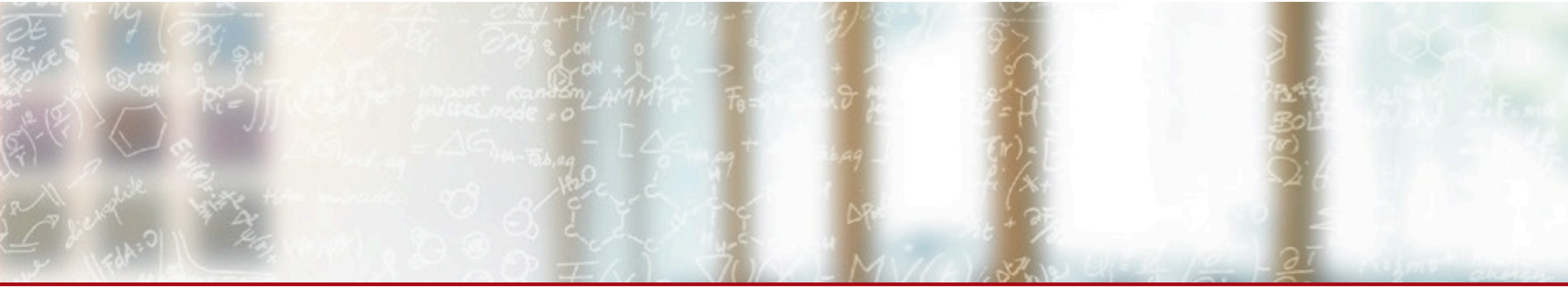
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Cray Programming Environments within Containers on Cray XC Systems

Maxime Martinasso
Swiss National Supercomputing Centre (CSCS), ETH Zurich, Lugano, Switzerland

Cray User Group 2019
Montréal, Canada

Containerizing a Cray Programing Environments

- Give access to all Cray modules and software inside a container
  - Identical setup as on the production machine
  - Run on any kind of machines (non-Cray, laptop,…)

- Improve maintainability of center provided applications:
  - Prepare for next CPE: easier testing and fixing, same procedure to build
  - Increase the degree of automation for regression building
  - Assess performance evolution of applications independently of the hardware
  - Test new CPE at scale on production machine (and not only on TDS)

- Manage user expectation
  - Acknowledge increase or decrease of performance
  - Minimize compiler bugs before CPE is installed in production

Cray Programing Environments (CPE)

- A software stack to run and build application
  - Scientific libraries
  - Communication libraries
  - Compilers and variety of tools
  - External vendor packages (Intel, Nvidia,…)

- CPE is provided inside CDT ISO files
  - Version number <year>.<month>-<number> (18.10-03PRE)

- CDT contains packages
  - RPMs
  - Cray specific installer
  - CDT are tailored to a Cray system
  - Download from CrayPort

- Install packages and create modules

Container technology

- Kernel namespaces:
  - Isolate and virtualize system resources (PID, FS, network)

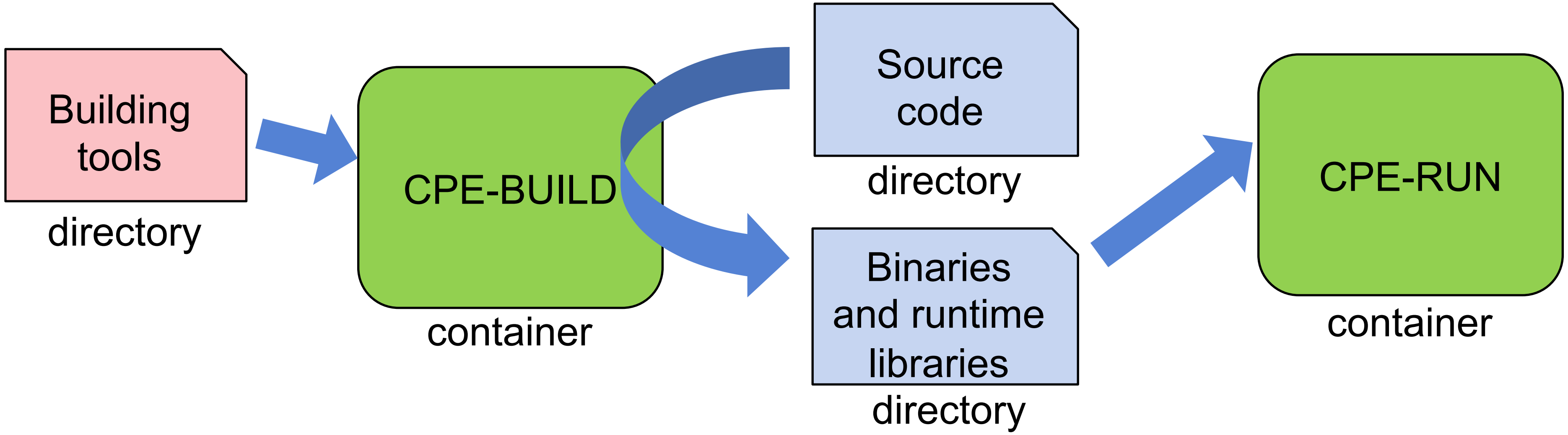
- Convenient way of packaging software stacks
  - Use a Linux distribution package manager
  - All software dependencies belong to the container
  - Kernel and drivers are outside of the container (cuda)

