Spack-Based Packaging and Development for HEP

Presented by K. Knoepfel on behalf of Chris Green, Jim Amundson, Lynn Garren & Patrick Gartung.

CHEP 2018
### What are we trying to do, and for whom?

#### What?
- Replace decades-old lab-written package deployment infrastructure ("UPS" and friends) that no longer meets evolved requirements.
- Use common infrastructure and FOSS wherever possible to minimize maintenance burden and maximize applicability.

#### For whom?
- All experiments, collaborations and projects currently relying on UPS, including those using the *art* framework and LArSoft (ArgoNEUT, DUNE, ICARUS, LArIAT, MicroBooNE, Mu2e, Muon g-2, NOvA, SBND, *artdaq*, CosmoSIS).
- Anyone else with similar needs (e.g. CMS has expressed interest, coordinating with HSF packaging group).
What is Spack?

- From Spack’s home page:

  Spack is a package manager for supercomputers, Linux, and macOS. It makes installing scientific software easy. With Spack, you can build a package with multiple versions, configurations, platforms, and compilers, and all of these builds can coexist on the same machine.

- Developed at LLNL for supporting HPC software.
What are we trying to replace?

UPS (UNIX Product Support)

- Manage binary packages for products\(^a\).
  - 40+ in-house / experiment products, 130+ third-party products.
  - Trivially-relocatable, achieved via \((\text{DY})\text{LD\_LIBRARY\_PATH}\) and environment variable substitution.
- **Variants** for (e.g.) compiler, C++ standard, optimization, MPI . . .
- setup of compatible dependencies to ensure a coherent system.
- Build-system agnostic, no “build language.”
- Ensures **exactly one active version / variant** of a given product.
- Heavily reliant on environment variables.

\(^a\text{Product: Geant4; package: Geant4 built for RHEL7 / x86-64 with GCC 7.3.0 / C++17 and Vecgeom.}\)
What are we trying to replace?

**cetbuildtools**
- CMake-based single-package build system.
- Dependencies satisfied via UPS.
- Produces UPS tarballs.
- Relies on UPS-set environment variables.

**MRB (Multi-Repository Build)**
- Develop and build multiple interdependent cetbuildtools-based packages simultaneously.

**ssibuildshims**
- *Ad hoc* build-helper scripting utilities.
Why switch?

- UPS maintenance burden – unique to Fermilab.
- Relocation via `DYLD_LIBRARY_PATH` not an option on SIP-enabled macOS.
- Users have asked for increased flexibility e.g. ability to try out new versions of packages.
- An eye to HPC.

What does Spack bring?

- Formalized “build language” and dependency system.
- Known and popular in HPC.
- Able to manage multiple versions and variants of each package.
- Active project, many contributors, receptive to contributions.
- >2700 recipes for scientific products and their dependencies contributed by a diverse scientific user community.
What did Spack **not** do out-of-the-box?

*a.k.a.* “Why is this an R&D project?”

- Binary package caching and relocation.
- Support package development.
- Complex variant propagation (*e.g.* C++ standard choice).
- Complex compiler dependency handling (sometimes we care, sometimes we don’t).
- User-defined recipe templates for similarly built products (*e.g.* those using *art*).
What’s been done?

- Successful build and test of the *art* suite and dependencies for one platform / compiler / C++ standard.
- Preliminary version of *cetmodules*: UPS-free replacement for *cetbuildtools*.
- Preliminary version of SpackDev, replacement for MRB.

Contributions to Spack (so far)

- *buildcache* binary package caching and relocation system.
- “Spack chains” feature to allow hierarchical Spack installations.
- New and improved recipes for many packages (ongoing).
- Other minor improvements: compiler features, PATH-like variable management, tests (ongoing).
What’s still to do?

- Finish “MVP” (Minimum Viable Product): a system sufficiently-featured to allow Fermilab experiments to start to play and get a feel for features and limitations:
  - One platform / compiler / C++ standard.
  - One art framework release.
  - Build in place: no binary distribution.

Remaining:

- Some recipes: e.g. Pythia6, improve Geant4.
- Verify and enhance SpackDev operation with art suite and experiments’ packages.
- Verify ability to use Python and Perl extensions built in cetmodules packages (extends() / spack activate vs PYTHONPATH, PERL5LIB).
- Configure installation and subsequent location of user configuration, data files, and binary plugin modules.
Beyond the MVP

- Support multiple concurrently-available compiler / C++ standard and releases without unnecessary builds:
  - Products for which only one package is required: executable / C-only, data-only, pure script (Python, R, etc.) packages\(^1\).
  - Identifier hashes for build configurations change when recipes are updated, invalidating already-built packages.

- Verify full *buildcache* and relocation functionality including libraries, *pkgconfig*, data, configuration file and plugin location.

- Define a versatile recipe template for *cetmodules* packages.

- Packages needed by experiments and collaborations, *e.g.* CodeSynthesis XSD, SMC and related build tools, GENIE, Pandora, *Vecgeom*, *etc.*

- Release and distribution management tools.

\(^1\) *e.g.* Geant4 data tables (~5GiB) don’t need opt / debug or Clang / GCC versions.
Conclusion

- Our requirements are a step or two beyond those Spack was originally conceived to satisfy.
- Definite progress toward the goal, but more work (and pull requests) will be necessary to achieve a full replacement for the existing system.
- Onward for the next 30 years!