GitLab and the USDS Playbook:

Delivering on the Promise of Digital Services

- How to use Agile to create a better CX
- How to accelerate the SDLC without compromising security
- Resources, references, tools & links

GitLab

Table of Contents

List of Figures

Introduction

In 2014, the United States Digital Service (USDS), a branch of the White House, published the Digital Services Playbook. It was created to provide a guide of 13 “key ‘plays” drawn from successful practices from the private sector and Government that, if followed collectively, will help Government build effective digital services.

The plays are:

1. Understand what people need
2. Address the whole experience, from start to finish
3. Make it simple and intuitive
4. Build the service using agile and iterative practices
5. Structure budgets and contracts to support delivery
6. Assign one leader and hold that person accountable
7. Bring in experienced teams
8. Choose a modern technology stack
9. Deploy in a flexible hosting environment
10. Automate testing and deployment
11. Manage security and privacy through reusable processes
12. Use data to drive decisions
13. Default to open

Agencies have found that the plays provide a reliable, repeatable, and tested approach to developing and delivering digital services. However, many agencies continue to struggle to successfully demonstrate improvement without the aid of data analysis from the tools they use for executing on these plays. In addition, some agencies discover one area of the organization is completely unaware of certain projects or that it has competing priorities that derail execution. The net result is often a delayed or misformed project that ultimately becomes an unhappy user or citizen experience.

GitLab, a complete DevSecOps platform delivered as a single application, enables Federal Government customers and contractors to leverage Digital Services plays in their IT projects. GitLab is designed around user-focused, Agile software development practices and supports a wide range of programming languages, frameworks, and tools. GitLab also provides industry leading CI/CD capabilities to allow developers to easily implement automated testing, security scanning, and deployment. In the following pages, we will discuss these capabilities and more in greater detail. GitLab furthers the vision of digital services by introducing planning and management aspects to the overall workflow.

Executing the USDS Plays

GitLab is committed to following Agile delivery practices and to leading and supporting our customers and community in delivering Agile services. We have built our business on the foundations of Agile transformation and we closely align to the principles proscribed in the USDS Digital Playbook. Using GitLab can enable agencies and contractors to follow the Playbook and to deliver better digital services faster.

Play #1: Understand what people need

"We must begin digital projects by exploring and pinpointing the needs of the people who will use the service, and the ways the service will fit into their lives. Whether the users are members of the public or government employees, policy makers must include real people in their design process from the beginning. The needs of people — not constraints of government structures or silos — should inform technical and design decisions. We need to continually test the products we build with real people to keep us honest about what is important."
— playbook.cio.gov/#play1

Continuous Quantitative and Qualitative User Input

GitLab is a complete DevOps platform for Agile development practices delivered as a single application. GitLab has been, since its inception, an open source product that allows anyone to contribute to the development of the software. With the support of our vibrant, contributing community, GitLab has embodied the approach of customer centricity by discovering what customers are *trying to accomplish*. We *capture* ***continuous*** *quantitative and qualitative input from, and collaborate* ***with****, current and prospective users of the platform* via Issues, Issue Boards, Epics, Milestones, and Product Roadmaps. Using these key components of the platform, Product owners are able to 1) *document findings about user goals, their needs* (and **why** they need what they say they need), *behaviors, and preferences,* 2) *test prototypes of solutions with real people, in the field,* and 3) *regularly test the solution with potential users to ensure it meets people's needs.*

Mapping Agile artifacts to GitLab features

Agile artifact	GitLab feature
User story	Issues
Task	Task lists
Epic	Epics
Points of estimation	Weights
Product backlog	Issue lists and prioritized labels
Sprint/iteration	Milestones
Burndown chart	Burndown charts
Agile board	Issue boards

FIGURE 1.1: MAPPING AGILE ARTIFACTS TO GITLAB FEATURES

An integral aspect of GitLab's culture is dogfooding, the practice of using our own product to conduct business every day and to further develop our product. Dogfooding enables GitLab to ensure the quality of the software product we produce and to continue to test it in a real-world environment. It also enables us to realize the benefits of operational efficiency derived from using a single application that does everything from project planning and source code management (SCM) to continuous integration/continuous delivery (CI/CD), monitoring, and security.

Key Findings and User Stories

Users of GitLab note that *documentation of key findings and user stories* collected in the research and collaboration process flow simply and easily—end-to-end—through the GitLab application, providing a single source of truth (SSOT). User stories are represented as Issues, which are addressed by design and development teams through Merge Requests (MRs)—the central tie between user stories and requirements—to design elements such as storyboards and code modifications, and then on to pipeline execution and CI/CD.

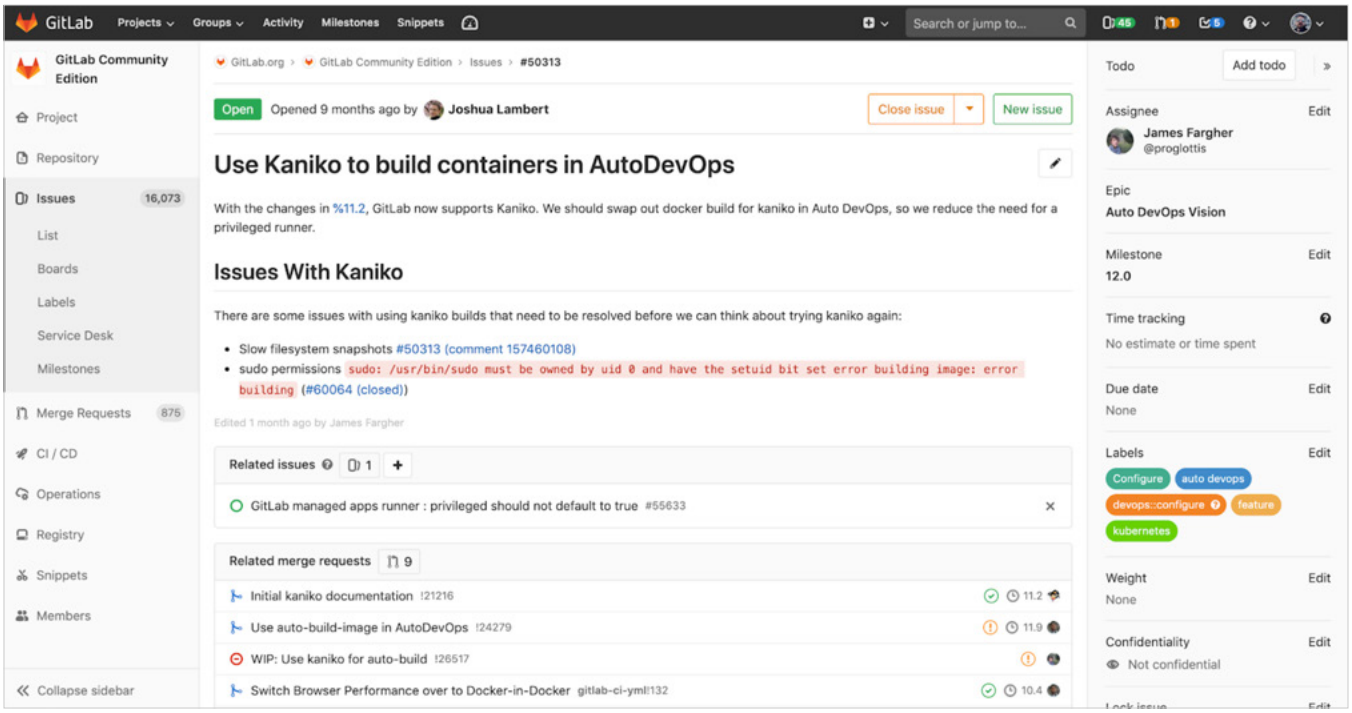


FIGURE 1.2: DEFINING USER STORIES AS ISSUES

Issues are collected and displayed in a Kanban-style board where users can arrange and rearrange them as work progresses.

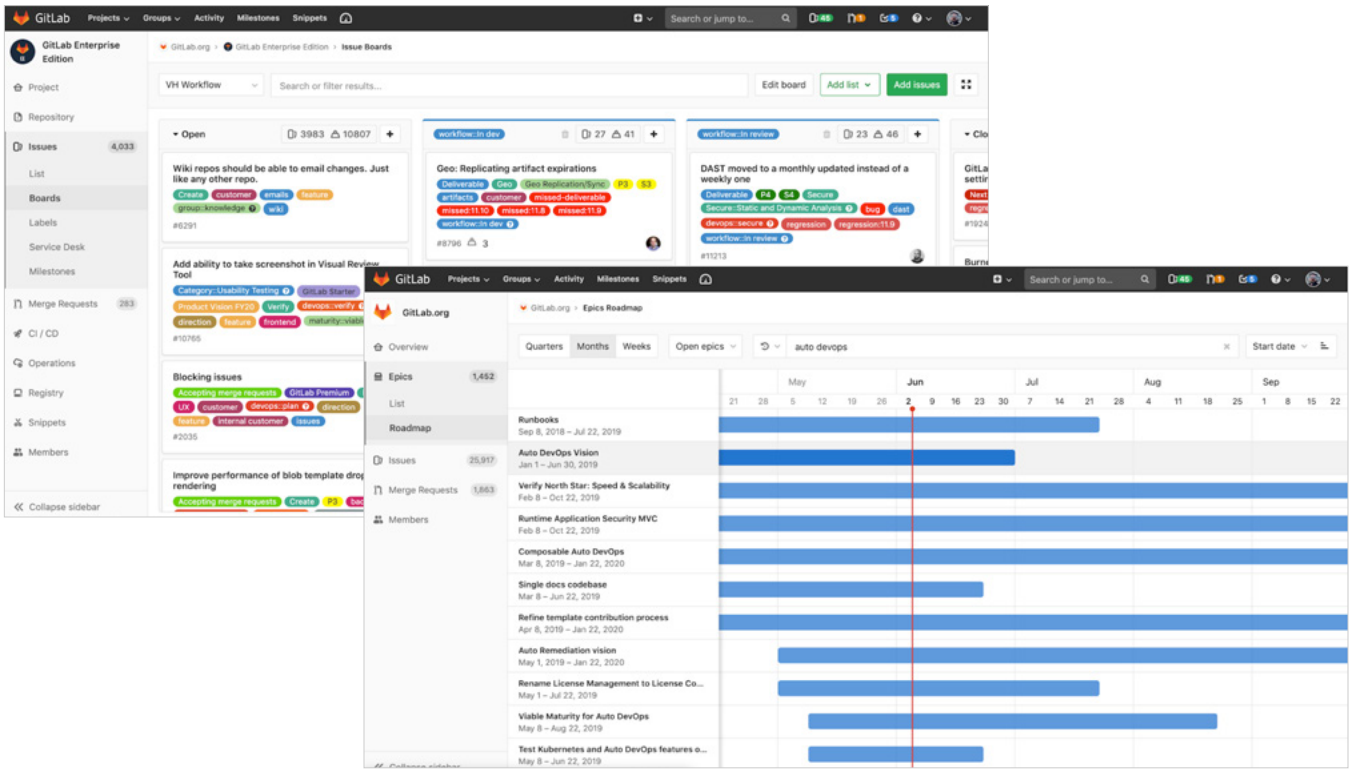


FIGURE 1.3: VISUALIZE AND MANAGE ISSUES, EPICS, ROADMAPS, AND MILESTONES WITH KANBAN ISSUE BOARDS, EPICS, AND PRODUCT ROADMAPS

The MR also provides a direct link to a running Review App so that the development team can quickly show what has been changed, with version control and MR history, to other team members and product owners for additional refinement.

Product Documentation

Product documentation is an essential part of GitLab. It is our goal to create documentation that is complete, accurate, and easy to use. We believe it should be easy to browse or search for the information you need, and easy to contribute to the documentation. GitLab's documentation is crafted to help users, admins, and decision-makers learn about GitLab features and to optimally implement and use GitLab to meet their DevOps needs. All documentation is published at docs.gitlab.com and at the **/help/** path on each GitLab instance's domain, including content for that instance's version and edition.

See: How to use GitLab for Agile software development: How Agile artifacts map to GitLab features and how an Agile iteration looks in GitLab

Play #2: Address the whole experience, from start to finish

"We need to understand the different ways people will interact with our services, including the actions they take online, through a mobile application, on a phone, or in person. Every encounter —whether it's online or offline—should move the user closer towards their goal."
— playbook.cio.gov/#play2

Addressing the Holistic User Experience

Using GitLab Epics, agencies can manage and decompose user stories to better address the entire user experience when developing digital services. For example, when planning a service application, the Product Owner should consider *the different points* and modalities *at which people will interact with the service—both online and in person.* Although each user interface ultimately relates back to the same digital service application, each interface may pull from different parts of the code and may be scheduled to be built and delivered separately and by different teams. In such cases, how can the entire delivery team communicate with one another and work iteratively to make the smallest change possible, while maintaining a holistic view of the application?

