How GitLab is Enterprise Class

What's in this page

Overview

This paper looks at some details about GitLab, focusing on how it operates in an enterprise environment.

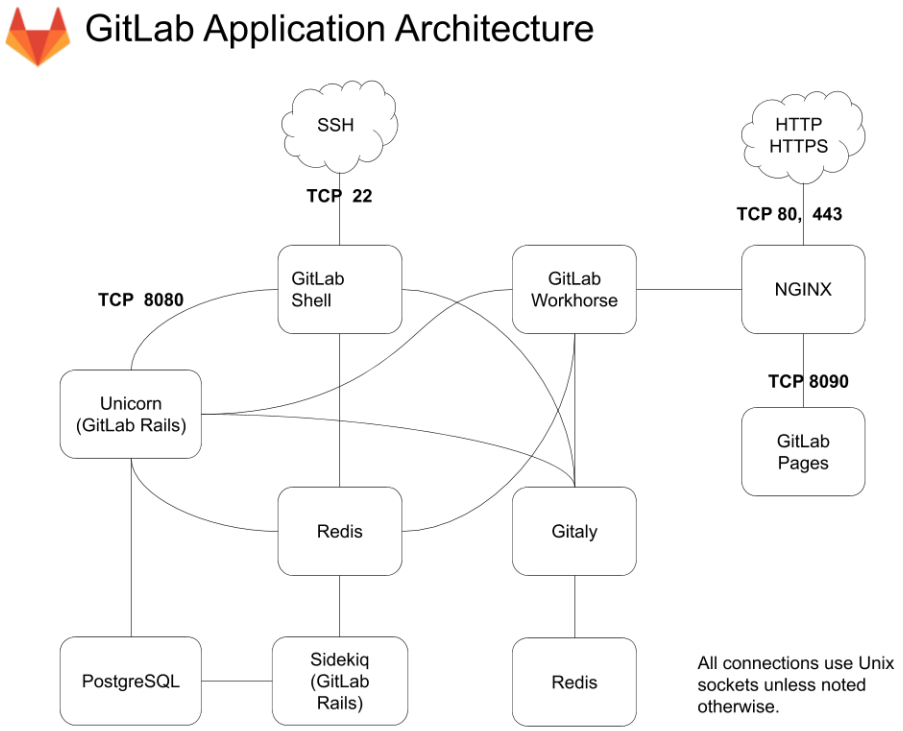
GitLab has been designed to be easy to deploy and maintain. A single system configuration can scale to support over 32,000 active users, requiring minimal administration effort, and a multi-node, distributed services configuration can scale to support 100's of thousands of active users. Implementing High Availability, Backups, and Disaster Recovery has been made simple with GitLab Omnibus packaging. The GitLab codebase is used in over 66% of the self-managed Git market and is tested in the crucible of GitLab.com, with over 2 million active users.

Although choosing an SCM tool and strategy is a very important decision for a software organization to make, figuring out how it will work with the rest of the tools in the DevOps toolchain is equally important. GitLab brings more than just an SCM tool, with it's built in issue tracking, CI/CD, security scanning, container management, deployment, and monitoring. And because this is all a single application covering the entire DevOps lifecycle, the training costs and total cost of ownership are significantly reduced over a hybrid toolchain consisting of several independently developed tools.

Architecture and Installation

GitLab is delivered in many ways to match many different installation scenarios (https://about.gitlab.com/installation). While GitLab can be installed from source, it also ships with all of the components necessary to run it in a single automated installation (whether by Omnibus or appliance) (https://docs.gitlab.com/omnibus/roles/README.html). The GitLab Omnibus installer can simplify the installation and upgrades to single machine and multi-machine installs.

GitLab installation of HA can be done using self contained appliance-like installations (https://about.gitlab.com/high-availability/#single-slave-server), but GitLab also offers the flexibility for administrators to configure HA at the individual component level or groups of related components (roles) (https://docs.gitlab.com/omnibus/roles/README.html), enabling the choice of what mechanisms fits best in their scenario (e.g. using Amazon RDS which offers Multi AZ for redundancy). We believe this approach gives the customer the most flexibility in deciding how to set up their best HA solution given their available resources, while still assisting with the necessary configurations.



