GitOps - Operations by Pull Request

At Weaveworks, developers are responsible for operating our Weave Cloud SaaS. "GitOps" is our name for how we use developer tooling to drive operations. This post talks about GitOps, which is 90% best practices and 10% cool new stuff that we needed to build. Fair warning: this is what works for us, and dear reader, you may disagree.

Git is a part of every developer's toolkit. Using the practices outlined in this post, our developers operate Kubernetes via Git. In fact, we manage and monitor all of our applications and the whole 'cloud native stack' using GitOps. It feels natural and less intimidating to learn, and the tools themselves are very simple.

Git as the Source of Truth

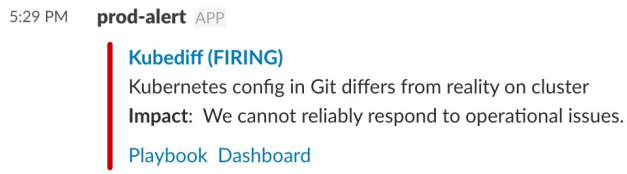
For the last two years, we've been running multiple Kubernetes clusters and Prometheus telemetry databases on Amazon Web Services. You can read more about how we provision Kubernetes in the blog post, "Provisioning And Lifecycle Of A Production Ready Kubernetes Cluster".

What exactly is GitOps? By using Git as our source of truth, we can operate almost everything. For example, version control, history, peer review, and rollback happen through Git without needing to poke around with tools like kubectl.

- Our provisioning of AWS resources and deployment of k8s is declarative

- Our entire system state is under version control and described in a single Git repository
- Operational changes are made by pull request (plus build & release pipelines)
- Diff tools detect any divergence and notify us via Slack alerts; and sync tools enable convergence
- Rollback and audit logs are also provided via Git

GitOps Alerts - an example



Let's say a new team member deploys a new version of a service to prod without telling the on-call team. Our diff tools detect that what is running does not match what is configured to run (ie. the image specified in the Git repo is different from the one deployed in production). The diff tools fires an alert. The on-call engineer asks about it and determines the intent of the change.

Declarative tools love using Git as source of truth

Kubernetes is just one example of many modern tools that are "declarative". Declarative means that configuration is guaranteed by a set of facts instead of by a set of instructions, for example, "there are ten redis servers", rather than "start ten redis servers, and tell me if it worked or not".

By using declarative tools, the entire set of configuration files can be version controlled in Git. This means that Git is the source of truth. And it also means you get code reviews, comments in the configuration files, and links to any issues in commit messages and PRs. All of this makes the system (and the reasons behind it!) discoverable and easier to operate, recover, and observe. You can even include version-controlled product user stories to help understand the evolution of intent as well as the implementation.

In the case of Kubernetes, we use version control not only for code but also for the YAML files that define the Kubernetes Deployments, Services, DaemonSets, etc. We also use Terraform and Ansible to provision Kubernetes on Amazon, and these are also version controlled in Git.

What if my system diverges from the source of truth?

Declarative provisioning tools are cool; they let you describe your desired true state in Git. But they suffer from the problem that what is “really true right now” is in the live system, and may differ from what is described in source control. How do we know when the live system has converged to the desired state? How do we get notified when this differs? What is the “canary in the coal mine” that can tell us when we are in trouble? How do we trigger convergence?

There is prior art here.

Tools like Chef, Puppet and Ansible support features like “diff alerts”. These help operators to understand when action may need to be taken to “converge” the live system to the intended state (as defined by the configuration scripts). And more recently, best practice is to deploy immutable images (eg. containers) so that divergence is less likely.

In the “GitOps” model, we use Git to solve for divergence and convergence, aided by a set of “diff” and “sync” tools that compare intended with actual state. A full write up is here.

Essential GitOps tools

In this section I shall talk about Weaveworks tools. This is an intro - more will follow in a second post on this topic.

Our product Weave Cloud provides tools for cloud native applications using GitOps patterns. The core of our GitOps machinery is the CICD tooling. For us, the critical piece is continuous deployment (CD) and release management. This is based on our open source project Weave Flux which supports Git-

cluster synchronisation, and so is designed for version controlled systems and declarative application stacks.