- Docker is the leader in container technology
  - HPC space: Singularity, Shifter, Sarus, CharlieCloud and others

CPE in container: Methodology

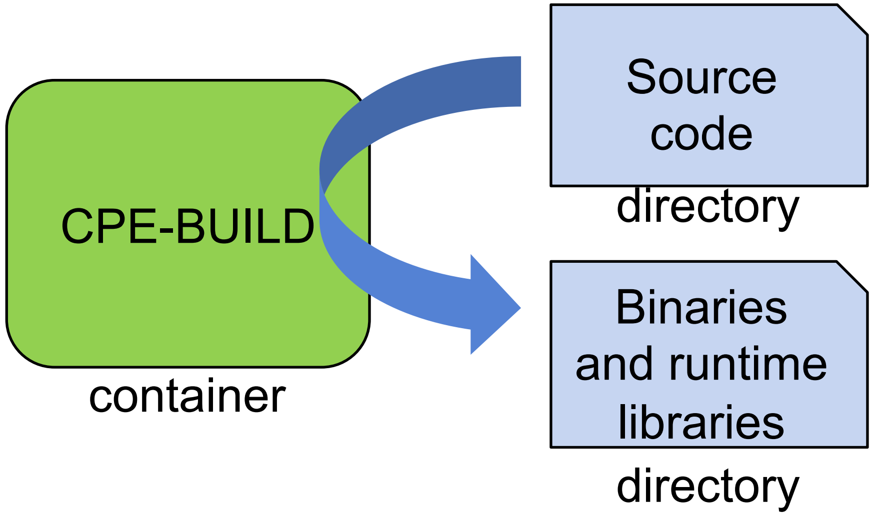
- A CPE-BUILD container can build any application
  - Input: source codes are mounted inside the container
  - Output: binaries are installed outside the container
  - One can add the required tools to build an app inside the container (easybuild,…)

- A CPE-RUN container can run an application built with a CPE-BUILD container
  - Hold the binaries and the runtime libraries
  - Special environment can be setup



CPE-BUILD

- Building the container with Docker
  - Create a Dockerfile
- Select a base image
  - Elogin node image, CLE
  - Use squashfs
- Install Cray packages
  - Configure CPE
  - Install Cray-installer
  - Install CPE
  - Install additional packages
- Setup environment
  - Input/output directories
  - Env. variables



CDT version	ISO file size	CPE container size
16.11-07	4.5 GB	34 GB
17.08-06	4.2 GB	32 GB
18.10-03	6.1 GB	50 GB

Table I
SIZE OF ISO FILE AND CPE CONTAINERS FOR VARIOUS VERSION OF CDT (INCLUDING THE CUDATOOLKIT PACKAGE).

Dockerfile(1)

Base image
Build arguments

Create user and input/output directories

Copy CDT RPMs

```
FROM elogin_prod:up07_20181205160931
ARG CDT_VERSION=18.10-03PRE
ARG CPU_TARGET=haswell
ARG ACCELERATORS=PASCAL
ARG CUDATOOLKIT=9.2

# Setup directories and pe user
RUN mkdir /root/${CDT_VERSION} && \
    mkdir /root/cuda && \
    mkdir /root/logs && \
    useradd -ms /bin/bash pe_user && \
    mkdir /home/pe_user/sources && \
    mkdir -p /home/pe_user/install/craype_runtime && \
    echo "CrayPe Version: cdt:${CDT_VERSION} cpu:${CPU_TARGET}
          acc:${ACCELERATORS} cudatoolkit:${CUDATOOLKIT}"\
       > /home/pe_user/install/craype_runtime/craype_version.txt

COPY volume/${CDT_VERSION} /root/${CDT_VERSION}
COPY cuda/ /root/cuda/
```

