What Is GitOps

June 15, 2021

A year ago we published an introduction to GitOps - Operations by Pull Request. This post described how Weaveworks ran a complete Kubernetes-based SaaS and developed a set of prescriptive best practices for cloud native deployment, management and monitoring.

The post was popular. Other people talked about GitOps and published new tools for git push, development, secrets, functions, continuous integration and more. Our website grew with many more posts and GitOps use cases. But people still had questions. How is this different from traditional infrastructure as code and continuous delivery? Do I have to use Kubernetes?

We soon realized that we needed a new jumping off point, with:

1. More examples and stories
2. A concise definition of GitOps
3. A comparison with traditional continuous delivery

This blog provides that. Read on for an updated introduction to GitOps, CICD and Developer Experience. We focus mainly on Kubernetes although

the model may be generalized.

A Story of GitOps

Imagine Alice. She runs a business "Family Insurance" that sells health, travel, car and property insurance packages to parents who are too busy to figure out the details by themselves. The business began as a side project when Alice was working as a data scientist in a bank and she realized she could use advanced data algorithms to analyse and assemble family packages more efficiently than by hand. Some investors funded the project and now the business turns over $20M per year and is growing fast with 180 staff in a variety of roles. Among those is a technology team that builds and operates the website, database and customer analytics. This team has 60 staff and is led by Alice's CTO, Bob.

Bob's team run their production systems in the cloud. Their main applications run on GKE - using Kubernetes on Google Cloud - and they use several data and analytics tools as well.

Family Insurance hadn't planned to use containers but got caught up in the excitement around Docker. They soon found that using GKE made it easy to set up clusters for testing new features. They added Jenkins to provide a CI solution and used Quay for the container registry. Then they wrote some Jenkins scripts that pushed new containers and config to GKE.

Today though, Alice and Bob are getting frustrated with the team's velocity, and how it affects the pace of their business. Introducing containers didn't seem to improve velocity as much as the team hoped. Sometimes, deployments break and it's not clear that a code change was responsible. Config changes are also hard to track. Often they have to create a new cluster, and re-deploy their apps, as it's the easiest way to undo the kind of mess the system turns into. Alice is worried this will get

worse as the app grows, and there is a new ML project looming on the horizon. Bob feels that he has automated a lot of manual processes - so why is the pipeline still brittle, requires manual intervention and unscalable?

Then they find out about GitOps and it enables them to move forward - faster.

Alice and Bob's team have spent years hearing about Git-based workflows, devops and infrastructure as code. It's not super new. But GitOps brings a set of opinionated and prescriptive best practices about applying these ideas in the context of Kubernetes. There are quite a few people and vendors talking about it, including on Weaveworks' blog.

The Family Insurance team adopt GitOps. Now they have an automated operating model that works for Kubernetes and combines 'speed' with 'stability' because they:

- Find that the team gets more than 2x the work done and nobody is distracted.
- Stop maintaining scripts. Instead: they can focus on new features and advance their engineering practices, e.g. implement canary deployments and improve testing.
- Have a deployment process that rarely breaks.
- Recover deployments from partial failures correctly without manual intervention.
- Gain greater confidence in their delivery systems. Alice and Bob now find it easy to split up the team into microservices teams all working in parallel.
- Execute 30-50 changes per day or more with each team, and try out new techniques.
- Onboard new developers quickly, who can deploy code to production ("by pull request") within hours.
- Easily pass SOC2 compliance.

What just happened?

GitOps is two things:

1. An operating model for Kubernetes and cloud native. It provides a set of best practices to join up deployment, management and monitoring for containerized clusters and applications. An elegant "1 slide" definition from Luis Faceira is shown below.

2. A path towards a developer centric experience for managing applications. We're applying the Git workflow to operations, as well as development. Note that this is not just about Git push, it's about how we set up the entire CICD toolchain and UI/UX.

A note on Git

If you are not familiar with version control systems and Git-based development workflow, we highly recommend diving in. Working with feature branches and pull requests can seem like black magic at first, but it is worth the effort. Here is a good article to get you started.

How Kubernetes works

In our story, Alice and Bob came to GitOps after using Kubernetes. Indeed, GitOps has a strong affinity with Kubernetes - it is an operating model for Kubernetes-based infrastructure and applications.

What does Kubernetes give users?

