EPJ Web of Conferences **251**, 03056 (2021) https://doi.org/10.1051/epjconf/202125103056
CHEP 2021

Building HEP Software with Spack: Experiences from Pilot Builds for Key4hep and Outlook for LCG Releases

Valentin Volkl[1], *Thomas* Madlener[2], *Tao* Lin[3], *Joseph* Wang[4], *Dmitri* Konstantinov[5], *Ivan* Razumov[5], *Andre* Sailer[1], and *Gerardo* Ganis[1]

[1]CERN, Geneva, Switzerland
[2]DESY, Hamburg, Germany
[3]IHEP, Beijing, China
[4]Bitquant Digital Services, Hong Kong
[5]IHEP/NRC Kurchatov

Abstract. Consistent, efficient software builds and deployments are a common concern for all HEP experiments. This paper describes the evolution of the usage of the Spack package manager in HEP in the context of the LCG stacks and the current Spack-based management of Key4hep software. Whereas previously Key4hep software used Spack only for a thin layer of FCC experiment software on top of the LCG releases, it is now possible to build the complete stack, from system libraries to FCC-, iLCSoft- and CEPC software packages with Spack. This pilot build doubles as a prototype for a Spack-based LCG release. The workflows and mechanisms that can be used for this purpose, potential for improvement as well as the roadmap towards a complete LCG release in spack are discussed.

1 Introduction

Spack [1] is a widely used build tool and package manager for scientific software, and is one of the candidates to replace LCGCMake [2], the tool currently used to build/manage the LCG stacks. Investigations of the use of Spack for LCG builds were initiated by the HEP Software Foundation Packaging Working Group and reported at CHEP [3–5]. The continuing interest in Spack triggered a more detailed investigation presented at the HSF framework meeting. The work restarted for in the context of Key4hep which stack is now managed with Spack. The purpose of this contribution is to discuss the issues and solutions/workarounds found so far. Any mentions of Spack refer to the v0.16.0 release and developments as of February 2021.

2 Spack: Overview and Status as of v0.16.0

Originating in the High-Performance Computing community, Spack's philosophy differs from traditional package managers. Its most important features include reproducibility of the builds, and the co-existence of multiple versions and configurations of the same package on one system.

Software is built according to a "recipe", a description of the build process of a package written in a domain-specific language on top of python – see listing 1 for an example. The length of recipes ranges from around 50 to 500 lines of code. Specifically the recipes of packages that use common build systems and require little configuration at build time may be very short and easy to maintain. The largest recipe of a HEP package is that of the ROOT package, which is composed of 510 lines of code from 52 commits.

