Dockter Spacklove

or: How I learned to stop worrying and love the package manager

Hadrien Grasland
LAL – Orsay

2018-09-19

Introduction

- **Good comment on a CHEP talk:**

  « Verrou looks like a really nice tool, but I wouldn't build a patched Valgrind just to try it out »

- **Encouraged me to work on a long-time pain point:**
  - Software packaging & portability challenges in HEP
  - How toolchain improvements could address them


(Conflicting)
Requirements

Two visions of portability

- **Most of CERN & WLCG:**
  - Software is portable if it runs on all SLC6 Grid nodes
  - Can assume CVMFS availability, slowly deprecating AFS
- **Rest of the world:**
  - Software is portable if it runs well on *my computer*
  - Using many Linux distros, some other Unices
  - Install in one command, integrate well with my system
  - Always-on & very fast Internet is not a given

Two user profiles

- **« Testers »: Experimenting around**
  - Want to get something running as quickly as possible
  - Can tolerate low integration with the host OS
- **« Developers »: Putting oneself at ease**
  - Want to leverage full system (dev tools, GUIs, drivers…)
  - Can tolerate an occasional large build if automated
- **Can assume some computing & Unix proficiency**


Figuring it out

Serving the « testers »

- **Containers seem best for this use case**
  - Lighter than a VM
  - Simpler to build and maintain
  - Download & run with 1-2 commands
  - Easier to attach system resources (e.g. filesystem)
- **Which implementation?***
  - Singularity is a good idea for scientific computing
  - But Docker wins *massively* on usability for now

* More detailed comparison available on demand

Scaling up to « developers »

- **Need a custom build for full system integration**
  - Use system compiler/libc/tools, link with HW drivers…
  - Avoid too much isolation from host system
- **Integration vs reproducibility trade-off**
  - Remember: no manpower to test many configurations!
  - Best provide our dependencies + opt-in to system libs
  - Need a package manager to sort the dependencies out

Picking package managers

- **Various forms of flexibility needed:**
  - Many programming langages (C++, Python, Fortran…)
  - Good portability across Linuxes, ideally Unices
  - Minimize tampering with user environment
  - Allow opt-in dependencies on host system
  - Coexistence of « cousin » builds (e.g. ROOT C++14 & 17)
- **Spack seems most appropriate* for this use case**

* More detailed comparison available on demand

What package managers bring to containers

- **Simplified maintenance**
  - Turn big Dockerfile into a few spack commands
  - Reusing work across containers is much simpler
- **Easier for « testers » to become « developers »**
  - Dockerfiles are specific to a base Linux distribution
  - But Spack commands are portable across systems
  - They help reproducing a container’s setup locally

In practice

Stress-testing the approach

- **Previously moved my build recipes to Dockerfiles**
  - Simplifies automated testing (easy rollback on failure)
  - Easier to move my dev environment to another machine
  - Quickly share a dev environment with coworkers
- **Here, tried to move most shell work to Spack**
  - Test projects: Verrou, Templight, ACTS, Gaudi
  - Aiming for full builds (~all optional dependencies)

