NHR FAU

Friedrich-Alexander-Universität Erlangen-Nürnberg

Using containers on the HPC systems at NHR@FAU

HPC Services, NHR@FAU

hpc-support@fau.de

https://doc.nhr.fau.de

High Performance Computing

Agenda

This presentation introduces the usage of container to manage software environments. They offer greater portability and reproducibility and are relevant as a replacement for the *python* or *conda* virtual environments.

1. **Introduction**
   - Why use container in your HPC project?
2. **Container basics**
   - OS-level virtualization
3. **Setting up your environment**
   - Starting with a fresh container
   - Utilizing existing container
   - Using our recipes as drop-in *venv* replacement
4. **Troubleshooting and Best Practices**
   - Recap of building and using container
   - First step for troubleshooting

NHR FAU

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Introduction & Motivation

Why use container for HPC/AI workloads?

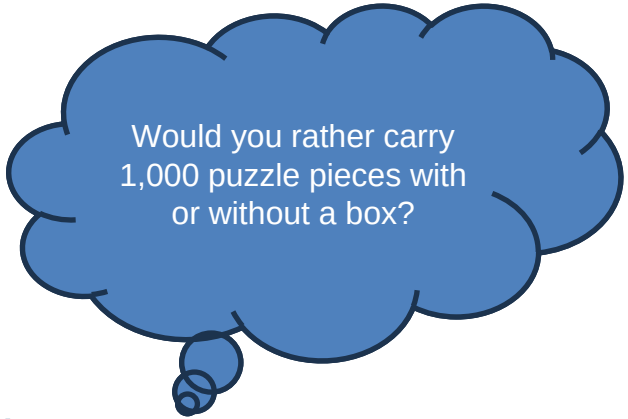
- Portability and Reproducibility
  - **BYOE**: Bring Your Own Environment!
  - **Less dependence** on software and their version present on cluster
  - **Easy copying** between systems
  - **Simple distribution** through hubs
- Flexibility
  - System packages can be installed easily inside the container
  - *Fakeroot* feature gives you **root access** not possible directly on cluster
  - Build on own devices and copy any HPC system

Source: https://arxiv.org/pdf/2005.14165

Why use container for HPC/AI workloads?

Storage performance

- Problem
  - *Pythons pip* packaging system needs virtual environments for preventing version conflicts of multiple projects, thereby **creating many small files**
  - Loading many small files from storage takes long and **slows down general file system** performance due to overhead



- Solution
  - One big container with all relevant files **loading fast** and running isolated form other projects
  - **Fast copying** of files

Why use container for HPC/AI workloads?

Because you should!

- Increasing problems with *python* conflicts and file counts
- *Python* modules will be removed from the newer clusters
- Limited inodes on Helma (quota of 500k per group)

Survey

Have you encountered slow environment setup and loading times?

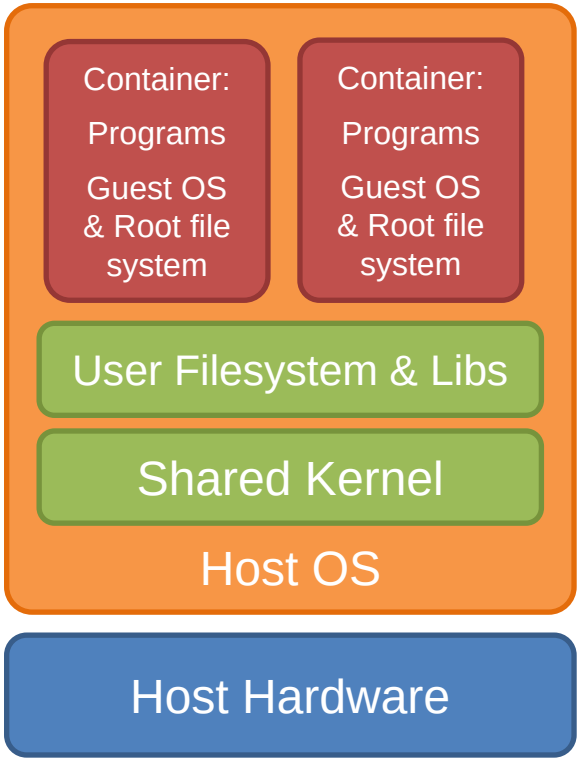
NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Container basics