The upstream Spack repository contains an extensive collection of recipes for general purpose as well as specialised HEP packages. This central repository facilitates sharing of both the maintenance burdens and expertise. The additional scrutiny of the recipes also brings build failures to light and increases the overall robustness of the builds. Currently, 67 HEP-specific software packages are part of Spack, and have been labelled accordingly by adding the `hep` tag. This allows librarians to monitor relevant changes that might otherwise be hard to keep track of in the high volume of overall Spack contributions. Over the last five years, 247 commits from 36 authors have contributed to these HEP package recipes. In addition, several groups maintain smaller repositories with recipes for experiment-specific packages, such as Key4hep.

```
1
2  class Dd4hep(CMakePackage):
3      """DD4hep is a software framework for providing a complete solution for
4         full detector description (geometry, materials, visualization, readout
       ,
5         alignment, calibration, etc.) for the full experiment life cycle
6         (detector concept development, detector optimization, construction,
7         operation). It offers a consistent description through a single source
8         of detector information for simulation, reconstruction, analysis, etc.
9         It distributed under the LGPLv3 License."""
10
11     homepage = "https://dd4hep.web.cern.ch/dd4hep/"
12     url      = "https://github.com/AIDASoft/DD4hep/archive/v01-12-01.tar.gz"
13     git      = "https://github.com/AIDASoft/DD4hep.git"
14
15     maintainers = ['vvolkl', 'drbenmorgan']
16
17     tags = ['hep']
18
19     version('master', branch='master')
20     version('1.15', sha256='992a...')
21     version('1.14.1', sha256='5b5...')
22     ...
23
24     generator = 'Ninja'
25
26     # Workarounds for various TBB issues in DD4hep v1.11
27     # See https://github.com/AIDASoft/DD4hep/pull/613 .
28     patch('tbb-workarounds.patch', when='@1.11')
29     patch('tbb2.patch', when='@1.12.1')
30
31     variant('xercesc', default=False, description="Enable 'Detector Builders'
        based on XercesC")
32     variant('geant4', default=False, description="Enable the simulation part
       based on Geant4")
33     variant('assimp', default=False, description="Enable CAD interface based
       on Assimp")
34     variant('hepmc3', default=False, description="Enable build with hepmc3")
35     variant('lcio', default=False, description="Enable build with lcio")
36     variant('edm4hep', default=True, description="Enable build with edm4hep")
37     variant('debug', default=False,
38             description="Enable debug build flag - adds extra info in"
```

```
            " some places in addtion to the debug build type")

    depends_on('cmake @3.12:', type='build')
    depends_on('ninja', type='build')
    depends_on('boost @1.49:')
    depends_on('root @6.08: +gdml +math +opengl +python +x')
    extends('python')
    depends_on('xerces-c', when='+xercesc')
    depends_on('geant4@10.2.2:', when='+geant4')
    ...

    def cmake_args(self):
        spec = self.spec
        cxxstd = spec['root'].variants['cxxstd'].value
        # root can be built with cxxstd=11, but dd4hep requires 14
        if cxxstd == "11":
            cxxstd = "14"
        args = [
            self.define_from_variant('DD4HEP_USE_EDM4HEP', 'edm4hep'),
            self.define_from_variant('DD4HEP_USE_XERCESC', 'xercesc'),
            self.define_from_variant('DD4HEP_USE_GEANT4', 'geant4'),
            ...
        ]
        return args

    def setup_run_environment(self, env):
        # used p.ex. in ddsim to find DDDetectors dir
        env.set("DD4hepINSTALL", self.prefix)
        env.set("DD4hep_DIR", self.prefix)
        env.set("DD4hep_ROOT", self.prefix)
```

Listing 1. Abridged example of the recipe for the DD4hep package, showing how versions, patches and build options can be declared in Spack. Sources are fetched from the specified url and patches can be applied depending on version ranges and other conditions. The build phase is determined by the base class "CMakePackage" which keeps the recipe compact and only requires custom build options to be added.

Spack calculates a hash from version, build options, dependencies and other information that identifies a packages. "Variants" are custom configurations of the packages that result in different installations. Usually these are mapped to build options and enable or disable optional parts of the package. This also allows to install multiple configurations of the same version of a package. This unique identifier helps reproducibility and allows the installation in arbitrary prefixes – when setting up a package the dependencies are identified by hash and loaded via RPATH and environment variables (depending on the build system type, these may include standard variables like `CMAKE_PREFIX_PATH` or custom ones as set in the package recipe).

Hashing the packages allows Spack to be very flexible with regards to installation areas: in contrast to traditional package managers, packages may be installed in arbitrary prefixes, split over multiple directories (when declaring them as "upstreams") and have custom directory path naming schemes, which can be configured as "projections". Installation folders can be moved or deleted by hand, when refreshing the index of packages afterwards with the dedicated `spack reindex` command. When uninstalling a package, Spack has a garbage-collection feature that allows to remove dependencies that are no longer in use by any other package.

