Hands-On Workshop: Containers on Wulver

NJIT High Performance Computing

04/28/2026

Overview

In this workshop you will practice running containers on Wulver using Apptainer (the HPC-native container runtime). By the end you will be able to:

- Pull container images from Docker Hub and other registries
- Start an interactive shell inside a container
- Execute commands and scripts inside a container
- Mount host directories into a container
- Submit container jobs to SLURM
- Run GPU-accelerated containers with PyTorch

> **NOTE:** You do NOT need to build containers in this workshop. All images are pulled from public registries.

Prerequisites

- A Wulver account (ucid@wulver.njit.edu)
- Basic Linux command-line familiarity
- SSH client (Terminal on Mac/Linux, PuTTY or access shell via ondemand.njit.edu)

Part 1 — Environment Setup

Step 1 — Log in to Wulver or via ondemand.njit.edu -> Wulver shell access

```
ssh <ucid>@wulver.njit.edu
```

Step 2 — Get a compute node

Never run container workflows on the login node. Request an interactive session on a compute node. Wulver provides a convenient interactive wrapper command:

```
# Recommended: Wulver interactive wrapper (1 CPU, 1 hr default)
interactive -a <PI_ucid> -q standard -j cpu
```

To request more CPUs or longer time:

```
# -n 2 = 2 CPU cores, -t 2 = 2 hours
interactive -a <PI_ucid> -q standard -j cpu -n 2 -t 2
```

Alternative: use srun directly for more control:

```
# srun: explicit flags, 2 CPUs, 4 GB/CPU (8 GB total)
srun --partition=general --qos=standard --account=<PI_ucid> \
      --nodes=1 --ntasks-per-node=2 --mem-per-cpu=4000M \
      --time=02:00:00 --pty bash -l
```

NOTE: Replace <PI_ucid> with your PI's UCID (e.g. abc12). Run 'interactive -h' to see all options. Your prompt will change to [ucid@n0001 ~]$ when you are on a compute node.

Step 3 — Set the cache directory

Apptainer stores downloaded image layers in a cache. Point it to your scratch space to avoid filling your home quota:

```
# Add to ~/.bashrc to make permanent
export APPTAINER_CACHEDIR=/scratch/$PI/$USER/apptainer
export APPTAINER_TMPDIR=/scratch/$PI/USER/apptainer/tmp

mkdir -p $APPTAINER_CACHEDIR
mkdir -p $APPTAINER_TMPDIR
```

TIP: Replace $PI with your PI's ucid if the variable is not already set. Example: export APPTAINER_CACHEDIR=/scratch/abc12/def456

Step 4 — Load Apptainer

```
module load apptainer/1.1.9
apptainer --version
```

Expected output: apptainer version 1.1.9

Part 2 — Your First Container

Exercise 1 — Interactive shell (no local image)

The quickest way to try a container is apptainer shell. Apptainer pulls the image on-the-fly, drops you into a shell, and cleans up when you exit:

```
# Pull AlmaLinux 8 from Docker Hub and open a shell
apptainer shell docker://almalinux:8
```

Inside the container, run:

```
cat /etc/os-release
whoami
exit
```

IMPORTANT: Notice: whoami still returns your ucid — you are never root inside Apptainer. This is by design and makes containers safe for shared HPC systems.

Exercise 2 — Pull and save an image

Pull saves the image as a Singularity Image File (.sif). Re-using a local .sif is much faster than pulling each time:

```
# Saves image to almalinux8.sif
mkdir -p /project/$PI/$USER/apptainer
cd /project/$PI/$USER/apptainer
apptainer pull almalinux8.sif docker://almalinux:8
```

Verify the file was created:

```
ls -lh almalinux8.sif
```

Exercise 3 — Run the default entry point

Some images define a default run-script. Use apptainer run to execute it:

```
# Runs the image's built-in startup script
apptainer pull hello-world.sif docker://hello-world
apptainer run hello-world.sif
```

Try the classic lolcow demo — pulled from GitHub Container Registry (ghcr.io):

```
# Pull from GitHub Container Registry and run
apptainer pull lolcow.sif docker://ghcr.io/apptainer/lolcow
apptainer run lolcow.sif
```

NOTE: ghcr.io is the GitHub Container Registry. Many open-source projects publish images there using the docker:// URI prefix, just like Docker Hub. lolcow prints an ASCII cow with a random quote — a classic Apptainer/Singularity hello-world.

Part 3 — Shell and Exec

Exercise 4 — Exec a single command

apptainer exec runs one command inside the container and then exits. Compare the Python version on the host vs. inside a Python container:

```
# Host Python (may be old or missing)
python3 --version


# Python inside container
apptainer pull python312.sif docker://python:3.12-slim
apptainer exec python312.sif python3 --version
```

TIP: apptainer exec is ideal for batch scripts: wrap any command with 'apptainer exec <image> <command>'.

Exercise 5 — Inspect a container

Learn what is inside an image without running it:

```
apptainer inspect python312.sif
apptainer inspect --runscript python312.sif
apptainer inspect --environment python312.sif
```

Part 4 — Bind Mounts

Background: what is a bind mount?

By default, Apptainer automatically mounts your $HOME, /tmp, and the current working directory into the container. Other directories on the host (such as /scratch and /project) must be explicitly mounted with --bind (-B).