Scalability

When configured with a single system, GitLab can scale to support over 32,000 active users. But in a multi-node, distributed services configuration GitLab can scale to support 100's of thousands of active users. An example of this level of scalability can be seen publically on GitLab.com, which hosts millions of users, with 100's of thousands of active users daily. The architecture for the GitLab.com installation can be found at https://about.gitlab.com/handbook/engineering/infrastructure/production/architecture/.

Hardware

Although GitLab encourages use of its active-active configuration to support high-loads, GitLab supports both active-active configurations and active-passive configurations. Both require at least doubling of the infrastructure, but in active-passive only half of the investment is actually used at any given time.

GitLab believes that highly scalable and highly available service requirements vary by implementation and that the product should be flexible enough to support customer needs given the environment and resources that the customer has to run it in. As such, GitLab supports several different types of clustering and high-availability configurations and specifically acknowledges in it's documentation that "all Highly Available solutions come with a trade-off between cost/complexity and uptime" (https://docs.gitlab.com/ee/administration/high_availability/README.html). GitLab is offered in a way which allows its customers to decide which of these trade-offs are best for their specific implementation.

Storage

As with any system providing file services to users, GitLab might eventually require expansion of attached volumes as usage increases. The process for expanding storage volumes will vary based on implementation. What shouldn't be missed here is that GitLab offers to be installed in many different types of configurations in the first place (https://docs.gitlab.com/ee/administration/high_availability/README.html). GitLab believes that this flexibility serves our customers well by enabling them to run GitLab the way that works best for them and the resources they have. For example, GitLab can be run on a single VM, in which case the effort to expand the repository storage space is very simple. But GitLab can also be installed in ways that leverage the scalability and redundancy capabilities of technologies such as AWS, GCP, Kubernetes, etc and expanding repository storage space could vary based on this.

While we're talking about repository storage consumption, it's also worth pointing out that GitLab offers tools to enable proactive management of storage con-

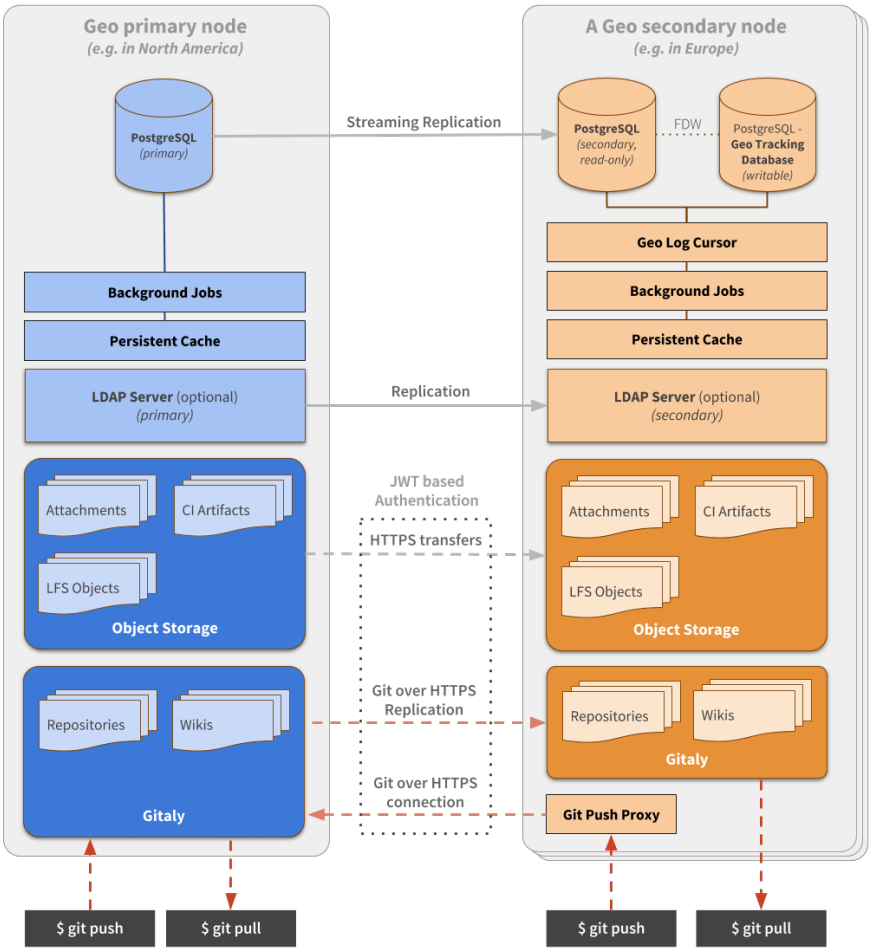
sumption to help use storage investments as efficiently as possible. Some examples are the ability to put limits on repo sizes (https://docs.gitlab.com/ee/user/admin_area/settings/account_and_limit_settings.html) and a repo clean-up utility which can properly remove old repo copies which have been archived elsewhere (https://docs.gitlab.com/ee/raketasks/cleanup.html).

