Introduction to Containers

Hui (Julia) Zhao
NJIT High Performance Computing

Workshop Roadmap

- **Today (1-Hour Lecture)**
  - What containers are and why they matter in HPC
  - Container engines, registries, and the Apptainer toolchain
  - Key commands — pull, shell, exec, run, bind, GPU support
  - Submitting container jobs with SLURM on Wulver
- **Next Session (Hands-On)**
  - Pulling and running pre-made containers on Wulver
  - Binding directories and submitting SLURM jobs
  - Building your own container is NOT required for the hands-on

Outline

- What is a Container?
- Container Engines and Registries
- Apptainer on Wulver
- Getting Container Images
- Running Containers
- Files, Directories and Bind Mounts
- GPU Support
- Integrating with HPC Workflows

Container Concepts

What is a Container?

Containers are lightweight, isolated processes that package an application together with its entire runtime environment, allowing it to run consistently across different systems.

- **Isolation**
  - Isolates filesystem, user IDs, and network from the host and from other containers
  - Containers' OS can differ from the host OS (e.g., Ubuntu container on RHEL host)
- **How It Works**
  - Leverages the host kernel — no kernel duplication
  - From the user's perspective, the container behaves like a standard OS
  - Lightweight and fast to start compared to a full Virtual Machine

What Goes In Container Images?

Unlike VMs, the OS kernel is NOT included in the container image.

- **Virtual Machine**
  - Includes Guest OS Kernel + all system libraries
  - Managed by a hypervisor (VM Manager) layer
  - Typical image size — several gigabytes
- **Container**
  - Includes only binaries + libraries the application needs
  - Shares the Host OS Kernel — no duplication
  - Typical image size — tens to hundreds of megabytes

Containers are smaller and faster than VMs because they share the host kernel.

Why Use Containers?

- **Shareability**
  - Share .sif files directly with colleagues
  - Use community images from Docker Hub, NVIDIA NGC, BioContainers
- **Portability**
  - Same image runs on any x86_64 system — laptop, cloud, HPC cluster
  - No reinstalling software on each new machine
- **Reproducibility**
  - Container environment is unaffected by host OS updates or cluster changes
  - Freeze exact software versions for a paper or workflow
- **Reusability**
  - Build once, run anywhere — across projects and collaborators

Containers turn 'it works on my machine' into 'it works everywhere'.

Containers vs Virtual Machine

VMs virtualize hardware; containers virtualize the operating system.

- **Containers are**
  - Lighter weight — less CPU and memory, faster start-up
  - Smaller in size — easier to transfer and share
  - Modular — multiple containers can work together
- **Important Limitation**
  - Containers do NOT virtualize hardware
  - Must be built for the same CPU architecture as the host (e.g., x86_64)

Think of a container as a packaged OS environment, not a full virtual machine.

Containers vs Conda Environment

Conda environments solve software dependency issues within an environment.

Containers can encapsulate an entire Conda environment

- Ensures software, dependencies, and the OS travel together to any system
- More portable than Conda alone — works across any compute platform

Containers go further than Conda — they package the OS itself, not just Python packages.

Container Engines and Registries

Popular Container Runtimes

Instant deployment to users on different devices!

- **HPC-Optimized**
  - Singularity (2015) → Apptainer (2021, Linux Foundation fork): designed for shared HPC systems
  - Shifter (2016): NERSC/Cray HPC environments
  - Charliecloud (2017): unprivileged, minimal (LANL)
- **General Purpose**
  - LXC (2008): low-level Linux container runtime (namespaces + cgroups)
  - Docker (2013): daemon-based, typically requires root privileges
  - Podman (2018): daemonless, rootless Docker alternative

Apptainer is the go-to choice for HPC — no root daemon, runs unprivileged by default

Docker vs Singularity / Apptainer

- **Docker**
  - Users inside a container have escalated (root) privileges on the host
  - Requires a persistent root-owned daemon process
  - Not permitted on most HPC systems for this reason
- **Singularity / Apptainer**
  - Same user inside and outside the container
  - User only has root privileges if explicitly elevated with sudo outside
  - Can pull and run existing Docker and OCI containers directly

Apptainer is preferred for HPC workloads — rootless by design.

Singularity and Apptainer History

Singularity was forked Apptainer when donated to the Linux Foundation in November 2021.

