HPCW: NVIDIA

CJ Newburn, Princial Architect for HPC

OUTLINE

- NVIDIA overview
- HPCCM – HPC container maker
- NVIDIA developments

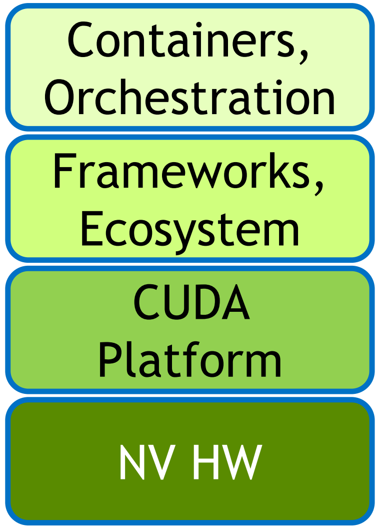
NVIDIA

NVIDIA OVERVIEW

- Containers and HPC
- What NVIDIA is doing
- NVIDIA GPU Cloud

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, scripts, and industry-specific pipelines 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

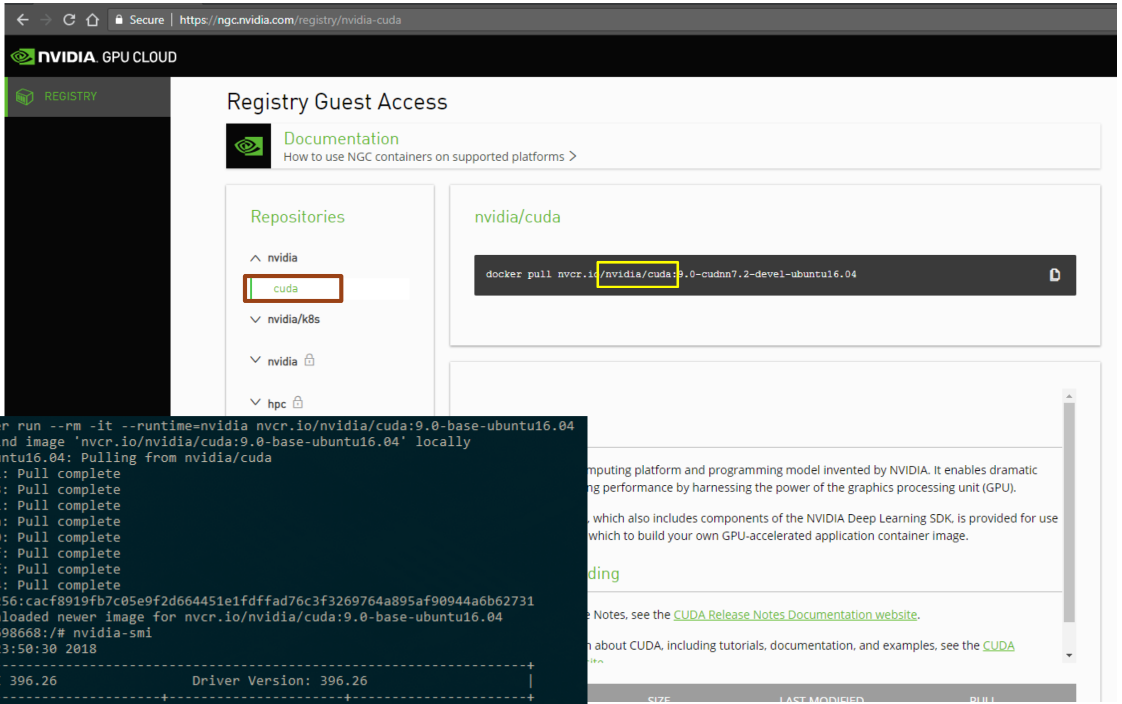
Simplify Deployments

Innovate Faster

Deploy Anywhere

CUDA CONTAINERS ON NVIDIA GPU CLOUD

- CUDA containers available from NGC Registry at **nvcr.io/nvidia/cuda**
- Three different flavors:
- **Base**
  - Contains the minimum components required to run CUDA applications
- **Runtime**
  - Contains *base* + CUDA libraries (e.g. cuBLAS, cuFFT)
- **Devel**
  - Contains *runtime* + CUDA command line developer tools. Some *devel* tags also include cuDNN



Registry Guest Access

Documentation
How to use NGC containers on supported platforms >

nvidia/cuda

```
docker pull nvcr.io/nvidia/cuda:9.0-cudnn7.2-devel-ubuntu16.04
```

