A New Nightly Build System for LHCb

M Clemencic and B Couturier
CERN, CH-1211 Genève 23, Switzerland
E-mail: marco.clemencic@cern.ch

Abstract. The nightly build system used so far by LHCb has been implemented as an extension of the system developed by CERN PH/SFT group (as presented at CHEP2010). Although this version has been working for many years, it has several limitations in terms of extensibility, management and ease of use, so that it was decided to develop a new version based on a continuous integration system.

In this paper we describe a new implementation of the LHCb Nightly Build System based on the open source continuous integration system Jenkins and report on the experience of configuring a complex build workflow in Jenkins.

1. Introduction
The LHCb experiment has been using the Nightly Build System based on the LCG Application Area one for several years[1]. The system works, but it has limitations that make it very difficult to extend and manage. We therefore decided to investigate the possibility of a new implementation of the Nightly Build System, based on the the open source continuous integration system Jenkins[2].

The outcome of the investigation is the system described in this paper, now in production.

1.1. LHCb Software in the Nightly Builds
LHCb Software is divided in projects (releasable entities) with dependencies between them, meaning that a project uses libraries produced in another projects. So, changes in a software project do not only affect the project itself, but also the projects depending on it. To validate the software, then, we need to build a consistent stack (dependency chain) of projects, which, in the Nightly Builds terminology, we call a slot. The slot definition includes the versions of the external libraries used to build the software too, so we can validate the same software stack in different contexts.

Our software is built on a selection of architectures, Linux OS flavors, compilers and flags. Each selected set is called a platform. Having several platforms is part of our Quality Assurance plan and increases portability.

To validate the changes of the software in the widest possible range of cases, in our Nightly Builds we build several slots (with different configurations) on several platforms. In some cases, we also need to prepare ad-hoc slot configurations for special temporary validation tests.

2. Requirements
From past experience we collected a minimal set of functionalities that the new Nightly Build System had to provide. Some of the functionalities were already present in the old system, such
as:

- build and test several slots on several platforms
- allow easy configuration of the content of the slots
- separate the builds of different platforms
- allow customized checkouts (i.e. non default versions of the packages)
- run the tests of a project while building the following one on the stack
- allow configuration of the parsing of the build logs (ignore some warnings and errors)
- distribute efficiently the load on a pool of build machines
- provide a dashboard updated incrementally with accessible links to problems and an overview on all the slots

but they had to be improved and with a simpler and more maintainable implementation.

In addition we wanted to have some long awaited new features:

- monitoring of the status of the builds
- allow to restart builds at different levels: everything, one slot, one platform of one slot
- produce archives of the checkout and of the builds
- allow easy creation of new slots (both production and testing)
- provide a manageable procedure for the development of the system itself

3. Design

The new Nightly Build System is divided in three main parts to address the different concerns: the configuration of Jenkins jobs (for the coordination and distribution of the tasks), the core tools (for the actual checkout and build tasks), the dashboard (summarized presentation of the results of the builds).

Although Jenkins allows arbitrary complex scripts in the build steps, it is suggested in the documentation to keep the build steps simple, wrapping the complexity of the build in tools that are distributed and developed together with the project.

In our case, the scripts used for the heavy-lifting part of the builds are hosted on a dedicated GIT[3] repository, instead of living within the software projects, because they are generic and apply to whole software stacks, i.e. interdependent sets of projects. The language of choice for the implementation of the scripts is Python[4] for many reasons, including the fact that Python is the official scripting language in LHCb and that the old system was already implemented in Python.

An important difference in the design of the new system with respect to the old one is that the various actions required in the nightly builds (checkout, build, test, etc.) are performed by dedicated independent scripts instead of being phases of a monolithic script. Thus it is possible to develop and test a single action without having to restart the whole process from scratch. Moreover, the core tools are meant to work and produce files in any directory, instead of using fixed locations as in the old system, further simplifying the development. Of course, whenever feasible, common code is factored out and shared between all the scripts.

The configuration of Jenkins required the installation of several plug-ins on top of a vanilla installation of the application. The jobs, in Jenkins terms, configured are of two main categories: jobs representing the nightly build slots and generic jobs for the individual steps.

