Modularization of the LHCb software environment and preparation for heterogeneous resources

Marco Clemencic[1,*] and *Ben* Couturier[1,**]

[1]CERN EP-LBC

Abstract. LHCb software runs in very different computing environments: the trigger farm at CERN, on the LHC Computing Grid (LCG), on shared clusters or on software developer's desktops... The old model assumes the availability of CVMFS and relies on custom scripts (a.k.a `LbScripts`) to configure the environment to build and run the software. It lacks flexibility and does not allow, for example running in container and it is very difficult to extend them to configure and run on new environments. This paper describes the steps taken to modularize those tools to allow for easier development and deployment (as standard Python packages), but also added integration with container technology to better support non standard environments.

1 Introduction

To be able to build and run its software in the different supported environments, the LHCb Experiment[1] has always used a collection of shell and Python[2] scripts. Until recently, those scripts were packaged with custom tools and installed on a dedicated CVMFS[3] directory. This development and deployment model was not very flexible and required one single installation to support multiple operating systems and versions of Python. Moreover, all the tools were in a single project, and the need to deploy a change or a fix to a script used only for development, implied a new release of the whole project, including the tools needed for running the software on the LHC Computing Grid, that have to be validated by the LHCbDIRAC[4] team to ensure that no regression affecting production jobs on the Grid was introduced.

Using the time window of CERN LHC Long Shutdown 2, we decided to review our development and deployment model for the development and runtime tools.

2 Design principles

The new model was designed around two main principles

- separation of concerns
- use of standard tools

*e-mail: marco.clemencic@cern.ch
**e-mail: ben.couturier@cern.ch

2.1 Separation of concerns

Using one monolithic project for the various tools did not take into account the different constraints and release cycles, in particular for what concerns the tools needed to prepare the runtime environment for LHCb Physics Software applications and the tools used for day to day development. Runtime environment tools are used on the Grid and need to be stable, because a regression may cause large scale failures of productions jobs. On the other hand, development tools are not used in Grid jobs, but are used interactively by developers, so improvements and new features should be released and deployed as soon as possible for a better development experience.

With this scenario in mind, we decided to reorganize the tools in small packages, grouping together tools that share a purpose, or that have to be released at the same time. In practice this lead us to separate the layer that performs hardware detections (a.k.a LbPlatformUtils) used in the default LHCb environment as well as in LHCbDIRAC, the scripts to run LHCb applications (LbEnv) and the developer tools (LbDevTools).

2.2 Use of standard tools

LHCb can devote only a limited amount of personpower to the maintenance of the various tools, so any in house development of packaging and deployment tools cannot compete with standard tools backed by a large number of developers.

We also took into account that most of the tools we developed are in Python, and Python has a well defined standard for packaging and deployment[5]. The tools written in shell scripting languages can be either deployed via the same tools as Python scripts or, better, converted into Python, to profit from support tools like testing frameworks.

Moving from a custom deployment system for our scripts to standard Python packages has the added advantage that we can easily make use of the great variety of existing libraries available on the Python Package Index[6] (PyPI), so we can focus our personpower only on what we have to develop for LHCb specific needs.

Bootstrapping an installation of the LHCb environment is now very easy, as it relies on the ubiquitous Python deployment tools. Environments can be created using virtualenv[7] or Conda[8], depending on the use cases and preferences of users.

3 Refactoring of the tools

The first step we took to migrate from a monolithic project to small packages was to identify the concerns each package was meant to address. At the minimum we wanted to separate the runtime environment setup from the development tools, to decouple the part required for Grid jobs and that needed by the developers. We decided to factor out the platform detection and compatibility tools because they are not LHCb specific and they were inspired by the HSF proposal for a common platform detection algorithm[9]. We also decided to reduce the development tools to a strict minimum, and keep the tools related to release management in a dedicated package. The scripts used by the LHCb Nightly Build System were already grouped in a dedicated project, so for those we only had to convert the custom project into a proper Python package.

Once identified the packages we wanted to have (see Figure 1), we started moving the actual code. This has been an occasion to remove obsolete tools from our environment, so we imported in the new Python packages only the tools needed to address the use cases we wanted to support. Often, the move of scripts or modules from the old project to the new packages required a non trivial adaptation of the code, but it also was a good occasion to introduce unit and integration tests to be run during the development and at every upload to the repositories hosted on the CERN Gitlab instance[10], via Gitlab continuous integration feature[11].

Of course, with the initial release, some use cases were missed or forgotten, but we quickly addressed them, profiting from the independent release cycles of the new packages to avoid disruptions.

