Continuous Integration Comparison: GitHub vs GitLab

Automated Software Testing and DevOps

Nikolai Limbrunner
ntli@kth.se

April 4, 2022

Contents

1 Introduction

Git is the standard version control system used for almost all larger software projects. Github and Gitlab are the two largest cloud-based platforms to store, share and collaborate on Git repositories. Leaving aside other providers like Bitbucket, which are primarily used in larger companies, the decision often comes down to Github vs Gitlab. Although the core task is to store git repositories, Continuous Integration (CI) is a essential feature for both platforms. They offer a good alternative to external dedicated servers, like Jenkins or CircleCI. They can provide the whole functionality in one place, can react directly to changes in the repository and are also easier to configure in most cases. This essay gives an insight into how the two platforms integrated CI, compares functionality and configuration and helps to decide whether Github or Gitlab is the better choice as CI platform.

2 Background

2.1 Gitlab

Gitlab was initiated by Dimitri Zaporozhets in 2011 who wanted to use Git for version control and have a collaboration tool at the same time. He started Gitlab as open source project alone besides his job. Today Gitlab employs more than 600 people. In addition to the open source variant, Gitlab offers additional features under a commercial license and sells support [4]. Gitlab currently has about 30 million registered users, of which about 1 million are active licensed users.

2.1.1 Gitlab CI

The Gitlab CI was initiated in 2013 as a separate project and only later integrated into the main project [4]. Today Gitlab promotes itself as: "Gitlab is The DevOps Platform, delivered as a single application" and follows the vision to combine the whole toolchain [12]. Gitlab provides tools for Continuous Integration, Delivery, and Deployment. This allows testing, verification, building, and publishing of software without the need for third-party applications. One of the main reasons to chose Gitlab are the extensive CI and Continuous Development features and is and is often referred as a leader in this field [1].

2.2 Github

GitHub was developed by Chris Wanstrath, PJ Hyett, Scott Chacon and Tom Preston-Werner[3] who wanted to created a code hosting platform and launched in 2008. Github itself is not open source, although it heavily promotes open source. In 2018, Microsoft acquired Github for $7.5 billion. This lead to a significant decrease in their users, since many switched to the competitor Gitlab. Nevertheless Github is by far the largest code hosting platform, with over 100 million repositories, 73 million users and employs current around 2500 people [19]. Often Github is also called "social coding platform, since it is the go-to place to find new projects, collaborate and discuss [14].

2.2.1 Github Actions

For over 10 years since launch, Github set the focus and storing code and sharing it. They acknowledged that the collaborative nature of open source software development benefits immensely from the integration of CI [14] by launching Github Actions in 2019. This allows to run the code to automate development workflows. "Easily build, package, release, update, and deploy your project in any language—on GitHub or any external system—without having to run code yourself" [15]. In addition, Actions not limited to CI. They allow running any command when certain changes in the repository appear. For example could this include sending notifications for certain events. Github advertises that Github Actions serve a similar purpose for DevOps as Github initial for Open source [13]. Since the introduction of Github Actions was received very positively by the community and a lot of large open source projects added workflow to their repositories.

3 Core components and concepts

Both platforms use some common concepts that to create a CI pipeline. The basic building blocks of a pipeline are jobs, which in turn consist of a series of commands and scripts. These jobs execute commands, for example, to compile the code, run tests and publish/deploy it. Github and Gitlab also allow creating dependencies between these jobs. For example, the job to publish a package will only be executed if the job that runs the tests was successful. Runners are the machines on which the jobs are executed. When creating a CI pipeline, it can be specified on which operating system the jobs should be executed. A workflow combines the different parts and describes

which jobs are executed by which runners and by which events are triggered. There are different kinds of events. Scheduled events are triggered in a certain preset interval, manual events when clicking on "run" or webhook events when creating pull requests or issues [13]. Other concepts used by both platforms are artifacts to persist data across multiple jobs, caching to temporarily store calculated values, as well as variables and secretes that can be configured and accessed by the pipeline [3].

