

## laC + PR = GitOps

BY THOMAS A. LIMONCELLI

# GitOps: A Path to More Self-Service IT

You have written a new Web application and would like it to be added to your organization's Web load balancer. The load balancer is complex; highly trained experts on the network operations team maintain its configuration. You file a ticket with the team, wait, wait, wait, have a discussion with the experts, wait, wait, wait, and finally your request is completed. Your application is now available, assuming they got it right on the first try.

At other companies the process looks more like this:

1. Find the Git repo that stores a logical description of the plumbing that connects the load balancer to various Web application servers.
2. Edit that file to add your new application.
3. The proposed revision is submitted to the Web team as a pull request (PR) the same way developers submit PRs for software projects.
4. At the same time that humans are reviewing

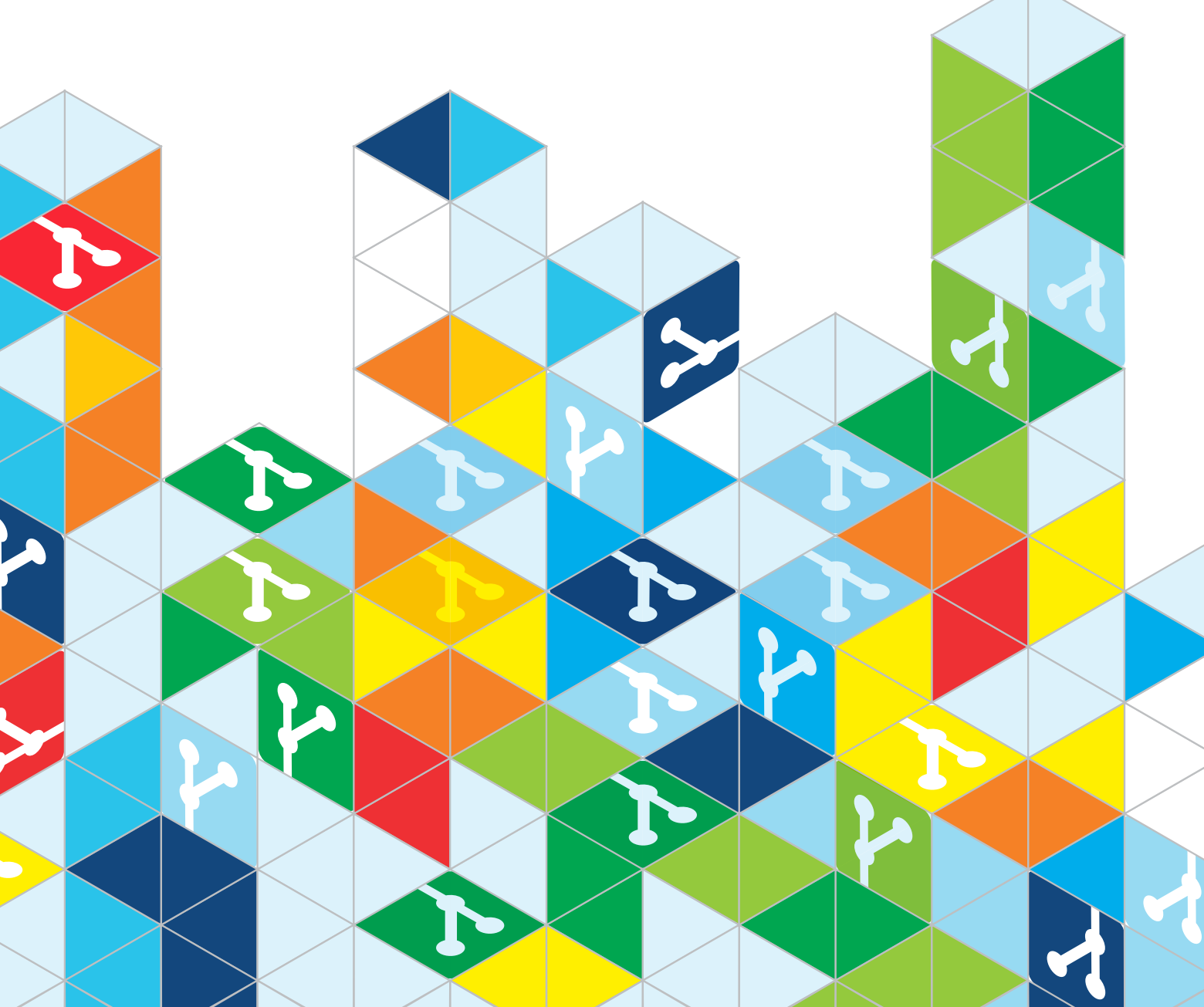
the PR, your continuous integration (CI) system (such as Jenkins or similar) is linting and unit testing your changes to the load balancer config (possibly in a container or VM).