Building and testing Gaudi* (before)

```
# === DOCKER-SPECIFIC HACKERY ===

FROM hgrasland/root-tests
LABEL Description="openSUSE Tumbleweed environment for Gaudi" Version="0.1"
CMD bash


# === SYSTEM SETUP ===

# Update the host system
RUN zypper ref && zypper dup -y

# Install non-ROOT requirements
RUN zypper in -y doxygen graphviz cppunit-devel gdb unzip libxerces-c-devel \
                 uuid-devel libunwind-devel gperftools gperftools-devel \
                 jemalloc-devel ncurses5-devel ninja wget python2-nose \
                 python2-networkx which curl libuuid-devel

# === INSTALL (OLDER) BOOST ===

# Download Boost v1.66 (Gaudi does not support v1.67+ yet)
RUN git clone --recursive -j8 --branch=boost-1.66.0 --depth=1 \
    https://github.com/boostorg/boost.git

# Build and install Boost
RUN cd boost \
    && ./bootstrap.sh --with-python=python2.7 \
    && ./b2 -j8 \
    && ./b2 install

# Work around Boost's brain damaged build system
RUN cp -rf boost/libs/program_options/include/boost/* \
          /usr/local/include/boost/ \
    && cp -rf boost/libs/utility/include/boost/* \
              /usr/local/include/boost/ \
    && cp -rf boost/libs/circular_buffer/include/boost/* \
              /usr/local/include/boost/ \
    && cp -rf boost/libs/ptr_container/include/boost/* \
              /usr/local/include/boost/ \
    && cp -rf boost/libs/assign/include/boost/* \
              /usr/local/include/boost/

# Get rid of the Boost build directory
RUN rm -rf boost


# === INSTALL C++ GUIDELINE SUPPORT LIBRARY ===

# Download the GSL
RUN git clone --depth=1 https://github.com/Microsoft/GSL.git

# Build the GSL
RUN cd GSL && mkdir build && cd build \
    && cmake -GNinja -DCMAKE_BUILD_TYPE=RelWithDebInfo \
             -DGSL_CXX_STANDARD=14 .. \
    && ninja


# Check that the GSL build is working properly
RUN cd GSL/build && ctest -j8

# Install the GSL
RUN cd GSL/build && ninja install

# Get rid of the GSL build directory
RUN rm -rf GSL


# === INSTALL RANGE-V3

# Download the range-v3 library (v0.3.5)
RUN git clone --branch=0.3.6 --depth=1 \
              https://github.com/ericniebler/range-v3.git

# Build range-v3
RUN cd range-v3 && mkdir build && cd build \
    && cmake -GNinja -DRANGES_CXX_STD=14 .. && ninja

# Check that the range-v3 build is working properly
RUN cd range-v3/build && ctest -j8

# Install range-v3
RUN cd range-v3/build && ninja install

# Get rid of the range-v3 build directory
RUN rm -rf range-v3


# === INSTALL AIDA ===

# Download, extract and delete the AIDA package
RUN mkdir AIDA && cd AIDA \
    && wget \
       ftp://ftp.slac.stanford.edu/software/freehep/AIDA/v3.2.1/aida-3.2.1.zip \
    && unzip -q aida-3.2.1.zip \
    && rm aida-3.2.1.zip

# Install the AIDA headers
RUN cp -r AIDA/src/cpp/AIDA /usr/include/

# Get rid of the rest of the package, we do not need it
RUN rm -rf AIDA


# === INSTALL CLHEP ===

# Download CLHEP
RUN git clone --branch=CLHEP_2_4_1_0 --depth=1 \
              https://gitlab.cern.ch/CLHEP/CLHEP.git

# Build CLHEP
RUN cd CLHEP && mkdir build && cd build \
    && cmake -GNinja .. && ninja

# Test our CLHEP build
RUN cd CLHEP/build && ctest -j8

# Install CLHEP
RUN cd CLHEP/build && ninja install

# Get rid of the CLHEP build directory
RUN rm -rf CLHEP


# === INSTALL HEPPDT v2 ===

# Download and extract HepPDT v2
RUN curl \
      http://lcgapp.cern.ch/project/simu/HepPDT/download/HepPDT-2.06.01.tar.gz \
      | tar -xz

# Build and install HepPDT
RUN cd HepPDT-2.06.01 && mkdir build && cd build \
    && ../configure && make -j8 && make install

# Get rid of the HepPDT build directory
RUN rm -rf HepPDT-2.06.01


# === INSTALL HEPMC v3 ===

# Download HepMC v3
RUN git clone --depth=1 https://gitlab.cern.ch/hepmc/HepMC3.git

# Build and install HepMC
RUN cd HepMC3 && mkdir build && cd build \
    && cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo .. \
    && make -j8 && make install

# Get rid of the HepMC build directory
RUN rm -rf HepMC3


# === INSTALL HEPMC v2 ===

# NOTE: Why are we overwriting our HepMC3 install with a HepMC2 one, you may
#       wonder? The answer has to do with RELAX being hopelessly broken, and
#       expecting the CMake files of HepMC3 together with the headers of HepMC2

# Download HepMC v2
RUN git clone --depth=1 https://gitlab.cern.ch/hepmc/HepMC.git

# Build HepMC
RUN cd HepMC && mkdir build && cd build \
    && cmake -Dmomentum=GEV -Dlength=MM .. && make -j8

# Test our build of HepMC
RUN cd HepMC/build && make test -j8

# Install HepMC and remove bits of HepMC3 (ugh...)
RUN cd HepMC/build && make install \
    && rm /usr/local/lib64/libHepMC.so /usr/local/lib64/libHepMC.a

# Get rid of the HepMC build directory
RUN rm -rf HepMC


# === INSTALL RELAX ===

# Downlad and extract RELAX (yes, this file is not actually gzipped)
RUN curl http://lcgpackages.web.cern.ch/lcgpackages/tarFiles/sources/RELAX-root6.tar.gz \
      | tar -x


# Build and install RELAX (wow, such legacy, much hacks!)
RUN cd RELAX && mkdir build && cd build \
    && ln -s `which genreflex` /genreflex \
    && export CXXFLAGS="-I/usr/local/include/root/" \
    && cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo \
    && make -j8 && make install \
    && rm /genreflex && unset CXXFLAGS

# Get rid of the RELAX build directory
RUN rm -rf RELAX


# === ATTEMPT A GAUDI TEST BUILD ===

# Clone the Gaudi repository
RUN git clone --origin upstream https://gitlab.cern.ch/gaudi/Gaudi/

# Configure Gaudi
RUN cd Gaudi && mkdir build && cd build \
    && cmake -DGAUDI_DIAGNOSTICS_COLOR=ON -GNinja ..

# Configure the run-time linker
#
# NOTE: I am not sure why this is needed for this build specifically, but the
#       Gaudi build will fail to find CLHEP if we don't do it.
#
RUN ldconfig

# Build Gaudi
RUN cd Gaudi/build && ninja

# Test the Gaudi build
#
# NOTE: Some Gaudi tests do ptrace system calls, which are not allowed in
#       unprivileged docker containers because they leak too much information
#       about the host. You can allow the container to run these tests
#       by passing the "--security-opt=seccomp:unconfined" flag to docker run,
#       but for some strange reason this flag cannot be passed to docker build.
#       Therefore, we disable these tests during the docker image build.
#
RUN cd Gaudi/build \
    && ctest -j8 -E "(google_auditors\.heapchecker|event_timeout_abort)"

# Remove build byproducts to keep image light
RUN cd Gaudi/build && ninja clean


# === FINAL CLEAN UP ===

# Discard the system package cache to save up space
RUN zypper clean
```

