CONTAINERS DEMOCRATIZE HPC

CJ Newburn, Principal Architect for HPC, NVIDIA

GTC'19

S9525 - Containers Democratize HPC

CJ Newburn - Principal Architect for HPC, NVIDIA Compute Software, NVIDIA

NVIDIA offers several containerized applications in HPC, visualization, and deep learning. We have also enabled a broad array of contain-related technologies for GPUs with upstreamed improvements to community projects and with tools that are seeing broad interest and adoption. In addition, NVIDIA is a catalyst for the broader community in enumerating key technical challenges for developers, admins and end users, and is helping to identify gaps and drive them to closure. Our talk describes NVIDIA's new developments and upcoming efforts. We'll detail progress in the most important technical areas, including multi-node containers, security, and scheduling frameworks. We'll also offer highlights of the breadth and depth of interactions across the HPC community that are making the latest, highly-quality HPC applications available to platforms that include GPUs.

Data Center/Cloud Infrastructure HPC and AI

Cloud Services General Government / National Labs Higher Education / Research

All technical, 50 minute talk

Tuesday, Mar 19, 1:00 PM - 01:50 PM

GTC TALKS & RESOURCES

L9128 - High Performance Computing Using Containers WORKSHOP TU 10-12

S9525 - Containers Democratize HPC TU 1-2

S9500 - Latest Deep Learning Framework Container Optimizations W 9-10

SE285481 - NGC User Meetup W 7-9

Connect With the Experts

- NGC W 1-2
- NVIDIA Transfer Learning Toolkit for Industry Specific Solutions TU 1-2 & W 2-3
- DL Developer Tool for Network Optimization W 5-6

OUTLINE

- What containers are good for
- Why container technologies matter to HPC
- What NVIDIA is doing to facilitate HPC containers
- NVIDIA GPU Cloud registry

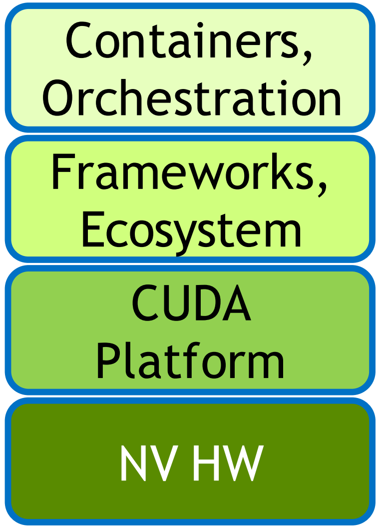
- What's new and what's coming
  - Multi-node containers
  - Community collaboration
  - Interfaces and standardization
  - Easy and robust access to CUDA-aware components

WHAT CONTAINERS ARE GOOD FOR

Ease deployments that enhance performance

- Make everything that's at user level be self-contained
  - → Encapsulate dependences vs. hunting them down
  - → Pre-combine components that are known to work together
  - → Enabling straddling of distros on a common Linux kernel
  - → Isolate and carefully manage resources

- Curate the runtime environment
  - Manage environment variables
  - Compress files
  - Employ special runtimes
  - Cache layers to minimize downloads



WHY CONTAINER TECHNOLOGIES MATTER TO HPC

Good for the community, good for NVIDIA

- Democratize HPC
  - Easier to develop, deploy (admin), and use
- Good for the community, good for NVIDIA
  - **Scale** → HPC; more people enjoy benefits of our scaled systems
  - Easier to deploy → less scary, less complicated → **more GPUs**
  - Easier to get all of the right ingredients → **more performance** from GPUs
  - Easier **composition** → HPC spills into adjacencies

WHAT NVIDIA IS DOING

Earning a return on our investment

- Container images, models, and scripts in NGC registry
  - Working with developers to tune scaled performance
  - Validating containers on NGC and posting them in registry
  - Used by an increasing number of data centers
- Making creation and optimization automated and robust with HPCCM (blog)
  - Used for every new HPC container in NGC, broad external adoption
  - Apply best practices with building blocks, favor our preferred ingredients, small images
- Moving the broader HPC community forward
  - CUDA enabling 3rd-party runtimes and orchestration layers
  - Identifying and addressing technical challenges in the community

