MAKING CONTAINERS EASIER WITH HPC CONTAINER MAKER

Scott McMillan (smcmillan@nvidia.com)

October 16, 2019

DIFFERENT ROLES. SAME GOALS.

Driving Productivity and Faster Time-to-Solutions

Data Scientists and Researchers

Eliminate mundane tasks, focus on science and research

Developers

Minimize support, focus on algorithms and code

Sysadmins

Simplify deployments, focus on uptime and new capabilities

CHALLENGES UTILIZING AI & HPC SOFTWARE

EXPERTISE

Building AI-centric solutions requires expertise

INSTALLATION

Complex, time consuming, and error-prone

OPTIMIZATION

Requires expertise to optimize framework performance

PRODUCTIVITY

Users limited to older features and lower performance

MAINTAINENCE

IT can’t keep up with frequent software upgrades

CONTAINERS – SIMPLIFYING AI & HPC WORKFLOWS

EMBEDDING EXPERTISE

Deliver greater value, faster

FASTER DEPLOYMENTS

Eliminates installations. Simply Pull & Run the app

OPTIMIZED SOFTWARE

Key DL frameworks updated monthly for perf optimization

HIGHER PRODUCTIVITY

Better Insights and faster time-to-solution

ZERO MAINTENANCE

Empowers users to deploy the latest versions with IT support

NGC CONTAINERS: ACCELERATING WORKFLOWS

WHY CONTAINERS

Simplifies Deployments

- Eliminates complex, time-consuming builds and installs

Get started in minutes

- Simply Pull & Run the app

Portable

- Deploy across various environments, from test to production with minimal changes

WHY NGC CONTAINERS

Optimized for Performance

- Monthly DL container releases offer latest features and superior performance on NVIDIA GPUs

Scalable Performance

- Supports multi-GPU & multi-node systems for scale-up & scale-out environments

Designed for Enterprise & HPC environments

- Supports Docker & Singularity runtimes

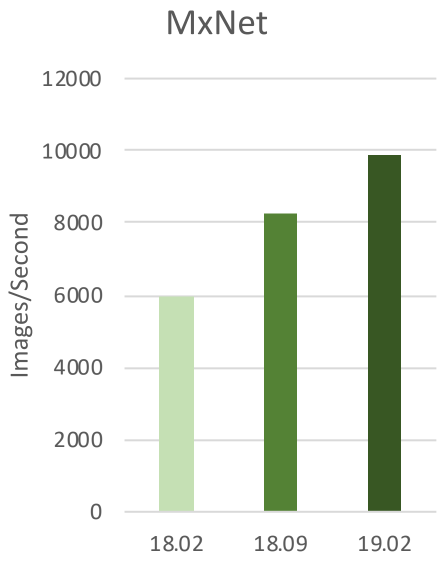
Run Anywhere

- Pascal/Volta/Turing-powered NVIDIA DGX, PCs, workstations, servers and top cloud platforms

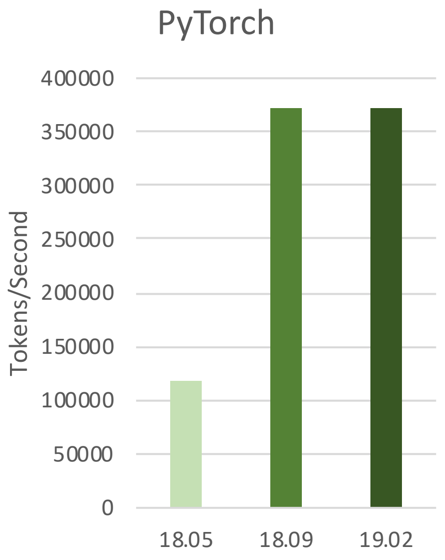
CONTINUOUS PERFORMANCE IMPROVEMENT

Developers' Software Optimizations Deliver Better Performance on the Same Hardware

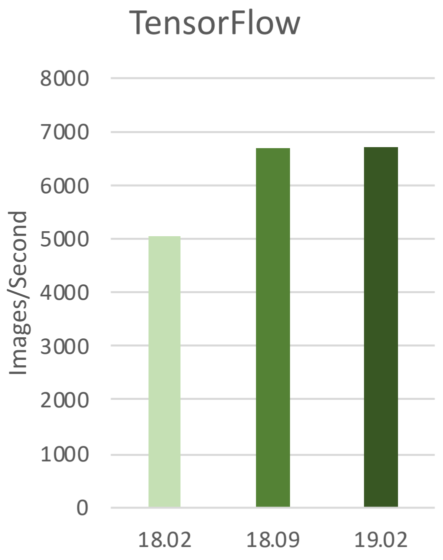
Monthly DL Framework Updates & HPC Software Stack Optimizations Drive Performance



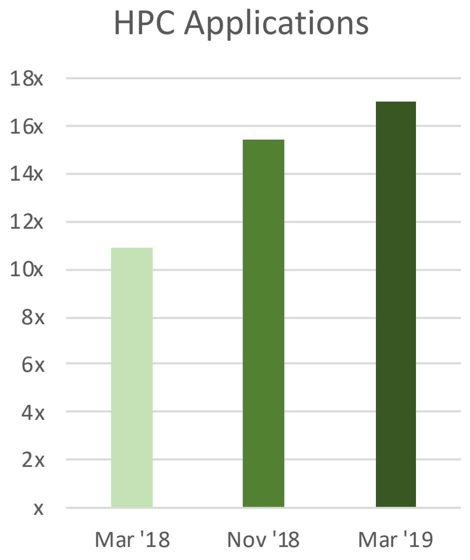
Mixed Precision | 128 Batch Size | ResNet-50 Training | 8x V100



Mixed Precision | 128 Batch Size | GNMT | 8x V100



Mixed Precision | 256 Batch Size | ResNet-50 Training | 8x V100



Speedup across Chroma, GROMACS, LAMMPS, QE, MILC, VASP, SPECFEM3D, NAMD, AMBER, GTC, RTM | 4x V100 v. Dual-Skylake | CUDA 9 for Mar '18 & Nov '18, CUDA 10 for Mar '19

