CONTAINERS DEMOCRATIZE HPC

CJ Newburn, Principal Architect for HPC, NVIDIA

GTC’19
NVIDIA offers several containerized applications in HPC, visualization, and deep learning. We have also enabled a broad array of container-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.
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

Containers Democratize HPC
GPU-OPTIMIZED SOFTWARE CONTAINERS
Over 50 Containers on NGC

DEEP LEARNING
TensorFlow | PyTorch | more

MACHINE LEARNING
RAPIDS | H2O | more

INFERENC
TensorRT | DeepStream | more

HPC
NAMD | GROMACS | more

GENOMICS
Parabricks

VISUALIZATION
ParaView | IndeX | more
THE DESTINATION FOR GPU-OPTIMIZED SOFTWARE

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

*Multi-node HPC containers
New since SC18

NGC registration not required as of Nov’18

10 containers
SOFTWARE ON THE NGC CONTAINER REGISTRY

48 containers

October 2017 ~ March 2019
READY TO RUN @ NGC.NVIDIA.COM
A CONSISTENT EXPERIENCE ACROSS COMPUTE PLATFORMS
From Desktop to Data Center To Cloud
NGC-READY SYSTEMS

VALIDATED FOR FUNCTIONALITY & PERFORMANCE OF NGC SOFTWARE

T4 & V100-ACCELERATED
## MULTI-NODE HPC CONTAINERS

Validated support that grows over time

<table>
<thead>
<tr>
<th>Trend</th>
<th>Validated support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared file systems</td>
<td>Mount into container from host</td>
</tr>
<tr>
<td>Advanced networks</td>
<td>InfiniBand</td>
</tr>
<tr>
<td>GPUs</td>
<td>P100, V100</td>
</tr>
<tr>
<td>MPI is common</td>
<td>OpenMPI (3.0.1+ on host)</td>
</tr>
<tr>
<td>New (M)OFED and UCX</td>
<td>Dynamically select best versions based on host IB driver</td>
</tr>
<tr>
<td>Many targets</td>
<td>Entry points picks GPU arch-optimized binaries, verifies GPU driver, sets up compatibility mode for non-NVIDIA Docker runtimes</td>
</tr>
<tr>
<td>Container runtimes</td>
<td>Docker images, trivially convertible to Singularity (v2.5+, <a href="#">blog</a>)</td>
</tr>
<tr>
<td>Resource management</td>
<td>SLURM (14.03+), PBS Pro - sample batch scripts</td>
</tr>
<tr>
<td>Parallel launch</td>
<td>Slurm srun, host mpirun, container mpirun/charmrun</td>
</tr>
</tbody>
</table>
| 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 |
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?
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

FROM nvidia/cuda:9.0-devel-centos7

Result: application binary suitable for that particular bare metal system
Containers Democratize HPC

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 /tmp/openmpi.tar.bz2 https://www.openmpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.bz2 | tar -xjf - && \
   cd openmpi-3.0.0 && ./configure --with-cuda=/usr/local/cuda --with-verbs --prefix=/usr/local/openmpi --disable-getpwuid && \
   make -j 4 && make all install && rm -rf openmpi-3.0.0
```

**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.openmpi.org/software/ompi/v1.10/downloads/openmpi-1.10.7.tar.gz && \
   tar -xzvf openmpi-1.10.7.tar.gz && \
   cd openmpi-1.10.7/ && ./configure --with-cuda=/usr/local/cuda --enable-mpi-cxx --prefix=/usr/local/openmpi && \
   make -j 32 &>1 | tee /logs/openmpi_config
   && make install 2>&1 | tee /logs/openmpi_install
   && cd /tmp
```

**E**
```
WORKDIR /tmp
ADD http://www.openmpi.org/software/ompi/v1.10/downloads/openmpi-1.10.7.tar.gz /tmp
RUN tar -xzvf openmpi-1.10.7.tar.gz && \
   cd openmpi-1.10.7/ && ./configure --with-cuda=/usr/local/cuda --enable-mpi-cxx --prefix=/usr/local/openmpi && \
   make -j 32 &>1 | tee /logs/openmpi_make && make install 2>&1
   && rm -rf openmpi-*
```

**F**
```
RUN wget -q -O /tmp/openmpi.tar.gz https://www.openmpi.org/software/ompi/v3.0/downloads/openmpi-3.0.0.tar.gz
RUN tar -xzvf /tmp/openmpi.tar.gz && \
   cd openmpi-3.0.0 && \
   CC=pgcc CXX=pgc++ FC=pgf90 ./configure --with-cuda=/usr/local/cuda --with-verbs --disable-getpwuid && \
   make -j 4 && make all install && rm -rf openmpi-3.0.0
```