Here are some key properties:

1. You can describe everything in the Kubernetes model as a declaration.
2. The Kubernetes API server accepts a declaration as an input, and then continually tries to drive the cluster to the state described in the declaration.
3. The declarations are adequate to describe and manage a broad class of workloads, or "applications".
4. Changes to the application and cluster are then due to either
    - Changes to container images, or
    - Changes to the declarative specification
    - Errors in the environment, eg. a container crashes

Kubernetes' Excellent Convergence Properties

If a group of configuration updates are made by a human operator, the Kubernetes orchestrator will apply changes to the cluster until its state

has *converged to the updated configuration*. This model works for any Kubernetes resource and is extensible using Kubernetes Custom Resource Definitions (CRDs). Therefore Kubernetes deployments have the following excellent properties:

- **Automation:** Kubernetes updates provide a mechanism for automating the process of applying a set of changes correctly and in a timely manner.
- **Convergence:** Kubernetes will keep trying to update until success.
- **Idempotence:** multiple applications of convergence have the same outcome.
- **Determinism:** assuming adequate resources, the updated cluster state depends only on the desired state.

How GitOps works

We've learnt enough about Kubernetes to explain GitOps.

Let's go back to Alice and Bob's microservices teams at Family Insurance. What are some of the workflows that they want to carry out? Have a look at the table below. If there are parts that seem new or strange, please suspend disbelief and bear with us. This is just an example workflow using Jenkins. There are many other workflows that use alternative tools.

What we can see is that every update ends up with changes to the config files and Git repos. These changes in Git then result in the "GitOps operator" updating the cluster.

Workflow	Tasks executed
Jenkins build passes: master branch	• Jenkins pushes tagged images to Quay • Jenkins pushes config and Helm charts to master storage bucket • Cloud function copies config+charts from master storage bucket to master git repo. • GitOps operator updates cluster

Jenkins build passes: release or hotfix branch	• Jenkins pushes untagged images to Quay • Jenkins pushes config and Helm charts to staging storage bucket • Cloud function copies config+charts from staging storage bucket to staging git repo • GitOps operator updates cluster
Jenkins build passes: develop or feature branch	• Jenkins pushes untagged images to Quay • Jenkins pushes config and Helm charts to develop storage bucket • Cloud function copies config+charts from develop storage bucket to develop git repo • GitOps operator updates cluster
New customer added	• LCM/ops invokes Gradle to do initial deployment and setup NLBs • LCM/ops commits new config file to setup deployment for updates • GitOps operator updates cluster

A concise description of GitOps

1. Describe the desired state of the whole system using a declarative specification for each environment. (In our story, Bob's team owns the whole system config in Git.)

    - A git repo is the single source of truth for the desired state of the whole system.
    - All changes to the desired state are Git commits.
    - All specified properties of the cluster are also observable in the cluster, so that we can detect if the desired and observed states are the same (converged) or different (diverged).

2. When the desired and observed states are not the same then:

    - There is a convergence mechanism to bring the desired and observed states in sync both eventually, and autonomically.

Within the cluster, this is Kubernetes.
    - This is triggered immediately with a “change committed” alert.
    - After a configurable interval, an alert “diff” may also be sent if the states are divergent.

3. Hence all Git commits cause verifiable and idempotent updates in the cluster.

    - Rollback is: “convergence to an earlier desired state”.

4. Convergence is eventual and indicated by:

    - No more “diff” alerts during a defined time interval.
    - A “converged” alert (eg. webhook, Git writeback event).

What is divergence?

To repeat: *All specified properties of the cluster are also observable in the cluster.*

Some examples of divergence:

- A change in a configuration file, due to a merged update in Git.
- A change in a configuration file, due to the GUI effecting a commit to Git.
- Multiple changes in the desired state due to a merged PR to Git followed by a container image build and subsequent config updates.
- The observed state changes in the cluster due to an error, resource conflicts causing poor behavior, or just random “drift” from the original state.

What is the convergence mechanism?

Some examples:

- For containers and clusters, the convergence mechanism is provided by Kubernetes.
- The same mechanism can be used to manage Kubernetes-based applications and frameworks (eg. Istio, Kubeflow).
- The mechanism to manage the workflow between Kubernetes, the image repos and Git is provided by the Weave Flux GitOps operator that is a part of Weave Cloud.
- For the underlying machines, the convergence mechanism should be declarative and autonomic. In our experience Terraform is the closest to this but does require human oversight. In this sense, GitOps extends the tradition of Infrastructure as Code.

GitOps combines Git with Kubernetes' excellent convergence properties to provide a model of operations.

GitOps lets us say: *only systems that can be described and observed can be automated and controlled*.

GitOps is for the whole Cloud Native stack - eg. Terraform & more

GitOps is not only about Kubernetes. In GitOps we want the whole system to be managed declaratively and using convergence. By a "whole system" we mean a collection of environments running Kubernetes, eg. "dev cluster 1", "production". Each environment includes machines, clusters, applications, as well as interfaces to external services eg. data, monitoring.

