Making Containers Easier with HPC Container Maker

Scott McMillan
NVIDIA Corporation
Champaign IL, USA
smcmillan@nvidia.com

***Abstract*—Containers make it possible for scientists and engineers to easily use portable and reproducible software environments. However, if the desired application is not already available from a container registry, generating a custom container image from scratch can be challenging for users accustomed to a bare metal software environment. HPC Container Maker (HPCCM) is an open source project to address these challenges. HPCCM provides a high level, configurable Python recipe format for deploying HPC components into container images according to best practices. HPCCM recipes are more portable and powerful than native container specification formats, easier to create, and produce smaller, well-constructed container images.**

***Index Terms*—containers, singularity, docker, high-performance computing**

I. Introduction

Containers simplify software deployments in HPC data centers by wrapping applications into isolated virtual environments. By including all application dependencies like binaries and libraries, application containers run seamlessly in any data center environment. Consequently, containers can be easily shared without concern to the underlying software environment on the deployed system, aside from the container runtime itself. A container can even run an entirely different Linux distribution compared to the host.

Containers also provide a reproducible software environment. With the entire user space software environment encapsulated, an important source of variation in results from system to system is eliminated. Checksums of the container image can be compared to verify that an identical container image is being used.

Unlike virtual machines, which provide similar software stack isolation benefits, the performance impact of containers is negligible relative to bare metal [1], [2]. Since containers share the host OS kernel, startup time is short compared to a virtual machine. Docker [3] has been instrumental in popularizing containers, and more recently HPC targeted container runtimes such as Singularity [4], Shifter [5], and Charliecloud [6] have emerged.

Container registries such as Docker Hub, Singularity Hub, and the NVIDIA GPU Cloud host pre-built container images that dramatically improve ease of application deployment while delivering optimized performance. However, if the desired application is not available on a container registry, building HPC container images from scratch may be daunting. Parts of the software environment traditionally provided by the HPC data center must be redeployed inside the container. This paper describes HPC Container Maker (HPCCM), an open source project to address the challenges of creating HPC application containers [7].

A. Related Work

Environment modules [8] are typically used on HPC systems to manage the software development and runtime environment. Whether building an application from source or using a pre-compiled binary, the user loads the set of environment modules appropriate for the application, e.g., a compiler, MPI library, math libraries, and other supporting software components. The software packages have been pre-installed on the system by the system administrator. A typical HPC system may have hundreds of inter-dependent environment modules. For example, multiple instances of a math library may be installed for every combination of compiler version and MPI library version. An environment modules setup is also specific to each HPC system, so application run scripts may not be portable, and in some cases application binaries may not be portable either.

EasyBuild [9] and Spack [10] are commonly used frameworks to manage HPC software components. EasyBuild and Spack automate all the steps necessary to build an application, including the download and installation of any required software components such as a compiler, MPI library, math libraries, or other supporting software components. The software component versions and build options are precisely specified in the recipe, so the "same" binary can be easily generated on any system. However, due to the runtime dependencies of the binary, the binary itself may not portable to other systems. The application is typically regenerated on each system.

Container images can be generated with EasyBuild and Spack. However, these tools were designed for the bare metal use case so the containerization process and results are suboptimal compared to a "native" container workflow. For instance, EasyBuild and Spack wrap an entire complex application build process into a single step. While this is ideal for the bare metal use case, it is a poor fit for containers. The application build may take hours to complete, and if there are any errors during the container build the whole process must be started again from scratch. If the build process were split into simpler, more granular steps, the successful steps could be cached using container layers [11], speeding up the overall

build time. In addition, the resulting container image sizes can be much larger than necessary since container best practices such as multi-stage builds cannot be used. Unfortunately, these bare metal tools cannot take advantage of these container best practices without a fundamental re-architecture.

B. Container Images

Container images bundle the application together with all necessary user space dependencies for a self-contained virtual environment. A container image is fully portable and can be run on nearly any system with the corresponding container runtime installed (Docker, Singularity, etc.)

To build a container image, container frameworks rely on an input specification file defining the contents of the corresponding container image. The specification file contains the precise set of steps to execute when building a container image. For a given component, the steps may include download of the source, configuration and building, installation, and finally cleanup. The input specification file is a Dockerfile [12] for Docker and many other container runtimes and a Singularity recipe file [13] for Singularity.

On a bare metal system with environment modules, using a software component is as simple as loading the corresponding module. However, including a software component in a container image requires knowing how to properly configure and build the component. This is specialized knowledge and can be further complicated when applying container best practices. For those accustomed to just loading the relevant environment modules, installing a compiler, MPI library, CUDA, and other core HPC components from scratch may be daunting. A method that has the simplicity of loading an environment module but also utilizes container best practices is needed.

