Github Actions or Jenkins? Making the Right Choice for You

GitHub Actions and Jenkins both get the job done. Let’s find out whether it’s worth considering moving from Jenkins

Over the past few years, DevOps have become a crucial part of the software life cycle. This fueled the growth of many leading DevOps tools and practices. You can find a range of tools to support the CI/CD process. Jenkins and GitHub Actions outstandingly stand among them.

In this article, I will compare GitHub Actions with Jenkins and provide you with the insight to make the right choice.

Introduction to Jenkins and GitHub Actions

Let's start with Jenkins. Here's a brief description,

"Jenkins is a free and Open-Source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery." ~ From Wikipedia

Similarly, GitHub Actions is the latest of the two offered by GitHub as a SaaS offering.

"GitHub Actions now makes it easier to automate how you build, test, and deploy your projects on any platform, including Linux, MacOS, and Windows. Run your workflows in a container or a virtual machine." ~ From the GitHub blog

Before deciding whether it's worth the shift, let's understand who should even consider this in the first place.

Should You Consider Shifting from Jenkins?

If everything is working out for you with Jenkins, you are confident in your setup while having full control and cost isn't an issue, I would recommend staying with Jenkins.

For those who use GitHub as the source control platform and already feel that you are not confident in your Jenkins setup and seek a better alternative, GitHub actions will become the primary choice for consideration.

Since GitHub Actions is a fully managed service by GitHub, you don't need to know how to scale and operate the infrastructure to run it.

This is the main reason I chose to move out from Jenkins, where I wasn't in full control of what's happening with my CI/CD pipelines.

Some of the challenges I had to deal with;

- Keeping the plugins up to date.
- My single Jenkins server build is costing money even if I don't run any builds.
- Not consistent on concurrent builds.
- I had to depend on several plugins, which come with updates that I need to deal with from time to time.

I know there are solutions with Jenkins to address some of these issues, but I had enough and moved towards a managed platform.

I hope I've set the right mindset to move to GitHub Actions if it applies to you. Let's look at the features that GitHub Actions offer to consider this move.

Ease of Setup — It's all Managed by GitHub

The first and foremost plus point of GitHub Actions over Jenkins, in my opinion, is the ease of setup in GitHub Actions. GitHub Actions operate in the cloud. You also have the option of running it locally, which is called a runner. On the contrary, Jenkins does not have an officially managed service offering.

And I might not go for any third-party managed offerings for Jenkins. I feel that it's too risky to hand over access to the source code and sensitive information to a third-party provider.

Due to this reason, the Jenkins server needs installation, whereas GitHub Actions does not need it. Consequently, the setting up process is much convenient in GitHub Actions. Besides, GitHub Actions are a series of docker runs. It only requires a `docker build` and `docker run`. This makes it very easy to run and debug.

Tight Integration with GitHub — Seamless Experience

At first, Jenkins seems more flexible than GitHub Actions. Jenkins is mainly based on accounts and triggers and centers on builds. These do not conform to GitHub events. In contrast to this, GitHub actions cover a wide range. Thus, there is a GitHub Action for every GitHub event.

GitHub Actions support many languages and frameworks, and they are also written in YAML. Therefore, these can be edited, reused, shared, and forked just like code.

