Software Packaging and Deployment in HEP

Ben Morgan and Graeme Stewart
(for the HEP Software Foundation Packaging Group)
What *are* “Packaging” and “Deployment”?

- **A highly context/person dependent term! Here:**
  - The problem of installing and deploying a software stack of N interdependent packages
  - *Not* the build of a single package - handled by package’s local build (CMake/Autotools/...)

- **Solution** is a process and tool(s) that, to install package “foo”:
  - Installs any packages that *foo* requires for build/run time
    - Either: Prepares *foo*’s source code, build environment, then builds/install it
    - Or: Installs pre-existing “binary package” of *foo*
  - Prepares a runtime environment for the use of *foo*
  - *May bundle* *foo* install into “binary package” for clients to install without build-from-source

- **Deployment**: push/pull of this *foo* install/binary package, and dependencies, to another host (via CVMFS, tarball, Container, ...)

2
HEP Software Foundation: Packaging Working Group

- Packaging and deploying a software stack is a problem faced right across the HEP and wider scientific community
  - Every experiment and software group has to put effort into doing this
  - Naively it seems an easy problem, but...
  - Quickly gets complicated...
  - Seemingly obvious solutions don’t meet requirements...

- Motivated formation of WG in 2015 as a forum for working together to improve:
  - Knowledge sharing on tools and workflows in and outside HEP
  - How to take most advantage of technologies like CVMFS/Containers
  - Support/Good Practices for developers building on, users running, a deployed software stack

- [http://hepsoftwarefoundation.org/activities/packaging.html](http://hepsoftwarefoundation.org/activities/packaging.html)
Knowledge Sharing on Packaging Tools

- Input from experiments large and small capturing an evolving picture of tools in use
- Initial report: HSF-TN-2016-03
- Inputs have helped to enumerate Requirements/Use Cases of packaging in HEP
- We’ll walk through some of these to highlight commonalities, differences and contradictions!
The Core Requirements Driving Complexity

- **Must** be able to install and deploy \( N \) Releases of a stack concurrently
  - “Release” defined by a set of packages, plus their versions
  - If a new Release doesn’t change versions of some packages, *should* reuse existing installs
  - A common requirement across the scientific community

- **Must** be able to install and deploy Release \( N \) against \( M \) arch-OS-toolchain “flavours”:
  - E.g. “Stack v1” built against x86_64-centos7-gcc7, ..., x86_64-macos1013-clang9

- This \( NxM \) space seems sparsely populated, but there are extra dimensions...
  - ISA extensions in arch, e.g. “x86_64+avx512”, to support heterogeneous resources
  - Toolchain ABIs, e.g. C++ Standard, Python 2 vs 3, glibc, Optimized vs Debug
  - Package Variants+Dependencies, e.g. “foo with-X”, “bar requires (foo with-X)”
Reproducibility Requirements

- Builds/installs must be deterministic and reproducible
  - Want data+software preservation and reproducible research!

- Builds/installs should be able to reuse base OS packages (rpms, debs)
  - Minimize wheel reinvention, share effort

- But... changes to base OS => new build/run may not be reproducible!
  - Issues like security upgrades may force package updates (e.g., openssl)
  - If BaseOS is not controlled by the experiment ensuring consistency is harder: updates will happen outside its control

- Also a tension between modern compilers/ABIs on older base OSes
  - Development needs modernity, but Infrastructure needs stability
Packaging for Users and Developers

- **System must** set up a correct runtime environment for:
  - users to run programs in the stack.
  - developers to build projects against packages in the stack
- **Runtime environment should** be capable of representing a subset of the stack
  - E.g. minimal analysis, or full end-to-end production
- **System must** be able to chain/layer these subsets to allow sharing and reuse
  - E.g. MyExp (uses) HepPackages (uses) Toolchain
  - Or, DevProject (uses) Toolchain
- **See following talk on Spack/SpackDev** for more on these topics
Packaging vs Software Development Practices

- Packages *should* be **relocatable** after install
  - Can minimize rebuilds from source if install prefix changes between build and install hosts
  - *Implies* support for developers on techniques (no hardcoded paths, self-location)
  - *Implies* packaging tool that supports patching/relocating packages