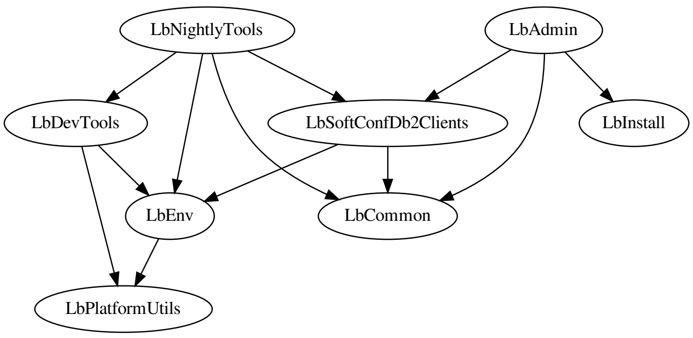

Figure 1. Example of interdependencies of some of the new LHCb Python packages. For example, `LbPlatformUtils` is used for host CPU architecture detection and compatibility support, `LbEnv` provides tools to set up the runtime environment of LHCb Physics Applications, `LbDevTools` contains scripts and resources to support building the software, and `LbAdmin` provides tools for maintenance of infrastructure and services.

4 Python 3 support

With the refactoring of the existing Python code, we decided to introduce Python 3 compatibility to get ready for the end of support for Python 2, scheduled for the 1st of January 2020.

There are several ways to address the migration of Python 2 code to Python 3, ranging from the one-way code conversion via the standard tool `2to3`[12], to third party tools aiming to produce code that is valid for both Python 2 and 3 (usually with the help of support modules).

For our tools we had to keep Python 2 compatibility, and we started by enabling the automatic `2to3` conversion at install time via the dedicated `setuptools`[13] flag. The main advantage of this is that, if tested, the compatibility with Python 3 does not require any work, so it can be introduced very quickly.

Of course, the automatic install time conversion of the code is not very practical in the long term, mostly because it requires that the developer keeps on writing for Python 2 and does not allow switching to a version of the code compatible only with Python 3. Because of this we scheduled a real migration to Python 3 for later in 2020.

As of the date of publishing, some of the packages have already been converted to support both Python 2 and Python 3 with the same code. The approaches adopted for this phase of the migration differ from one package to another, based on various considerations, like the list of acceptable dependencies. In particular, `LbPlatformUtils`, which is meant to be stand-alone and with minimal dependencies, has been initially converted with `2to3`, then the changes have been reviewed and tuned or fixed to work with Python 2. Another case is `LbDevTools`, where we used in some places the module `six`[14], because it has a number of 3rd party dependencies including (indirectly) `six`.

5 Integration with containerization technologies

The support for containerization technologies consists of two parts, detection and enabling. Dedicated functions in the package `LbPlatformUtils` check the presence and usability of tools used to start containers, while the runtime setup tools in `LbEnv` wrap the execution of the user requested command in a container supporting the version of the compiled code.

At the moment we only support the `Singularity`[15] containerization technology, which recently became a de-facto standard on Grid computing resources, but the detection and enabling concept can be applied to other technologies like `Docker`[16] or `Podman`[17].

The refactoring and modularization of the LHCb environment script was key to enable containerization support and its integration in the LHCb software tools and Grid middleware: the `LbPLatformUtils` Python package is installed in the LHCb default environment as well as in the LHCbDIRAC middleware used to run jobs on the grid.

6 Deployment

Although the new tools can be installed simply with a call to the standard Python package manager command `pip`, it is very practical for LHCb users to have a shared installation accessible from CERN's interactive login cluster (`lxplus`).

To create the shared installation we used the Python tool `virtualenv`[7], which allows the creation of contained and independent installations of collections of Python packages, such that users can easily switch from one to another.

The selection of preinstalled packages we offer to our users can vary with time, following the evolution of both LHCb specific and 3rd party packages, so to avoid disruptions to the developers workflows we deploy new versions in three stages:

- *unstable*: where we test new features of possibly disruptive changes
- *testing*: where we deploy automatically bug fixes or versions that have been validated in the *unstable* stage
- *stable*: after a validation period the versions in the *testing* stage get manually promoted to *stable*

By default, upon login to the `lxplus` service, the users have access to the packages in the *stable* stage, but they can choose a different set of versions via a configuration file in the home directory. Of course, power users and developers of the tools can also opt for a private installation.