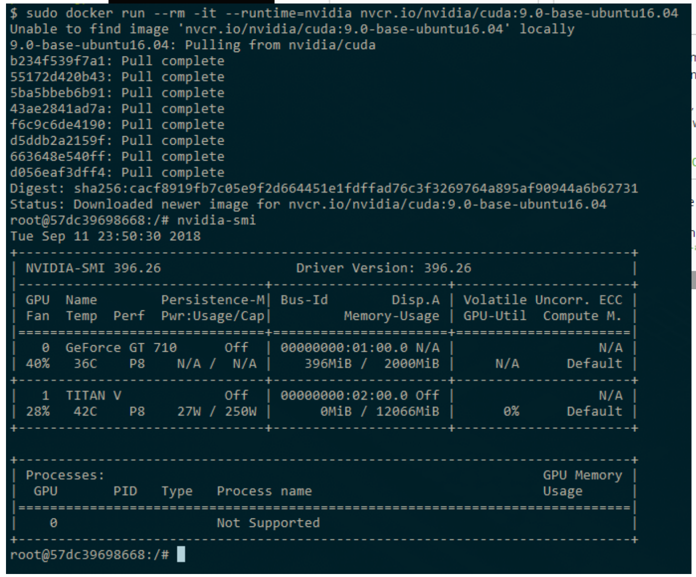
mputing platform and programming model invented by NVIDIA. It enables dramatic
ng performance by harnessing the power of the graphics processing unit (GPU).

, which also includes components of the NVIDIA Deep Learning SDK, is provided for use
which to build your own GPU-accelerated application container image.

ding

e Notes, see the CUDA Release Notes Documentation website.

n about CUDA, including tutorials, documentation, and examples, see the CUDA



```
$ sudo docker run --rm -it --runtime=nvidia nvcr.io/nvidia/cuda:9.0-base-ubuntu16.04
Unable to find image 'nvcr.io/nvidia/cuda:9.0-base-ubuntu16.04' locally
9.0-base-ubuntu16.04: Pulling from nvidia/cuda
b234f539f7a1: Pull complete
55172d420b43: Pull complete
5ba5bbeb6b91: Pull complete
43ae2841ad7a: Pull complete
f6c9c6de4190: Pull complete
d5ddb2a2159f: Pull complete
663648e540ff: Pull complete
d056eaf3dff4: Pull complete
Digest: sha256:cacf8919fb7c05e9f2d664451e1fdffad76c3f3269764a895af90944a6b62731
Status: Downloaded newer image for nvcr.io/nvidia/cuda:9.0-base-ubuntu16.04
root@57dc39698668:/# nvidia-smi
Tue Sep 11 23:50:30 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.26                 Driver Version: 396.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GT 710      Off  | 00000000:01:00.0 N/A |                  N/A |
| 40%   36C    P8    N/A /  N/A |    396MiB /  2000MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+
|   1  TITAN V             Off  | 00000000:02:00.0 Off |                  N/A |
| 28%   42C    P8    27W / 250W |      0MiB / 12066MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0                    Not Supported                                       |
+-----------------------------------------------------------------------------+
root@57dc39698668:/# 
```

HPC CONTAINER MAKER

HPC CONTAINER MAKER

- Motivation
- Problems
- Benefits
- Offerings
- How it works
- Challenges

HPC CONTAINER MAKER - HPCCM

“h-p-see-um”

- Collect and codify best practices
- Make recipe file creation easy, repeatable, modular, qualifiable
- Using this as a reference and a vehicle to drive collaboration
- Container implementation neutral

- Write Python code that calls primitives and building blocks vs. roll your own
    - Leverage latest and greatest building blocks

- *“Without this tool it is much less likely that we would have even started using containers for HPC: …more consistent results… easier to share builds … We are using HPCCM with all of our important applications so it is quickly becoming a critical part of our toolchain.” ~Robert Galetto, PerfLab HPC/DL Manager*

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-3.0.0 &&  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

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

What’s new

MULTI-NODE HPC CONTAINERS

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)
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; most under 300MB

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 version 4.4+ detected
  - Mellanox “accelerated” verbs transports available when enabled

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

NVIDIA DEVELOPMENTS

NVIDIA DEVELOPMENTS

- Goals
- Broad container technology support

GOALS

- Broad container technology support
- Multiple optimized GPU architectures within a single image
- Optimized multi-node cluster support
- Smallest image size possible

CONTAINER TECHNOLOGY SUPPORT

- HPCCM outputs Docker files, Singularity recipe files, shell scripts
- NGC images are in Docker format, convertible to Singularity
  - Docker format enables multi-stage builds more easily
  - Singularity supports pulling from NGC natively
    - $ singularity pull docker://nvcr.io/hpc/lammps:24Oct2018
- A custom entrypoint handles setup in a way that's usable by Docker and Singularity
  - Must assume image FS is read-only vs. writing to /usr/lib vs. bind-mounted dir (mofed)
  - User is whoever starts the container, no sudo, no apt get install
- Documentation provides application specific best practices for both runtimes
  - Examples of running canonical problems/benchmarks provided

