Building the Key4hep Software Stack with Spack

Juan Miguel Carceller[1]*, *Wouter* Deconinck[2], *Thomas* Madlener[3], and *André* Sailer[1]

[1]CERN, Geneva, Switzerland
[2]University of Manitoba, Winnipeg, Manitoba, Canada
[3]Deutsches Elektronen-Synchrotron DESY, Germany

Abstract. The Key4hep software stack enables studies for future collider projects. It provides a full software suite for doing event generation, detector simulation as well as reconstruction and analysis. In the Key4hep stack, over 600 packages are built using the Spack package manager and deployed via the cvmfs software distribution system. In this contribution, we explain the current setup for building nightly builds and stable releases that are made every few months or as needed. These builds are made available to users, who have access to a full and consistent software stack via a simple setup script. Different operating systems and compilers are supported and some utilities are provided to make development on top of the Key4hep builds easier. Both the benefits of the community-driven approach followed in Spack and the issues found along the way are discussed.

1 Introduction

Key4hep is a software framework for future collider studies. By providing a common framework for different experiments, development efforts can be shared and reused. The Key4hep project aims to provide a complete software stack for future collider projects, including event generation, detector simulation, reconstruction, and analysis.

Figure 1 illustrates the three fundamental components in Key4hep: Gaudi [1], an event-processing framework; DD4hep [2, 3], which handles geometry descriptions essential for simulation, reconstruction, and analysis as well as detector simulation; and EDM4hep [4–7], the event data model implemented thanks to podio [8, 9], which also enables an efficient implementation as well as input and output capabilities. These components form an integrated system that can handle the complete data processing pipeline.

Several experiments participate in Key4hep: CEPC, CLIC, EIC [10], ILC, FCC, and the Muon collider. The source code for all components under direct development of the Key4hep project is hosted on GitHub[1]. Weekly open meetings are held to discuss ongoing developments and issues in Key4hep. Newcomers are always welcome to join the meetings or contribute to the developments.

Key4hep is supported by a software stack deployed on CVMFS [11]. The stack is built using Spack [12], a tool that provides both a repository of thousands of software package recipes and the code to build them. Spack's public library of recipes enables anyone to contribute, making it a time-saving and collaborative approach. Contributions to recipes benefit

*e-mail: j.m.carcell@cern.ch
[1]https://github.com/key4hep

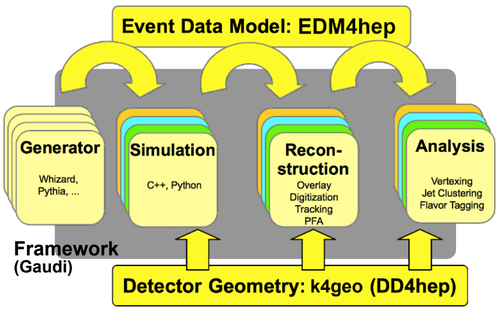

Figure 1. Main ingredients for the Key4hep project: geometry information, event data information, and an event-processing framework.

both the Key4hep project and the wider community, as others can improve or rely on the same recipes. Our complete Spack setup includes an additional repository, `key4hep-spack` [13], which contains recipes for packages not yet integrated into the main Spack library, as well as the configuration needed to build the Key4hep stack.

2 The Key4hep Stack

The Key4hep software stack contains approximately 600 packages built using Spack and deployed via CVMFS. It is self-consistent, with almost all package dependencies included in the stack[2]. To use the stack, users only need a host system running one of the supported operating systems with CVMFS installed. Setup requires just a single command, which adjusts the environment to use the Spack-installed packages. As a fundamental component of Key4hep, the stack significantly reduces entry barriers for new users by eliminating the need to compile software for many tasks.

2.1 The `key4hep-spack` repository

`key4hep-spack` is the repository that contains both the Spack configuration needed to build the Key4hep stack and the recipes for packages not yet available in the main Spack repository. The repository is publicly accessible and open to contributions from anyone, serving as the primary platform where users can report issues, request for new packages to be added to the stack, and engage with the project's development.

The build configuration is organized across several files based on build type. The `key4hep-common` folder contains the shared configuration used by all builds, while separate folders exist for specific build configurations, such as `key4hep-nightly-opt` for optimized nightly builds and `key4hep-release-dbg` for debug-mode release builds. A Python script merges these configuration files into a single file to prevent Spack from overwriting package settings, which would otherwise occur based on the inclusion order in Spack's main `config.yaml` file.

A meta-package called `key4hep-stack` serves as the entry point for the entire Key4hep stack, containing dependencies for all included packages. When building the stack, Spack only needs to be instructed to build this single meta-package, which pulls in all required packages like `EDM4hep` and their dependencies. `key4hep-stack` generates a shell script that

[2]With the exception of graphics and visualization dependencies, which are taken from the host system.

configures all necessary environment variables for using the Key4hep stack. Users access this configuration through a general setup script in a fixed location, rather than sourcing the generated script directly, since the location of the script produced by `key4hep-stack` changes with each build.