Tracking User Metrics End-to-End

At GitLab, we use Epics to better plan and track UX efforts over time. The term 'epic' is most commonly associated with Agile methodology. In Agile, an epic is a collection of user stories that describe a larger user flow, typically consisting of multiple features. So what does 'epic' mean at GitLab? Here, Epics contain a title and description, much like an Issue, and allow the user to attach multiple Issues, and even child Epics, to indicate hierarchy. In short, an Epic is a feature that allows the user to manage a portfolio of projects more efficiently and with less effort by tracking groups of Issues that share a theme, across projects, and milestones, thus enabling designers *to develop (and track) metrics that will measure how well the service is meeting user needs at each step of the service.*

Epics give UX teams an efficient way to plan, track, and execute a group of thematically related Issues. Each Issue, taken on its own, represents just one piece of a much bigger picture. Epics enable teams to define the goal and then organize Issues specific to that effort. Epics also give other departments better visibility into what UX considers important.

See: How the GitLab UX team uses Epics

Play #3: Make it simple and intuitive

"Using a government service shouldn't be stressful, confusing, or daunting. It's our job to build services that are simple and intuitive enough that users succeed the first time, unaided."
— playbook.cio.gov/#play3

Simple and Intuitive Workflows

With a "user-and-newbie-friendly" interface that is *simple and intuitive,* GitLab allows teams to work effectively, both from the command line and from the UI itself. It's not only useful for developers, but can also be integrated across the complete team to bring everyone into a single and unique platform. We know how important it is to *give users clear information about where they are in each step of the process,* so we developed GitLab workflow, a logical sequence of possible actions to be taken during the entire lifecycle of the software development process. The GitLab workflow logic is intuitive and predictable, making the platform easy-to-use and easier to adopt.

GitLab workflow facilitates improved team collaboration by accelerating ideas to production. GitLab Flow—the integration of Git workflow with our issue tracking system—unites code version management with project and deployment management tools in order to create the easiest way to work with Git. While development teams can continue to seamlessly use Git, all team members can adopt easy-to-use features to simplify workflows and releases, enabling everyone to participate in a DevOps transformation as an organization scales. All members of the team—including executives—have end-to-end visibility into the entire DevSecOps lifecycle.

- Common data model uniquely allows for insights across the entire DevSecOps lifecycle
- Configurable insights dashboard shows status of work items over time
- Cycle analytics data helps identify areas of improvement of cycle times
- Security insights provide a roll-up of vulnerabilities
- Program level insights help keep projects on-track

Enabling Accessibility

GitLab also supports and encourages designers and developers to *follow accessibility best practices to ensure all people can use the service.* We recommend the following testing and audit tools and materials for use with the GitLab platform:

- Chrome Accessibility Developer Tools are useful for testing for potential accessibility problems in GitLab
- The axe browser extension (available for Firefox and Chrome) is also a handy tool for running

audits and getting feedback on markup, CSS, and even potentially problematic color usages

» Accessibility best-practices and more in-depth information are available on the Audit Rules page for Chrome Accessibility Developer Tools

» The "awesome a11y" resource list is a useful compilation of accessibility-related material

See: GitLab's VPAT Conformance Report (Revised Section 508 Edition)

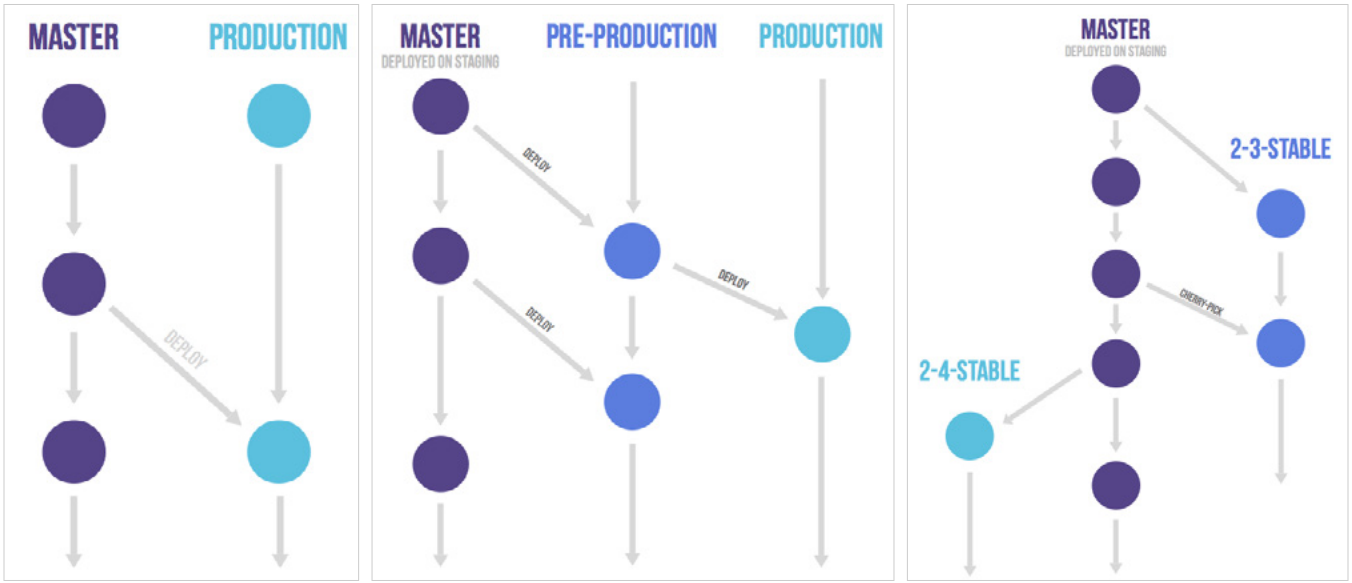
Play #4: Build the service using Agile and iterative practices

"We should use an incremental, fast-paced style of software development to reduce the risk of failure. We want to get working software into users' hands as early as possible to give the design and development team opportunities to adjust based on user feedback about the service. A critical capability is being able to automatically test and deploy the service so that new features can be added often and be put into production easily."

— playbook.cio.gov/#play4

Agile, Iterative Practices

Iterative development with early user feedback is critical to creating *simple and intuitive services (Play #3)* and are a core part of the GitLab approach to software development. Support for iterative, incremental development and early user feedback are built into every part of the application. As a complete DevSecOps platform, delivered as a single application, GitLab inherently facilitates collaboration between users and developers throughout the design and implementation process. For example, the *GitLab Review Apps* collaboration tool allows automatic live preview of changes made in a feature branch. Designers, product managers, testers, and end users can review changes prior to the branch being merged—and approve, comment, or reject changes as necessary. GitLab also includes support for Feature Flags, which enable flexible delivery of new features for controlled testing and allows for quick rollback, if necessary. As shown in the diagram below, GitLab Flow provides built-in, flexible structure and conversational development capability which enables many small iterations to quickly deliver software at lower risk.



Production branch **Environment branches** **Release branches**

FIGURE 4.1: GITLAB FLOW ENABLES ITERATIVE DEVELOPMENT WHILE LOWERING RISK

Minimum Viable Product

At GitLab, we believe in the Agile principle of *shipping a functioning "minimum viable product" (MVP) that solves a core user need as soon as possible.* In fact, we do the smallest thing possible and get it out as quickly as possible. We call these small things 'Minimum Viable Changes' (MVCs) and encourage them to be as small as possible. We always look to make the quickest change possible to improve the user's outcome. If we validate that the change adds more value than what is there now, then we do it. No need to wait for something more robust. This value applies to everything we do, in all functions, though specifically for product MVCs, *we ensure the individuals building* our product validate and *communicate closely* with customers that we're adding useful functionality without obvious bugs or usability Issues. GitLab Flow enables our Public Sector customers to follow this same approach.

Consistent, Rapid Releases

Releasing features and improvements multiple times each month is in GitLab's DNA. GitLab ships a non-trivial version **minor** release (when new backward-compatible functionality is introduced to the public API, a minor feature is introduced, or when a set of smaller features is rolled out) every single month on the 22nd. For significant changes, or when any backward-incompatible changes are introduced to the public API, we ship a **major** release. Major releases happen yearly. The last major release was GitLab 13.0 May 22, 2020. Subsequent major releases will be scheduled for May 22 each year, by default. And of course, we ship patches (for backward-compatible bug fixes that fix incorrect behavior or for critical security releases) whenever needed.

We have shipped a release/update every month, as of this writing (version 13.0), 100+ consecutive months and we deploy improvements to our website (where we house and maintain our standards of operations) more than 500 times per day. The major planned features on our roadmap can be found on our upcoming releases page.

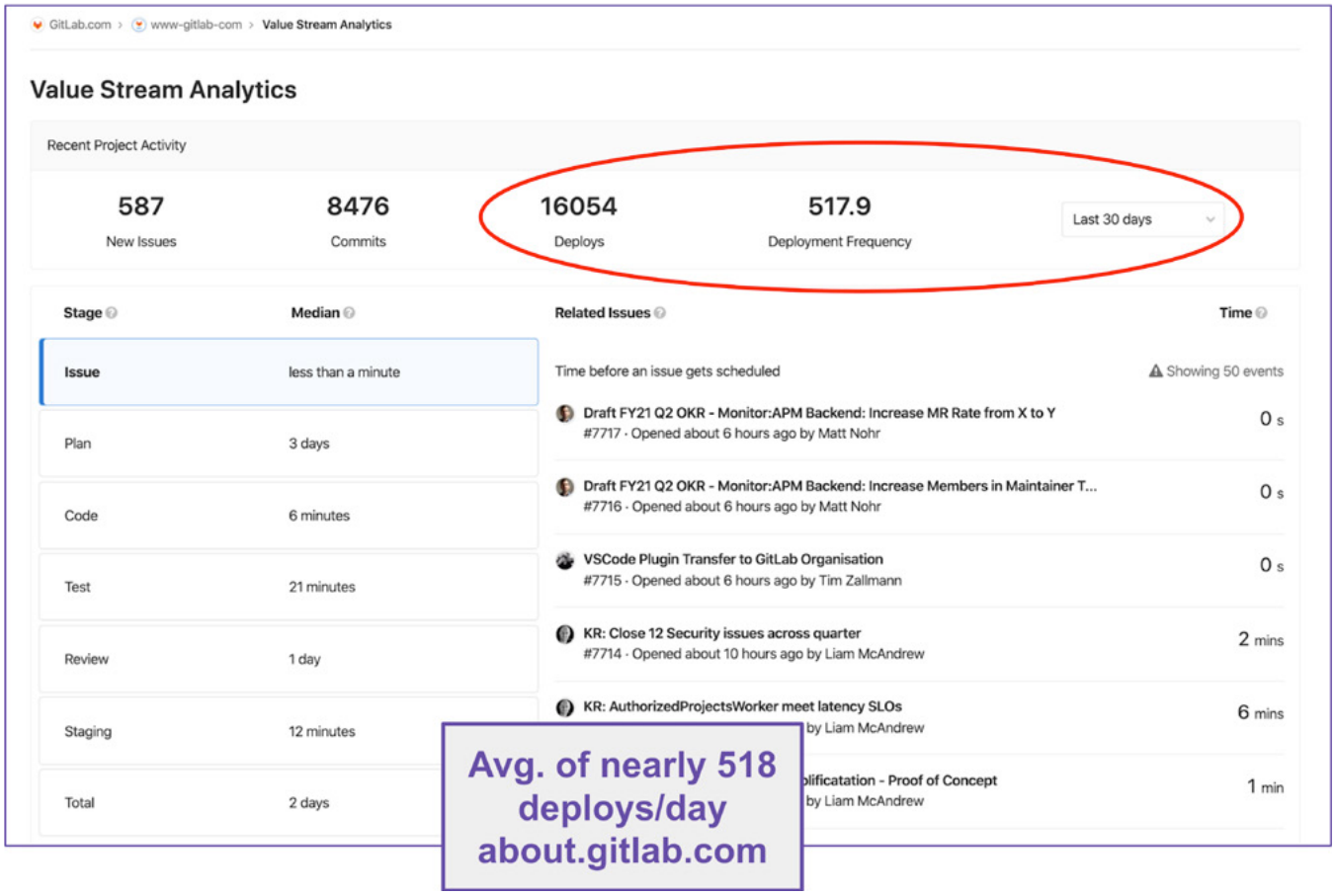


FIGURE 4.2: VIEW OF GITLAB'S MONTHLY WEBSITE IMPROVEMENT DEPLOYS (AS OF MAY 2020)

How We Help Our Customers Release Faster

We help our customers improve cycle time—the elapsed time between starting to work on an idea and delivering code to production—by streamlining developer tools and processes, enabling control, visibility, and collaboration, and integrating quality and security reviews and resolution throughout the SDLC. Streamlining tools and processes into a single application enables better security and quality without sacrificing speed to delivery because it minimizes the friction and handoffs between teams who can now collaborate at the point of code change instead of waiting for a sequential 'turn' to do their work. Teams can also see, measure, and report on productivity and code quality, which increase as they spend more time writing code and less time maintaining and waiting for their toolchain.