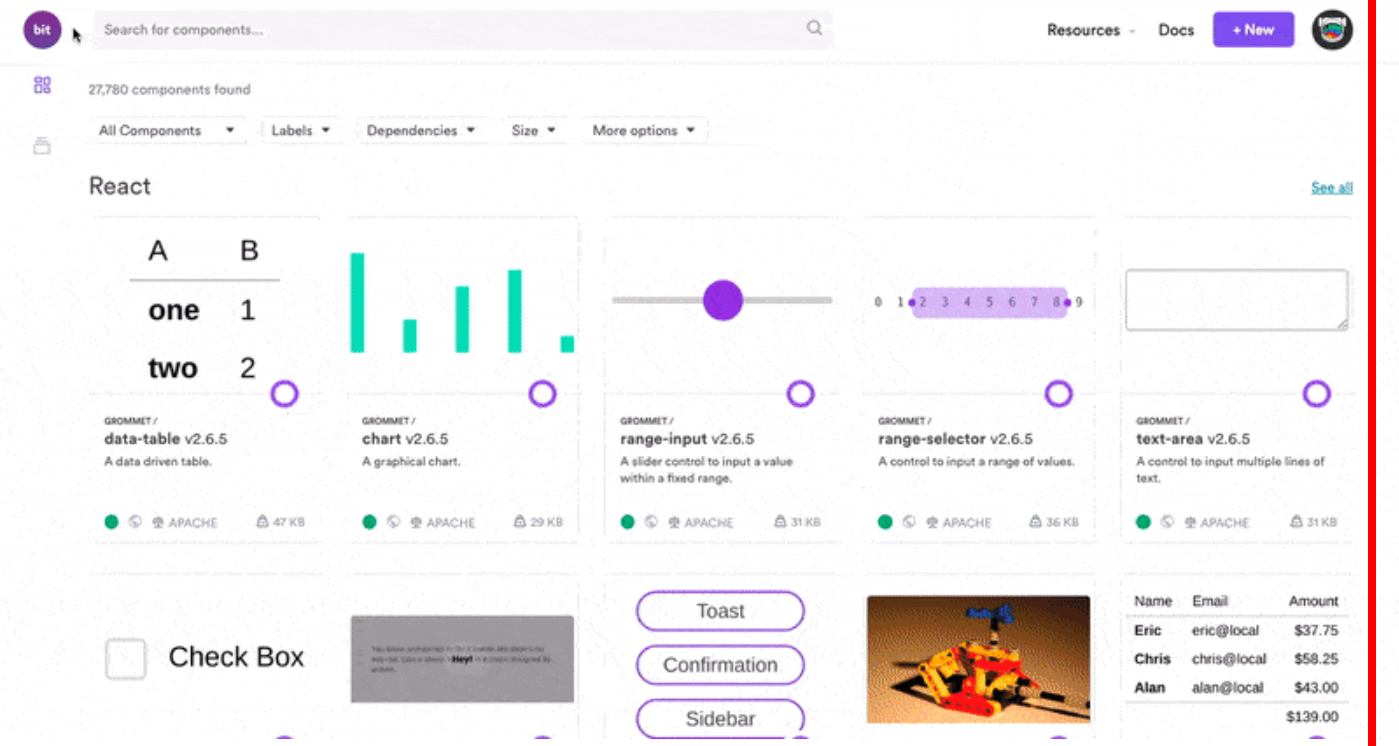
It is straightforward to use with GitHub because when you fork a repository, the actions automatically get forked.

This allows you to test and build projects very efficiently and even run them closer to the developer. Also, you have readily available access to the GitHub API, making it more popular among developers.

One popular use-case of such tight integration can be seen when using Bit (Github). Bit is a tool and platform that makes it easy to share JS components (Node, React, Vue, Angular, etc.) from any repository to Bit's cloud service, and from there to other repositories.

Bit's cloud service can auto-generate pull-requests to all Github repositories that are affected by a change made to one shared component. These auto-generated PRs can serve as triggers to Github Actions.

— What all that means is that **a change made to one single (shared) component can propagate across all repositories using it**, triggering CIs that validate nothing got broken across all projects 😮