In addition to the package recipes and configuration for building with Spack, the `key4hep-spack` repository also hosts the setup scripts that users source from CVMFS. These scripts are picked up from a workflow and updated on CVMFS, see Section 2.5.

2.2 Building with Spack: Releases and Nightly builds

Two types of builds are available for Key4hep. The first is stable releases, hosted at `/cvmfs/sw.hsf.org`. These releases are produced every few months when needed, triggered by major changes or specific requirements such as preserving the software environment used for a particular dataset. Stable releases use only tagged versions of packages and occasionally involve a complete rebuild of all packages from scratch.

Nightly builds are available at `/cvmfs/sw-nightlies.hsf.org` and follow a two-step process. First, a Python script fetches the latest commits for a subset of packages and merges multiple yaml configuration files into a single file containing all the required variants and package versions. A complete build from scratch is then performed, using the latest commits for some packages and the latest tagged versions for others.

For the following weeks, daily incremental builds are created on top of this base, rebuilding only the packages that have changed along with their dependents (see Figure 2). This approach balances build time, maintainer effort, and stability: packages under active development by the Key4hep community are updated daily, while more fundamental ones, such as ROOT or Geant4, are only updated during full rebuilds. Unlike stable releases, nightly builds are periodically deleted from CVMFS, and not suitable for production use.

Debug builds have recently been introduced, as shown in Figure 3. These builds leverage the optimized builds as their upstream, reusing packages from them. Starting from a configurable point in the dependency chain (e.g., all packages after ROOT), the remaining packages are compiled in Debug mode.

The supported operating systems include AlmaLinux 9, Ubuntu 22.04, Ubuntu 24.04 and previously CentOS 7 (until June 2024). Optimized builds are performed in CMake's `Release` mode, while debug builds use the `Debug` mode. As of this writing, GCC 14 has been added on AlmaLinux 9 for nightly builds. For Ubuntu 22.04 and Ubuntu 24.04, builds are performed using the OS-provided compiler, GCC 11 and GCC 13, respectively.

To enable build reproduction, additional files are deployed to CVMFS with each build. These files include the commit hash of the Spack version used for the build, the commit hash of the `key4hep-spack`, and a list of cherry-picks applied to the current Spack version. This approach is essential when creating a new build from scratch, as certain fixes may require cherry-picked pull requests to Spack to ensure functionality. With these three files, the exact state of the Spack and `key4hep-spack` repositories used for the build can be fully restored if reproduction is needed in the future.

The nightlies and release builds are typically built on nodes with 16 threads. On a node with IceLake CPUs, a complete build of the 600 packages takes around 6 hours, while a nightly build on top of the previous one takes around 30 minutes.

2.3 User interaction

The interaction between the Key4hep stack and the user begins with setting up the Key4hep stack. With a simple command, the user can source a script to set up either the stable releases or the nightlies. The commands are the following:

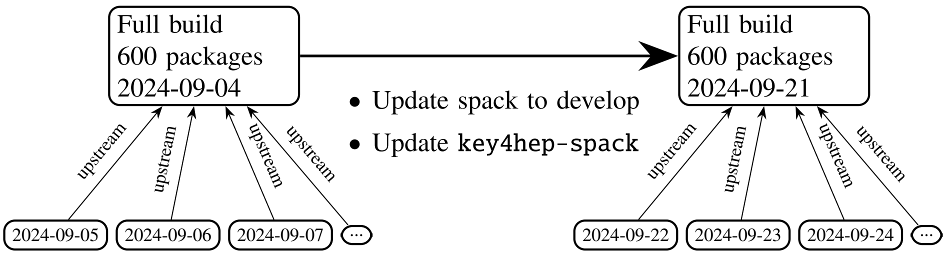

Figure 2. Schematic drawing of the nightly build process: A full build is completed initially, and then the subsequent nightly builds reuse this complete build over the following days. This reuse mechanism is known in Spack as *upstream*. After a few weeks, a new complete build is performed. Once this happens, the previous full build and all its dependent nightly builds that used it as upstream can be deleted, since the new set of nightly builds operates independently of the previous one.

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

or

```
source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
```

The setup script will automatically detect the operating system and configure the Key4hep stack accordingly. After sourcing, a message appears with important information, such as where to ask for help and how to reproduce the current environment in the future. By default, the scripts source the latest build. For example, for the nightly builds, the build completed on the current day will be used. Several command line parameters can be passed to the setup scripts: `-r` with the name of a release (typically a date) to set up the build that was completed that day, `-c` to select the compiler, and other parameters to list the existing releases and packages.

