Introduction to Containers

Hui (Julia) Zhao

NJIT High Performance Computing

Outline

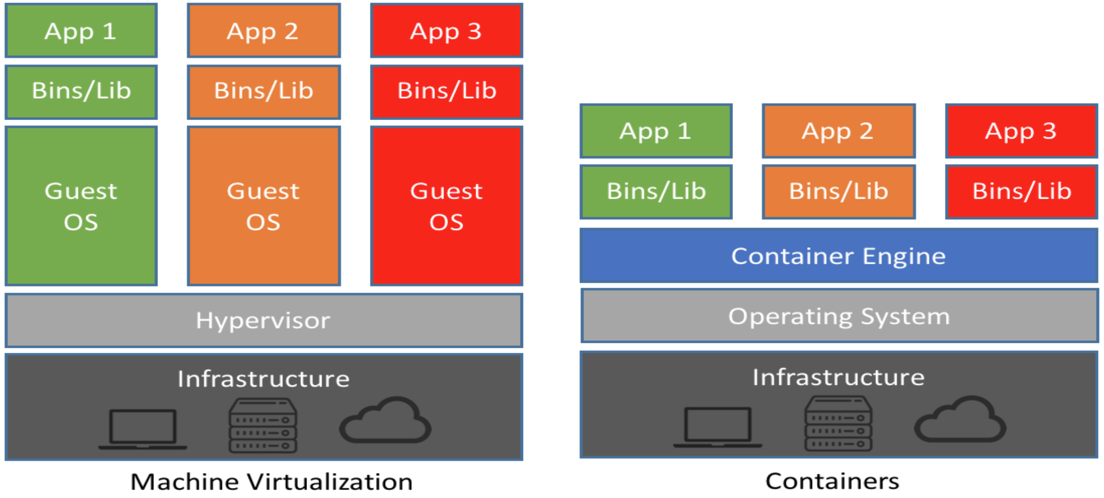
- What is a container?
- Container Engines and Registries
- Running and Managing pre-made Containers
- Container Security
- Integrating Containers with HPC Workflows

What is a Container?

A container is a self-contained entity that provides an isolated software environment for applications and their dependencies.

- Isolate computing environments
- Containers' OS running over host OS
  - Containers' OS can be different from host's OS
  - Containers leverage the host for low-level operations (kernel, network, I/O)
  - From user's view, containers' OS behave like standard OS

Containers vs Virtual Machine



Source: https://pawseysc.github.io/hpc-container-training/11-containers-intro/index.html

Containers vs Virtual Machine - cont

VMs virtualize **hardware** while containers virtualize **operating systems**.

Containers are:

- lighter weight to run (less CPU and memory usage, faster start-up times)
- smaller in size (easier to transfer and share)
- modular (possible to combine multiple containers that work together)

Containers do not virtualize the hardware, containers must be built using the same architecture as the machine they are going to be deployed on. For example, containers built on x86_64 architecture cannot run on the machines using arm64.

Containers vs Conda Environment

- Conda environment: solve software dependency issues within an environment.

- Containers can encapsulate an entire Conda environment itself. This encapsulation ensures that the software, along with its specific dependencies and the operating system, can be transported and replicated on any other system.

Benefits of using containers

- Reproducibility
- Portable software environment
- Access to software not compatible with host OS
- Build and use your own software
- Share software among research groups
- Bundles complex software for easy distribution

Why Use Containers in HPC Environments?

Portability Across Different Systems

- **Consistent Runtime Environment:**
  - Containers encapsulate applications and dependencies
  - run consistently across different systems and environments.
- **Cross-Platform Compatibility:**
  - Easily move workloads between local development environments, on-premises HPC clusters, and cloud platforms.

Simplified Software Deployment

- **Reduced Installation Complexity:**
  - Bundles all dependencies, libraries, and binaries, reducing the need for complex software installations.

Why Containers in HPC Environments? (Cont.)

