Containers on LUMI-C and LUMI-G

Kurt Lust
LUMI User Support Team (LUST)
University of Antwerp

May 2024

Containers

LUMI

This is about containers on LUMI-C and LUMI-G!

- What can they do and what can't they do?
- Getting containers onto LUMI
- Running containers on LUMI
- Enhancements to the LUMI environment to help you
- Using some of our pre-built AI containers

- But remember: LUMI is an HPC infrastructure, not a container cloud!

What do containers not provide?

LUMI

- **Full reproducibility of results** is a myth
- **Full portability**: Not every container prepared on your Ubuntu or CentOS cluster or workstation will work on LUMI.
  - Containers that rely on certain hardware, kernel modules and/or kernel versions may fail.
  - Problem cases: High-performance networking (MPI) and GPU (driver version)
- **Performance portability**:
  - A container built from sources on one CPU will not be optimal for another one.
  - Containers built from downloaded binaries may not exploit all architectural features of the CPU.
  - No support for the LUMI interconnect may lead to fall-down to slower protocol that works

But what can they then do on LUMI?

LUMI

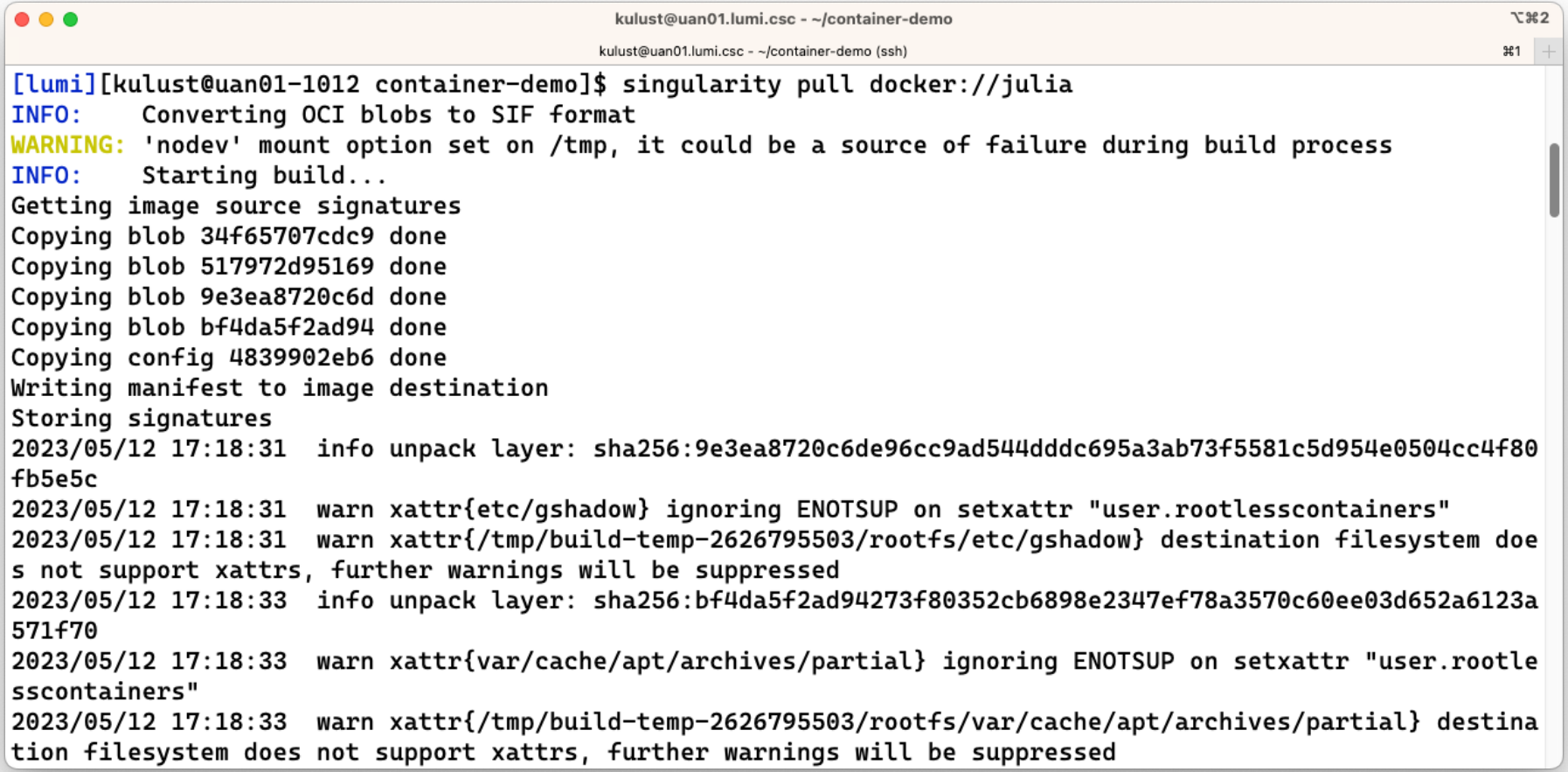
- **Storage manageability:** Lower pressure on the filesystems (for software frameworks that access hundreds of thousands of small files) for better I/O performance and management of your disk file quota.
  - E.g., conda installations are not appreciated straight on the Lustre file system
- **Software installation:** Can be a way to install software with an installation process that is not aware of multi-user HPC systems and is too complicated to recompile.
  - E.g., GUI applications that need a fat library stack
  - E.g., experiment with software that needs a newer version or ROCm, though with limitations
- **But note:** You're the system administrator of your container, not LUST!