Some help is provided for development when using the Key4hep stack. The setup scripts define a function called `k4_local_repo` that, when called from a repository, will clear the environment from all the paths with the same name as the current repository and will add paths to point to the local installation. For most packages, this is enough to set up the environment so that the current package is used instead of the one that could be found in the Key4hep stack from CVMFS. While this is enough for building one package, if multiple packages need to be built then `k4_local_repo` needs to be called for each of them.

When users encounter issues, they typically report them in the `key4hep-spack` repository. Additionally, there is a pinned issue [3] containing known problems, allowing users to stay informed about what is not expected to work. There is also a mailing list available for any Key4hep-related topics, where, for example, new releases are anounced.

2.4 Testing, validation and continuous integration

After a build has been completed, a set of tests is run. These tests are primarily derived from issues that users have encountered in the past. The build is deployed only if every package is built successfully and all tests pass. Additionally, many of the Key4hep repositories have a workflow that runs the tests of each repository using the current nightly build, making use of the build deployed on CVMFS.

[3] https://github.com/key4hep/key4hep-spack/issues/502

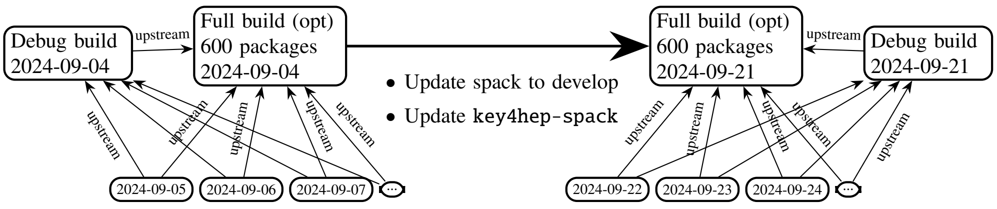

Figure 3. Schematic drawing of the nightly builds after debug builds have been added. Compared to Figure 2, the main difference is that each optimized build now has a corresponding debug build. While each optimized build still has only one upstream dependency, the debug builds have two upstream dependencies: the complete optimized build and the complete debug build.

Fast workflows were developed for continuous integration. Every time there is a pull request on many of the Key4hep repositories, a build is executed both on top of the latest stable release and the latest nightly. This build doesn't use Spack but simply builds the package (almost always a CMake package) using the stack, and serves also as a test that the package can be built by a user without using Spack. These workflows utilize a build cache produced by an earlier run, making the build process very quick. As a result, most of the time spent by these build workflows is typically the duration of the tests.

Additionally, there is another workflow to test how the current changes in a pull request affect dependent repositories. This workflow uses Spack to find out which packages depend on the current one, but the builds are performed going over each package and building with CMake. This workflow also allows setting up a coherent build with several pull requests in different repositories by parsing the text of the pull request. In practice, this feature is complex to use, and people who work on changes that affect many repositories typically make these builds themselves, so it is rarely used.

Several of these workflows are centralized: they live in a common repository, `key4hep-actions`, and, using custom tools, they are pushed to all the relevant Key4hep repositories, guaranteeing consistency and easing maintainability since they only need to be changed in one place. There are plans to centralize more workflows, like code formatting.

2.5 The k4-deploy repository

The `k4-deploy` repository is the final piece of the Key4hep stack. It contains the workflows that are used to build and deploy the Key4hep stack. All the workflows are defined in a `.gitlab-ci.yaml` file, and the repository is hosted in the CERN GitLab instance where jobs run on CERN-provided nodes. The build jobs run in Docker containers whose Dockerfiles can be found in another repository called `key4hep-images`. `key4hep-images` hosts the Dockerfiles for the build images, as well as images with CVMFS that can be used for continuous integration or development work.

Several workflows exist. In addition to building the stack (either as a complete build or reusing a previous one), there are also workflows to create and update the setup scripts in CVMFS, to update and push the Docker images used for the jobs, to update the setup scripts that users source on CVMFS, and to delete nightly builds from CVMFS.

3 Issues related to Spack

After having used Spack for providing builds of the Key4hep Software Stack for several years, several pain points remain, although the community is or has been working on ironing out some of these difficulties. One example is the concretization process, which consists on resolving all the dependencies and requirements for all the packages. Concretization requires several minutes for a stack of Key4hep's size. Debugging becomes time-consuming since many modifications, including simple tasks like adding a package for local development, require new concretization. However, since v0.22 in Spack, it is possible to include already concretized environments in the current one, therefore making the concretization process much faster as only a subset of the packages need to be solved. Another issue is that sometimes it is impossible to concretize because there are conflicts between versions and requirements, and the error messages do not always tell clearly where the issue is. This has, however, improved much with respect to the past, where no information was given when concretization failed.