Dockerfile(2)

Setup and Install CDT

Install missing RPMs

Setup env.
to use modules

```
# Edit configuration and install packages
RUN cd /root/${CDT_VERSION}/installer && \
    rpm -ivh craype-installer-*.rpm --upgrade && \
    cp /opt/cray/craype-installer/default/conf/install-cdt.yaml /root && \
    sed -i -e "s/LOGS_DIR: NEED-TO-SPECIFY/LOGS_DIR : \/root\/logs/" \
        -e "s/ISO_MOUNT_DIR: NEED-TO-SPECIFY/ISO_MOUNT_DIR : \/root\/${CDT_VERSION}/" \
        -e "s/INSTALL_PGI_LIBRARIES: NO/INSTALL_PGI_LIBRARIES : YES/" \
        -e "s/INSTALL_INTEL_LIBRARIES: NO/INSTALL_INTEL_LIBRARIES : YES/" \
        -e "s/CRAY_CPU_TARGET: NEED-TO-SPECIFY/CRAY_CPU_TARGET : ${CPU_TARGET}/" \
        -e "s/ACCELERATORS: NONE/ACCELERATORS : ${ACCELERATORS}/" \
        /root/install-cdt.yaml && \
    cd /root/ && \
    /opt/cray/craype-installer/default/bin/craype-installer.pl --install \
                        --install-yaml-path install-cdt.yaml --network ari && \
    rpm -ivh /root/cuda/cray-cudatoolkit${CUDATOOLKIT}-*.rpm \
            /root/cuda/cray-nvidia-libcuda-396.44_3.1.33-6.0.7.1_3.2__gac01daf.ari.x86_64.rpm

USER pe_user
WORKDIR /home/pe_user/sources

ENV MODULEPATH /opt/cray/pe/perftools/default/modulefiles:/opt/cray/pe/craype/default/modulefiles:
            /opt/cray/pe/modulefiles:/opt/cray/modulefiles:/opt/modulefiles:/opt/cray/ari/modulefiles:
            /opt/cray/craype/default/modulefiles
```

Create base image
Import squashfs image into Docker

```
# build and import the base image
$ sudo unsquashfs -f -d unsquashfs elogin_prod_up07_20181205160931.squashfs
$ sudo tar -C unsquashfs -c . | docker import - elogin_prod:up07_20181205160931
```

Directory structure and file locations

```
# directory structure
$ ls .
Dockerfile cuda volume

$ ls cuda/
cray-cudatoolkit8.0-8.0.61_2.4.9-6.0.7.0_17.1__g899857c.x86_64.rpm
cray-cudatoolkit9.0-9.0.103_3.15-6.0.7.0_14.1__ge802626.x86_64.rpm
cray-cudatoolkit9.2-9.2.148_3.19-6.0.7.1_2.1__g3d9acc8.x86_64.rpm
cray-nvidia-libcuda-390.46_3.1.30-6.0.7.0_24.8__g83596c3.ari.x86_64.rpm
cray-nvidia-libcuda-396.44_3.1.33-6.0.7.1_3.2__gac01daf.ari.x86_64.rpm

$ ls volume/16.11-07/
TRANS.TBL    conf    docs    installer    packages    release_info
```