Managing containers

LUMI

- Supported runtimes
  - Docker is **NOT** directly available in the user environment (and will never be)
  - Singularity Community Edition is natively available (as a system command) on the login and compute nodes
- But you can convert docker containers to singularity: Pulling containers
  - DockerHub and other registries (example: Julia container)
    `singularity pull docker://julia`
  - Singularity uses a flat (single) sif file for storing the container and the pull command makes the conversion
  - Be carefull: cache in `.singularity` dir can easily exhaust your storage quota for larger images
    - May want to set `SINGULARITY_CACHEDIR` to move the cache

singularity pull docker://julia



singularity pull docker://julia

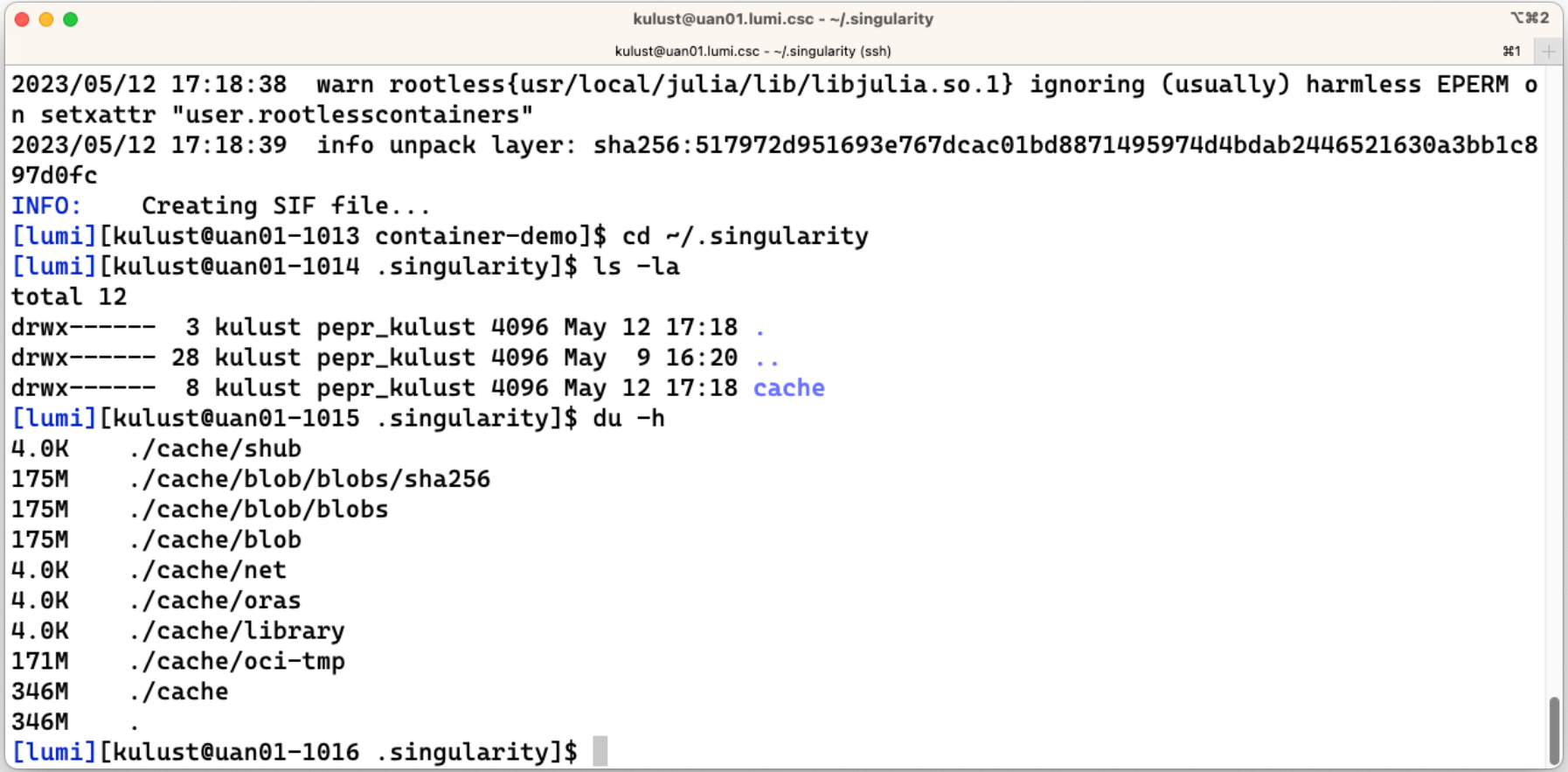
LUMI

kulust@uan01.lumi.csc - ~/container-demo

kulust@uan01.lumi.csc - ~/container-demo (ssh)

```
PERM on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libumfpack.so.5} ignoring (usually) harmless
 EPERM on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libunwind.so} ignoring (usually) harmless EP
ERM on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libunwind.so.8} ignoring (usually) harmless
EPERM on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libuv.so} ignoring (usually) harmless EPERM
on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libuv.so.2} ignoring (usually) harmless EPER
M on setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libz.so} ignoring (usually) harmless EPERM o
n setxattr "user.rootlesscontainers"
2023/05/12 17:18:36  warn rootless{usr/local/julia/lib/julia/libz.so.1} ignoring (usually) harmless EPERM
 on setxattr "user.rootlesscontainers"
2023/05/12 17:18:38  warn rootless{usr/local/julia/lib/libjulia.so} ignoring (usually) harmless EPERM on
setxattr "user.rootlesscontainers"
2023/05/12 17:18:38  warn rootless{usr/local/julia/lib/libjulia.so.1} ignoring (usually) harmless EPERM o
n setxattr "user.rootlesscontainers"
2023/05/12 17:18:39  info unpack layer: sha256:517972d951693e767dcac01bd8871495974d4bdab2446521630a3bb1c8
97d0fc
INFO:    Creating SIF file...
[lumi][kulust@uan01-1013 container-demo]$
```

singularity pull docker://julia



Managing containers (2)

LUMI

- Building containers
  - Support for building containers is very limited on LUMI: no elevated privileges but also no user namespaces.
    We can support `proot` though.
  - You should either pull or copy containers from outside
  - Singularity can build from existing (base) container in some cases
    - Build type called "Unprivileged proot builds" in the Singularity CE manual
    - Needs `proot` from the `systools/23.09` module in CrayEnv and LUMI/23.09.
  - We provide some base images adapted for LUMI

