weaveworks

Automating Kubernetes with GitOps

weaveworks

Automating Kubernetes with GitOps

WHITEPAPER

Companies that want to go fast, need to deploy more often, more reliably and with less overhead. GitOps is a fast, and secure method for developers to maintain and update complex applications running in Kubernetes.

Since Kubernetes and many other **cloud native technologies** are almost entirely declarative, infrastructure definitions can be kept alongside application code in git. Keeping your entire system in git means that your development team uses familiar git-based workflows and pull requests to apply both application and infrastructure changes to Kubernetes.

With the entire state of your cluster kept under source control, diff tools and synchronization agents can compare what's running in production with what's under source control and when a divergence is detected between the two, an alert can be sent, effectively creating a feedback and control loop for managing your cluster.

AN OPERATING MODEL FOR BUILDING CLOUD NATIVE APPLICATIONS

At its core, GitOps is these two things:

1. An operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify deployment, management and monitoring for containerized clusters and applications.
2. A path towards a developer experience for managing applications; where end-to-end CICD pipelines and git workflows are applied to both operations, and development.

Kubernetes automation with freedom of choice

GitOps gives you the freedom to choose the best tools for the different parts of your CICD pipelines. You can select tools from the open source ecosystem or from closed source and depending on your needs, you may even combine them.

Whatever tools you choose for your deployment and delivery pipelines, applying GitOps best practices should be an integral component of your continuous delivery process. Doing so will make building and adopting a continuous delivery culture into your organization easier.

KEY BENEFITS OF GITOPS

Applying GitOps best practices are far reaching and provide:

Stronger security guarantees

Git's strong correctness and security guarantees, backed by the strong cryptography used to track and manage changes, as well as the ability to sign changes to prove authorship and origin are key to a correct and secure definition of the cluster's desired state. If a security breach does occur, the immutable and auditable source of truth can be used to recreate a new system independently of the compromised one, reducing downtime and allowing for a much better incident response.

Also, the separation of responsibility between packaging software and releasing it to a production environment embodies the security principle of least privilege, reducing the impact of compromise and providing a smaller attack surface.

Increased speed and productivity

Continuous deployment automation with an integrated feedback and control loop speeds up your mean time to deployment. Declarative definitions kept in git allows developers to use familiar workflows, reducing the time it takes to spin up a new development or test environment to deploying new features to a cluster. Your teams can ship more changes per day and this translates into a faster turnaround for new features and functionality to the customer.

Reduced Mean Time to Recovery

The amount of time it takes to recover from a cluster meltdown is also decreased with GitOps best practices. With git's built-in capability to revert/rollback and fork, you gain stable and reproducible rollbacks. Since your entire system is described in git, you have a single source of truth from which to recover after a cluster failure, reducing your meantime to recovery (MTTR) from hours to minutes.

Improved stability and reliability

Because GitOps provides a single operating model for making infrastructure, and apps, you have consistent end-to-end workflows across your entire organization. Not only are your continuous integration and continuous deployment pipelines all driven by pull request, but your operations tasks are also fully reproducible through git.

Easier compliance and auditing

With git's capability to manage your Kubernetes cluster, you automatically gain a convenient audit trail of who did what and when to all cluster changes outside of Kubernetes that can be used to meet SOC 2 compliance and ensure stability.

Since changes are tracked and logged in a secure manner, compliance and auditing are made trivial. The use of software tools like **kubediff**, terradiff and ansiblediff also allow you to compare a trusted definition of the state of the cluster with the actual running cluster, ensuring that the tracked and auditable changes match reality.

WHAT YOU NEED FOR GITOPS

For those who want to implement GitOps workflows to their CICD pipelines, the following need to be in place.

#1. An entire system declaratively described.

Kubernetes is one of many modern cloud native tools out there that are declarative and that can be treated as code. Declarative means that configuration is guaranteed by a set of facts instead of by a set of instructions. With your application's declarations versioned in git, you have a single source of truth. Your apps can then be easily deployed and rolled back to and from Kubernetes. And even more importantly and critical to the GitOps story is that when disaster strikes, your cluster can also be dependably and quickly reproduced.

#2. The desired system state canonically versioned in git.

