Feedback and Control - an Essential GitOps Component

An important and often overlooked component of GitOps is the concept of a feedback and control loop. In this post, we're going to take a look at what this means and why this is essential to GitOps. You may not be doing GitOps, if you don't have a way of monitoring, observing and alerting on divergences from your 'source of truth'.

GitOps and Continuous Delivery

Briefly, GitOps is a way to do continuous delivery. It is an operations model for managing your deployments to Kubernetes. It works by keeping all of your declarations in Git alongside your code. To make a change to your cluster or to your code requires an approved pull request. And as we'll expand on in this post, GitOps done right, means that your system can monitor and then alert you whenever there is a difference between 'the source of truth' kept in Git and your running cluster.

The ultimate goal of GitOps is to speed up development so that your team can make changes and updates safely and securely to complex applications running in Kubernetes.

What do we mean by a feedback and control loop?

When we talk about feedback and control, we mean that developers have observability built-in to their deployment workflows that provide the data for an informed decision. Before the deployment is rolled out, a final health check can be made against your running cluster before committing to that update that gives you a final piece of mind.

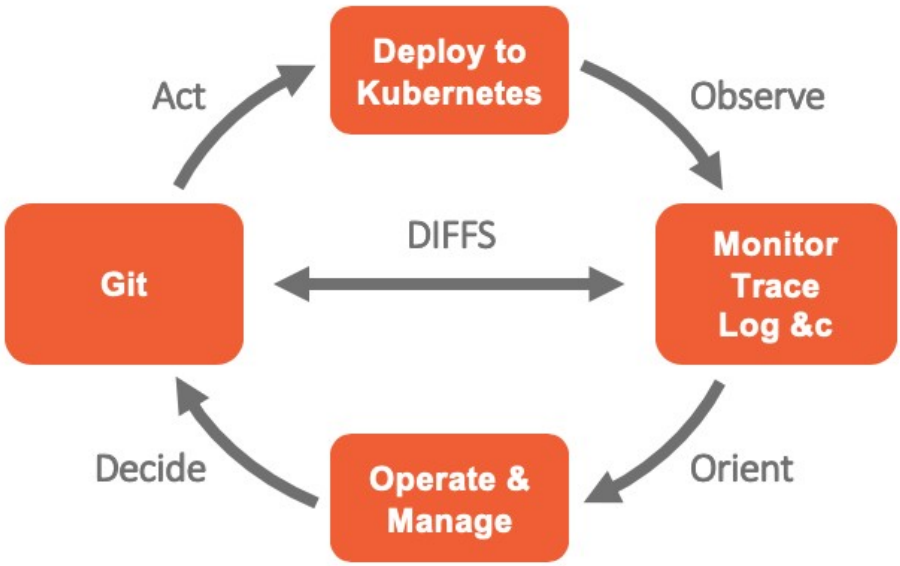
With a feedback control loop you can effectively answer the following questions:

- How do I know if my deployment will succeed?
- How do I know if the live system has converged to the desired state?
- Can I be notified when this differs?
- Can I trigger a convergence between the cluster and source control?.

While Git is the source of truth for the desired state of the system, Observability provides a benchmark for the actual production state of the running system. In GitOps we take advantage of both to manage our applications.

With “GitOps”, divergence and convergence is achieved with a set of “diff” and “sync” tools (kubediff, as well as terradiff and ansiblediff) that compare the intended state with actual state. Diff tools are also used to alert developers when the deployment is out of sync.

A feedback loop with diffs and built-in observability looks something like this where observability provides feedback for developers and operators to make key decisions about deployments and the system:



Real-Time Feedback with Built-in Observability

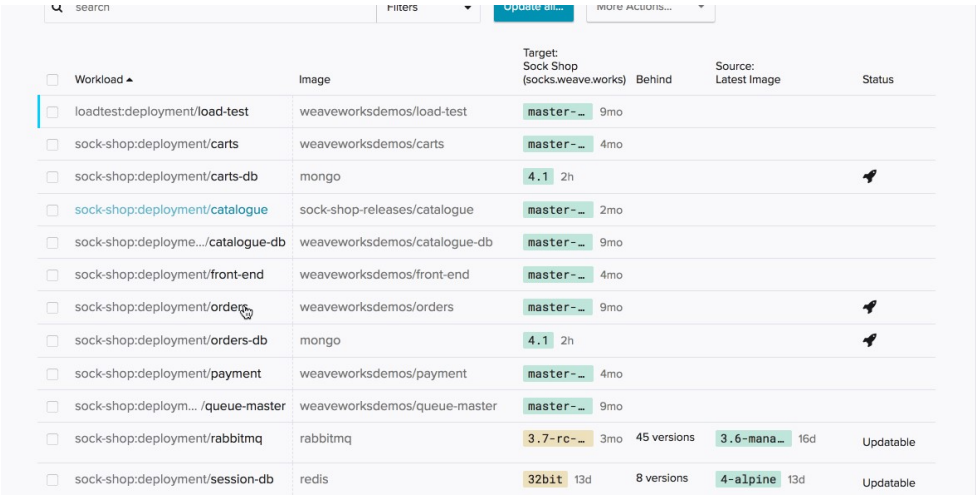
Our product, Weave Cloud provides continuous deployment (CD) and Git-cluster synchronization with built-in observability. This means your team can be alerted when deployments don't go as planned and that your team can make informed decisions before deploying updates.