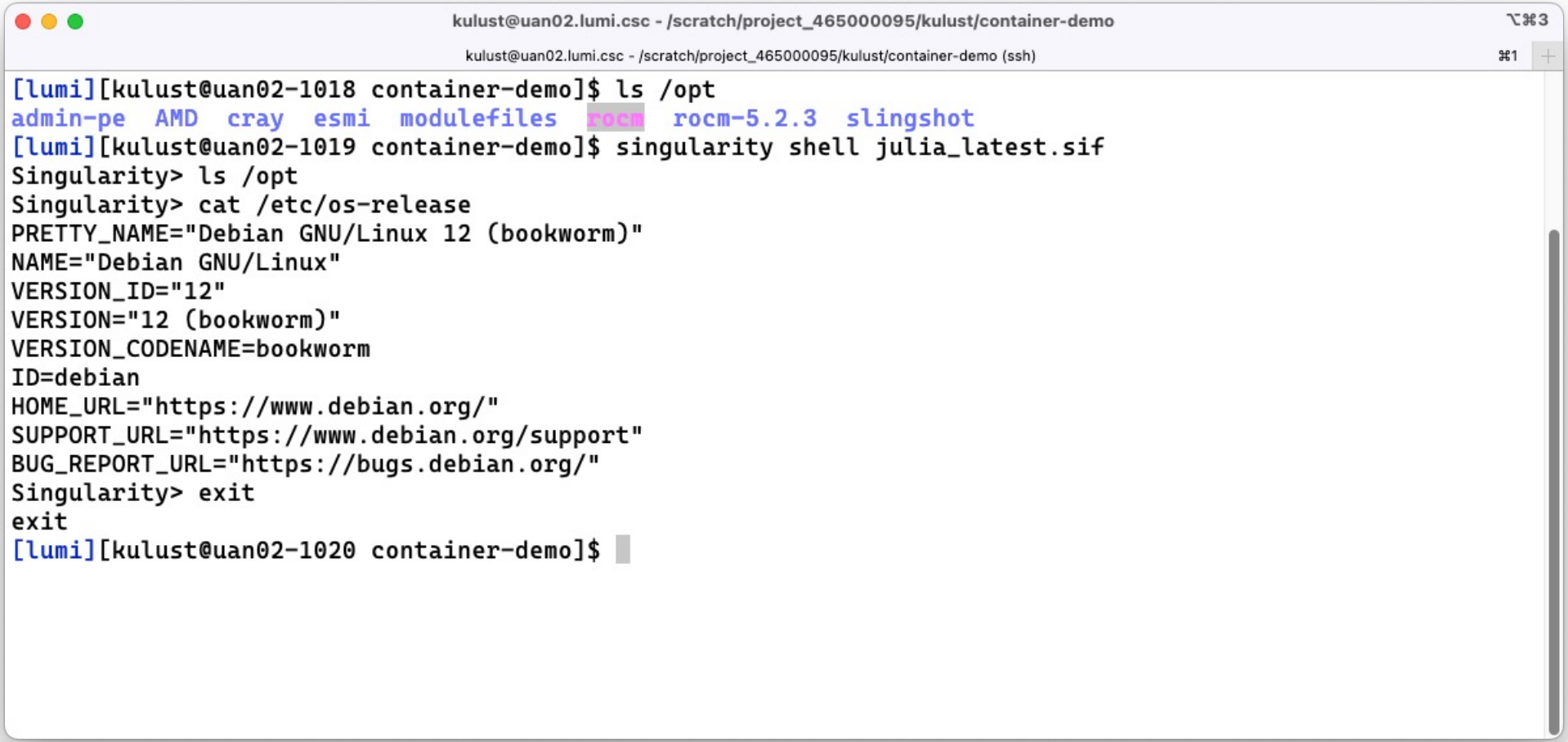
Interacting with containers

LUMI

- Accessing a container with the shell command
  `singularity shell container.sif`
- Executing a command in the container with exec
  `singularity exec container.sif uname -a`
- "Running" a container
  `singularity run container.sif`
- Inspecting run definition script
  `singularity inspect --runscript container.sif`
- Accessing host filesystem with bind mounts
  - Singularity will mount `$HOME`, `/tmp`, `/proc`, `/sys`, `/dev` into container by default
  - Use `--bind src1:dest1,src2:dest2` or the `SINGULARITY_BIND(PATH)` environment variable to mount other host directories (like `/project` or `/appl`)

singularity shell julia_latest.sif

LUMI



singularity exec julia_latest.sif uname -a

LUMI

kulust@uan02.lumi.csc - /scratch/project_465000095/kulust/container-demo ⌥⌘3

kulust@uan02.lumi.csc - /scratch/project_465000095/kulust/container-demo (ssh) ⌘1

```
[lumi][kulust@uan02-1021 container-demo]$ uname -a
Linux uan02 5.14.21-150400.24.81_12.0.75-cray_shasta_c #1 SMP Thu Sep 7 00:12:59 UTC 2023 (1027017) x86_64
 x86_64 x86_64 GNU/Linux
[lumi][kulust@uan02-1022 container-demo]$ singularity exec julia_latest.sif uname -a
Linux uan02 5.14.21-150400.24.81_12.0.75-cray_shasta_c #1 SMP Thu Sep 7 00:12:59 UTC 2023 (1027017) x86_64
 GNU/Linux
[lumi][kulust@uan02-1023 container-demo]$ singularity exec julia_latest.sif cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
[lumi][kulust@uan02-1024 container-demo]$
```

```
singularity run julia_latest.sif
singularity inspect -runscript julia_latest.sif
```

LUMI

kulust@uan02.lumi.csc - /scratch/project_465000095/kulust/container-demo ⌥⌘3

kulust@uan02.lumi.csc - /scratch/project_465000095/kulust/container-demo (ssh) ⌘1

```
[lumi][kulust@uan02-1025 container-demo]$ singularity run julia_latest.sif
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.2 (2024-03-01)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
[lumi][kulust@uan02-1026 container-demo]$ singularity inspect --runscript julia_latest.sif
#!/bin/sh
OCI_ENTRYPOINT='"docker-entrypoint.sh"'
OCI_CMD='"julia"'

# 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 "$SINGULARITY_NO_EVAL" ]; then
        # ENTRYPOINT only - run entrypoint plus args
        if [ -z "$OCI_CMD" ] && [ -n "$OCI_ENTRYPOINT" ]; then
                set -- 'docker-entrypoint.sh' "$@"
```

Running containers on LUMI

LUMI

- Use SLURM to run containers on compute nodes
- Use srun to execute MPI containers

```
srun singularity exec --bind ${BIND_ARGS} \
${CONTAINER_PATH} my_mpi_binary ${APP_PARAMS}
```

