Use Cases for Packaging and Deployment Tools

V1.0, 2018-03-21
10.5281/zenodo.3634722
HEP Software Foundation Packaging Group

This brief document summarises the major use cases for packaging and deployment systems in High Energy Physics. It is intended as a guide that can be used for determining objective criteria for different packaging and deployment systems.

It is also intended to provide background for the “required and desired” packaging features given in section 1.2 of the HSF Packaging WG Report (HSF-TN-2016-003).

The document is structured such that major numbered bullets express use cases, with sub-bullets expressing requirements derived from these that the group considers important.

Build

1. Experiments build software stacks for production use that must be completely reproducible. Experiment software librarians must be able to specify exactly how software is built and be assured of the same result.
    a. The packaging system should encapsulate all knowledge about a particular build of multiple software packages (versions, build options). This should include specifying build options for any implicit dependencies.
    b. The packaging system must be able to build each component of the system correctly by tracking inter-package dependencies and modifying the build time environment for each component appropriately.
2. Experiments need to build multiple ‘flavours’ that can target different production and development environments (e.g., CPU architecture or instruction sets) or utilise different compilers or build options (e.g., optimised and debug).
    a. The packaging system must be able to express this information in such a way that build options are not duplicated unnecessarily.
    b. The packaging system must be flexible about the location of a build, so that different build flavours can live on the same filesystem.
    c. Currently support for builds on major linux distributions is considered essential; OS X is desirable; Windows is optional.[1]
3. Software librarians must be able to control the use of toolchains and libraries from any external source for the build, including the underlying operating system.

[1] Containers can, to some extent, solve the problems of building or deploying onto OS X and Windows.

    a. The packaging system should be able to specify those cases where ‘external’ sources are used in an easy way.[2]
    b. The packaging system should also be able to express when this should not be done (this is particularly useful for developers’ private builds)
    c. Specifically, we consider that stacks may be built incrementally, with some dependencies being satisfied generically for many users (e.g., the LCG project).
        i. Packages themselves, and their internal build system, should provide interoperability information, e.g., pkg-config, which the package manager should capture in the build artefacts.
4. Experiments are constrained in their build resources, which need to be used efficiently. The time to build a release of the software should be optimal, within the constraints of build dependencies themselves.
    a. The package manager should be able to avoid duplication of packages for different compilers and architectures when there is no need to do so (e.g., header-only, data or other, "noarch" packages, executable-only and C-library-only packages).
    b. The ability to reuse binary pre-built packages (a.k.a.build artefacts) is required.
5. Librarians should be able to override an element of an ‘external’ element of the software.
    a. It is very desirable that the whole stack does not need to be rebuilt in this case;
    b. However, dependencies of the overridden component should be rebuilt (e.g., rebuilding a ‘twig’ of part of the stack, where then only a few leaf packages will need rebuilt, not the entire stack).
6. Builders of software need to be able to easily share knowledge of package build recipes (i.e., the specific details of how to build a package and its dependencies) and should be able to incorporate special build recipes on top of a more general community recipe book.
    a. The packaging system needs to support this flexible hierarchy of recipes.[3]
    b. The recipe formats should be backward compatible so that updated versions of the tool can build old recipes without needing a schema migration.
7. Librarians need to be able to store build artefacts separately from the build area, with all relevant information for an installation is contained in the build artefact itself.
    a. Support for different archive types may be desirable, depending on how installation will be done.

[2] Some HEP build systems have had an implicit use of ‘external’ software whenever a stack requirement was unmet. Others require quite explicit enabling of the use of ‘externals’. Note that there is always a danger of built software failing if an underlying library changes in an unexpected way, however the option to take this risk should be available to the librarian.
[3] This is usually implemented as a concept of repositories with a well defined hierarchy, which will be searched in order.

Deployment

1. Librarians need to be able to install a particular build of experiment software to different target paths, e.g., in CVMFS (e.g, `/cvmfs/my.expt/sw`), or a local install (`/usr/local`) as well as into containers.
    a. This implies *install time relocation*, which may be poorly supported by the underlying packages.
    b. Installation without root or other administrative privileges must be supported.
2. For installation to various OSs standard native package formats (RPM, deb, pkg, etc.) can be used.
    a. These should should be supported, e.g., through additional tools such as FPM.
3. Installation of different versions and/or flavours of built software must be supported with no interference between them.
    a. If two versions or flavours of the stack share artefacts it is highly desirable that the installations share these artefacts to avoid wasting space (and build time).
4. Removal of a whole release, or a part of a release, is necessary to control both the use of space and to enable deprecation of software. This should properly track dependencies in the case of sharing between installations.
    a. This is also the use case for testing new versions of software at scale (e.g., a new Geant4 patch) or for installing then removing nightly builds.
5. Update of certain elements of a release must be supported (i.e., patching in fixes, including rebuilt dependent packages).

Development

Developers require the use cases of the build system as listed above, however, this should recognise that this is not their full time activity and thus use of the build system should be accessible to physicist programmers.

1. Developers need to be compile against any deployed release, with the appropriate runtime environment setup.
    a. The ability to set up a runtime for a consistent subset of a build is required (so called ‘views’).
2. Developers need to override pieces of the runtime when developing a different version of a software package.
    a. This is necessary for a developer working on some new version of a package or on one small piece of an experiment’s software (one algorithm or module).
3. Developers need to setup the environment needed for development of any external or component of the complete stack.