Jenkins vs GitLab

GitLab compared to other DevOps tools

On this page

Summary

Jenkins is one of the most popular self-managed open source build automation and CI/CD developer tool in the world. It derives it's incredible flexibility from incorporating capabilities from it's hundreds of available plugins, enabling it to support building, deploying and automating any project.

At the Q3 2018 Jenkins World conference CloudBees (the primary maintainers of Jenkins) announced their intention to revive the competitiveness of Jenkins by splitting it and focusing on a cloud native version, as well as a simplified, opinionated version (Jenkins Evergreen). There is also a Jenkins sub-project called Jenkins X, meant to make running a pipeline out of the box with Kubernetes easier.

Jenkins X natively integrates Jenkins CI/CD server, Kubernetes, Helm, and other tools to offer a prescriptive CI/CD pipeline with best practices built-in, such as using GitOps to manage environments. It uses deployment of Jenkins into Kubernetes containers to get around the complexities of installing and integrating Jenkins. However, it is a complex pairing of many tools including the fragile Jenkins server.

In contrast, GitLab already provides more than what Jenkins is hoping to evolve to, by providing a fully integrated single application for the entire DevOps lifecycle. More than Jenkins' goals, GitLab also provides planning, SCM, packaging, release, configuration, and monitoring (in addition to Cthe I/CD that Jenkins is focused on).

Comments/Anecdotes

- Jenkinstein - From the article DevOps World 2018: â€˜ Jenkinsteinâ€™ and a Cloud Native Jenkins
  - Describing the snowflake server and the "Brent" situation and how it slows down everything. This is one of the main issues which Jenkins users face today and which Jenkins has no easy fix for:

    > Acting as the gatekeeper for that channel is someone CloudBees CEO Sacha Labourey described as â€œthe Jenkins guyâ€¦ this superstar devoted to making Jenkins great on his team. Because this person is the authority on deployment within his organization, multiple teams come to rely on him to meet their scheduling goals. Yet this leads to a technical issue that few folks outside of IT operations take time to consider: The Jenkins Master . . . (the server managing a distributed scheme with multiple agents) becomes bloated, like an old telephone directory or the Windows XP System Registry.
    >
    > And because that organizationâ€™s Jenkins deployment is not only dependent upon the Guy, but somewhat bound to his choices of plug-ins, the result is what the CEO called â€œFrankenstein Jenkins,â€ and what other developers and engineers at the conference Tuesday had dubbed â€œJenkinstein.â€

  - Again, CloudBees CEO Sacha Labourey describing the common problem with current Jenkins:

    > Jenkins becomes bloated, slow to start. When it crashes, it takes forever to start. Hundreds of developers are pissed. And nobody wants to fix it, because if you touch it, you own it, right?â€

- From note on 2018-08-31 from CloudBees CTO and Jenkins creator Kohsuke Kawaguchi (https://jenkins.io/blog/2018/08/31/shifting-gears/):

    - "Our Challenges. . . Service instability. . . Brittle configuration. . . Assembly required. . . Reduced development velocity. . ." (see above link for details on each)
    - "Path forward. . . Cloud Native Jenkins. . . continue the incremental evolution of Jenkins 2, but in an accelerated speed"
    - **Key takeaways**:
        - They plan on BREAKING backward compatibility in their upcoming releases
        - They plan on introducing a NEW flavor of Jenkins for Cloud native
        - If youâ€™re a Jenkins user today, itâ€™s going to be a rough ride ahead
- From Jenkins Evergreen project page (identified in Kohsuke letter above as key for changes that need to come) - https://github.com/jenkinsci/jep/blob/master/jep/300/README.adoc
    - "Pillars . . . Automatically Updated Distribution . . . Automatic Sane Defaults. . . Connected. . . Obvious Path to User Success"
    - "The "bucket of legos" approach is . . . not productive or useful for end-users [5] who are weighing their options between running Jenkins, or using a CI-as-a-Service offering such as Travis CI or Circle CI."
    - "existing processes around "Suggested Plugins", or any others for that matter, result in many "fiefdoms" of development rather than a shared understanding of problems and solutions which should be addressed to make new, and existing, users successful with Jenkins."
- From "Problem" definition page of Jenkins Evergreen project page (https://github.com/jenkinsci/jep/blob/master/jep/300/README.adoc#problem):