Build the container with build arguments

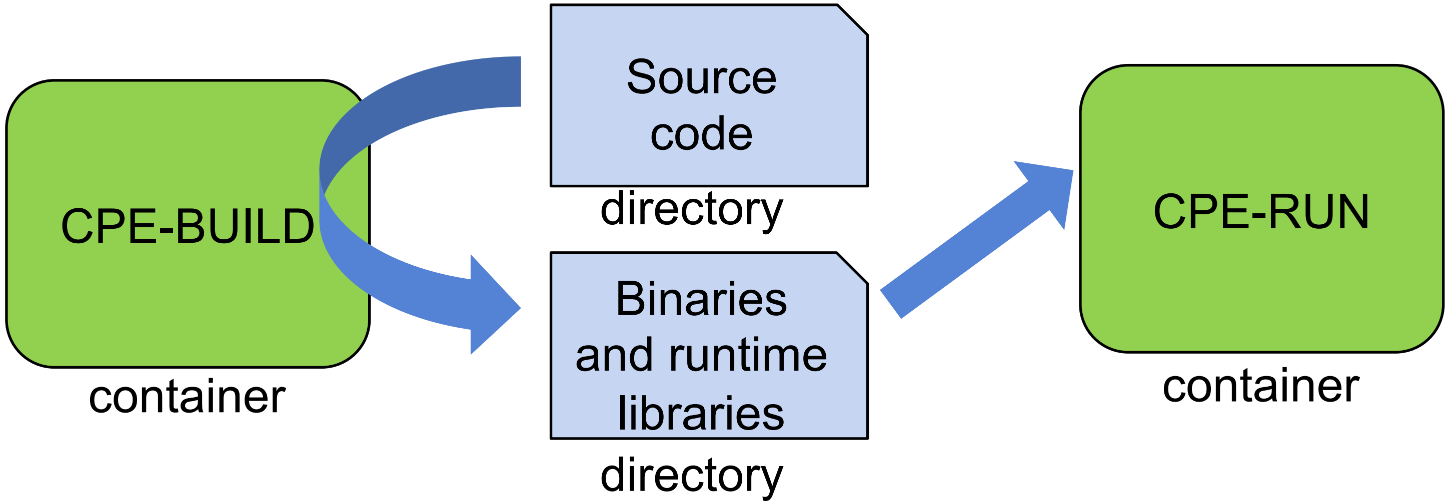
```
$ docker build --build-arg CDT_VERSION=16.11-07 --build-arg CUDATOOLKIT=8.0 \
        -t craype:cdt16.11-07.haswell.pascal.cudatoolkit8.0 .
```

Start interactively the CPE-BUILD, set input and output directories

```
$ docker run -v /Users/maximem/dev/docker/my_source:/home/pe_user/sources \
        -v /Users/maximem/dev/pe_container/my_binaries:/home/pe_user/install \
        --rm -it craype:cdt16.11-07.haswell.pascal.cudatoolkit8.0 /bin/bash
```

CPE-RUN: Lightweight container

- Compiled binary
  - Outside of container
  - Dependency inside container
- Extract dependencies
  - Python script *ldd* and *strings*
  - Lookup for *dlsym* libraries



- Binaries and dependencies are inside one directory outside of CPE-BUILD
  - Option 1: Copy directory, set *LD_LIBRARY_PATH* and run
  - Option 2: Copy inside a lightweight container with proper env. setup

Example of script output

```
$ ldd_parser --binaries /opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libfi.so
/opt/cray/pe/gcc-libs/libstdc++.so.6
/opt/cray/pe/gcc-libs/libgfortran.so.3
/opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libf.so.1
/opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libcsup.so.1
/opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libu.so.1
/opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libcraymath.so.1
# Duplicated reference of libraries:
/opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libquadmath.so.0
/opt/gcc/6.2.0/snos/lib/../lib64/libquadmath.so.0
/opt/cray/pe/gcc-libs/libgcc_s.so.1
/opt/gcc/6.2.0/snos/lib/../lib64/libgcc_s.so.1
# dlsym libraries:
libmemkind.so.0
libnuma.so
```

Limitations and challenges

- There is a variability in installed packages depending on the CDT version
  - Getting more regular with later CDT versions
  - Not all packages are installed
  - Might need a Dockerfile per CDT version

- Cuda and drivers
  - Careful which version of the driver is installed, backward compatibility works
  - Forward compatibility: cuda10

- Identifying dependencies
  - Not automated, need a human in the loop to check script output

- Built binaries should run on a Cray system
  - Careful with the license agreement