- **Be aware your container must be compatible with Cray MPI** (MPICH ABI compatible) for good performance
  - Configure suggestion: see next slide
- Open MPI based containers need workarounds and are not well supported on LUMI at the moment (and even more problematic for the GPU)

Environment enhancements (1)

LUMI

- LUMI specific tools for container interaction provided as modules
- **singularity-bindings/system** (available via easyconfig)
  - Sets the environment to use Cray MPICH provided outside the container
  - Requires a LUMI software stack
  - Use EasyBuild-user module and eb --search singularity-bindings to find the easyconfig or copy from our LUMI Software Library web site
  - Provides basic bind mounts for using the host MPI in the container setting `SINGULARITY_BIND` and `SINGULARITY_LD_LIBRARY_PATH`
- **lumi-vnc** (LUMI and CrayEnv software stacks)
  - Provides basic VNC virtual desktop for interacting with graphical interfaces via a web browser or VNC client
  - Open OnDemand a better alternative for many

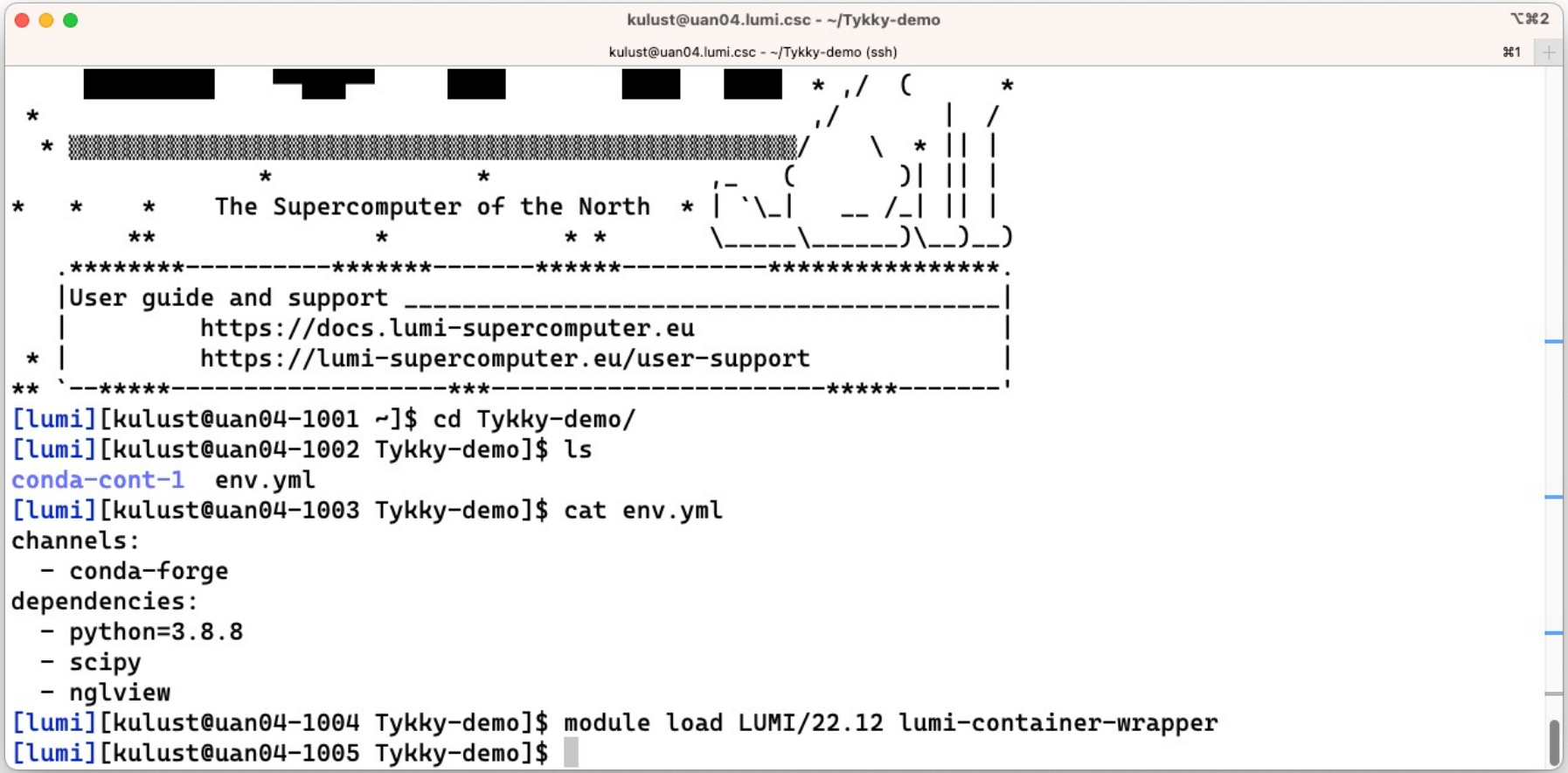
Environment enhancements (2) Containerising tools

LUMI

- **cotainr** (LUMI and CrayEnv software stacks)
  - A tool to pack conda installations in a singularity container
  - Use the singularity commands as shown on earlier slides to run
- **lumi-container-wrapper** (LUMI and CrayEnv software stacks)
  - Supports conda and pip environments
  - With pip: Python provided by the cray-python module (so there is an optimised NumPy etc.)
  - Software installation in two parts: a base container and a SquashFS file which is mounted in that container with the conda/pip environment
  - Provides wrappers to encapsulate your custom environment in a container (so you don't use singularity commands directly)
  - Still helps with quota on the number of files in your project and I/O performance

lumi-container-wrapper (1)

LUMI



lumi-container-wrapper (2)

