Build

5th High Performance Container Workshop - ISC19

Scope and Introduction

This segment focuses on **BUILD** aspects

We do not talk about distribute or orchestrate.

We might talk about what runtime support is needed. :)

Rootless build with BuildKit

Akihiro Suda (`@_AkihiroSuda_`)
NTT Software Innovation Center

What is BuildKit?

- **Next-generation `docker build` with focus on performance and security**
  - Accurate dependency analysis
  - Concurrent execution of independent instructions
  - Support injecting secret files...

- **Integrated to Docker since v18.06 (`export DOCKER_BUILDKIT=1`)**

- **Non-Docker standalone BuildKit is also available**
  - Works with Podman and CRI-O as well :P

Rootless mode

- **Rootless mode allows building images as a non-root user**
  - Dockerfile `RUN` instructions are executed as a "fake root" in UserNS (So `apt-get/yum` works!)

- **Produces Docker image / OCI image / raw tarball**
  - Compatible with Rootless Docker / Rootless Podman / … whatever

- **Even works inside a container**
  - Good for distributed CI/CD on Kubernetes
  - Works with default `securityContext` configuration
    (but seccomp and AppArmor needs to be disabled for nesting containers)

Rootless BuildKit vs kaniko

- https://github.com/GoogleContainerTools/kaniko

- **Kaniko runs as the root but “unprivileged”**
  - No need to disable seccomp and AppArmor because kaniko doesn’t nest containers on the kaniko container itself

- **Kaniko might be able to mitigate some vuln that Rootless BuildKit cannot mitigate - and vice versa**
  - Rootless BuildKit might be weak against kernel vulns
  - Kaniko might be weak against runc vulns

buildah

Valentin Rothberg <rothberg@redhat.com>
@vlntnrthbrg

What is Buildah?

- Buildah is a tool for **building** container images
- Parts of it's source code is used in podman-build
- Buildah's functionality goes beyond Dockerfiles
- Meant to be used as a low-level coreutiles for building images
- Developed at *github.com/containers/buildah*
    - *github.com/containers/image* for image management
    - *github.com/containers/storage* for local storage (overlay, btrfs, vfs, etc.)

buildah

Buildah ABC

- Supports Dockerfiles
  - Extended to support #include directive to allow decomposition of Dockerfiles
  - $ buildah **b**uild-**u**sing-**d**ockerfile -f Dockerfile .
  - Or shorter via $ buildah **bud** ...
- Can run rootless
- Daemon-less architecture like Podman
- Focus on OCI standards and open development
- Easy to integrate into K8s pipelines
  - Official images available at **quay.io/buildah/stable:latest**

buildah

Does Buildah have a scripting language?

Perhaps Buildahfile?

BASH - Buildah's ultimate scripting language

BASH - Buildah's ultimate scripting language

```
$ newcontainer=$(buildah from scratch)
$ scratchmount=$(buildah mount $newcontainer)
$ # manipulate rootfs of the build-container in $scratchmount
$ buildah unmount $newcontainer
$ buildah commit $newcontainer image:tag
```

buildah

Buildah Resources

- Upstream development and community
  - github.com/containers/buildah
  - #buildah of Freenode
  - buildah@lists.buildah.io
  - buildah.io
- Demos
  - github.com/containers/demos
- Available on *most* Linux distributions
  - Red Hat Enterprise Linux, Fedora
  - openSUSE, Manjaro, Gentoo
  - Archlinux, Ubuntu, Debian (soon)

Sylabs.io
Container Science

Singularity Build - 5 min

Michael Bauer - HPC Container Workshop ISC19

20 June 2019

Build from Docker

```
# Building a Singularity container (SIF) from DockerHub
$ singularity build python.sif docker://python:latest
…
# Running a shell directly from DockerHub
$ singularity shell docker://ubuntu:latest
Singularity ubuntu_latest.sif:~/demo> cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"
Singularity ubuntu_latest.sif:~/demo> exit
$ singularity exec docker://centos:latest cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
```

Singularity Recipe

- Singularity uses a **recipe** file to describe the contents of the container
- Containers created from prebuilt sources at **DockerHub Sylabs SCS**, **SingularityHub**, or your own repository
- Package, Deploy, Visualize with reproducible results

```
BootStrap: yum
OSVersion: 7
MirrorURL:<http...>
Include: yum

# If you want updates then uncomment
#UpdateURL:<http...>

%runscript
    echo "This is what happens when you
run the container..."


%post
    echo "Hello from inside the
container"

yum -y install vim-minimal
```