Use case: Reproducible experiments

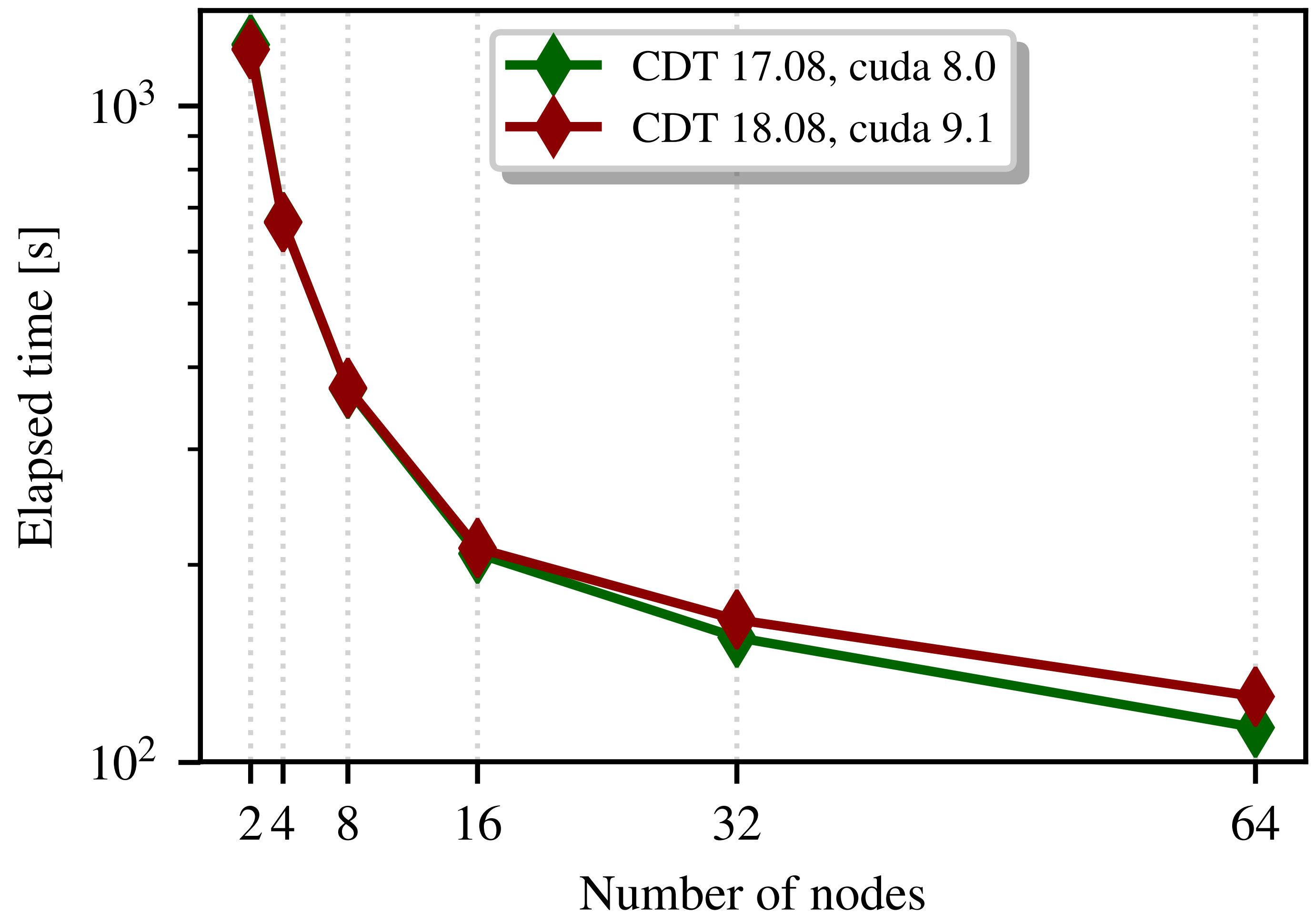
- Institute for Atmospheric and Climate Science
  - COSMO: weather and climate application that has been ported to GPU
  - COSMO: Fortran + OpenACC and C++ with Cuda
  - COSMO-OPCODE version: no major update since 2015
  - Ensure reproducible experiments over 2 to 4 years time due to publication requirements

- Built with CDT 16.11 at the time of publication
  - Consume a large effort to update COSMO-OPCODE to new CPE
  - Current is 18.09, does not compile with CCE > 8.5.5, no resource to investigate
  - Install CDT 16.11 on Piz Daint (number of deployed CPEs, image size)

- Reproducibility, built within a container and run on Piz Daint
  - It passes the COSMO-OPCDE test suite
  - Produces a "useful climate”? (expert to check)

Use case: Performance variability

- CP2K built with 2 CPEs
  - 17.08 Cuda 8.0
  - 18.08 Cuda 9.1
- Standard benchmarks
  - H2O_256
  - 1 MPI +12 OpenMP / node
  - Piz Daint, 12 cores, P100
- CDT 17.08 scales better
  - 10% faster at 64 nodes
  - Was MPI getting slower?



Conclusion

- CPE in container is easy and helpful
  - Same procedure to install CPEs
  - No extra requirements on the container technology

- Should Cray provide containers instead of CDT iso files?
- Should Cray provide a package manager to use inside a Dockerfile?

- Open a build service with containerized CPEs to users?
  - Maintain multiple CPEs per applications
  - Easy integration with various building tools

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Thank you for your attention.