NGC: GPU-OPTIMIZED SOFTWARE HUB

Simplifying DL, ML and HPC Workflows

INDUSTRY SOLUTIONS

SMART CITIES

Parking Management | Traffic Analysis

DeepStream SDK

MEDICAL IMAGING

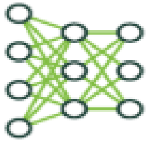
Organ Segmentation

Clara SDK

DEEP LEARNING MODEL SCRIPTS

Classification | Translation | Text to Speech | Recommender | . . .

50+ Containers
DL | ML | HPC



35 Models

Simplify Deployments

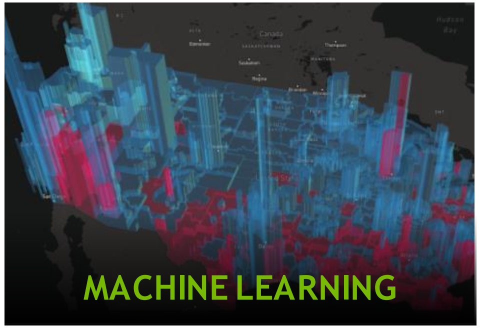
Innovate Faster

Deploy Anywhere

GPU-OPTIMIZED SOFTWARE CONTAINERS

Over 50 Containers on NGC

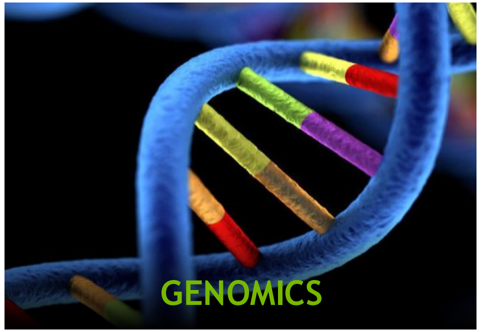
TensorFlow | PyTorch | more



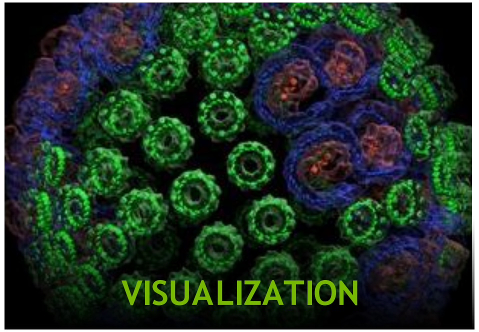
RAPIDS | H2O | more

TensorRT | DeepStream | more

NAMD | GROMACS | more



Parabricks



ParaView | IndeX | more

THE DESTINATION FOR GPU-OPTIMIZED SOFTWARE

HPC	Deep Learning	Machine Learning	Inference	Visualization	Infrastructure
BigDFT	Caffe2	Dotscience	DeepStream	CUDA GL	Kubernetes on NVIDIA GPUs
CANDLE	Chainer	H2O Driverless AI	DeepStream 360d	Index*	
CHROMA*	CT Organ Segmentation	Kinetica	TensorRT	ParaView*	
GAMESS*	CUDA	MapR	TensorRT Inference Server	ParaView Holodeck	
GROMACS	Deep Cognition Studio	MATLAB		ParaView Index*	
HOOMD-blue*	DeepStream 360d	OmniSci (MapD)		ParaView Optix*	
LAMMPS*	DIGITS	RAPIDS		Render server	
Lattice Microbes	Kaldi			VMD*	
Microvolution	Microsoft Cognitive Toolkit				
MILC*	MXNet				
NAMD*	NVCaffe				
Parabricks	PaddlePaddle				
PGI Compilers	PyTorch				
PIConGPU*	TensorFlow*				
QMCPACK*	Theano				
RELION	Torch				
	TLT Stream Analytics IVA				