In addition, we have 3 main 'diff' tools: kubediff, ansiblediff, and terradiff. Each one compares the latest Git to what's running in a deployed environment. Thus, we run these tools on our dev cluster to see the diff between Git and dev, and on our prod cluster to see the diff between Git and prod. There's a Prometheus metric for "is there a diff". This metric fires a warning that goes off if there's a diff for more than 1h (or something similar).

For example:

kubediff is a command-line tool that shows the differences between your running configuration and your version controlled configuration. If there is a difference between them, it returns a non-zero exit code. [from "Monitoring Your Kubernetes Infrastructure With Prometheus"]

Why does this matter? Without something like this, small ad-hoc changes will gradually build up, and you can no longer distinguish between how the system should be and how it is—you don't know what's deliberate and what's accidental. It also means that whatever is in the Git repo will become hopelessly out of date.

An example use case: An engineer changes a flag on dev to test out a new change that might increase performance. They do the experiment, gather the results, but then forget to reset the flag. Luckily, an alert pops up in Slack 30 minutes later reminding them to clean up after themselves.

GitOps is a way to manage systems like Kubernetes

Let's recap - the use case is that we are provisioning a full cloud native stack including Kubernetes. As set out in the full write-up of this case:

The system presented here allows for the entire state of system to be stored in version control, and for the that state to be checked and enforced by three jobs: terradiff, ansiblediff and

kubediff. All the configuration files for the cluster are version controlled, and modification to any component can be rolled out with zero downtime and rollback easily if they break anything. We're starting to live up the Google SRE saying of "changing the tires of a race car as it's going 100mph".

Recovery from total wipeout of your system

So let me tell you a story about how the GitOps approach has helped us.

The scene: spring 2016, a peaceful morning in London. The sun is shining. Birds tweet.

"I'm about to make a change that will probably wipe out all our systems"

"Tom, are you sure we want to do that?"

<click>

"Oops - I've just deleted all our Kubernetes clusters on AWS"

CHAOS ENSUES

Recovery from a total system wipeout is different from fixing day to day failures. What happened next? It took the team less than 45 minutes to completely rebuild our entire systems. This included setting up multiple AWS services: EC2 AMIs, ELBs, SQS, DynamoDB, etc, all our Kubernetes clusters and applications, and our entire observability stack. 45 minutes was a pretty good outcome - made possible by "GitOps". (Read up on how we are running Kubernetes on AWS and what you need to know)

Our wipeout story shows how important it is to optimise for mean time to recovery (MTTR). This also leads to customer happiness. If I may quote Brian Hatfield channeling Charity Majors on Twitter, good operations practices lead to "tight iteration times and high value to customers".

GitOps makes life easier

Let's summarise.

At Weaveworks, our mission is to empower developers to do operations.

Git has moved the state of the art forward in development. A decade of best practices says that config is code, and code should be stored in version control. Now it is paying that benefit forward to Ops. At Weaveworks we now find that we prefer to fix a production issue via a pull request, than to make changes to the running system.

None of this matters unless it makes your application clusters easier to understand and use. GitOps practices have made Kubernetes and cloud native apps much easier for us. We truly think you will find this too. The tools themselves are very simple. We're baking them into our product, Weave Cloud.

Contact us for a demo any time, or just sign up.

In future blog posts, I shall try to set out a list of GitOps principles & "how to do GitOps". This will include information about how we use containers, CD pipelines (The GitOps Pipeline - Part 2), and a version-controlled observability stack. I'll also talk more about how GitOps is an iteration of existing approaches: DevOps, "infrastructure as code", etc. Watch this space.

UPDATE (August 2018): Read an updated intro here

-- Alexis

weave-cloud continuous-delivery

ABOUT THE AUTHOR

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

NEXT >

You may also like:

Kubernetes | Architecture

Why You Need a Multicloud Strategy?

Kubernetes | Architecture

Firekube - Fast and Secure Kubernetes Clusters Using Weave Ignite

Kubernetes | Hot new technology

Automate EKS Cluster Configuration with GitOps and Eksctl