Containers in High-Performance Computing

Dr. Axel Kohlmeyer

Assistant Dean for High-Performance Computing
Associate Director, ICMS
Associate Director, TMI
College of Science and Technology
Temple University, Philadelphia
axel.kohlmeyer@temple.edu

External Scientific Associate
International Centre for Theoretical Physics, Trieste, Italy
akohlmey@ictp.it

Challenges of Software Installation on HPC Clusters

- Software needs to be compiled specifically for the HPC cluster, to match the available libraries and compilers
- Different research projects may require different compilers or libraries or specific versions
- Using environment modules (lmod) allows to have concurrent installation of multiple variants
- Supporting all permutations of variants with environment modules is cumbersome, if there are multiple levels of dependencies

Supporting Different Linux “Flavors”

- The Linux variant used on desktops or laptops is often different from the one on HPC clusters (e.g. Ubuntu vs. CentOS/RHEL) and newer
- Complex software packages are often tested only one Linux “flavor” and it is difficult to reproduce results on others
- Already building a complex software can be demanding because of the “maze” of software dependencies (AKA “dependency hell”)

Virtual Machines

- Using a Virtual Machine software allows to install and run a different OS inside a host OS
- Virtualization comes with a performance penalty
- Interfacing virtual machines to local file systems is done through (emulating) a networked FS
- Installation and maintenance of a virtual machine disk image similar to maintaining a real (hardware) machine
- Abstraction of hardware requires special drivers

Enter the Container

- Container software (e.g. docker) is a more lightweight approach to have a different OS “flavor” inside a host OS
- Uses kernel/hardware features to “contain” access to the host OS and can bootstrap a process (e.g. bash) into the environment of a different OS flavor
- Container management software allows to tailor disk images by adding modifications on top of a base image and leverage existing solutions

Challenges of Using Containers

- Creating/using a container usually requires superuser privilege (if only temporary) and that can be used to gain root access to a machine
- Root privilege is particularly required to build and update container disk images
- Still challenging to provide access to accelerators and fast networks
- => Singularity container software addresses many of these issues (unlike docker)

The Singularity Approach

- Single immutable container image file defines the “contained” environment
  => root access (e.g. via sudo) is needed to build the container, but not to use it
- Container images can be pre-build by vendors or users on one machine (e.g. your workstation) and then would run on any singularity instance
- Can handle MPI (need same version of MPI lib inside and outside container) or access to GPU
- Cryptographic validation of container images

Using Singularity

- Singularity frontend command: singularity
- A variety of subcommands to do specific operations, for example:
  - **`build`** to construct a new container image
  - **`pull`** to download a pre-built container image
  - **`push`** to upload a pre-built container image
  - **`shell`** to start a shell inside a container
  - **`exec`** to run a command inside a container
  - **`verify`** to check validity and signature of image

Building Singularity Container

- Use image file (.sif) or writable file system are
- Writable area needs root to run (→ testing)
- Use descriptor file (.def) to script/automate building of a container
- Use descriptor can be bootstrapped from existing singularity images, docker images, linux installer bootstrap images/repositories
- Script sections in descriptor file define what other installation/setup steps to do and more

Example: Building a Container for Compiling/Debugging LAMMPS

```
BootStrap: library
From: centos:7

%post
        yum -y install epel-release
        yum -y update
        yum -y install vim-enhanced ccache gcc-c++
gcc-gfortran clang gdb valgrind-openmpi make cmake
cmake3 ninja-build patch which file git libpng-
devel libjpeg-devel openmpi-devel mpich-devel
python-devel python-virtualenv fftw-devel voro++-
devel eigen3-devel gsl-devel openblas-devel enchant

%labels
        Author akohlmey
```

Example: Building a Container for Compiling/Debugging LAMMPS

Now we clone the LAMMPS repository, build a local container image file from the descriptor file, create a build folder, open a shell inside the container and then compile LAMMPS there:

```
cd some/work/directory
git clone --depth 500
git://github.com/lammps/lammps.git lammps
mkdir build-centos7
cd build-centos7
sudo singularity build centos7.sif \
../tools/singularity/centos7.def
singularity shell centos7.sif
cmake -C ../cmake/presets/most.cmake -D
CMAKE_CXX_FLAGS="-O3 -g -fopenmp -std=c++11" ../cmake
make
```

Example: Installing a Software with many Dependencies: Sage Math

```
bootstrap: docker
from: ubuntu:18.04
%runscript
exec /opt/SageMath/sage $@
%post
    apt-get -y update
    apt-get -y install wget bzip2 python gfortran
    wget http://mirrors.mit.edu/sage/linux/64bit/sage-
8.9-Ubuntu_18.04-x86_64.tar.bz2 -P/tmp
    tar xvjf /tmp/sage-8.9-Ubuntu_18.04-x86_64.tar.bz2
-C /opt
    rm /tmp/sage-8.9-Ubuntu_18.04-x86_64.tar.bz2
    /opt/SageMath/sage -v
cd /
. /scif/apps/sage/scif/env/01-base.sh
```

Containers in High-Performance Computing

Dr. Axel Kohlmeyer

Assistant Dean for High-Performance Computing
Associate Director, ICMS
Associate Director, TMI
College of Science and Technology
Temple University, Philadelphia
axel.kohlmeyer@temple.edu

External Scientific Associate
International Centre for Theoretical Physics, Trieste, Italy
akohlmey@ictp.it