*Multi-node HPC containers
New since SC18

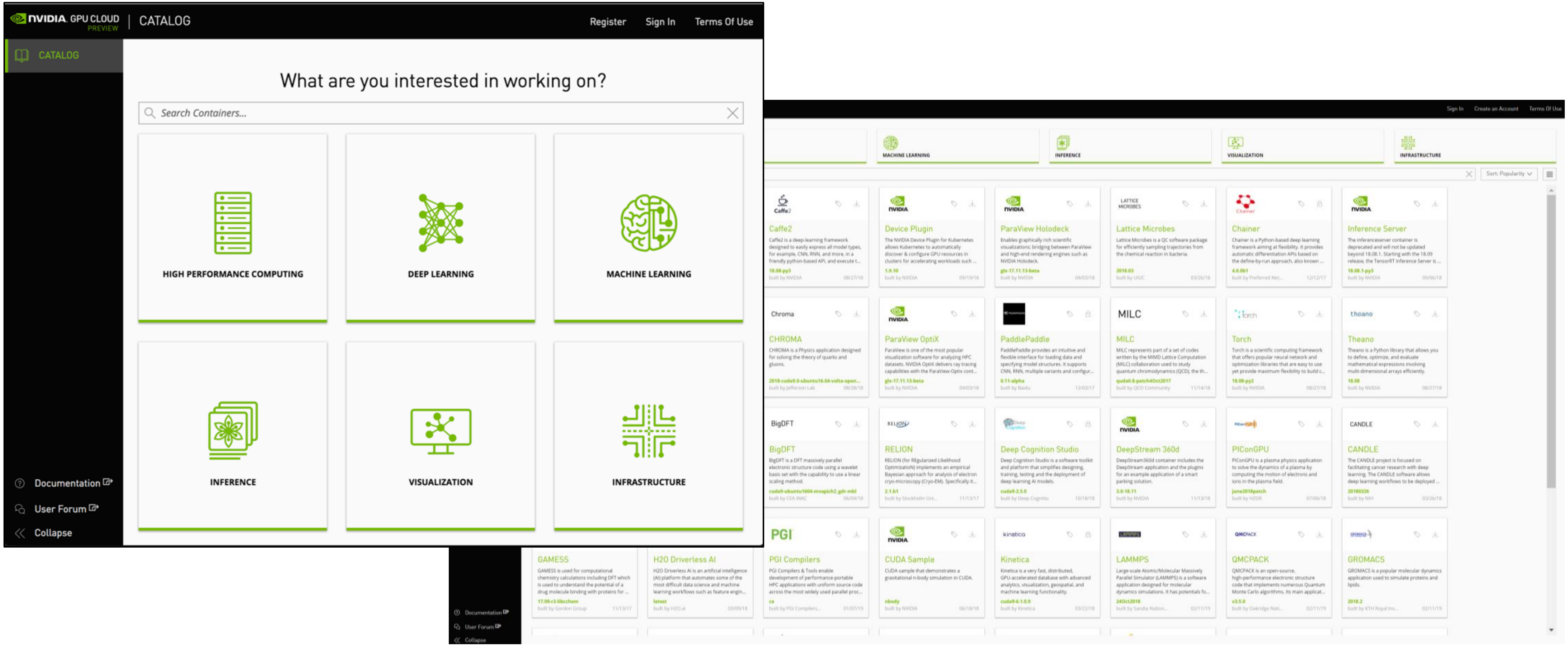
NGC registration not required as of Nov'18