II. Making Containers Easier

HPC Container Maker is an open source project that addresses the challenges of creating HPC application containers. HPCCM encapsulates into modular building blocks the best practices of deploying core HPC components with container best practices, to reduce container development effort and minimize image size. HPCCM makes it easier to create HPC application containers by separating the choice of what should go into a container image from the specification and syntax details of how to configure, build, and install a component. This separation also enables the best practices of HPC component deployment to transparently evolve over time.

With HPCCM, the container specification file, or recipe, is a Python script. This provides several advantages. It abstracts the recipe from the specific syntax details of a Dockerfile or Singularity recipe file. Given a Python recipe file and the desired output format, HPCCM generates the corresponding Dockerfile or Singularity recipe file. The resulting specification file can be used as normal in the corresponding container image build process.

A Python based recipe also allows the full capabilities of the Python programming language to be used. For example, a recipe might validate input, branch depending on the Linux distribution of the base image, modify a complex build process based on the selected numerical precision, or search for the latest available version of a particular component to download.

Most significantly, the high-level recipe specification separates the choice of what should go into a container image from the low-level syntax details of the container framework specification file. The process of converting the high-level recipe specification also allows the seamless application of container best practices such as effective Docker image layering and multi-stage builds and abstracts any differences due to the Linux distribution of the base container image. Best practices for building each software component can also be seamlessly employed and be transparently updated without impacting the content of the Python recipe specification.

A. Building Blocks

A key feature of HPCCM is its set of building blocks, high-level abstractions of key HPC software components. Building blocks are roughly equivalent to environment modules, except that building blocks are configurable. A list of the building blocks currently available with HPCCM are shown in Table I. CUDA should be included via an appropriate base image [14], hence a CUDA building block is not provided.

The openmpi building block illustrates the benefits. This simple two line Python HPCCM recipe generates the Dockerfile or the Singularity recipe file shown in Fig. 1 and Fig. 2, respectively.

```
Stage0 += baseimage(
    image='nvidia/cuda:9.2-devel-ubuntu16.04')
Stage0 += openmpi()
```

This statement adds OpenMPI to the first stage of the Dockerfile. Docker, but not Singularity, supports multi-stage builds to reduce the size of the resulting container image. To add a component to the second stage of a multi-stage recipe, use Stage1 instead of Stage0.

HPCCM building blocks are Linux distribution aware. In this example, the base image is derived from the Ubuntu Linux distribution, so the apt package manager is used to install any required packages. If the base image was derived from CentOS, the yum package manager would have been used instead. The base image Linux distribution detection is automatic and normally requires no action by the user.

Most building blocks also have configuration options to enable customization. For instance, the openmpi building block has options to specify the version, the installation path, the compiler toolchain to use, whether to enable CUDA and InfiniBand support, and so on. Reasonable defaults are set so the configuration is usually optional. Documentation for all of the building blocks and their configuration options is included with the HPCCM package.

Some building blocks may require a license to use, although currently only the Intel Parallel Studio (intel_psxe) building block requires a license. In those cases, HPCCM expects the user to provide their own valid license and the license information can be specified via a building block configuration option. By using multi-stage Docker builds, licensed software can be

```
FROM nvidia/cuda:9.2-devel-ubuntu16.04

# OpenMPI version 3.0.0
RUN apt-get update -y && \
    apt-get install -y --no-install-recommends \
        bzip2 \
        file \
        hwloc \
        make \
        openssh-client \
        perl \
        tar \
        wget && \
    rm -rf /var/lib/apt/lists/*
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 --with-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
```

Fig. 1. Dockerfile corresponding to the openmpi building block.

```
BootStrap: docker
From: nvidia/cuda:9.2-devel-ubuntu16.04
%post
    . /.singularity.d/env/10-docker.sh

# OpenMPI version 3.0.0
%post
    apt-get update -y
    apt-get install -y --no-install-recommends \
        bzip2 \
        file \
        hwloc \
        make \
        openssh-client \
        perl \
        tar \
        wget
    rm -rf /var/lib/apt/lists/*

%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 --with-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
```

Fig. 2. Singularity recipe file corresponding to the openmpi building block.

used to build an application without needing to redistribute the licensed software or license itself.

B. Primitives

While the container specification file syntax may differ depending on the container runtime, the same types of operations are performed, e.g., executing shell commands, copying files into the container image, setting the environment, etc. A list of the primitives currently available with HPCCM are shown in Table II. HPCCM primitives are wrappers around these basic operations that translate the operation into the corresponding container specific syntax. All the building blocks are implemented on top of primitives to simplify supporting multiple container specification output formats.