How do GitOps Workflows and Observability Work Together?

Because about to be released services can be observed in real-time within the running cluster before you release, you can deploy with confidence and deliver better quality features more quickly.

Observability is a principal driver of the Continuous Delivery cycle for Kubernetes since it describes the actual running state of the system at any given time. The running system is observed in order to understand and control it. After new features and fixes are merged to git, the deployment pipeline is triggered, and once the image is ready to be released, it can be observed in real-time against the running cluster. At this point, the developer may return

to the beginning of the pipeline based on this feedback or deploy and release the image to the production cluster.

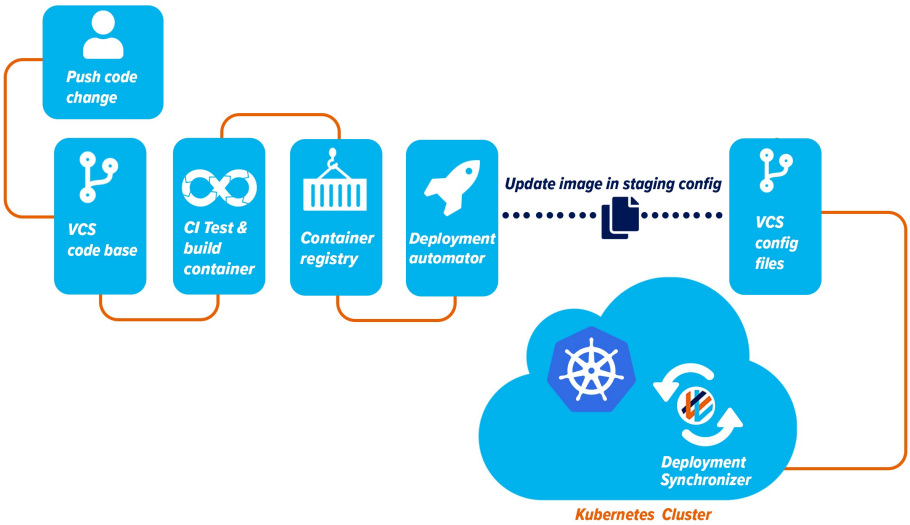


Built-in Observability Dashboards

Typical Deployment Workflow

This is the typical developer workflow for creating or updating a new feature in Weave Cloud:

1. A pull request for a new feature is pushed to GitHub for review.
2. The code is reviewed and approved by a colleague. After the code is revised, it is merged to Git.
3. The Git merge triggers the CI and build pipeline, runs a series of tests and then eventually builds a new image and pushes the image to the new image to a registry.
4. Weave Cloud's deployment automator watches the image registry, notices the image, pulls the new image from the registry and then updates its manifest and checks them into the config repo.
5. The Weave Cloud deployment synchronizer detects that the cluster is out of date. It pulls the changed manifests from the config repo and then deploys the new feature to production.



Final Thoughts

In this post, we discussed what we mean by a feedback and control loop and how observability is an important part of GitOps. Without built-in observability the overall health of the system and their deployments are impossible to assess.

Download our eBook: A Practical Guide To GitOps

ABOUT THE AUTHOR

Anita has over 20 years experience in software development. She's written technical guides for the X Windows server company, Hummingbird (now OpenText) and also at Algorithmics, Inc. She's managed product delivery teams, and developed and marketed her own mobile apps. Currently, Anita leads content and other market-driven initiatives at Weaveworks.

< PREVIOUS

NEXT >

You may also like:

DECEMBER 27, 2017

Gitops

GitOps - modern best practices for high velocity application development

OCTOBER 17, 2017

Gitops | Prometheus | Kubernetes | Product features

GitOps Part 3 - Observability

AUGUST 07, 2017

Gitops | Kubernetes | Product features

GitOps - Operations by Pull Request