Geo-Replication

When organizations have widely distributed teams, certain data-intensive actions (namely clones and fetches) can take a long time to execute due to network latency. For example, a developer based in Bangalore attempting to clone a repository from a Git server located in the US could experience performance issues. Customers who encounter this kind of latency often request what's known as geo-replication.

GitLab Geo-replication offering is called GitLab Geo and is available in Premium and Ultimate tiers. GitLab Geo replicates Git repository data and also large binary files (Git LFS) (https://about.gitlab.com/blog/2017/11/22/gitlab-10-2-released/#gitlab-geo-is-now-generally-available). GitLab Geo allows GitLab admins to set up read-only mirrors of the primary GitLab instance. This means that users can retrieve data from the replica servers, but commits still go to the primary instance, which are then replicated back to the mirrors.

The following illustration outlines how GitLab Geo works:



GitLab Geo requires end users and CI servers to manually configure a different URL matching the physically closest replica server. Introducing this feature to an existing setup will require changes to the development setups using the new server.

Replica servers have to talk to the main server to get user data for logins (API), to clone/pull from repositories if the ssh protocol is used and to retrieve LFS objects and attachments. As a consequence, GitLab Geo can speed up the last mile between developer machine and the replica server whose URL is used. When a developer accesses the web interface of a replica server, the UI reflects that write actions are not possible and the user is not viewing the primary instance.

High Availability

Some competitor's built in HA solutions use the warm failover strategy to address node failure. In this strategy a complete second instance of the server is configured and kept in sync using replication tools. Failing over involves switching the backup to primary. This warm failover strategy requires a doubling of the infrastructure requirements and complexity, but without receiving the benefit of the extra infrastructure, until a failover event occurs.

GitLab provides guidance in setting up HA in many different configurations, to be able to support our customer's various environments and resource constraints. GitLab supports two types of high availability configurations:

- An active-active configuration which splits web traffic across a number of servers, but requires separate HA configuration for the database, Redis, the file system, and ElasticSearch. This solution trades off simpler configuration for higher flexibility, and is recommended to support large installations.
- An active-passive configuration is also available where all components are installed on a single box.

GitLab recommends the use of DRBD when using an active-passive HA configuration because it works well and is a well known and tested open source replication solution which has been a part of the Linux kernel since 2009. GitLab opts for open source, transparent, non-proprietary technology to solve it's customer's problems. We believe that if you are going to trust your code and hence business to our software, that you should be able to see it completely, and even contribute to it. In contrast, the replication utilities provided by some competitors are closed source and so not available to be openly looked at. Which would you rather trust this all important service to?

Backup and Disaster Recovery

GitLab comes with built-in backup and recovery tools which are simple to run (https://docs.gitlab.com/ee/raketasks/backup_restore.html). To enable flexible, scalable, and resilient installations GitLab can be configured in more complex ways than some other competitors. GitLab backup procedures vary based on the complexity of the deployment configuration.

GitLab backup utilities default to retrieving the entirety of each data store for each backup, but can be set to retrieve a subset based on data type. There are several ways in which backups can be done depending on the specific configuration (GitLab backup utilities, EBS snapshots, LVM snapshots, etc). None of these methods requires system downtime (https://docs.gitlab.com/ee/raketasks/backup_restore.html).