For example, the ”shell” primitive runs a list of commands:

```
shell(commands=['a', 'b', 'c'])
```

In this case, HPCCM generates the following Dockerfile syntax (note that the commands are expressed as a single Docker layer):

```
RUN a && \
    b && \
    c
```

And HPCCM generates the following Singularity recipe file syntax:

```
%post
    a
    b
    c
```

Where a building block is available it should be used instead of the primitive equivalent.

Primitives also hide many of the differences between the Docker and Singularity container image build processes so that behavior is consistent regardless of the output configuration specification format. This adheres to the principle of least surprise for users accustomed to writing container specification files in only one of the formats. For example, the Dockerfile `ENV` instruction sets environment variables immediately, i.e.,

TABLE I
HPCCM BUILDING BLOCKS

Building Block	Description
apt_get	Apt package manager
boost	Boost
cgns	CFD General Notation System
charm	Charm++
cmake	CMake
fftw	FFTW
gnu	GNU compilers
hdf5	HDF5
intel_mpi	Intel MPI Library
intel_psxe	Intel Parallel Studio
mkl	Intel Math Kernel Library
mlnx_ofed	Mellanox OpenFabrics Enterprise Distribution (OFED)
mvapich2	MVAPICH2
mvapich2_gdr	MVAPICH2-GDR
netcdf	NetCDF
ofed	OpenFabrics Enterprise Distribution (OFED) - Linux distribution packages
openblas	OpenBLAS
openmpi	OpenMPI
packages	Linux distribution aware package management
pgi	PGI Community Edition compilers
pnetcdf	Parallel NetCDF
python	Python
yum	Yum package manager

TABLE II
HPCCM PRIMITIVES

Primitive	Description
baseimage	Set the container base image
blob	Insert a file containing a specification blob
comment	Comment string
copy	Copy file(s) from the host into the container image
environment	Set environment variables
label	Set container image labels
raw	Insert verbatim (”raw”) text
shell	Execute shell commands
user	Set the user
workdir	Set the working directory

the value of an environment variable can be used in any subsequent instructions, while the Singularity `%environment` block sets environment variables only when the container is running. Therefore the environment primitive generates an additional Singularity `%post` block by default (the behavior can be disabled with a configuration option.)

```
environment(variables={'A': 'a',
                       'B': 'b',
                       'C': 'c'})
```

For the above primitive, HPCCM generates the following Dockerfile syntax:

```
ENV A=a \
    B=b \
    C=c
```

And HPCCM generates the following Singularity recipe file syntax:

TABLE III
HPCCM TEMPLATES

Template	Description
ConfigureMake	Configure / make / make install workflow
git	Clone git repositories
sed	Modify files
tar	Work with tar archive files
wget	Download files

```
%environment
    export A=a
    export B=b
    export C=c
%post
    export A=a
    export B=b
    export C=c
```

C. Templates

Some operations are very common and invoked by multiple building blocks, such as cloning a git repository or executing the configure / make / make install workflow. HPCCM templates abstract these basic operations for consistency and to avoid code duplication. A list of the building blocks currently available with HPCCM are shown in Table III.

Templates are primarily intended to be used by building blocks and thus are not exported by default for use in recipes.

D. Recipes

An application recipe uses HPCCM building blocks and primitives, as well as other Python code, to generate the corresponding Dockerfile or Singularity recipe file when processed by HPCCM. The goal of HPCCM is to provide the building blocks for creating application recipes, not application recipes themselves. HPCCM includes some working application recipes as examples to demonstrate how to do this.

HPCCM also includes several ”base” recipes for several combinations of compilers and MPI libraries. The base recipes may be used to generate base container images that can be used as the starting point for building application containers. Integrating HPCCM into the workflow of generating application container images is discussed further in Section III.

E. Recipe Example: MPI Bandwidth

MPI Bandwidth [15] is a simple MPI micro-benchmark and proxy application. The complete HPCCM recipe for a container image is shown in Fig. 3 and is included with HPCCM. The corresponding Dockerfile and Singularity recipe file can be generated with the commands:

```
hpccm --recipe mpi_bandwidth.py --format docker

hpccm --recipe mpi_bandwidth.py --format singularity
```