GPU-OPTIMIZED SOFTWARE CONTAINERS

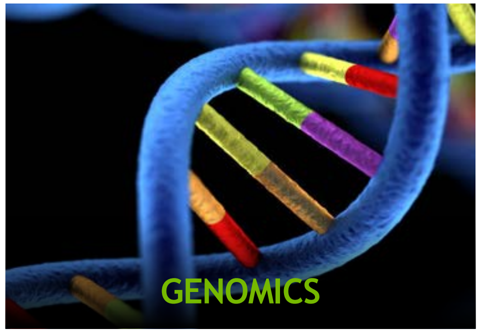
Over 50 Containers on NGC

TensorFlow | PyTorch | more

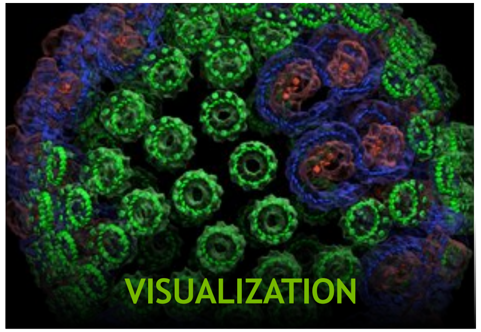
RAPIDS | H2O | more

TensorRT | DeepStream | more

NAMD | GROMACS | more



Parabricks



ParaView | IndeX | more

WHAT IF A CONTAINER IMAGE IS NOT AVAILABLE FROM NGC?

BUILDING CONTAINER IMAGES FROM SCRATCH

Recommended workflow:

1. Specify the content of container images with HPC Container Maker
2. Build container images with Docker
3. Convert the Docker images to Singularity images
4. Use Singularity to run containers on your HPC system

BUILDING CONTAINER IMAGES WITH SINGULARITY

HELLO WORLD!

- A Singularity definition file is a plain text file that specifies the instructions to create your container image.

```
BootStrap: docker
From: ubuntu:16.04

%files
    # Copy Hello World source code into the container image
    sources/hello.c /var/tmp/hello.c

%post
    # Install the necessary development environment
    apt-get update -y
    apt-get install -y --no-install-recommends build-essential gcc
    rm -rf /var/lib/apt/lists/*

    # Build Hello World
    gcc -o /usr/local/bin/hello /var/tmp/hello.c
```

https://www.sylabs.io/guides/latest/user-guide/quick_start.html#singularity-definition-files

HELLO WORLD!?!

- Build and run the container image file

```
$ sudo singularity build hello-world.sif Singularity.def

$ singularity run hello-world.sif hello
Hello World!
```

- The container image is a single file, hello-world.sif
- The Hello World image is 93 MB
  (the base Ubuntu image is 36 MB and the Hello World binary is 10 KB)

HELLO WORLD?

```
BootStrap: docker
From: ubuntu:16.04

%files
    # Copy Hello World source code into the container image
    sources/hello.c /var/tmp/hello.c

%post
    # Install the necessary development environment
    apt-get update -y
    apt-get install -y --no-install-recommends build-essential gcc
    rm -rf /var/lib/apt/lists/*

    # Build Hello World
    gcc -o /usr/local/bin/hello /var/tmp/hello.c

    # cleanup
    rm -rf /var/tmp/hello.c
    apt-get purge -y build-essential gcc
    apt autoremove -y
```

BUILDING CONTAINER IMAGES WITH DOCKER

DOCKERFILES

- A Dockerfile is a plain text file that specifies the instructions to create your container image

```
FROM ubuntu:16.04

RUN date > /build-info.txt
RUN uname -r >> /build-info.txt
```

https://docs.docker.com/engine/reference/builder/

IMAGE LAYERS

- Build and run the container image

```
$ sudo docker build -t layer-example -f Dockerfile .

$ sudo docker run --rm -it layer-example cat /etc/build-info.txt
Mon Oct  7 17:32:21 UTC 2019
4.4.115-1.el7.elrepo.x86_64
```

- Inspect the image

```
$ sudo docker history layer-example

IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
aa96aa9e145e        9 seconds ago       /bin/sh -c uname -r >> /build-info.txt          57B
f90d395de987        11 seconds ago      /bin/sh -c date > /build-info.txt               29B
4a689991aa24        11 months ago       /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           11 months ago       /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>           11 months ago       /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>           11 months ago       /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   745B
<missing>           11 months ago       /bin/sh -c #(nop) ADD file:01a5c4f2b1dcc8f8a…   116MB
```

CONTAINER IMAGE FORMATS

- Singularity images
  - "Flat"
  - Single file
  - Signing and encryption
- "Docker" images
  - Layered
  - Opaque
  - Open Containers Initiative (OCI) standard

Singularity can easily import Docker images

HELLO WORLD!

```
# Start from a basic Ubuntu 16.04 image
FROM ubuntu:16.04

# Install the necessary development environment
RUN apt-get update -y && \
    apt-get install -y --no-install-recommends \
        build-essential \
        gcc && \
    rm -rf /var/lib/apt/lists/*

# Copy Hello World source code into the container image
COPY sources/hello.c /var/tmp/hello.c

# Build Hello World
RUN gcc -o /usr/local/bin/hello /var/tmp/hello.c
```

HELLO WORLD!?!

- Build and run the container image file

```
$ sudo docker build -t hello-world -f Dockerfile .

$ sudo docker run --rm -it hello-world hello
Hello World!
```

- The container image is opaque
- The Hello World image is 308 MB
(the base Ubuntu image is 116 MB and the Hello World binary is 10 KB)

MULTISTAGE HELLO WORLD!

```
# Start from a basic Ubuntu 16.04 image
FROM ubuntu:16.04 AS build

# Install the necessary development environment
RUN apt-get update -y && \
    apt-get install -y --no-install-recommends \
        build-essential \
        gcc && \
    rm -rf /var/lib/apt/lists/*

# Copy Hello World source code into the build stage
COPY sources/hello.c /var/tmp/hello.c

# Build Hello World
RUN gcc -o /usr/local/bin/hello /var/tmp/hello.c
```

"Build" stage

```
# Start from a basic Ubuntu 16.04 image
FROM ubuntu:16.04

# Copy the hello binary from the build stage
COPY --from=build /usr/local/bin/hello /usr/local/bin/hello
```

"Runtime" stage

BARE METAL VS. CONTAINER WORKFLOWS

Login to system (e.g., CentOS 7 with Mellanox OFED 3.4)

```
$ module load PrgEnv/GCC+OpenMPI

$ module load cuda/9.0

$ module load gcc

$ module load openmpi/1.10.7
```

Steps to build application

Result: application binary suitable for that particular bare metal system

```
FROM nvidia/cuda:9.0-devel-centos7
```

OPENMPI DOCKERFILE VARIANTS

Real examples – which one should you use?