Custom installations are also very easy: as long as Python and `pip` are available, setting up the environment to run or develop Python packages is very easy. The required tools are:

- *Python* >= 2.7 and *pip*: to install the base LHCb scripts
- *CVMFS*: Needed to run released software and to find the externals packages needed for development. This is however optional as a local installation is possible using the *lbinstall* tool, a Python package. Git can also be used to get a copy of the conditions database.
- *singularity* or *docker*: On unsupported operating systems (e.g. Debian linux), a containerization technology is required to ensure the compatibility with the LCG stack and installation of the compilers.

7 New architectures

As HEP experiments now have access to non-x86_64 HPC systems, featuring CPUs with architectures such as the ARM[18] or IBM Power[19] ones, LHCb faces the challenge of porting its software stack. While the challenges are many, the environment scripts were a problem from the start: the monolithic approach made it hard to separate the minimal functionality, and the use of LCG version of Python required a port of the LCG stack to even be able to start the effort. The new design makes this a lot easier and:

- separates concern by allowing the use of a local version of Python compiled for the system
- allows easily working on the platform detection features, located within the `LbPlatformUtils` Python package

This allows focusing the efforts on the real issues at stake i.e. the port of the physics applications.

8 Conclusions

The LHCb Core Software group successfully migrated the user and developer environment from the a monolithic project, deployed with custom tools, to a collection of interdependent packages, deployed using standard Python tools.

The development of the tools used by LHCb users and on Grid resources is now very flexible and agile. In particular, we managed to separate the components that have different release cycles and different requirements for stability.

The new modularity and the use of standard tools allow us to avoid that the Grid jobs rely on the shared installation of the tools, as it happened with the old system, instead they carry their own copy of a validated version of the only two Python packages they actually need.

Moreover, the use of standard tools allows us to develop our own functionalities on top of the great number of already existing 3rd party libraries.

References

[1] The LHCb Collaboration, A.A. Alves, L.M.A. Filho, A.F. Barbosa, I. Bediaga, G. Cernicchiaro, G. Guerrer, H.P. Lima, A.A. Machado, J. Magnin et al., *The LHCb Detector at the LHC*, Journal of Instrumentation **3**, S08005 (2008)

[2] *Python Programming Language*, [software], `https://www.python.org/`, [accessed 2020-03-12]

[3] 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: v2.7.0* (2019), language: eng, `https://zenodo.org/record/3608672`, [accessed 2020-03-12]

[4] The LHCb Collaboration, *LHCbDIRAC* (2018), [software], `https://zenodo.org/record/1451768`, [accessed 2020-03-12]

[5] *Python Packaging Authority — PyPA documentation*, [software], `https://www.pypa.io/en/latest/`, [accessed 2020-03-12]

[6] *PyPI · The Python Package Index*, [software], `https://pypi.org/`, [accessed 2020-03-12]

[7] *Virtualenv — virtualenv 20.0.10 documentation*, [software], `https://virtualenv.pypa.io/en/stable/`, [accessed 2020-03-12]

[8] *Conda — Conda documentation*, [software], `https://docs.conda.io/en/latest/`, [accessed 2020-03-12]

[9] B. Hegner, *HSF Platform Naming Conventions - A Proposal* (2018), publisher: Zenodo

[10] A.G. Alvarez, B. Aparicio Cotarelo, A. Lossent, T. Andersen, A. Trzcinska, D. Asbury, N. Hłimyr, H. Meinhard, *Extending software repository hosting to code review and testing*, J.Phys.Conf.Ser. **664**, 062018 (2015)

[11] *Continuous integration and delivery*, [software], `https://about.gitlab.com/ci-cd/`, [accessed 2020-03-12]

[12] *2to3 - Automated Python 2 to 3 code translation — Python 3.8.2 documentation*, [software], `https://docs.python.org/3/library/2to3.html`, [accessed 2020-03-12]

[13] *Welcome to Setuptools' documentation! — setuptools 45.3.0 documentation*, [software], `https://setuptools.readthedocs.io/en/latest/`, [accessed 2020-03-12]

[14] *Six: Python 2 and 3 Compatibility Library — six 1.13.0 documentation*, [software], `https://six.readthedocs.io/`, [accessed 2020-03-12]

[15] *Singularity*, [software], `https://sylabs.io/singularity/`, [accessed 2020-03-12]

[16] *Empowering App Development for Developers | Docker*, [software], `https://www.docker.com/`, [accessed 2020-03-12]

[17] *Podman*, [software], `https://podman.io/`, [accessed 2020-03-12]

[18] *The ARM architecture*, `https://www.arm.com/why-arm/architecture`, [accessed 2020-03-13]

[19] *The IBM Power architecture*, `https://www.ibm.com/it-infrastructure/power/accelerated-computing`, [accessed 2020-03-13]