LHCb Build and Deployment Infrastructure for run 2

This content has been downloaded from IOPscience. Please scroll down to see the full text.
(http://iopscience.iop.org/1742-6596/664/6/062008)

View the table of contents for this issue, or go to the journal homepage for more

Download details:

IP Address: 188.184.3.52
This content was downloaded on 25/02/2016 at 13:38

Please note that terms and conditions apply.
LHCb Build and Deployment Infrastructure for run 2

M.Clemencic, B.Couturier
E-mail: ben.couturier@cern.ch

Abstract. After the successful run 1 of the LHC, the LHCb Core software team has taken advantage of the long shutdown to consolidate and improve its build and deployment infrastructure. Several of the related projects have already been presented like the build system using Jenkins, as well as the LHCb Performance and Regression testing infrastructure. Some components are completely new, like the Software Configuration Database (using the Graph DB Neo4j), or the new packaging installation using RPM packages. Furthermore all those parts are integrated to allow easier and quicker releases of the LHCb Software stack, therefore reducing the risk of operational errors. Integration and Regression tests are also now easier to implement, allowing to improve further the software checks.

1. Introduction
Software is a critical part of particle physics experiments: millions of lines of code are needed to process and analyse the recorded data. Testing such a large code base is vital to ensure the quality of the results. The Large Hadron Collider beauty experiment (LHCb) has long run nightly builds and tests of its software but the home made system showed its limits and a project to review the whole infrastructure was started in 2013. This paper presents the architectural evolution put in place for the LHC run 2 for both the continuous integration and the release systems, as well as their planned evolution.

2. Continuous build system
Until 2013, LHCb was using a custom made nightly build system, based on the one described in [1]. While this allowed for highly customized features, the system lacked many that are common in continuous build systems used in industry. Furthermore, it was composed of interdependent scripts that could not be used outside the context of the nightly builds. It was therefore decided to:

- Redesign the build tools in a modular fashion, so that they could be used outside of the continuous builds (e.g. to recompile a complete stack with custom compilation options)
- Evaluate various continuous build systems used in industry, and choose the most appropriate to build the LHCb software.

2.1. The Jenkins build system
The evaluations led to the choice of the Jenkins[3] build system as presented in [2] at CHEP 2013. Deployed in 2013, experience showed that this was a good decision: Jenkins is a mature system that eased up the management of build nodes and helped make best use of the resources. It is however not without drawbacks and the new system suffers from the following flaws:
In order to support the LHCb nightly build structure, where the builds are grouped in slots (a set of projects built in a similar manner), projects, and platforms, a complex Jenkins job structure had to be put in place. While this works, monitoring the Jenkins jobs can be an issue. A dedicated monitoring page was therefore prepared to follow up on the status of the nightly build as presented in [4]. This complex structure could be simplified by the development of a custom plugin for LHCb that understands how to perform the builds.

The maintenance of the Jenkins system can be quite complex, and upgrading the plugins often implies a restart of the whole instance. This means the loss of the state of the Jenkins slave processes on the various build nodes, and can only be done when all nightly builds have finished.

In spite of these issues, Jenkins has proved extremely useful for the LHCb physics software continuous builds, and also allowed standard continuous integration jobs (e.g. for LHCbDirac [5], [6], or for the LHCb Masterclasses software[7]) to be run in the same instance.

2.2. Result reporting
The new system publishes all the results to a CouchDB database[8], instead of generating static HTML files with the results of the builds and tests as was done before.

The same CouchDB interface also gathers information for the monitoring page [4] which can be used to check the status and results of the continuous builds.

Using a database to store the results has many advantages: detailed management and archive is much easier, and it is possible to have automatic statistics on the results of the slots, with better alerts in case of failures. CouchDB proved more practical than a relational database to store this type of data.
2.3. Periodic tests and regression

Until 2014, the only tests runnable by the LHCb Nightly builds were unit tests, run on every slot/project/platform for every build. While this is necessary to ensure the integrity of the produced binaries, it does not allow long running integration tests as would be needed for regression or performance testing.

It was decided to add this capability to the test system, with the following features:

- The tests are driven by a cron-like XML file that starts the tests jobs in Jenkins. The structure of the file is adapted to the LHCb nightly builds structure and it is possible to select the builds to test by pattern (e.g. test Brunel in slot lhcb-head for x86_64-slc6-gcc*- would test the gcc46 and 48 builds, both in -opt and -dbg versions).
- The tests reuse the Jenkins jobs used by normal tests (as much as possible, an extra Jenkins job was added to read the configuration and start the tests) therefore maximizing code reuse.
- The test options to be run have to be committed either to the standard LHCb codebase, or to a specific software package, dedicated to the storage of the test options that can be run against multiple versions of the LHCb applications.
- The test results can be pushed either to CouchDB, as for normal tests, or to the LHCb Performance and Regression testing application (LHCbPR)[9].

The LHCbPR System[9] is another key component in the LHCb Software development, as it allows viewing and comparing the results of performance and regression tests run on the LHCb applications.

2.4. Distributing the nightly build binaries

Until 2014, the code built by the LHCb nightly build system was copied to the AFS shared file system at CERN. This would allow people connecting to CERN clusters, as well as other institutes using AFS[11] to directly run with the build prepared during the night.