Shorter cycle time is beneficial because it enables teams to respond to changing needs faster, ship smaller changes (and better manage regressions, rollbacks and bugs because you're shipping smaller changes), and make more accurate predictions and forecasts. Perhaps the biggest benefit to a citizen or user experience-driven organization is that shorter cycle times directly translate to an increased focus on improving the experience because you're suddenly able to respond to their needs much more quickly.

Teams require end-to-end visibility and traceability of Issues throughout the software development lifecycle—from idea to production—in order to ship features at the speed users and citizens demand. An increase in visibility reduces silos and facilitates collaboration, ensuring that everyone is aware of what's going on and where they're needed. For example, we calculated the average time to deliver all the features in our product release 11.8 at 250 days. With an increase in visibility, we're able to see that some features we shipped took only 30 days, while others were in our backlog for three years. Since we're on a monthly release cadence, knowing that many of our features were started and delivered in 30 days helps us determine whether we're quickly delivering value to our customers and where we can improve.

When agencies experience long cycle times, they risk failure to meet delivering projects and programs on-time and within budget and ultimately, failure to meet their mission and mandates. Quickly delivering what users and citizens want requires a modernized SDLC that saves time, effort, and cost. A continuous delivery approach automates testing and deployment capabilities so that software can be developed and deployed rapidly and reliably.

Bug & Feature Tracking

When developing software, it's important to be able to be able to *create a prioritized list of features and bugs.* GitLab has a great Issue tracker (that can be used for, among other purposes, accepting feature proposals, questions, support requests, and bug reports) but you can also use an external one such as Jira, Redmine, YouTrack, Bugzilla, Marker.io or a Custom Issue Tracker. External Issue trackers are configurable per GitLab project. However, to enable an external Issue tracker, you must first configure the appropriate service. Bugs can then be managed and prioritized using *Issue Lists, Issue Boards, Issue References, and Epics*.

Source Code Management

Source code management (SCM) is where development team sharing and collaboration begins. *Source code version control* in GitLab gives *the entire project team access to the Issue tracker and*

version control system so they can share, collaborate, and maximize their productivity in a world-class SCM. Leveraging the industry-leading, open source, distributed version control system Git, GitLab makes developers more efficient, effective, and responsive. As part of the complete DevSecOps lifecycle, SCM ensures that there is a critical single source of truth (SSOT) for teams to manage their work.

Code Review

Code review, or engineers manually *reviewing code* as it is being developed, is one of several tools that organizations have available to maintain and *ensure code quality*. Having a clean codebase allows developers to quickly build new features, which comes in handy if you find yourself needing to react promptly to support your mission.

Since teams can feel stressed to skimp on code reviews due to deadline pressure, it's important to create a clear, repeatable process that becomes a habitual part of the workflow. In addition to the business benefits of keeping code quality high, a robust code review process also carries benefits for productivity, like helping share technical knowledge across teams, and looping in the right people during the process instead of at the end. With a tool like GitLab, where the intuitive user interface (UI) and discussion features make it easy for non-technical people to contribute, code review can also become a mechanism for uniting stakeholders in your organization.

While using GitLab to build GitLab, we apply a quality-conscious mindset throughout the development process, sharing the responsibility among everyone instead of seeing review as an obstacle at the end. We believe that this approach, which doesn't segregate code review to the end of the process to entirely burden the reviewer, places responsibility on all of us. If something is broken, it is because we all failed collectively, so we do a retrospective and think about how to fix it. In a very concrete way, code reviews help enforce conventions, consistency, and technical standards among your team. Newer team members can accelerate their education about the product by creating code and then reviewing it critically. A collaborative code review process helps engineering and product teams work closely together to make the best decisions.

Automated Test and Deploy

Continuous Integration is the practice of integrating code into a shared repository and building/testing each change automatically, as early as possible—usually several times a day. **Continuous Delivery** is a software engineering approach in which continuous integration, automated testing, and automated deployment capabilities allow software to be developed and deployed rapidly, reliably, and repeatedly with minimal human intervention. Software can be released to production at any time, often by automatically pushing changes to a staging system. The benefits of continuous delivery are

clear: Developers say 'product/project managers are 25% more likely to have a better sense of Dev team capacity in a CD organization than in a company that deploys between once a month and once every six months. And 47% agree those same managers are in a better position to accurately plan and scope features in a CD environment'. (*2019 GitLab Developer Survey*)

With **Continuous Deployment**, every code change goes through the entire pipeline and is put into production automatically, resulting in many production deployments every day. It does everything that Continuous Delivery does, but the process is fully automated; there's no human intervention at all. GitLab recognizes the criticality of being able to *automatically test and deploy a service so that new features can be added often and be put into production easily.* For example, our website, **about.GitLab.com**, is continuously deployed. We commit hundreds of times a day to feature-branches, and every push triggers a parallel test and build. Every time we merge to the master branch (and we do that a lot, every day), the code is tested, built, and deployed to the production environment, passing through the entire pipeline. There's no further manual action that triggers the deployment: it is an automated process, controlled by GitLab CI.

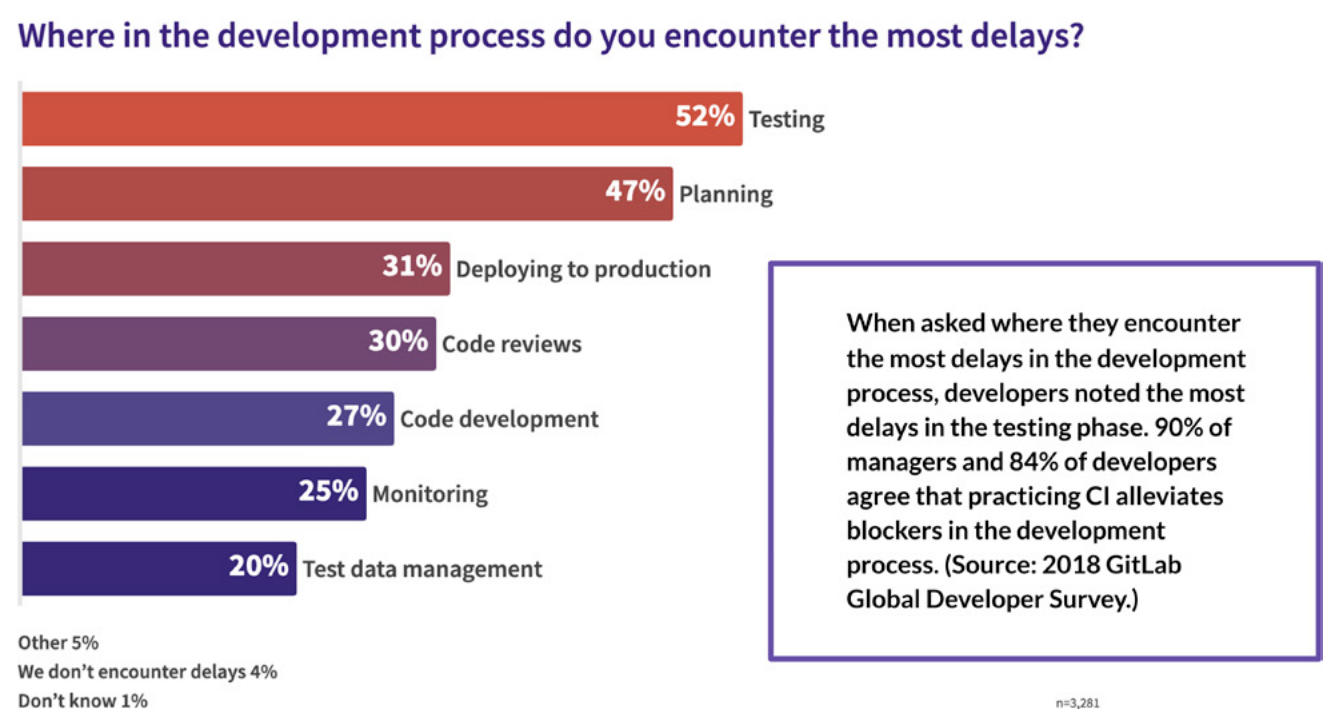


FIGURE 4.3: WHERE THE MOST DELAYS OCCUR IN THE DEVELOPMENT PROCESS

GitLab immediately executes the CI pipeline process to take the code, build the application, automatically run test (Functional as well as Security) and deploy into the designated environment (Development, Test, Canary, or Production). The CI pipeline executes as soon as the code is committed to the source repository, automatically, every time. This enables a continuous and consistent process.

> **Rated #1 in the Forrester CI Wave™**
>
> **"GitLab supports development teams with a well-rounded installation and configuration process, an easy-to-follow UI, and a flexible per-seat pricing model that supports self service. GitLab's vision is to serve enterprise-scale, integrated software development teams that want to spend more time writing code and less time maintaining their tool chain"**
>
> *— FORRESTER CI WAVE™*

GitLab CI/CD is an integrated, open source, seamless and scalable approach that provides multi-language, multi-platform support in a single application. GitLab CI/CD accelerates how quickly your team can deliver results for your customers and stakeholders. CI helps you catch and reduce bugs early in the development cycle and CD moves verified code to your applications faster.

Refer to: about.gitlab.com/2018/03/05/gitlab-for-agile-software-development
See: Code Review Guidelines (Advice & Best Practices)

Play #5: Structure budgets and contracts to support delivery

"To improve our chances of success when contracting out development work, we need to work with experienced budgeting and contracting officers. In cases where we use third parties to help build a service, a well-defined contract can facilitate good development practices like conducting a research and prototyping phase, refining product requirements as the service is built, evaluating open source alternatives, ensuring frequent delivery milestones, and allowing the flexibility to purchase cloud computing resources.

The TechFAR Handbook provides a detailed explanation of the flexibilities in the Federal Acquisition Regulation (FAR) that can help agencies implement this play."
— playbook.cio.gov/#play5

Open Source

GitLab has an open core business model and ships both open and closed source software. Our open core development model allows anyone to contribute to the functionality of the product. This uniquely transparent product development process enables us to engage customers, partners, and a community of 100,000+ organizations and millions of users to harness *open source* innovations within the product experience.

The GitLab open core model allows our users to adopt better tools faster through open source innovation, attract and retain better talent (developers love GitLab!), and reduce costs because multiple customers can contribute code to solve similar problems. Organizations also lower risk since they:

- Are not locked into feature sets we choose to build and have the flexibility to add to the product as needed
- Enjoy a product that is widely used and adopted at internet scale
- Avoid technology obsolescence because of our vibrant community of contributors
- Can receive access for product and technical support as needed

Flexible Cloud Hosting

Applications developed on GitLab may be deployed to any commercial cloud computing provider, government cloud environment, or datacenter using GitLab CI/CD pipelines, which allows organizations *the flexibility to purchase cloud computing resources as needed.* This allows agencies to choose the hosting environment that provides best value.

Flexible Pricing Models

GitLab has established a flexible per-user/per-year pricing model that enables the Government to pay for only what is needed and reduces up-front costs, so agencies can use *tools, services, and hosting* that provide a *variety of pricing models.* GitLab may be purchased as a SaaS offering and run in government authorized cloud environments, which eliminates fixed infrastructure spending, or run on-premises. GitLab is available via multiple types of contract vehicles, including but not limited to the GSA Schedule, SEWP, and state & local contracts.

Ensuring Delivery Milestones

Milestones in GitLab are a great way to track Issues and Merge Requests created to *ensure frequent delivery milestones* in a designated period of time. Milestones allow you to organize Issues and Merge Requests into a cohesive group, with an optional start date and an optional due date.