Exercise 6 — Mount /scratch into a container

```
# Mounts host /scratch/$PI/$USER to /data inside container
# Create a test file on the host
echo 'Hello from the host' > /scratch/$PI/$USER/test.txt
```

```
# Shell into the container with /scratch mounted
apptainer shell --bind /scratch/$PI/$USER:/data almalinux8.sif
```

Inside the container:

```
cat /data/test.txt
exit
```

Exercise 7 — Mount multiple directories

```
# Mount both scratch and project storage
apptainer shell \
  --bind /scratch/$PI/$USER:/scratch \
  --bind /project/$PI/$USER:/project \
  almalinux8.sif
```

> **TIP:** You can also set APPTAINER_BINDPATH to avoid repeating --bind flags. Example: export APPTAINER_BINDPATH=/scratch/$PI/$USER,/project/$PI/$USER

Part 5 — SLURM Batch Jobs

Background

For production workloads, submit a SLURM batch script instead of running interactively. Replace your normal command with apptainer exec <image> <command>.

Exercise 8 — Submit a batch job

Create the script:

```
cat > /project/$PI/$USER/apptainer/run_python.sh << 'EOF'
#!/bin/bash -l
#SBATCH --job-name=container_test
#SBATCH --partition=general
#SBATCH --qos=standard
#SBATCH --account=<PI_ucid>
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --mem-per-cpu=4000M
#SBATCH --time=00:30:00
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err

module load apptainer/1.1.9
```

```
export APPTAINER_CACHEDIR=/scratch/$PI/$USER

apptainer exec \
  --bind /scratch/$PI/$USER:/scratch \
  /project/$PI/$USER/apptainer/python312.sif \
  python3 -c "import sys; print('Python', sys.version)"
EOF
```

Submit and monitor:

```
sbatch /project/$PI/$USER/apptainer/run_python.sh
squeue -u $USER
cat container_test_*.out
```

Part 6 — GPU Containers

Background

Pass --nv to Apptainer to expose the host NVIDIA drivers and CUDA libraries to the container. No CUDA installation is needed inside the container — only the CUDA runtime libraries provided by the image.

Exercise 9 — Interactive GPU test

First, request a GPU node using the interactive wrapper:

```
# Recommended: 1 GPU, default CPUs and memory
interactive -a <PI_ucid> -q standard -j gpu
```

Alternative using srun directly:

```
srun --partition=gpu --qos=standard --account=<PI_ucid> \
      --nodes=1 --ntasks-per-node=4 --gres=gpu:a100_10g:1 \
      --mem-per-cpu=4000M --time=01:00:00 --pty bash
```

Load modules and pull PyTorch:

```
module load apptainer/1.1.9
export APPTAINER_CACHEDIR=/scratch/$PI/$USER
cd /project/$PI/$USER/apptainer
#use the sif I have pulled under:
Ls
#below is the command I used. For workshop, use the one I downloaded
apptainer pull docker://pytorch/pytorch:2.2.2-cuda12.1-cudnn8-runtime
```

Test GPU access:

```
# --nv passes NVIDIA drivers from host to container
apptainer exec --nv
/project/kjc59/g07396/apptainer/training/pytorch_2.2.2-cuda12.1-cudnn8-runtime.sif \
  python3 -c "import torch; print('GPUs:', torch.cuda.device_count())"
```

TIP: Expected output: GPUs: 1 (or more). If you see 0, confirm you are on a gpu-partition node and the --nv flag is present.

Exercise 10 — GPU batch job

```
cat > /project/$PI/$USER/apptainer/run_gpu.sh << 'EOF'
#!/bin/bash -l
#SBATCH --job-name=gpu_container
#SBATCH --partition=gpu
#SBATCH --qos=standard
#SBATCH --account=<PI_ucid>
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu:a100_10g:1
#SBATCH --mem-per-cpu=4000M
#SBATCH --time=01:00:00
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err

module load apptainer/1.1.9
export APPTAINER_CACHEDIR=/scratch/$PI/$USER

apptainer exec --nv \
  --bind /scratch/$PI/$USER:/scratch \
 /project/kjc59/g07396/apptainer/training/pytorch_2.2.2-cuda12.1-cudnn8-runtime.sif \
  python3 -c "
import torch
print('PyTorch:', torch.__version__)
print('CUDA available:', torch.cuda.is_available())
print('GPU name:', torch.cuda.get_device_name(0) if torch.cuda.is_available() else
'N/A')
"
EOF

sbatch /project/$PI/$USER/apptainer/run_gpu.sh
```

Quick Reference

Common Apptainer Commands

Command	Purpose
`apptainer shell <image>`	Interactive shell inside container
`apptainer exec <image> <cmd>`	Run one command inside container
`apptainer run <image>`	Run image's default entry point
`apptainer pull <name>.sif <uri>`	Download and save image as .sif
`apptainer inspect <image>`	Show metadata / run-script
`--bind <host>:<container>`	Mount a host directory
`--nv`	Enable NVIDIA GPU support

URI Schemes for apptainer pull / shell / exec

URI	Source
`docker://ubuntu:22.04`	Docker Hub
`docker://nvcr.io/nvidia/pytorch:...`	NVIDIA NGC registry
`docker://ghcr.io/apptainer/lolcow`	GitHub Container Registry (ghcr.io)
`library://lolcow`	Sylabs Cloud Library
`oras://ghcr.io/...`	OCI registries (GitHub etc.)

Key Environment Variables

```
export APPTAINER_CACHEDIR=/scratch/$PI/$USER        # cache location
export APPTAINER_BINDPATH=/scratch/$PI/$USER,/project/$PI/$USER  # auto bind mounts
```