Evolution of the LHCb Continuous Integration system

Robert Currie[1,*], *Rosen* Mataev[2,**], and *Marco* Clemencic[2,***]

[1]University of Edinburgh
[2]CERN

Abstract. The physics software stack of LHCb is based on Gaudi and is comprised of about 20 interdependent projects, managed across multiple GitLab repositories. At present, the continuous integration (CI) system used for regular building and testing of this software is implemented using Jenkins and runs on a cluster of about 300 cores.

LHCb CI pipelines are python-based and relatively modern with some degree of modularity, i.e. the separation of test jobs from build jobs. However, these still suffer from obsoleted design choices that prevent improvements to scalability and reporting. In particular, the resource use and speed have not been thoroughly optimized due to the predominant use of the system for nightly builds, where a feedback time of 8 hours is acceptable. We describe recent work on speeding up pipelines by aggressively splitting and parallelizing checkout, build and test jobs and caching their artifacts. The current state of automatic code quality integration, such as coverage reports, is shown.

This paper presents how feedback time from change (merge request) submission to build and test reports is reduced from "next day" to a few hours by dedicated on-demand pipelines. Custom GitLab integration allows easy triggering of pipelines, including linked changes to multiple projects, and provides immediate feedback as soon as ready. Reporting includes a comparison to tests on a unique stable reference build, dynamically chosen for every set of changes under testing. This work enables isolated testing of changes that integrates well into the development workflow, leaving nightly testing primarily for integration tests.

1 Introduction

Within LHCb the Real Time Analysis (RTA) project develops and maintains the real-time processing of LHCb's data for Run 3 and beyond. One of the specific goals of this project is to improve and maintain the quality assurance of code being used by the experiment.

A useful tool to contribute to this effort is the development of a Continuous Integration system for the LHCb experiment. Regularly and reliably testing the code being used improves the experience for everyone.

[*]e-mail: rcurrie@cern.ch
[**]e-mail: Rosen.Matev@cern.ch
[***]e-mail: Marco.Clemencic@cern.ch

2 LHCb Software Development

The LHCb physics software stack is composed of software with code spread out across multiple git repositories hosted on the CERN Gitlab [1] service. An overview of the framework, libraries and applications that make up the complete software stack is shown in Figure 1. Each project within LHCb is managed using a unique git repository, this provides a large degree of flexibility for developers working to support this software.

Building the software stack in such a modular way allows for targeted testing of code and features within each project. This comes at the cost of hiding the impact of changes on the rest of the codebase. An example of this hidden complexity is that an innocuous change within a framework project can have an adverse impact on a physics analysis using a derrived application.

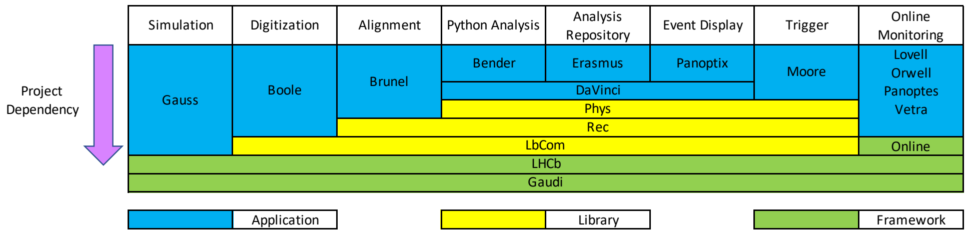


Figure 1. Overview of the LHCb Software Stack. Applications are top level projects which are written to perform computational physics tasks. Some of these applications make use of common libraries which are in turn built atop common framework projects.

To fully evaluate the impact of code change in a framework package for instance requires a full build of the software stack in order to run the all the tests in the high-level applications. With development happening on multiple projects at the same time it can prove difficult to determine the origins of problems when they are highlighted. A good development practice to quickly find and understand problems is to employ continuous integration testing.

3 Continuous Integration for LHCb

The LHCb experiment makes use of Continuous Integration to catch problems in development before new developments are merged. In this development model the master branch of projects is stable with all of the dedicated tests passing. This is achieved through regular building and testing of the software stack against different platforms and environments. The LHCb nightly build system [2],[3], provides a framework for automating the building and testing of the complete software stack in a reproducible way.