10 containers — October 2017

SOFTWARE ON THE NGC CONTAINER REGISTRY

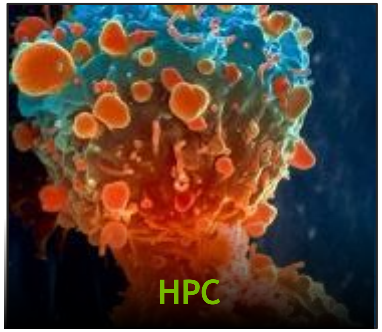
48 containers — ~March 2019

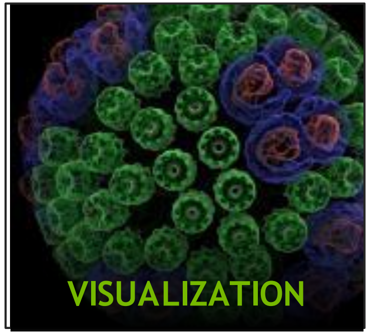
READY TO RUN @ NGC.NVIDIA.COM



A CONSISTENT EXPERIENCE ACROSS COMPUTE PLATFORMS

From Desktop to Data Center To Cloud





docker

Atos
CISCO
CRAY
DELL EMC
aws
Azure
Hewlett Packard Enterprise
Lenovo
NVIDIA
SUPERMICRO
Google Cloud
ORACLE CLOUD PLATFORM

THE UNIVERSITY OF ARIZONA
CLEMSON UNIVERSITY
PSC PITTSBURGH SUPERCOMPUTING CENTER
Stanford University

NGC-READY SYSTEMS

VALIDATED FOR FUNCTIONALITY & PERFORMANCE OF NGC SOFTWARE

T4 & V100-ACCELERATED

Atos

cisco

CRAY

DELL EMC

FUJITSU

Hewlett Packard Enterprise

HUAWEI

inspur

Lenovo

Sugon

SUPERMICRO

MULTI-NODE HPC CONTAINERS

What’s new

Validated support that grows over time

Trend	Validated support
Shared file systems	Mount into container from host
Advanced networks	InfiniBand
GPUs	P100, V100
MPI is common	OpenMPI (3.0.1+ on host)
New (M)OFED and UCX	Dynamically select best versions based on host IB driver
Many targets	Entry points picks GPU arch-optimized binaries, verifies GPU driver, sets up compatibility mode for non-NVIDIA Docker runtimes
Container runtimes	Docker images, trivially convertible to Singularity (v2.5+, blog)
Resource management	SLURM (14.03+), PBS Pro - sample batch scripts
Parallel launch	Slurm srun, host mpirun, container mpirun/charmrun
Reduced size (unoptimized can be 1GB+)	Highly optimized via HPCCM (Container Maker) LAMMPS is 100MB vs. 1.3GB; most under 300MB NAMD was reduced to 200MB from 1.5GB

What’s new

MULTI-NODE CONTAINERS: OPENMPI ON UCX

A preferred layering

- Supports optimized CPU & GPU copy mechanisms when on host
  - CMA, KNEM, XPMEM, gdrcopy (nv_peer_mem)
- OFED libraries used by default
  - Tested for compatibility with MOFED 3.x,4.x host driver versions
- MOFED libraries enabled when versions 3.3-4.5 detected
  - Mellanox “accelerated” verbs transports available when enabled

WHAT IF A CONTAINER IMAGE IS NOT AVAILABLE FROM NGC?

Courtesy of Scott McMillan, NVIDIA solutions architect

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
```

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

Canonical expansion

```
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='3.1.2')                                      # 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

Manual loads

python™

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

```
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: application binary suitable for that particular bare metal system

Result: portable application container capable of running on any system

New since SC18

INCLUDED BUILDING BLOCKS

As of version 19.2

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

- Compilers
  - GNU, LLVM (clang)
  - PGI
  - Intel (BYOL)
- HPC libraries
  - Charm++, Kokkos
  - FFTW, MKL, OpenBLAS
  - CGNS, HDF5, NetCDF, PnetCDF
- Miscellaneous
  - Boost
  - CMake
  - Python

- Communication libraries
  - Mellanox OFED, OFED (upstream)
  - UCX, gdrcopy, KNEM, XPMEM
- MPI
  - OpenMPI
  - MPICH, MVAPICH2, MVAPICH2-GDR
  - Intel MPI
- Visualization
  - Paraview/Catalyst
- Package management
  - packages (Linux distro aware), or
    - apt_get, yum
  - pip

BUILDING APP CONTAINER IMAGES WITH HPCCM

Application recipe

- $ cat mpi-bandwidth.py