4 Comparison

4.1 Runners

While Github relies on *Virtual Machines* (VM) for the hosted runners, the runners on Gitlab run via Docker containers. Gitlab, therefore, provides better native integration for Docker containers. But since virtual machines provide a complete operating system, a Docker container can run inside a VM. From a performance point of view, Docker containers are superior to VM, primarily because they do not need the virtualization layer, which can lead to shorter runtimes of the pipeline [17]. In the use-case describes in Section 5 the Gitlab pipeline is around 25% faster (Note that there could be many other influence factors like the underlying hardware). Additionally, both platforms allow using self-hosted runners to meet specific hardware requirements. In this case, both technology's can be used by installing the respective executor software [13] [10].

4.2 Functionality

As discussed in Section 3, Gitlab CI and Github Actions rely on the same core principles, and a large part of the functionality is available on both platforms. However, they follow a slightly different approach to how they make it work. Gitlab has integrated all functionality directly into the platform and therefore works without 3rd party plugins. All features are developed directly by Gitlab, are verified, and work together seamlessly. In contrast, Github heavily relies on 3rd party plugins. The plugins are not directly developed by Github and therefore are not guaranteed that all work without problems [8]. At the same time, this results in superior integration of external services. Although Gitlab CI is significantly older, Github Actions offers a comparable solution for most tasks. Differences in functionality can be only found with more advanced and special features. These include *Pipeline orchestration* and *Merge orchestration*. Pipeline orchestration allows the

creation parent and child pipeline which can then also be run concurrently. This can shorten the time it takes to run pipelines. Merge orchestration allows additional logic to ensure that automatic merges are performed in the correct order and merge conflicts are avoided. [8]. Other minor features and details of Gitlab that are appreciated by developers include the ability to delete failed workflow runs or skip jobs [18]. When comparing the road-maps of the two platforms, it can be observed that Gitlab has already implemented features that Github is still working on, like action workflow visualization [7]. But it can be expected that Github will catch up very quickly.

4.3 Configuration

Both platforms use the YAML format to create pipelines. YAML is a human-friendly serialization language and alternative for the widely used JSON format. The abbreviation stands for "YAML Ain't Markup Language" and thus emphasizes that it is designed for data and not documents [16]. The configuration files for both platforms look similar, although they use slightly different keywords, and Github is usually a bit more verbose [3]. Many pipelines do similar tasks, so it makes sense to reuse configurations to avoid always having to start from scratch. Here both platforms have chosen a slightly different approach. Gitlab provides a feature called *Auto DevOps*. This allows one-click scanning of the project to detect the programming language, and project dependencies and chooses automatically the respective commands and tools to build, test, deploy and monitor the project. This feature works especially well for simple CI pipelines, and for complicated ones, there is a helpful template that can be customized and can save a lot of time. Github does not offer this feature but instead relies on the Github Marketplace. This marketplace offers a large number of templates that have been developed by a very large and active community. Furthermore, Github provides suggestions based on the repository. In most cases, these require fine-tuning to the respective pipeline but facilitate the process enormously. Github allows executing actions within a job instead of shell commands. These actions can be written in any programming language, e.g. in Javascript with Node.js and the actions toolkit. Which can then also be published via the marketplace [2]. Gitlab does not offer this functionality directly, but as a workaround, you could save the Javascript file in the repository and then execute it via a shell command.

Due to the fact that Gitlab offers an enormous number of features, the graphical interface is somewhat cluttered, which makes it seem less beginner-

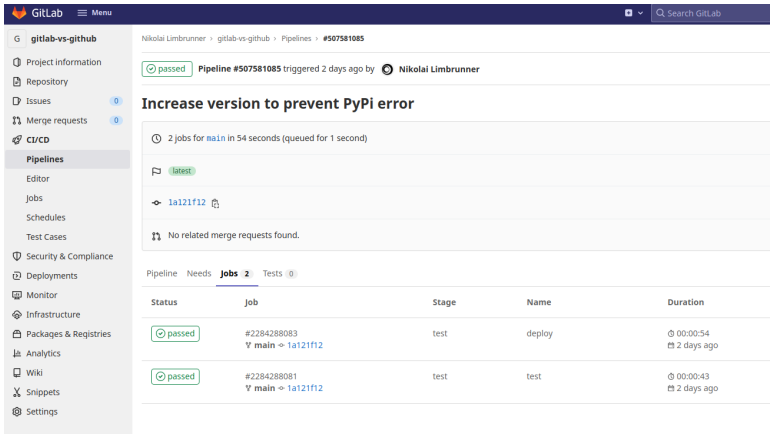

Figure 1: Gitlab CI Interface

friendly (Fig. 1). While Github offers a little fewer features and also offers a cleaner user interface (Fig. 2). Despite the differences mentioned here, the configuration of the pipelines is very similar, which makes it relatively easy to migrate a pipeline from Gitlab to Github or vice versa [3].