Singularity Recipe - Multistage

```
Bootstrap: docker
From: golang:1.12.3-alpine3.9
Stage: one

%post
    # prep environment
    export PATH="/go/bin:/usr/local/go/bin:$PATH"
    export HOME="/root"
    cd /root

    cat << EOF > hello.go
package main

import "fmt"
func main() {
    fmt.Printf("Hello World!\n")
}
EOF

    # build
    go build -o hello hello.go
```

```
# Install binary into image without go tools
Bootstrap: library
From: alpine:3.9
Stage: two

#install binary from stage one
%files from one
     /root/hello /bin/hello

%runscript
     hello
```

Hardware Optimized OCI Images via MetaHub Registry Proxy

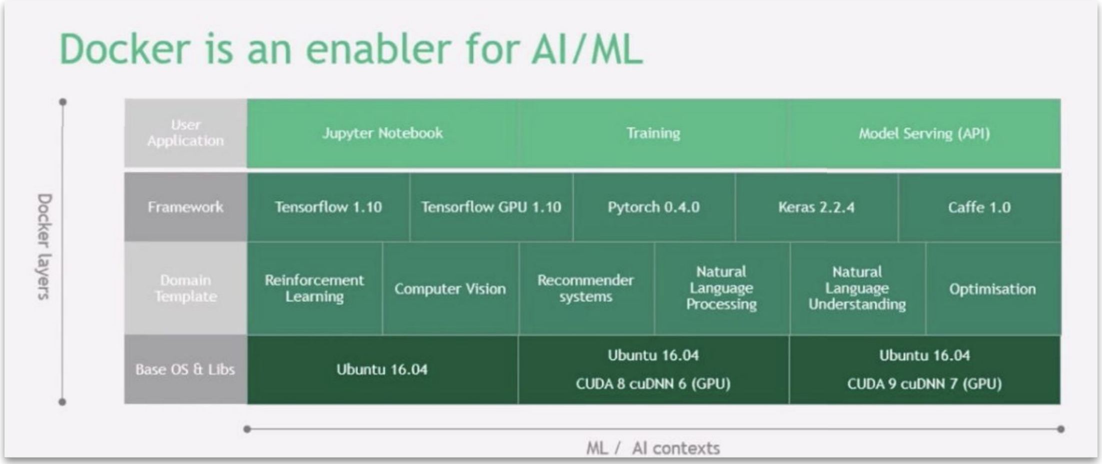
High Performance Container Workshop - ISC19

Christian Kniep, QNIB Solutions
christian@qnib.org

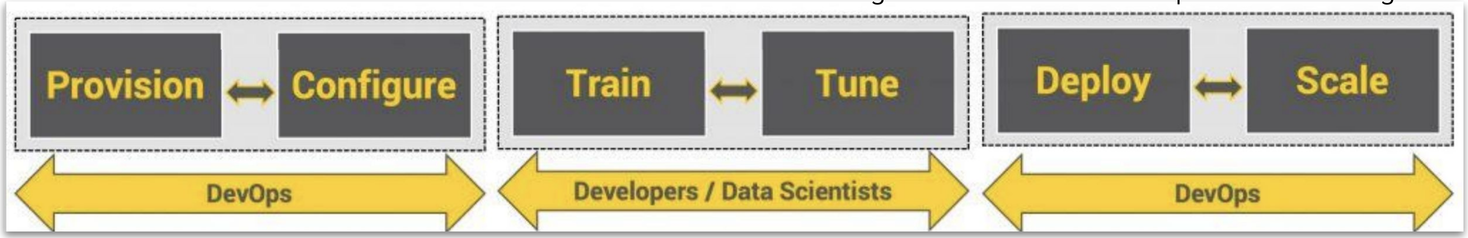
Image Lifecycle

Image Lifecycle needs Composability and Workflow

- Multiple phases in ML livecycle
- Modular Assembly of Software Stacks
- With drivers and growing set of hardware and software, curating these stacks is hard.