The HPCCM Python recipe is just 8 lines of actual code and 398 bytes (excluding comments). By comparison, the Dockerfile is 35 lines of code and 2378 bytes and the Singularity recipe file is 48 lines of code and 2561 bytes. The

```
"""
MPI Bandwidth
Contents:
  CentOS 7
  GNU compilers (upstream)
  Mellanox OFED
  OpenMPI version 3.0.0
"""

Stage0 += comment(__doc__, reformat=False)

# CentOS base image
Stage0 += baseimage(image='centos:7')

# GNU compilers
Stage0 += gnu(fortran=False)

# Mellanox OFED
Stage0 += mlnx_ofed()

# OpenMPI
Stage0 += openmpi(cuda=False, version='3.0.0')

# MPI Bandwidth
Stage0 += shell(commands=[
    'wget -q -nc --no-check-certificate -P /var/tmp https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bandwidth.c',
    'mpicc -o /usr/local/bin/mpi_bandwidth /var/tmp/mpi_bandwidth.c'])
```

Fig. 3. HPC Container Maker recipe for the MPI Bandwidth workload (mpi_bandwidth.py).

opportunity for errors in the container specification is less for the considerably smaller and higher-level recipe. HPCCM users have also reported significant increases in productivity due to not having to be concerned with low level syntax or best practices for containerizing specific software components.

III. Workflows

HPC Container Maker enables three workflows for creating HPC application container images.

A. Application Recipes

The MPI Bandwidth example showed how the entire specification of an application container image can be expressed as a HPCCM recipe. One of the other examples included with HPCCM is GROMACS [16], a popular molecular dynamics application.

The GROMACS recipe demonstrates how to use multi-stage Docker builds to minimize the size of the resulting container image. The first stage includes the required building blocks and GROMACS source code and then builds the application binary. Build artifacts such as the application source code, compiler, and development versions of the software toolchain are part of the image at this stage. Since only the final application binary and its runtime dependencies are needed when deploying the container image, the container image size can potentially be reduced significantly. The second stage uses a runtime version of the CUDA base image and copies the application binary and the required runtime dependencies from the first stage to generate an optimal size container image. HPCCM building blocks provide a method to easily copy the runtime from the first stage into the second stage.

The two-stage, optimized size container specification can be generated with the command:

```
hpccm --recipe gromacs.py
```

For comparison, a container specification for the first stage only can be generated with the command:

```
hpccm --recipe gromacs.py --single-stage
```

The Docker image size for the two-stage recipe is 1.08 GB compared to 3.27 GB for the single-stage image, over a 3X reduction in size for no loss of functionality. For additional comparison, similar EasyBuild and Spack generated GROMACS container images are 6.91 and 3.49 GB, respectively. Minimizing container image size is necessary for broad container adoption by HPC data centers. Users may work with several container images at a time and each user may have personal copies of an application container image, consuming considerable disk space if the container image size is not optimized.

This workflow is the most portable since the HPCCM recipe can be used to generate either Docker or Singularity container specification files. Python HPCCM recipes are also more flexible than either Dockerfile or Singularity recipe file syntax.

B. Base Image Generation

On a bare metal system using environment modules, the typical workflow is to first setup the necessary software environment by loading the appropriate modules. With the proper software environment loaded, an application can be built or run.

This ”base” software environment is analogous to the base image used when building a container image. A base image with the proper set of HPC software components present can be used as the starting point for building an application container image.

HPCCM includes base recipes for all combinations of the GNU and PGI compilers with the OpenMPI and MVAPICH2 MPI libraries, plus commonly used software components such as the Mellanox OpenFabrics Enterprise Distribution, Python, FFTW, and HDF5. The provided base recipe can be easily customized to change component versions or add or subtract building blocks.

For instance, to generate a GNU compiler and OpenMPI base image from the included base recipe:

```
hpccm --recipe hpcbase-gnu-openmpi.py --single-stage
    ↪  > Dockerfile
sudo docker build -t hpcbase-image -f Dockerfile .
```

The resulting local container image can then be referenced by an application Dockerfile or Singularity recipe file.

```
FROM hpcbase-image
# Application build instructions
...
```

```
BootStrap: docker
From: hpcbase-image
# Application build instructions
...
```

C. Template Generation

Instead of going through the intermediate step of building a base image with the necessary HPC software components, the container specification file produced by HPCCM can be extended directly to include the application build instructions. In other words, HPCCM generates the boilerplate container specification syntax to install the necessary HPC software components with the application specific content appended by the user. The drawback to this approach is that it may be cumbersome to incorporate HPCCM building block improvements.

IV. Conclusion

Containers offer the huge benefits of HPC application portability and reproducibility, especially when curated container images are available from a container registry. However, when an application container is not available, creating the container image from scratch can be challenging for HPC users accustomed to traditional bare metal environments.