LUMI

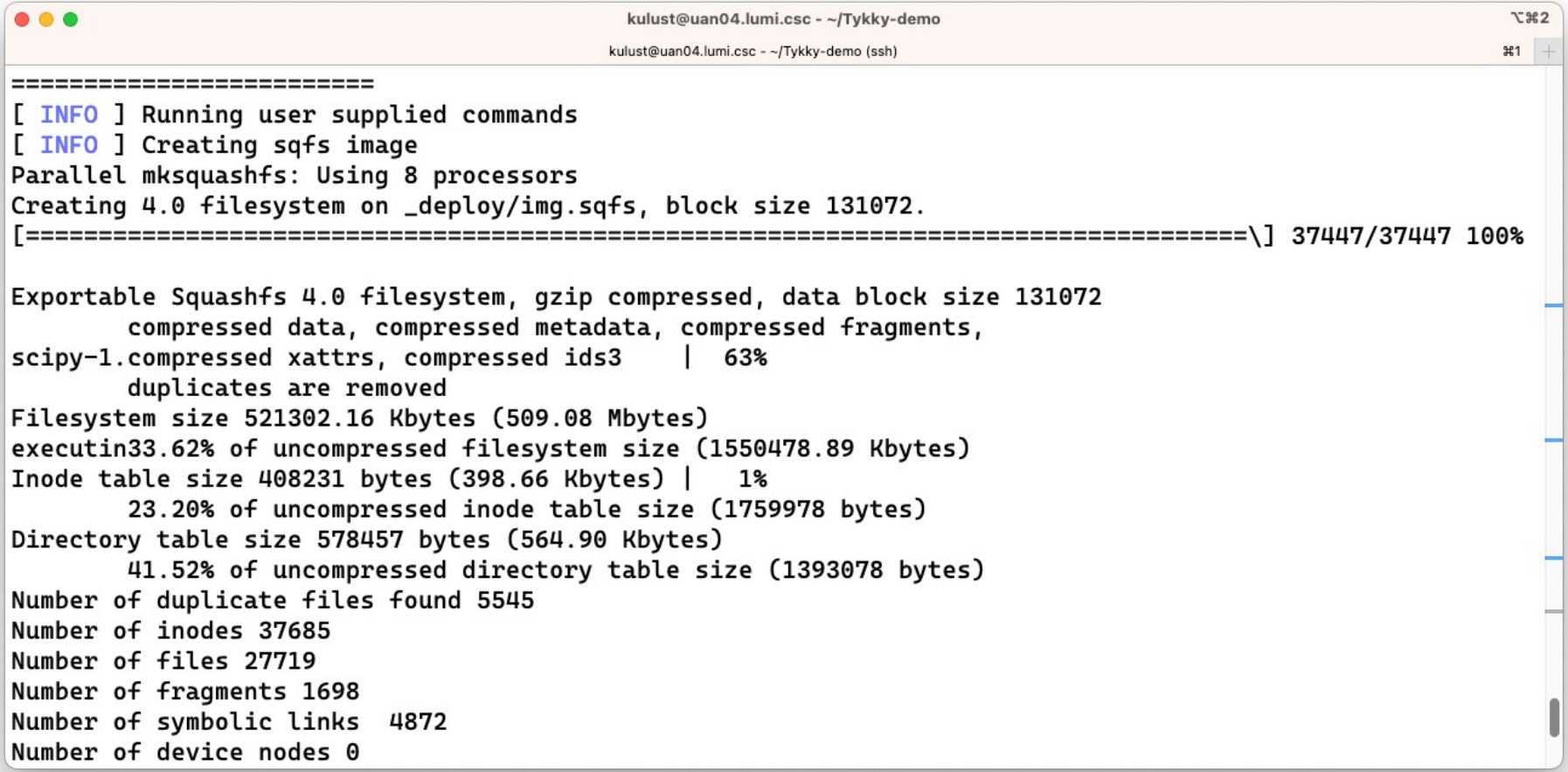
kulust@uan04.lumi.csc - ~/Tykky-demo ⌥⌘2

kulust@uan04.lumi.csc - ~/Tykky-demo (ssh) ⌘1

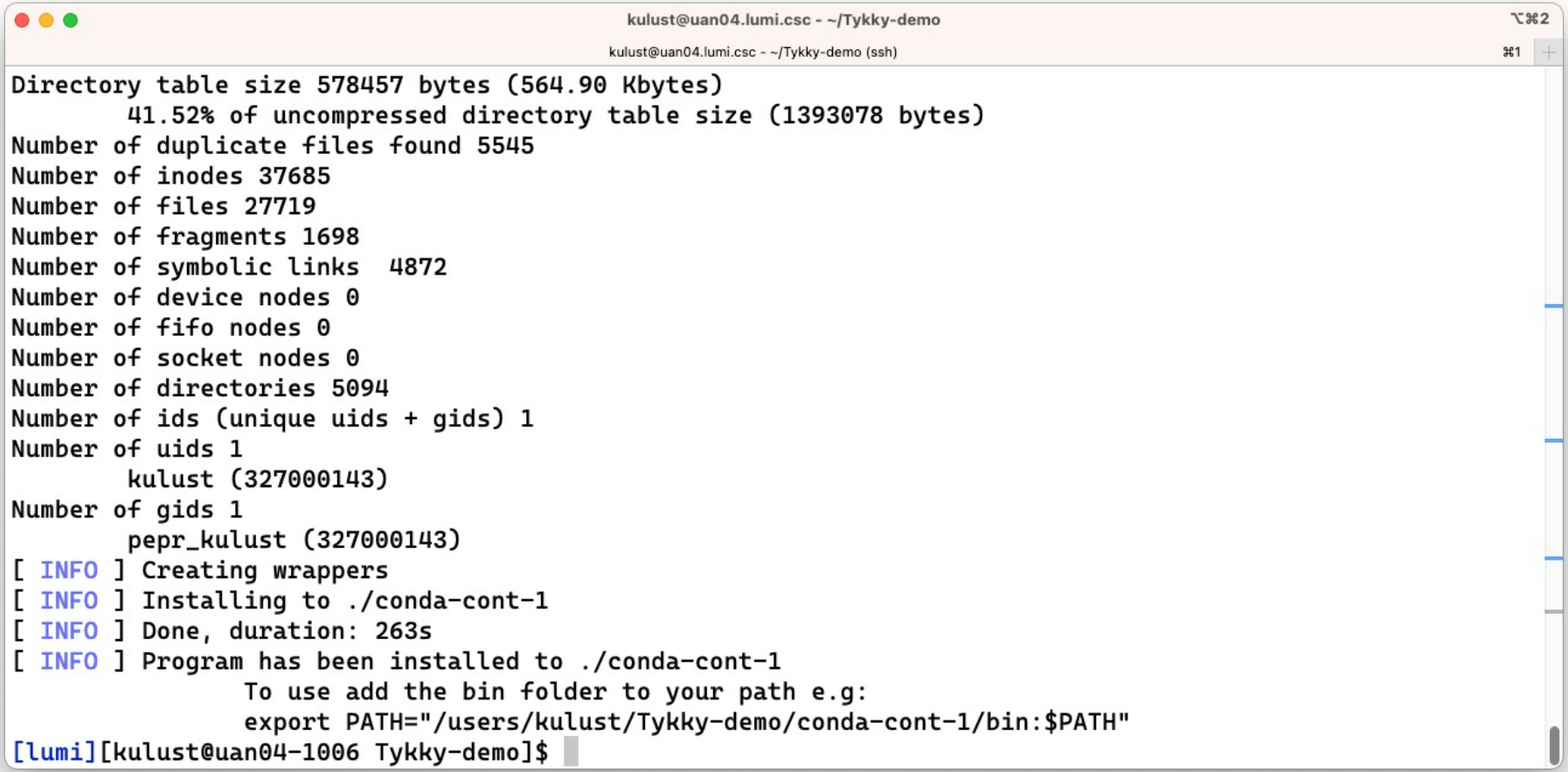
```
[lumi][kulust@uan04-1004 Tykky-demo]$ module load LUMI/22.12 lumi-container-wrapper
[lumi][kulust@uan04-1005 Tykky-demo]$ conda-containerize new --prefix ./conda-cont-1 env.yml
[ INFO ] Constructing configuration
[ INFO ] Using /tmp/kulust/cw-YSVL4M as temporary directory
[ INFO ] Fetching container docker://opensuse/leap:15.4
[ INFO ] Running installation script
[ INFO ] Using miniconda version Miniconda3-latest-Linux-x86_64
[ INFO ] Installing miniconda
==========================
PREFIX=/LUMI_TYKKY_4EJoer8/miniconda                                    P
REFIX=/LUMI_TYKKY_4EJoer8/miniconda                                     P

Preparing transaction: ...working... done
Executing transaction: ...working... done
installation finished.
WARNING:
    You currently have a PYTHONPATH environment variable set. This may cause
    unexpected behavior when running the Python interpreter in Miniconda3.
    For best results, please verify that your PYTHONPATH only points to
    directories of packages that are compatible with the Python interpreter
    in Miniconda3: /LUMI_TYKKY_4EJoer8/miniconda
==========================
[ INFO ] Creating env, full log in /tmp/kulust/cw-YSVL4M/build.log
```