The community-driven approach to sharing package recipes has proven beneficial, as updating to newer Spack versions automatically provides access to updated recipes. However, this collaborative model introduces potential complications. Changes made by other contributors may create incompatibilities that manifest specifically when building the Key4hep stack, yet remain undetected in other package combinations due to the impossibility of testing all possible version and variant combinations. A new pipeline has been added recently[4] in Spack to build common HEP software used by Key4hep, which will address this problem.

A limitation of Spack is that its core library and package recipes are stored together in the same repository. This prevents users from selectively updating package recipes without upgrading the Spack core library, even though recipe updates are the primary motivation for version upgrades in Key4hep. The Spack community is addressing this concern by discussing the separation of core library and package recipes for the upcoming version 1.0, to be released around summer 2025.

Previous investigations into utilizing build caches for Key4hep identified several issues, mostly related to relocating packages to a different location from their original installation [14]. Further exploration of build cache usage could potentially reduce build times for the Key4hep stack.

4 Summary

Key4hep continues to provide a software stack for future colliders, built with Spack. The stack has been in production for several years, offering stable releases and nightly builds. Recent improvements include the addition of debug builds, expanded support for more operating systems and compilers, and new workflows for continuous integration and testing. The community-driven approach adopted in Spack has been highly beneficial for the Key4hep project. Even though some issues remain, we have demonstrated that Spack can be used successfully to support complex software stacks.

Acknowledgements

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments (CERN-OPEN-2018-006). This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 101004761.

[4]https://github.com/spack/spack/pull/48412

References

[1] G. Barrand et al., GAUDI - A software architecture and framework for building HEP data processing applications, Comput. Phys. Commun. **140**, 45 (2001). 10.1016/S0010-4655(01)00254-5

[2] M. Frank, F. Gaede, M. Petric, A. Sailer, AIDASoft/DD4hep, `https://doi.org/10.5281/zenodo.592244`

[3] M. Frank, F. Gaede, C. Grefe, P. Mato, DD4hep: A Detector Description Toolkit for High Energy Physics Experiments, J. Phys. Conf. Ser. **513**, 022010 (2013). 10.1088/1742-6596/513/2/022010

[4] V. Volkl, T. Madlener, F. Gaede, A. Sailer, C. Helsens, P.F. Declara, G.A. Stewart, W. Deconinck, J. Smiesko, L. Forthomme et al., key4hep/EDM4hep, `https://doi.org/10.5281/zenodo.4785062`

[5] F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer, G.A. Stewart, V. Volkl, J. Wang, EDM4hep and podio - The event data model of the Key4hep project and its implementation, EPJ Web Conf. **251**, 03026 (2021). 10.1051/epjconf/202125103026

[6] F. Gaede, T. Madlener, P. Declara Fernandez, G. Ganis, B. Hegner, C. Helsens, A. Sailer, G. A. Stewart, V. Voelkl, EDM4hep - a common event data model for HEP experiments, PoS **ICHEP2022**, 1237 (2022). 10.22323/1.414.1237

[7] T. Madlener, EDM4hep - The common event data model for the Key4hep project, in *Proceedings of the CHEP 2024 conference (to be published)* (Krakow, Poland, 2024), `https://indico.cern.ch/event/1338689/contributions/6015945/`

[8] F. Gaede, B. Hegner, P. Mato, PODIO: An Event-Data-Model Toolkit for High Energy Physics Experiments, J. Phys. Conf. Ser. **898**, 072039 (2017). 10.1088/1742-6596/898/7/072039

[9] F. Gaede, B. Hegner, G.A. Stewart, PODIO: recent developments in the Plain Old Data EDM toolkit, EPJ Web Conf. **245**, 05024 (2020). 10.1051/epjconf/202024505024

[10] Lawrence, David, Eic software overview, EPJ Web of Conf. **295**, 03011 (2024). 10.1051/epjconf/202429503011

[11] J. Blomer, B. Bockelman, P. Buncic, B. Couturier, D.F. Dosaru, D. Dykstra, G. Ganis, M. Giffels, H. Nikola, N. Hazekamp et al., The CernVM File System: v2.7.5 (2020), `https://doi.org/10.5281/zenodo.4114078`

[12] 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 (Austin, Texas, USA, 2015), Supercomputing 2015 (SC'15), lLNL-CONF-669890, `https://github.com/spack/spack`

[13] V. Volkl, P. Gartung, T. Lin, T. Madlener, A. Sailer, J. Pöttgen, G. Ganis, B. Hegner, J. Wang, B. Viren et al., key4hep/key4hep-spack (2021), `https://doi.org/10.5281/zenodo.5654666`

[14] HSF Software Developer Tools & Packaging Working Group meeting, `https://indico.cern.ch/event/1301872` (2023)