Milestones can be used as Agile sprints. Simply set the Milestone start date and due date to represent the start and end of your Agile sprint, set the Milestone title to the name of your Agile sprint (such as November 2019 sprint) then add an Issue to your Agile sprint by associating the Milestone to the Issue. Milestones can also be used as releases. Set the Milestone due date to represent the release date of your release (leaving the Milestone start date blank), set the milestone title to the version of your release (such as Version 9.4), then add an Issue to your release by associating the Milestone to the Issue. Visibility into delivery progress and the amount of work still remaining on a project or group of projects is available via *Project* and *Group Burndown Charts*.

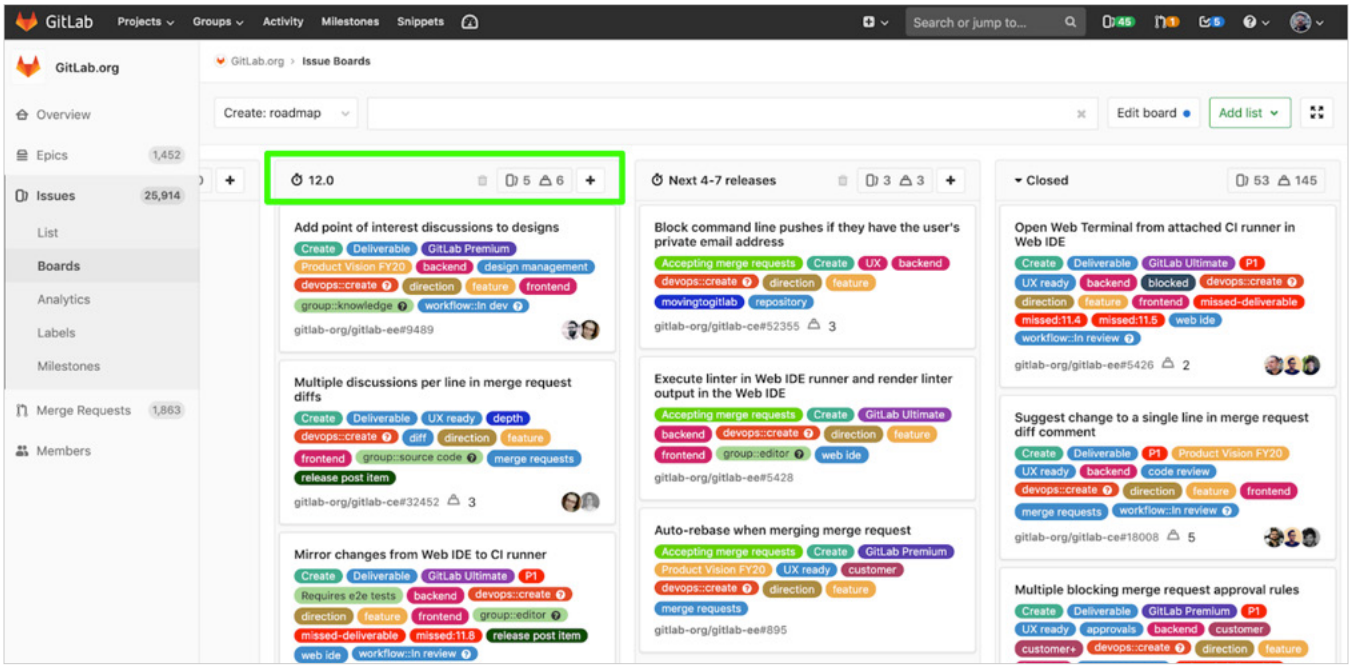


FIGURE 5.1: PLAN, MANAGE, AND TRACK MILESTONE DELIVERY WITH GROUP ISSUE BOARDS

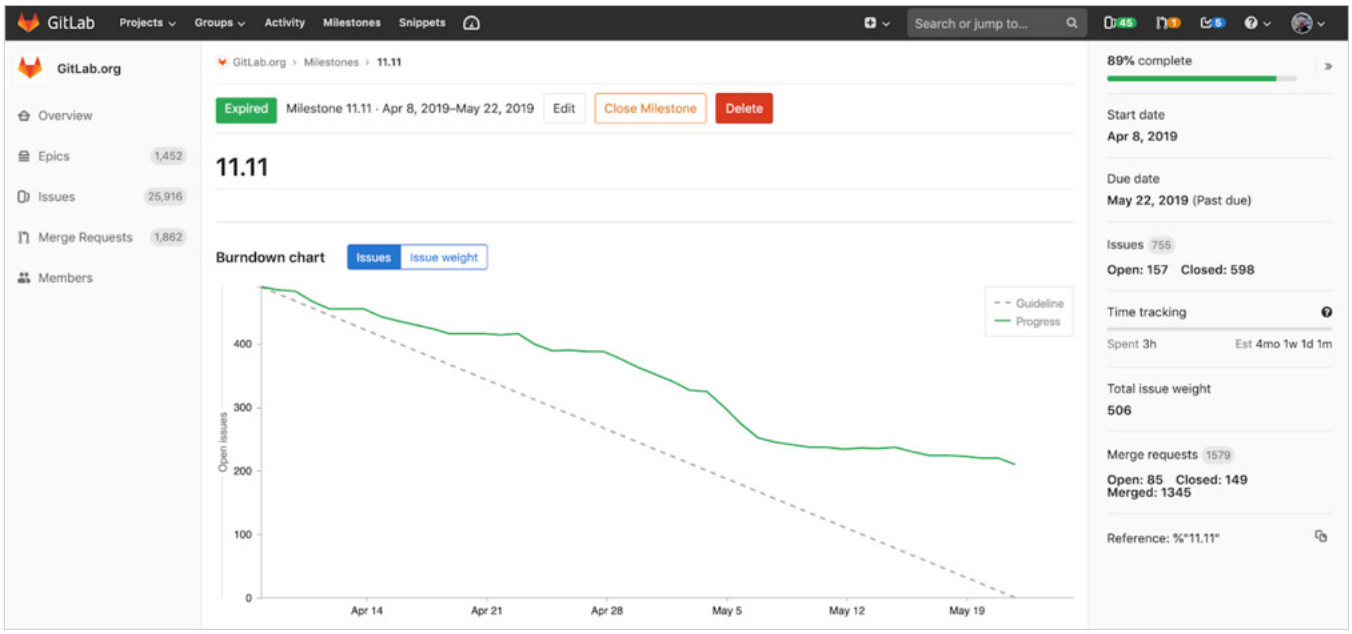


FIGURE 5.2: BURNDOWN CHARTS FOR MILESTONES PROVIDE PROGRESS-TRACKING AGAINST THE BACKLOG FOR THE MILESTONE

Feature Prioritization & Delivery Schedule Evolution

Product or business owners typically create user stories to reflect the needs of the organization and customers, *adjusting feature prioritization and delivery schedule as the project evolves.* User stories (Issues) are prioritized in a product backlog to capture urgency and desired order of development. The product owner communicates with stakeholders to determine the priorities and constantly refines the backlog. In GitLab, there are dynamically generated Issue lists which users can view to track their backlog. Labels can be created and assigned to individual Issues, which then allows you to filter the Issue lists by a single label or multiple labels. This allows for further flexibility. Priority labels can even be used to also order the Issues in those lists.

Play #6: Assign one leader and hold that person accountable

"There must be a single product owner who has the authority and responsibility to assign tasks and work elements; make business, product, and technical decisions; and be accountable for the success or failure of the overall service. This product owner is ultimately responsible for how well the service meets the needs of its users, which is how a service should be evaluated. The product owner is responsible for ensuring that features are built and managing the feature and bug backlogs."

— playbook.cio.gov/#play6

Accountable Leadership

As an operational step, having one empowered, responsible leader who owns the final deliverable is one of the most critical aspects outlined in the USDS Digital Playbook. *Assigning one leader and holding them accountable* is addressed by providing the designated owner the appropriate visibility into the project. As a single source of truth (SSOT), GitLab provides the complete visibility and traceability of who did what and when and exactly what was delivered and deployed. And in true DevSecOps fashion, any operational feedback goes right back to that individual who can then prioritize work and remediation appropriately.

Product or business owners can create user stories (GitLab Issues) to reflect the needs of their project, program, or customers. Those Issues are prioritized in a product backlog to capture urgency and desired order of development. The product owner communicates with stakeholders to determine the priorities, constantly refining the backlog, and *has the authority to assign tasks and make decisions about features and technical implementation details* as needed. The product owner and the development team can then meet to decide work that is in scope for the upcoming sprint, including Issues into that sprint by assigning them to that particular Milestone and assigning not-in-scope work into backlog.

Owners are essentially group-admins on GitLab. They can give access to groups and have destructive capabilities and workflow-enforcing permissions.

Play #7: Bring in experienced teams

"We need talented people working in government who have experience creating modern digital services. This includes bringing in seasoned product managers, engineers, and designers. When outside help is needed, our teams should work with contracting officers who understand how to evaluate third-party technical competency so our teams can be paired with contractors who are good at both building and delivering effective digital services. The makeup and experience requirements of the team will vary depending on the scope of the project."
— playbook.cio.gov/#play7

Attracting & Retaining Experienced Teams

Great software is built by great people—not "magic bullet" tools or technologies. However, research reveals that 'the tools you choose for your team may have a greater impact on developer happiness and retention that you thought. In fact, a whopping 81% of developers say that it's critical organizations use the latest development tools and 36% go as far as to say they would reject a job if the employer didn't use the latest tools.'

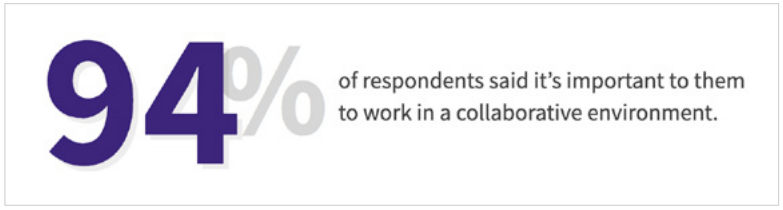
In the 2018 GitLab Global Developer Report, we heard from more than 5,000 software developers, architects, IT managers, and executives from around the world, *experienced with modern development and operations (DevOps) techniques like continuous integration and continuous deployment,* and uncovered the impact of workplace culture, workflow, and tooling on the needs, preferences, and satisfaction of today's developer. We confirmed that high-performing teams are more likely to report having a strong DevOps culture, resulting in better visibility and collaboration than their lower-functioning counterparts. High-performing teams also report having better access to the best development tools, spend less time context switching, and are more likely to be using cloud-based tools. For leaders of high-performing teams, automating the software development lifecycle (SDLC) is a top priority.

FIGURE 7.1: ORGANIZATIONS' TOP TECHNOLOGY INVESTMENTS, CHALLENGES, AND TOOLS IN 2018

Moving forward, IT organizations that successfully adopt continuous improvement practices and seamless automation across their SLDC will have happier, more collaborative, and better functioning teams who are well-positioned to meet their goals and objectives.

GitLab is leading the next advancement of DevSecOps. Built on open source, GitLab leverages contributions from a growing, passionate, global community of thousands of developers and millions of users, representing more than 100,000 of the world's most demanding organizations.

Collaboration: The Key to Effective Teams



Assembling an experienced team is only effective if team members communicate with each other consistently and collaboratively. Survey respondents place a strong value on working in a collaborative environment, where they feel their voices are heard and they have the ability to make decisions about their work. Teams should move forward towards a single purpose, a single mission of making the digital services and applications as simple and intuitive as possible for citizens and users. Effective communication is possible if the teams use collaboration tools such as GitLab Issues and ChatOps software such as Mattermost. These tools enable communication among team members while integrating and automating repetitive processes into GitLab.

Automated Testing Frameworks

Development teams—and their managers—want to be able to automatically test code in a live production-like environment. They know manual testing reduces application development speed and threatens code quality. These two disadvantages can be project and program killers, especially in such a mission-driven development landscape. Test automation also makes it possible for testers to use their skills where they add more mission value: Creating great, seamless user experiences.

One advantage of using a platform like GitLab is that we provide a variety of continuous testing solutions. End-to-end testing can be used to check whether your application works as expected across the entire software stack and architecture, including integration of all micro-services and components that are supposed to work together. We use Omnibus GitLab to build GitLab packages and then we test these packages using the GitLab QA orchestrator tool, which is a black-box testing framework for the API and the UI. We run scheduled pipelines each night to test nightly builds created by Omnibus. (You can find these nightly pipelines at gitlab-org/quality/nightly/pipelines.) We also run scheduled pipelines each night to test staging. (You can find these nightly pipelines at gitlab-org/quality/staging/pipelines.) There are two main options for running build tests. If you simply want to run the existing tests against a live GitLab instance or against a pre-built Docker image, you can use the GitLab QA Orchestrator with very little effort. However, if you would like to run against a local development GitLab environment, you can use the GitLab Development Kit (GDK). (Refer to the instructions in the QA README.) In order to write new tests, you'll first need to learn more about GitLab QA Architecture.