5. Once the PR is approved and "the builds are green," the continuous deployment (CD) pipeline (often another Jenkins job or similar) will take care of generating the new config file for the production load balancer and deploying it, usually with the help of a config management system such as Puppet or Chef.

This kind of workflow is known as GitOps: Empowering users to do their own IT operations via PRs.

GitOps lowers the bar for creating self-service versions of common IT (or site reliability engineering, or DevOps) processes. As the bar is lowered, it be-





comes easier to meet the return in the ROI (return on investment) calculation. GitOps not only achieves this, but also encourages desired behaviors in IT systems: better testing, reduction of “bus factor,”<sup>a</sup> reduced wait time, more infrastructure logic being handled programmatically with infrastructure as code (IaC), and directing time away from manual toil toward creating and maintaining automation.

GitOps makes it easy for IT organizations to provide self-service IT operations. It does all that while reducing the amount of code that must be written and maintained. It encourages an incremental approach to automation: approvals are mostly manual at the beginning but become more au-

tomated over time. The only code that gets written is code that has proven to be needed.

Most importantly, GitOps improves the ability to operate the system safely, even when new users need to make big changes. This safety improves iteratively as more tests are automated. It defeats the Leftover Principle by enabling the Compensatory Principle.<sup>b</sup>

### GitOps In a Nutshell

The GitOps workflow has the following qualities:

- Configurations are stored in Git or another VCS (version control system).

The configuration file is in a declarative language or otherwise permits idempotent updates;

- People from outside the team (customers) are empowered to send change requests as PRs. User-facing documentation is available to walk them through the process;

- A CI system runs automated tests to validate the file, and also after approval;

- A human approves the PR, which kicks off the CI/CD system. Some tests are done manually but are automated over time; and

- When the PR is approved and the tests are all passing, a CD system deploys the change into production.

Much of this is not new. People were storing configuration files in VCS decades before Git was invented. IaC

<sup>b</sup> See <https://queue.acm.org/detail.cfm?id=2841313>, <https://queue.acm.org/detail.cfm?id=3197520>, or the primary source, John Allspaw’s article “A Mature Role for Automation,” <https://bit.ly/2Jdt1b6/>

<sup>a</sup> [https://en.wikipedia.org/wiki/Bus\\_factor/](https://en.wikipedia.org/wiki/Bus_factor/)

# IaC: Declarative Languages for System Configuration

A declarative language describes a desired state rather than instructions of what to do. For example, a desired state for a VM configuration system might state, “There should be three virtual machines named foo1, foo2, and foo3.” When the file is processed the first time, the three VMs are created. Processing the same configuration file a second time will leave the system unchanged, as the machines already exist. The configuration file is idempotent. If for some reason foo2 were deleted, processing the file again would re-create foo2.

Contrast this with an imperative configuration language that states, “Add three new virtual machines.” This would generate three additional machines every time it is run, eventually exhausting all resources.

Processing a declarative configuration repeatedly brings a system to (or closer to) a desired state. In fact, many configuration management systems run hourly to detect and fix deviations.

advocates have been extolling the benefits of using declarative languages for configuration files for years.

What’s new is enabling people outside the IT team to submit PRs, the extensive use of automated testing, and using a CI system to integrate all of this.