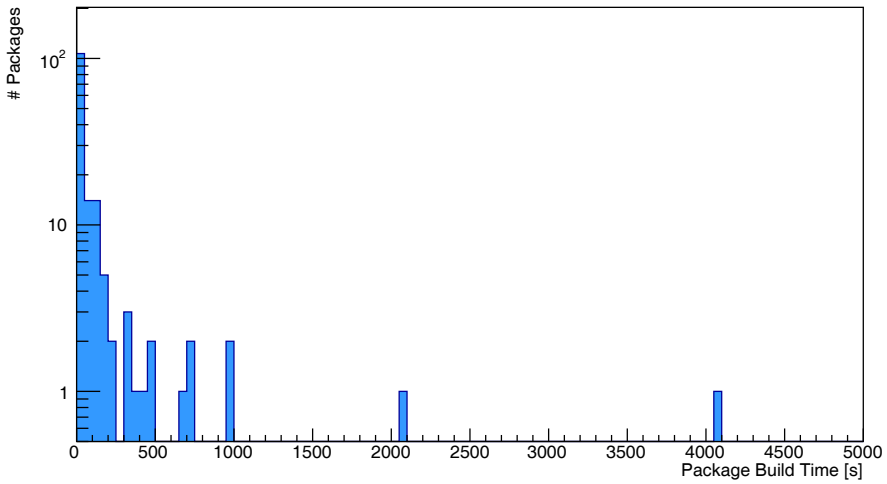

Figure 1. Distribution of build times for a full build of the Key4hep software stack on a CERN openstack vm with 4 cores (flavor "m2.large"). The full build time is just above 5 hours. Only one package (ROOT) takes more than one hour to build.

3 The LCG stacks and derivatives

The LCG stacks are deployments of almost 500 HEP and general-purpose software packages that are used as dependencies for numerous experiments like ATLAS and LHCb. New LCG builds are released several times per year, in 10-20 configurations for different compilers, release/debug build types, operating systems and architectures. Additional software packages, often experiment-specific, can be released in "layers" on top of the base release. LCGCMake [2] is CMake-based build tool specifically developed for this purpose.

4 Spack for Key4hep

Key4hep is a collaborative project with the aim to create a common software for several future collider study projects. Participating experiments include CEPC, FCC and CLIC/ILC. Besides a common data model and a common framework, Key4hep provides a common build infrastructure based on Spack. Future collider studies have diverse requirements for event generators, but the fast and full simulation and reconstruction toolchain is comparable to offline software of running experiments and thus a good prototype for the LCG releases.

The Key4hep-specific package recipes as well as configuration files and scripts for the Gitlab-based deployment workflow are maintained in a dedicated repository [6].

Fig. 1 shows the distribution of build times for the pilot build of Key4hep software and thus gives an indication of size and complexity of the Key4hep software stack. While initially only built in a single configuration for CentOS 7 / GCC 8.3.0, the builds are being expanded to include Debug/Release build types, other compilers and other platforms.

The Key4hep builds are deployed to `/cvmfs/sw.hsf.org/` (and nightly builds to `/cvmfs/sw-nightlies.hsf.org`) by a simple copy operation (using rsync) from the build machine to the CVMFS publisher. This implies that, to avoid relocation, the build machine

needs write permissions on `/cvmfs`. Currently writing to `/cvmfs` is achieved by maintaining a copy of the CVMFS installations on the build machines instead of running CVMFS on it. However, future developments in CVMFS (in particular the `cvmfs_server enter` command) may help to create a writable layer on a machine with CVMFS access [7].

5 Open Issues and Relevant Future Developments

This section details the problems encountered during the pilot build, and lists workarounds and relevant development lines in spack.

5.1 Concretizer

A central component of Spack is the so-called "concretizer", which determines a suitable set of package configurations under the constraint of dependencies and additional configurations. In practice, some manual overspecification (explicitly writing down package variants that the concretizer could have inferred from the dependency tree) in the configuration needs to be done, in order to avoid concretization errors [8]. While these shortcomings do not affect the functionality of spack as a build tool, maintenance of the workarounds can be time-consuming. With v0.16.0 a new concretizer based on clingo, a constraint solver library has been introduced to spack. The new concretizer already removes the need for most of the workarounds, shortens the concretization time, and provides the technical foundation to improve further in a future release as the rules for the constraint solver are updated.

5.2 Data Packages

In some cases, notably Geant4, software packages include large data sets whose format is independent of the compiler or operating system used for a build. When building releases for multiple platform/compiler configurations, it is desirable to deduplicate and share these files among all builds. This is a priori not possible in spack; packages are always tied to a compiler and there is no concept of architecture-agnostic packages. This issue can, however be solved by a manual intervention in the concretization process, namely by declaring the data packages as external and not buildable in the package configuration. While again fully functional, this solution comes again with the increased burden of a separate installation of the data packages and maintenance of the package configuration. Improvements to the dependency model of compilers that would solve this issue are also on the development timeline of spack.