Previous efforts to integrate continuous testing into the LHCb development workflow have focussed on merging all pending code changes from across the software stack into the nightly builds for automated building and testing. This required users making extensive use of the nightly build system [2],[3]. Using this identifies many problems before they impact the experiment in production. However, it can make debugging problems more difficult as it's not immediately clear where conflicts may first have been introduced. The effect of this is that there can be a considerable lead time of 24+hours for developers on LHCb to potentially understand the impact that their developments have on the wider software stack.

3.1 LHCb Nightly Build System

The LHCb nightly build system makes use of a collection of approximately 300 cores which are managed by a centralised Jenkins [4] instance. Details of the implementation of this service have been presented previously [2],[3] with an overview of the Jenkins workflow presented in Figure 2. Within the LHCb Jenkins service CI pipelines are used to configure and build the LHCb software stack. The building of the software stack using these pipelines can be triggered by a nightly timer or through an external web-hook.

Nightly builds of the LHCb software stack are triggered by a timer and produce build artefacts which are used for further testing and development. Builds triggered by web-hooks are configured in a different way but make use of the same infrastructure and pipelines.

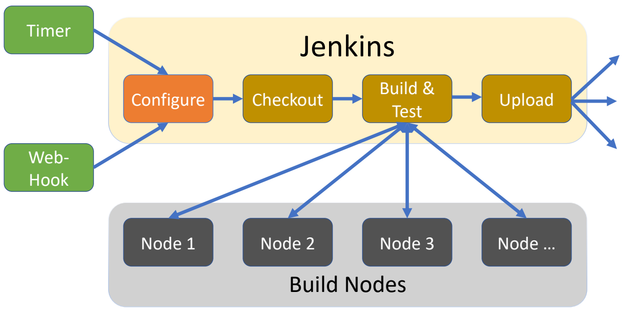


Figure 2. Overview of the LHCb Jenkins configuration

4 Isolated Testing for LHCb

Due to the LHCb software stack being composed of multiple projects as in Figure 1, isolated testing during development can prove to be a difficult task. The main difficulty when developing is understanding the impact of isolated work on the rest of the software stack whilst active development is still ongoing. Potential interactions between git commits on multiple projects is illustrated in Figure 3.

During development, a given set of code changes within a single LHCb software project is submitted to the CERN gitlab as a Merge Request (MR). In order to understand the impact of an individual MR the nightly tests within the software stack are run on builds from before and after the code changes have been merged. The results from these tests are then compared. Testing individual MRs this way provides isolated testing of a developers changes.

In order to perform isolated testing for MRs within the LHCb software a new tool has been developed. This tool allows for manual triggering of a web-hook to configure two automated builds of the LHCb software stack. One with and one without the changes in a given set of MRs to be tested. This allows developers to test their code without having to worry about external project dependencies. Access to this is provided through integration with the CERN Gitlab for LHCb projects.

An important consideration of this tool is how to handle external dependencies across multiple projects. In order to take this into account external projects are frozen at the last commit in time before a user forked their MR.

Using these new tools it is also possible to evaluate the impact of merging a MR onto the current HEAD of a project. This is done through compiling testing the software stack with and without a given MR merged into the codebase and comparing the test results.

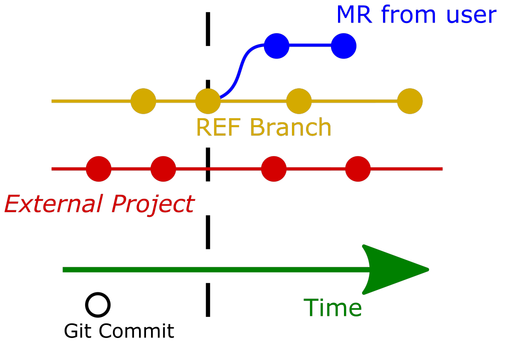


Figure 3. Evolution of git projects over time. Here a developer may want to commit a Merge Request (blue) to a Reference branch on a project (yellow) which depends on code from an external project (red). After the time of forking a MR it's not possible to guarantee that the new code has not broken as a result of independent work on an external project
If changes from an external project, or the reference branch (after development of an MR was started) is required, the best solution is to rebase an MR on a future commit of the Ref branch which will include these changes in a consistent way. It is possible to build a given LHCb project (Ref and MR) against a given git commit from an external project, but this is expected to be an uncommon use-case.