lumi-container-wrapper (3)



lumi-container-wrapper (4)



lumi-container-wrapper (5)

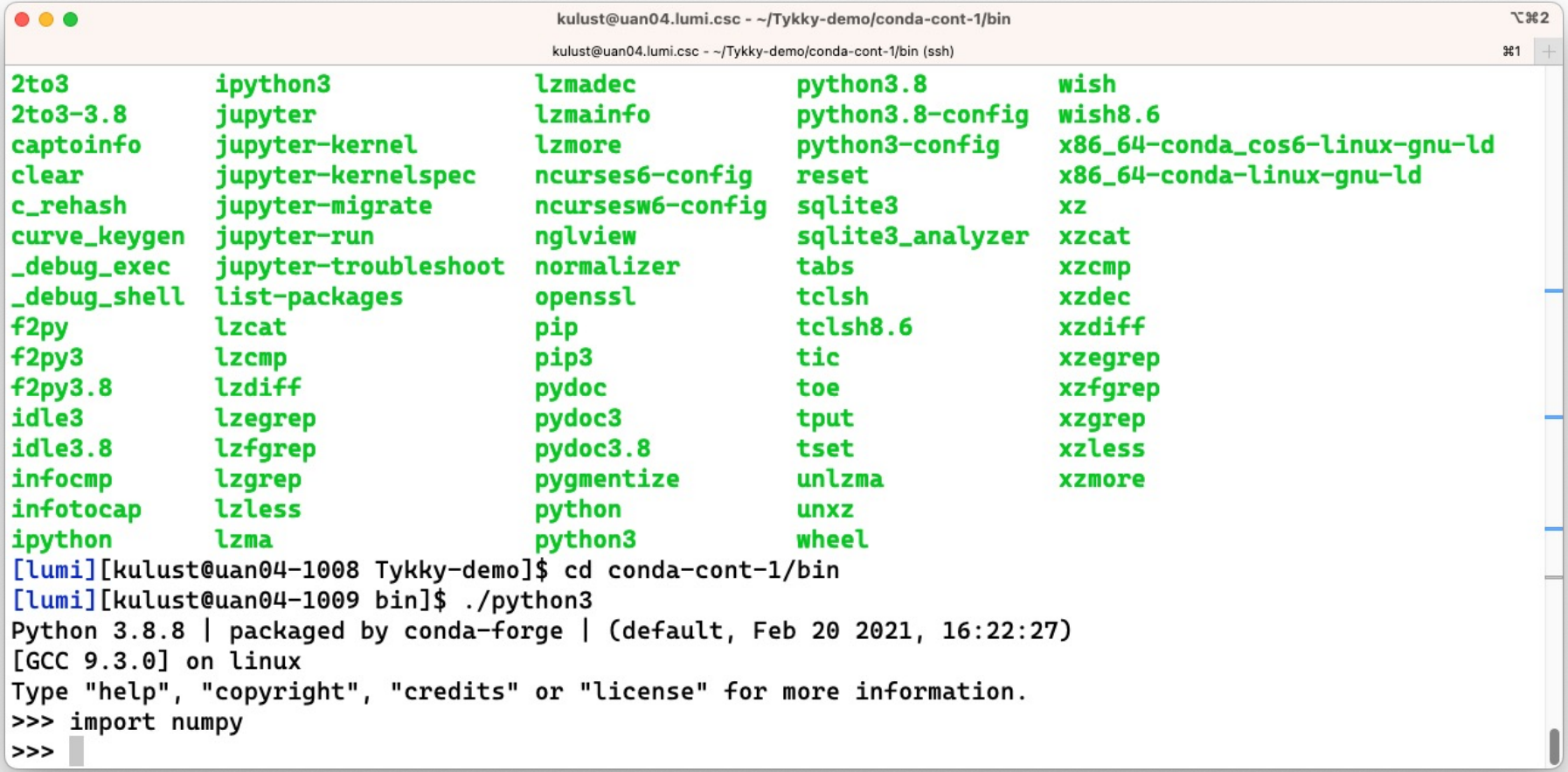
LUMI

kulust@uan04.lumi.csc - ~/Tykky-demo ⌥⌘2

kulust@uan04.lumi.csc - ~/Tykky-demo (ssh) ⌘1

```
[ INFO ] Program has been installed to ./conda-cont-1
               To use add the bin folder to your path e.g:
               export PATH="/users/kulust/Tykky-demo/conda-cont-1/bin:$PATH"
[lumi][kulust@uan04-1006 Tykky-demo]$ ls conda-cont-1/
_bin  bin  common.sh  container.sif  img.sqfs  share
[lumi][kulust@uan04-1007 Tykky-demo]$ ls conda-cont-1/bin
2to3           ipython3              lzmadec           python3.8         wish
2to3-3.8       jupyter               lzmainfo          python3.8-config  wish8.6
captoinfo      jupyter-kernel        lzmore            python3-config    x86_64-conda_cos6-linux-gnu-ld
clear          jupyter-kernelspec    ncurses6-config   reset             x86_64-conda-linux-gnu-ld
c_rehash       jupyter-migrate       ncursesw6-config  sqlite3           xz
curve_keygen   jupyter-run           nglview           sqlite3_analyzer  xzcat
_debug_exec    jupyter-troubleshoot  normalizer        tabs              xzcmp
_debug_shell   list-packages         openssl           tclsh             xzdec
f2py           lzcat                 pip               tclsh8.6          xzdiff
f2py3          lzcmp                 pip3              tic               xzegrep
f2py3.8        lzdiff                pydoc             toe               xzfgrep
idle3          lzegrep               pydoc3            tput              xzgrep
idle3.8        lzfgrep               pydoc3.8          tset              xzless
infocmp        lzgrep                pygmentize        unlzma            xzmore
infotocap      lzless                python            unxz
ipython        lzma                  python3           wheel
[lumi][kulust@uan04-1008 Tykky-demo]$
```

lumi-container-wrapper (6)



Environment enhancements (3): Prebuilt containers for AI (and some others)

LUMI

- Currently available
  - PyTorch: Best tested
  - TensorFlow
  - JAX
  - AlphaFold
  - ROCm and mpi4py
- Where to find?
  - `/appl/local/containers/sif-images`: Links to the latest version of each container
  - `/appl/local/containers/easybuild-sif-images`: Images for EasyBuild
    - Recommended for inexperienced users
  - `/appl/local/containers/tested-containers`: Images linked to and docker tarballs
- Recommend to keep your own copy of the image you depend upon!

Running the AI containers (Complicated way)

LUMI

- The containers have everything they need to use RCCL and/or MPI on LUMI
- Need to take care of bindings:
  - Need `-B /var/spool/slurmd,/opt/cray,/usr/lib64/libcxi.so.1,/usr/lib64/libjansson.so.4` at the minimum (and this list may change after a system update)
  - And add access to your space in `/project`, `/scratch` and/or `/flash` (default is only the home directory)