Enhanced Security

- **Isolated Environments:**
  - Containers isolate applications and their dependencies
  - reducing the risk of conflicts and security vulnerabilities
- **Non-Root Execution:**
  - Containers like Singularity/Apptainer allow non-root execution
  - safer in multi-user HPC environments

Performance Optimization

- **Minimal Overhead:**
  - minimal performance overhead compared to virtual machines
  - GPU and Hardware Acceleration

Container engines

A **container engine** is a software tool that automates the process of running applications in isolated, lightweight environments called containers.

Docker

- Well established
- docker hub for container sharing
- Problematic with HPC

Singularity, Apptainer

- Designed for HPC, user friendly
- Support for MPI, GPUs

Charliecloud, Shifter, Podman

- Also designed for HPC focus
- Simple to use, but may be less practical for complex workflows

Docker vs Singularity(Apptainer)

Docker

- Inside a Docker container the user has escalated privileges, making them root on the host system
- This is **not supported** by most administrators of High Performance Computing (HPC) centers

Singularity/Apptainer

- Integrated with traditional HPC
- Same user inside and outside the container
- User only has root privileges if elevated with sudo
- Run (and modify) existing Docker containers

Singularity and Apptainer

Singularity

- Originally developed by Sylabs in 2015
- Continued development by Sylabs with an emphasis on enterprise features and support
- Remained open source but company interests diverged
- Company (Sylabs) runs the Sylabs Cloud

Apptainer

- Fork of Singularity, started in late 2021
- Funded by the Linux Foundation
- Rootless container build
- Focused on community-driven development and open governance

Container registries

Docker Hub

- **URL:** https://hub.docker.com
- The most popular public container registry with millions of Docker images.
- Hosts official images, community-contributed images, and private repositories.

NVIDIA NGC (NVIDIA GPU Cloud) Registry

- **URL:** https://ngc.nvidia.com
- Specialized registry for GPU-optimized containers.
- Offers containers for deep learning, machine learning, HPC, and data analytics, all optimized for NVIDIA hardware.

Sylabs Container Library (Singularity)

- **URL:** **https://cloud.sylabs.io/library**
- Container registry specifically for Singularity containers.
- Supports secure container images designed for use in high-performance and scientific computing.

Singularity vs Apptainer Commands

Command Purpose	Singularity Command	Apptainer Command
Run a Container	`singularity run <container>`	`apptainer run <container>`
Shell into Container	`singularity shell <container>`	`apptainer shell <container>`
Execute a Command	`singularity exec <container> <cmd>`	`apptainer exec <container> <cmd>`
Build a Container	`singularity build <output> <source>`	`apptainer build <output> <source>`
Pull a Container	`singularity pull <container>`	`apptainer pull <container>`
Inspect a Container	`singularity inspect <container>`	`apptainer inspect <container>`
Sign a Container	`singularity sign <container>`	`apptainer sign <container>`
Verify a Signature	`singularity verify <container>`	`apptainer verify <container>`

Container Module on Wulver

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

```
apptainer --help
Linux container platform optimized for High Performance
Computing (HPC) and Enterprise Performance Computing (EPC)

Usage:
  apptainer [global options...]

Examples:
  $ apptainer help <command> [<subcommand>]
  $ apptainer help build
  $ apptainer help instance start
```

The Apptainer Container Image

- The actual container image, executed by Apptainer as a stand-alone operating system, is stored in a **.sif** file.
- sif *stands for "Singularity Image File"*
- The instructions for constructing the .sif file are provided by the .def definition file
- The .sif file is a compression of all of the files in the stand-alone operating system that comprises a "container".
- Apptainer can use the .sif file to reconstitute the container at runtime.

Using prebuilt images with Apptainer

- *Don't run the* apptainer build *command on the login server! Building the container image can be an intensive process and can consume the resources of the login server.*
- *On the High Performance system, launch an interactive Slurm session*
- **Apptainer pull**
- **Apptainer build**

apptainer pull

```
apptainer pull [pull options...] [output file] <URI>
```

From a library (This prefix typically refers to images stored in the default Docker library)
$ **apptainer pull alpine.sif library://alpine:latest**

From Docker Hub
$ **apptainer pull tensorflow.sif docker://tensorflow/tensorflow:latest**

