A gateway between GitLab CI and DIRAC

Chris Burr[1,*] and *Ben* Couturier[1,**]

[1]CERN

Abstract. GitLab's Continuous Integration has proven to be an efficient tool to manage the lifecycle of experimental software. This has sparked interest in uses that exceed simple unit tests, and therefore require more resources, such as production data configuration and physics data analysis. The default GitLab CI runner software is not appropriate for such tasks, and we show that it is possible to use the GitLab API and modern container orchestration technologies to build a custom CI runner that integrates with DIRAC, the middleware used by the LHCb experiment to run its job on the Worldwide LHC Computing Grid. This system allows for excellent utilisation of computing resources while also providing additional flexibility for defining jobs and providing authentication.

1 Introduction

The LHCb experiment uses GitLab [1] to manage its physics software lifecycle. The first use of this continuous integration system (GitLab CI) was to run unit tests. Very quickly it became obvious that it could also be extended to validate the configuration for data production jobs, or even to check and run user data analysis scripts. While standard GitLab CI runners are appropriate to run unit tests or small test jobs, data analysis production jobs validation is CPU intensive and exceeds the capacities of standard shared runners with run-times varying from a few minutes to tens of hours. We therefore decided to develop a GitLab CI gateway that would fulfil all use cases, and allow for more flexible use of resources (e.g. by running the jobs on the Worldwide LHC Computing Grid [2] instead of dedicated hosts).

1.1 Motivations

As the use of GitLab CI was extended to Physics data analysis and to validate configurations for data production, the standard GitLab runners software forced LHCb to dedicate resources to this use case. We observed the following drawbacks:

- Managing similar CI configurations across multiple projects is error prone.
- Dedicated runners are idle most of the time, leading to the under-use of resources.
- Injecting credentials for merge requests (e.g. user's Grid proxy) in the GitLab jobs in a secure way is difficult.

*e-mail: christopher.burr@cern.ch
**e-mail: ben.couturier@cern.ch

2 Base project and infrastructure

GitLab CI uses a simple REST interface to allow runners to register, request jobs and return results. While this is not publicly documented, it can be easily reverse engineered. We therefore decided to develop a custom GitLab runner[3] to solve the issues encountered with the standard software. As the LHCb middleware, DIRAC [4] (one of the main external packages that should be integrated), is developed in Python [5], it was natural to develop this new tool as a Python package.

A scalable way to manage the GitLab CI jobs was required, both at the software level and at the infrastructure level. This needed to be capable of managing tasks with durations varying from a minute up to many days. We decided to use the Celery [6] distributed task management system, a Python tool that requires external software to manage the queues of requests. Celery supports many messaging backends and we chose to deploy the RabbitMQ [7] message queue system. The Red Hat Openshift [8] container platform was chosen in order to have a scalable infrastructure to run the processes required by Celery.

A web frontend is also needed for users to register their GitLab project with the custom GitLab runner. For this purpose, a Flask [9] application was developed and deployed. Figure 1 shows the overall architecture, the Celery processes being split in two parts: the Celery Beat is the scheduling engine that allows triggering periodic tasks, and the Workers perform the tasks of interacting with DIRAC and GitLab.

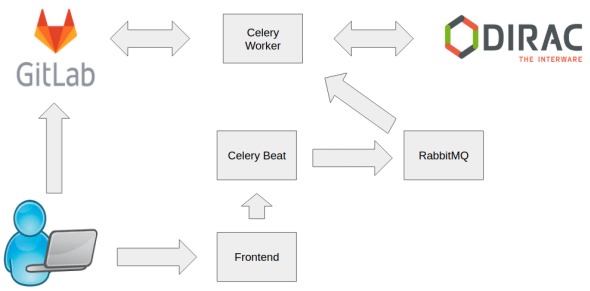

Figure 1: Overall system architecture

2.1 Runner registration

The first step to run GitLab CI jobs within the Gateway is to register it with the system. This is a crucial step as the owner of a GitLab repository needs to be able to delegate his/her credentials to the Gateway, in order for the service to request jobs.

Figure 2 shows the registration process. The user provided secret is registered in the gateway using the web frontend. This is used to obtain a dedicated runner token that is stored in a dedicated database. The frontend then triggers the polling of GitLab for jobs related to this project by the Celery Beat.