## Evolutionary Progress

Earlier I described the experiences of a user making a request. The experience from the provider side is more interesting because it evolves over time. In this example I’ll trace a somewhat fictionalized evolution of the use of DNSControl.<sup>c</sup> This is a domain specific language (DSL) that describes domain name system (DNS) configurations (zone data) and a compiler that processes it and then updates various DNS service providers such as AWS (Amazon Web Services) Route 53, Google Cloud DNS, ISC BIND, and more.

The DNSControl configuration file is stored in Git. A CI system is configured so that changes trigger the test-and-push cycle of DNSControl, and updates to the file result in the changes propagating to the DNS providers. This is a typical IaC pattern. When changes are needed, someone from the IT team updates the file and commits it to Git.

To turn this into GitOps, just add documentation. The team’s wiki is updated to point people to the correct Git repository. The file format is pretty obvious: find the domain you want to modify and add the record. The command `dnscontrol check` does some basic checks. Send the PR and you’re off to the races.

The PRs are reviewed by the IT team. The review includes technical concerns such as syntax and string lengths, but also institutional policies such as ordering, capitalization, and a ban on foul language. Initially these checks are done manually. Once the PR is approved, the CI system takes over and triggers the test-and-push cycle.

It is important that the manual checks performed by the IT team are enumerated in the user-facing documentation. This improves the likelihood that a PR is approved on the first try (plus, it is rather unethical to enforce a rule that isn’t advertised). Having the checks enumerated in writing also allows the IT team to be consistent in their policy enforcement. It also serves as a training mechanism: new IT team members can perform the same as anyone else.

This is often good enough. The system meets the basic requirements: it is self-service, work is distributed down to the requestor, and automation does the boring parts. All or most of the checklist is performed manually; but at a low rate of PRs, that is OK.

If the rate of PRs increases, it might be time to evolve the system by automating some of the manual checks. Maybe one is often forgotten or frequently done incorrectly. Maybe an outage could have been prevented by additional bounds checking, and new input validation can be implemented. In fact, after every deployment-related outage, you should pause to ask if a validation or other check could have prevented it.

Syntax checking and file-format validation are often easy to add and reduces frustration. JSON (JavaScript Object Notation) files, in particular, have a syntax that is easy to check with code but not with the eyeball. Without basic automated checks you will see a common pattern of PRs being submitted, followed quickly by a PR with an exasperated comment such as “Damn JSON! Damn comma rule!”

These checks can be added at many stages in the CI pipeline. Git has the pre-commit checks that run on the submitter’s client and block Git commits if they fail, but this can be disabled, or they may fail entirely if they depend on particular operating systems or services. I prefer configuring the PR system to run tests instead. CI systems can be configured to run tests on any new PRs before they are sent to the team for approval. This assures the checks are not skipped and can include more deep testing than can be included in pre-commit checks, which may be difficult to write in an operating system-agnostic manner. Pre-vetted PRs reduce the amount of work for the IT team.

As more of the checklist is automated, the team can handle a larger volume of PRs.

What I like about this evolution is it points coding projects to where they are needed the most. Rather than trying to automate every test and never shipping, you can ship early and often, adding tests that solve real problems as you learn which errors are the most common.

By starting with many manual checks and automating issues only as demand directs your attention, low-value work is discouraged. As stated in books such as *The Goal* (by E.M. Goldratt, 1984) and *The Phoenix Project* (by G. Kim et al., 2013), it is best only to expend effort directly at the bottleneck: improvements upstream of the bottleneck simply create a bigger backlog; improvements downstream of the bottleneck are premature optimizations.

If all the validations, checks, and testing can be automated, the system can automatically approve PRs. While rare, the result is that users can get their needs met 24/7 without delay, even if the IT team is asleep, sick, or on vacation.

<sup>c</sup> <https://github.com/StackExchange/dnscontrol/>

To recap, a GitOps system evolves like this:

1. Basic: Configs in repo as a storage or backup mechanism.
2. IaC: PRs from within the team trigger CI-based deployments.
3. GitOps: PRs from outside the team, pre-vetted PRs, post-merge testing.
4. Automatic: Eliminate the human checks entirely.

