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
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 languages (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)

* With most optional components + assuming ROOT is already installed
Building and testing Gaudi* (after)

* With most optional components + assuming ROOT is already installed

- 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!
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
  – Tighter 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:

<table>
<thead>
<tr>
<th>Package</th>
<th>Multi-language</th>
<th>Multi-OS</th>
<th>Env-preserving</th>
<th>Vendored deps</th>
<th>Native deps</th>
<th>Easy</th>
<th>Fast</th>
<th>Cousin builds</th>
<th>Non-HEP users</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCGCMak</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>APT / RPM</td>
<td>Y</td>
<td>~</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Spack</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>~</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Nix</td>
<td>Y</td>
<td>~</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

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