- Components that need further initialisation:
  - MIOpen
  - RCCL needs to be told the right network interfaces to use if you run across nodes
  - GPU-aware MPI may need to be set up (see earlier in the course)
  - Your AI package may need some too (e.g., `MASTER_ADDR` and `MASTER_PORT` for distributed learning with PyTorch)
- Containers with Python packages are built using Conda
  - Need to initialise the Conda environment via `$WITH_CONDA` in the container

Running the AI containers EasyBuild (1)

LUMI

- We provide EasyBuild recipes to "install" the containers and provide a module.
  - For those packages for which we know generic usage patterns, we provide some scripts that do most settings
  - Define a number of environment variables to make life easier, e.g., generic bindings and a variable referring to the container
  - Newer versions (will) come with a Python virtual environment pre-initialised to add your own packages
    - No more `$WITH_CONDA` needed as the module takes care of injecting environment variables in the container that have the same effect as the Conda and Python virtual environment activate scripts
    - Management of the Python virtual environment: Create a SquashFS file from the installation
- Someone with some EasyBuild experience may further extend the recipe to, e.g., already install extra packages

Running the AI containers EasyBuild (2)

LUMI

- Install:
  - Set up your user environment for EasyBuild (`EBU_USER_PREFIX`)
  - Run
    ```
    module load LUMI partition/container EasyBuild-user
    eb PyTorch-2.2.0-rocm-5.6.1-python-3.10-singularity-20240315.eb
    ```
  - After that the container module is available in all LUMI stacks and in CrayEnv
- Best to clean up afterwards before running (or take a new shell)
- Will copy the .sif-file to the software installation directory.
  - To delete:
    ```
    module load PyTorch/2.2.0-rocm-5.6.1-python-3.10-singularity-20240315
    rm –f $SIF
    module load PyTorch/2.2.0-rocm-5.6.1-python-3.10-singularity-20240315
    ```
  - At your own risk as we may remove the image in `/appl/local/containers` without notice

Running: Example: Distributed learning Without EasyBuild (1)

LUMI

- Create file `get-master.py`:

```
import argparse
def get_parser():
    parser = argparse.ArgumentParser(description="Extract master node name from Slurm node list",
             formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument("nodelist", help="Slurm nodelist")
    return parser

if __name__ == '__main__’:
    parser = get_parser()
    args = parser.parse_args()

    first_nodelist = args.nodelist.split(',')[0]

    if '[' in first_nodelist:
        a = first_nodelist.split('[‘)
        first_node = a[0] + a[1].split('-')[0]

    else:
        first_node = first_nodelist

    print(first_node)
```

Running: Example: Distributed learning Without EasyBuild (2)