MULTI-TARGET SUPPORT

- Single image optimized for Pascal/Volta/Turing when possible
- nvcc can create multi-arch GPU binary targeting all desired architectures
- If build system doesn't support multi-arch compilation, use multiple bins
- entrypoint validates and selects correct binary based on host GPU

MULTINODE: OPENMPI

- Plugin/component-based architecture makes it very flexible
  - Many NGC images use OpenMPI which supports Slurm, PMI2, PMIx, UCX
  - Whereas MPICH seems to require static compile-time config
  - Most decisions made at runtime, ideal for portable containers
- Provides robust GPU-aware MPI support

- Use of .la metadata files inhibits our flexibility via rpath mechanism

MULTINODE: UCX

- Alternate choices
  - IB component is default starting with OpenMPI/4.0
  - Or could use legacy OpenIB byte transfer layer
  - Can compare perf between these without recompilation
- UCX features
  - IB, GDRcopy, CUDA IPC, xpmem, knem, cma optimized transports
  - Picks optimized transport at runtime based upon host capabilities
    - Compile-time decisions based upon detection of MOFED on host
    - Only enables GDRcopy if GDRcopy kernel modules available
    - Requires shipping multiple versions in the container

MULTINODE: INFINIBAND

- Support for Mellanox InfiniBand through MOFED/RDMA-Core
  - User/Kernel driver components not cross version compatible until 4.4+
- Support GPU extensions(nv_peer_mem, gdrcopy)
- Passing in host driver libs can be problematic due to varying transitive dependencies
  - rhel libnl.so <≠> ubuntu libnl-3.so

MULTINODE: INFINIBAND

- Package multiple MOFED/RDMA-Core releases within container
- Selection handled by entrypoint application

- Relocate libibverbs, libdapl, librdmacm, libmlx4, libmlx5
- Read host kernel driver version from /sys/module/mlx5_core/version
- Set LD_LIBRARY_PATH to best matching libibverbs, libdapl, librdmacm libraries
- Set IBV_DRIVERS to point to best matching libmlx4, libmlx5 driver libraries

MULTINODE: PMI

- Glue between resource mgrs, process managers, and processes
- Three common APIs(PMI, PMI2, PMIx)
- Implementations not ABI compatible, even within same API
- PMIx/3.x has robust backwards compatibility and solves many container issues
- PMIx/3.x supported by OpenMPI and Slurm
- PMI2 support useful for legacy Slurm integration

MULTINODE: LAUNCH WITH HOST MPIRUN

- $ mpirun cmd
- $ mpirun singularity run --nv nvidia.simg cmd

- Pros
  - Familiar interface: prefix cmd with Singularity
  - Maintains integration with host resource manager
- Cons
  - Requires compatible host OpenMPI/PMI installation
  - OpenMPI/4.x with PMIx provide good cross version compatibility
  - External mpi may default to using components not in container build

MULTINODE: LAUNCH WITH HOST SRUN

- $ srun cmd
- $ srun --mpi=pmix singularity run --nv nvidia.simg cmd
- $ srun --mpi=pmi2 singularity run --nv nvidia.simg cmd

- Pros
  - Familiar interface: prefix cmd with Singularity and set PMI
  - Maintains integration with host resource manager
- Cons
  - Requires compatible PMI installation; Slurm PMI2 available on most systems

MULTINODE: LAUNCH W/ CONTAINER MPIRUN

- $ mpirun cmd

- HOSTFILE=".hostfile.${SLURM_JOB_ID}"
  for host in $(scontrol show hostnames); do
    echo "${host}" >> ${HOSTFILE}
  done
- OMPI_MCA_plm=rsh
  OMPI_MCA_plm_rsh_args='-o PubkeyAcceptedKeyTypes=+ssh-dss -o ...'
  OMPI_MCA_orte_launch_agent="singularity run --nv nvidia.simg orted"
- singularity run nvidia.simg mpirun --hostfile $HOSTFILE cmd

MULTINODE: LAUNCH W/ CONTAINER MPIRUN

- Pros
  - Works on most systems without external compatibility issues
  - Workload is better contained, better reproducibility
- Cons
  - No integration with host resource manager
  - Exact SSH arguments depend on host specifics

IMAGE SIZE

- Image size important to users and administrators alike
- Heavy use of Docker multi-stage builds to ensure smallest image possible
- Use tools such as dive to audit image size
- LAMMPS container ~100MB, single "baremetal" binary ~70MB