- **What Changed**
  - Name changed from Singularity to Apptainer
  - Maintained by the Linux Foundation (open source, v1.0 released March 2022)
  - Sylabs continues to develop SingularityCE (commercial fork)
- **What Stayed the Same**
  - All commands are identical (apptainer provides a singularity symlink)
  - SIF image format unchanged — full compatibility
  - SINGULARITY_* environment variables still accepted (with deprecation warning)

Old scripts using the singularity command continue to work on Apptainer installations.

Apptainer Features

- **Runtime and Builder**
  - Container runtime and image builder in one tool
  - Reads and converts Docker / OCI images natively
- **Security**
  - Same user inside and outside — never root by default
  - Immutable SIF images prevent accidental or malicious modifications
- **HPC Integration**
  - Works seamlessly with MPI, GPUs, and SLURM
  - Designed for high-performance cluster technology from the start

Read more: https://apptainer.org/docs/user/main/

Container Registries

- **General Purpose**
  - Docker Hub — docker://: most popular public registry
  - Quay.io — docker://quay.io/: Red Hat / CoreOS registry
- **HPC and GPU**
  - NVIDIA NGC — docker://nvcr.io/: GPU-optimized containers for deep learning and HPC
  - Sylabs Library — library://: registry for Singularity/Apptainer SIF containers
- **Scientific**
  - BioContainers — docker://biocontainers/: bioinformatics and life science tools
  - GitHub Container Registry — docker://ghcr.io/: org-scoped OCI images on GitHub

Apptainer on Wulver

Container Module on Wulver

- Apptainer is the container platform available on Wulver. Load the Apptainer module before running any commands.
- Apptainer is intended for use on compute nodes — avoid running on login nodes (CPU-intensive tasks).

```
# Load the module
$ module load apptainer/1.1.9
$ apptainer --version
apptainer version 1.1.9
# Get help on any subcommand
$ apptainer help build
$ apptainer help shell
$ apptainer help exec
```

The Apptainer Container Image

The actual container image is stored in a .sif file (Singularity Image File).

- **Key Files**
  - .sif file: the compressed, read-only container image — everything needed to run
  - .def file: the definition (recipe) file — instructions for building the .sif
- **At Runtime**
  - Apptainer reconstitutes the container from the .sif file
  - No installation required on the host — just the .sif and Apptainer module
  - SIF is a squashfs archive — immutable and safe on network filesystems

Container Image Formats

Apptainer supports two image formats with different use cases.

- **SIF File (default)**
  - Read-only, compressed squashfs archive
  - Safe for network filesystems like /project
  - Use for production SLURM jobs
- **Sandbox Directory**
  - Writable directory tree for development and testing
  - Created with --sandbox flag
  - NOT for production — mainly for development.

Use .sif for production jobs; use --sandbox only during development.

APPTAINER_CACHEDIR

Default cache: ~/.apptainer/cache — can fill your $HOME quota on Wulver.

- Redirect cache to /scratch to avoid HOME quota issues when pulling images
- Set APPTAINER_TMPDIR to /scratch to avoid /tmp space issues during builds/pulls
- Set in ~/.bashrc for a permanent fix

```
export APPTAINER_CACHEDIR=/scratch/$PI/$USER/apptainer
export APPTAINER_TMPDIR=/scratch/$PI/USER/apptainer/tmp

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

Always set **APPTAINER_CACHEDIR** and **APPTAINER_TMPDIR** to /scratch before pulling images on Wulver.

Where to Store .sif Files on Wulver

- **Recommended**
  - /project/$PI/$USER/apptainer/ — large quota, persistent storage
- **Avoid**
  - $HOME — limited quota, large .sif files will fill it quickly
  - /scratch — large quota but purged periodically, not for long-term storage
- **Check Your Usage**
  - homespace (home) · **quota_info** (project/scratch)

```
# Create a dedicated containers directory in /project
mkdir -p /project/$PI/$USER/apptainer
apptainer pull /project/$PI/$USER/apptainer/pytorch.sif docker://pytorch/pytorch:latest
```

Never store large .sif files in $HOME — use /project instead.

Getting Container Images

Recommended Container Workflow for HPC

1

Local Workstation

Build your container image

```
docker build  /  apptainer build
```

2

Container Registry

Docker Hub / GHCR / NGC