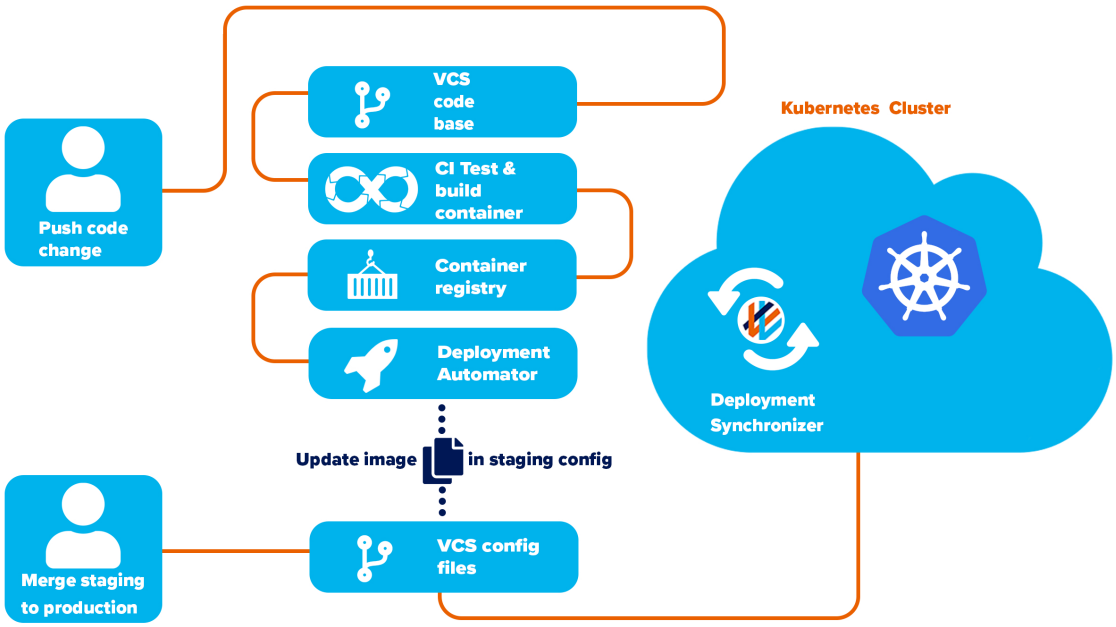
With the declaration of your system stored in a version control system, and serving as your canonical source of truth, you have a single place from which everything is derived and driven. This trivializes rollbacks and you can use a `git revert` to go back to a previous state. With git's excellent security guarantees, an SSH key signs commits to enforce strong security guarantees about the authorship and the code's provenance.

#3. The ability to automatically apply approved changes.

Once you have the declared state kept in git, the next step is to have the ability to automatically apply any state changes to your system. What's significant about this is that you don't need specific cluster credentials to make a change to your system. With GitOps, there is a segregated environment that the state definition lives outside of. This allows your team to separate what they actually do from how they are going to do it.

#4. Software agents to ensure correctness.

With the state of your entire system kept under version control, you can now employ software agents to inform you whenever reality doesn't match your expectations. The use of diff and sync tools also ensures that your entire system is really self-healing. And by self-healing, we don't mean when nodes or pods fail—those are handled by Kubernetes—but in a broader sense, like in the case of human error. In this case, software agents act as the feedback and control loop for your operations.



HOW KUBERNETES HANDLES CONFIGURATION UPDATES

The way Kubernetes handles deployments, lends itself very well to GitOps workflows. For example, when a group of configuration updates are made by a human operator, the Kubernetes orchestrator will keep applying those changes until the cluster's state is converged to the updated configuration made by the human. The same is true for any type of Kubernetes resource.

Kubernetes deployments have the following properties that make it perfect for GitOps style deployment workflows:

- **Automation:** Kubernetes provides a built-in mechanism for automating the deployments. A Kubernetes cluster applies a set of changes in a correct order and in a timely manner.
- **Convergence:** Kubernetes will keep trying to make the update until it eventually succeeds.
- **Idempotence:** Multiple and simultaneous convergence instances will all have the same outcome.
- **Determinism:** Assuming that the cluster has adequate resources available, the updated cluster state will depend only on the desired state.

Kubernetes deployments can also be extended and automated using **Kubernetes Custom Resource Definitions (CRDs) with the operator pattern**. These agents can then be used to automatically detect and apply configuration changes from outside of the system when you need them, essentially creating feedback and control loops.

