requiring special permissions
- is the only container technology supported on our clusters

Docker is another popular container technology that is commonly used.

- Docker has a significant possible security issues making it not ideal for use on our clusters.
- Apptainer can build images from Docker images.

Table of Contents

Installing Apptainer

On our clusters:

- Singularity v3.8 is already installed.
  - `module load singularity`
- Apptainer v1.0 will be installed soon.
  - `module load apptainer`

You can also install Apptainer on your own computer:

- on a **Linux** system,
- in a *virtual machine* running **Linux**, or,
- within *Vagrant* for Windows or MacOS.
- **URL:** `https://apptainer.org/docs/admin/1.0/installation.html`

Table of Contents

Loading Apptainer

To use Apptainer on our systems, **load the module** for it:

- for Apptainer: `module load apptainer`
- for Singularity: `module load singularity`

After it is installed on our clusters, when you **switch to using Apptainer**, do the following:

- **rename** the `singularity` command to be `apptainer`,
- **rename all** `SINGULARITY_*` environment variables to be `APPTAINER_*`, and,
- **rename all** `SINGULARITYENV_*` environment variables to be `APPTAINERENV_*`.

Loading Apptainer (con't)

If you are running MPI programs inside your container…

- Be sure to **load MPI version 3 or higher *before* using your container.** MPI version 4 or higher is recommended.
  - e.g., `module load openmpi/4.1.1`
- When submitting a Slurm `sbatch` job script **use `srun` to run your MPI program.**
  - i.e., do not use `mpirun` or `mpiexec`
  - e.g., `srun apptainer run image.sif /path/to/your-mpi-program`
  - e.g., `srun singularity run image.sif /path/to/your-mpi-program`

Table of Contents

Container Images

To use Apptainer you need to already **have, obtain, or create** an **image** file.

- Inside an image file is a **filesystem containing everything needed** to **run programs inside your container.**
- Each image is an **installation** of a **Linux distribution** with some **tools/software.**
- The image does **not need** the operating system **kernel** or **boot loader** software installed.

Container Images (con't)

Apptainer allows you to:

- build an image from **Docker Hub** (`https://hub.docker.com/`),
- use a container image file you already have access to,
  - `*.sif`: default format (read-only); Singularity version v3+ / Apptainer
  - `*.sqsh`: SquashFS (read-only); Singularity version 2.4+
  - `*.img`: EXT3 (read-write); oldest format
- build an image in a **sandbox directory**, and,
- create an image using a **definition file**.
  - This is beyond the scope of this presentation.

Container Images (con't)

Some examples of building a SIF image files from Docker Hub:

- **CentOS:** `apptainer build centos-latest.sif docker://centos:latest`
- **CentOS v8:** `apptainer build centos8.sif docker://centos:8`
- **Debian:** `apptainer build debian-latest.sif docker://debian:latest`
- **Debian v10:** `apptainer build debian10.sif docker://debian:10`
- **Ubuntu:** `apptainer build ubuntu-latest.sif docker://ubuntu:latest`
- **RAPIDS:** `apptainer build rapids.sif docker://nvcr.io/nvidia/rapidsai/rapidsai:cuda11.0-runtime-centos7`
- etc.

Docs: `http://apptainer.org/docs/user/1.0/build_a_container.html`

Container Images (con't)

WARNING: Building images can fail!

- Use a **fast local hard drive** —not a networked file system:
  - This may or may not be possible on our clusters.
    - `/localscratch`, if present, *might* be a local hard drive.
    - Also Slurm doesn't set `$SLURM_TMPDIR` in interactive jobs so such cannot be used.
  - If you have Apptainer installed on your own system, build the image there.
  - If you also have root permissions on your own system, consider prefixing the build command with `sudo`.
    - e.g., `sudo apptainer build ubuntu-latest.sif docker://ubuntu:latest`
- Ensure you have Internet access if downloading an image:
  - On clusters with no Internet access on compute nodes, you must build your image using a login node.
  - If the cluster you are using permits Internet access from a compute node, use an interactive job session instead.