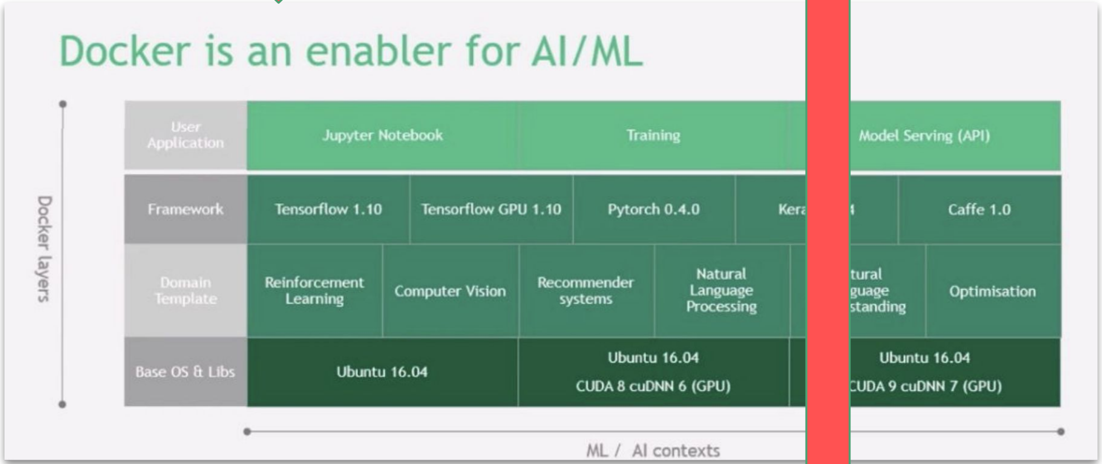
DockerCon EU 2018: Lessons in Using Docker to Close the Loop on Industrializing AI and ML Applications



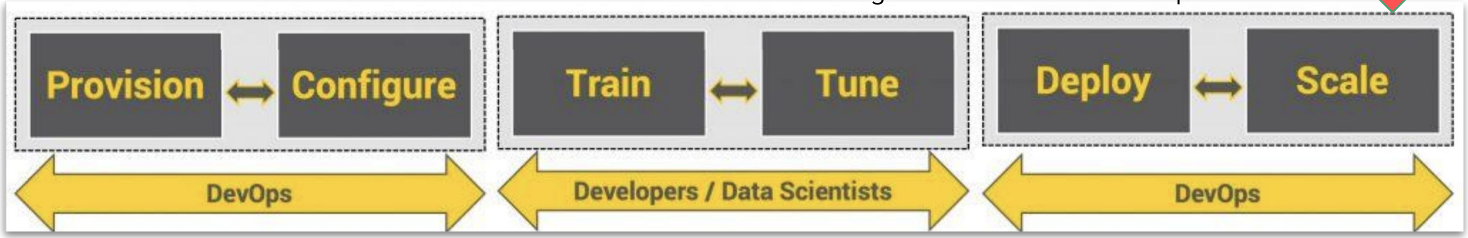
source: https://thenewstack.io/an-introduction-to-the-machine-learning-platform-as-a-service/

Image Lifecycle needs **Composability** and **Workflow**

- Multiple phases in ML livecycle
- Modular Assembly of Software Stacks



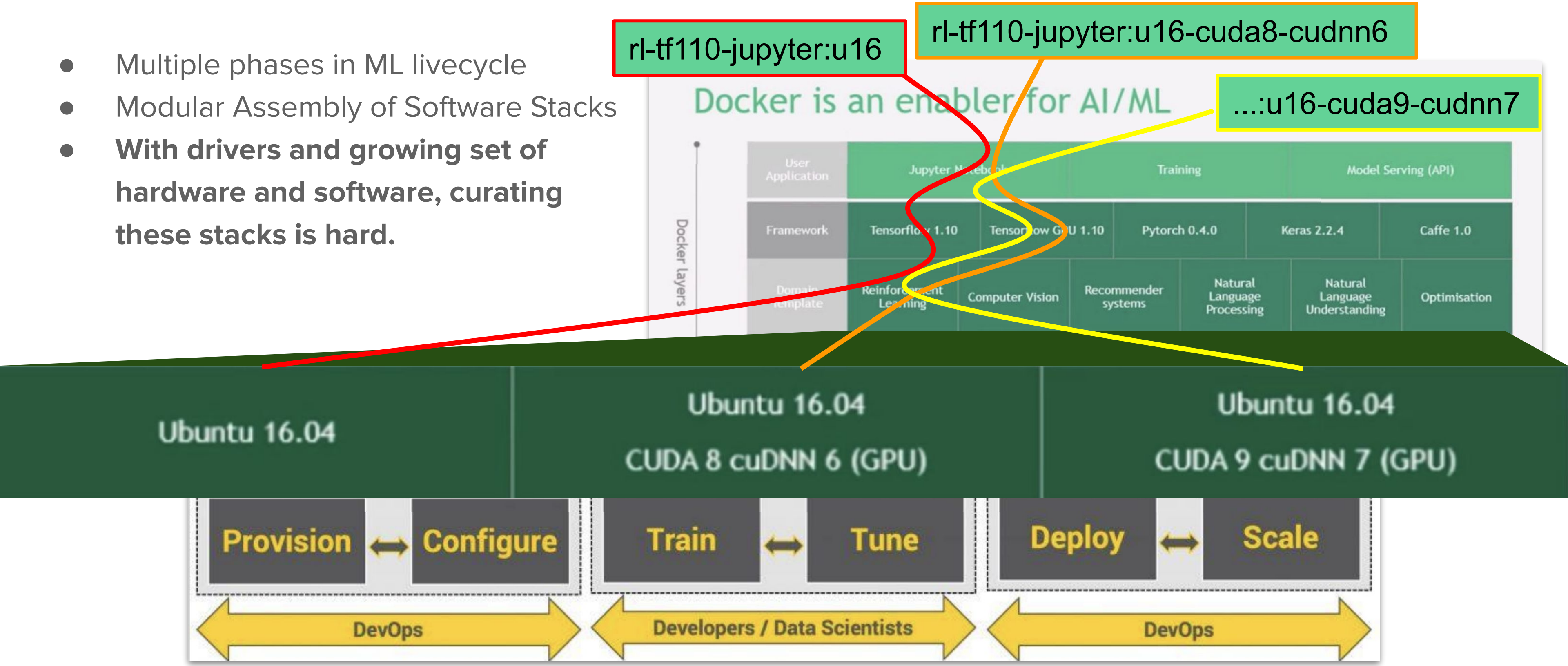
DockerCon EU 2018: Lessons in Using Docker to Close the Loop on Industrializi[illegible]I and ML Applications