4.1 Developer Feedback

One of the key features of the LHCb nightly system is a web interface [5] developed to summarise and compare the results of software tests run on different builds. This interface allow developers to examine the impact of changes to the software stack from recent developments or changes in external dependencies.

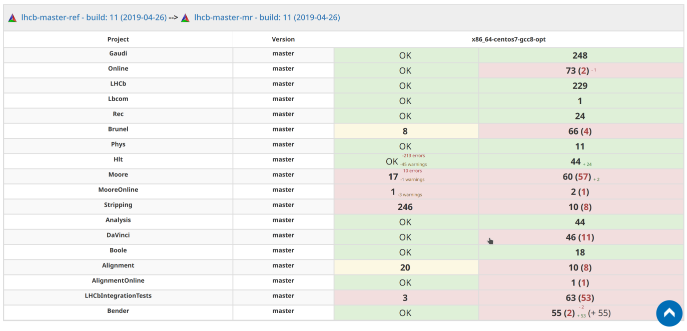
Once a developer has requested for a test build of the LHCb nightly stack they are notified through Gitlab with a link which directs them to the LHCb nightly web dashboard. This dashboard allows for the developer to compare the output from running the LHCb unit tests against multiple builds. Here changes are readily identified and are highlighted as needing to be understood before a change can be merged. An example of this comparison is shown in Figure 4.

Changes in high level tests may indicate desirable changes such as performance changes which are advantageous. However these should be understood by an expert before this is merged.

5 Summary

The LHCb nightly build system has proven to be highly flexible and able meet the increasing demands of the LHCb experiment during developments toward Run 3. Making use of this build system to perform automated testing reduces the feedback time to developers from over 24 hours to just a few hours. With additional caching of build artefacts within Jenkins the time delay in giving this feedback to a developer is expected to decrease.

The ability to perform automated isolated testing allows developers to focus less on compilation and testing of multiple projects and their dependencies and more on code development. Increased testing using tools will result in fewer bugs being introduced into production on the experiment. This will be especially important during Run 3 when code from physics projects is relied on for real-time physics selections.



lhcb-master-ref - build: 11 (2019-04-26) --> lhcb-master-mr - build: 11 (2019-04-26)

Project	Version	x86_64-centos7-gcc8-opt	
Gaudi	master	OK	**248**
Online	master	OK	**73** (2) -1
LHCb	master	OK	**229**
Lbcom	master	OK	**1**
Rec	master	OK	**24**
Brunel	master	**8**	**66** (4)
Phys	master	OK	**11**
Hlt	master	OK -213 errors -45 warnings	**44** + 24
Moore	master	**17** 10 errors -1 warnings	**60** (57) + 2
MooreOnline	master	**1** -3 warnings	**2** (1)
Stripping	master	**246**	**10** (8)
Analysis	master	OK	**44**
DaVinci	master	OK	**46** (11)
Boole	master	OK	**18**
Alignment	master	**20**	**10** (8)
AlignmentOnline	master	OK	**1** (1)
LHCbIntegrationTests	master	**3**	**63** (53)
Bender	master	OK	**55** (2) -2 +53 (+ 55)

Figure 4. An example of feedback to a developer who has requested a test build to be performed on a given MR. Red on the right column indicates changes in the test results of high level tests across different projects. These changes should be understood by a project maintainer before a MR is merged. Numbers in this table indicate how many tests have passed (in bold) or failed (in brackets) after building the head of a MR and running the nightly tests.
'OK' (green background) indicates that the tests run against this MR are the same as when they were run on the Ref branch. The change in number of tests failing due to isolated changes within an MR are summarised with additional numbers in this table (red and yellow backgrounds).

References

[1] https://gitlab.com

[2] Kruzelecki, Karol and Roiser, Stefan and Degaudenzi, Hubert, Journal of Physics: Conference Series **219**, 042042 (2010)

[3] Clemencic, Marco and Couturier, Ben, Journal of Physics: Conference Series **513**, 052007 (2014)

[4] https://jenkins.io

[5] Clemencic, Marco and Couturier, Ben, and Kyriazi, Sofia, Journal of Physics: Conference Series **664**, 062025 (2015)