2.2 GitLab CI job processing

Figure 3 details the interactions between components, in the case of jobs run on the WLCG using LHCbDirac (i.e. the LHCb instance of DIRAC). The use of Celery on OpenShift, with

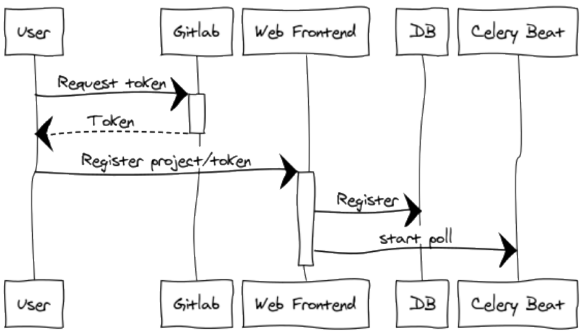

Figure 2: GitLab runner project registration

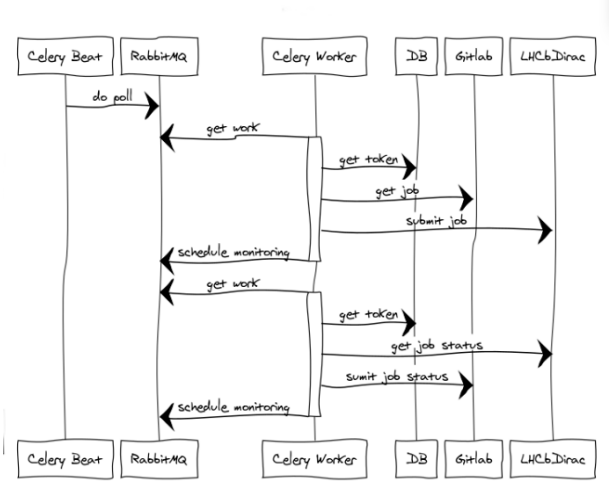

Figure 3: GitLab CI job processing

RabbitMQ to manage the queues, makes the system resilient to being quickly restarted and updated without the need to drain long running tasks out of the system.

GitLab presents a REST [10] application programming interface [11] that can be used (with the appropriate credentials) to query the jobs to be run for a specific project, and to publish back the results. Updates have to be provided on a regular basis or GitLab considers the runner as dead, and therefore restarts the job. This API is the basis for the interaction between the Gateway and the GitLab system.

The Gateway installs and runs python packages. It is therefore possible to install using standard Python tools (e.g. pip [12]), and the use of the Python entry points package mechanism [13] allows to decouple the Gateway from the code to be run.

The use of OpenShift (and of the underlying Kubernetes [14] container orchestrator) allows to scale the project with the number of repositories registered: it is possible to increase the size of the group of processes running each task (so-called "pods" in Kubernetes) to adapt to the needs of the application.

2.3 Integration with DIRAC

The GitLab CI gateway is generic, and can be integrated with many systems. The integration with the LHCb instance of DIRAC is however a crucial use case for the experiment, as it is the way to run jobs on the WLCG.

Running GitLab CI jobs on the Grid is possible because the LHCb software is deployed on the CernVM file system [15] (CVMFS) which is easy to access from GitLab jobs but also from all WLCG nodes. One issue requires caution, it is the one of user authentication and authorisation.

Trust is required between the GitLab system, the team running the GitLab gateway and the Grid team. Indeed, the Gateway has to trust the identity of the user submitting the job fetched from GitLab itself. Special attention has to be taken to the security of the system, as running jobs from any GitLab merge request (if this is possible publicly) would potentially allow any user to run code from their branch. Two options are possible on the LHCbDirac side: either running the jobs as a specific user or run the jobs on behalf of the user triggering the CI job. The latter implies a mapping between the GitLab accounts and the Grid accounts as well as the right to impersonate Grid users to start jobs on their behalf. This has not been implemented yet as it implies discussion with the involved parties in order to limit and audit the code.

2.4 GitLab CI gateway prototype

The current prototype allows testing running test productions on the Grid using LHCbDirac. It allows a dynamic number of jobs to be launched, one per dataset processed whereas this is not possible with the standard GitLab runner. This has been used to dynamically spawn many hundreds of tasks from a single CI job. The status summary is reported to GitLab CI and additional logs and output for each production are accessible via the web frontend as shown in figures 4, 5 and 6.