```
apptainer pull
```

3

HPC Cluster

Run with .sif image

apptainer shell / exec / run

Best practice: Build containers off-cluster → pull read-only .sif images to run on compute nodes

Using Prebuilt Images with Apptainer

- Do NOT run apptainer pull or build on the login node — resource-intensive and impacts all users
- Launch an interactive SLURM session on a compute node first
- Two main commands for getting images: apptainer pull and apptainer build
- Most workflows only need pull — building from scratch is rarely required

Always pull or build images from a compute node, not the login node.

Apptainer pull

- Saves a remote container image as a local `.sif` file
- URI prefix (e.g., `docker://`) selects the image source

Syntax

```
apptainer pull [output.sif] <URI>
URI schemes:
  docker://    – OCI registries (Docker Hub, GHCR, NGC, etc.)
  oras://      – OCI Registry as Storage
  shub://      – Singularity Hub (deprecated)
```

Output filename is optional — defaults to <image>_<tag>.sif

Apptainer pull — Examples

- URI prefix selects the image source
- Use `docker://` for any OCI registry (Docker Hub, GHCR, NGC, etc.)
- Avoid `:latest` — use specific tags for reproducibility
- If no output name is given, Apptainer saves as `<image>_<tag>.sif`

```
# From Docker Hub
$ apptainer pull tensorflow_gpu_2.15.0.sif docker://tensorflow/tensorflow:2.15.0-gpu
# From GitHub Container Registry (ghcr.io)
$ apptainer pull docker://ghcr.io/apptainer/lolcow
# From NVIDIA NGC
$ apptainer pull pytorch_22.12.sif docker://nvcr.io/nvidia/pytorch:22.12-py3
```

docker:// pulls directly from registries — Docker installation is not needed.

apptainer build

Builds a container image from a source — Docker Hub, or a .def file. Must specify an output .sif filename.

Syntax

```
apptainer build [options] <IMAGE PATH> <BUILD SPEC>
```

```
# Build from Docker Hub
$ apptainer build my_ubuntu.sif docker://ubuntu:22.04

# Build GPU image from NVIDIA NGC
$ apptainer build pytorch_24.08_py3.sif \
    docker://nvcr.io/nvidia/pytorch:24.08-py3
```

Building from a sandbox or .def file requires root or --fakeroot. Do this locally — see Appendix.

pull vs build

```
# pull – download as-is, filename auto-generated
$ apptainer pull docker://alpine:latest
# build – explicit output name, convert to latest format
$ apptainer build alpine.sif docker://alpine:latest
```

```
$ apptainer exec alpine.sif cat /etc/os-release
NAME="Alpine Linux"    ID=alpine    VERSION_ID=3.23.4
PRETTY_NAME="Alpine Linux v3.23"
HOME_URL="https://alpinelinux.org/"
```

`pull` Downloads image as-is; filename auto-generated (e.g., alpine_latest.sif)

`build` Must name the .sif explicitly; can build from scratch via a .def file

Tip: After pulling, run `mv alpine.sif alpine_3.23.sif` to rename with a meaningful version tag.

Running Containers

Three Ways to Launch Processes

Apptainer has three methods for launching processes inside a container.

- **Interactive: apptainer shell**
  - Opens an interactive shell inside the container
  - Explore, debug, or prototype interactively
- **Batch Processing: apptainer exec**
  - Runs a specific command inside the container
  - The standard approach in SLURM batch scripts
- **Container-as-Executable: apptainer run**
  - Executes the container's default runscript
  - Equivalent to running the container as a standalone program

For SLURM jobs, use apptainer exec — it gives you full control over what runs.

apptainer shell

Opens an interactive shell inside the container. Also supports direct pull-and-run from a registry using URI schemes (docker://, library://).

```
# From a local .sif file
$ apptainer shell pytorch.sif
INFO:    underlay of /etc/localtime required more than 50 (72) bind mounts
Apptainer> python -c "import torch; print(torch.version.cuda)"
12.1
Apptainer> exit
# Directly from a registry (pulls image on demand)
$ apptainer shell docker://nvcr.io/nvidia/pytorch:22.12-py3
$ apptainer shell docker://ghcr.io/apptainer/lolcow
```

Use apptainer shell for interactive exploration; it is not suitable for SLURM batch jobs.

Users within an Apptainer Container

Once inside an Apptainer container, you are the same user as on the host system.