- Requirement overlaps with community efforts on improving Software Development practices
  - *In this case, how to make your software easy to package!*

- It *doesn’t matter* what tool (CMake, Autotools, ...) a software project uses, but it *does matter* that it follows common practices/standards...
  - HSF C/C++/CMake Project Bootstrap Tool
Early Observations on Tools: FOSS Community

- **Nix**
  - Pure functional package manager
  - See Poster: [Software packaging and distribution for LHCb using Nix](#)
  - Builds deep, own libc - excellent reproducibility
  - Excellent support for multiple versions and flexibly constructed sub-environments
  - Not binary relocatable - install path (default, `/nix`) is a part of the package hash

- **Portage**
  - Package manager from Gentoo Linux...
  - ... but via [Gentoo Prefix](#), can be installed “on top of” Linux base OS, even macOS
  - Builds deep, own libc
  - Supports multiple versions, upgrade and rollback, in each “prefix”
    - Can have several prefixes, plus “overlays” to add your own packages
  - Does support relocation
    - See related presentation tomorrow: [Robust Linux Binaries](#)
Early Observations on Tools: HPC/Science

- **Spack**
  - Developed at LLNL for supporting HPC software
  - Significant number of other users across the scientific community
  - Builds deep (not quite down to libc), but can be told about system libraries
  - *Support for relocation and layered builds being added by HEP users*
  - Runtime/development environment is a WIP (Next talk: [SpackDev - Jim Amundson, FNAL](#))

- **Aware of other tools in this domain (EasyBuild, conda), but limited/no experience of them in recent WG meetings**
  - See [HSF-TN-2016-03](#)
Early Observations on Tools: HEP Community

- aliBuild
  - Developed and used by ALICE, now used by FAIR, NICA, and SHiP
  - Optimised for HEP use
  - Very flexible in use (or not) of system libraries
  - Robust relocation

- LCGCMake
  - Developed by CERN EP-SFT, deployed build products used by:
    - ATLAS, LHCb, SWAN, CERN Beams Department
  - Shallower builds by default (different default from other systems)
  - Small user community (CERN EP-SFT)
  - Supports relocation
  - See presentation tomorrow: Building, testing and distributing common software for the LHC experiments
Test Driving Packaging Tools

- “Test Drive” demos prepared for Nix, Portage, Spack, aliBuild tools
  - Exercise use cases through install of [test HEP package stack](https://github.com/HSF/packaging/tree/master/testdrive)
- Each is an End-to-End walkthrough:
  - CentOS7 Docker image to install base system + packaging tool
  - Install of single package
  - Install of HEP test stack
  - How to add a new package
- Basic documentation to walk you through, linking to official docs for full details
- You are very welcome to try them out!
  - [https://github.com/HSF/packaging/tree/master/testdrive](https://github.com/HSF/packaging/tree/master/testdrive)
Current R&D Activities

- Containers as **the** deployment mechanism for production and development?
  - Reduce dependency between what sites install and what experiments need to Container/Kernel
  - Reduce tension between reusing system packages and rebuilding “everything”?
  - Sweet spot may actually be one of the extreme ends

- “Shallow” stack: allows concentration on packaging HEP Software...
  - ... but needs modern, supported OS+toolchain

- “Deep” stack: guarantees consistency, but requires upstream support
  - ... potentially by Nix and Portage communities. [Effort in LHCb on investigating Nix](#)

- Not so far apart: leverage extensive work and testing by other communities

- Support for developers to easily and consistently use stacks is paramount
  - [Spack/SpackDev effort by FNAL](#)
Summary

- HSF Packaging WG, via community input, explored the space of tools and methods employed to package HEP software stacks
  - Initial Technical Note
  - More in depth enumeration of requirements and use cases

- Extremely useful overview, illustrating complexity...
  - ... but highlighting tools and techniques to cope with this
  - ... and overlap with non-HEP communities

- Test drives of tools to illustrate use cases and techniques
- R&D efforts on Spack, Nix, and Containers underway across community

- A community effort, so fresh input is always welcome!