> For novice-to-intermediate users, the time necessary to prepare a Jenkins environment "from scratch" into something productive for common CI/CD workloads, can span from hours to days, depending on their understanding of Jenkins and itâ€™s related technologies. The preparation of the environment can also be very error prone and require significant on-going maintenance overhead in order to continue to stay up-to-date, secure, and productive.
>
> Additionally, many Jenkins users suffer from a paradox of choice [6] when it comes to deciding which plugins should be combined, in which ways, and how they should be configured, in order to construct a suitable CI/CD environment for their projects. While this is related to the problem which JEP-2 [7] attempted to address in the "Setup Wizard" introduced in Jenkins 2.0, Jenkins Evergreen aims to address the broader problem of providing users with a low-overhead, easily maintained, and solid distribution of common features (provided by a set of existing plugins) which will help the user focus on building, testing, and delivering their projects rather than maintaining Jenkins.

- GitLab blog on feedback about new Jenkins improvement efforts - https://about.gitlab.com/2018/09/03/how-gitlab-ci-compares-with-the-three-variants-of-jenkins/
- Project analysis on Jenkins, pointed to by CloudBees documentation as proof for need of change - https://ghc.haskell.org/trac/ghc/wiki/ContinuousIntegration#Jenkins

> Pros
>
> - We can run build nodes on any architecture and OS we choose to set up.
>
> Cons
>
> - Security is low on PR builds unless we spend further effort to sandbox builds properly. Moreover, even with sandboxing, Jenkins security record is troublesome.
> - Jenkins is well known to be time consuming to set up.
> - Additional time spent setting up servers.
> - Additional time spent maintaining servers.
> - It is unclear how easy it is to make the set up reproducible.
> - The set up is not forkable (a forker would need to set up their own servers).

- From GitLab PMM

> - â€œJenkins had to build a whole new separate project in order to work with Kubernetes. GitLab has natively adopted Kubernetes from the get-go.â€

- Jenkins X adoption is tiny. Most folks looking to go to Kubernetes will be on Jenkins proper, so thePinterest anecdote applies.
- Although Jenkins X works with Kubernetes, itâ€™s not a single application like GitLab. You still have to integrate to your PPM, SCM, security tools, etc. You have to manage permissions and access across all that which GitLab gives you out of the box, but Jenkins X does not (value of a single app).

Resources

- Jenkins OSS Website - Open Source project website
- CloudBees Jenkins Website - https://www.cloudbees.com/products/cloudbees-core
- Jenkins X Website

Pricing

- Jenkins OSS
  - No cost (and Open Source)
  - But Total Cost of Ownership is not zero, given maintenance requirements
- CloudBees Jenkins (vague) - https://www.cloudbees.com/products/pricing
  - CloudBees Core - Jenkins distribution with upgrade assistance on monthly incremental upgrades, cloud native architecture, centralized management, 24/7 support and training, enterprise-grade security and multi-tenancy, and plugin compatibility testing
    - starting at $20k/year for 10 users, with tiered pricing for lower per-user cost for larger organizations
- Jenkins X
  - No cost (and Open Source)
  - But Total Cost of Ownership has cost (see Pinterest anecdote)

Comparison

FEATURES		
Built-in CI/CD GitLab has built-in Continuous Integration/Continuous Delivery, for free, no need to install it separately. Use it to build, test, and deploy your website (GitLab Pages) or webapp. The job results are displayed on merge requests for easy access. Learn more about CI/CD	✓	✓
Application performance monitoring GitLab collects and displays performance metrics for deployed apps, leveraging Prometheus. Developers can determine the impact of a merge and keep an eye on their production systems, without leaving GitLab. Learn more about monitoring deployed apps	✗	✓

Application performance alerts

GitLab allows engineers to seamlessly create service level indicator alerts and be notified of any desired events, all within the same workflow where they write their code.