```
$ whoami
g07396

$ apptainer shell pytorch_22.12.sif
INFO:    underlay of /etc/localtime required more than 50 (72) bind mounts
Apptainer> whoami
g07396
Apptainer> id
uid=580857(g07396) gid=580857(g07396) groups=580857(g07396),65534(nogroup)
Apptainer> hostname
n0096
```

You are never root inside Apptainer unless you explicitly used sudo — safe for shared HPC systems.

apptainer exec

Starts the container and executes a specific command. More flexible than run — you specify exactly what runs.

Syntax

```
apptainer exec [options] <container> <command> [args...]
```

```
# Compare host OS vs container OS
$ grep "^NAME" /etc/os-release
NAME="Red Hat Enterprise Linux"
$ apptainer exec pytorch.sif grep "^NAME" /etc/os-release
INFO:    underlay of /etc/localtime required more than 50 (72) bind mounts
NAME="Ubuntu"
# Run a Python script inside the container
$ apptainer exec pytorch.sif python3 myscript.py
```

apptainer run vs inspect

apptainer run executes the container's default runscript. apptainer inspect -r shows you what that runscript contains.

```
$ apptainer run lolcow.sif
---------------------------
 \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

$ apptainer inspect -r pytorch_22.12.sif
#!/bin/sh
OCI_ENTRYPOINT='"/opt/nvidia/nvidia_entrypoint.sh"'
OCI_CMD=''

# When SINGULARITY_NO_EVAL set, use OCI compatible behavior that does
# not evaluate resolved CMD / ENTRYPOINT / ARGS through the shell, and
# does not modify expected quoting behavior of args.
… …
```

apptainer run / exec Summary

- **apptainer run**
  - Executes the default command defined in the container's runscript
  - Syntax: apptainer run [--nv] myimage.sif
- **apptainer exec**
  - Executes a specific command — more flexible than run
  - Syntax: apptainer exec [--nv] myimage.sif python3 myscript.py
- **apptainer inspect**
  - Use -r to view the runscript: apptainer inspect -r myimage.sif
  - Use -e to view environment variables set by the container

Both run and exec automatically download the image if a URI is given instead of a .sif path.

Files, Directories and Bind Mounts

Files and Directories in a Container

By default, Apptainer automatically binds several host paths into the container.

- **Default Bind Mounts**
  - $HOME — your home directory
  - $PWD — the current working directory when you ran apptainer
  - /tmp — temporary files
- **Environment Variables**
  - All host environment variables (e.g., $TMPDIR, $PATH) are inherited by default
  - Variables from module load are visible inside the container
- **Adding More Directories**
  - Use --bind (-B) flag to mount additional host directories
  - Syntax: --bind <host_dir>:<container_dir>

apptainer exec --bind

Execute a command inside the container while mounting host directories into the container filesystem.

Syntax

```
apptainer exec --bind <host_dir>:<container_dir> <container> <command>
apptainer exec -B <host_dir>:<container_dir> <container> <command>
```

```
# Bind a /project path to /data inside the container
$ apptainer exec --bind /project/PI_ucid/$USER/data:/data \
    pytorch.sif python /data/analysis.py
# Multiple bind mounts (comma-separated)
$ apptainer exec -B /project/PI_ucid/$USER/data:/data,/scratch/$PI/$USER:/scratch \
    pytorch.sif python /data/run_model.py
```

Your data lives on the host — use --bind to bring it into the container.

GPU Support

Enabling GPU Support

Use the --nv flag to bring NVIDIA drivers and CUDA libraries from the host into the container.

```
apptainer exec  --nv <container> <command>
apptainer shell --nv <container>
apptainer run   --nv <container>
```

```
# Verify GPU access inside container
$ apptainer exec --nv pytorch_22.12.sif python -c \
    "import torch; print('GPU Name:', torch.cuda.get_device_name(0))"
GPU Name: NVIDIA A100-SXM4-80GB
# Pull and run directly from NVIDIA NGC with GPU support
$ apptainer shell --nv docker://nvcr.io/nvidia/pytorch:22.12-py3
```

Container must have a compatible CUDA version — same or older than the host driver.

Running Containers on GPU Nodes

- **Three Requirements**
  - 1. Reserve GPU resources: --partition=gpu and --gres=gpu:<type>:<n>
  - 2. Use a GPU-compatible image (e.g., from NVIDIA NGC)
  - 3. Pass the --nv flag to Apptainer
- **Interactive GPU Session**
  - Request a GPU node with srun first, then run apptainer

```
# Request an interactive GPU node
srun -p gpu -n 1 --gres=gpu:1 --time=59:00 \
    --account=PI_ucid --qos=standard --pty bash
# Inside the interactive session:
module load apptainer/1.1.9
apptainer shell --nv docker://nvcr.io/nvidia/pytorch:22.12-py3
```

Integrating with HPC Workflows

SLURM Batch Job — GPU Container

Use a SLURM batch script to run Apptainer containers on GPU nodes.