OS-level virtualization

- Operating system virtualization paradigm where the kernel allows multiple isolated user space instances to exist
- Programs running environment is limited to the container's contents and individually bound libraries from outside (like CUDA/MPI)
- Shared kernel approach where all containers share the same OS kernel, unlike full virtualization that requires separate operating systems



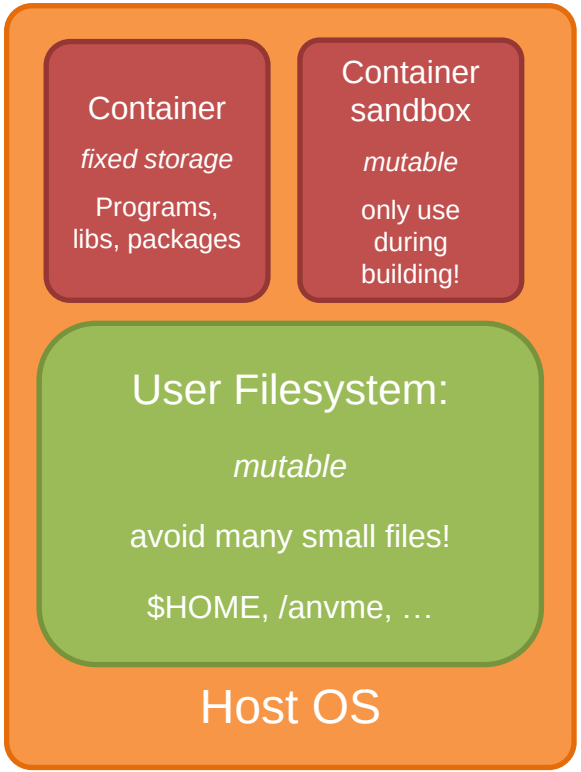
Apptainer

- Open source container application optimized for HPC systems
- Began 2015 as project at the Lawrence Berkeley National Laboratory as Singularity
- Now part of the Linux Foundation as Apptainer

- Support of InfiniBand and Intel Omni-Path
- PCIe device support like GPUs
- Open MPI support via two approaches
- Integrates into SLURM

Storage Concept

- Once build, a containers storage is packed as one **immutable** .sif file
- Set up container as environment and add libraries or static dependencies inside
  - → use your host file system as storage of the changing project files (e.g. python/c++ files)
- Container sandboxes should only used during building and removed afterwards (ideally on own system or in $TMPDIR)!
- After development build one container with all files as a production version and share it