The dashboard was implemented as a web application plus a database, both hosted on an Apache CouchDB server[5]. To simplify the development of the web application, we used the framework CouchApp[6] via the command line tool erica[7]. The code for the web application is kept in the same GIT repository as the scripts.
4. Implementation

4.1. Core Tools

In the following sections we describe in some details the main components of the Core Tools scripts.

4.1.1. Configuration

The old nightly build system uses an XML-based configuration describing in one file the slots to be built, their content, the platforms they should be built for etc. During the prototyping phase, we reviewed the layout and the content of the old configuration file looking for opportunities for simplification.

We decided to start from a minimal configuration file describing a single slot, using the JSON format (semi-structured) instead of XML (structured). The format and the internal representation of the configuration are still under discussion and may change in the future.

The new configuration format contains the same information as the old one, with the addition of that needed to enable the new features. For example, in the new configuration file it is possible to specify the Python function to be used for the checkout of a software project, that can be a predefined one or a user-defined one, while it was not possible in the old configuration.

To allow for a smooth transition from the old system to the new one and for testing the new system in parallel with the old one, we introduced a simple translation layer that allows the use of the old configuration in the new system.

In order to simplify the development and testing in the new system, we decided that the configuration file must be passed as command line argument to each nightly build script.

4.1.2. Checkout

Each software project specified in the slot configuration is checked out in a temporary directory from the code repository using a customizable Python function (each project can use a different checkout method). The default function is functionally equivalent to the code present in the old system, with some improvements and simplifications. Other methods are provided, like checkout from GIT or extraction from a source tar file (used for testing purposes), but others can easily by defined in user provided Python modules without changing the code of the checkout script.

To ensure self consistency of the slot, we edit the project configurations and record the changes in a patch file (not done with the old system).

The source files are then packaged, project by project, in a dedicated directory (artifacts) for later use.

4.1.3. Build and Test

Building and testing actions are bundled in one script because they are tightly connected: the tests of a project in the slot are run while other projects are built. The behavior of the build script can be tuned with documented command line options to allow easy and effective testing of the build system.

The source packages prepared during the checkout step are unpacked in the directory builds, cleaned beforehand. Then we build each project (optionally with a number of parallel processes), one after the other in order of dependencies (declared in the configuration). Once a project is built, we first scan the build log to extract the compilation warnings and errors reported during the build, then we pack the build products in a new archive file in the artifacts directory. At this point we start a subprocess in background to run the tests of the project while the next project in the slot is being built.

During the build of the slot, the main script sends information about the partial results of the various projects to the dashboard database and, after all the builds are completed, it waits until all the spawned subprocesses complete before exiting.
4.1.4. Stand-alone Use  We developed the Nightly Build scripts as stand-alone tools to speed up the development of the new system and to extend its usability. A casual user can call them by hand to reproduce the behavior of the nightly builds, either for testing or to work at the same time on several projects. In this way the Nightly Build System becomes a relatively thin layer on top of the functionalities provided by the scripts.

4.2. Jenkins Configuration

Jenkins is a very flexible and extensible application used to manage automated build and test jobs. Its features span from regular, on demand or commit triggered builds to complex build work-flows or just monitoring of tasks. Its plugin system allows to add functionalities like integration with different tools (e.g. Coverity Static Analysis[8] or JIRA[9]) and more reports from the builds.

None of the existing Jenkins plugins could fit our use case, but we managed anyway to find a combination of existing plugins that allowed us to deploy a feature complete system.

With the number of slots, projects and platforms we build and test every night it would be impossible to use a single machine so we use a pool of eight machines as Jenkins slaves, for a total of 96 logical CPUs, and we build using distcc[10,11], which extends our build capacity to the order of a few hundreds CPUs. Usually Jenkins machines are configured with one executor per CPU, but we use a different ration because our build jobs use on average ∼2 CPUs. On the master node we need to run many jobs that do not use CPU, so we configured it with ∼200 executors.

Another reason why we use a pool of slaves is that we need to build on different operating systems, each of them able to build a set of platforms. To control on which machines a build job can be executed we flag the slaves with strings identifying the operating system.

The jobs we configured for the nightly builds are divided in two categories: the workers and the slots. For simpler monitoring, the jobs of the two categories are grouped in two views.

4.2.1. The Workers  The workers are generic jobs that take care of the actual checkouts and builds of the slots. All the jobs that use the scripts described in 4.1 are configured to retrieve the latest version of the scripts from the GIT repository when they are started.