From Shub
$ **apptainer pull apptainer-images.sif shub://vsoch/apptainer-images**

```
$apptainer pull pytorch.sif docker://pytorch/pytorch:latest
Pytorch.sif

$apptainer pull docker://pytorch/pytorch:latest
pytorch_latest.sif
```

apptainer build

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

“apptainer build” – needs to name the sif file

$ **apptainer build my_ubuntu.sif docker://ubuntu:latest**

apptainer build - cont

To create the .sif file from the .def file

```
$sudo apptainer build my-container.sif my-container.def
```

- *Don’t build container from .def file on Wulver. You need to work on your local system where you have sudo privilege*

pull vs build

```
$ apptainer pull docker://alpine

$ apptainer build alpine.sif docker://alpine
```

Using the **Build** Command in Apptainer

1. Must Name Your Container
2. Image Conversion
   build will convert your image to the latest Apptainer image format
3. Creating Images
   - *Create images from other images*
   - *Create images from scratch using a definition file*
4. Format Conversion
   - Can also use build to convert an image between the container formats supported by Apptainer.

Exploring and Inspecting Container Images

apptainer shell: Opens an interactive shell inside the container, allowing you to run commands manually within the container's environment.

apptainer inspect: Inspect will show you labels, environment variables, apps and scripts associated with the image determined by the flags you pass. By default, they will be shown in plain text.

Running commands within a container

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

apptainer shell

```
apptainer shell [shell options...] <container>
```

apptainer shell: Opens an interactive shell inside the container, allowing you to run commands manually within the container's environment.

```
apptainer shell <path/URL to image>
```

```
$ 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> python
Python 3.10.13 (main, Sep 11 2023, 13:44:35) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.version.cuda)
12.1
```

apptainer shell: using a direct HTTP URL

Apptainer shell also works with the docker://, library://, and shub://

```
$apptainer shell docker://nvcr.io/nvidia/pytorch:22.12-py3
```

nvcr.io/nvidia/pytorch:22.12-py3 is the NVIDIA PyTorch container for version 22.12 with Python 3, hosted on NVIDIA NGC.

Users within a Apptainer Container

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

```
$ apptainer shell pytorch.sif
INFO:    underlay of /etc/localtime required more than 50 (72) bind mounts

Apptainer> whoami
hz3

Apptainer> id
uid=439576(hz3) gid=439576(hz3) groups=439576(hz3),65534(nogroup)
```

apptainer inspect

apptainer inspect -r: Displays the runscript or the default command that will be executed when the container is run, without actually running the container.

```
$ apptainer inspect -r pytorch.sif

#!/bin/sh
OCI_ENTRYPOINT=''
OCI_CMD='"/bin/bash"'
# 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.
if [ -n "$SINGULARITY_NO_EVAL" ]; then
… …
```

apptainer exec

apptainer exec starts the container from a specified image and executes a command inside it.

- Execute a command inside of your container with apptainer exec <path/URL> <command> `apptainer exec myimage.sif python myscript.py`

```
$ 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"
```

apptainer run/exec

```
apptainer run [run options...] <container> [args...]

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

run/exec will download if needed and run

- run: Executes a default command inside the container.
- exec: Executes a specific application/command inside the container as specified with ARGS. More flexible than run command.

```
$ apptainer run myimage.sif
$ apptainer run my_web_app.sif
```

Files and directories within a container

How to access outside directories when running a container?

By default, Apptainer binds:

- The user's home directory ($HOME)
- The current directory when the container is executed ($PWD)
- System-defined paths: /tmp, etc.

You can specify the directories to bind using the --bind or -B flag.

```
$ apptainer shell --bind
/project/hpcadmins/hz3/apptainers/pytorch:/mnt pytorch.sif
```

The colon : separates the path to the directory on the host (/project/hpcadmins/hz3/apptainers/pytorch) from the mounting point (/mnt/) inside of the container.

apptainer exec --bind

1. Executes a command inside the container
2. Bind host directory inside the container

```
apptainer exec -- bind host_dir:container_dir  <path/URL/container_image>  <command>
```

```
$ apptainer exec --bind
/project/hpcadmins/hz3/apptainers/pytorch:/mnt
pytorch.sif python /mnt/check_cuda_version.py
```

Environment Variables

If you are unsure if you are running inside or outside your container, you can run:

```
echo $APPTAINER_NAME
```

If you get back text, you are inside your container.

