Engineer

GitLab

a git platform you can also run yourself

What it is

GitLab is a single application built around git repositories: it hosts the code and adds merge requests for review, issues for planning, and a built-in CI/CD system that builds, tests and deploys from the same place. What sets it apart is the choice of where it runs. The same product is offered as GitLab.com, the service GitLab runs itself; as GitLab Self-Managed, which you install on servers of your own; and as GitLab Dedicated, a single-tenant instance that GitLab hosts and maintains for one customer in the AWS region that customer prefers.

Underneath is ordinary git. Every project is a git repository that can be cloned, pushed to and fetched from with the usual commands, and the history in a clone is complete. Everything GitLab adds on top, from review threads and issues to pipeline results and permissions, is kept by GitLab itself, outside the repository, and a clone carries none of it.

A merge request is GitLab's name for a proposed change waiting to be merged, the same idea other platforms call a pull request. The documentation describes it as a central place to review code, hold the discussion and track the change. One page shows the description, the code changes with inline review comments, the state of the CI/CD pipelines, the reports on whether the change can be merged, and the list of commits. A merge request can be linked to an issue and close that issue automatically when it merges. By default the project's default branch, usually main, is protected, and only people with the Maintainer role or higher may merge into it.

The CI/CD system is configured in a file named `.gitlab-ci.yml` at the root of the project. It declares the stages, jobs and scripts that together form a pipeline: stages set the order, such as build, then test, then deploy, and jobs are the tasks performed within each stage. The jobs are executed by GitLab Runner, a separate

program written in Go and shipped as a single binary. A runner connects to a GitLab instance, waits for jobs, runs each one on the machine itself, inside a Docker container or on a remote server over SSH, and reports the result back.

Use the concept in a decision

Put “What is GitLab?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

Most of what a software team does between writing code and shipping it can happen inside one GitLab instance: the code, the review, the pipeline that tests it, the issue that asked for it and the record of who approved what. One application means one set of accounts and permissions, one place to search, and links that hold between an issue, the merge request that answered it and the pipeline that checked it.

The self-managed option is the reason many organisations choose GitLab at all. When code cannot leave a company’s own infrastructure, because of a contract, a regulator or plain policy, a hosted service is out, and GitLab can be installed inside the network that has to keep it. The documentation treats this as a first-class use: it publishes reference architectures for installations sized from 1,000 to 50,000 users, alongside the Linux package and a Helm chart for Kubernetes.

That control has a price, and it is paid in operations rather than in licence fees alone. Running GitLab yourself means sizing a server for it, keeping its PostgreSQL database and its Redis or Valkey store healthy, providing machines for the runners, taking backups that can actually be restored, and moving through upgrades in the order the project requires. The hosted options hand that work to GitLab; the self-managed one keeps it with you.

What it looks like in practice

The company’s own work has met GitLab from both directions.

Over several years of data engineering, CI/CD practices built on GitHub and GitLab, with Bash and Python doing the automation underneath, replaced manual steps in both the data analysis and the software development. The way work moved from development to delivery was standardised so it was the same every time rather than reinvented per project, and the analysis side, long treated as a separate and more manual world, was brought into the same pipeline as the development work. The page records roughly 4,000 hours saved, and work that shipped more consistently because it was no longer done a slightly different way each time.

The company's own code shows the other end of self-hosting. When it moved off a third-party hosting service, it went to a forge it runs itself, that is a single binary, SSH-first and with no administrative web interface, chosen on the principle that the less interface there is, the less there is to defend. That forge shares a public-services host with 512 MB of memory, about 464 MB of it usable, with a web server for seven domains, a Tor onion service, a Gemini server, a Gopher server and intrusion banning. Backup automation is implemented separately; its implementation is distinct from active backups.

Neither page mentions GitLab, and the comparison that follows is this article's own arithmetic. GitLab documents 8 GB of memory as the least a single-node installation can run with; the whole of that host has a sixteenth of it. Being able to run GitLab yourself means having a server sized for GitLab, and a small, single-purpose forge is the other way to keep code on your own machines.

A minimal pipeline file shows the shape of the CI/CD configuration, with two stages that run in order and one job in each:

```
stages:
    - build
    - test

build-site:
    stage: build
    script:
        - make build

run-tests:
    stage: test
    script:
        - make test
```

Where it goes wrong

- **A small server makes the whole instance slow.** GitLab's documentation gives 16 GB of memory as the baseline for a single-node installation and 8 GB as the least it can run with in a memory-constrained environment. It also warns that under load, swap can degrade performance significantly, and asks for swap to be disabled where possible or for enough memory that GitLab never touches it.
- **A backup restores into an instance that cannot read its own data.** The backup command leaves out the configuration directory, `/etc/gitlab`, and with it `gitlab-secrets.json`, the keys that decrypt encrypted values in the database. Without that file, a restored database keeps those values locked. The documentation asks for the configuration to be backed up separately, at the very least `gitlab-secrets.json` and `gitlab.rb`.
- **An upgrade jumps over a required stop.** Upgrade paths include required stops, versions that must be installed before any later one, and since GitLab 17.5 they fall at x.2, x.5, x.8 and x.11. The background migrations of each stop have to finish before the next upgrade starts, so an instance that falls several versions behind catches up one stop at a time.
- **A managed data store turns out to be unsupported.** PostgreSQL is the only supported database. Redis or Valkey holds sessions and background tasks and has to be a standalone instance: Redis Cluster and serverless variants of Redis and Valkey are not supported. A managed cloud service is allowed, so long as it is one of the standalone kinds.
- **Pipelines wait on machines nobody planned for.** GitLab.com offers ready-made runners, but on a self-managed instance the administrator provides and manages the infrastructure where CI/CD jobs run, which includes making sure the runners have enough capacity for the organisation's workload. A busy team with one small runner queues behind it.

Further reading

- GitLab Docs — the full documentation for using, installing and administering GitLab.
- Installation requirements — memory, CPU, storage and the supported PostgreSQL and Redis versions.
- Back up GitLab — what a backup contains, what it leaves out and how to store the secrets beside it.
- Upgrade paths — the required upgrade stops and why background migrations must finish between them.
- Get started with GitLab CI/CD — the `.gitlab-ci.yml` file, stages, jobs and pipelines.

- GitLab Runner — the program that runs the jobs, and the ways it can run them.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is GitLab?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/gitlab/