A

```
RUN apt-get update \
  && apt-get install -y --no-install-recommends \
    libopenmpi-dev \
    openmpi-bin \
    openmpi-common \
  && rm -rf /var/lib/apt/lists/*
ENV LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/openmpi/lib
```

B

```
RUN OPENMPI_VERSION=3.0.0 && \
    wget -q -O - https://www.open-
mpi.org/software/ompi/v3.0/downloads/openmpi-
${OPENMPI_VERSION}.tar.gz | tar -xzf - && \
    cd openmpi-${OPENMPI_VERSION} && \
    ./configure --enable-orterun-prefix-by-default --with-cuda --
with-verbs \
                --prefix=/usr/local/mpi --disable-getpwuid && \
    make -j"$(nproc)" install && \
    cd .. && rm -rf openmpi-${OPENMPI_VERSION} && \
    echo "/usr/local/mpi/lib" >> /etc/ld.so.conf.d/openmpi.conf &&
ldconfig
ENV PATH /usr/local/mpi/bin:$PATH
```

C

```
COPY openmpi /usr/local/openmpi
WORKDIR /usr/local/openmpi
RUN /bin/bash -c "source /opt/pgi/LICENSE.txt && CC=pgcc CXX=pgc++
F77=pgf77 FC=pgf90 ./configure --with-cuda --
prefix=/usr/local/openmpi”
RUN /bin/bash -c "source /opt/pgi/LICENSE.txt && make all install"
```

D

```
RUN mkdir /logs
RUN wget -nv https://www.open-
mpi.org/software/ompi/v1.10/downloads/openmpi-1.10.7.tar.gz && \
    tar -xzf openmpi-1.10.7.tar.gz && \
    cd openmpi-*&& ./configure --with-cuda=/usr/local/cuda \
    --enable-mpi-cxx --prefix=/usr 2>&1 | tee /logs/openmpi_config
&& \
    make -j 32 2>&1 | tee /logs/openmpi_make && make install 2>&1
| tee /logs/openmpi_install && cd /tmp \
    && rm -rf openmpi-*
```

E

```
WORKDIR /tmp
ADD http://www.open-
mpi.org//software/ompi/v1.10/downloads/openmpi-1.10.7.tar.gz /tmp
RUN tar -xzf openmpi-1.10.7.tar.gz && \
    cd openmpi-*&& ./configure --with-cuda=/usr/local/cuda \
    --enable-mpi-cxx --prefix=/usr && \
    make -j 32 && make install && cd /tmp \
    && rm -rf openmpi-*
```

F

```
RUN wget -q -O - https://www.open-
mpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.bz2 | tar -
xjf - && \
    cd openmpi-3.0.0 && \
    CXX=pgc++ CC=pgcc FC=pgfortran F77=pgfortran ./configure --
prefix=/usr/local/openmpi --with-cuda=/usr/local/cuda --with-verbs
--disable-getpwuid && \
    make -j4 install && \
    rm -rf /openmpi-3.0.0
```

OPENMPI DOCKERFILE VARIANTS

Real examples – which one should you use?

A

```
RUN apt-get update \
  && apt-get install -v --no-install-recommends \
    libopen
    openmpi
    openmpi
  && rm -rf
ENV LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/openmpi/lib
```

From Linux distribution
What version? How was it built?

B

```
RUN OPENMPI_VERSION=3.0.0 && \
    wget -q -O - https://www.open-
mpi.org/software/ompi/v3.0/downloads/openmpi-
${OPENMPI_VERSION}.tar.gz | tar -xzf - && \
    cd openmpi-${OPENMPI_VERSION} && \
    ./configure                              --with-cuda --
with-verbs \
                                             getpwuid && \
    make -j"$(nproc)
    cd .. && rm -rf openmpi-${OPENMPI_VERSION} && \
    echo "/usr/local/mpi/lib" >> /etc/ld.so.conf.d/openmpi.conf &&
ldconfig
ENV PATH /usr/local/mpi/bin:$PATH
```

Pretty good build, but…
different version

C

```
COPY openmpi /usr/local/openmpi
WORKDIR
RUN /bin                                          X=pgc++
F77=pgf7
prefix=/
RUN /bin/bash -c "source /opt/pgi/LICENSE.txt && make all install"
```

Bad layering, uses local source tree
and PGI compiler?

D

```
RUN mkdir /logs
RUN wget -nv https://www.open-
mpi.org/software/ompi/v1.10/downloads/openmpi-1.10.7.tar.gz && \
    tar -xzf                                          cuda \
    cd openmp                                         enmpi_config
    --enable-
&& \
    make -j 32 2>&1 | tee /logs/openmpi_make && make install 2>&1
| tee /logs/openmpi_install && cd /tmp \
    && rm -rf openmpi-*
```

Not bad, but what's with the
build logs?

E

```
WORKDIR /tmp
ADD http://www.open-
mpi.org//s                                     .gz /tmp
RUN tar -x
    cd ope                                     la \
    --enab
    make -j 32 && make install && cd /tmp \
    && rm -rf openmpi-*
```

Redistributes source tarball with
container image

F

```
RUN wget -q -O - https://www.open-
mpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.bz2 | tar -
xjf - && \
    cd openmpi-3.0.0
    CXX=pgc++ CC=pgcc                          an ./configure --
prefix=/usr/local/ope                          al/cuda --with-verbs
--disable-getpwuid &&
    make -j4 install && \
    rm -rf /openmpi-3.0.0
```

Different version
PGI compiler?