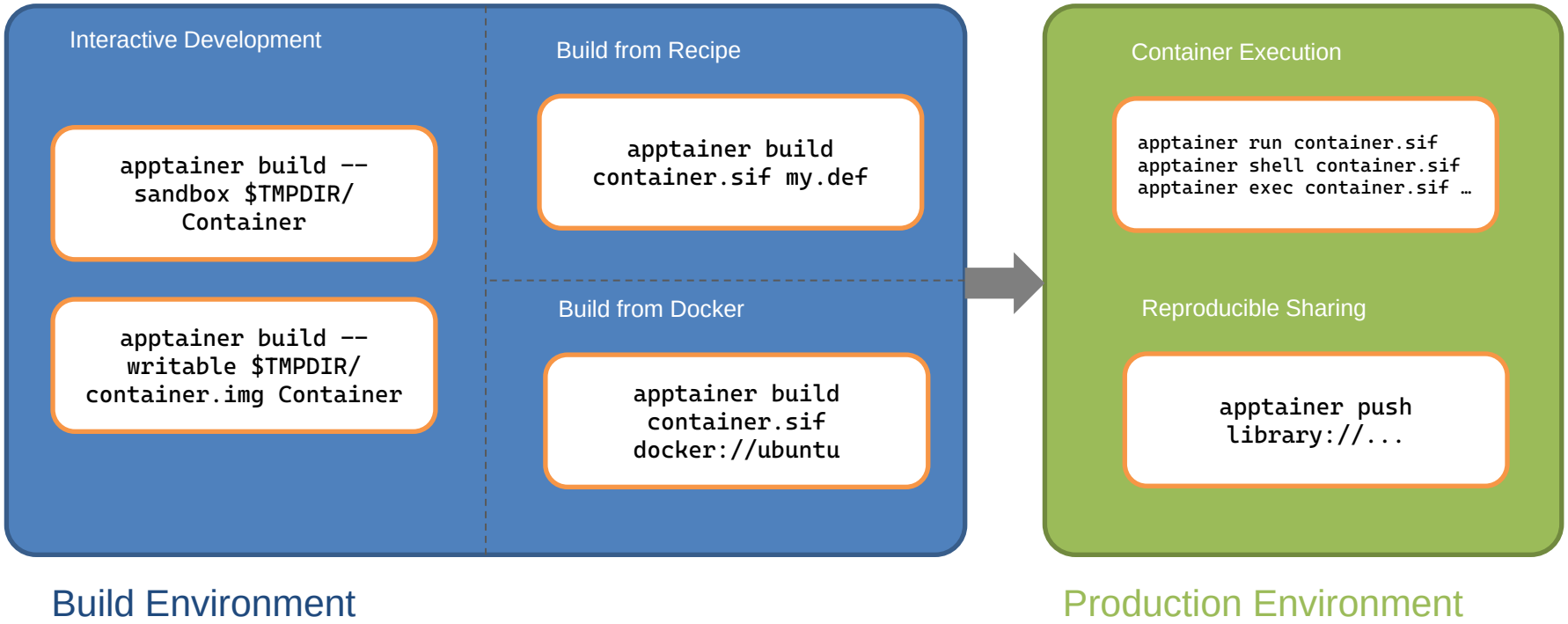


NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Setting up your environment

Workflow for using Apptainer



Introductory Example: Using Existing Container

```
apptainer pull docker://ghcr.io/apptainer/lolcow
apptainer exec lolcow_latest.sif cowsay moo

_____
< moo >
_____
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
              ||----w |
              ||      ||
```

Build your own container interactively

- Create sandbox
  - `apptainer build --sandbox <container_name>/ docker://<base_container>`
- Enter writable container
  - `apptainer shell --writable <container_name>`
- Convert sandbox to image and back
  - `apptainer build <container_name>.sif <container_name>`
  - `apptainer build --sandbox <container_name> <container_name>.sif`
- Make sure to set python packages location inside the container and not in your home file system!
- Delete unpacked sandbox afterwards!

Build your own container from def file

- Build container
  - `apptainer build <container_name>.sif <definition_file>`
- Def file: python example

```
Bootstrap: docker
From: dockerhub-mirror.rrze.uni-erlangen.de/python:3.10

%files
            requirements.txt

%environment
            export LISTEN_PORT=54321

%post
            pip install --root-user-action=ignore -r requirements.txt

%runscript
            echo "Container was created $NOW"
            echo "Arguments received: $*"
            exec echo "$@"
```

Define base container

Files needed from file system

Set environment variable persistently present in the container

Put your system setup here

Executed when the container is run

Build your own container from def file

- Def file: From conda environment

```
Bootstrap: docker
From: dockerhub-mirror.rrze.uni-erlangen.de/condaforge/miniforge3:latest

%files
            environment.yaml /

%post
            export CONDA_OVERRIDE_CUDA=12.8
            ENV_NAME=$(head -1 /environment.yaml | cut -d' ' -f2)
            echo ". /opt/conda/etc/profile.d/conda.sh" >> $APPTAINER_ENVIRONMENT
            echo "conda activate $ENV_NAME" >> $APPTAINER_ENVIRONMEN

            . /opt/conda/etc/profile.d/conda.sh
            conda env create -f /environment.yaml -p /opt/conda/envs/$ENV_NAME
            conda clean -all

%runscript
            exec "$@"
```

Build your own container from def file

- Def file: For spack

```
Bootstrap: docker
From: almalinux:8.10

%post
  dnf update -y && dnf upgrade -y
  dnf install -y python3 tar
  dnf install -y gcc gcc-c++ gcc-gfortran git make patch cpp autoconf automake libtool m4 openssl curl
  dnf install -y epel-release glibc-devel glibc kernel-headers mpfr libmpc openssh libzip
  dnf install -y numactl perl
  git clone --depth=100 --branch=v0.23.1 https://github.com/spack/spack.git /opt/spack

  /opt/spack/bin/spack compiler find
  /opt/spack/bin/spack install ...
```

Run your container

- Execute command
  - `apptainer exec <container_name>.sif command…`
- Start shell
  - `apptainer shell <container_name>.sif`
- Run pre-defined run script
  - `apptainer run <container_name>.sif`
- Options
  - `--bind`: give the container access to additional file systems
    - `apptainer shell --bind /opt,/data:/mn <container_name>.sif`