In addition, customers have integrated us with SaaS-based testing solutions or even their own homegrown Selenium grids. GitLab also integrates with JavaScript platforms like Cypress.io to help teams create continuous integration test pipelines.

Securing Digital Services

Software professionals often hear about the imperative to undergo digital transformation, the importance of writing secure code, the need to increase visibility, and the urgency to reduce cycle time. Security—even for those with *experience securing digital services*—is perhaps the most polarizing aspect of software development today and that's clear from the results of our 2019 GitLab Developer Survey. Nearly 70% of developers said they are 'expected to write secure code, but the mechanisms to make that happen often remain elusive.'

However, balancing speed-to-mission with security is possible. With GitLab, DevSecOps architecture is built into the CI/CD process. DevSecOps integrates security best practices in the DevOps workflow and automates security workflows to create an adaptable process for your development and security

teams. Every Merge Request is scanned through its pipeline for vulnerabilities in your code and its dependencies. This enables some magic to happen:

- Every piece of code is tested upon commit, without incremental cost
- The developer can remediate now, while they are still working in that code, or create an Issue with one click
- The dashboard for the security pro is a roll-up of remaining vulnerabilities that the developer did not resolve on their own
- Vulnerabilities can be efficiently captured as a by-product of software development
- A single tool also reduces cost over the approach to buy, integrate, and maintain point solutions

Unlike traditional application security tools primarily intended for use by security pros, GitLab secure capabilities are built into the CI/CD workflows, where the developers live. We empower developers to identify vulnerabilities and remove them early, while at the same time, providing security pros a dashboard to view items not already resolved by the developer, across projects. Developers immediately see the cause and effects of their own specific changes so they can iteratively address security flaws alongside code flaws. When using GitLab, no additional integration is needed between app sec and ticketing, CI/CD, etc. and auto remediation applies patches to vulnerable dependencies, even re-running the pipeline to evaluate the viability of the patch.

GitLab secure capabilities include:

- **Static Application Security Testing (SAST):** Prevents vulnerabilities early in the development process, allowing code to be fixed before deployment
- **Dynamic Application Security Testing (DAST):** Once code is deployed, it prevents exposure to your application from a new set of possible attacks as you are running your web applications
- **Dependency Scanning:** Automatically finds security vulnerabilities in your dependencies while you are developing and testing your applications, such as when you are using an external (open source) library with known vulnerabilities
- **Container Scanning:** Analyzes your container images for known vulnerabilities
- **Auto Remediation:** Will aim to automatically detect, analyze, fix, and remove security vulnerabilities affecting apps
- **Secret Detection:** Scans each commit for secrets within SAST that need to be protected
- **IAST and Fuzzing:** Includes future features GitLab will add to its Security capabilities, see the visions for IAST and Fuzzing

GitLab gives all stakeholders the real-time visibility, efficiency, and governance needed to ship secure code at the speed the Government demands. For security professionals, DevSecOps embeds automated security, code quality, vulnerability management, and policy enforcement across the SDLC to keep things moving quickly while remaining compliant. Every important activity is logged in a single audit log that covers the entire DevSecOps lifecycle—always on, accessible, and accurate—to allow tight control over how code is deployed, eliminating guesswork.

Play #8: Choose a modern technology stack

"The technology decisions we make need to enable development teams to work efficiently and enable services to scale easily and cost-effectively. Our choices for hosting infrastructure, databases, software frameworks, programming languages and the rest of the technology stack should seek to avoid vendor lock-in and match what successful modern consumer and enterprise software companies would choose today. In particular, digital services teams should consider using open source, cloud-based, and commodity solutions across the technology stack, because of their widespread adoption and support by successful consumer and enterprise technology companies in the private sector."
— playbook.cio.gov/#play8

GitLab prides itself as an independent product that integrates with many large software and cloud infrastructure providers. Because of this, and the open source origins of the product, GitLab has capabilities to adopt and adapt to various technologies. Our open source contributions and community and customer feedback remain active and dynamic. This continued interactive flow has provided GitLab a unique ability to grow with DoD, IC, and Civilian agency customers and become the cornerstone of their software development strategy, *enabling development teams to work efficiently.* GitLab continues to evolve by delivering new capabilities on a monthly release frequency (one of the highest frequencies in the industry), with a large number of features delivered per release.

Scaling Easily and Cost-Effectively

In a DevOps culture, IT and development teams seamlessly collaborate to increase delivery and productivity. Two of the greatest obstacles organizations face when implementing a DevOps approach are adapting principles to address the unique challenges of scaling and *enabling services to scale easily and cost-effectively.*

Fortunately, GitLab has been designed to be easy-to-deploy and maintain. A single system configuration can scale to support more than 32,000 active users, requiring minimal administration effort, and a multi-node, distributed services configuration can scale to support 100s 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.

Avoiding Vendor Lock-in

As organizations continue to go all-in on cloud-first strategies, optimizing their cloud architectures is becoming a top priority. It's estimated that investments in infrastructure to support cloud computing

account for more than a third of all IT spending. Using multiple cloud providers with multiple cloud services requires an architecture that enables workflow portability, and enterprises need an unbiased, multi-cloud strategy to make that a reality.

Workflow portability enables a seamless workflow, regardless of where you deploy. Benefits include greater flexibility to use the right tools and cloud for your specific needs, ability for existing systems to work within another agency organization's infrastructure, even if both are using separate cloud providers, increased resilience and continuity of operations, and improved cloud negotiations.

The GitLab open core model allows our users to adopt better tools faster through open source innovation, attract and retain better talent, and reduce costs because multiple customers can contribute code to solve similar problems. The Government can also lower risk because agencies are never locked into feature sets we choose to build; they have the flexibility to add to the product as needed, whenever they need it. And because our global community is constantly contributing and improving our widely used product, agencies don't need to worry about technology obsolescence. Access for product and technical support is also available.

The Choice of Modern Software Companies

GitLab makes it easier for *private-sector companies* to achieve software excellence so that they can unlock great software for their customers by reducing the cycle time between having an idea and seeing it in production. GitLab's *software framework* helps teams accelerate software delivery from weeks to minutes, reduce development costs, and reduce the risk of application vulnerabilities while increasing developer productivity. More than 100,000 organizations of the *world's most successful modern consumer and enterprise software companies*, from startups to global enterprise organizations, including NASA, the U.S. Air Force, the National Archives, the European Space Agency, CERN, the Government of Canada, Interpol, the University of Washington, The University of Pennsylvania Medical Center, Lockheed Martin, CapGemini, RedHat, ServiceNow, and the Cloud Native Computing Foundation *choose* GitLab to deliver great software at new speeds.

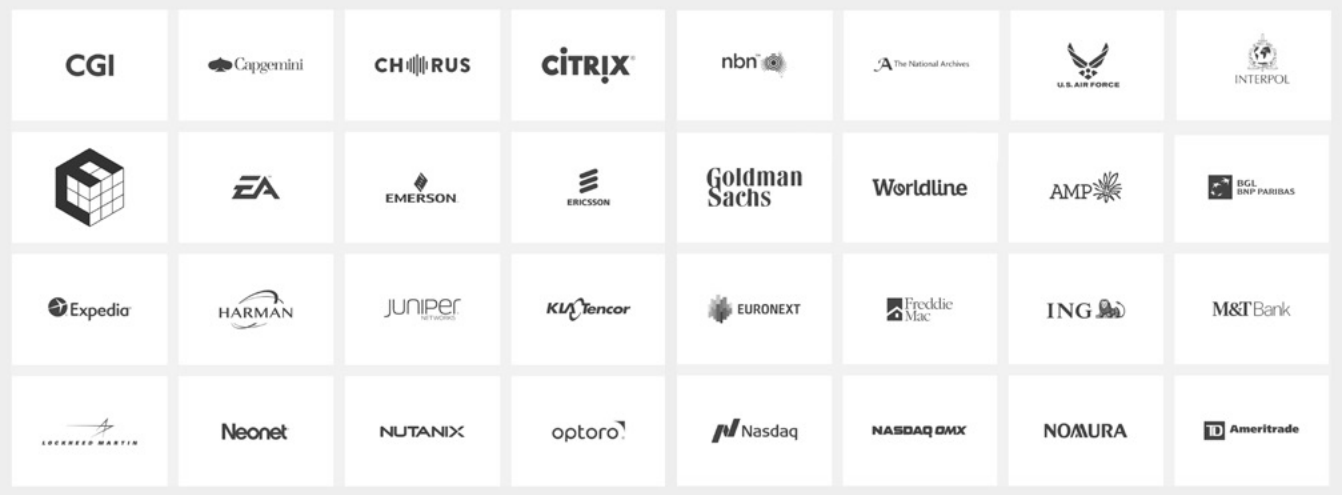


FIGURE 8.1: SELECTION OF GITLAB'S COMMERCIAL AND PUBLIC SECTOR ENTERPRISE CUSTOMERS

What Our Private-Sector Customers Are Saying

"One tool for SCM+CI/CD was a big initial win. Now wrapping security scans into that tool as well has already increased our visibility into security vulnerabilities. The integrated Docker registry has also been very helpful for us. Issue/Product management features let everyone operate in the same space regardless of role."

— ADAM DEHNEL, PRODUCT ARCHITECT AT BI WORLDWIDE

"GitLab has allowed us to dramatically increase the velocity of development in our Engineering Division. We believe GitLab's dedication to helping enterprises rapidly and effectively bring software to market will help other companies achieve the same sort of efficiencies we have seen inside Goldman Sachs. We now see some teams running and merging 1000+ CI feature branch builds a day!"

— ANDREW KNIGHT, MANAGING DIRECTOR AT GOLDMAN SACHS

"It's clearly a powerful tool to do our operations, code collaboration and record discussions on our development and deployment process. We can do more because we can handle more complex projects. As an individual, I'm able to be involved with several large projects because I can rely on GitLab, and the other development tools that we have deployed around GitLab, to keep track of things."

— ALEX LOSSENT, VERSION CONTROL SYSTEMS SERVICE MANAGER AT CERN IT DEPARTMENT

Setting Up a Local Environment

At GitLab, we love to document everything, so if you need to know how to do anything, chances are pretty high that we have you covered. For *understandable instructions for setting up a local development environment,* developers can check out our "Setting up your development environment" page and our "Environments and deployments" page. We also have extensive developer documentation on "Setting up a build environment" and a zillion other things on docs.gitlab.com.

Adding & Deleting Team Members

Team members can be quickly added or removed from projects on GitLab. Users with Maintainer or Owner permissions can manage groups and users and their access levels across all of your projects and personalize the access level for each user, per-project. New users can be immediately added to your project, even if the user you want to give access to doesn't have an account on your GitLab instance, just by typing their email address in the user search field. Access and permission levels can be revised or revoked easily at any time by the project Maintainer or Owner.

See: GitLab customer use stories: about.gitlab.com/customers
Peer Reviews of GitLab: about.gitlab.com/handbook/marketing/product-marketing/proof-points

Play #9: Deploy in a flexible hosting environment

"Our services should be deployed on flexible infrastructure, where resources can be provisioned in real-time to meet spikes in traffic and user demand. Our digital services are crippled when we host them in data centers that market themselves as "cloud hosting" but require us to manage and maintain hardware directly. This outdated practice wastes time, weakens our disaster recovery plans, and results in significantly higher costs."
— playbook.cio.gov/#play9

Flexible Infrastructure

Applications developed using GitLab can be deployed to all major cloud computing providers, government hosted clouds, or on-premises data centers. GitLab itself can be *deployed on flexible infrastructure* such as a SaaS solution that requires no management (gitLab.com) or it can be hosted in cloud environments that do *not require you to manage and maintain hardware directly* or on-premises. GitLab also supports High Availability (HA) configurations and Geographic Distributed 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 Hawaii attempting to clone or partially clone a repository from a Git server located in Virginia could experience performance Issues. Customers who encounter this kind of latency often request what's known as GitLab Geo-replication (GitLab Geo).

GitLab Geo replicates Git repository data and large binary files (Git LFS), *enabling resources to be available in multiple regions.* 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.