DEVELOPER TOOLING THAT DRIVES OPS AND ENGINEERING

Introducing GitOps into your organization means:

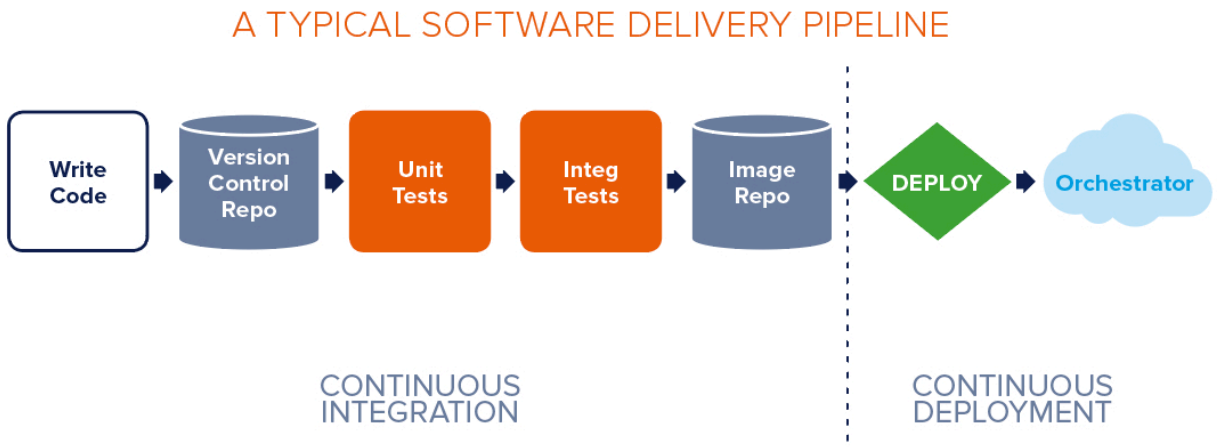
- Any developer that uses git can start deploying new features to Kubernetes
- The same workflows are maintained across development and operations
- All changes can be triggered, stored, validated and audited in git
- Ops changes can be made by pull request including rollbacks
- Ops changes can be observed and monitored

Having everything in one place means that your operations team can use the same workflow to make infrastructure changes by creating issues, and reviewing pull requests. GitOps allows you to roll back any kind of change made in your cluster. In addition to this, built-in observability enables your teams to have more autonomy and to make more changes and experiment without worrying about breaking the build.

WHAT DOES THE TYPICAL CICD PIPELINE LOOK LIKE?

Most organizations that set off on their journey to continuous delivery, start by automating a CICD pipeline. With this newly minted pipeline in place, the keen development team will furiously start writing and pushing code to git.

In this simplified example, let's say there is a single microservice repository that bundles the microservice's code with its deployment YAML manifest files. YAML files if you recall are what define or declare how the microservice runs in the cluster. When the developer pushes the code to git, a continuous integration tool kicks off unit tests that eventually build the Docker container image that gets pushed to the container registry.



With this typical CICD push-based pipeline, Docker container images are then deployed to the actual cluster using some sort of bespoke bash scripts or through some other method that talks directly to the Kubernetes' API.

SECURITY AND THE TYPICAL CI/CD PIPELINE

The typical CI/CD pipeline has security problems.

With this approach, your CI tooling pushes and deploys images to the cluster. For the CI system to apply the changes to a cluster, you have to share your API credentials with the CI tooling and that means your CI tool becomes a high value target. If someone breaks into your CI tool, they will have total control over your production cluster, even if your production cluster is highly secure.

What happens when your cluster goes down?

Also what happens when you need to recreate your cluster in the case of a total meltdown? How do you restore the previous state of your application? You would have to run all of your CI jobs to rebuild everything and then re-apply all of the workloads to the new cluster. The typical CI pipeline doesn't have its state easily recorded like it is when you're using GitOps.

Let's see how you can improve the typical CI/CD pipeline with GitOps.