HPC CONTAINER MAKER

HPC CONTAINER MAKER

- Tool for creating HPC application Dockerfiles and Singularity definition files
- Makes it easier to create HPC application containers by encapsulating HPC & container best practices into building blocks
- Open source (Apache 2.0)
  https://github.com/NVIDIA/hpc-container-maker
- `pip install hpccm`

BUILDING BLOCKS TO CONTAINER RECIPES

```
Stage0 += openmpi()
```

hpccm ⬇ Generate corresponding Dockerfile instructions for the HPCCM building block

```
# OpenMPI version 3.1.2
RUN yum install -y \
        bzip2 file hwloc make numactl-devel openssh-clients perl tar wget && \
    rm -rf /var/cache/yum/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp https://www.open-
mpi.org/software/ompi/v3.1/downloads/openmpi-3.1.2.tar.bz2 && \
    mkdir -p /var/tmp && tar -x -f /var/tmp/openmpi-3.1.2.tar.bz2 -C /var/tmp -j && \
    cd /var/tmp/openmpi-3.1.2 &&  CC=gcc CXX=g++ F77=gfortran F90=gfortran FC=gfortran ./configure --
prefix=/usr/local/openmpi --disable-getpwuid --enable-orterun-prefix-by-default --with-cuda=/usr/local/cuda --with-verbs
&& \
    make -j4 && \
    make -j4 install && \
    rm -rf /var/tmp/openmpi-3.1.2.tar.bz2 /var/tmp/openmpi-3.1.2
ENV LD_LIBRARY_PATH=/usr/local/openmpi/lib:$LD_LIBRARY_PATH \
    PATH=/usr/local/openmpi/bin:$PATH
```

HIGHER LEVEL ABSTRACTION

Building blocks to encapsulate best practices, avoid duplication, separation of concerns

```
openmpi(check=False,                                  # run “make check”?
        configure_opts=['--disable-getpwuid', …],     # configure command line options
        cuda=True,                                    # enable CUDA?
        directory='',                                 # path to source in build context
        infiniband=True,                              # enable InfiniBand?
        ospackages=['bzip2', 'file', 'hwloc', …],     # Linux distribution prerequisites
        prefix='/usr/local/openmpi',                  # install location
        toolchain=toolchain(),                        # compiler to use
        ucx=False,                                    # enable UCX?
        version='4.0.1')                              # version to download
```

Examples:

```
openmpi(prefix='/opt/openmpi', version='1.10.7’)
openmpi(infiniband=False, toolchain=pgi.toolchain)
```

Full building block documentation can be found on GitHub

EQUIVALENT HPC CONTAINER MAKER WORKFLOW

Login to system (e.g., CentOS 7 with Mellanox OFED 3.4)

```
$ module load PrgEnv/GCC+OpenMPI

$ module load cuda/9.0

$ module load gcc

$ module load openmpi/1.10.7
```

Steps to build application

Result: application binary suitable for that particular bare metal system

```
Stage0 += baseimage(image='nvidia/cuda:9.0-
devel-centos7')
Stage0 += mlnx_ofed(version=‘3.4-1.0.0.0’)

Stage0 += gnu()

Stage0 += openmpi(version='1.10.7')
```

Steps to build application

Result: portable application container capable of running on any system

INCLUDED BUILDING BLOCKS

As of version 19.9

CUDA is included via the base image, see https://hub.docker.com/r/nvidia/cuda/

- Compilers
  - GNU, LLVM (clang)
  - PGI
  - Arm, Intel (BYOL)
- HPC libraries
  - Charm++, Kokkos
  - FFTW, MKL, OpenBLAS
  - CGNS, HDF5, NetCDF, PnetCDF
- Miscellaneous
  - Boost
  - CMake
  - Julia, Python
- Communication libraries
  - Mellanox OFED, OFED (upstream)
  - UCX, gdrcopy, KNEM, XPMEM
- MPI
  - OpenMPI
  - MPICH, MVAPICH2, MVAPICH2-GDR
  - Intel MPI
- Visualization
  - Paraview/Catalyst, VisIT/Libsim
- Package management
  - packages (Linux distro aware), or
    - apt_get, yum
  - pip
  - Scientific Filesystem (SCI-F)

HELLO WORLD!

```
Stage0 += baseimage(image='ubuntu:16.04', _as='build')
Stage0 += gnu(fortran=False)

# Use HPCCM primitives to copy the source into the container image
# and then compile the application.
Stage0 += copy(src='sources/hello.c', dest='/var/tmp/hello.c')
Stage0 += shell(commands=['gcc -o /usr/local/bin/hello /var/tmp/hello.c'])

#################

Stage1 += baseimage(image='ubuntu:16.04')

# Include runtime components from build stage
Stage1 += Stage0.runtime()

# Copy the binary from the previous stage
Stage1 += copy(_from='build', src='/usr/local/bin/hello', dest='/usr/local/bin/hello')
```

ONE RECIPE, DOCKERFILE OR SINGULARITY

```
hpccm --recipe hello-world.py –format docker
```

```
FROM ubuntu:16.04 AS build

# GNU compiler
RUN apt-get update -y && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        g++ \
        gcc && \
    rm -rf /var/lib/apt/lists/*

COPY sources/hello.c /var/tmp/hello.c

RUN gcc -o /usr/local/bin/hello /var/tmp/hello.c

FROM ubuntu:16.04

# GNU compiler runtime
RUN apt-get update -y && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        libgomp1 && \
    rm -rf /var/lib/apt/lists/*

COPY --from=build /usr/local/bin/hello /usr/local/bin/hello
```

```
hpccm --recipe hello-world.py –format singularity
```

```
# NOTE: this definition file depends on features only available in
# Singularity 3.2 and later.
BootStrap: docker
From: ubuntu:16.04
Stage: build
%post
    . /.singularity.d/env/10-docker*.sh

# GNU compiler
%post
    apt-get update -y
    DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        g++ \
        gcc
    rm -rf /var/lib/apt/lists/*

%files
    sources/hello.c /var/tmp/hello.c

%post
    cd /
    gcc -o /usr/local/bin/hello /var/tmp/hello.c

BootStrap: docker
From: ubuntu:16.04
%post
    . /.singularity.d/env/10-docker*.sh

# GNU compiler runtime
%post
    apt-get update -y
    DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        libgomp1
    rm -rf /var/lib/apt/lists/*

%files from build
    /usr/local/bin/hello /usr/local/bin/hello
```