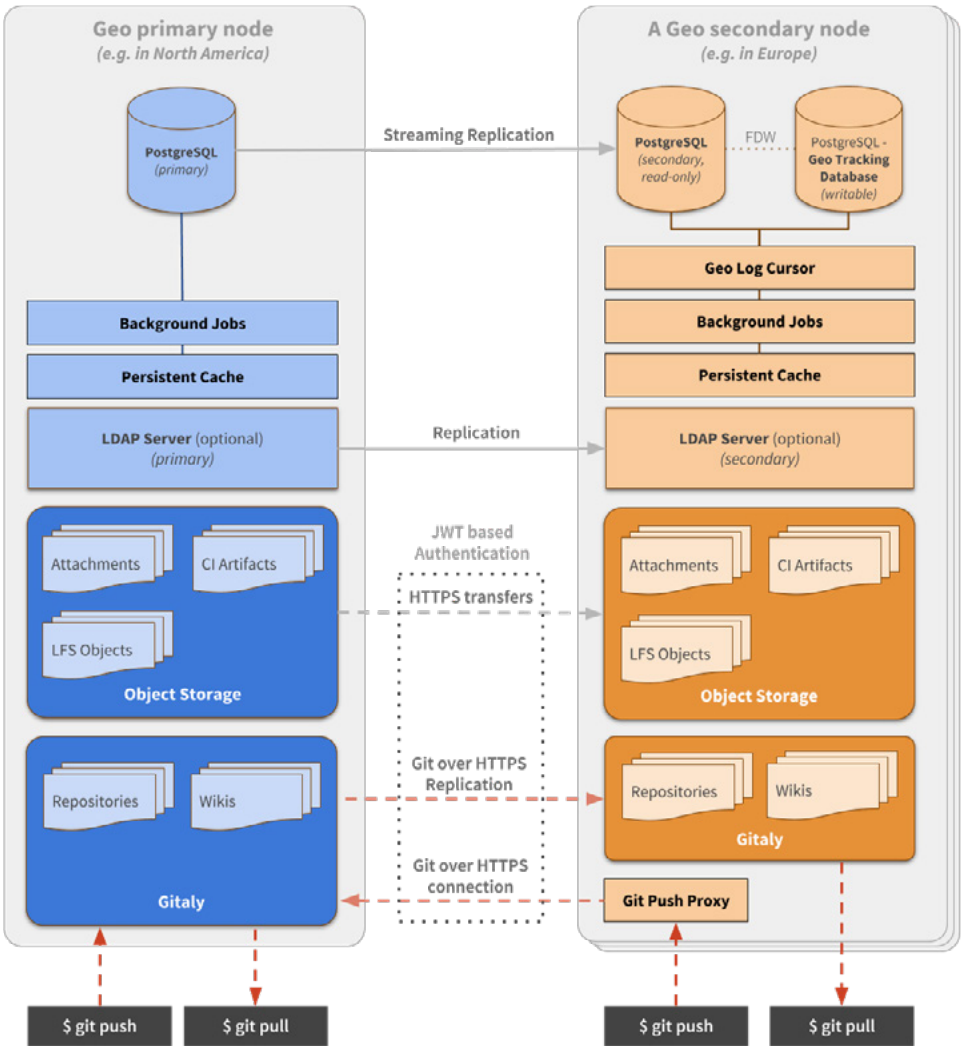


FIGURE 9.1: ILLUSTRATION OF HOW GITLAB GEO-REPLICATION (GITLAB GEO) WORKS

GitLab also has the capability to scale its Runners *on demand* for services spikes when development teams may be heavy in development and then automatically return to pre-spike levels during light development periods. GitLab architecture provides very flexible development architecture to enable the deployment of your applications developed through GitLab.

Autoscaling is one way that teams can reduce the costs associated with running concurrent jobs. Autoscaling Runners split this work across multiple servers and spin up or down automatically to process queues—so developers don't have to wait on builds and teams use only as much capacity as needed.

Play #10: Automate testing and deployments

"Today, developers write automated scripts that can verify thousands of scenarios in minutes and then deploy updated code into production environments multiple times a day. They use automated performance tests which simulate surges in traffic to identify performance bottlenecks. While manual tests and quality assurance are still necessary, automated tests provide consistent and reliable protection against unintentional regressions, and make it possible for developers to confidently release frequent updates to the service."

— playbook.cio.gov/#play10

Automated Deployments through Continuous Integration & Continuous Delivery (CI/CD)

The continuous methodologies of software development are based on automating the execution of scripts to minimize the chance of introducing errors while developing applications. They require less human intervention, or even no intervention at all, from the development of new code until its deployment. It involves continuously building, testing, and deploying code changes at every small iteration, reducing the chance of developing new code based on buggy or failed previous versions.

GitLab CI/CD—created to enable you to *perform deployments automatically*—is a powerful tool built into the platform that allows you to apply all the continuous methods (Continuous Integration, Delivery, and Deployment) to your software with no third-party application or integration needed.

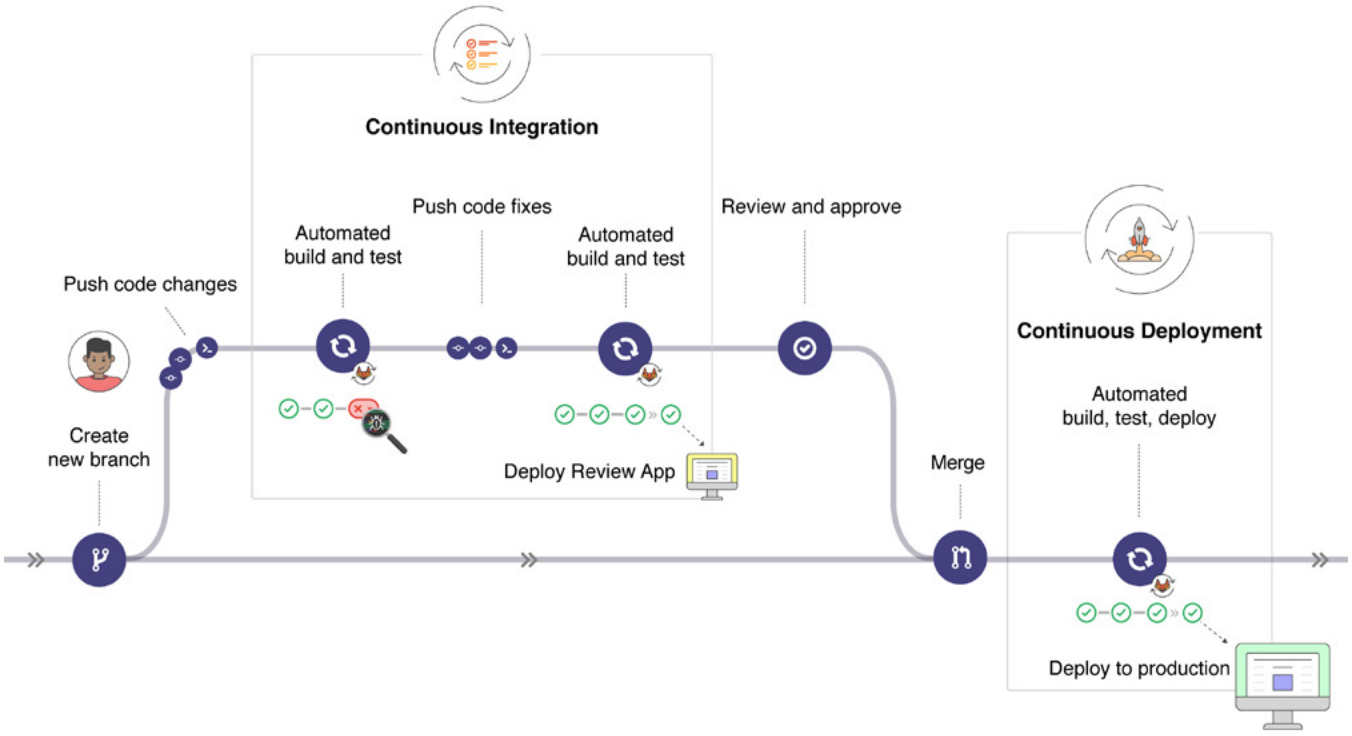


FIGURE 10.1: BASIC GITLAB CI/CD & DEPLOYMENT WORKFLOW

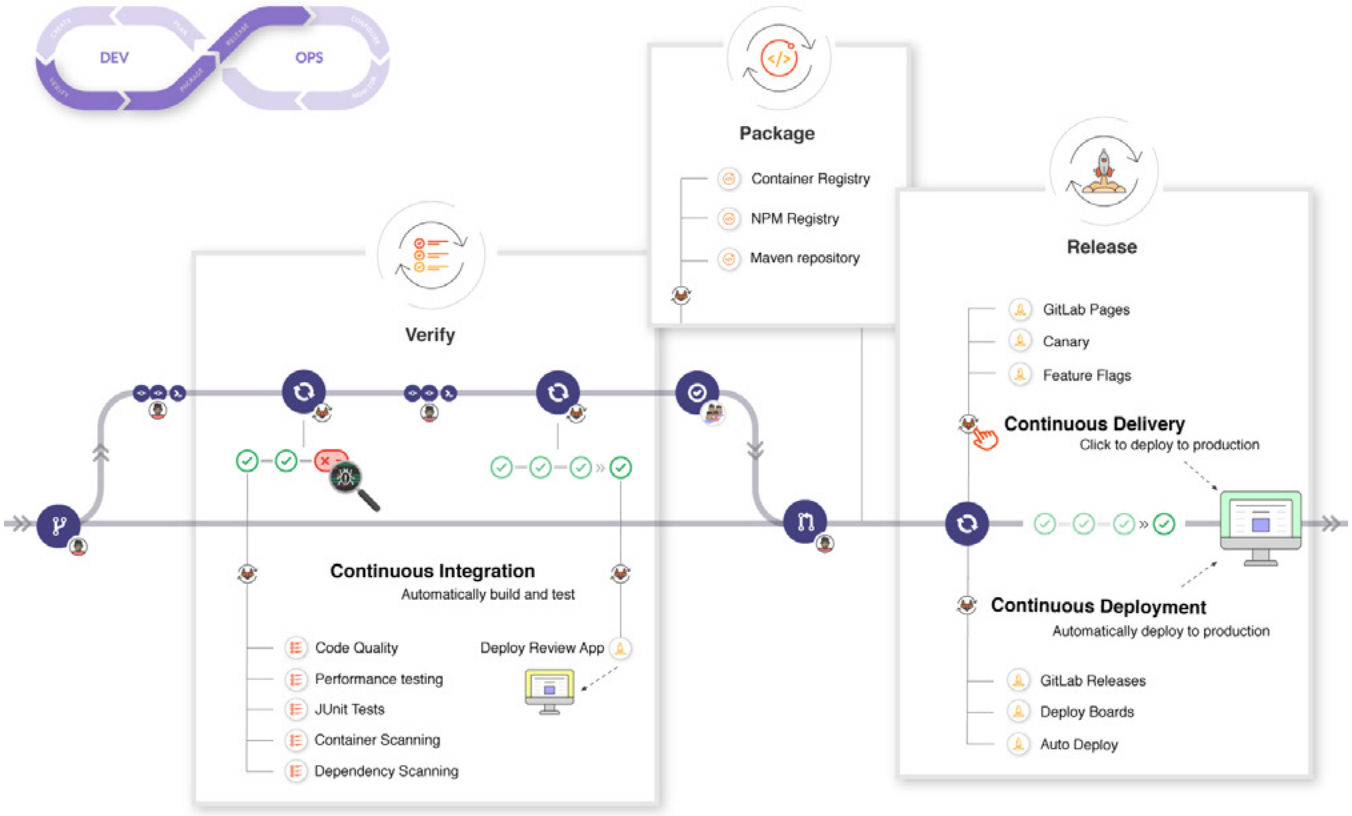


FIGURE 10.2: GITLAB CI/CD & DEPLOYMENT DEEPER DIVE (VERIFY, PACKAGE, AND RELEASE)

With GitLab CI/CD you can also:

» Easily set up your app's entire lifecycle with Auto DevOps
» Deploy your app to different environments
» Install your own GitLab Runner
» Schedule pipelines
» Check for app vulnerabilities with Security Test Reports

Automated Testing

Any testing to be performed on an application built using GitLab is *automatically run as part of the build process* every time the CI pipeline executes. This automated testing enables the development team to see results of functional, security, performance, and regression testing and make changes appropriately prior to deploying it. *In terms of creating automated tests that verify all user-facing functionality,* we have made progress on building a full suite of end-to-end automated tests for our enterprise features. We now have 183 automated tests for enterprise features and are now tracking existing and planned tests via our internal test case management system. In addition, to ensure we are guarding every release, we have rolled out a process for reliable tests to promote high value and highly stable tests that must pass with every deployment.