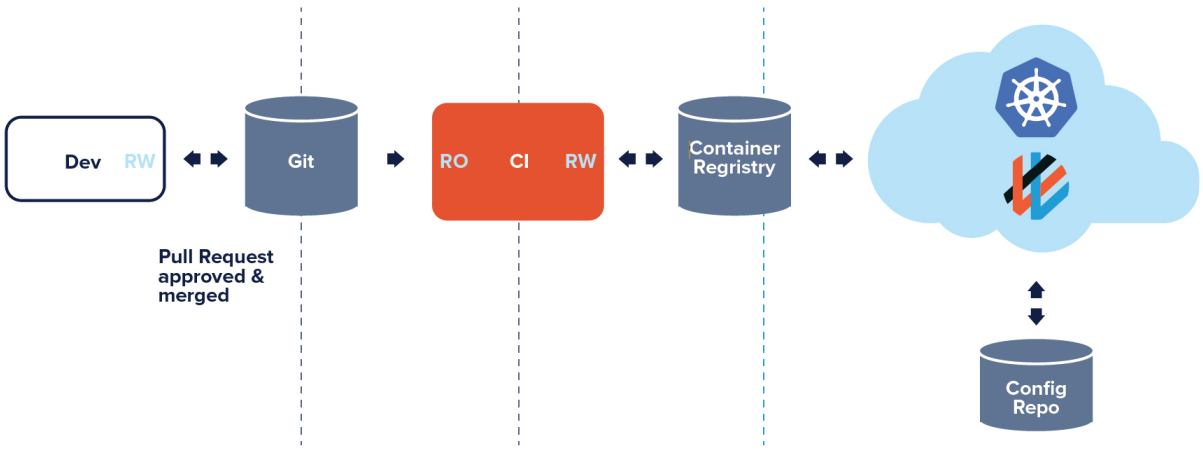
THE GITOPS DEPLOYMENT PIPELINE

GitOps implements a Kubernetes controller that listens for and synchronizes deployments to your Kubernetes cluster. The controller uses the operator pattern. This is significant on two levels:

1. It is more secure.
2. It automates complex error prone tasks like having to manually update YAML manifests.

With the operator pattern, an agent acts on behalf of the cluster. It listens to events relating to custom resource changes, and then applies those changes based on a deployment policy. The agent is responsible for synchronizing what's in git with what's running in the cluster, providing a simple way for your team to achieve continuous deployment.

GitOps Deployment Workflow



GITOPS IS A MORE SECURE WAY TO DEPLOY CHANGES

The image is pulled using Read Only access to the repository. The CI tool is not granted cluster privileges, and therefore is not introducing significant security risks to your pipeline.

GitOps Separation of Privileges

GitOps separates CI from CD and this is one reason it is a more secure method of deploying applications to Kubernetes. The table below shows how GitOps separates read/write privileges between the cluster, your CI and CD tooling and the container repository, providing your team with a secure method of creating updates.

CI TOOLING TEST, BUILD, SCAN, PUBLISH	**CD TOOLING** RECONCILIATION BETWEEN GIT AND THE CLUSTE
Runs outside the production cluster	Runs inside the production cluster
Read access to the Code Repository	Read/Write access to configuration repo
Read/Write access to Container Repository	Read access to image repo
Read/Write access to the Continuous Integration environment	Read/Write access to the production cluster

Table 1: GitOps Separation of Privileges

OBSERVABILITY AS A DEPLOYMENT CATALYST

An essential component to GitOps is feedback and control. But what is meant exactly by this? In order to have control so that developers can go faster, they need observability built-in to their deployment workflows. Built-in observability allows engineers to make informed decisions on real-time data. For example, when a deployment is being rolled out, a final health check can be made against your running cluster before committing to that update or perhaps an update didn't go as planned and can be easily rolled back to a good state.