4.4 Security

The idea of DevSecOps is to integrate security into automated pipelines. This allows spot potential security issues while still developing the code, instead of at the end of the development cycle which makes it easier to fix and can prevent the deployment of the vulnerable software. Gitlab offers dedicated tools for scanning pipelines and monitoring and protecting containerized applications. Furthermore, there are additional security features included in the commercialized (ultimate) version to provide *Actionable Scan Results*, which included detailed information about the vulnerability and *Auto Remediation* to automatically patch known security issues [9]. Github Actions also offers the possibility to automatically test the code for security issues with a pipeline with the usage of an external plugin. Microsoft does not advertise the DevSecOps for Github that heavily and also recommends using Azure for more complex analysis or monitoring of pipelines [5].

4.5 Costs

Both platforms offer their CI functionality to a certain extent free of charge. Gitlab offers 400 free CI/CD minutes per month and charges 10$ per 1000

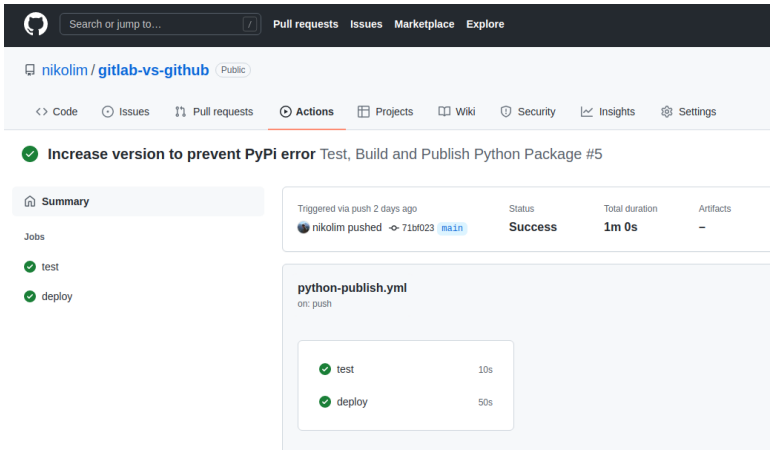

Figure 2: Github Actions Interface

minutes. Gitlab has recently drastically reduced its free minutes after an analysis showed that 98% of their users use under 400 minutes per month [11]. Github offers in contrast 2000 minutes free of charge per month, but these are only applicable to public repositories. Additional CI runtime is charged per minute and the price is based on the operating system of the selected virtual machine. They charge 0.008$ per minute for Linux, 0.016$ for Windows and 0.08$ for macOS [6]. Comparing the Linux price, on Github 1000 minutes costs 8$ and is therefore very close to the price on Gitlab.

5 Case study

After comparing the theoretical differences between the two platforms in the previous section, the approach and the user experience will be compared here using a concrete example. For this we want to write a pipeline that tests Python code, builds a package and then publishes it to PyPi so that it can be installed via pip. Note that this is a relatively simple pipeline that cannot compare all functions, especially the more complex ones, but it gives a first insight.

5.1 Gitlab CI

To create a pipeline on Gitlab, a file named *.gitlab-ci.yml* must be added to a repository in the root directory. This file is automatically picked up

and executed by Gitlab. Gitlab uses only a single main configuration file, but for more complex pipelines there is the possibility to out-source jobs and then include them in the main pipeline. In the *.gitlab-ci.yml* we start by specifying the image the Docker image the pipeline should run in. We are using *python:3.8* in this case. Next we specify the commands which should be run before each script. Here we setup the Python environment by installing the required packages. Afterwards we can define the test and deploy job. For the deploy job we install and use a package and provide the username and password via environment variables. Additionally we configure to the deploy job to only run on the main branch. The whole config file can be found in the appendix or in the repository [1].