source: https://thenewstack.io/an-introduction-to-the-machine-learning-platform-as-a-service/

Image Lifecycle needs Composability and Workflow

- Multiple phases in ML livecycle
- Modular Assembly of Software Stacks
- **With drivers and growing set of hardware and software, curating these stacks is hard.**



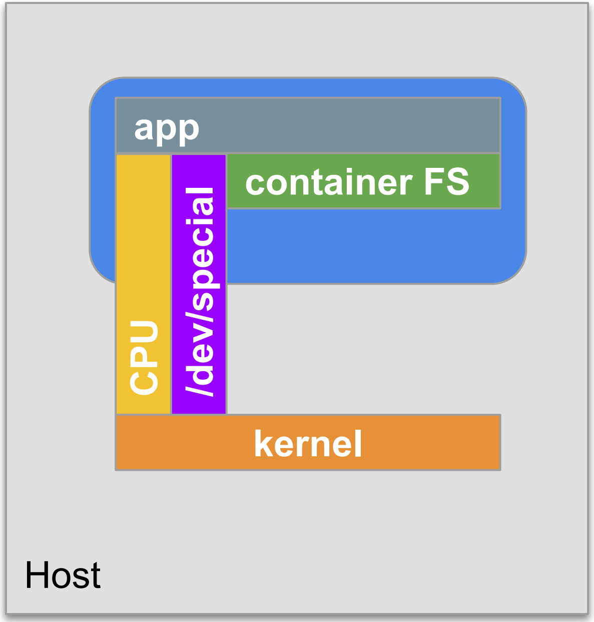
source: https://thenewstack.io/an-introduction-to-the-machine-learning-platform-as-a-service/

Hardware Optimization & Matching

Host-Agnostic vs. Host-Specific

Containers use Kernel-level isolation; being portable among many different systems as a result.
But the portability only guarantees executing, not performance.

Optimization for CPU architectures are only possible if they do not limit the execution.
avx512 is not necessarily available everywhere.



Platform Definition within OCI

The platform object within the “OCI Image Spec” differentiate the architecture by using $GOARCH.

But a container runtime can not pick a container tailored to a microarchitectures.

$GOOS	$GOARCH
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips

- `platform` *object*

  This OPTIONAL property describes the minimum runtime requirements of the image. This property SHOULD be present if its target is platform-specific.

  - `architecture` *string*

    This REQUIRED property specifies the CPU architecture. Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language document for `GOARCH`.

  - `os` *string*

    This REQUIRED property specifies the operating system. Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language document for `GOOS`.

https://github.com/opencontainers/image-spec/blob/master/image-index.md

Platform Definition within OCI

The platform object within the “OCI Image Spec” differentiate the architecture by using $GOARCH.

But a container runtime can not pick a container tailored to a microarchitectures.

$GOOS	$GOARCH
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips

- `platform` *object*

  This OPTIONAL property describes the minimum runtime requirements of the image. This property SHOULD be present if its target is platform-specific.

  - `architecture` *string*

    This REQUIRED property specifies the CPU architecture. Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language document for `GOARCH`.

  - `os` *string*

    This REQUIRED property specifies the operating system. Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language document for `GOOS`.