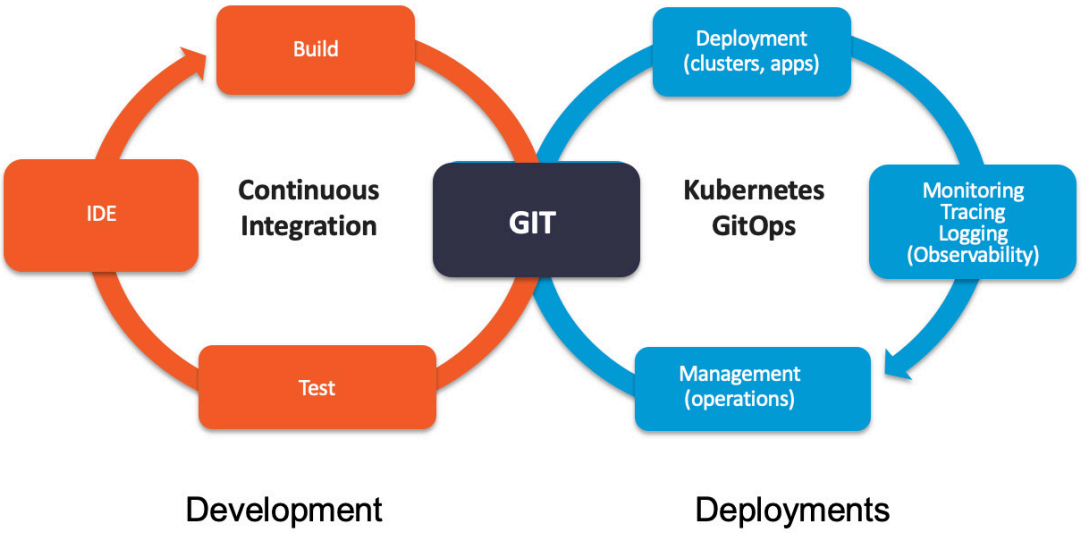
With a feedback control loop you can effectively answer the following questions:

- 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. GitOps takes advantage of both to manage 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:



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

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.

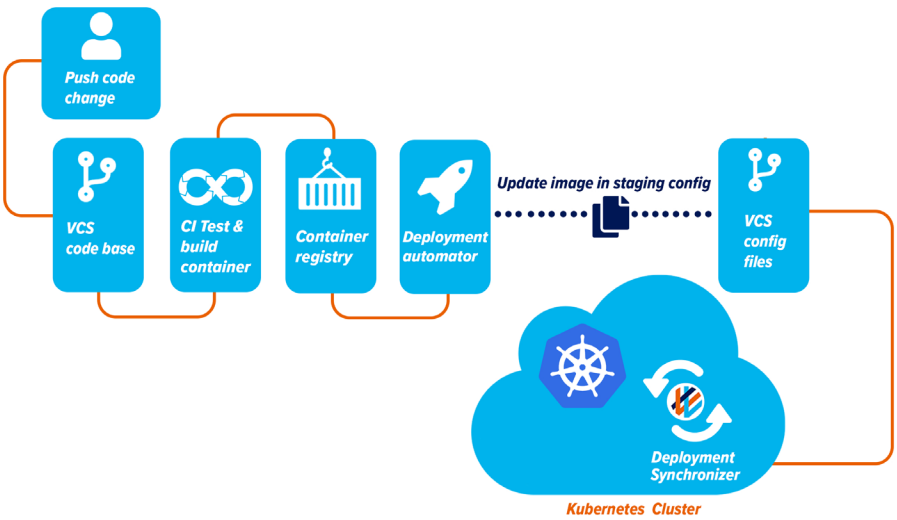
After new features and fixes are merged to git, the deployment pipeline is triggered, and once the image is ready to be released, it can be observed in real-time against the running cluster. At this point, the developer may return to the beginning of the pipeline based on this feedback or deploy and release the image to the production cluster.

A GITOPS WORKFLOW

The GitOps core machinery is in its CICD tooling with the critical piece being continuous deployment (CD) that supports git-cluster synchronization. It is designed specifically for version controlled systems and declarative application stacks. Every developer on your team is familiar with git and can make pull requests. Now they can use git to accelerate and simplify application deployments to Kubernetes as well.

