GitHub Actions status

This page describes the current status of GitHub Actions for Apache Software Foundation projects. This page is maintained by the community.

Summary of the GitHub Actions Status

TL;DR; summary Updated: 31.01.2020

If you are a Committer/PMC member of an ASF project and thinking about migrating to GitHub Actions, this is the current status:

- You should hold off with switching to GitHub Actions until we resolve performance and security problems that we are currently experiencing at the ASF level.
- If you want to use GitHub Actions, consider using your own self-hosted runner, but only if you can afford to build and maintain your own self-hosted infrastructure (this is not an easy task due to security limitations of the official GitHub Actions runners).
- If you decide to use GitHub Actions, be very careful to mitigate some of the security problems you might have if you follow the GA setup using the existing examples. There is extra hardening required in your workflows if you want to protect your project from 3rd-party dependencies having WRITE access to your project.

Overall status of GitHub Actions for Apache Software Foundation projects

There are already quite a few projects using GitHub Actions. However, there are unresolved problems with performance of the ASF-wide GitHub Actions Enterprise account, and there are some potential security implications that you might have to be aware of when starting to use GitHub Actions.

There are a few discussions that you can read at builds@apache.org about these issues:

- GA Again unreasonably slow
- Failure with GitHub Actions from outside of the organization (all of a sudden!)
- Issue with new GitHub Actions security policy

The issues with GitHub Actions revolve around Billing, Performance/Scalability and Security. Billing is not a problem on its own, but it impacts the Performance/Scalability issue.

Detailed status

Billing

All public projects, resources, images, etc. on GitHub are generally free (not only Apache Software Foundation ones). No problem with that. You will not incur any costs as long as you do not create any "private" resources, so there is no way you can create billing consequences. However there is an important caveat: as more projects use GitHub Actions, the more they all compete for a shared job queue (more on performance below). Apache Software Foundation has an "Enterprise" organisation status in GitHub.

Performance/Scalability

Status

Currently "native" GitHub Runner Performance/Scalability is far below the requirements of ASF projects using it (due to the 180 parallel job limit all ASF projects have).

If you are going to use GA you WILL experience severe and unacceptable performance issues, especially if your users and builds are predominantly in the EU/US time zone (this is the case for most of the biggest projects using GitHub Actions).

Apache Airflow (Tobiasz Kdzierski, a contributor to Apache Airflow) built a very crude dashboard showing the use of GA workflows (it’s not jobs but workflows) which clearly shows the extent of the problem.
This chart is imperfect - we currently do not have details "per-job" because GitHub is unable to give us the data. We've asked for it at the meeting organised by INFRA on 14 January, 2021, with people from GitHub Actions present, but (at least I am not aware of it) we do not have the data from GitHub available in a usable way (neither as raw data, nor dashboards). The meeting notes and preparation are here (no meeting notes yet available though): ASF Build Infra Meeting 14th of Jan 2021.

The chart is built using the GitHub API and due to API limitations (API call quota) we cannot drill down to the job level. But it's good enough to show the queues and numbers. we are talking about.

Regularly during the EU day/ US morning we have now 500-600 workflows in progress at a time from ASF projects (2 months ago it was 200-300 and then it was pretty OK). We ran our workflows in Apache Airflow for ~ 8 months without problems, but the last 1.5 months have been really problematic and I'd discourage using GA until those problems are solved.

From that dashboard we've built - the projects that seem to use GA most are: pulsar, spark, incubator-pinot, dubbo-samples, camel-k-runtime, netbeans, beam, airflow, incubator-daffodil, and commons-text.

**The reason for the issue**

The main reason for the issue is the limit of the job queue ASF (as any organisation) has. The ASF has an agreement (which is great on its own) with GitHub as an "Enterprise Organisation" (for free - this is GitHub’s donation to the ASF).

This means that ASF projects have 180 slots in the GitHub Actions Jobs queue allocated and no more than 180 GA jobs can run in parallel. This is far too small for the current demand. It has already caused a number of problems in the past when too many jobs for too many projects have been started at the same time. In January, 2021, during the weekdays in the EU day/US morning, we consistently experienced 5-6 hour queues for the jobs. This basically means that when you submit a PR, you have to wait 5-6 hours before it even STARTS running. This is unbearable and not sustainable.

We've implemented a multitude of optimizations in Airflow and we encouraged and helped other projects (such as Apache Beam, Apache SuperSet, and Apache SkyWalking) to optimize their workflows - including a few custom actions (Cancel Workflow Runs for example).

Unfortunately, there are no tools or mechanisms that could give ASF Infra the possibility of limiting the use of Actions per-project, and until this is solved any approach to limit the use of Actions for each project is destined to fail. Much of the effort we put into optimizing workflows in one of the projects has been very quickly consumed by other projects using more (for example Apache Airflow optimized the use of our workflows and decreased use by roughly 70%). There is also an ongoing effort from other projects to decrease the strain, for example:

- **issue** and design doc where maintainers of Pulsar discuss ways of decreasing the strain (with some help from the Apache Airflow team, who have already implemented the savings).
- **Kamil Bregula** from Apache Airflow, opened a number of PRs to implement a "Cancel Workflow Runs" action (in Pulsar, Spark, Pinot for example).
- The Apache SuperSet PR where they implemented their custom "cancel duplicates" python script.

To be perfectly clear - this is not a complaint, just a statement of the facts - those projects have no tool or mechanism to limit and monitor the usage of their workflows and there is no mechanism for ASF to enforce any limits per-project.