* With most optional components + assuming ROOT is already installed

Building and testing Gaudi* (after)

```
FROM hgrasland/root-tests:latest-cxx17
LABEL Description="openSUSE Tumbleweed environment for Gaudi" Version="0.1"
CMD bash

# Use my Gaudi package development branch
#
# TODO: Remove this once it's integrated in Spack
#
RUN cd /opt/spack && git fetch HadrienG2 && git checkout gaudi-package

# Build a spack spec for Gaudi
RUN echo "export GAUDI_SPACK_SPEC=\"gaudi@develop +tests +optional             \
                                    ^ ${ROOT_SPACK_SPEC}\"" >> ${SETUP_ENV}

# Build Gaudi and its dependencies using Spack
RUN spack build ${GAUDI_SPACK_SPEC}

# Test the Gaudi build
#
# NOTE: Some Gaudi tests do ptrace system calls, which are not allowed in
#       unprivileged docker containers because they leak too much information
#       about the host. You can allow the container to run these tests
#       by passing the "--security-opt=seccomp:unconfined" flag to docker run,
#       but for some strange reason this flag cannot be passed to docker build.
#       Therefore, we disable these tests during the docker image build.
#
RUN spack cd --build-dir ${GAUDI_SPACK_SPEC}                                  \
    && cd spack-build                                                         \
    && spack env gaudi+tests+optional                                         \
           ctest -j8 -E "(google_auditors\.heapchecker|event_timeout_abort)"

# Drop the build to save space in the final Docker image
#
# (This will preserve dependencies, therefore reinstalling should be quick)
#
RUN spack clean ${GAUDI_SPACK_SPEC}
```

- **Does the same thing…**
  - In 36 LOCs instead of 234
  - 2/3 of which are comments
- **Where did the logic go?**
  - Much is handled by Spack
  - Remainder is in packages
  - More easily reusable!

* With most optional components + assuming ROOT is already installed

Conclusion

- **Containers and package managers are friends**
  - Share recipes across Dockerfiles & with others
  - Delegate repetitive churn to the package manager
  - Smooth user transition from virtualized to local build
- **State of my experiment:**
  - Verrou & Templight are merged in official Spack repo
  - Gaudi & ACTS* are waiting for ROOT in [WIP] PRs

* Only the DD4hep part is submitted at the moment, to avoid stacking [WIP] PRs.


Questions?

VMs vs Containers

- **Advantages of VM:**
  - Better portability to non-Linux / ancient kernels
  - Tigher isolation of untrusted code from host
  - Much better for GUI and other GPU work
- **Advantages of containers:**
  - Easier for end users
  - Lighter system resource usage
  - Can document & reproduce the build process

Docker vs Singularity

- **Singularity is a better idea for scientific compute**
  - No root-equivalent permission needed
  - No unnecessary isolation from host system
  - Self-contained images reduce need for registries
- **Docker is a more mature product**
  - More reliable toolchain
  - More straightforward workflows
  - Much better documentation on the Web

Comparing package managers

- **Options which I studied most closely:**

	Multi-language	Multi-OS	Env-preserving	Vendored deps	Native deps	Easy	Fast	Cousin builds	Non-HEP users
LCGCMake	Y	N	Y	Y	N	N	N	N	N
APT / RPM	Y	~	N	N	Y	Y	Y	N	Y
Spack	Y	Y	Y	Y	~	Y	N	Y	Y
Nix	Y	~	Y	Y	N	N	N	Y	Y

- **Important points that did not matter for *this* study:**
  - Institutional support from WLCG & CERN
  - Relocation & distribution of pre-built binaries