Notice how important Terraform is for the bootstrapping problem here. Kubernetes has to start somewhere, and using Terraform means that we can apply the same GitOps workflows to create a control plane that underpins Kubernetes and applications. This is a helpful best practice.

Applying GitOps concepts to the layers above Kubernetes is an exciting

area of active development. So far we have seen GitOps type solutions for Istio, Helm, Ksonnet, OpenFaaS and Kubeflow, as well as eg. Pulumi who are re-inventing the app dev layer for cloud native.

Kubernetes CICD - comparing GitOps to other approaches

We said GitOps is two things:

1. An operating model for Kubernetes and cloud native, described above
2. A path towards a developer centric experience for managing applications

For many people, GitOps is all about the Git push workflow. We like this too :-) But there is more to it: Let's now look at CICD pipelines.

GitOps provides Continuous Deployment for Kubernetes

GitOps provides a mechanism for continuous deployment that removes any need for standalone "deployment management systems". Kubernetes does the work for you.

- A management update to the application requires an update in Git. This is a transactional update to the desired state. "Deployment" is then enacted within the cluster by Kubernetes itself based on the updated description.
- Because of how Kubernetes works, these updates are convergent. This provides a mechanism for continuous deployment in which all updates are atomic.
- Note: Weave Cloud provides a GitOps operator that integrates between Kubernetes and Git to enable CD by reconciling desired and cluster state.

No kubectl, no scripts

You should avoid using Kubectl to update the cluster and especially avoid using scripts to group kubectl commands. Instead, with a GitOps pipeline in place a user can update their Kubernetes cluster via Git.

Benefits include:

1. **Correctness.** A group of updates may be applied, converged and finally validated, which gets us closer to the goal of atomic deployment. By contrast using scripts provides no guarantee of convergence. We say more about this below.
2. **Security.** To quote Kelsey Hightower, "Limit the scope of access to a Kubernetes cluster to automation tools and cluster administrators who may have to debug it or keep it running." See also my blog post on Security and Compliance as well as this recent use case of Homebrew being hacked via credentials in a careless Jenkins script.
3. **Developer Experience.** Kubectl exposes the machinery of the Kubernetes object model, which is quite complex. Ideally users should interact with the system at a higher level of abstraction. Again I shall defer to Kelsey here - please see this summary.

The difference between CI and CD

GitOps improves on existing CICD models.

The modern CI server is an orchestration tool. Specifically it is a tool for *orchestrating CI pipelines*. These include build, test, merge to trunk, etc. CI servers automate management of complex multi-step pipelines. A common temptation is to script a set of Kubernetes updates and execute that script as a pipeline step to "push" changes to the cluster. Indeed many people do this. However it is sub-optimal and we shall now talk about WHY.

CI should be used to merge updates to trunk, and the Kubernetes cluster should update itself to manage CD “internally” based on those updates. We call this the “pull” model for CD in contrast to CI “push”. CD is part of *runtime orchestration*.

Why CI servers should not do CD using direct updates to Kubernetes

Don't use a CI server to orchestrate direct updates to Kubernetes as a set of CI jobs. This is an anti-pattern as set out in this blog post.

Let’s go back to Alice and Bob.

What kind of problems did they encounter? Bob’s CI server is applying changes to the cluster, but if it fails, then Bob doesn't know what state the cluster is in, or is supposed to be in, or how to rectify it. This is also true if it succeeds.

Let’s assume that Bob’s team built a new image and then patched their deployments to deploy the image, all from a CI pipeline.

If the image builds OK, but the pipeline fails, the team is left to figure out:

- Did the update deploy or not?
- Do we trigger a new build? Will that result in side-effects that we don't want to do again -- and with the potential of having two builds of the same immutable image hanging around?
- Do we wait for another update before triggering the build?
- Which bit failed? What steps need to be replayed (and which steps are safe to replay)?

Using Git-based workflows doesn't guarantee that Bob’s team will not encounter these problems. They could still fail to push a commit, or move a tag, or whatever; but, it's much closer to being a clear pass-or-fail.

In summary - the reason CI servers should not do CD is:

- Update scripts are not always deterministic and it's easy to make mistakes.
- CI servers do not converge to a declarative model of the cluster, they fail out.
- Idempotency guarantees are hard. Users need to understand deep system semantics.
- Recovery from partial failure is harder.

NOTE on Helm: If you want to use Helm, we recommend combining it with a GitOps operator, eg. Flux-Helm. This will help with convergence. Helm itself is not deterministic or atomic.