Learn more about creating SLI alerts

GitLab server monitoring

GitLab comes out of the box enabled for Prometheus monitoring with extensive instrumentation, making it easy to ensure your GitLab deployment is responsive and healthy.

Learn more about monitoring the GitLab service

Cycle Analytics

GitLab provides a dashboard that lets teams measure the time it takes to go from planning to monitoring. GitLab can provide this data because it has all the tools built-in: from the idea, to the CI, to code review, to deploy to production.

Learn more about Cycle Analytics

Built-in Container Registry

GitLab Container Registry is a secure and private registry for Docker images. It allows for easy upload and download of images from GitLab CI. It is fully integrated with Git repository management.

Documentation on Container Registry

Preview your changes with Review Apps

With GitLab CI/CD you can create a new environment for each one of your branches, speeding up your development process. Spin up dynamic environments for your merge requests with the ability to preview your branch in a live environment.

Learn more about Review Apps

A comprehensive API

GitLab provides APIs for most features, allowing developers to create deeper integrations with the product.

Read our API Documentation

CI/CD Horizontal Autoscaling

GitLab CI/CD cloud native architecture can easily scale horizontally by adding new nodes if the workload increases. GitLab Runners can automatically spin up and down new containers to ensure pipelines are processed immediately and minimize costs.

Learn more about GitLab CI/CD Horizontal Autoscaling


Built for containers and Docker

GitLab ships with its own Container Registry, Docker CI Runner, and is ready for a complete CI/CD container workflow. There is no need to install, configure, or maintain additional plugins.


Cloud Native

GitLab and its CI/CD is Cloud Native, purpose built for the cloud model. GitLab can be easily deployed on Kubernetes and used to deploy your application to Kubernetes with support support out of the box.

Kubernetes integration


Container debugging with an integrated web terminal

Easily debug your containers in any of your environments using the built-in GitLab Web Terminal. GitLab can open a terminal session directly from your environment if your application is deployed on Kubernetes. This is a very powerful feature where you can quickly debug issues without leaving the comfort of your web browser.

Learn more about the web terminal


Comprehensive pipeline graphs

Pipelines can be complex structures with many sequential and parallel jobs. To make it a little easier to see what is going on, you can view a graph of a single pipeline and its status.

Learn more about pipeline graphs


Browsable artifacts

With GitLab CI you can upload your job artifacts in GitLab itself without the need of an external service. Because of this, artifacts are also browsable through GitLab's web interface.

Learn more about using job artifacts in your project


Scheduled triggering of pipelines

You can make your pipelines run on a schedule in a cron-like environment.

Learn how to trigger pipelines on a schedule in GitLab


Code Quality

Code Quality reports, available in the merge request widget area, give you an early insight into how the change will affect the health of your code before deciding if you want to accept it.

Learn more about Code Quality reports

 

Multi-project pipeline graphs

With multi-project pipeline graphs you can see how upstream and downstream pipelines are linked together for projects that are linked to others via triggers as part of a more complex design, as it is for micro-services architecture.

Learn more about multi-project pipeline graphs

 

Protected variables

You can mark a variable as "protected" to make it available only to jobs running on protected branches, therefore only authorized users can get access to it.

Learn how to use protected variables

 

Environments and deployments

GitLab CI is capable of not only testing or building your projects, but also deploying them in your infrastructure, with the added benefit of giving you a way to track your deployments. Environments are like tags for your CI jobs, describing where code gets deployed.

Learn more about environments

 

Environments history

Environments history allows you to see what is currently being deployed on your servers, and to access a detailed view for all the past deployments. From this list you can also re-deploy the current version, or even rollback an old stable one in case something went wrong.

Learn more about history of an environment

 

Environment-specific variables

Limit the environment scope of a variable by defining which environments it can be available for.

Learn how to configure environment-specific variables

 

Group-level variables

Define variables at the group level and use them in any project in the group.

Learn how to configure variables

 

Customizable path for CI/CD configuration

You can define a custom path into your repository for your CI/CD configuration file.

Learn how to configure a custom CI/CD configuration file

 

Run CI/CD jobs on Windows

GitLab Runner supports Windows and can run jobs natively on this platform. You can automatically build, test, and deploy Windows-based projects by leveraging PowerShell or batch files.