### User Benefits

The user benefits from GitOps in many ways. The primary benefit is that a request is completed faster. Waiting for approval is faster than waiting to meet and discuss the request, waiting for the IT team to do the required work, and verifying that it was all done correctly. Even if the PR requires a few iterations and refinements, the process is faster or at least more enjoyable because it is more collaborative.


GitOps offers more transparency. The user can see all the details of the request, thus avoiding potential errors, typos, and the common problem of information being lost in translation.

Technical people value opportunities to learn. Technical users enjoy learning the system as the documentation walks them through the process. The code that processes the PR, checks errors, and deploys the result is often visible to the user, and a curious user can explore it. Requests for additional features and error checks are often pushed to the user, who may enjoy collaboratively improving the system. This can be a recruiting tool for the IT team.


GitOps enhances dignity in the treatment of requests. Requests for environment changes should, ideally, be granted on the basis of whether they conform to architectural and engineering principles, regardless of the source. In a traditional organization, requests are often achieved as a factor of political and bureaucratic success, and force of personality. GitOps moves us toward the better way.

The user also benefits because when it becomes easier for the IT team to create self-service tools, more such tools are created. GitOps lowers the bar to creating these tools.

The downside is that Git has a steep learning curve. This is not so much an issue for developers who already use it and may view GitOps as simply more



**GitOps makes it easy for IT organizations to provide self-service IT operations. It does all that while reducing the amount of code that must be written and maintained.**



things they can submit PRs to. It is a burden for others, however, especially nontechnical users, but also technical people who do not already use Git, such as those in the network operations center, on the help desk, or in other purely operational roles. Luckily, Git is so frustratingly incomprehensible that it has spawned a cottage industry of GUI systems and editor plugins that sit on top of Git and make it easier to use. Some popular GUIs include GitHub Desktop and TortoiseGit. Emacs, vim, and VSCode all have excellent Git integrations.

### IT Benefits

GitOps has all the benefits of IaC. When files that describe an infrastructure are stored in a VCS, all the benefits of using a VCS emerge: version history, log of who made what change, rollback, and so on.

GitOps has benefits far beyond IaC. It democratizes and delegates work. By having users create their own PRs, it creates a division of labor where the users who are experts in the details of the request create the PR, and the team with expertise in the system approves the PRs. It scales the IT team better because approving PRs is relatively easy. It also lets the IT team focus on quality assurance and improving system safety, which is a better use of their time.

While it may take a senior engineer to set up the initial system, automating tests is a good way for junior engineers to build up experience. Thus, GitOps creates mentoring and growth opportunities.

GitOps has security benefits, too. Having a VCS log of all changes makes security audits easier. This turns an organization often perceived as a blocker into an ally. If your VCS is Git or another cryptographically hashed system, silently altering files becomes very difficult. Commits can be cryptographically signed for additional assurance.

Lastly, GitOps enables managers to be better managers. Middle managers can gather metrics from the PR system or the configuration files. Line managers can keep track of what their team is doing by following the PRs rather than nagging their direct reports for status updates.

I like to configure our GitOps sys-



tems to BCC me on any PR updates. As a result, I have become omnipresent and rarely have to nag for status updates. When I do find myself micromanaging a project, it is usually for a project that has not subscribed to the GitOps way of doing things. Another GitOps management trick that is helpful when dealing with low performers: reviewing the person's history of PRs can be enlightening and used as evidence to show the employee when you are coaching him or her on specific problems.

## ROI

The biggest benefits, however, are derived from GitOps' ability to improve the ROI for automation. The truth is that we do not have time to automate all requests, and not all requests have sufficient ROI to make automation worthwhile. GitOps reduces the I, making it easier to achieve the R.

Traditionally, self-service IT systems involve creating a Web-based portal that permits users to perform well-defined, transactional requests without the involvement of a human approver. Such systems are difficult to create, however, requiring Web UI design skills that are often beyond those of a typical system administrator. The workflow is sufficiently complex that advanced user experience (UX) research and testing would be required to create a system that is less confusing than just opening a ticket. Sadly, most companies do not have UX research staff, and those that do will not allocate them to internal IT projects.