While this is practical for development purposes, there is a need to run on hosts without AFS, and the best way to release the code on a large scale is to use the CERN VM File System (CVMFS) [15]. The release of the LHCb nightly builds on CVMFS was first done using an experimental backend from CERN Openlab[12] using hardware presenting the Amazon S3 interface [13]. This was a good opportunity to stress test the aforementioned hardware, as well as the S3 backend to the CVMFS main server (stratum-0). The result of this study was presented in [14]. All tests being very positive, a new CVMFS backend is now being prepared to finally deploy the LHCb nightly builds on CVMFS some time during 2015.

3. Software Release system

Until 2014, the release of LHCb software was done manually by the a so-called ”release shifter” in charge of gathering requests for release (from the JIRA [10] bug and request tracking system) and to build and deploy the software. This approach was very time consuming, error prone and was not taking advantage of the code available to build the nightly builds. It was therefore decided to review the whole procedure, limiting as much as possible manual interventions. Some components were however missing in the LHCb tools to perform such functionality: e.g. no database was available to specify which platform to build for which project. The Software Configuration Database (SoftConfDB) was therefore introduced for this purpose.

3.1. Software Configuration DB

The LHCb software is grouped in ”stacks”, i.e. projects based on the same dependencies of the same version of the Gaudi framework. This information is embedded in the dependencies of the projects (be they in CMT[17] or CMake[18] files) but it is not easily available for the
management tools unless there is a copy of the project of disk, or unless they query directly the subversion repository. In both cases, reverse queries of the style: which project depends on project X are hard to do unless you actually check the whole disk/SVN repository.

Several prototypes of databases to store all dependency information were investigated, and the Neo4j [19] Graph Database was chosen to implement the system. Its use within LHCb was pioneered in the Ariadne system, to track dependencies between condition data, as presented in [16], and it was an excellent match to store and exploit the dependency data between projects, which is itself a graph.

The Software Configuration Database (SoftConfDB) was implemented and deployed in production with the following functionality:

- Dependency tracking between projects: Before release, the LHCb applications have to be declared in the SoftConfDB. The command used for declaration traverses the project dependency tree, and inserts the relationships in the database, therefore allowing to quickly query the dependencies of a project, or list projects with specific dependencies.
- Platform specification: When adding new versions of the Gaudi framework, and its dependencies, the LHCb Software Librarian specifies the list of available platforms. Knowing the dependencies of a project, and having the list of platforms available for these dependencies, allows the SoftConfDB to know which platform should be built for a given project.
- Release Request: When a release of an application is ready for deployment, the release manager can insert the new pair (project, version) in the SoftConfDB and flag it for deployment. The Jenkins build system polls that flag and triggers the build job.

3.2. Release build

Building the released software is now performed by the Jenkins system. With its knowledge of dependencies, the Software Configuration Database groups the projects to be released in consistent stacks: it is therefore possible to build/release several interdependent projects in one go. This makes the releases a lot easier and allows to prepare new stacks much faster than
before. At the end of the build, the software packages are directly available on the LHCb central build machine, alongside the build reports (presented in CouchDB) with warnings and errors in case of problems (all unit tests are run after the build and it is up to the release manager to check the results before release).

A manual intervention by the release manager is however necessary to promote the build and perform the installation on AFS and CVMFS.

### 3.3. Release packaging

Packaging used to be another weak point of the LHCb Software chain: A custom python script was used to distribute and install tar files as a user. Functionality was added to the system over the years (e.g. archive checksum...) but much was still missing.

For this reason, a new system was developed based on the RPM Package Manager (RPM [20]), in order to leverage the functionality of this solid industry-standard tool. The constraints placed on the LHCb installation tools are such that:

- the packages have to be relocatable: the installation on the install should be doable on any random location of the disk, and specified at installation time.
- the packages have to be installable by somebody without superuser privileges.

The first idea was to prepare repositories for the YUM Package Manager (YUM [21]) containing the RPMs, and use the YUM client to preform the installation. Due to limitations in the tool, it was not possible to follow this approach and we could either used a patched version of YUM, or write a thin client with the functionality needed by LHCb, which was the approach chosen in order to maximize client functionality: the lbpkr[22] client was implemented.

This new approach has many advantages:

- The software repository follows the YUM structure, so standard YUM management tools can be used (creation of the SQLite database etc...)
- The lbpkr tool is not mandatory, users with full control on their hosts can install the software directly with YUM (in that case it is installed on /opt/lhcb).
- package management is made a lot easier, it is easy to add/remove versions of the software and check their integrity.

The packaging work was done in collaboration with the team managing the software shared between the various LHC experiments. Indeed, for this effort to be successful, the common software also had to be packaged as RPM and this is now the case.

### 4. Conclusion

The LHCb Core Software Team has modernized and modularized its build and release infrastructure for the LHC run 2. The approach taken was to use standard tools as much as possible, limiting custom code while retaining the functionality used within the LHCb collaboration. This already produced great time saving for the build and release of the software, as well as improved the quality of the released packages. The new system is however not without its flaws and the effort to upgrade the infrastructure is continuing.

### References

[17] CMT Configuration Management Tool (http://www.cmtsite.net/)
[18] CMake build system (http://www.cmake.org/)
[19] The Neo4j Graph Database (http://neo4j.com)
[20] The RPM Package Manager (http://www.rpm.org/)
[21] The Yum Package manager (http://yum.baseurl.org/)
[22] lhpkr LHCb software installation tool (https://github.com/lhcb-org/lhpkr)