Install GitLab Runner on Windows

 

Run CI/CD jobs on macOS

GitLab Runner supports macOS and can run jobs natively on this platform. You can automatically build, test, and deploy for macOS based projects by leveraging shell scripts and command line tools.

Install GitLab Runner on macOS

 

Run CI/CD jobs on Linux ARM

GitLab Runner supports Linux operating systems on ARM architectures and can run jobs natively on this platform. You can automatically build, test, and deploy for Linux ARM based projects by leveraging shell scripts and command line tools.

Install GitLab Runner on Linux

 

Run CI/CD jobs on FreeBSD

GitLab Runner supports FreeBSD and can run jobs natively on this platform. You can automatically build, test, and deploy for FreeBSD-based projects by leveraging shell scripts and command line tools.

Install GitLab Runner on FreeBSD

 

Show code coverage rate for your pipelines

GitLab is able to parse job output logs and search, via a customizable regex, any information created by tools like SimpleCov to get code coverage. Data is automatically available in the UI and also as a badge you can embedd in any HTML page or publish using GitLab Pages.

Learn how to generate and show code coverage information in GitLab

 

Manage JUnit reports created by CI jobs

Many languages use frameworks that automatically run tests on your code and create a report: one example is the JUnit format that is common to different tools. GitLab supports browsing artifacts and you can download reports, but we're still working on a proper way to integrate them directly into the product.

Read more on the issue


Details on duration for each command execution in GitLab CI/CD

Other CI systems show execution time for each single command run in CI jobs, not just the overall time. We're reconsidering how job output logs are managed in order to add this feature as well.

Read more on the issue


Auto DevOps

Auto DevOps brings DevOps best practices to your project by automatically configuring software development lifecycles by default. It automatically detects, builds, tests, deploys, and monitors applications.

Read more about Auto DevOps in the documentation


Protected Runners

Protected Runners allow you to protect your sensitive information, for example deployment credentials, by allowing only jobs running on protected branches to access them.

Read more on the issue


Easy integration of existing Kubernetes clusters

Add your existing Kubernetes cluster to your project, and easily access it from your CI/CD pipelines to host Review Apps and to deploy your application.

Read more on the issue


Easy creation of Kubernetes clusters on GKE

Create a Kubernetes cluster on GKE directly from your project, just connecting your Google Account and providing some information. The cluster can be used also by Auto DevOps to deploy your application.

Read more on the issue


Support for multiple Kubernetes clusters

Easily deploy different environments, like Staging and Production, to different Kubernetes clusters. This allows to enforce strict data separation.

Read more on the issue

 

Easy Deployment of Helm, Ingress, and Prometheus on Kubernetes

Install Helm Tiller, Nginx Ingress, Prometheus and GitLab Runner directly into your cluster from the GitLab Web UI with one click.

Read through the documentation on installing applications on GKE clusters

 

Canary Deployments

GitLab Enterprise Edition Premium can monitor your Canary Deployments when deploying your applications with Kubernetes.

Learn more about configuring Canary Deployments

 

Minimal CI/CD configuration

GitLab CI/CD requires less configuration for your pipelines than other similar setups like Jenkins.

Learn more about GitLab CI/CD

 

Automatic Retry for Failed CI Jobs

You can specify a retry keyword in your .gitlab-ci.yml file to make GitLab CI/CD retry a job for a specific number of times before marking it as failed.

Learn more about Automatic Retry for Failed CI Jobs

 

Pipelines security

The ability of running CI/CD pipelines on protected branches is checked against a set of security rules that defines if you're allowed or not. It includes creating new pipelines, retrying jobs, and perform manual actions.

Learn more about pipeline security

 

Include external files in CI/CD pipeline definition

You can include external files in your pipeline definition file, using them as templates to reuse snippets for common jobs.

Learn more about including external files

 

Step folding for CI/CD logs

Collapse the job log output for each command.

 

Read more on the issue

View Kubernetes pod logs

Quickly and easily view the pod logs of an app deployed to Kubernetes.

 

Learn more about viewing Kubernetes pod logs