Watch: GitLab CI Live Demo video for a deeper overview of GitLab CI/CD

Play #11: Manage security and privacy through reusable processes

"Our digital services have to protect sensitive information and keep systems secure. This is typically a process of continuous review and improvement which should be built into the development and maintenance of the service. At the start of designing a new service or feature, the team lead should engage the appropriate privacy, security, and legal officer(s) to discuss the type of information collected, how it should be secured, how long it is kept, and how it may be used and shared. The sustained engagement of a privacy specialist helps ensure that personal data is properly managed. In addition, a key process to building a secure service is comprehensively testing and certifying the components in each layer of the technology stack for security vulnerabilities, and then to re-use these same pre-certified components for multiple services."

— playbook.cio.gov/#play11

GitLab strongly believes that application security must be part of the development workflow and should be integrated into the entire software development lifecycle (SDLC), from source control to CI/CD, on through to application monitoring. By integrating tightly into the SDLC, each code change is automatically tested for every code commit and results are presented to the developer for instant remediation. A single workflow without separate silos prevents teams from wasting extensive application security time and budget on tracking vulnerabilities for severity, potential risk, and time-to-remediation. By integrating with the CI/CD process, the tracking becomes a by-product of development steps already being taken, and attention can then be focused on the remediation itself. Crucially, when the application security tool and the source control tool are the same, there is no need to identify incremental new code by looking from the outside; the code repo definitively reveals what has changed. The developer's action of committing code to the repository kicks off security scans, then the CI/CD component spins up a review app for the developer to test their code interactively before it is merged. Dynamic scans are performed by the developer on the piece of code they are changing, before it is merged with other code. Security is automated and embedded into a development tool teams already know that enables and empowers them to quickly resolve potential Issues where *personal data* could be compromised.

Engagement and Collaboration

Highly functional teams are the ones that communicate well, early, and often. A set of tools that enables that collaboration will provide the basis for the team to be more productive. Security and privacy are exceptionally susceptible to communication degradation. Knowing that a vulnerability

exists in a delivered system months after the security team gets a chance to review it can have far reaching consequences to a project. Therefore, GitLab's view is to *engage the appropriate privacy, security, and legal* teams early into the process by allowing them to openly participate and contribute to the work being performed.

For example, within the context of a Merge Request, GitLab has the ability to display all security findings that are discovered by performing Static Analysis (SAST), Dynamic Analysis (DAST), License Compliance, Dependency Scanning (Software Composition Analysis), Container security scans, and secret detection all in one place, on a single Security Dashboard. This single dashboard gives security, privacy, and legal teams an early view into potential vulnerabilities, while the product is still in development. Additionally, these teams can be made responsible to approve the merge of the code, thus making it easier for them to immediately take necessary action to fix the problem, before it gets inserted into the overall code base.

Developing with Building Blocks for Reuse

Small primitives are building blocks in GitLab. They are an abstraction, not at the technical level, but truly at the product level. Small primitives can be combined, built-upon further, and otherwise leveraged to create new functionality in GitLab.

For example, the label lists in Issue Boards use the smaller primitive of Labels. Some small primitives are intended to be mutable and iterated upon. For example, when a feature is new in a product, it may need to be iterated upon until it meets the original design specifications and may need further iteration before it can serve appropriately as a building block for other functionality. Other small primitives can be intended to be immutable, secured components that are intended to be static long term. These set of components have value in that they are unchanging so they can be relied upon as building blocks without fear of deprecation. APIs are a good example of building block that work best when they are immutable. Having a policy where new APIs can be created, but you continue to support the old ones without deprecating them, gives integration developers confidence that the software they build on your APIs won't mysteriously stop working in the future.

Small primitives are especially powerful because they usually take less effort and provide higher leverage than you would get from a more "complete" but standalone feature. Think of how simple Unix command line utilities can be chained together to do really complicated things, much easier (and certainly more flexibly) than you could have done with a dedicated tool.

When iterating on GitLab, our product team strongly contemplates using small primitives instead of creating new abstractions, especially when considering minimal viable change (MVC) features

that will provide the foundations for further improvements. To do this, we start with easy to apply concepts that meet the needs of intermediate to advanced users; from there we document the usage clearly and ensure we think about discoverability. The UX can very often be refactored or enhanced later when there's a demonstrated need for refinement, onboarding for less sophisticated users, or other new abstractions needed that were identified through real-world usage.

Continuous Review and Improvement

Iteration is one of GitLab's core values. We do the smallest thing possible and get it out as quickly as possible. In fact, we have a saying that nothing is ever finished because we're always looking for MVC, and then iterating. By shipping smaller pieces, we not only deliver faster, but we learn from what's been shipped, and then iterate smarter. *Continuous review and improvement are built into the development and maintenance of* the way the GitLab product evolves every single day and it is built into the platform so that customers can also make continual strides toward better services for users and citizens.

Pre-Certify Infrastructure for FedRAMP

It is recommended that Government customers *"pre-certify" the hosting infrastructure used for the project using FedRAMP* to build a secure foundation for their service. The majority of GitLab's government customers use an on-premises instance of our software product in their FedRAMP authorized cloud environment, such as AWS, Google Cloud, Azure, or a hybrid or multi-cloud environment. Other customers find that they can install and run a self-managed instance of the GitLab product by placing it in a FedRAMP-authorized data center and using a FedRAMP-authorized Cloud Service Provider.

See: Behind the scenes: How we built Review Apps for an example of GitLab iteration in action

Play #12: Use data to drive decisions

"At every stage of a project, we should measure how well our service is working for our users. This includes measuring how well a system performs and how people are interacting with it in real-time. Our teams and agency leadership should carefully watch these metrics to find Issues and identify which bug fixes and improvements should be prioritized. Along with monitoring tools, a feedback mechanism should be in place for people to report Issues directly."
— playbook.cio.gov/#play12

Data-Driven Decisioning

At GitLab, we love data. We use lots of *data to drive decisions* and we provide application data—and monitoring tool integrations (e.g., Prometheus and Splunk)—to our customers to equip them with clear monitoring of running applications deployed through the CI/CD pipeline. Monitoring metrics such as mean-time-to-acknowledge (MTTA), mean-time-to-resolution (MTTR), mean-time-between-failures (MTBF), etc.—and combining them with contribution graphs and lifecycle process of the software being developed—provides a clear picture of where the process needs further improvements. GitLab visualizes monitoring data in graphs and dashboards that are built using Grafana.

GitLab offers powerful integration with Prometheus for monitoring key metrics of your apps, directly within GitLab. Metrics for each environment are retrieved from Prometheus, and then displayed within the GitLab interface. GitLab can also seamlessly deploy and manage Prometheus on a connected Kubernetes cluster, making monitoring of your apps easy.

Once configured, GitLab will attempt to retrieve performance metrics for any environment which has had a successful deployment. Custom metrics can be monitored by adding them on the monitoring dashboard page. By default, all projects include a GitLab-defined Prometheus dashboard, which includes a few key metrics, but you can also define your own custom dashboards.

Actionable Feedback

Deployments should never be fire and forget. GitLab will give you immediate feedback on every deployment on any scale. This means that GitLab can tell you whether performance has improved on the application level, but also whether business metrics have changed.

Concretely, we can split up monitoring and feedback efforts within GitLab in three distinct areas: execution (cycle analytics), business, and system feedback. Monitoring and feedback activities

require either Kubernetes or manual environment specification and monitoring mechanisms be built into the software.

Business feedback

With the power of monitoring and an integrated approach, we have the ability to do amazing things within GitLab. GitLab will be able to automatically test commits and versions through feature flags and A/B testing.

Business feedback exists on different levels:

» Short term: how does a certain change perform? Choose A/B based on data
» Medium term: did a particular new feature change conversions or engagement?
» Long term: how do larger efforts relate to changes in conversations, engagement, revenue?
» A/B Testing of branches

Application feedback

Your application should perform well after changes are made. GitLab will be able to see whether a change is causing errors or performance issues on application level. Think about:

» Response times (e.g. a backend API)
» Error rates and occurrences of new bugs
» Changes in API calls

System feedback

We can now go beyond CI and CD. GitLab will able to tell you whether a change improved performance or stability. Because it will have access to both historical data on performance and code, it can show you the impact of any particular change at any time and can also *monitor system performance in real-time (e.g. response time, latency, throughput, and error rates).*

System feedback happens over different time windows:

» Immediate: see whether changes influence availability and alert if they do
» Short-medium term: see whether changes influence system metrics and performance
» Medium-long term: did a particular effort influence system status
» Implemented: Performance Monitoring
» Status monitoring and feedback
» Feature monitoring

Execution Feedback & Cycle Analytics

GitLab is able to speed up cycle time for any project. To provide feedback on cycle time, GitLab will continue to expand cycle analytics so that it not only shows you what is slow, it'll help you speed up with concrete, clickable suggestions.

» Cycle Speed Suggestions

User Feedback

GitLab's Service Desk module provides a *feedback mechanism for people to report Issues directly.* It allows teams to connect with any external party through email inside GitLab; no external tools are required. An ongoing conversation right where your software is built ensures that user feedback ends up exactly where needed, helping you build the right features to solve your user's real problems and eliminating the complexity and inefficiencies of multiple tools and external integrations while significantly shortening the cycle time from feedback to software update. Users can email bug reports, feature requests, or any other general feedback directly into your GitLab project as a new Issue and your team can then send a response from the project.

See: Cycle Analytics: about.gitlab.com/product/cycle-analytics
Monitoring: about.gitlab.com/features/#monitor
"Everyone can contribute!"

Play #13: Default to open

"When we collaborate in the open and publish our data publicly, we can improve Government together. By building services more openly and publishing open data, we simplify the public's access to government services and information, allow the public to contribute easily, and enable reuse by entrepreneurs, nonprofits, other agencies, and the public."
— playbook.cio.gov/#play13

Default to Open

GitLab is an open core company that develops software for the entire software development lifecycle (SDLC) and we help teams around the world create and share open core products and services. From our founding, we've embodied openness and transparency and have elevated these principles to being core values of the company.

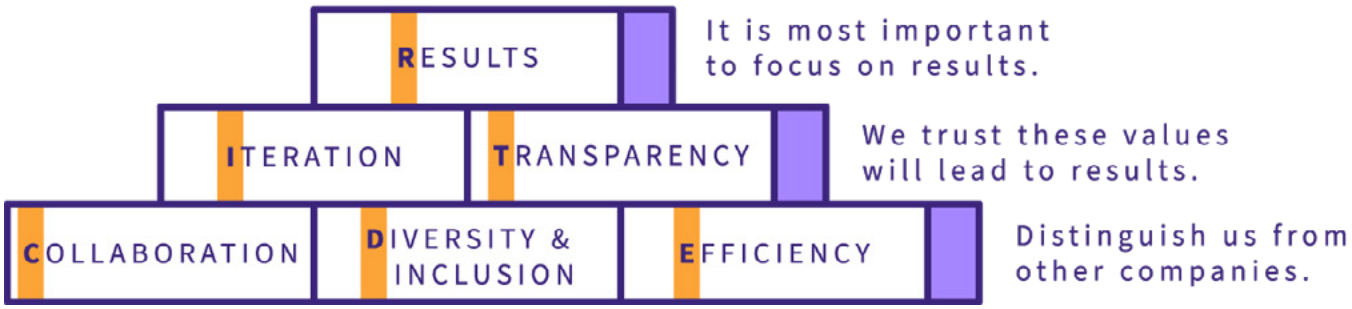


FIGURE 13.1: GITLAB'S VALUES: COLLABORATION, RESULTS, EFFICIENCY, DIVERSITY & INCLUSION, ITERATION AND TRANSPARENCY (CREDIT)

We *default to open* as a company and understand the benefits of this approach. Everything at GitLab is public by default. If something is not public, there is a reference in our public-facing handbook (our standard operating procedures) that states a confidential decision was taken and with a link to our 'Not Public' guidelines, unless our legal team feels it carries undue risk. The public process does two things: it allows others to benefit from the conversation and acts as a filter. Since there is only a limited amount of time, we prioritize conversations from which a wider audience can benefit. We *collaborate in the open and publish our data publicly* so that our more than 100,000 customer organizations and our active community of 4800+ can participate and *contribute easily* in our open core development process.