If you have problems building an image, submit a ticket asking for help.

Table of Contents

Using Apptainer: Overview

There are a number of ways to use Apptainer:

1. Run a **single command** which executes and then stops running.
   - `apptainer run --nv rapids.sif my_script.sh`
2. Run **many commands** in an interactive session.
   - `apptainer shell --nv rapids.sif`
3. Run a container instance to run **daemons** and have **backgrounded processes**.
   - `apptainer instance start image.sif a_name`
   - `apptainer run image.sif instance://a_name ps -eaf`
   - `apptainer run image.sif instance://a_name nohup find / -type d >dump.txt`
   - `apptainer run image.sif instance://a_name ps -eaf`
   - `apptainer instance stop image.sif a_name`

Using Apptainer on Our Clusters

When using the `run`, `shell`, `instance`, and `exec` Apptainer commands:

- Always use one of the `-C`, `-c`, or `-e` options.
  - **`-C`**: hides filesystems, PID, IPC, and environment
    - Prefer this option.
    - Requires explicitly bind mounting all paths needed and explicitly passing desired environment variables to the container (or they cannot be seen).
  - **`-c`**: uses minimal `/dev`, shared-with-host directories will appear empty, e.g., `/tmp`, unless explicitly bind mounted
  - **`-e`**: clean environment before running container

NOTE: You don't want things from your shell's environment to be present when using your container: many things are not inside your container and therefore cannot be accessed.

Using Apptainer on Our Clusters (con't)

When using the `run`, `shell`, `instance`, and `exec` Apptainer commands:

- Always use the `-W dir` option with `dir` being a path to a real directory that you have write-access to.
  - Not doing this can result in programs failing to run / crashing.
  - In `sbatch` scripts, set `dir` to `$SLURM_TMPDIR`.
- When using NVIDIA GPUs, use `--nv` to expose the NVIDIA hardware devices to the container.
- When access to host directories is needed, bind mount the top-level directories of those filesystems, or, the desired directories themselves.
  - e.g., `apptainer run --nv -B /home image.sif -W $SLURM_TMPDIR some_program`
    - Useful bind mounts: `-B /home -B /project -B /scratch`

Table of Contents

Examples: RAPIDS (Using a ”newhome” Bind Mounted Directory)

Help Wiki URL: `https://docs.computecanada.ca/wiki/RAPIDS`.

Step 1: Build a RAPIDS' SIF file in `/localscratch`:

```
export WORKDIR=/localscratch/tmp/$HOME
mkdir -v -p $WORKDIR
export APPTAINER_TMPDIR=$(mktemp -d -p $WORKDIR tmpdir-XXXXXXXXXX)
export APPTAINER_CACHEDIR=$WORKDIR/cachedir
mkdir -v -p $APPTAINER_CACHEDIR $APPTAINER_TMPDIR
module load apptainer
apptainer build $HOME/rapids.sif \
  docker://nvcr.io/nvidia/rapidsai/rapidsai:cuda11.0-runtime-centos7
rm -rf $WORKDIR
```

NOTE: The above steps might run out of memory and fail. If so use a different cluster to build the image, or, submit a ticket asking for help.

Examples: RAPIDS (Using a ”newhome” Bind Mounted Directory) (con’t)

Step 2: Configure the image using an interactive session

```
# Adjust the next line per your resource needs...
salloc --ntasks=1 --cpus-per-task=2 --mem=10G --gres=gpu:t4:1 --time=1:0:0 \
  --account=def-someuser
cd
module load apptainer
mkdir -p -v a-temp-dir newhome
export APPTAINERENV_NEWHOME=$(pwd)/newhome
apptainer shell -C --nv -B /home -B /project -B /scratch \
  -W ./a-temp-dir rapids.sif
> export HOME=$NEWHOME
# i.e., set the HOME directory inside the container to be NEWHOME
> cd $HOME
> pwd
```