LUMI

- Create file `run-pytorch.sh`:

```
#!/bin/bash -e

# Make sure GPUs are up
if [ $SLURM_LOCALID -eq 0 ] ; then
    rocm-smi
fi
sleep 2

$WITH_CONDA

# Set MIOpen cache to a temporary folder.
export MIOPEN_USER_DB_PATH="/tmp/$(whoami)-miopen-cache-$SLURM_NODEID"
export MIOPEN_CUSTOM_CACHE_DIR=$MIOPEN_USER_DB_PATH

if [ $SLURM_LOCALID -eq 0 ] ; then
    rm -rf $MIOPEN_USER_DB_PATH
    mkdir -p $MIOPEN_USER_DB_PATH
fi
sleep 2

# Set ROCR_VISIBLE_DEVICES so that each task uses the proper GPU
export ROCR_VISIBLE_DEVICES=$SLURM_LOCALID

# Report affinity
echo "Rank $SLURM_PROCID --> $(taskset -p $$)"

# Set interfaces to be used by RCCL.
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3

# Set environment for the app
export MASTER_ADDR=$(python get-master.py "$SLURM_NODELIST")
export MASTER_PORT=29500
export WORLD_SIZE=$SLURM_NPROCS
export RANK=$SLURM_PROCID

# Run app
python -u mnist_DDP.py --gpu --modelpath model
```

Running: Example: Distributed learning Without EasyBuild (3)

LUMI

- Create job script `my-job.sh`:

```
#!/bin/bash -e
#SBATCH --nodes=4
#SBATCH --gpus-per-node=8
#SBATCH --tasks-per-node=8
#SBATCH --output="output_%x_%j.txt”
#SBATCH --partition=standard-g
#SBATCH --mem=480G
#SBATCH --time=00:10:00
#SBATCH --account=project_<your_project_id>

PROJECT_DIR=/project/your_project/your_directory
SIF=/appl/local/containers/easybuild-sif-images/lumi-pytorch-rocm-5.6.1-python-3.10-pytorch-v2.2.0-dockerhash-7392c9d4dcf7.sif

c=fe
MYMASKS="0x${c}000000000000,0x${c}00000000000000,0x${c}0000,0x${c}000000,0x${c},0x${c}00,0x${c}00000000,0x${c}0000000000”

srun --cpu-bind=mask_cpu:$MYMASKS \
  singularity exec \
    -B /var/spool/slurmd \
    -B /opt/cray \
    -B /usr/lib64/libcxi.so.1 \
    -B /usr/lib64/libjansson.so.4 \
    -B $PROJECT_DIR:/workdir \
    $SIF /workdir/run-pytorch.sh
```

Running: Example: Distributed learning With EasyBuild

LUMI

- Create job script `my-job.sh`:

```
#!/bin/bash –e
#SBATCH --nodes=4
#SBATCH --gpus-per-node=8
#SBATCH --tasks-per-node=8
#SBATCH --output="output_%x_%j.txt”
#SBATCH --partition=standard-g
#SBATCH --mem=480G
#SBATCH --time=00:10:00
#SBATCH --account=project_<your_project_id>

module load CrayEnv PyTorch/2.2.0-rocm-5.6.1-python-3.10-singularity-20240315

c=fe
MYMASKS="0x${c}000000000000,0x${c}00000000000000,0x${c}0000,0x${c}000000,0x${c},
0x${c}00,0x${c}00000000,0x${c}0000000000”

srun --cpu-bind=mask_cpu:$MYMASKS \
  singularity exec $SIF \
    conda-python-distributed -u mnist_DDP.py --gpu --modelpath model
```

Extending container 1: cotainr

LUMI

- It is possible to use the ROCm containers in `/appl/local/containers/sif-images` as a base image for cotainr and build your own AI container
  - Be careful which version of the AI software you use as wheels are likely for a specific ROCm version (and you don't want to pick up wheels for NVIDIA)
  - MPI may be a problem as those containers do not yet provide a suitable mpi4py
- Process:
  - Create a yaml file with the setup for Conda (see notes)
  - Run cotainr:

```
module load LUMI/22.12 cotainr
cotainr build my-new-image.sif \
    --base-image=/appl/local/containers/sif-images/lumi-rocm-rocm-5.4.6.sif \
    --conda-env=py311_rocm542_pytorch.yml
```

- Run as a regular container
  - Or find someone who want to make an EasyConfig to create a module and point EasyBuild to the container .sif file with `--sourcepath`

Extending container 2: singularity build

LUMI

- Build a singularity-compatible container definition file, e.g.,

```
Bootstrap: localimage

From: /appl/local/containers/easybuild-sif-images/lumi-pytorch-
rocm-5.6.1-python-3.10-pytorch-v2.2.0-dockerhash-f72ddd8ef883.sif

%post

zypper -n install -y Mesa libglvnd libgthread-2_0-0 hostname
```

- And run:

```
module load LUMI/23.09 systools
singularity build my-new-container.sif my-container-definition.def
```

- Good way to add SUSE packages that may be needed to install extra software
- Tip: See demo 1: Start from a container with an EasyBuild module and the module might still work…

Extending container 3: Python virtual environment (1)

LUMI

- Some newer containers installed with EasyBuild have a pre-initialised virtual environment
  - In the container available as `/user-software/venv/<MyVEnv>`
  - Outside the container: `$CONTAINERROOT/user-software/venv/<MyVEnv>`
  - And `/user-software` can also be used to install other software if needed…
- How?

```
$> module load LUMI
$> module load PyTorch/2.2.0-rocm-5.6.1-python-3.10-singularity-20240315
$> singularity shell $SIF
Singularity> pip install pytorch-lightning
```

Extending container 3: Python virtual environment (2)

LUMI

- But what about the many small files?
  - Convert `$CONTAINERROOT/user-software` to a SquashFS file
    `make-squashfs`
    And reload the module…
  - You can then delete the `$CONTAINERROOT/user-software` subdirectory if you need the space (or file quota) and reconstruct it if needed with `unmake-squashfs`
  - To add additional packages afterwards:
    - Make sure the `$CONTAINERROOT/user-software` exists (outside the container)
    - Delete `$CONTAINERROOT/user-software.squashfs`
    - Reload the module
    - And start a shell in the container…
- You can of course do this with any container with Python, also when not using EasyBuild-built modules but the manual procedure takes a few more steps.

Container limitations on LUMI

LUMI

- Containers use the host's operating system kernel which may be different from your system. Containers do not abstract hardware.
- A generic container may not offer sufficiently good support for the Slingshot 11 interconnect on LUMI and fall back to TCP sockets resulting in poor performance, or not work at all.
  - Solution by injecting Cray MPICH, but only for containers with ABI compatibility with MPICH.
  - Distributed AI: Need to inject the proper RCCL plugin.
- AMD driver version may pose problems also.
- Only very limited support for building containers on LUMI due to security concerns.


LUMI
Questions?