GitLab backup utilities have built-in capability to stream backups directly to cloud or other remote storage (https://docs.gitlab.com/ee/raketasks/backup_restore.html#uploading-backups-to-a-remote-cloud-storage). Some competitors do include more sophisticated backup utilities with capabilities such as incremental backups, managing multiple snapshots, and deduplication.

Upgrades and Patching

GitLab releases monthly functional upgrades. Patch releases usually only include bug fixes and are only done for the current stable release.

The GitLab upgrade process is extremely simple in a single server configuration when using either the Docker Compose (https://docs.gitlab.com/omnibus/docker/README.html#update-gitlab-using-docker-compose) or Omnibus (https://docs.gitlab.com/omnibus/update/README.html#updating-methods) installation methods. In this scenario, simply download the new release and as admin run a single command to apply the upgrade.

GitLab's higher frequency of releases means that new capabilities, fixes, and customer value is made available to customers 3 times faster than with competitors. This does not mean that customer's need to apply the new versions at the higher release frequency. However, should GitLab customers decide to upgrade at the release frequency, GitLab has made it very simple to execute major, minor, and patch updates and upgrades as zero downtime upgrades. (https://docs.gitlab.com/ee/update/README.html#upgrading-without-downtime and https://docs.gitlab.com/omnibus/update/README.html#zero-downtime-updates).

The upgrade path for GitLab varies depending upon deployment. Rolling upgrades to the app servers are possible with the active/active deployment, but only if there are no database schema changes. The GitLab Docker and Omnibus packaging includes upgrading of all necessary system components for a functional GitLab application. Like with ANY deployment, externally integrated applications need to be managed separately.

Upgrading a single-machine deployment using the Omnibus installer is straightforward and can be done with zero downtime (https://docs.gitlab.com/ee/update/README.html#upgrading-without-downtime and https://docs.gitlab.com/omnibus/update/README.html#zero-downtime-updates).

As with other competitors, upgrading the Docker deployment requires downtime to stop the container, remove it, pull in the new container, launch it, and then let it run the necessary upgrade process after launch. Some competitors refer to this as "maintenance mode" and state it is not downtime, but during this time the system is not available to users.

GitLab offers live upgrade assistance as part of their Premium and Ultimate paid tiers. Live assistance is not necessary for regular GitLab Premium or Ultimate upgrades, but it is at least available should a customer have questions, need

assistance, or just want GitLab to partner with them to boost their confidence. Offering this assistance illustrates GitLab's philosophy of partnering with its customers to make sure that they have a good experience at every turn, including during upgrades.

Search

GitLab comes with basic search built-in. Advanced search functionality can be added by installing an included but separate ElasticSearch deployment, which customers must install, configure, and maintain separately from their GitLab instance. However, the installation can be simplified by using different available deployment techniques of ElasticSearch, such as Docker appliances (https://www.docker.elastic.co/) or the Amazon AWS ElasticSearch offering (https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-gsg.html).

GitLab search capabilities when using ElasticSearch are quiet comprehensive, and comparative to other competitor's (https://docs.gitlab.com/ee/user/search/advanced_global_search.html). However, when GitLab community edition is used, search requests cannot span multiple Git repositories.

Adoption and Integrations

Quote from a competitor, "Developers time is very valuable, why force them to context switch between different tools? GitLab couldn't have said it better ourselves! But not just for developers, for EVERYONE involved in delivering software changes. That's why GitLab is one single application which covers user needs all the way from planning to monitoring. Not only does having all this in one application increase collaboration between teams and reduce friction in delivering, but it also means less time integrating separate tools. GitHub, the top code collaboration website, is based on Git and has commendably taught millions of developers how to use Git. GitLab is also based on Git, and has a similar user experience to GitHub, yet adds a lot functionality that is desired by enterprises self-hosting a Git repository (https://about.gitlab.com/devops-tools/github-vs-gitlab.html). This is why more than 66% of enterprises self-hosting a Git repository have chosen GitLab (https://about.gitlab.com/blog/2017/06/29/whats-next-for-gitlab-ci).