```
#!/bin/bash
#SBATCH --job-name=pytorch_job              # Job name
#SBATCH --output=output_%j.out              # Output file
#SBATCH --error=error_%j.err                # Error file
#SBATCH --nodes=1                           # Number of nodes
#SBATCH --cpus-per-task=4                   # Number of CPUs per task
#SBATCH --time=01:00:00                     # Time limit (hh:mm:ss)
#SBATCH --partition=gpu                     # Partition name (adjust as needed)
#SBATCH --account=PI_ucid                   # Replace with PI_ucid
#SBATCH --qos=standard
#SBATCH --gres=gpu:a100:1

module load apptainer/1.1.9
apptainer exec --nv --bind /project/PI_ucid/$USER/data:/data \
    /project/$PI/$USER/apptainer/pytorch_22.12.sif \
    python3 /data/train_model.py
```

Submit with: sbatch submit_gpu.sh

SLURM Batch Job — CPU Container

For CPU-only workloads, omit --gres and --nv.

```
#!/bin/bash
#SBATCH --job-name=my_analysis                # Job name
#SBATCH --output=output_%j.out                # Output file
#SBATCH --error=error_%j.err                  # Error file
#SBATCH --nodes=1                             # Number of nodes
#SBATCH --cpus-per-task=4                     # Number of CPUs per task
#SBATCH --time=01:00:00                       # Time limit (hh:mm:ss)
#SBATCH --partition=general                   # Partition name (adjust as needed)
#SBATCH --account=PI_ucid                     # Replace with PI_ucid
#SBATCH --qos=standard

module load apptainer/1.1.9
apptainer exec --bind /project/PI_ucid/$USER/data:/data \
    /project/$PI/$USER/apptainer/my_python.sif \
    python3 /data/analysis.py
```

--bind is still needed to access your data in /project.

Example – run rocky8 container with Gromacs

```
#!/bin/bash
#SBATCH --job-name=pytorch_job         # Job name
#SBATCH --output=output_%j.out         # Output file
#SBATCH --error=error_%j.err           # Error file
#SBATCH --nodes=1                      # Number of nodes
#SBATCH --cpus-per-task=4              # Number of CPUs per task
#SBATCH --time=01:00:00                # Time limit (hh:mm:ss)
#SBATCH --partition=general            # Partition name (adjust as needed)
#SBATCH --account=kjc59                # Replace with PI_ucid
#SBATCH --qos=standard
#SBATCH --gres=gpu:a100:1

# Load the Apptainer module
module load apptainer

# Run commands inside the container using 'exec'
apptainer exec --nv --bind /apps/easybuild:/apps/easybuild rocky8-module.sif bash -c '
    echo "=== Container OS Info ==="
    cat /etc/os-release
    echo "========================"

    # Load modules and run GROMACS
    module purge
    module use /apps/easybuild/modules/all/Core
    module load foss/2021b
    module load GROMACS/2021.5
    gmx mdrun -v -deffnm success
'
```

Submit Jobs with Apptainer and SLURM

- **GPU Jobs**
  - Add --gres=gpu:<n> to SBATCH directives
  - Add --nv to your apptainer exec or shell command
  - Use a GPU-capable image (NVIDIA NGC recommended)
- **CPU Jobs**
  - Use any standard partition (e.g., general)
  - No --gres or --nv flags needed

Both: load apptainer module, use --bind for data, submit with sbatch.

HPC Support Resources at NJIT

Getting Started
Access to Wulver cluster
hpc.njit.edu/clusters/get_started_on_Wulver

Software List
Available software on Wulver
https://hpc.njit.edu/Software/

Running Jobs
Job submission documentation
https://hpc.njit.edu/Running_jobs/

Open OnDemand
Web-based HPC portal
ondemand.njit.edu

MIG Information
GPU multi-instance resources
https://hpc.njit.edu/MIG

System Updates
News & message of the day
hpc.njit.edu/news

Open a Ticket
Email HPC support directly
hpc@njit.edu

Questions?

Register for next week – Tuesday April 28 Hands on session:

https://hpc.njit.edu/HPC_Events_and_Workshops/