Such systems are brittle, as they are tightly coupled to the system they control. I have seen portals that were abandoned when the underlying system changed because less effort was required to return to manual updates than to update the portal. This is often not because the change was complex, but because the knowledge of how to update the portal left with the person who originally created it. Often portals are crafted by people different from those responsible for the systems they control.

GitOps lowers the bar for creating self-service systems since the UI is the existing PR system that the company already has.

## Tips

My advice for getting started is to use the existing VCS, PR, and CI systems in place at your organization. People are familiar with them already, which reduces the learning curve. They often have many nice features such as a way to manage the queue of PRs waiting for approval, integration with ticket systems, and so on. I'm fond of systems that can announce the arrival of new PRs in my team's chat room. You can even make PR approvals as easy as sending a message to the chatbot.

## Use Cases

Look for opportunities to use GitOps. DNS is an obvious place to start, as are VM creation, container maintenance and orchestration, firewall rules, website updates, blog posts, email aliases and mailing lists, and just about any virtual infrastructure or one with a configuration file or API.

GitOps is pervasive in our industry. At OpenStack the entire infrastructure is controlled via GitOps, including the GitOps infrastructure itself. Employees at GitHub Inc. report the use of GitOps is pervasive, and even nontechnical employees are trained on how to use Git.

The popular open source dashboarding application Grafana is moving toward a GitOps workflow for dashboards. Recognizing how much business logic is encoded in these dashboards, Grafana now offers the option to provision dashboards from JSON dashboard definitions dropped into a specific location on the file system. With a GitOps approach, dashboard creators can use whatever tooling and process makes sense for their particular workflow—including the use of PRs on the Git repo storing the dashboards.

One company maintained an inventory of where equipment was positioned in computer racks around the globe as a set of YAML (YAML Ain't Markup Language) files. Technicians kept the files up to date via PRs that triggered automatic calculations to detect overloaded power systems and validate other system constraints. The CI system also generated HTML pages with diagrams of the racks. When a Web-based GUI was request-

ed, they created a system that, under the hood, was updating the YAML files in their VCS.


## Conclusion

GitOps lowers the cost of creating self-service IT systems, enabling self-service operations where previously they could not be justified. It improves the ability to operate the system safely, permitting regular users to make big changes. Safety improves as more tests are added. Security audits become easier as every change is tracked.

GitOps isn't perfect. Not every IT system or service can be configured this way, and it requires a commitment to writing user-facing documentation that may take some getting used to. It also requires higher security controls around your VCS, as it is now a production attack vector.

When GitOps becomes embedded in the technical culture of the company, however, new systems are built with GitOps in mind, and we move one step closer to a world where operations are collaborative, shared, and safe.

## Acknowledgments

This article benefited from feedback and suggestions from Alice Goldfuss, SRE, GitHub Inc.; Elizabeth K. Joseph, developer advocate, Mesosphere; Chris Hunt, SRE, Stack Overflow Inc.; Eric Shamow, lead platform engineer, StanCorp; Jonathan Kratter, SRE, Uber Technologies LLC; and Mark Henderson, SRE, Stack Overflow Inc. 

## Related articles on [queue.acm.org](https://queue.acm.org)

### Are You Load Balancing Wrong?

Thomas A. Limoncelli

<https://queue.acm.org/detail.cfm?id=3028689>

### Making Sense of Revision-Control Systems

Bryan O'Sullivan

<https://queue.acm.org/detail.cfm?id=1595636>

### Containers Will Not Fix Your Broken Culture

Bridget Kromhout

<https://queue.acm.org/detail.cfm?id=3185224>

**Thomas A. Limoncelli** is the SRE manager at Stack Overflow Inc. in New York City. His books include *The Practice of System and Network Administration*, *The Practice of Cloud System Administration*, and *Time Management for System Administrators*. He blogs at [EverythingSysadmin.com](http://EverythingSysadmin.com) and tweets @YesThatTom

Copyright held by owner/author.  
Publication rights licensed to ACM. \$15.00.

Copyright of Communications of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.