Analysis production for pipeline #1053143

BsToJpsiPhi

Commit: 2c1d786e
Message: Dump CI variables
Colour code: Running Success Failed Unknown

Raw YAML Show

Rendered YAML Show

Click on a job for more information

Job Name	Test job statistics			Log messages			Estimated total	
	Events	Runtime	Size	Warn	Error	Fatal	Size	Kept
MC_2017_MagDown_ALLSTREAMS_Bd2JpsiKstar_no_str_g_strip	2640	0:03:14	93.4 MB	3	0	0	170.9 GB	False
MC_2017_MagDown_ALLSTREAMS_Bd2JpsiKstar_no_str_g_tuple	331	0:01:35	3.0 MB	84	0	0	5.5 GB	True
MC_2017_MagDown_ALLSTREAMS_Bu2JpsiKplus_no_str_e_strip	2600	0:05:41	208.6 MB	3	0	0	786.3 GB	False
MC_2017_MagDown_ALLSTREAMS_Bu2JpsiKplus_no_str_e_tuple	824	0:02:59	4.4 MB	83	1	0	16.8 GB	True
MC_2017_MagUp_ALLSTREAMS_Bd2JpsiKstar_no_str_g_strip	2393	0:03:01	83.0 MB	3	0	0	169.6 GB	False
MC_2017_MagUp_ALLSTREAMS_Bd2JpsiKstar_no_str_g_tuple	309	0:01:39	2.8 MB	83	0	0	5.7 GB	True
MC_2017_MagUp_ALLSTREAMS_Bu2JpsiKplus_no_str_e_strip	2686	0:05:15	209.0 MB	2	0	0	779.5 GB	False
MC_2017_MagUp_ALLSTREAMS_Bu2JpsiKplus_no_str_e_tuple	832	0:02:33	4.5 MB	86	1	0	16.6 GB	True

Figure 4: Example production configuration validation job

Analysis production job for BsToJpsiPhi

MC_2017_MagDown_ALLSTREAMS_Bd2JpsiKstar_no_str_g_strip

Status	Commit	Requested	Processed	Runtime	Kept
Success	2c1d786e	-1 events	2640 events	0:03:14	False

Input

Name	Size	Total
/lhcb/MC/2017/LDST/00091367/0000/00091367_00001603_5.ldst	768.9 MB	1.4 TB

Output

Name	Size	Total (estimated)
00012345_00006789_1.AllStreams.anaprod_bstojpsiphi.ldst	93.4 MB	170.9 GB

Reproduce locally Show

Job log Show

DIRAC log Show

Figure 5: Job summary in the web frontend

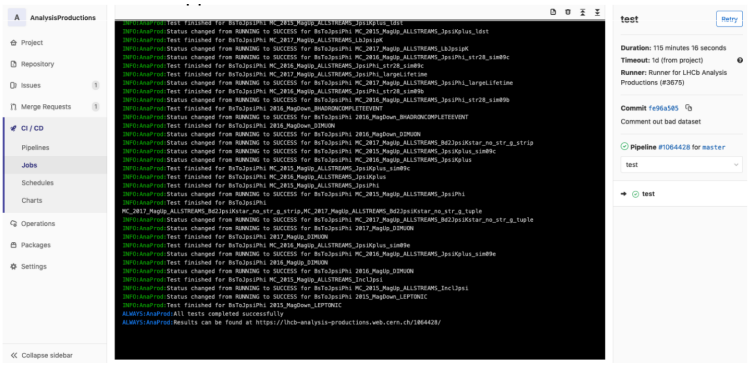

Figure 6: Job log as visible in GitLab

3 Use cases for the Gateway

3.1 Automating ntuple production using LHCbDirac

In many cases, LHCb Data Analyses start with the extraction of the relevant quantities from the LHCb dataset. This stage is performed manually by the analysts, who have to monitor its progress and make sure it concludes successfully. Automation of this task is possible but requires strong quality checks to avoid wasting Grid resources. This is where GitLab CI can play a role, and a prototype[16] was developed that functions in the following manner:

- The `LbAnalysisProductions.ci.pull_job(runner)`, pull jobs from a specific GitLab repository(`lhcb-datapkg/AnalysisProductions`), in which each folder corresponds to an analysis (e.g. `Charm/d2hll_Run2`). It then
  - finds folders which have changed, these are “productions”
  - iterates those folders to check which testing “steps” they define
  - generates signed URLs so jobs can directly upload their output to S3

  - Return $N_{\text{productions}} \cdot N_{\text{steps}}$ monitoring tasks

- Use `LbAnalysisProductions.ci.check_status` to check the status in LHCbDirac and update the log in GitLab CI every 30 seconds

A separate website is available to display detailed information about previous tests. It provides a read only view of the database and gives access to signed URLs to retrieve files from storage (using the Amazon S3 interface).

Combining the ease of use of GitLab and its capability to manage secure workflows to update the analysis code, with the GitLab CI to LHCbDirac gateway provides LHCb with a very powerful tool to manage ntuple extraction from the LHCb dataset in an organized and efficient manner.

3.2 Deployment to CVMFS

The deployment of newly built/released software to CVMFS is also a good candidate for the use of the GitLab Gateway: it is easy to write a `LbCVMFSDeployment.ci.pull_job(runnner)`, that has the credentials to pull jobs from repositories like LHCbDirac, AnalysisProductions, LbEnv (or a deployment repository) and trigger the installation on CVMFS. It should also be able to check for feedback and report to GitLab. As LHCb refactors its CVMFS deployment installation, the plan is to develop such a runner.

3.3 Analysis preservation workflows

Analysis preservation workflows can also profit from using the GitLab CI Gateway: such use cases rely on having the credentials to access LHCb Grid data, and on being able to access significant CPU resources to process their data.

Several physics groups within the LHCb experiment already use GitLab CI for some analyses, with runners dedicated to their projects. The LHCb tutorials recommend automation with workflow management systems such as Snakemake [17][18], which allow re-running only part of the analysis of the data for which the input data or code has changed. Such workflows however rely on a cache of intermediary artefacts that can be reused between executions, to avoid re-computing everything from scratch.

Such a caching is not available at this stage but could be added to the system in a generic fashion, saving the local files after each job (to a scalable storage such as the CERN EOS or Ceph systems), and recovering them before the next one. We are investigating ways to integrate this into the system.

4 Conclusion

The GitLab CI Gateway demonstrated that it is possible replace standard GitLab runners by a custom one that allows integrating in a smoother manner with experiment resources. The current system is appropriate for some use cases (e.g. the Production Configuration validation) while more features are needed to handle other use cases, such as Analysis preservation. The system nonetheless proved its worth and its development will continue as the basis for LHCb GitLab runners.

References

[1] *The gitlab devops platform*, https://about.gitlab.com/
[2] *The worldwide lhc computing grid*, http://wlcg.web.cern.ch/
[3] *Python api for running gitlab ci jobs*, https://gitlab-runner-api.readthedocs.io
[4] *Dirac, the interware*, http://diracgrid.org/
[5] *The python programming language*, https://www.python.org/
[6] *The celery distributed task queue*, http://www.celeryproject.org
[7] *Rabbitmq messaging broker*, https://www.rabbitmq.com
[8] *The redhat openshift container platform*, https://www.openshift.com
[9] *Flask micro web framework*, https://palletsprojects.com/p/flask/
[10] R.T. Fielding, *Architectural styles and the design of network-based software architectures* (2000)
[11] *The gitlab rest api*, https://docs.gitlab.com/ee/api/
[12] *Python package installer*, https://pypi.org/project/pip/
[13] *Python packages entry points specification*, https://packaging.python.org/specifications/entry-points/
[14] *The kubernetes container orchestrator*, https://kubernetes.io/
[15] *The cernvm file system*, https://cernvm.cern.ch/portal/filesystem
[16] *Lhcb analysis productions*, https://gitlab.cern.ch/lhcb-dpa/analysis-productions/LbAnalysisProductions/
[17] *The snakemake workflow management system*, https://snakemake.readthedocs.io/en/stable/
[18] *Analysis automation with snakemake*, https://lhcb.github.io/starterkit-lessons/second-analysis-steps/analysis-automation-snakemake.html