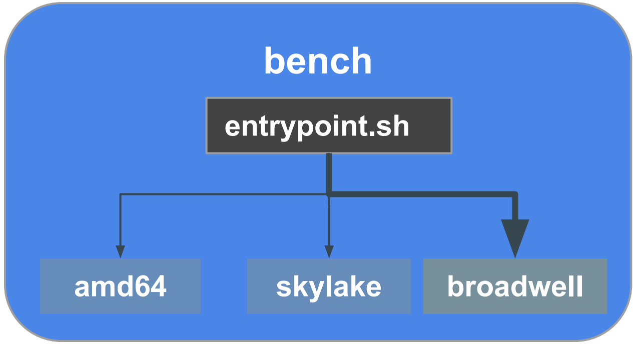
https://github.com/opencontainers/image-spec/blob/master/image-index.md

Year	Micro-architecture	Pipeline stages	max. Clock	Tech process
2018	Cannon Lake	14 (16 with fetch/retire)	3200 MHz	10 nm
2018	Whiskey Lake	14 (16 with fetch/retire)	4600 MHz	14 nm
2018	Amber Lake	14 (16 with fetch/retire)	4200 MHz	14 nm
2017	Coffee Lake	14 (16 with fetch/retire)	5000 MHz	14 nm
2017	*Goldmont Plus*	? 20 unified with branch prediction ?	2800 MHz	14 nm
2016	*Goldmont*	20 unified with branch prediction	2600 MHz	14 nm
2016	Kaby Lake	14 (16 with fetch/retire)	4500 MHz	14 nm
2015	*Airmont*	14-17 (16-19 with fetch/retire)	2640 MHz	14 nm
2015	Skylake	14 (16 with fetch/retire)	4200 MHz	14 nm
2014	Broadwell	14 (16 with fetch/retire)	3700 MHz	14 nm

Current State of Affairs

For the container to be portable, it needs to fall back to the lowest common denominator of the target systems.
Worst case: amd64 (x86_64).

Some try to mitigate the problem by using an entrypoint, which chooses the appropriate binary at runtime.



But the image still needs to be changed every time a new hardware configuration is added.

Current State of Affairs [cont]

Images optimized for different CPU µ-arch are picked by name.

As described in the previous slides.


```
$ docker run -ti --rm qnib/bench:cpu-broadwell
>> This container is optimized for: cpu:broadwell
```

```
$ docker run -ti --rm qnib/bench:cpu-skylake
>> This container is optimized for: cpu:skylake
```

```
$ docker run -ti --rm qnib/bench:generic
>> This container is not optimized for a specific microarchitecture
```

```
$ docker run -ti --rm qnib/bench:cpu_broadwell-nvcap_52
>> This container is optimized for: cpu:broadwell,nvcap:5.2
```

Thank you!

HPCW: NVIDIA

CJ Newburn, Princial Architect for HPC

HPC CONTAINER MAKER

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

MOTIVATIONAL STORIES FOR CONTAINERS

War stories from the trenches

Developers:

- Lack of a reference design
  - Many variants, some better than others
- Encapsulating pipelines reduces complexity
- Reproducibility

Admins:

- Hard to configure and install HPC apps
- Better startup times with fewer libs loaded from bottlenecked metadata servers
- Will a given app run on a new platform?

Users:

- App updates get delayed
- Experimental/simulation hybrid molecular modeling as a service

OPENMPI DOCKERFILE VARIANTS

Developers

Real examples: lots of ways, some better than others

```
RUN OPENMPI_VERSION=3.0.0 && \
    wget -q -O - https://www.open-
mpi.org/software/ompi/v3.0/downloads
${OPENMPI_VERSION}.tar.gz | tar -xzf
    cd openmpi-${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_
    echo "/usr/local/mpi/lib" >> /etc
ldconfig
ENV PATH /usr/local/mpi/bin:$PATH
```

Enable many versions with parameters to common interface

Parameters vary

Control environment

```
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
```

Functional, simpler, but not CUDA or IB aware

```
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"
```

Different compilers

```
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-*
```