At the "Build Infra" meeting 14 January, 2021 developer advocates from GitHub mentioned that there might be a way to increase the queue. The ASF - rightfully so - cannot really pay for the increase (this is totally understandable if they have no tools to manage and control it). I am not aware about the results of this yet. Such an increase will only help for a short while, though. This is the same story as with motorways: if you have traffic jams and you widen the roads, it only takes a short time for the traffic to reach the capacity again as people start using the roads more.

**A potential solution**

One of the solutions that might be sustainable is to deploy self-hosted runners if your project has some infrastructure money (from stakeholders/sponsors) they can spend. We have money in Airflow (from the AWS Open-Source initiative and Astronomer; also Google promised to donate some GCP time). This is, however, (currently) inherently insecure. With the "PR-s from forks" approach of Apache projects, the current model of GitHub Runners is not secure by default. In fact, there is a recommendation from GitHub to NEVER use self-hosted runners for public repositories. Apache Airflow team forked the Runner and we are working on hardening the Self-hosted runners from GitHub, and we set up auto-scaling runners in our donated infrastructure (PMC member of Airflow - Ash Berlin-Taylor is working on it), but this is a big project on its own.
While Airlow has had some early successes and has a POC working, it’s already taking a few weeks to secure and test it, even if it is done together with a DevOps person to make it robust and secure. Ash Berlin-Taylor shared his early thoughts in the Self-hosted GitHub Runners document. This is a very rough description of what needs to be done and has a lot of "security" disclaimers and lacks full context (and needs some updates after further learnings).

The solution Apache Airlow introduced is a bit brittle. It is based on what the RUST team has done and it relies on dynamic patching of the runner from GitHub as soon as it is released because they have an aggressive policy of disabling old runners pretty much immediately after a new version is released. Thus it is prone to disruption of service if the patch does not apply cleanly. At the meeting 14 January, 2021 we learned that GitHub is not planning to improve security of the self-hosted runners any time soon (for sure we should not expect anything until mid-2021). So we are on our own for quite a while.

Security

There are a number of security problems you have to be aware of. The 3rd-party actions and 3rd-party dependencies are a huge security risk if not used appropriately (basically if you are using Actions as the examples suggest you are open for easy exploitation by the Action authors). If you do not securely add the Actions you are ripe to any kind of uncontrolled "write" modifications to your repository (!) by 3rd-party Action owners AND (as we’ve learned recently) by 3rd-party dependencies you install in your build pipeline. One of the problems caused INFRA action to disable the "direct" use of 3rd-party Actions at the organisation level (see the discussion), but there are many more risks that you have to be aware of.

There are two critical security vulnerability reports opened by Jarek Potiuk 30 December 2020 with GitHub Actions - both of them triaged and awaiting for actions on the GitHub side. GitHub Security Lab who in December encouraged users to post their experiences is engaged as well. Those issues can be all mitigated (Apache Airlow implemented all mitigation) but they are not what most projects do.

Mitigations

If you decide to use GitHub Actions, those are recommendations (there are varying opinions on sub-modules use, though):

- **ALWAYS** limit your GitHub write token to as little scope as possible. As of April 2021 there is a possibility of specifying scopes for the permissions of the token you automatically get during your build. https://github.blog/changelog/2021-04-20-github-actions-control-permissions-for-github_token/. This could help preventing sophisticated supply-chain attacks like the recent codecov attack.
- **NEVER** use 3rd-party actions directly in your workflows - use the "submodule" pattern. Example PR Tobiasz Kdzierski opened in SuperSet showing how this could be done. Also ASF INFRA allow-listed some of the popular Actions out there, including my "cancel workflow" action, but I there is no public list of those available. The nice things about submodules is that they do not bring action code to your repo. They link to commit hashes of the Actions, and that integrates well with the GitHub review process so that committers have better chance to review the changes before they are merged. By using submodules, you are automatically following the GitHub recommendations for hardening of security for 3rd-party actions.
- **ALWAYS** add "persist-credentials: false" to all your checkout actions. This is not done by default and is a huge security risk because it leaves your repository (and hundreds of thousands of others) open to 3rd-party dependencies to modify your repository (!) if you have any kind of "master" builds enabled. This is a "hidden" feature of the checkout action that is not at all obvious, but it leaves write access to your repository widely open to any code that you install during the build process. This is a very dangerous default.
- **NEVER** directly run code that might come with "forked" PRs in your workflows. There are certain exotic (but useful) workflows that are dangerous. For example, with "workflow_run" you might need to cancel duplicate workflows. Those workflows by default run with "master" code, but sometimes you might need to check out the incoming PR code for those. The host environment can have access (in various ways) to the "WRITE" GITHUB_TOKEN that has permission to modify your repository WITHOUT RESTRICTION OR NOTIFICATION. **NEVER** run the code that is checked out from the PR in your host environment. If you need to, run it in Docker Container to provide isolation from the host environment to avoid the "write" access leaking to users who prepare such a PR from their fork.
- **NEVER** install and run 3rd-party dependencies in the host of your build workflow code. Again there are ways those dependencies can obtain the WRITE GITHUB_TOKEN and change anything in your repository without your knowledge. There are very common "schedule" and "push" workflows that are especially prone to such abuse. Those run with "WRITE" access, and again there are ways to obtain the GitHub Token by these Actions and code that runs in your workflow. If you execute any 3rd-party code, run it in Docker containers to keep isolation from your "build" host environment to avoid leaking "write" access to those 3rd parties.