BUILDING APPLICATION CONTAINER IMAGES WITH HPCCM

Application recipe

```
# Setup GNU compilers, Mellanox OFED, and OpenMPI
Stage0 += baseimage(image='centos:7')
Stage0 += gnu()
Stage0 += mlnx_ofed(version='3.4-1.0.0.0')
Stage0 += openmpi(cuda=False, version='3.0.0')

# Application build steps below
# Using “MPI Bandwidth” from Lawrence Livermore National Laboratory (LLNL) as an example
# 1. Copy source code into the container
Stage0 += copy(src='mpi_bandwidth.c', dest='/tmp/mpi_bandwidth.c')
# 2. Build the application
Stage0 += shell(commands=['mpicc -o /usr/local/bin/mpi_bandwidth /tmp/mpi_bandwidth.c'])

# CentOS base image
Stage1 += baseimage(image='centos:7')

# Runtime versions of all components from the previous stage
Stage1 += Stage0.runtime()

# MPI Bandwidth
Stage1 += copy(_from='0', src='/usr/local/bin/mpi_bandwidth', dest='/usr/local/bin/mpi_bandwidth')
```

BUILDING APPLICATION CONTAINERS IMAGES WITH HPCCM

Application recipes

CentOS base image

GNU compiler

Mellanox OFED

OpenMPI

MPI Bandwidth

Dockerfile

```
FROM centos:7
```

```
# GNU compiler
RUN yum install -y \
        gcc \
        gcc-c++ \
        gcc-gfortran && \
    rm -rf /var/cache/yum/*
```

```
# Mellanox OFED version 3.4-1.0.0.0
RUN yum install -y \
        libnl \
        libnl3 \
        numactl-libs \
        wget && \
    rm -rf /var/cache/yum/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp http://content.mellanox.com/ofed/MLNX_OFED-3.4-1.0.0.0/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-
x86_64.tgz && \
    mkdir -p /var/tmp && tar -x -f /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64.tgz -C /var/tmp -z && \
    rpm --install /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibverbs-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-
x86_64/RPMS/libibverbs-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibverbs-utils-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libibmad-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibmad-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libibumad-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibumad-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libmlx4-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libmlx5-*.x86_64.rpm && \
    rm -rf /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64.tgz /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64
```

```
# OpenMPI version 3.0.0
RUN yum install -y \
        bzip2 \
        file \
        hwloc \
        make \
        openssh-clients \
        perl \
        tar \
        wget && \
    rm -rf /var/cache/yum/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp https://www.open-mpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.bz2 && \
    mkdir -p /var/tmp && tar -x -f /var/tmp/openmpi-3.0.0.tar.bz2 -C /var/tmp -j && \
    cd /var/tmp/openmpi-3.0.0 &&   ./configure --prefix=/usr/local/openmpi --disable-getpwuid --enable-orterun-prefix-by-default --without-cuda --with-verbs && \
    make -j4 && \
    make -j4 install && \
    rm -rf /var/tmp/openmpi-3.0.0.tar.bz2 /var/tmp/openmpi-3.0.0
ENV LD_LIBRARY_PATH=/usr/local/openmpi/lib:$LD_LIBRARY_PATH \
    PATH=/usr/local/openmpi/bin:$PATH
```

```
COPY mpi_bandwidth.c /tmp/mpi_bandwidth.c


RUN mkdir -p /workspace && \
    mpicc -o /workspace/mpi_bandwidth /tmp/mpi_bandwidth.c
```

Singularity definition file

```
BootStrap: docker
From: centos:7
%post
    . /.singularity.d/env/10-docker.sh
```

