

A white wolf is walking towards the viewer through a snowy, futuristic cityscape at night. The city is illuminated by blue and white lights, with tall buildings and a grid-like pattern of light trails in the background. The wolf is in the foreground, slightly to the right of the center, and its fur is white and fluffy. The overall atmosphere is cold and technological.

# LUMI

Running containers  
on LUMI

Julius Roeder – Danish e-infrastructure Consortium (DeiC)

# Containers – bring your own user space



- When running containers, the kernel, drivers and hardware is still provided by the host (LUMI) - but the user space (directory tree) changes
- Benefits of using containers:
  - **Enhanced reproducibility:** A fixed (read-only) user space for each computational experiment
    - Have a fully self-contained software environment
    - Pin all versions of the software packages used
    - Version control software environments
  - **Enhanced portability:** Run your container on other systems – as long as the system libraries are compatible
    - Makes the same environment work on all compute platforms – from laptop to supercomputer
    - Makes it easier to share your software environment with others – just share the container
  - **Easily test and trash:** Try a new container – if it doesn't work just trash the container and start over again
  - **Bonus:** Your software environment is a single file (the container) on the Lustre shared filesystems, which is much less stressful to Lustre and more performant, making for a much nicer experience for everyone on LUMI.

LUMI directory tree

```
/
├── appl -> /pfs/lustrep4/appl
├── bin
├── boot
├── cray
├── dev
├── etc
├── flash
├── home
├── image
├── lib
├── lib64
├── local
├── mnt
├── opt
├── pfs
├── proc
├── projappl
├── project -> /projappl
├── root
├── run
├──/sbin
├── scratch
├── selinux
├── srx
├── sys
├── tmp
├── users
├── usr
└── var
```

Ubuntu container directory tree

```
/
├── bin -> usr/bin
├── boot
├── dev
├── environment -> .singularity.d/env/90-gn
├── etc
├── home
├── lib -> usr/lib
├── lib32 -> usr/lib32
├── lib64 -> usr/lib64
├── libx32 -> usr/libx32
├── media
├── opt
├── proc
├── project
├── root
├── run
├── sbin -> usr/sbin
├── singularity -> .singularity.d/runscript
├── srx
├── sys
├── tmp
├── users
├── usr
└── var
```

These are generally  
NOT the same

# Running containers on LUMI



- On LUMI, you can run Singularity/Apptainer containers
  - Singularity/Apptainer are HPC container runtimes that allow you to run unprivileged containers, i.e. no need for root or sudo
  - Singularity/Apptainer is not Docker, but if you have an existing Docker container, you can run it using Singularity/Apptainer
- Main singularity commands:
  - Getting (pulling) a container from a container registry  
`singularity pull my_container.sif docker://ubuntu:22.04`
  - Opening a shell inside the container  
`singularity shell my_container.sif`
  - Executing a command inside the container  
`singularity exec my_container.sif python3 my_script.py`
- Running containers on compute nodes
  - Launch computation using srun  
`srun <options> singularity exec my_container.sif python3 my_script.py`

# Bind mounting parts of the host file system (1)

- When running a container on LUMI, where is /project, /scratch, etc.?
- You may "inject" parts of the host (LUMI) file system into the container by bind mounting it

```
singularity exec --bind /project/<project_ID>  
my_container.sif tree -L 1 /
```

```
your_lumi_username@uan03:> singularity exec --bind /project/project_465001363 ubuntu_tree.sif tree -L 1 /  
/  
-- bin -> usr/bin  
-- boot  
-- dev  
-- environment -> .singularity.d/env/90-environment.sh  
-- etc  
-- home  
-- lib -> usr/lib  
-- lib32 -> usr/lib32  
-- lib64 -> usr/lib64  
-- libx32 -> usr/libx32  
-- media  
-- mnt  
-- opt  
-- proc  
-- project  
-- root  
-- run  
-- sbin -> usr/sbin  
-- singularity -> .singularity.d/runscript  
-- srv  
-- sys  
-- tmp  
-- users  
-- usr  
-- var
```

LUMI directory tree

```
/  
-- appl -> /pfs/lustrep4/appl  
-- bin  
-- boot  
-- cray  
-- dev  
-- etc  
-- flash  
-- home  
-- image  
-- lib  
-- lib64  
-- local  
-- mnt  
-- opt  
-- proc  
-- pfs  
-- projappl  
-- project -> /projappl  
-- root  
-- run  
-- sbin  
-- scratch  
-- selinux  
-- srv  
-- sys  
-- tmp  
-- users  
-- usr  
-- var
```

ubuntu container directory tree

```
/  
-- bin -> usr/bin  
-- boot  
-- dev  
-- environment -> .singularity.d/env/90-environment  
-- etc  
-- home  
-- lib -> usr/lib  
-- lib32 -> usr/lib32  
-- lib64 -> usr/lib64  
-- libx32 -> usr/libx32  
-- media  
-- mnt  
-- opt  
-- proc  
-- root  
-- run  
-- sbin -> usr/sbin  
-- singularity -> .singularity.d/runscript  
-- srv  
-- sys  
-- tmp  
-- users  
-- usr  
-- var
```