5.2 Github Actions

To create a pipeline on Github, we need to create the folder *.github/workflows/* and then put in it files describing the pipeline with the extension *.yml* or *.yaml* in there. Putting the file into this dedicated has the advantage that you can store several pipelines files in this folder, which are then automatically executed by Github. Unlike Gitlab, we do not specify a Docker container here but *ubuntu-latest* as a virtual machine. Then we use predefined actions for both the test and the deploy job to check out the repository (actions/checkout@v3) and to set up python (actions/checkout@v3) and then install additional dependencies with pip. To not have to repeat these commands, we would either have to write a github action or outsource the commands to a script, since there is no "before_script" tag like in Gitlab. For testing the code we install and run pytest analog to Gitlab. For publishing the package to PyPi there is a Github action we can use, for this to work we just need to provide this action with a token which we have stored as a secret environment variable. Again, the whole config file can be found in the appendix or in the repository [2].

6 Conclusions and reflection

The comparison has shown that both platforms provide very similar functionality. The competition between these two already very good CI platforms also ensures that they are constantly being further developed. Gitlab offers slightly more functionality in the area of CI/CD than Github, gives

[1] https://gitlab.com/nlimbrun/gitlab-vs-github
[2] https://github.com/nikolim/gitlab-vs-github

slightly little more freedom in the configuration and might be the better choice if DevSecOps is important. Github on the other hand has the advantage that it hosts a large part of the open source projects and therefore has an extremely active community. This means that there are many ready-made Github actions that can be added directly via the Github web interface. Moreover, contrary to my expectations, the differences in the functionality were very small. Gitlab has the reputation of being the better CI/CD platform, but Github can do almost all jobs equally well. Thus, the CI functionality would not be a decisive point for me to decide on one of the two platforms. Github has the advantage of being the top dog for open source projects and thus projects might reach more people and find make contributors.

References

[1] Condo, C. Cloud-native continuous integration tools. *The Forrester Wave* (2019).

[2] Docs, G. Creating a javascript action. https://docs.github.com/en/actions/creating-actions/creating-a-javascript-action, 2022.

[3] Docs, G. Migrating from gitlab ci/cd to github actions. https://docs.github.com/en/actions/migrating-to-github-actions/migrating-from-gitlab-cicd-to-github-actions/, 2022.

[4] Evertse, J. *Mastering GitLab 12*. Packt Publishing, 2019.

[5] Github. Devsecops in github. https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/devsecops-in-github, 2022.

[6] Github. Pricing. https://about.gitlab.com/pricing/, 2022.

[7] Gitlab. Ci-cd roadmap comparison. https://about.gitlab.com/devops-tools/github-vs-gitlab/ci-cd-roadmap-comparison/, 2022.

[8] Gitlab. Continuous integration comparison. https://about.gitlab.com/devops-tools/github-vs-gitlab/ci-missing-github-capabilities/, 2022.

[9] Gitlab. Devsecops with gitlab. https://about.gitlab.com/solutions/dev-sec-ops/, 2022.

[10] Gitlab. Gitlab docs: Runner. https://docs.gitlab.com/runner/, 2022.

[11] Gitlab. Pricing. https://about.gitlab.com/pricing/, 2022.

[12] Gitlab. What is gitlab. https://about.gitlab.com/what-is-gitlab/, 2022.

[13] Heller, P. *Automating Workflows with GitHub Actions*. Packt Publishing, 2021.

[14] Kinsman, T., Wessel, M. S., Gerosa, M. A., and Treude, C. How do software developers use github actions to automate their workflows? *CoRR abs/2103.12224* (2021).

[15] Lardinois, F. Github launches actions, its workflow automation tool. https://techcrunch.com/2018/10/16/github-launches-actions-its-workflow-automation-tool/, 2018.

[16] Org, Y. Yaml docs. https://yaml.org/, 2022.

[17] Potdar, A. M., D G, N., Kengond, S., and Mulla, M. M. Performance evaluation of docker container and virtual machine. *Procedia Computer Science 171* (2020), 1419–1428. Third International Conference on Computing and Network Communications (CoCoNet'19).

[18] Stackshare. Github actions vs gitlab ci. https://stackshare.io/stackups/github-actions-vs-gitlab-ci, 2022.

[19] Warren, T. Github is back online after a two-hour outage. https://www.theverge.com/2021/11/27/22805076/github-down-outage-service-issues, 2022.