Examples: RAPIDS (Using a ”newhome” Bind Mounted Directory) (con’t)

```
> nvidia-smi
# i.e., confirm that the GPU is available
> source /opt/conda/etc/profile.d/conda.sh
> conda activate rapids
> jupyter-lab --ip $(hostname -f) --no-browser
# Start an SSH tunnel in order to connect to Jupyter Lab.
# Hit Ctrl-C when done.
> exit
# i.e., leave the Singularity/Apptainer interactive shell.
exit
# i.e., leave the Slurm interactive job.
rm -rf a-temp-dir
# i.e., a-temp-dir no longer needed.
```

Examples: RAPIDS (Using a ”newhome” Bind Mounted Directory) (con’t)

Step 3: Submitting non-interactive jobs to Slurm

First, create a script, `my_rapids_run_script.sh`, that will run *inside* the container, e.g.,

```
#!/bin/bash
export HOME=$NEWHOME
source /opt/conda/etc/profile.d/conda.sh
conda activate rapids
nvidia-smi
python /path/to/my_rapids_code.py
```

- Remember to `chmod +x my_rapids_run_script.sh`.
- Notice the script activates conda.

Examples: RAPIDS (Using a ”newhome” Bind Mounted Directory) (con’t)

Second, write an `sbatch` script that runs `my_rapids_run_script.sh`, e.g.,

```
#!/bin/bash
#SBATCH --gres=gpu:t4:1
#SBATCH --cpus-per-task=2
#SBATCH --mem=10G
#SBATCH --time=dd:hh:mm
#SBATCH --account=def-someuser
module load apptainer
export APPTAINERENV_NEWHOME=$(pwd)/newhome  # ensure newhome dir exists!
apptainer run -C --nv -B /home -B /project -B /scratch -W "$SLURM_TMPDIR" \
  rapids.sif my_rapids_run_script.sh
```

and submit the job with `sbatch`.

Examples: Spack (Using an Overlay Image File)

Spack URL: https://spack.readthedocs.io/en/latest/getting_started.html.

Step 1: Create an overlay image file.

```
module load apptainer
apptainer overlay create --size 2048 myoverlay.img
# i.e., create a 2GB overlay image file
mkdir -v -p ./temp-dir
apptainer shell -C --overlay ./myoverlay.img -W ./temp-dir your.sif
> mkdir -v -p /newhome
# NOTE: This is done inside the myoverlay.img file.
> export HOME=/newhome
> cd
> git clone -c feature.manyFiles=true https://github.com/spack/spack.git
# i.e., install Spack
```

Examples: Spack (Using an Overlay Image File) (con't)

```
> df -h
# e.g., to see the space available in the "overlay" filesystem
> . spack/share/spack/setup-env.sh
# i.e., activate Spack
> spack install nano
# e.g., install nano
# etc.
> exit
exit
rm -rf ./temp-dir
```

Examples: Spack (Using an Overlay Image File) (con't)

Step 2: Submitting non-interactive jobs to Slurm

First, create a script, `my_run_script.sh`, that will run *inside* the container, e.g.,

```
#!/bin/bash
export HOME=/newhome
. spack/share/spack/setup-env.sh
# more commands here
```

- Remember to `chmod +x my_run_script.sh`.
- Remember to set HOME and activate Spack.

Examples: Spack (Using an Overlay Image File) (con't)

Second, write an `sbatch` script that runs `my_run_script.sh`, e.g.,

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=16G
#SBATCH --time=dd:hh:mm
#SBATCH --account=def-someuser
module load apptainer
apptainer run -C -B /home -B /project -B /scratch -W "$SLURM_TMPDIR" \
  your.sif my_run_script.sh
```

and submit the job with `sbatch`.

Examples: Spack (Using an Overlay Image File) (con't)

To check your overlay image file's filesystem run:

- `e2fsck -f myoverlay.img`

To resize your overlay image file's size to 4GB:

- `resize2fs -p myoverlay.img 4G`

To shrink your overlay image file's size to be as small as possible (no free space) run:

- `resize2fs -p -M myoverlay.img`

Table of Contents

Questions

Questions.