5.3 glibc

While spack can build compilers and some fundamental packages, currently it is not possible to build glibc. This could allow for distribution-agnostic builds similar to Gentoo Prefix, but requires a redesign of the way compiler dependencies are handled in Spack, which is foreseen for one of the next releases.

5.4 Binary caches

Binary caches provide some of the functionality normally found in system package managers (apt, yum/dnf), i.e. installing packages without building them from source. This implies that binaries need to be relocated by patching the `RPATH` section of ELF files (executables and libraries). This approach is considered superior to that of `lcgcmake`, where RPATH sections

are stripped, and `LD_LIBRARY_PATH` (`DYLD_LIBRARY_PATH` on MacOS) environment variable is populated with paths to dependencies. However, several packages in the Key4hep stack are by their design not relocatable, and using binary caches is therefore prone to hard-to-detect runtime errors. Workflows that avoid relocating binaries are therefore considered preferable. Furthermore, the relocation procedure in its present implementation introduces a significant overhead.

If there is no need for relocation (the prefixes on both host and target machine are identical), binaries can also be transferred by a simple copy operation and an update of Spack's package index.

5.5 Distribution of spack on read-only file systems

As of v0.16.0, Spack needs write-access to certain subdirectories of its own installation and will therefore break when installed on read-only file systems. This means that Spack itself cannot be deployed via CVMFS, a significant downside, as the `spack` command could be used by users to modify their environment to set up packages in a fine-grained manner and install packages locally on top of the CVMFS installation. As this requires some configuration and set up of Spack itself, a central installation is considered a requirement for non-expert users. The changes needed for this feature are however already under review, and are expected for the next version.

6 Conclusion and Outlook

Spack has been proven to be a practical solution for software builds of HEP experiments, most recently by the adoption of Spack as the sole build tool for the Key4hep software. While several other build tools offer similar functionality (see [9] for an overview), the large number of contributors to Spack and growing number of HEP packages maintained by members of the community effectively reduces the maintenance burden on individual experiments. A number of possible improvements are discussed in this paper, but none present an insurmountable obstacle to building other HEP software stacks and the commonly used LCG releases, a prototype of which (with some missing packages) can be released in the immediate future.

Acknowledgements

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments (https://cds.cern.ch/record/2649646/, CERN-OPEN-2018-006).

References

[1] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. de Supinski, S. Futral, *The Spack package manager: bringing order to HPC software chaos*, in *SC15: International Conference for High-Performance Computing, Networking, Storage and Analysis* (IEEE Computer Society, Los Alamitos, CA, USA, 2015), pp. 1–12, ISSN 2167-4337, `https://doi.ieeecomputersociety.org/10.1145/2807591.2807623`

[2] J. Cervantes Villanueva, G. Ganis, D. Konstantinov, G. Latyshev, P. Mato Vila, P. Mendez Lorenzo, R. Pacholek, I. Razumov, EPJ Web Conf. **214**, 05020 (2019)

[3] C. Green, J. Amundson, L. Garren, P. Gartung, M. Paterno, EPJ Web Conf. **214**, 05013 (2019)

[4] G.A. Stewart, B. Morgan, J.C. Villanueva, H.A. Willett, EPJ Web Conf. **245**, 05016 (2020)

[5] C. Green, J. Amundson, L. Garren, P. Gartung, E. Sexton-Kennedy, EPJ Web Conf. **245**, 05035 (2020)

[6] V. Volkl, P. Gartung, T. Lin, A. Sailer, J. Pöttgen, T. Madlener, B. Hegner, B. Viren, B. Coutourier, B. Morgan et al., *key4hep/key4hep-spack 2021-02-25a-opt* (2021), https://doi.org/10.5281/zenodo.4569110

[7] A. Valenzuela et al., *Cernvm-fs ephemeral writable shell demo* (2021), https://indico.cern.ch/event/885212/contributions/4214041/

[8] https://spack.readthedocs.io/en/latest/known_issues.html#variants-are-not-properly-forwarded-to-dependencies

[9] T. Gamblin et al., *Spack: A package manager for HPC systems* (2019), https://spack.io/files/spack-rd100-2019-final.pdf