Bad layering

```
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-*
```

```
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 - 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

Result: application binary suitable for that particular bare metal system

python™

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

Stage0 += gnu()

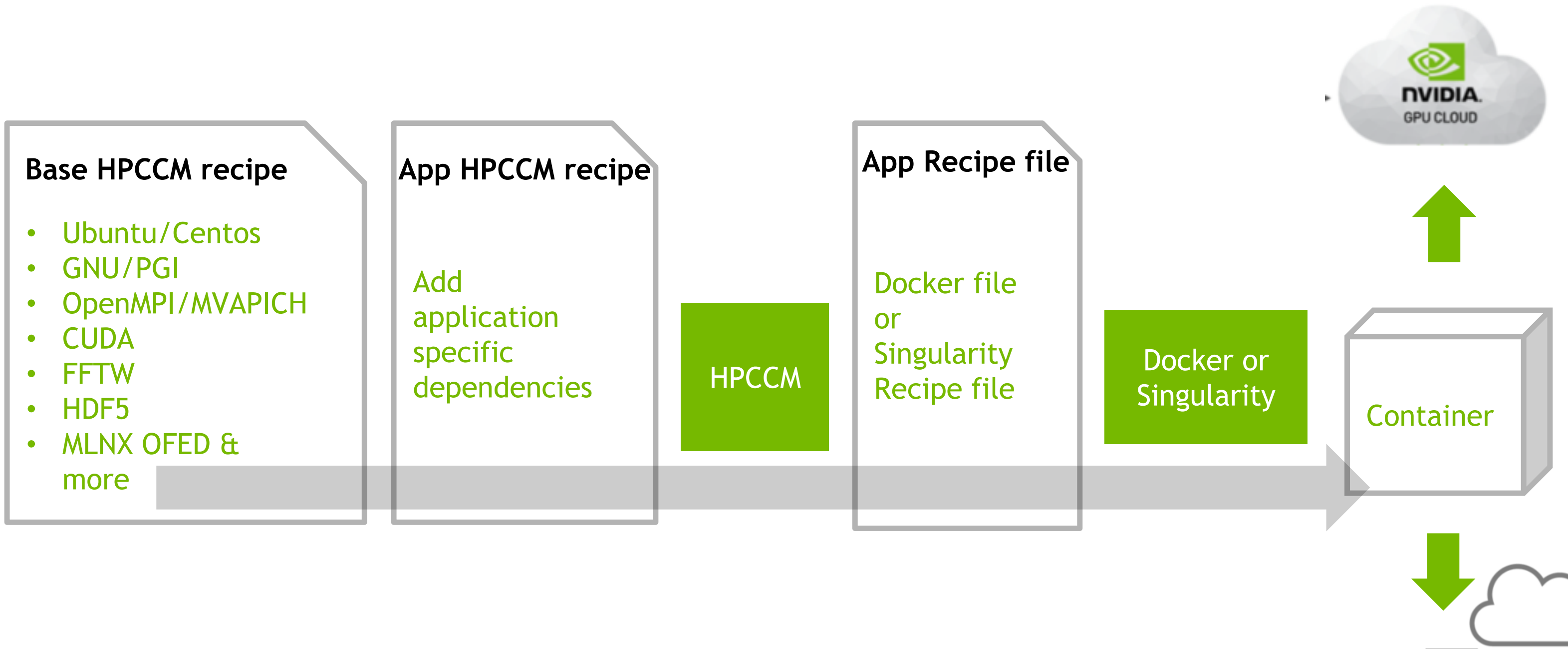
Stage0 += openmpi(version='1.10.7')

Steps to build application

Result: portable application container capable of running on any system

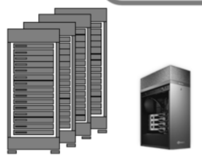
HPC CONTAINER MAKER

SIMPLEST WAY TO BUILD CONTAINERS



1. Follow best practices
   - Layered builds
   - Optimized container size
   - Configuration, deployment

2. Ready-to-use building blocks
   - Simplify efforts
   - Faster builds
   - Higher reliability
   - Optimized performance



https://github.com/NVIDIA/hpc-container-maker
https://devblogs.nvidia.com/making-containers-easier-with-hpc-container-maker/
PyPi: pip install hpccm

RECIPES INCLUDED WITH CONTAINER MAKER

HPC Base Recipes:

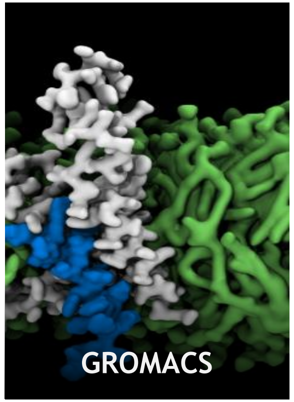
Ubuntu 16.04
CentOS 7

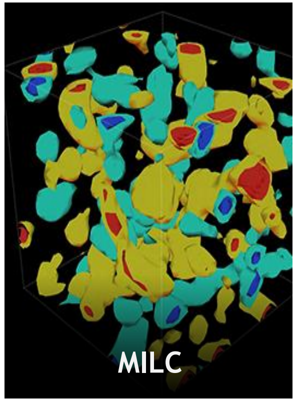
GNU compilers
PGI compilers

OpenMPI 3.0.0
MVAPICH2 2.3b

CUDA 9.0
FFTW 3.3.7
HDF5 1.10.1
NetCDF 4.6.1
Mellanox OFED 3.4-1.0.0.0
Python 2 and 3
Intel MKL
apt_get, yum
cmake…

Reference Recipes:





… or create your own …

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 BLOCKS: WHATIF FOR SIERRA…

As of ?

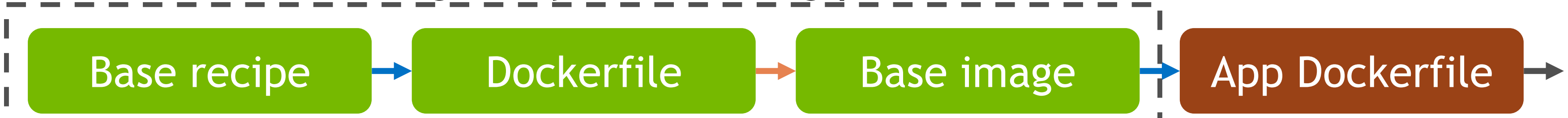