HPC Container Maker is an open source project that addresses these challenges by providing a high-level configurable Python recipe format for deploying HPC components into container images. HPCCM supports both Dockerfiles and Singularity recipe files and provides several advantages over specifying an application container image in those formats. HPCCM is also available as a PyPi module for easy installation [17].

Acknowledgment

The author thanks Logan Herche, Jeff Layton, CJ Newburn, Chintan Patel, and Adam Simpson of NVIDIA Corporation for feedback and encouragement during the development of HPC Container Maker.

References

[1] R. Xu, F. Han, and N. Dandapanthula. "Containerizing HPC Applications with Singularity." [Online]. Available: https://downloads.dell.com/manuals/all-products/esuprt_software/esuprt_it_ops_datcentr_mgmt/high-computing-solution-resources_white-papers10_en-us.pdf

[2] N. Sundararajan, J. Weage, and N. Dandapanthula. "Performance of LS-DYNA on Singularity Containers." [Online]. Available: https://downloads.dell.com/manuals/all-products/esuprt_software/esuprt_it_ops_datcentr_mgmt/high-computing-solution-resources_white-papers83_en-us.pdf

[3] D. Merkel. "Docker: lightweight Linux containers for consistent development and deployment", *Linux J.*, vol. 2014, no. 239, 2014.

[4] G. M. Kurtzer, V. Sochat, and M. W. Bauer. Singularity: Scientific containers for mobility of compute," *PLoS ONE*, vol. 12, no. 5, 2017.

[5] [5] D. M. Jacobsen and R. S. Canon. "Contain this, unleashing Docker for HPC," *Proceedings of the Cray User Group*, 2015.

[6] R. Priedhorsky and T. Randles, "Charliecloud: unprivileged containers for user-defined software stacks in HPC," *SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, 2017, pp. 1-10.

[7] HPC Container Maker. [Online]. Available: https://github.com/NVIDIA/hpc-container-maker

[8] R. McLay, K. W. Schulz, W. L. Barth and T. Minyard, "Best practices for the deployment and management of production HPC clusters," *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WA, 2011, pp. 1-11.

[9] M. Geimer, K. Hoste and R. McLay, "Modern scientific software management using EasyBuild and Lmod," *2014 First International Workshop on HPC User Support Tools*, New Orleans, LA, 2014, pp. 41-51.

[10] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, S. Futral, "The Spack package manager: bringing order to HPC software chaos," *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, TX, 2015, pp. 1-12.

[11] OCI Image Format Specification. [Online]. Available: https://github.com/opencontainers/image-spec

[12] Dockerfile Reference. [Online]. Available: https://docs.docker.com/engine/reference/builder/

[13] Singularity Container Recipes. [Online]. Available: https://www.sylabs.io/guides/latest/user-guide/container_recipes.html

[14] nvidia/cuda Docker Hub Repository. [Online]. Available: https://hub.docker.com/r/nvidia/cuda/

[15] MPI Bandwidth. [Online]. Available: https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bandwidth.c

[16] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation," Computer Physics Communications, vol. 91, no. 1-3, pp. 43-56, 1995.

[17] Python Software Foundation. [Online]. Available: https://pypi.org/project/hpccm/

Appendix

A. Abstract

The examples and data in this paper are reproducible with the HPC Container Maker source code available on GitHub. The data given in this paper corresponds to the version tagged v18.8.0.

B. Description

1) Check-list (artifact meta information):

- **Program:** HPC Container Maker
- **Publicly available?:** Yes

2) How software can be obtained : The software may be obtained from GitHub: https://github.com/NVIDIA/hpc-container-maker

3) Hardware dependencies: None

4) Software dependencies: The software directly depends on Python (versions 2.7, 3.4, 3.5, or 3.6) and Python modules enum34 and six.

The software indirectly depends on Docker and/or Singularity to actually build container images from the HPCCM generated specification files.

5) Datasets: The example recipes are available from the GitHub repository in the recipes directory.

C. Installation

The software may be installed from PyPi.

```
pip install hpccm
```

D. Experiment workflow

Given an input recipe, HPCCM outputs the corresponding Dockerfile or Singularity recipe file to standard output. For example:

```
hpccm --recipe <file> --format docker
```

```
hpccm --recipe <file> --format singularity
```

If the format is not specified, the Dockerfile format is the default.

E. Evaluation and expected result

The expected results are valid Dockerfiles and Singularity recipe files.

F. Experiment customization

The recipes may be customized by adding building block options, adding or subtracting building blocks, etc.

G. Notes

HPCCM documentation is available from the GitHub repository in Markdown format. README.md is an overview, and the recipe syntax, building blocks, etc. are described in RECIPES.md.