Containers on LUMI

- Reasons to use containers on LUMI
  - **Productivity** - reproduce sophisticated user environment, ie. in Python
  - **Storage manageability** - lower pressure on filesystem (for software frameworks that access hundreds of thousands of small files) - for I/O performance and management of your disk quota
  - **Flexible service** - avoid complicated manual build process or installation impossible to maintain
- What do containers not necessary provide on LUMI
  - **Portability** – not every container will run on LUMI (expect problems with multi-node, memory distributed runs or GPU containers)
  - **Performance** – specific interconnect of LUMI may not be supported by generic containers or supported only with low performance

Managing containers

- Supported runtimes
  - Docker is **NOT** directly available from user environment
  - **Singularity** is natively available (as a system command) on a login and compute nodes
- Pulling containers
  - DockerHub and other registries (here Julia container as example)
    `singularity pull docker://julia`
  - Singularity uses flat (single) `sif` file for storing container and pull command makes the conversion
  - Beware of unsuccessful pulls – cache in `.singularity` dir or `$XDG_RUNTIME_DIR` can easily exhaust your storage quota for larger images
- Building containers
  - There is no building service provided on LUMI
  - You should either pull or copy containers from outside
  - Singularity can build from existing (base) container
  - We plan to provide a set of base LUMI images

Interacting with containers

- Accessing container with shell command
  `singularity shell container.sif`
- `Executing 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 `SINGULARITY_BINDPATH` env to mount other host directories (like `/projappl` or `/appl` on LUMI)

Running containers on 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 CrayMPI (MPICH ABI compatible)**
- OpenMPI based containers need workarounds and are not well supported on LUMI at the moment

Environment enhancements

- LUMI specific tools for container interaction provided as modules
- Require `LUMI` module (Software Stack module)
  - **`HPC-container-Wrapper`** (available in the Software Stack)
    - Provides wrappers to encapsulate your custom environment in the container
    - Supports conda and pip environments
    - Helps with quota on the number of files in your project and I/O performance
    - Python provided by the `cray-python` module
  - **`lumi-vnc`** (available in the Software Stack)
    - Provides basic VNC virtual desktop for interacting with graphical interfaces via web browser
  - **`singularity-bindings`** (available via easyconfig)
    - Use `EasyBuild-user` module and `eb --seach singularity-bindings` to find the easyconfig
    - Provides basic mount points for using host MPI in the container

Container limitations

- Container uses host's operating system kernel
- Interconnect may not be supported with generic container
- MPI requires ABI compatibility with MPICH
- Building containers is not currently supported on LUMI