Run your container

- Execute command
  - `apptainer exec <container_name>.sif command…`
- Start shell
  - `apptainer shell <container_name>.sif`
- Run pre-defined run script
  - `apptainer run <container_name>.sif`
- Options
  - `--bind`: give the container access to additional file systems
  - `--fakeroot`: enabling operations requiring root access inside of the container
  - `--contain`: restrict filesystem to container (by default all file systems are bound)
  - `--H`: specify the default home directory
  - `--nv`/`--nvccli`/`--rocm`: GPU feature support (`--nv` is set by default on Alex/Helma)

Start container from SLURM script

- Example: batch script with ddp python ML code

```
#!/bin/bash -l

#SBATCH --job-name=fno
#SBATCH --output=output/slurm-%j.out
#SBATCH --error=output/slurm-%j.err
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --gpus-per-node=4
#SBATCH --cpus-per-task=32
#SBATCH --time=03:00:00
#SBATCH --partition=a100
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

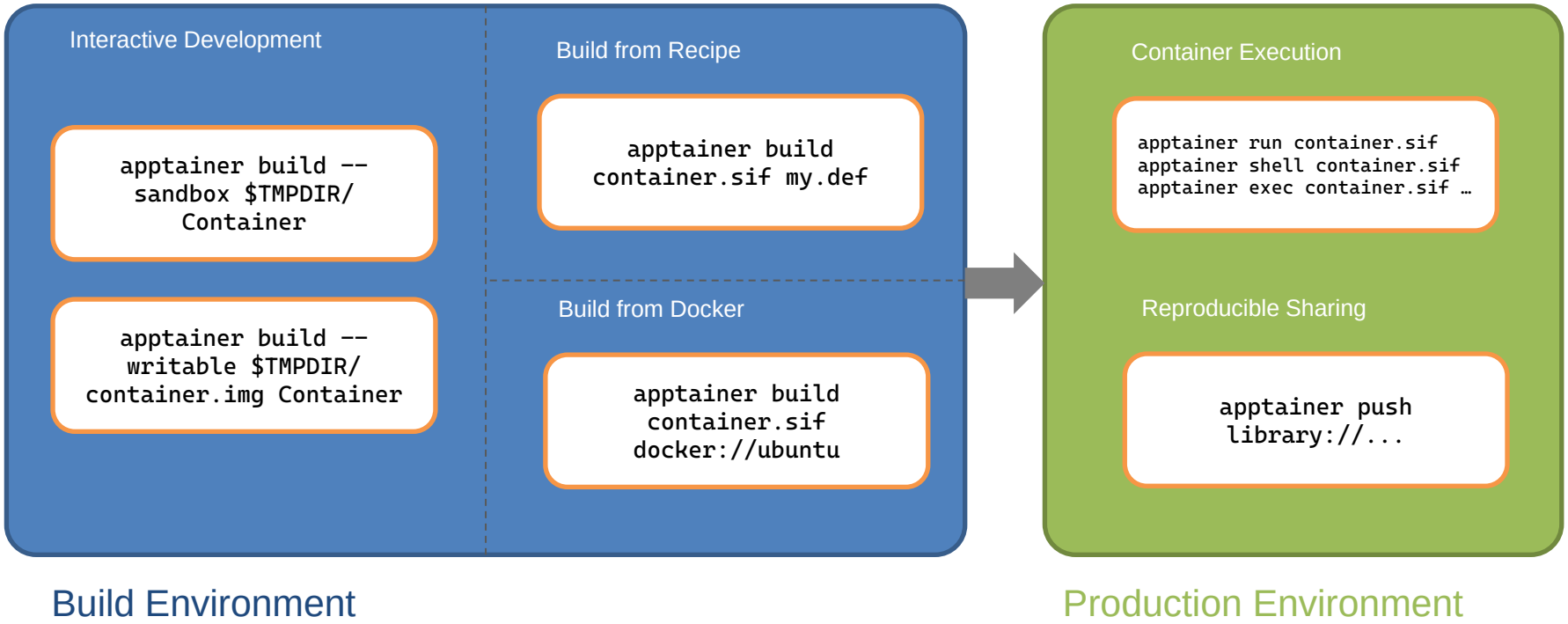
export
CONTAINER_PATH="/anvme/workspace/…container/envFNO.sif"
export CODE_PATH="/anvme/workspace/projects/perm"
export GPUS=4

export EXPERIMENT=1
export LAYER_SIZE=32

cd $CODE_PATH/permFNO
```

```
# Run the command with specified resources and environment
variables
for layers in 2 4 8 12; do
    srun --ntasks=$GPUS \
         --ntasks-per-node=$GPUS \
         --kill-on-bad-exit=1 \
         apptainer exec --nv \
             --env GPUS=$GPUS \
             --env WORLD_SIZE=$GPUS \
             --env MASTER_PORT=12348 \
             --env MASTER_ADDR=$(scontrol show hostnames
"$SLURM_JOB_NODELIST" | head -n 1) \
             --bind /hnvme \
             $CONTAINER_PATH \
             python $CODE_PATH/permFNO/main.py \
                 --layers $layers \
                 --layer_size $LAYER_SIZE \
                 --distributed
done
```