A high level orchestrator job is used to coordinate and trigger the execution of lower level jobs. We use parameters of the job to choose the slot to build, to give a numeric id to the build and, optionally, to override the default list of platforms to build. These parameters are passed to the triggered jobs.

The first job the orchestrator triggers is the checkout job, which retrieves the latest version of the slot configuration and calls the checkout script. The archive files produced are recorded in Jenkins as artifacts and stored on a dedicated directory on a server used to keep them and serve them via HTTP. The configuration of the slot is also stored in the database of the dashboard.

Once the checkout job completes, if successful, the orchestrator job triggers a builder job which receives the list of platforms to build and spawns one subjob per platform, that will be scheduled on a compatible machine. These subjobs retrieve the artifacts produced by the checkout job (using the slot name and the build id to identify the correct ones) and call the build scripts.

4.2.2. The Slots  For convenience, each configured slot has a corresponding job configured in Jenkins. These simple jobs, when started, trigger the generic orchestrator job passing their name as slot name and the build number as numeric build id to it. In this way we can trigger the build of a slot with a simple click or regularly with the job scheduler provided by Jenkins (the most common case).
4.3. Dashboard

It is extremely important for our developers and project managers to have a quick overview on the status of the nightly builds, and this view should be tunable so that each user can see only the information relevant to her.

To reduce the design time and to reduce the impact on users, we modeled the new dashboard after the layout and the functionalities of the old one. The main change is that the data used to populate the tables in the dashboard is stored in a database instead of files.

4.3.1. CouchDB Database

For each build of each slot we store in a CouchDB database the configuration used for the build and, for each platform, one entry when the build job is started, the build and test summaries for each project, and a final entry to flag the completion of the platform. All the documents are processed through map functions (views) to select and format the relevant data, indexing them with appropriate keys.

4.3.2. Web Page

The web page uses heavily AJAX techniques and the JavaScript libraries jQuery[12] and jQuery-UI[13].

For a given day, the main script of the web application retrieves the list of slots that have been started (asynchronous HTTP request) and prepares the empty summary tables. The details of each platform in each slot are retrieved asynchronously and used to fill the spaces in the tables with the number of warnings and errors, and with links to the full logs of builds and tests.

5. Conclusions and Outlook

The new Nightly Build System we developed has been used in production for several months demonstrating a good stability and reliability.

All the features of the old system have been implemented in the new one, together with some additional required features. Now the development is concentrated on improving the usability and extending the features even more, both on the builds and the database sides.

One of the most important advantages of using Jenkins for the scheduling of Nightly Build System is that the same Jenkins instance can be used for the continuous integration of other projects too, for example it is used by the DIRAC and LHCbDIRAC projects, and the Nightly Build scripts themselves are tested in Jenkins after every commit.

Using a database to record the status and results of the builds allowed for a clean and flexible design of the dashboard web page. Having CouchDB as database back end makes it very easy to evolve the database “schema” beyond the original design, and the synchronizations feature of CouchDB made it extremely easy to develop and test changes before pushing them into production.

References

[1] Kruzelecki K, Roiser S and Degaudenzi H 2010 J.Phys.Conf.Ser. 219 042042
[2] Jenkins CI URL http://jenkins-ci.org/
[3] GIT Distributed Version Control System URL http://git-scm.com/
[4] Python Programming Language – Official Website URL http://python.org/
[5] Apache CouchDB – a database for the web URL http://couchdb.apache.org/
[6] CouchApp – a framework for web applications hosted on CouchDB URL http://couchapp.org/
[7] Erica – a tool to manage CouchDB applications URL http://github.com/benoitc/erica
[8] Coverity Home Page URL http://www.coverity.com/
[9] Atlassian JIRA, Issue & Project Tracking Software URL https://www.atlassian.com/software/jira
[10] distcc: a fast, free distributed C/C++ compiler URL http://code.google.com/p/distcc/
[11] CERN distcc Pilot Service URL https://twiki.cern.ch/twiki/bin/view/LinuxSupport/DistccPilotService
[12] jQuery – a fast, small, and feature-rich JavaScript library URL http://jquery.com
[13] jQuery-UI – an extension of jQuery for User Interfaces URL http://jqueryui.com