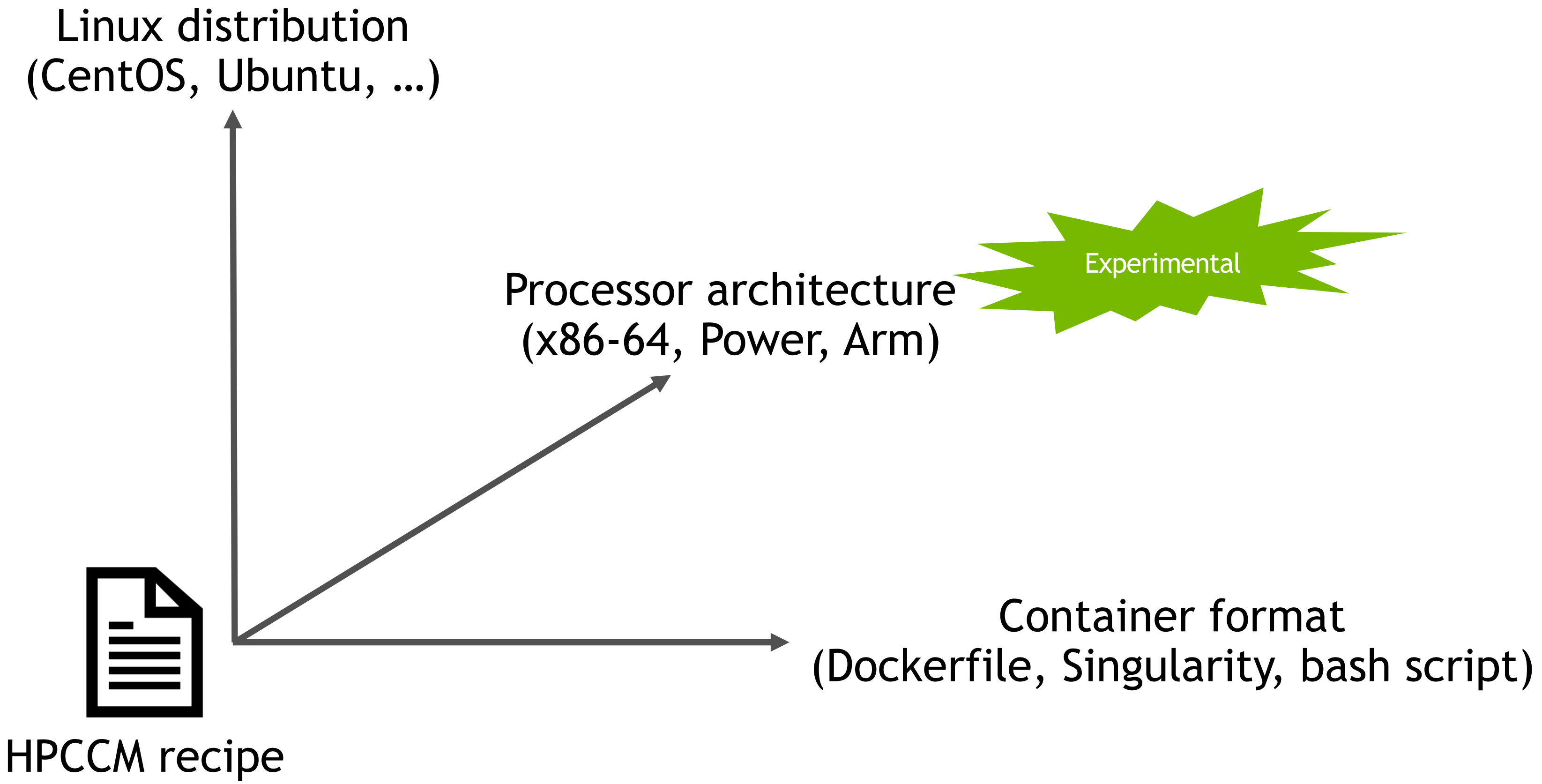
```
# GNU compiler
%post
    yum install -y \
        gcc \
        gcc-c++ \
        gcc-gfortran
    rm -rf /var/cache/yum/*
```

```
# Mellanox OFED version 3.4-1.0.0.0
%post
    yum install -y \
        libnl \
        libnl3 \
        numactl-libs \
        wget
    rm -rf /var/cache/yum/*
%post
    mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp http://content.mellanox.com/ofed/MLNX_OFED-3.4-1.0.0.0/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-
x86_64.tgz
    mkdir -p /var/tmp && tar -x -f /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64.tgz -C /var/tmp -z
    rpm --install /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibverbs-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-
x86_64/RPMS/libibverbs-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibverbs-utils-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libibmad-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibmad-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libibumad-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libibumad-devel-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-
rhel7.2-x86_64/RPMS/libmlx4-*.x86_64.rpm /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64/RPMS/libmlx5-*.x86_64.rpm
    rm -rf /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64.tgz /var/tmp/MLNX_OFED_LINUX-3.4-1.0.0.0-rhel7.2-x86_64
```

```
# OpenMPI version 3.0.0
%post
    yum install -y \
        bzip2 \
        file \
        hwloc \
        make \
        openssh-clients \
        perl \
        tar \
        wget
    rm -rf /var/cache/yum/*
%post
    mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp https://www.open-mpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.bz2
    mkdir -p /var/tmp && tar -x -f /var/tmp/openmpi-3.0.0.tar.bz2 -C /var/tmp -j
    cd /var/tmp/openmpi-3.0.0 &&   ./configure --prefix=/usr/local/openmpi --disable-getpwuid --enable-orterun-prefix-by-default --without-cuda --with-verbs
    make -j4
    make -j4 install
    rm -rf /var/tmp/openmpi-3.0.0.tar.bz2 /var/tmp/openmpi-3.0.0
%environment
    export LD_LIBRARY_PATH=/usr/local/openmpi/lib:$LD_LIBRARY_PATH
    export PATH=/usr/local/openmpi/bin:$PATH
%post
    export LD_LIBRARY_PATH=/usr/local/openmpi/lib:$LD_LIBRARY_PATH
    export PATH=/usr/local/openmpi/bin:$PATH
```

```
%files
    mpi_bandwidth.c /tmp/mpi_bandwidth.c


%post
    mkdir -p /workspace
    mpicc -o /workspace/mpi_bandwidth /tmp/mpi_bandwidth.c
```

ONE RECIPE, MULTIPLE TARGETS



CHARM, 4 WAYS

```
BootStrap: docker
From: centos:7
...

# Charm++ version 6.9.0
%post
    yum install -y \
        ...
        wget
    rm -rf /var/cache/yum/*
%post
    cd /
    mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp
http://charm.cs.illinois.edu/distrib/charm-6.9.0.tar.gz
    mkdir -p /usr/local && tar -x -f /var/tmp/charm-6.9.0.tar.gz -C /usr/local -z
    cd /usr/local/charm-6.9.0 && ./build charm++ multicore-linux-x86_64 --build-shared --with-
production -j4
    rm -rf /var/tmp/charm-6.9.0.tar.gz
%environment
    export CHARMBASE=/usr/local/charm-6.9.0
    ...
```

hpccm --recipe charm.py --format singularity

```
FROM ubuntu:16.04
...

# Charm++ version 6.9.0
RUN apt-get update -y && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        ...
        wget && \
    rm -rf /var/lib/apt/lists/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp
http://charm.cs.illinois.edu/distrib/charm-6.9.0.tar.gz && \
    mkdir -p /usr/local && tar -x -f /var/tmp/charm-6.9.0.tar.gz -C /usr/local -z && \
    cd /usr/local/charm-6.9.0 && ./build charm++ multicore-linux-x86_64 --build-shared --with-
production -j4 && \
    rm -rf /var/tmp/charm-6.9.0.tar.gz
ENV CHARMBASE=/usr/local/charm-6.9.0 \
    ...
```

hpccm --recipe charm.py --userarg image=ubuntu:16.04

```
Stage0 += baseimage(image=USERARG.get('image', 'centos:7'))
Stage0 += gnu()
Stage0 += charm()
```