Containers Democratize HPC 18 © NVIDIA
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.openmpi.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, 
  configure_opts=['--disable-getpwuid', ...], 
  cuda=True, 
  directory='', 
  infiniband=True, 
  ospackages=['bzip2', 'file', 'hwloc', ...], 
  prefix='/usr/local/openmpi', 
  toolchain=toolchain(), 
  ucx=False, 
  version='3.1.2')

# run “make check”? 
# configure command line options 
# enable CUDA? 
# path to source in build context 
# enable InfiniBand? 
# Linux distribution prerequisites 
# install location 
# compiler to use 
# enable UCX? 
# 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

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
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

New since SC18
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 ...
$ 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()
COMMUNITY INTEREST IN HPCCM

<table>
<thead>
<tr>
<th>version</th>
<th>country</th>
<th>system_name</th>
<th>download_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.12.0</td>
<td>US</td>
<td>Linux</td>
<td>49</td>
</tr>
<tr>
<td>19.1.0</td>
<td>US</td>
<td>Linux</td>
<td>46</td>
</tr>
<tr>
<td>19.1.0</td>
<td>RU</td>
<td>Linux</td>
<td>21</td>
</tr>
<tr>
<td>18.11.0</td>
<td>US</td>
<td>Linux</td>
<td>14</td>
</tr>
<tr>
<td>18.7.0</td>
<td>US</td>
<td>Linux</td>
<td>7</td>
</tr>
<tr>
<td>19.1.0</td>
<td>None</td>
<td>Linux</td>
<td>5</td>
</tr>
<tr>
<td>18.12.0</td>
<td>RU</td>
<td>Linux</td>
<td>4</td>
</tr>
<tr>
<td>19.2.0</td>
<td>None</td>
<td>Linux</td>
<td>4</td>
</tr>
<tr>
<td>19.1.0</td>
<td>DE</td>
<td>Linux</td>
<td>3</td>
</tr>
<tr>
<td>19.1.0</td>
<td>DE</td>
<td>Darwin</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>156</td>
</tr>
</tbody>
</table>
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
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
<table>
<thead>
<tr>
<th>Category</th>
<th>Business process / services (Enterprise)</th>
<th>HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work management</td>
<td>Service, process</td>
<td>Job</td>
</tr>
<tr>
<td>Batching</td>
<td>ETL/data pipeline</td>
<td>All shapes and sizes</td>
</tr>
<tr>
<td>Type of job</td>
<td>Dynamically scaled, async services</td>
<td>Planned schedule. Synchronous MPI. Sensitive to jitter.</td>
</tr>
<tr>
<td>Job size, complexity</td>
<td>Broken into small, independent services</td>
<td>May be long running, multi-staged</td>
</tr>
<tr>
<td>Limits</td>
<td>Few</td>
<td>Wall time</td>
</tr>
<tr>
<td>Coupling</td>
<td>Async services may span multiple nodes</td>
<td>Sync MPI</td>
</tr>
<tr>
<td>Job scaling</td>
<td>Auto-scaled based on load. K8s: within a pod (horizontal) so far. Cross pod (vertical) is under development</td>
<td>Preplanned</td>
</tr>
<tr>
<td>Multi-user model</td>
<td>Services act on behalf of users</td>
<td>Many simultaneous users running apps; backed by a POSIX id/Unix account</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Often no concept of a queue, few jobs until Poseidon. HTCondor brokering. Gang scheduling @ Kube-Batch.</td>
<td>May be long wait times, larger # of jobs handled. Gang scheduling is common.</td>
</tr>
<tr>
<td>Storage</td>
<td>HDFS, wider variety, object stores, S3</td>
<td>Shared parallel fs; POSIX + {HDF5, etc.} Often pull down from object store to shared fs.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Transactional</td>
<td>Checkpointing</td>
</tr>
<tr>
<td>Access patterns</td>
<td>Managed; hosted services</td>
<td>Direct shell, direct resource usage</td>
</tr>
<tr>
<td>File systems</td>
<td>Difficult</td>
<td>Integral, vetted</td>
</tr>
<tr>
<td>Typical deployments</td>
<td>Cloud, on prem, hybrid</td>
<td>On prem per institution</td>
</tr>
</tbody>
</table>
INCREASING CLARITY AROUND K8S/SCHEDULERS

- K8s over schedulers like 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

Components
- nvidia-container-runtime-hook
- libnvidia-container
- NVIDIA Driver

Containerized Applications
- Caffe
- PyTorch
- TensorFlow
- GROMACS
- NAMD
- CHROMA
...
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