One competitor in this space proudly states that they "choose to focus on the core aspects of the developer experience". This is a key difference between the GitLab philosophy and everybody else. While we agree that the developer experience is important, GitLab is focused on a great experience for the entire DevOps team, from developers, to QA, to Security, and Ops. Further, GitLab is doing it in one single application that everyone can collaborate through, rather

than setting up a platform and then expecting everyone to piece the integrations together to achieve what they need.

Forrester Research states that "Today's application delivery tool chain is complex and fragmented" (https://www.forrester.com/report/The+Quest+For+SpeedPlusQuality+Drives+Agile+And+DevOps+Tool+Selection/-/E-RES122555 - pay wall). While GitLab agrees that sometimes special tools are needed (and we can integrate with them), we also see that the majority of what DevOps teams spend their time integrating to their platform are common tools with common capabilities. GitLab strives to eliminate this waste and provide teams with the majority of what they need, out of the box, with advanced collaboration capabilities which typically would have to be created/customized by dev teams when they are piecing a solution together from disparate tools.

User Experience

GitLab is a complete DevOps platform, delivered as a single application. This means that a user can do everything from planning, coding, building, testing, securing, packaging, releasing, configuring, to monitoring, all from the single GitLab UI. This enables being in context all the time, not having to learn different tools and logins to those tools, and being able to concurrently collaborate with the team about every aspect of the work being done. Working in this manner leads to higher efficiency and more effective output. The loss of efficiency from context switching to different tools while working should not be underestimated.

In online discussions and forums about GitLab or development tools you can find users praising the GitLab UI and product in general. This is natural praise from customers who love GitLab. 2,000+ community members are active contributors to the code base because they believe in what GitLab stands for and want to help achieve it. These are all users who love their experience with GitLab.

Administration Experience

Administering self-managed GitLab is by no means a lonely experience. Due to the large number of organizations running GitLab self-managed (over 66% of self-managed git repositories) there is a large community of administrators who can help each other. Additionally, because GitLab is open source and open core, administrators can easily gain full access to the innards of the system as necessary, and can make changes to accommodate their needs, hopefully sharing back their changes. GitLab has a very healthy community with over 2,000 active contributors.

GitLab provides the flexibility of deployment configuration which enables it to be fitted into a wide variety of circumstances. A GitLab configuration can

range from a single simple to maintain appliance-like VM installation to a fully redundant, multi-system, HA, configuration which takes more administrative overhead.

When considering the administrative experience a key consideration is that one deployment of GitLab, in any configuration, has the potential to replace almost all of the tools across the DevOps lifecycle, eliminating the typical integration nightmares and significant maintenance efforts. GitLab provides the necessary capabilities across the DevOps lifecycle without requiring administrators to manage multiple integration points, and to suffer endless upgrades of each point tool.

Enterprise Partner

GitLab exists to make it so everyone can contribute. When everyone can contribute, then we have the highest potential for greatness. We believe this to our core and it reflects in everything we do. GitLab is not a company simply selling a piece of software. Our goal is to enable organizations to deliver better software (and ultimately, customer value) faster, and we do this by using the power of Concurrent DevOps.

At GitLab, one of our most fundamental values is transparency. At GitLab we have nothing to hide. We are proud of how we operate, what we do, and how we act. Our company handbook is publicly available and open for contributions from anyone, as is all of our product code, our issue tracking, development discussions, and roadmap. We invite the community to participate, and 2,000 of our community members from around the world do.

GitLab is used by over 66% of the companies self-hosting git repositories. GitLab.com is SOC 2 Type 1 certified - validating our compliance to strict information security policies and procedures, encompassing the security, availability, processing, integrity, and confidentiality of your data stored in the cloud. There are over 100,000 organizations that count on GitLab to accelerate their software delivery cycles, including top Fortune 500 financial, retail, telecom, and technology companies. GitLab is also trusted by many public sector and defense industry organizations (for more details see https://about.gitlab.com/customers/). GitLab is well funded, has world-wide sales and support organizations, and is ready to help.