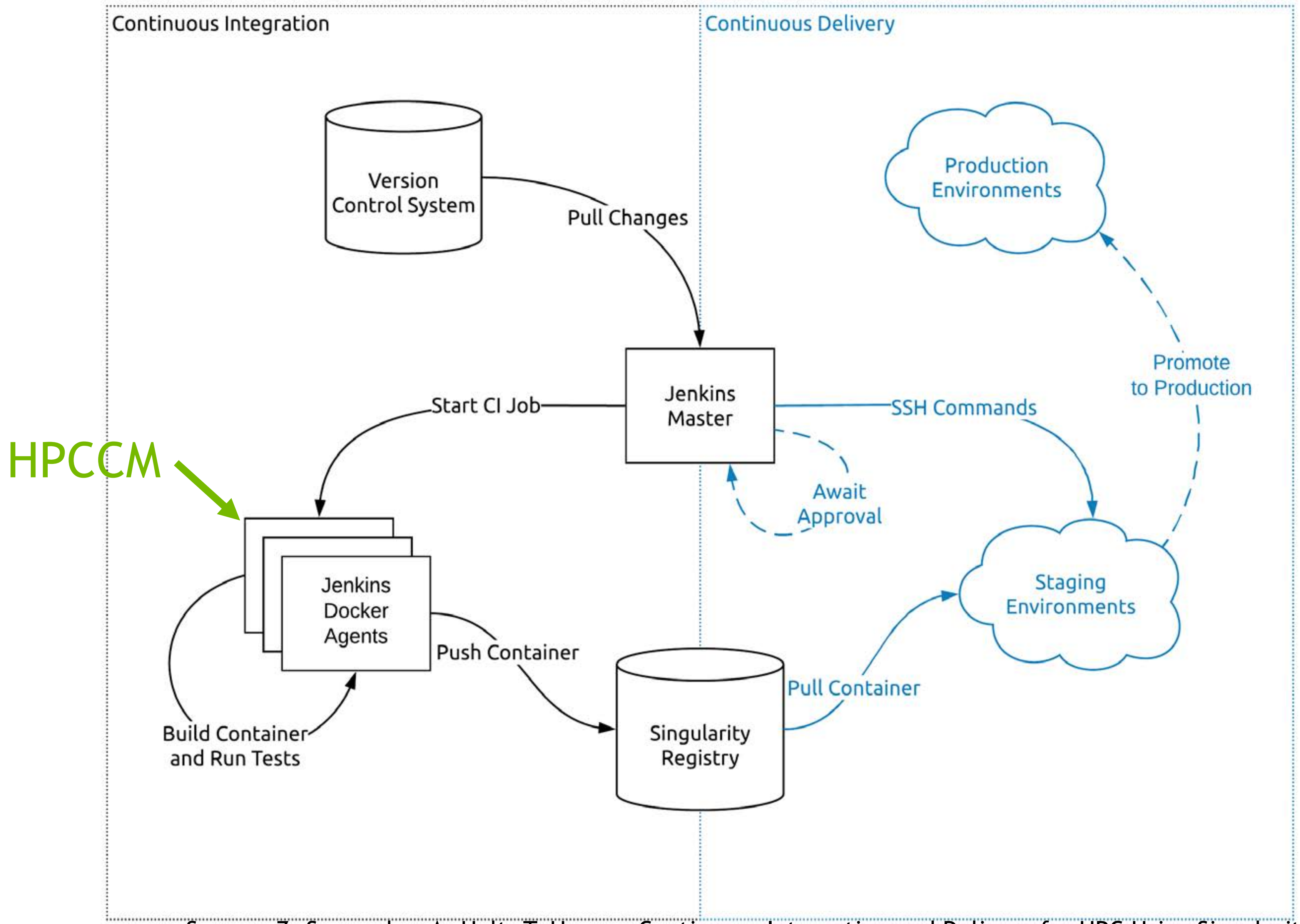
hpccm --recipe charm.py --userarg image=arm64v8/centos:7

```
FROM arm64v8/centos:7
...

# Charm++ version 6.9.0
RUN yum install -y \
        ...
        wget && \
    rm -rf /var/cache/yum/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp
http://charm.cs.illinois.edu/distrib/charm-6.9.0.tar.gz && \
    mkdir -p /usr/local && tar -x -f /var/tmp/charm-6.9.0.tar.gz -C /usr/local -z && \
    cd /usr/local/charm-6.9.0 && ./build charm++ multicore-arm8 --build-shared --with-
production -j4 && \
    rm -rf /var/tmp/charm-6.9.0.tar.gz
ENV CHARMBASE=/usr/local/charm-6.9.0 \
    ...
```

hpccm --recipe charm.py

```
FROM centos:7
...

# Charm++ version 6.9.0
RUN yum install -y \
        ...
        wget && \
    rm -rf /var/cache/yum/*
RUN mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/tmp
http://charm.cs.illinois.edu/distrib/charm-6.9.0.tar.gz && \
    mkdir -p /usr/local && tar -x -f /var/tmp/charm-6.9.0.tar.gz -C /usr/local -z && \
    cd /usr/local/charm-6.9.0 && ./build charm++ multicore-linux-x86_64 --build-shared --with-
production -j4 && \
    rm -rf /var/tmp/charm-6.9.0.tar.gz
ENV CHARMBASE=/usr/local/charm-6.9.0 \
    ...
```

HPCCM + CONTAINERS FOR CI/CD



Source: Z. Sampedro, A. Holt, T. Hauser, Continuous Integration and Delivery for HPC Using Singularity and Jenkins. PEARC'18.

MANAGING COMBINATIONAL EXPLOSION

- As a developer of HPC software, I need to validate my code on a breadth of configurations
- Compiler
  - GNU, LLVM, PGI
- MPI Library
  - MPICH, MVAPICH2, OpenMPI
- Linux Distribution
  - Centos 7, Ubuntu 16.04 & 18.04

$\}$ 27 combinations to test
(not including component versions)