```
# 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=['mkdir -p /workspace',
                          'mpicc -o /workspace/mpi_bandwidth /tmp/mpi_bandwidth.c’])
```

- $ hpccm --recipe mpi-bandwidth.py --format …

BUILDING APP CONTAINERS IMAGES WITH HPCCM

Application recipes

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
```

CentOS base image

GNU compiler

Mellanox OFED

OpenMPI

MPI Bandwidth

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
```

MULTISTAGE RECIPES

Only supported by Docker

```
$ cat recipes/examples/multistage.py
# Devel stage base image
Stage0.name = 'devel'
Stage0.baseimage('nvidia/cuda:9.0-devel-ubuntu16.04')

# Install compilers (upstream)
Stage0 += gnu()

# Build FFTW using all default options
Stage0 += fftw()

# Runtime stage base image
Stage1.baseimage('nvidia/cuda:9.0-runtime-ubuntu16.04')

# Install runtime versions of all components from the first stage
Stage1 += Stage0.runtime()
```

RECIPES INCLUDED WITH CONTAINER MAKER

HPC Base Recipes:

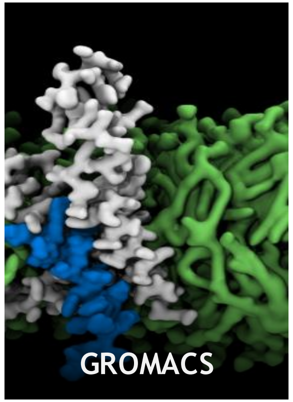
Ubuntu 16.04
CentOS 7

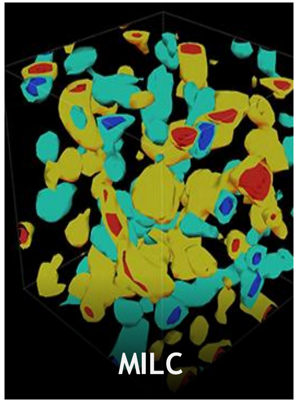
GNU compilers
PGI compilers

OpenMPI
MVAPICH2

CUDA
FFTW
HDF5
Mellanox OFED
Python

Reference Recipes:



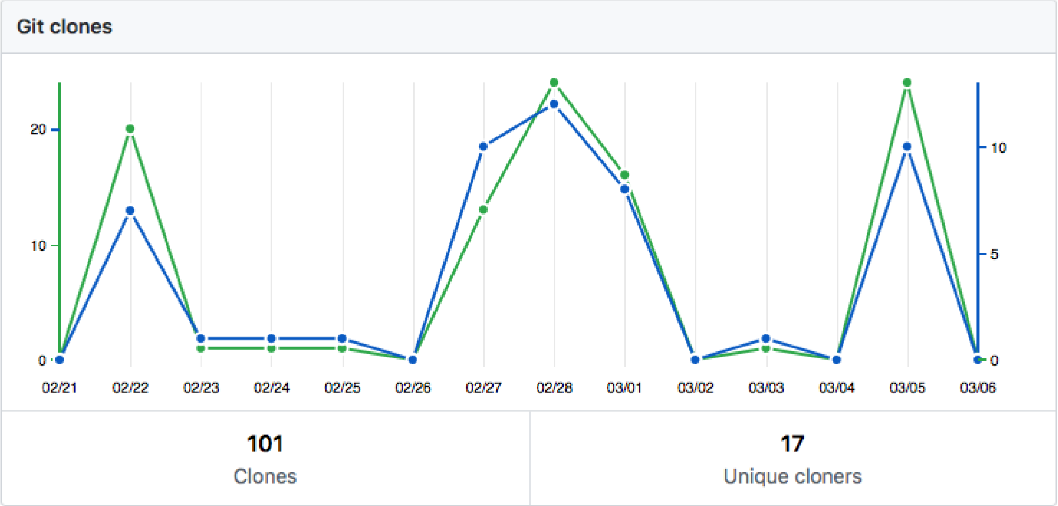


easybuild

Spack

MPI Bandwidth

COMMUNITY INTEREST IN HPCCM



HPCCM downloads over the last 90 days

version	country	system_name	download_count
18.12.0	US	Linux	49
19.1.0	US	Linux	46
19.1.0	RU	Linux	21
18.11.0	US	Linux	14
18.7.0	US	Linux	7
19.1.0	None	Linux	5
18.12.0	RU	Linux	4
19.2.0	None	Linux	4
19.1.0	DE	Linux	3
19.1.0	DE	Darwin	3
Total			156

HPCCM SUMMARY

Making the build process easier, more consistent, more updatable

- 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 frameworks that use Dockerfiles) and Singularity
- Open source: https://github.com/NVIDIA/hpc-container-maker
- `pip install hpccm`

- Refer to this code for NVIDIA's best practices
- HPCCM input recipes are starting to be included in images posted to registry

What’s working

COMMUNITY COLLABORATION

Accelerating technology, adoption by acting as a catalyst

- Created HPC Container Advisory Council