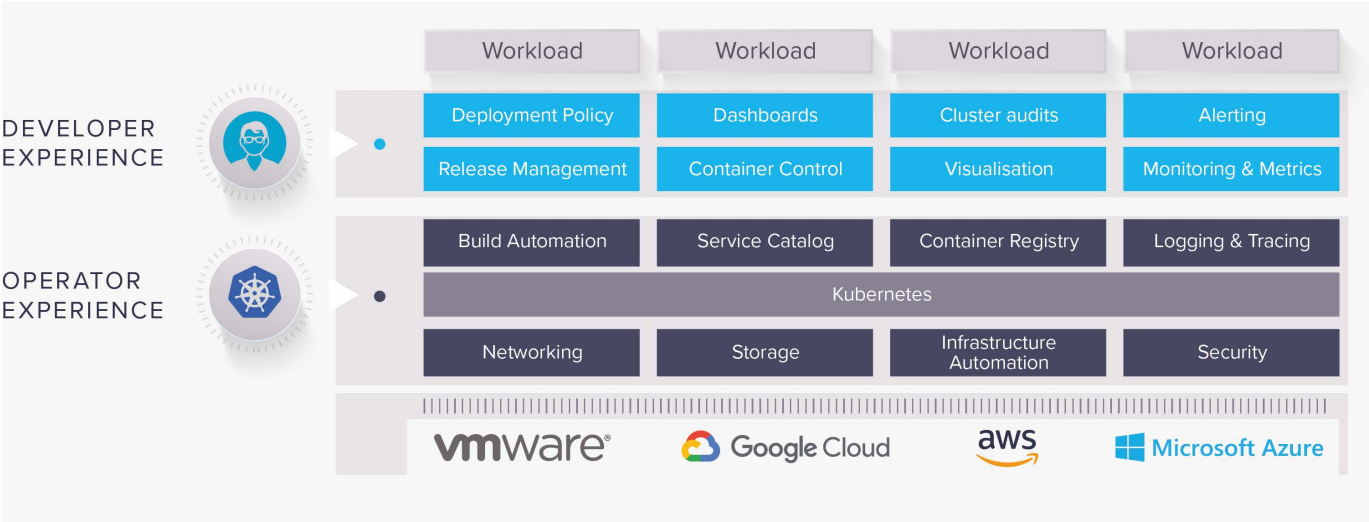
Here is a typical developer workflow for creating or updating a new feature:

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, and re-approved 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 deposits to the new image to a registry.
4. The ‘Deployment Automator’ watches the image registry, notices the image, pulls the new image from the registry and updates its YAML in the config repo.
5. The ‘Deployment Synchronizer’, detects that the cluster is out of date, pulls the changed manifests from the config repo and deploys the new feature to production.



BUILD A CLOUD NATIVE PLATFORM WITH GITOPS AND WEAVE GITOPS ENTERPRISE

Weave GitOps is a continuous delivery product to run applications in any Kubernetes. It enables simple, declarative workflows to take complete control of Kubernetes a Pull Request at a time. Cloud native teams are lightning fast by pushing everything to their environment . Environment immutability, application deployment, progressive delivery and cluster health lifecycle, all from within Git. Take control of the cloud, the edge and your machines to deliver incremental value continuously.



GITOPS CONFIGURATION MANAGEMENT AUTOMATION

With GitOps at the centre of your operational model, teams can install and manage production ready Kubernetes on-premise, in a public cloud such as AWS and even onto pre-created OpenStack nodes. GitOps can be used to initiate a cluster patch or a minor version upgrade or add and remove cluster nodes all without having to rebuild your entire cluster from the ground up.

With your entire cluster configuration stored in Git and managed through GitOps, you can reproduce an entire cluster in a repeatable and predictable way. This brings advantages when you are building test environments and pipelines, and producing clusters for different teams with the same base configuration, or improving your disaster recovery capability.

Key features

- Reliable, stable and upstream version of Kubernetes
- Unified cluster-API based installer for Kubernetes
- Critical cluster addons for all major cloud platforms and on-premise versions
- Node and cluster lifecycle management based on GitOps workflows
- Inclusive cluster observability components:
    - Flexible monitoring and logging configuration
    - Prometheus, the best monitoring choice for Kubernetes

Choice of add-ons:

- Network authentication protocol (Kerberos)
- Safe automatic node reboots (Kured)
- Continuous and auditable deployment (Weave Flux)
- Diff alerting on running and desired state (Kubediff)
- Integration ready for third party metrics, logging and other tooling choices

Find out more about the **Weave GitOps**

HAVE QUESTIONS ON WHAT YOU NEED TO CREATE A CLOUD NATIVE PLATFORM?

The Weaveworks team can help you navigate the vast landscape of cloud native technologies – OSS and paid. Together we can create a cloud native reference architecture that fits your business needs. You can benefit from a Weaveworks' validated design or you can design, review and select technology options with our help.

Contact us for a demo of Weave GitOps Enterprise.