Use a single HPCCM recipe to generate containers for all configurations

REVISIT MPI BANDWIDTH

- Consider MPI Bandwidth as a proxy for a HPC application code
- For each configuration, does it build successfully and does it pass tests?
- One Python script utilizing HPCCM to generate multiple container specifications

```
$ mpibw.py –help
usage: mpibw.py [-h] [--compiler {gnu,llvm,pgi}]
           [--format {docker,singularity}]
           [--linux {centos,ubuntu16,ubuntu18}]
           [--mpi {mpich,mvapich2,openmpi}]

$ mpibw.py --compiler gnu --linux centos --mpi mpich
$ mpibw.py --compiler pgi --linux ubuntu16 --mpi openmpi
```

```
#!/usr/bin/env python

import argparse
import hpccm
from hpccm.building_blocks import *
from hpccm.primitives import baseimage, copy, shell

### Parse command line arguments
parser = argparse.ArgumentParser(description='MPI Bandwidth')
parser.add_argument('--compiler', type=str, default='gnu',
                    choices=['gnu', 'llvm', 'pgi’])
parser.add_argument('--format', type=str, default='docker',
                    choices=['docker', 'singularity’])
parser.add_argument('--linux', type=str, default='centos',
                    choices=['centos', 'ubuntu16', 'ubuntu18'])
parser.add_argument('--mpi', type=str, default='openmpi',
                    choices=['mpich', 'mvapich2', 'openmpi'])
args = parser.parse_args()

### Create container Stage
Stage0 = hpccm.Stage()

### Linux distribution
if args.linux == 'centos':
    Stage0 += baseimage(image='centos:7')
elif args.linux == 'ubuntu16':
    Stage0 += baseimage(image='ubuntu:16.04’)
elif args.linux == 'ubuntu18':
    Stage0 += baseimage(image='ubuntu:18.04')
```

```
### Compiler
if args.compiler == 'gnu':
    compiler = gnu()
elif args.compiler == 'llvm':
    compiler = llvm()
elif args.compiler == 'pgi':
    compiler = pgi(eula=True)
Stage0 += compiler

# Mellanox OFED
Stage0 += mlnx_ofed()

# MPI Library
if args.mpi == 'mpich':
    Stage0 += mpich(toolchain=compiler.toolchain)
elif args.mpi == 'mvapich2':
    Stage0 += mvapich2(cuda=False, toolchain=compiler.toolchain)
elif args.mpi == 'openmpi':
    Stage0 += openmpi(cuda=False, toolchain=compiler.toolchain)

# MPI Bandwidth
Stage0 += copy(src='mpi_bandwidth.c', dest='/var/tmp/mpi_bandwidth.c')
Stage0 += shell(
  commands=['mpicc -o /usr/local/bin/mpi_bandwidth /var/tmp/mpi_bandwidth.c'])

# In lieu of a test suite...
if args.mpi == 'mpich':
  Stage0 += shell(commands=['mpirun -n 2 mpi_bandwidth'])
elif args.mpi == 'mvapich2':
  Stage0 += shell(commands=['MV2_SMP_USE_CMA=0 mpirun -n 2 mpi_bandwidth'])
elif args.mpi == 'openmpi':
  Stage0 += shell(commands=['mpirun -n 2 --allow-run-as-root mpi_bandwidth'])

### Set container specification output format
hpccm.config.set_container_format(args.format)

### Output container specification
print(Stage0)
```

REAL WORLD CASE STUDY

- QUDA is a library for lattice Quantum Chromodynamics (QCD) on GPUs
    - Used by BQCD, Chroma, CPS, and MILC
- CI = GitHub + Jenkins + Singularity + HPCCM
- Using HPCCM to generate Singularity definition files for combinations of:
    - Ubuntu 14, 16, and 18
    - GNU and LLVM compilers (multiple versions)
    - CUDA toolkit versions 7.5 through latest and compute capabilities sm_35 (Kepler) through sm_70 (Volta)
    - CMake 3.8 and 3.12

"HPCCM simplifies my life a lot. We use Singularity containers on our CI server for QUDA to test different compilers, CUDA toolkits, etc. This is much nicer than fiddling with different environment variables and/or modules and provides a clean and stable environment."
Mathias Wagner, QUDA contributor and NVIDIA Compute DevTech engineer

WRAP UP

SAMPLE RECIPES INCLUDED WITH HPC CONTAINER MAKER

HPC Base Recipes:

Ubuntu 16.04
CentOS 7

✖

GNU compilers
PGI compilers

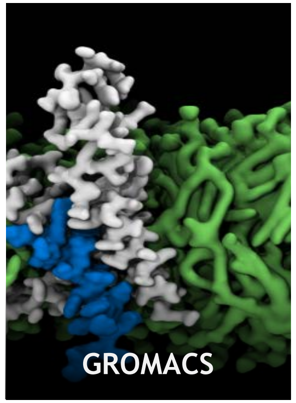
✖

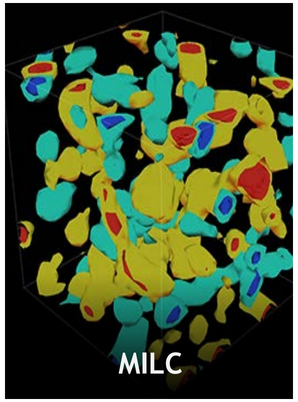
OpenMPI
MVAPICH2

➕

CUDA
FFTW
HDF5
Mellanox OFED
Python

Reference Recipes:





easybuild

Spack

MPI
Bandwidth

BUILDING CONTAINER IMAGES FROM SCRATCH

Recommended workflow:

1. Specify the content of container images with HPC Container Maker
   ```
   $ hpccm --recipe my-recipe.py --format docker
   ```
2. Build container images with Docker
   ```
   $ sudo docker build -t my-image -f Dockerfile .
   ```
3. Convert the Docker images to Singularity images
   ```
   $ singularity build my-image.sif docker-daemon://my-image:latest
   ```
4. Use Singularity to run containers on your HPC system
   ```
   $ scp my-image.sif user@cluster:
   ```

SUMMARY

- HPC Container Maker simplifies creating a container specification file
  - Best practices used by default
  - Building blocks included for many popular HPC components
  - Flexibility and power of Python
  - Supports Docker (and other image builders that use Dockerfiles) and Singularity
- Open source: https://github.com/NVIDIA/hpc-container-maker
- `pip install hpccm`

CALL TO ACTION

Driving Productivity and Faster Time-to-Solutions

Data Scientists and Researchers

Using NGC containers

ngc.nvidia.com

Developers

Distributing software as container images

devblogs.nvidia.com/making-containers-easier-with-hpc-container-maker/

Sysadmins

Making the latest containers automatically available

devblogs.nvidia.com/automating-downloads-ngc-container-replicator/

THANK YOU!