Components shared on Bit.dev

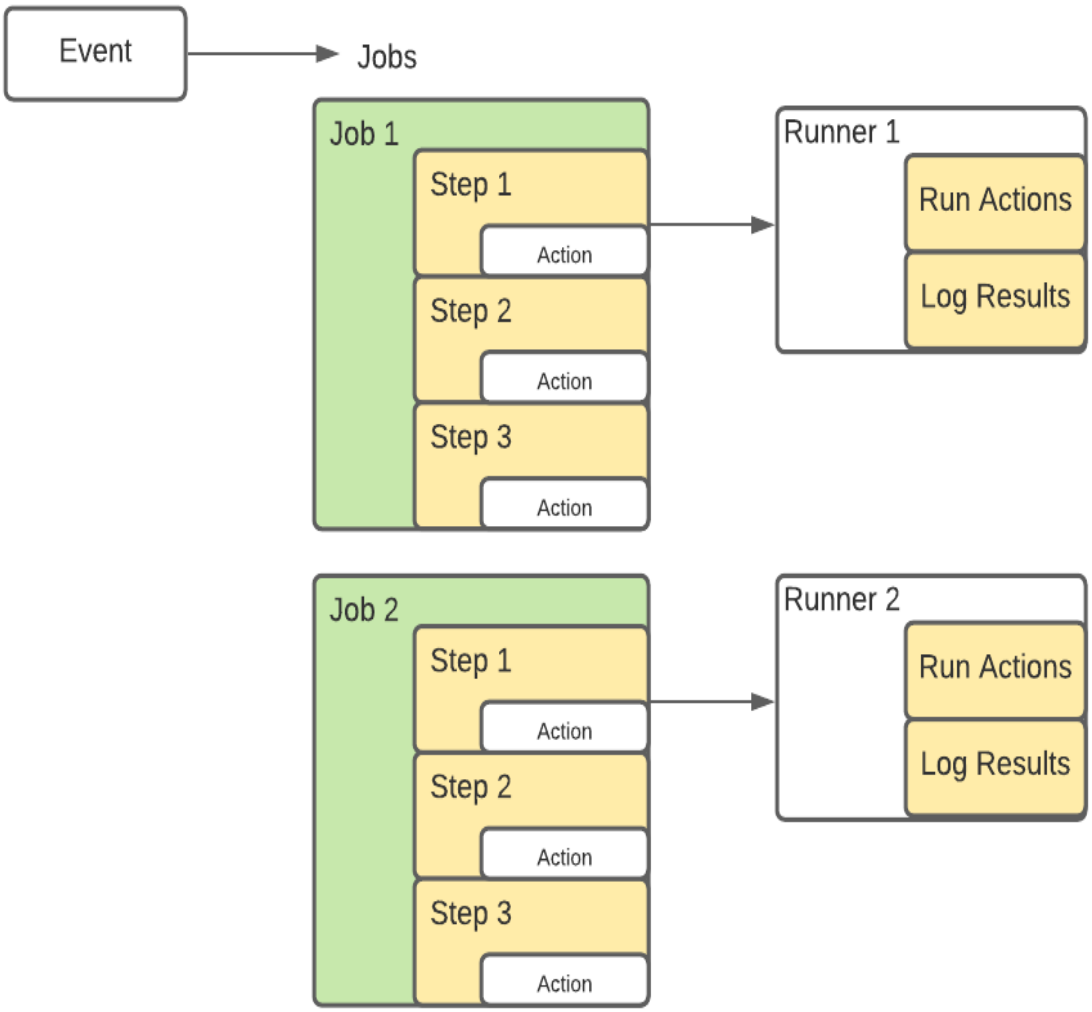
Read more about Bit <> Github integration here.

Another great "feature" of GitHub Actions is that they can be shared among each other via the GitHub Marketplace. You could reuse Actions written by other developers, which may save you an ample amount of time and avoid rewriting already available code.

Coordinator and Build Nodes — Built for Scale

GitHub Actions by default follow the master-slave (coordinator and build nodes) pattern as opposed to the sequential pipeline Jenkins offers us.

However, it's important to note that a similar setup is possible with Jenkins but will require additional effort and knowledge to get it up and running.



Multiple GitHub Actions components that work together to run jobs

If you use Jenkins, the default setup will run each step in the deployment pipeline synchronously. For example, if you need to run Unit tests, Integration tests, and some Sonar verifications, they will have to run in a single server environment. It might delay the

executions based on the available resources in your server. Besides, you won't have to put extra effort into making the pipelines reliable.

With the use of GitHub Actions, these can be parallelized, as shown in the image above. e.g., Job 1 could be Unit tests and Integrations tests, Job 2 could be Sonar verifications.

Summary

We critically looked at a few areas where GitHub Actions precede Jenkins as far as their advantages are concerned. Moreover, GitHub Actions are growing faster than Jenkins, where thousands of GitHub Actions are released to the GitHub marketplace. The community around this is also improving where there are dedicated repositories for GitHub Actions. What does that mean?

The crowd loves it!

Let's look at a summary of the comparison.

Jenkins	GitHub Actions
Server needs installation	No installation required as it is on the cloud
Tasks or jobs will be synchronous, which will consume more time to deploy a product to market	Asynchronous CI/CD is achieved
Is based on accounts and triggers and centers on builds which does not conform to GitHub events	Provides actions to every Github event and supports many languages and frameworks
Need to run on a Docker image for environment compatibility	Compatible with any environment
Plugins are available to support caching mechanisms	Have to write your own caching mechanism if you require caching
Does not have the capability to be shared	Can be shared via the GitHub Marketplace.

However, whether you use GitHub Actions or Jenkins in your project is up to you. At present, GitHub Actions are free to use for public repositories. For private repositories, it has a pay-as-you-go mechanism.

I hope you have already realized that GitHub Actions is a dominant choice over Jenkins, mainly because of its flexibility. For those starting with a new project or using GitHub as their source control platform, it's a no brainer to move towards GitHub Actions.