  - 93 participants, 38 institutions that include vendors, labs, academic data centers
- Sample areas of interest
  - What makes HPC usages different than enterprise
  - Container runtimes and OCI, interaction and control by schedulers, resource managers
  - Container orchestration
  - Compatibility, interop: target diversity, driver versions orchestration/container runtimes
  - Container image format, size, layering, encryption, signing
- High Performance Containers Workshop @ ISC19 (CFP)
- HPCCM: rapid extension driven by community requests

CLARIFYING USAGE: ENTERPRISE VS. HPC

Category	Business process / services (Enterprise)	HPC
Work management	Service, process	Job
Resource management	Greedy. Usually simple. Oversubscription likely	Fair. Compute, memory, accelerators, network bandwidth. Oversubscr rare
Batching	ETL/data pipeline	All shapes and sizes
Type of job	Dynamically scaled, async services	Planned schedule. Synchronous MPI. Sensitive to jitter.
Job size, complexity	Broken into small, independent services	May be long running, multi-staged
Limits	Few	Wall time
Coupling	Async services may span multiple nodes	Sync MPI
Job scaling	Auto-scaled based on load. K8s: within a pod (horizontal) so far. Cross pod (vertical) is under development	Preplanned
Multi-user model	Services act on behalf of users	Many simultaneous users running apps; backed by a POSIX id/Unix account
Scheduling	Often no concept of a queue, few jobs until Poseidon. HTCondor brokering. Gang scheduling @ Kube-Batch.	May be long wait times, larger # of jobs handled. Gang scheduling is common.
Storage	HDFS, wider variety, object stores, S3	Shared parallel fs; POSIX + {HDF5, etc.} Often pull down from object store to shared fs.
Reliability	Transactional	Checkpointing
Access patterns	Managed; hosted services	Direct shell, direct resource usage
File systems	Difficult	Integral, vetted
Other support	Huge pages, NUMA, topology-aware routing coming.	Huge pages, NUMA, topology-aware routing pretty standard.
Typical deployments	Cloud, on prem, hybrid	On prem per institution

INCREASING CLARITY AROUND K8S/SCHEDULERS

- K8s over schedulers lik SLURM is growing in interest and popularity
- Both
  - Accept jobs and batches to be scheduled, potentially by both K8s and scheduler