Workflow for using Apptainer



NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Troubleshooting and Best Practices

https://doc.nhr.fau.de/environment/apptainer

Survey

What do you use as current environment?

Cache

- The cache is stored in `$HOME/.apptainer/cache` as default and can fill up fast!
- Set cache folder
  - `export APPTAINER_CACHEDIR=$WORK/…/<user>-container/`
  - Add to your `~/.bashrc` or `.bash_profile` for persistent change
- Clean cache
  - `apptainer cache clean [clean options…]`
- Options
  - `--days int:` remove all cache entries older than specified number of days
  - `--force:` suppress any prompts and clean the cache
  - `--type strings:` a list of cache types to clean (possible values: library, oci, shub, blob, net, oras, all) (default [all])

Choice of Base Container

- Miniforge3
  - Use if you want to use conda packages or have a drop in `.yaml` config environment
- CUDA
  - Because binding outside libraries in combination with `module load` can be problematic
- Ubuntu / other distributions
  - Simplest base with freedom to build most essential setup without conflicts
- Your individual application
  - Highest chance of working out-of-the-box
  - Usually with git repository like setup

Ubuntu problems

- Containers have to be built on a system where you have enough permissions and the security model allows it
- Containers can be build on the AlmaLinux based cluster frontend nodes (Fritz, Alex, Helma, Woody, Meggie)
- It is not possible to build them on our Ubuntu based systems (TinyX, Testcluster)

```
ERROR  : Could not write info to setgroups: Permission denied
ERROR  : Error while waiting event for user namespace mappings: Success
FATAL:   While performing build: while running engine: exit status 1
```

- Replace `apptainer build <options>` with `/apps/singularity/apptainer-wrapper.sh build <options>`
- `apptainer build` should switch automatically to working script on affected systems

Open MPI Binding

- Two options
  - NOT RECOMDENDED: Hybrid
    - Install the exact MPI version inside of the container that communicates with the outside versions
    - Complicated setup, error prone and needs additional dependencies (Slurm, InfiniBand,…)
  - Binding
    - Add MPI folder to your section of your .def file
    - ```
      %environment
         export PATH="$OPENMPI_ROOT/bin:$PATH"
         export LD_LIBRARY_PATH="$OPENMPI_ROOT/lib:$LD_LIBRARY_PATH"
      ```
    - Execute with `--bind`
    - `mpirun -n <threads> apptainer exec --bind "$OPENMPI_ROOT" <container>.sif <executable>`
- Include RDMA/InfiniBand **`rdma-cora`**, **`libibverbs1`**, etc to communicate efficiently
- Check NCCL performance when using pytorch to verify correct usage
      ```

Open MPI Binding

- Outlook
  - Future Open MPI v5.0.x release adds compatibility to previous versions
  - Version specific names are standardized
  - Should make hybrid approach reliable and simple

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

THANK YOU.

NHR@FAU
https://doc.nhr.fau.de
hpc-support@fau.de

HPC High Performance Computing

Sources & Useful Links

- NHR@FAU docs: https://doc.nhr.fau.de/environment/apptainer
- Apptainer docs: https://apptainer.org/docs/user/latest/
- NHR@Göttingen: Declutter your Python environments: https://gitlab-ce.gwdg.de/hpc-team-public/science-domains-blog/-/blob/main/20230907_python-apptainer.md
- HPC Café: Handling many small files and managing AI data sets (from Februrary 11, 2025) https://hpc.fau.de/2025/02/04/monthly-hpc-cafe-efficient-packing-and-handling-of-large-data-sets-hybrid-event/
- OpenMPIv4: https://docs.open-mpi.org/en/v5.0.x/building-apps/abi-compatibility.html