Reporting Mechanism and Responsiveness

We encourage anyone who wants to make our product or company better to propose features, report bugs, or share an idea about the company. They can submit an Issue to the relevant product feature

tracker, post to a Slack channel, or create a Merge Request (our *feedback mechanisms for people to report bugs, Issues,* and ideas). Our customers use some of the same feedback tools within GitLab to capture and action upon user feedback, Issues, and software bugs. (*Play #12*) GitLab is *responsive to these reports* and many of these features, fixes, and ideas are incorporated into the product by our community or by our internal team.

Sharing the Development Process and Progress

We publish our roadmap so that everyone knows what we are doing and what we plan to do in our upcoming monthly releases. We share *our development process* through Issues that are open to the public and we share our development *progress publicly* via our release blog, social media, developer media articles, and other channels.

Open AND Protected

While the goal is to be open and transparent, defaulting to open, GitLab also understands the need for our customers to make the decision to share information and assets for themselves. We provide flexibility for the Government to share or not share repositories as appropriate and enable you to host on GitLab.com, in a secure cloud, or internally on-premises. Projects can also be set to public, internal, or private status as appropriate.

Conclusion

As U.S. Federal Government agencies continue to transform by providing more valuable digital services, the practice of using one tool will emerge as the key to unlocking the potential of implementing the 13 plays in the USDS Playbook. The alternative is a lengthy process of acquiring various tools, investing in integrating and learning the different tools to assist with each of the plays, and managing and maintaining them. The net result is a lack of visibility, a lack of consistency, insufficient skills/expertise/resources to manage toolchains, a higher cost of ownership, and difficulty in ensuring security across the software development lifecycle (SDLC).

Recommendations

Based on and in support of the measures outlined in the USDS Digital Services Playbook, GitLab recommends DevSecOps teams implement the additional following actions:

1. **Implement a single application for the entire SDLC lifecycle** to reduce toolchain complexity and create a software 'factory' that supports centralized collaboration and efficiency.
2. **Combine source code management and continuous integration/continuous delivery** to deliver better applications faster.
3. **Dogfood the applications your team develops and delivers** to discover ways to improve the user experience and reduce friction.
4. **Use a single source of truth** to collect, manage, and deliver on user stories from start to finish for end-to-end insight and visibility.
5. **Create a collaborative and transparent user experience**, enabling internal stakeholders, customers, warfighters, contractors, and citizens to influence features they want and need.
6. **Use open source so that everyone can contribute** to product functionality and drive rapid innovation.
7. **Leverage hybrid environments to take advantage of flexible hosting options** to support scaling for your evolving mission requirements.
8. **Implement built-in security and compliance** to move security testing earlier in the development lifecycle, reduce risk, and protect your code from cradle to grave.
9. Avoid cloud service provider lock-in and work the same way across clouds by **deploying your applications anywhere using workflow portability.**
10. Attract and retain talent and build high-performing teams by **implementing the latest development software** developers want to use to collaborate and do their best work.

About GitLab

GitLab is a complete DevOps platform, delivered as a single application, fundamentally changing the way Development, Security, and Ops teams collaborate. GitLab helps teams accelerate software delivery from weeks to minutes, reduce development costs, and reduce the risk of application vulnerabilities while increasing developer productivity. GitLab provides unmatched visibility, radical new levels of efficiency and comprehensive governance to significantly compress the time between planning a change and monitoring its effect. Now, fast paced teams no longer have to integrate or synchronize multiple DevOps tools and are able to go faster by working seamlessly across the complete lifecycle.

GitLab delivers complete real-time visibility of all projects and relevant activities across the entire DevOps lifecycle. For the first time, teams can see everything that matters. Changes, status, cycle times, security and operational health are instantly available from a trusted single source of data. Information is shown where it matters most, e.g. production impact is shown together with the code changes that caused it. And developers see all relevant security and ops information for any change. With GitLab, there is never any need to wait on synchronizing your monitoring app to version control or copying information from tool to tool. GitLab frees teams to manage projects, not tools. These powerful capabilities eliminate guesswork, help teams drive accountability and gives everyone the data-driven confidence to act with new certainty. With GitLab, DevOps teams get better every day by having the visibility to see progress and operate with a deeper understanding of cycle times across projects and activities.

GitLab drives radically faster cycle times by helping DevOps teams achieve higher levels of efficiency across all stages of the lifecycle making it possible for Product, Development, QA, Security, and Operations teams to work at the same time, instead of waiting for handoffs. Teams can collaborate and review changes together before pushing to production. GitLab eliminates the need to manually configure and integrate multiple tools for each project. GitLab makes it easy for teams to get started, they can start with GitLab using one or two use cases they need to improve, and then begin their evolution to a single end-to-end experience for all of DevSecOps.

Only GitLab delivers DevOps teams powerful new governance capabilities embedded across the expanded lifecycle to automate security, code quality and vulnerability management. With GitLab, tighter governance and control never slow down DevOps speed.

Faster releases. Better code. Less pain. Mission Accomplished!

About the U.S. Digital Service (USDS)

USDS is a group of technologists from diverse backgrounds working across the federal government to transform critical services for the people. These specialists join for tours of civic service to create a steady influx of fresh perspectives. Their mission is to do the greatest good for the greatest number of people in the greatest need.

USDS deploys small, responsive groups of technology experts to work with and empower civil servants. These multi-disciplinary teams bring best practices and new approaches to untangle some of our nation's most important problems.

USDS Objectives

1. Transform critical, public-facing services
2. Expand the use of common platforms, services, and tools
3. Rethink how the Government buys digital services
4. Bring top technical talent into civic service

www.usds.gov/mission

Acknowledgments

Thank you to all of the GitLabbers—past and present—who contributed to the content of this guide. In true GitLab spirit, we collaborate zealously, passionately, constantly, and without reserve to make our product, the company, and the open source community better every day. Special thanks to the army of prolific GitLab bloggers, engineers and developers, solution architects, technical account managers and sales teams, product managers, product and technical marketing managers, documentation and content specialists, technical writers, community contributors, and a host of others for the wealth of content material and support they contributed to this guide. Truly, this is a team effort and each of you is greatly appreciated.

APPENDIX: References, Resources & Links

United States Digital Service (2014). The Digital Services Playbook — from the U.S. Digital Service. [online] Playbook.cio.gov. Available at: playbook.cio.gov [Accessed 26 May 2020].

Play #1:

- » GitLab's 'dogfooding' culture: https://about.gitlab.com/handbook/values/#dogfooding
- » GitLab product documentation: https://docs.gitlab.com/
- » How to use GitLab for Agile software development: https://about.gitlab.com/2018/03/05/gitlab-for-agile-software-development/

Play #2:

- » How the GitLab UX team uses Epics: https://about.gitlab.com/2018/03/19/use-cases-for-epics/

Play #3:

- » GitLab Workflow: https://about.gitlab.com/solutions/gitlab-flow/
- » Chrome Accessibility Developer Tools: https://github.com/GoogleChrome/accessibility-developer-tools
- » Axe browser extension: https://www.deque.com/axe/
- » GitLab audit rules page: https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-Rules
- » Awesome-a11y resource list: https://github.com/brunopulis/awesome-a11y
- » GitLab's VPAT Conformance Report (Revised Section 508 Edition): https://design.gitlab.com/accessibility/vpat/

Play #4:

- » GitLab Release and Maintenance Policy: https://docs.gitlab.com/ee/policy/maintenance.html
- » Upcoming GitLab releases: https://about.gitlab.com/upcoming-releases/
- » Velocity over predictability: https://about.gitlab.com/handbook/engineering/#velocity-over-predictability
- » GitLab Issue Tracker: https://docs.gitlab.com/ee/user/project/Issues/index.html
- » External Issue Tracker integrations:
  - » Jira: https://docs.gitlab.com/ee/user/project/integrations/jira.html
  - » Redmine: https://docs.gitlab.com/ee/user/project/integrations/redmine.html
  - » YouTrack: https://docs.gitlab.com/ee/user/project/integrations/youtrack.html
  - » Bugzilla: https://docs.gitlab.com/ee/user/project/integrations/bugzilla.html
  - » Marker.io: https://about.gitlab.com/2019/01/09/marker-io-gitlab-integration/

- Customer Tracker: https://docs.gitlab.com/ee/user/project/integrations/custom_Issue_tracker.html
- More about GitLab CI/CD: https://about.gitlab.com/product/continuous-integration/
- GitLab for Agile Development: https://about.gitlab.com/2018/03/05/gitlab-for-agile-software-development/
- Code Review Guidelines: Best practices & advice: https://docs.gitlab.com/ee/development/code_review.html

Play #5:

- GitLab annual per user pricing: https://about.gitlab.com/pricing/#self-managed

Play #6:

- Permissions: https://docs.gitlab.com/ee/user/permissions.html

Play #7:

- GitLab 2018 Global Developer Report: https://about.gitlab.com/developer-survey/previous/2018/
- Omnibus GitLab: https://gitlab.com/gitlab-org/omnibus-gitlab
- GitLab QA Orchestrator: https://gitlab.com/gitlab-org/gitlab-qa
- Nightly Build Pipelines: https://gitlab.com/gitlab-org/quality/nightly/pipelines
- Nightly Staging Pipelines: https://gitlab.com/gitlab-org/quality/staging/pipelines
- GitLab Development Kit (GDK): https://gitlab.com/gitlab-org/gitlab-development-kit/
- QA README File: https://gitLab.com/gitlab-org/gitLab/tree/master/qa/README.md#how-can-i-use-it
- GitLab QA Architecture: https://gitlab.com/gitlab-org/gitlab-qa/blob/master/docs/architecture.md
- GitLab 2019 Global Developer Report: https://about.gitlab.com/developer-survey/previous/
- Auto Remediation: https://gitlab.com/groups/gitlab-org/-/epics/759
- Topics:
  - Vulnerabilities: https://docs.gitlab.com/ee/user/application_security/sast/
  - Preventing exposure: https://docs.gitlab.com/ee/user/application_security/dast/
  - Dependencies: https://docs.gitlab.com/ee/user/application_security/dependency_scanning/
  - Container images: https://docs.gitlab.com/ee/user/application_security/license_compliance/
  - Auto remediation: https://about.gitlab.com/direction/secure/#security-is-a-team-effort
  - IAST: https://gitLab.com/groups/gitlab-org/-/epics/344
  - Fuzzing: https://gitLab.com/groups/gitlab-org/-/epics/818

Play #8:

- Setting up your development environment: https://docs.gitlab.com/omnibus/development/setup.html
- Environments and deployments: https://docs.gitlab.com/ee/ci/environments.html
- Setting up a build environment: https://docs.gitlab.com/omnibus/build/prepare-build-environment.html
- Developer documentation site: https://docs.gitlab.com/
- GitLab customer user stories: https://about.gitlab.com/customers/
- Peer reviews of GitLab: https://about.gitlab.com/handbook/marketing/product-marketing/peer-reviews/

Play #9:

- GitLab Geo: https://about.gitlab.com/2017/11/22/gitlab-10-2-released/#gitlab-geo-is-now-generally-available
- Autoscaling Runners: https://about.gitlab.com/2016/03/29/gitlab-runner-1-1-released/

Play #10:

- GitLab CI/CD Topics:
  - Auto DevOps: https://docs.gitlab.com/ee/topics/autodevops/index.html
  - Environments: https://docs.gitlab.com/ee/ci/environments.html
  - GitLab Runner: https://docs.gitlab.com/runner/
  - Schedule Pipelines: https://docs.gitlab.com/ee/user/project/pipelines/schedules.html
  - Security Test Reports: https://docs.gitlab.com/ee/user/project/merge_requests/index.html#security-reports-ultimate
- GitLab live CI demo: https://www.youtube.com/watch?v=pBe4t1CD8Fc

Play #11:

- Behind the scenes: How we built Review Apps: https://about.gitlab.com/2017/01/04/behind-the-scenes-how-we-built-review-apps/

Play #12:

- Cycle Analytics: https://about.gitlab.com/product/cycle-analytics/
- Monitoring: https://about.gitlab.com/features/#monitor
- Everyone can contribute: https://about.gitlab.com/submit-feedback/

Play #13:

- Open Core: https://about.gitlab.com/2016/07/20/gitlab-is-open-core-github-is-closed-source/

Other Nifty Stuff You Can Use:

» GitLab University: https://docs.gitlab.com/ee/university/

» The GitLab Cookbook: https://www.packtpub.com/application-development/gitlab-cookbook

» GitLab YouTube Channel: https://www.youtube.com/channel/UCnMGQ8QHMAnVIsI3xJrihhg

about.gitlab.com/solutions/public-sector


GitLab