## Bind mounting parts of the host file system (2)



You typically want to bind mount your project folders (/project, /scratch, /flash). A shortcut is:

```
module use /appl/local/containers/ai-modules
```

```
module load singularity-AI-bindings
```

- You may need to bind mount some of the host libraries to fully exploit the hardware
  - On LUMI you need this to get optimal performance when using the Slingshot 11 interconnect
  - **WARNING:** Be careful when bind mounting libraries from the host system. You can easily end up in a broken state if mixing the container libraries with host libraries.

```
$ module use /appl/local/containers/ai-module
$ module load singularity-AI-bindings
$ singularity exec ubuntu_tree.sif tree -L 1 /
/
|-- appl
|-- bin -> usr/bin
|-- boot
|-- dev
|-- environment -> .singularity.d/env/90-environment.sh
|-- etc
|-- flash
|-- home
|-- lib -> usr/lib
|-- lib32 -> usr/lib32
|-- lib64 -> usr/lib64
|-- libx32 -> usr/libx32
|-- media
|-- mnt
|-- opt
|-- pfs
|-- proc
|-- projappl
|-- project
|-- root
|-- run
|-- sbin -> usr/sbin
|-- scratch
|-- singularity -> .singularity.d/runscript
|-- srv
|-- sys
|-- tmp
|-- users
|-- usr
`-- var
```

# Official LUMI containers



Official LUMI containers available on LUMI under `/appl/local/containers/sif-images` :

- Application images:
  - JAX
  - mpi4py
  - PyTorch
  - Tensorflow + Horovod
- Base images:
  - `Lumi-rocm-rocm-X.Y.Z.sif`: ROCm + aws-ofi-rccl + MI250X (gfx90a) MIOpen kernels + rccltest
- Remember to copy these to your project folder
  - We may remove/replace the container under `/appl/local/containers/sif-image` at any time!
  - If you like EasyBuild and modules, we also provide a set of easyconfigs to "install" the containers.

```
your_lumi_username@uan03:~$ ls -l /appl/local/containers/sif-images/  
gcc.sif.sif  
lumi-jax-rocm-6.2.0-python-3.12-jax-0.4.28.sif  
lumi-mpi4py-rocm-6.2.0-python-3.12-mpi4py-3.1.6.sif  
lumi-pytorch-rocm-5.7.3-python-3.12-pytorch-v2.2.2.sif  
lumi-pytorch-rocm-6.0.3-python-3.12-pytorch-v2.3.1.sif  
lumi-pytorch-rocm-6.1.3-python-3.12-pytorch-v2.4.1.sif  
lumi-pytorch-rocm-6.2.0-python-3.10-pytorch-v2.3.0.sif  
lumi-pytorch-rocm-6.2.0-python-3.12-pytorch-20240801-vllm-c7a3a47.sif  
lumi-pytorch-rocm-6.2.1-python-3.12-pytorch-20240918-vllm-4075b35.sif  
lumi-pytorch-rocm-6.2.3-python-3.12-pytorch-v2.5.1.sif  
lumi-rocm-rocm-5.7.3.sif  
lumi-rocm-rocm-6.0.3.sif  
lumi-rocm-rocm-6.1.3.sif  
lumi-rocm-rocm-6.2.0.sif  
lumi-rocm-rocm-6.2.1.sif  
lumi-rocm-rocm-6.2.2.sif  
lumi-tensorflow-rocm-6.2.0-python-3.10-tensorflow-2.16.1-horovod-0.28.1.sif  
perpetto4rocm.sif.sif  
your_lumi_username@uan03:~$
```

## Utilize Slingshot 1.1 interconnect

- These LUMI containers are built against the Cray Programming Environment (CPE) However, the CPE is NOT included in the container due to license restrictions
- To fully utilize the Slingshot 1.1 interconnect with these containers, you need to bind mount parts of the CPE when running the container

```
singularity exec --bind /var/spool/slurmd,/opt/cray,/usr/lib64/libcxi.so.1,  
/usr/lib64/libjansson.so.4 <program>
```

  - For the containers making use of MPI (mpi4py and Horovod), this is required
  - For all other containers it is optional. If you don't include it, RCCL internode communication falls back to using slower TCP/IP sockets
  - Shortcut to getting the binds right:

```
module use /appl/local/containers/ai-modules  
module load singularity-AI-bindings
```

For the LUMI application containers, you need to run `$WITH_CONDA` in the container to activate the conda environment in which the application, e.g. PyTorch, is installed

## Further reading

- LUMI Docs running containers page:  
<https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/container-jobs/>
- LUMI (EasyBuild) Software Library:  
<https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/>