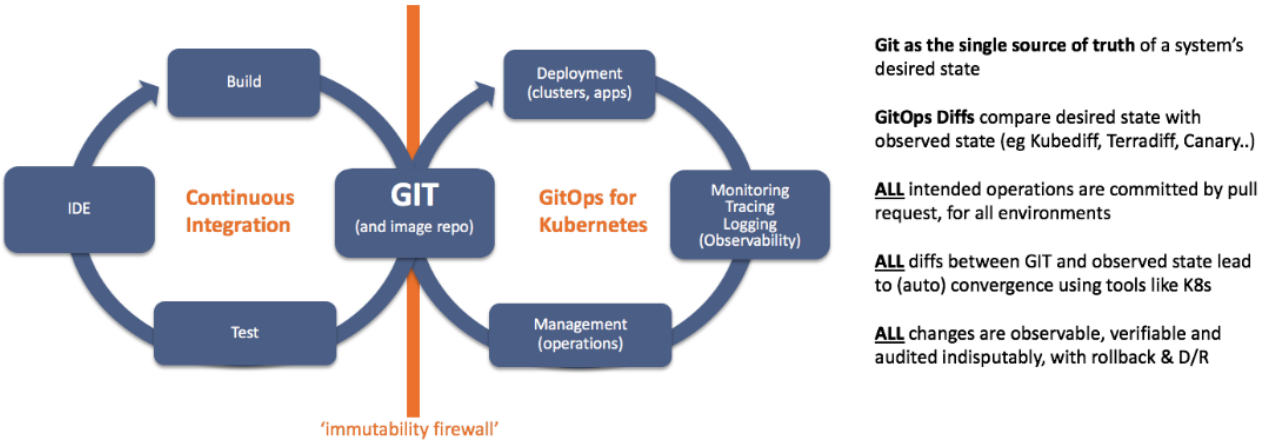
GitOps is the best way to do Continuous Delivery for Kubernetes

Alice and Bob's team adopt GitOps and find that it is much easier to deliver their products, at high velocity and without losing stability. Let's finish this blog post with some illustrations of what their new set-up looks like. Note that we are talking below about applications and services primarily, but you can also use GitOps to manage the whole platform.

Operating Model for Kubernetes

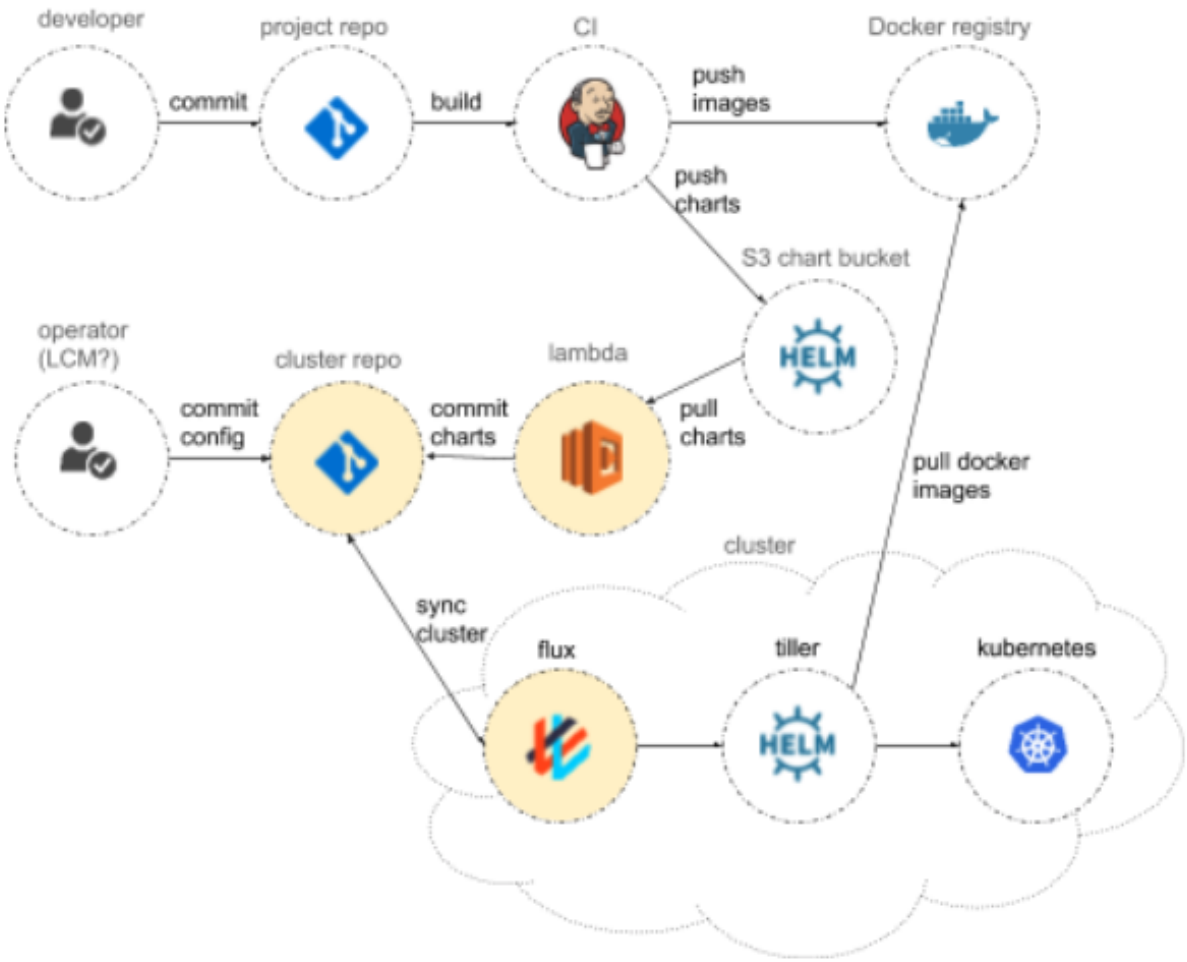
Please review the following diagram. This shows Git and the container image repository as shared resources between two orchestrated lifecycles:

