Status of Spack Development / Migration

LArSoft Coordination Meeting 2018-05-22

Chris Green, FNAL.
Goals

• Replace UPS with an off-the-shelf build / packaging / deployment system based on Spack.
• Support SIP-enabled MacOS systems.
• Provide tools for developers of experimental software packages (*cf* cetbuildtools, mrb).
• Provide facilities for release preparation, management and deployment (*cf* buildFW, pullProducts, copyToSciSoft, Jenkins scripts).
• Continue to provide relocatable binary distributions and satisfy other requirements that people have relied on with the current suite of tools.
Spack overview

• *De facto* standard tool for HPC.
• Python-based (2 or 3); “specs” -> Python classes.
• Spack “installation”: Spack code + specs from one (“builtin”) or more “Spack repos” + installed packages based on those specs.
• Specs do not generally specify exact versions (although they can). A particular build of a dependency tree can specify versions.
• “Variants” control configurable aspects of a spec such as language standard, optional features.
• Can satisfy dependencies from the system rather than building the world.
• All interoperating packages are tied to a compiler (e.g. Clang 5.0.1, GCC 7.3.0).
• Specs control build instructions, setup behavior, dependencies – essentially part of the installed system as well as describing the build environment.
• A package built in a particular way has an unique hash.
Achievements so far

• Requirements, plans.
• “Buildcache” (Patrick G.): binary packages suitable for relocation / deployment.
• ”Spackdev” (Jim A.): Spack-based multi-package development / test environment.
• “Spack chains” (Jim A.): manage a hierarchy of Spack installations.
• “Cetmodules” (Lynn, Chris): UPS-less cetbuildtools.
• New / improved specs for external packages, other contributions to Spack proper in various stages of discussion / approval (Chris, Lynn, Patrick).
Current efforts

• Produce an “MVP” – Minimum Viable Product:
  – Software stack with one OS / compiler / C++ standard / optimization level to allow experiments to, well, experiment (SLF7 / GCC 7.3 / C++17 / prof).
  – Keep track of issues along the way but achieve the narrow goal first and go back for the others later.
  – Everything built “our way” to maximize realism & compatibility for experiments.
  – Use system-available packages via packages.yaml where possible.
  – First demonstration of Cetmodules.
  – Provide art, gallery and dependencies thereof.
  – NOT “release”-oriented.
  – NOT a collection of every piece of software every experiment is likely to need.
  – NOT a solution to every problem.
  – NOT a guarantee that every remaining problem can be solved.
MVP: status

• **Done** (external packages): Boost, and everything up through Root (except Pythia6, for now), as currently built (as far as possible).

• **Done** (art suite): cetlib_except, cetlib building based on art-v2-develop branches, with all tests passing.

• **Upcoming**: fhicl-cpp, messagefacility should be straightforward; canvas may require work on dictionary building / finding facilities.

• **Upcoming**: shakedown of Spackdev with art suite.
Post-MVP

- Everything is currently tied to the compiler, including things that don’t need to be (e.g. C and Python packages, data-only packages, etc.).
- Unclear how to manage multiple concurrent releases with suitable package reuse.
- Lots of glue to keep things consistent, easy to organize and manage everything from development through release preparation through distribution / deployment.
- More package specs (Geant4, etc., etc.).
- Experiment / Collaboration signoff, migration plan.