CUDA for POWER is included via the base image, see
https://hub.docker.com/r/nvidia/cuda/

- Compilers
  - GNU
  - PGI (BYOL)
  - XL (BYOL)
- HPC libraries
  - ESSL, PESSL
  - FFTW, OpenBLAS
  - HDF5, NetCDF, PnetCDF
- Miscellaneous
  - Python
  - Boost
  - CMake
- InfiniBand
  - Mellanox OFED
  - OFED (upstream)
- MPI
  - SpectrumMPI
  - OpenMPI
- Package management
  - packages (Linux distro aware), or
    - apt_get
    - Yum
  - Easybuild
  - SPACK

BUILDING AN HPC APPLICATION IMAGE

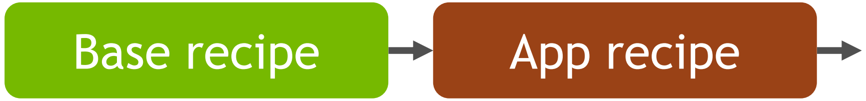
Analogous workflows for Singularity

1. Use the HPC base image as your starting point



2. Generate a Dockerfile from the HPC base recipe Dockerfile and manually edit it to add the steps to build your application

3. Copy the HPC base recipe file and add your application build steps to the recipe



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

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

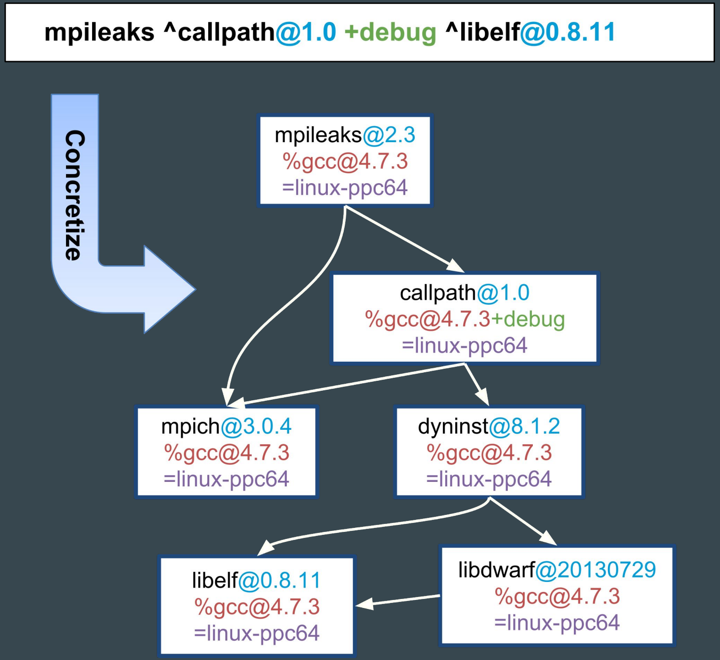
- HPC Container Maker simplifies creating a container specification file
  - Best practices used by default
  - Building blocks included for many popular HPC components
  - Flexibility and power of Python
  - Supports Docker (and other frameworks that use Dockerfiles) and Singularity
- Open source: https://github.com/NVIDIA/hpc-container-maker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

Build [Optimized] Images with Spack

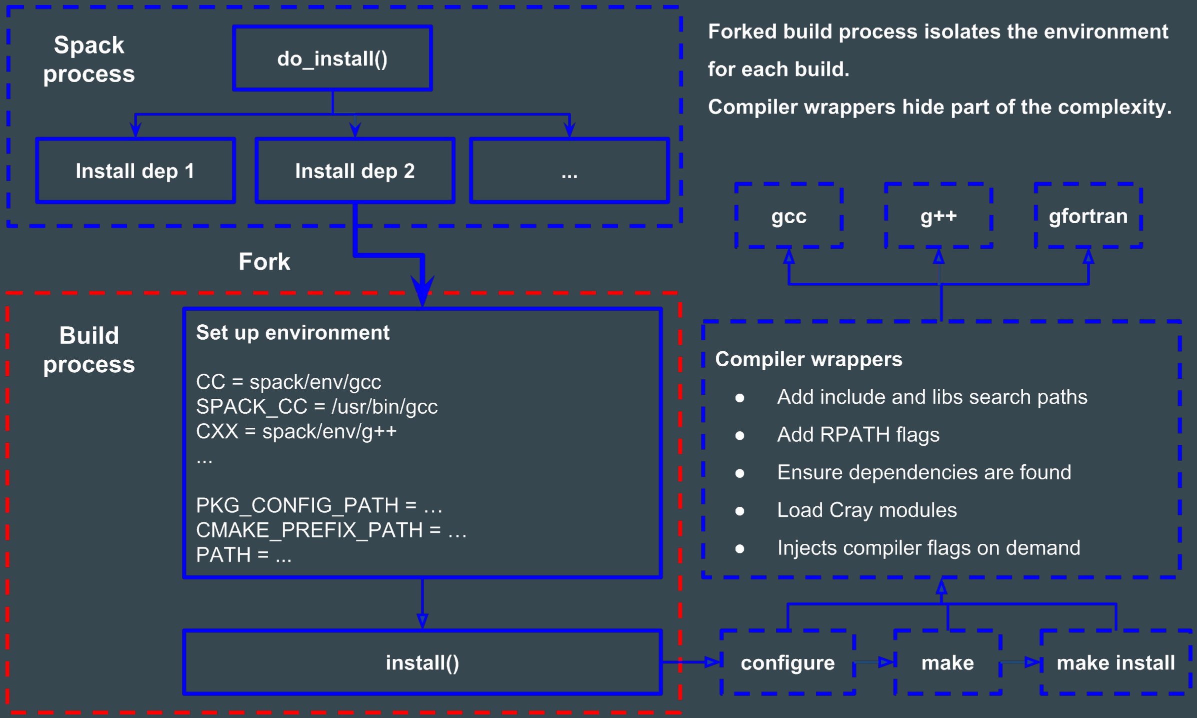
Massimiliano Culpo, EPFL

Spack manages complex application DAGs