- A continuous integration pipeline that reads and writes files to Git and can update the container image repository
- A GitOps runtime pipeline that combines deployment with management and observability, which reads and writes files to Git and can load container images



What are the key takeaways?

1. **Separation of Concerns:** Note that the two pipelines can only communicate by updating Git or the image repo. In other words there is a firewall between CI and runtime operations. We call this the immutability firewall because all updates to the repos create a new version. See also slides 72-87 of this presentation for more info on this topic.
2. **Use any CI and Git server:** GitOps works with ANY of these. Please keep on using your favorite CI servers, Git, image repos, and test kits! Almost all the other Continuous Delivery tools in the market want you to use their own CI, or Git server, or image repo. That can be a blocking factor in the adoption of cloud native. With GitOps, you can keep using your existing tools.
3. **Events as an integration tool:** Whenever Git is updated, the runtime is notified by Weave Flux (or Weave Cloud operator). Whenever Kubernetes accepts a set of changes, then Git is updated. This provides a simple integration model for creating workflows for GitOps, as illustrated below.



To conclude

GitOps provides strong update guaranties which should be required of any modern deployment of “CICD” tool:

- Automation
- Convergence
- Idempotence
- Determinism

This is important because it provides an operating model for developers in cloud native.

- Traditional systems management and monitoring tools are associated with operations teams working from a runbook associated with a

specific deployment.
- In cloud native systems management, observability tooling is the best way that a development team can measure the effects of deployments fast enough to react to them.

Imagine having many clusters deployed across many clouds, and many services each with their own teams and deployment plans. GitOps provides a scale invariant model for managing all this.

Get Started or Find Out More

To try GitOps:

- Sign up for Weave Cloud and use it with any Kubernetes cluster
- Contact sales to discuss managing your Kubernetes platform with GitOps
- Go the open source route and try Weave Flux

To find out more you can:

- Check out our other blog posts on GitOps
- Read our "GitOps - what you need to know" page
- Read one of our Getting Started Guides

Alexis is the co-founder and CEO of Weaveworks. He is also the chairman of the TOC for CNCF, and the co-founder of the Coed:Code meetups. Previously he was at Pivotal, as head of products for Spring, RabbitMQ, Redis, Apache Tomcat and vFabric. Alexis was responsible for resetting the product direction of Spring and transitioning the vFabric business from VMware. Alexis co-founded RabbitMQ, and was CEO of the Rabbit

company acquired by VMware in 2010, where he worked on numerous cloud platforms. Rumours persist that he co-founded several other software companies including Cohesive Networks, after a career as a prop trader in fixed income derivatives, and a misspent youth studying and teaching mathematical logic.

< Previous

Racy conntrack and DNS lookup timeouts

Next >

How to Correctly Handle DB Schemas During Kubernetes Rollouts

You may also like:

June 22, 2021

GitOps Days 2021 Day 1 Recap: Evolution, Revolution, and Vision

GitOps boosts collaboration, reliability and autonomy for MediaMarktSaturn

June 10, 2021

The GitOps Maturity Model - 4 evolutionary steps to continuous delivery