Dockter Spacklove

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

Hadrien Grasland
LAL – Orsay

2018-09-19

Introduction

- **Un commentaire sur ma présentation CHEP:**

  « Verrou semble être un outil très intéressant, mais je n'irais pas jusqu'à patcher Valgrind pour l'essayer »

- **Cela m'a motivé à affronter de vieux problèmes :**
  - La portabilité logicielle en physique des hautes énergies
  - Quels outils permettraient d'améliorer la situation

Cahier des charges

Quand on dit « portabilité logicielle »

- **Le CERN et WLCG comprennent ceci :**
  - Fonctionne sur n'importe quel nœud x86 sous SLC6
  - CVMFS est toujours disponible, AFS l'est généralement
- **Le reste du monde comprend cela :**
  - Fonctionne bien sur *ma machine de travail*…
  - …qui peut utiliser divers systèmes Linux & Unix
  - Tout s'installe en une commande, s'intègre bien à l'OS
  - Je peux l'utiliser sans connexion Internet permanente

Deux profils d’utilisateur

- **Le « testeur » veut juste essayer un logiciel**
  - Il faut pouvoir l’installer aussi vite que possible
  - Ce n’est pas grave si tout n’est pas très bien intégré
- **Le « développeur » se met à l’aise**
  - Veut utiliser son système à fond (outils, GUIs, pilotes...)
  - Tolère une compilation, si elle est bien automatisée
- **Les deux ont un bon niveau de maîtrise d’Unix**

En quête d’une solution

Commençons par les « testeurs »

- **Les conteneurs semblent une bonne solution***
  - Plus léger qu'une machine virtuelle
  - Plus simple à construire et à maintenir
  - Plus simple à utiliser (tout est faisable en une commande)
  - Plus facile d'attacher des ressources OS (ex : dossier partagé)
- **Quelle implémentation?***
  - Singularity semble plus pertinent pour l'info. scientifique
  - Mais l'écosystème Docker semble *nettement* plus mature

* On peut en discuter davantage

Satisfaire les « développeurs »

- **Il faut s'intégrer complètement au système hôte**
  - Compilateur, libc, outils de dev', accès au matériel…
  - On doit être prudent avec les mécanismes d'isolation
- **Compromis entre intégration et reproductibilité**
  - Pas de main-d'oeuvre pour tester divers systèmes…
  - Bon équilibre : deps fournies + version native utilisable
  - Pour gérer cela, il faut un gestionnaire de paquets

Choisir un gestionnaire de paquets

- **Pas mal de contraintes :**
  - Nombreux langages (C++, Python, Fortran…)
  - Portabilité Linux, idéalement Unices (macOS, BSD…)
  - Eviter de perturber l'environnement utilisateur
  - Autoriser l'utilisation des bibliothèques de l'hôte
  - Coexistence de paquets « cousins » (ex : ROOT C++14/17)
- **Spack offre une bonne réponse* à ces besoins**

* On peut en discuter davantage

Apport des paquets aux conteneurs

- **Maintenance des conteneurs plus simple**
  - Plus de grosse Dockerfile, juste quelques commandes
  - Le partage de logiciels entre conteneurs est facilité
- **Un « testeur » devient facilement « développeur »**
  - Le contenu d'une Dockerfile est spécifique à un système
  - Les commandes Spack, en revanche, sont portables
  - Il est donc facile de les reproduire localement

Mise en pratique

Un cas d’utilisation

- **J’avais des environnements de build Dockerisés**
  - Simplifie le test automatique (et le retour en arrière)
  - Permet de transférer ces environnements entre machines
  - On peut se les échanger facilement entre collègues
- **J’ai migré tout ce que je pouvais à Spack**
  - Projets logiciels testés : Verrou, Templight, ACTS, Gaudi
  - But : versions complètes, deps. optionnelles comprises

Dockerfile pour Gaudi* (avant)

```dockerfile
# === 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
```

* Avec les dépendances optionnelles, en supposant que ROOT est déjà installé

Dockerfile pour Gaudi* (après)

```
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}
```

- **On fait le même travail…**
  - En 36 lignes au lieu de 234
  - 2/3 étant des commentaires
- **Où est passé le code?**
  - Spack en fait beaucoup
  - Le reste est dans les paquets
  - Ceux-ci sont réutilisables !

* Avec les dépendances optionnelles, en supposant que ROOT est déjà installé

Conclusion

- **Conteneurs ? Paquets ? Pourquoi pas les deux !**
  - On partage plus facilement du travail entre Dockerfiles
  - On confie au gestionnaire de paquets la basse besogne
  - L'utilisateur passe facilement du virtualisé au local
- **Où en suis-je arrivé ?**
  - Les paquets Verrou & Templight sont intégrés à Spack
  - Gaudi & ACTS* attendent une PR concernant ROOT

* Je n'ai soumis que la partie DD4hep pour éviter d'accumuler les [WIP]s...

Des questions?

Machines virtuelles vs conteneurs

- **Avantages des VMs :**
  - Marchent mieux sur les vieux noyaux, les OS non-Linux
  - Isolation plus poussée du code virtualisé
  - Applications graphiques plus faciles d’utilisation
- **Avantages des conteneurs :**
  - Configuration utilisateur bien plus simple
  - Moins gourmands en ressources système
  - Construction documentée et reproductible

Docker ou Singularity ?

- **Singularity : mieux adapté pour le scientifique ?**
  - Pas besoin d'être (équivalent) root
  - L'isolation est minimale → meilleure intégration à l'hôte
  - Plus besoin de « registries » centralisés
- **Docker est un produit plus mature**
  - Outillage plus fiable et messages d'erreur plus clairs
  - Interface assez informative, facile à prendre en main
  - Documentation abondante sur internet

Choix du gestionnaire de paquet

- **Ce que j'ai examiné de plus près :**

	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

- **Aspects importants, mais non prioritaires *ici* :**
  - Support institutionnel du CERN et de WLCG
  - Relocation et distributions de binaires précompilés