```
$ apptainer shell pytorch.sif
INFO:    underlay of /etc/localtime required more than 50 (72) bind mounts

Apptainer> echo $APPTAINER_NAME
pytorch.sif
```

Apptainer on GPU Nodes

- --nv flag will setup the basics with CUDA are setup properly for use within a container
- --nv runtime flag brings the drivers from the host
- need to have a compatible CUDA installed in the container (older than or the same as the driver)

Enabling GPU Support

--nv runtime flag brings the drivers from the host

```
apptainer exec --nv pytorch.sif python check_cuda_version.py
```

```
$ apptainer exec --nv pytorch.sif python -c "import torch;
print('GPU Name: ' + torch.cuda.get_device_name(0))"
INFO:    underlay of /etc/localtime required more than 50 (72)
bind mounts
INFO:    underlay of /usr/bin/nvidia-smi required more than 50
(276) bind mounts
GPU Name: NVIDIA A100-SXM4-80GB
```

```
apptainer shell --nv docker://nvcr.io/nvidia/pytorch:22.12-py3
```

Running Containers Using GPU

If we want to run containers that use GPU, these conditions must be met:

1. Reserve the GPU resources (--partition= gpu)
2. Use an image that is compatible with the use of GPU’s
3. --nv runtime flag

in an interactive session, request a GPU node first:

```
srun -p gpu -n 1 --ntasks-per-node=8 --qos=standard --account=PI_ucid --mem-per-cpu=2G --gres=gpu:2 --time=59:00 --pty bash
```

```
apptainer shell --nv docker://nvcr.io/nvidia/pytorch:22.12-py3
```

Integrating Containers with HPC Workflows

```
#!/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:2

# Load any necessary modules
module load apptainer

# Run the tensorflow container
apptainer exec --nv --bind
/project/hpcadmins/hz3/apptainers/tensorflow_container:/workspace
tensorflow_24.08-tf2-py3.sif python3 /workspace/test_tensorflow_gpu.py
```

Submit Jobs with Apptainer and slurm

- For jobs that require GPU resources, ensure that your SLURM job script includes --gres=gpu:<number_of_gpus> and that Apptainer is configured to access the GPU through the NVIDIA runtime.

- Submit your SLURM job script using the sbatch command:
  $ **sbatch submit.sh**

MPI job inside an Apptainer container

```
#!/bin/bash
#SBATCH --job-name=mpi-job
#SBATCH --output=mpi_output.log
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=2
#SBATCH --time=04:00:00
#SBATCH --account=PI_ucid                    # Replace with PI_ucid
... ...

# Load necessary modules
module load apptainer
module load mpi


# Execute MPI job inside Apptainer
mpirun apptainer exec /path/to/mpi_container.sif my_mpi_program
```

Container design strategies

Different ways to design containers when the purpose is to encapsulate *pipelines*.

Make the orchestration **either inside the container**, or to make it from **outside the container** and simply make the calls to software located inside the container.

Features and Benefits of Containers for Scientific Computing

- **Reproducibility**
- **Isolation**
- **Portability**
- **Scalability**
- **Rapid Deployment**

Resources

1. Getting Started with Apptainer (https://apptainer.org/get-started/)
2. Apptainer User Guide (https://apptainer.org/docs/user/main/)
3. Apptainer Official Documentation (https://apptainer.org/documentation/)
4. Apptainer GitHub Repository (https://github.com/apptainer/apptainer)

Resources for Getting Answers at NJIT

Getting Started: [Access to Wulver]

List of Software: [Wulver Software]

Installing Python packages via Conda: [Conda Documentation]

Request Software: [HPC Software Installation]

Contact: Please visit [HPC Contact]

Open a ticket: email to hpc@njit.edu

Consult with Research Computing Facilitator: [HPC User Assistance]

System updates

- Read Message of the Day on login
- Visit [NJIT HPC News]

Questions?