```
# Install a specific version by appending @
$ spack install hdf5@1.10.1

# Specify a compiler (and optional version), with %
$ spack install hdf5@1.10.1 %gcc@4.7.3

# Add special boolean compile-time options with +
$ spack install hdf5@1.10.1 +szip

# Add custom compiler flags
$ spack install hdf5@1.10.1 cppflags="-O3
-floop-block"

# Cross-compile on a Cray or Blue Gene/Q
$ spack install hdf5@1.10.1 target=backend

# Recurse to dependency with ^
$ spack install hdf5 ^zlib@1.2.8
```

Compiler flags can already be injected programmatically



Target support will be extended in the near future

```
# Compile for skylake. This will automatically
# inject flags like -march=skylake -mtune=skylake
$ spack install hdf5@1.10.1 target=skylake
```

Improvements on target support in the near future include:

- Better auto-detection of host micro-architecture
- Mapping micro-architecture and compilers to a proper set of optimization flags

The long term goal is to support software cross compilation, even across architectures.

https://github.com/spack/spack/pull/3206

Entire environments can be described in YAML format

```yaml
# This is a Spack Environment file.
#
# It describes a set of packages to
# be installed, along with configuration
# settings.
spack:
  # add package specs to the `specs` list
  specs:
  -gromacs target=skylake
  -cp2k    target=skylake
```

Spack environments are built on the *manifest + lockfile* concept pioneered by Bundler.

We have experimental Spack containers: no need to replicate the set-up from base images to have Spack.

```
Bootstrap: docker
From: ubuntu:18.04

%files
$PWD/spack.yaml <workdir>/spack.yaml
...

%post
# Install basic system dependencies
apt-get update && ...
# Install spack + set the mirror
mkdir -p $SPACK_ROOT
curl -s -L <spack-url> \
    | tar xzC $SPACK_ROOT --strip 1

cd <workdir>
/home/spack/bin/spack install

%runscript
...
```

Features and improvements that are being discussed

- Closer integration with different container technology:
  - https://github.com/spack/spack/pull/7204 (docker)
  - https://github.com/spack/spack/pull/10952 (hpc-container-maker)
  - https://github.com/spack/spack/pull/11367 (singularity)
- Strip binaries to reduce container size
- Support for multi-stage builds (build deps pruned from the final image)
- Optionally unroll the DAG to make use of the container cache where available
- …