  - Schedule jobs and appropriate level of abstraction
  - Coordinate communication among jobs, at appropriate level of abstraction
- K8s
  - Can recover from denial of availability by nested final authority
  - Supports pluggable scheduling
  - Tends to dynamically schedule fine-grained services
- Scheduler
  - Master arbitrator of resources
  - Tends to preschedule MPI jobs

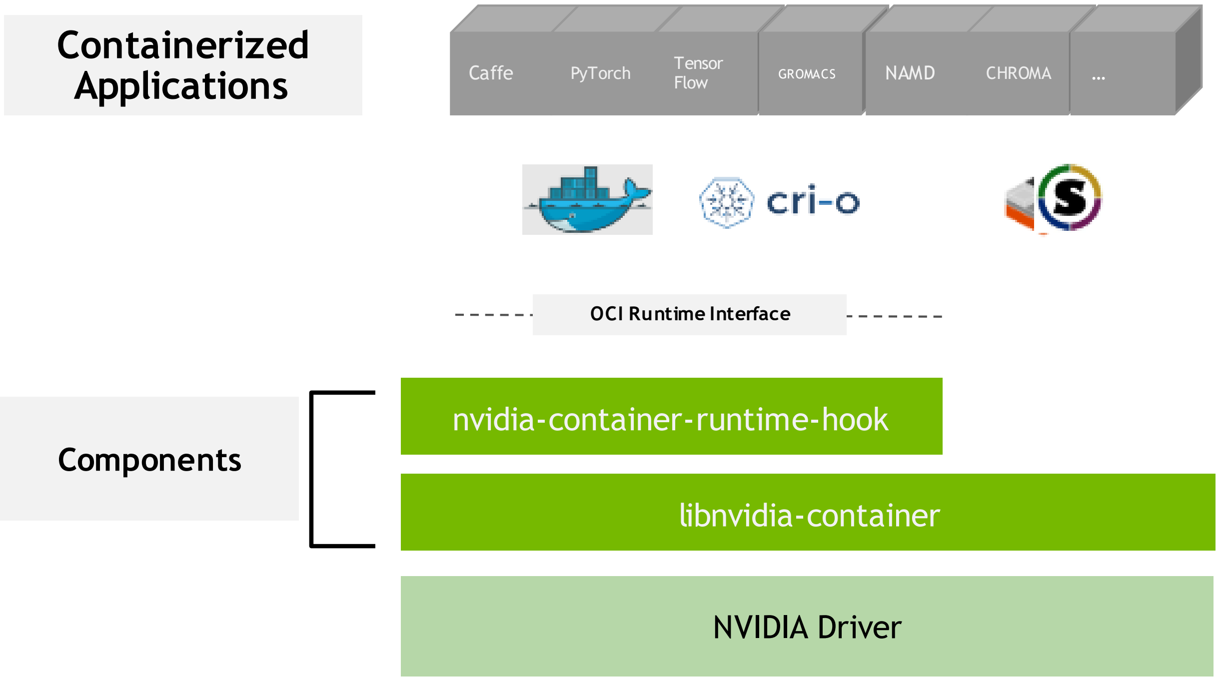
CUDA AWARE: EASY, ROBUST, ACCESSIBLE

Make what’s best for NVIDIA the easiest option

- Identifying SW components for best NVIDIA experience
  - Network, sharing: compatible mofed vs. ofed, nv_peer_mem, CUDA-aware MPI
  - Containers, orchestration: NVIDIA container runtime, Kubernetes
  - Math, deep learning, data science, visualization libs
  - System software: monitoring, health, virtualization
- Examining optimized distribution
  - OSVs, registries
  - Remote access to third party drivers and libs
- Increasing robustness over time
  - Pre-validated combinations

NVIDIA CONTAINER RUNTIME

Enables GPU support in various container runtimes



- Integrates Linux container internals instead of wrapping specific runtimes (e.g. Docker)
- Includes runtime library, headers, CLI tools
- Backward compatibility with NVIDIA-Docker 1.0
- Support new use-cases - HPC, DL, ML, Graphics

PLATFORM SUPPORT

NVIDIA Container Runtime

- Pre-built packages for different OS distributions are available on the NVIDIA repository (Amazon, CentOS, Ubuntu, Debian)
- Updated with Docker releases (most recent 18.09.3)
- LXC includes NVIDIA GPU support (since 3.0.0)
- Singularity support using the --nv option
- Working toward increased integration with Kubernetes
- Read our blog post for more technical details:
    - https://devblogs.nvidia.com/gpu-containers-runtime/

SUMMARY AND CALL TO ACTION

- Container momentum broadens HPC adoption, we’re influencing the experience
- Moving from simpler cases to richer usages
- Making it easier for us all to enable best practices

- Try out container images on NGC with Docker, Singularity, etc.
- Containerize your apps and work with us to get them on NGC
  - Especially interested in HPC + X combinations