Guide To GitOps

Pioneered in 2017, GitOps is a way to do Kubernetes cluster management and application delivery. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what's running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case. With Git at the center of your delivery pipelines, developers use familiar tools to make pull requests to accelerate and simplify both application deployments and operations tasks to Kubernetes.

An operating model for building cloud native applications

GitOps can be summarized as these two things:

1. An operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git 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.

Principles of GitOps

To start managing your cluster with GitOps workflows, the following must be in place:

#1. The entire system described declaratively.

With Gitops, Kubernetes is just one example of many modern cloud native tools 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, when disaster strikes, your cluster’s infrastructure can also be dependably and quickly reproduced.

#2. The canonical desired system state versioned in Git.

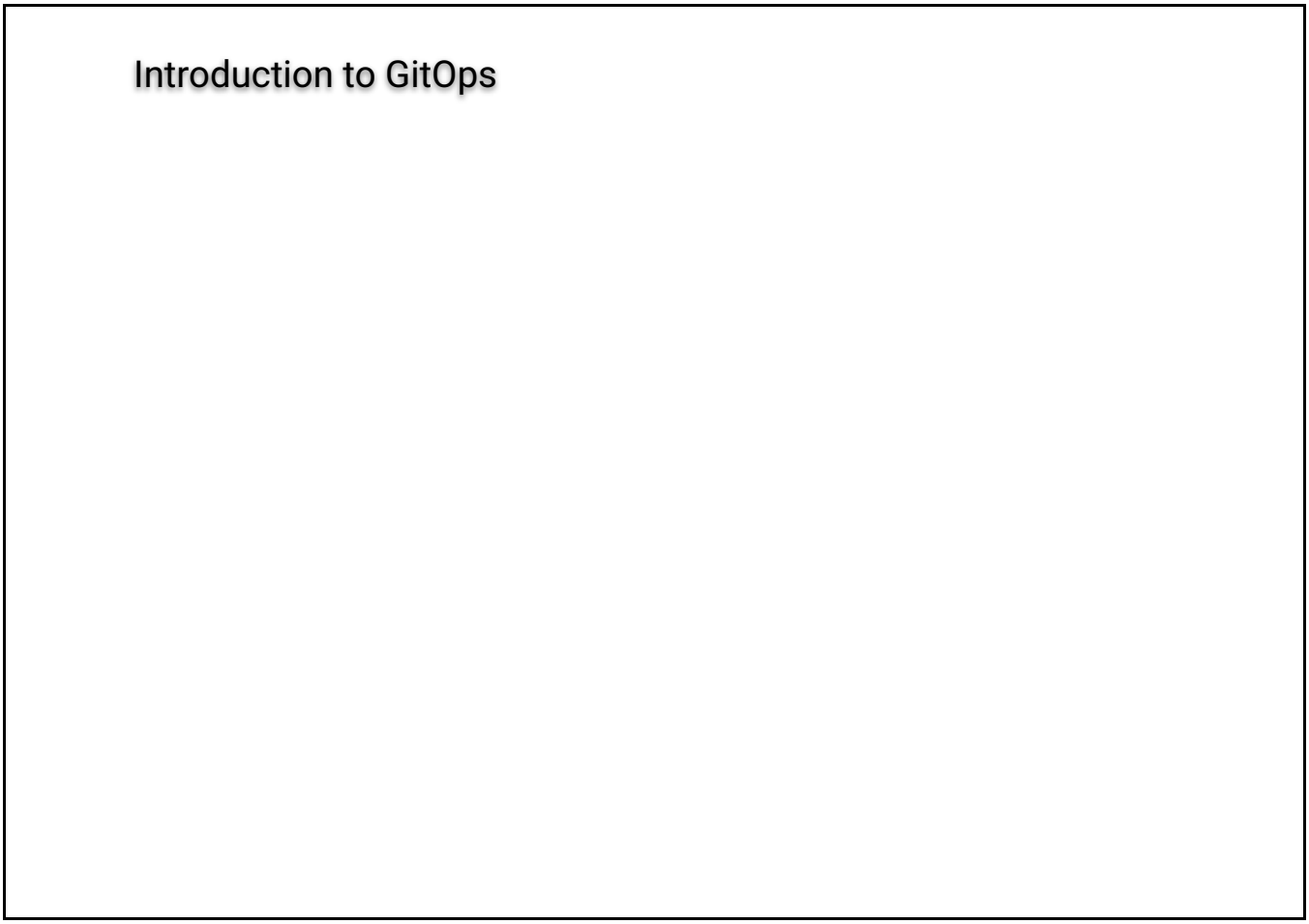
With the declaration of your system stored in a version control system, and serving as your canonical source of truth, you have a single place from which everything is derived and driven. This trivializes rollbacks; where you can use a `Git revert` to go back to your previous application state. With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

#3. Approved changes that can be automatically applied to the system.

Once you have the declared state kept in Git, the next step is to allow any changes to that state to be automatically applied to your system. What's significant about this is that you don't need cluster credentials to make a change to your system. With GitOps, there is a segregated environment of which the state definition lives outside. This allows you to separate what you do and how you're going to do it.

#4. Software agents to ensure correctness and alert on divergence.

Once the state of your system is declared and kept under version control, software agents can inform you whenever reality doesn’t match your expectations. The use of agents also ensures that your entire system is self-healing. And by self-healing, we don’t just 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.



Links:

- GitOps: Operations by Pull Request
- The GitOps Pipeline - Part 2
- GitOps: 'Git Push' all the things (The New Stack)
- The Best CI/CD Tool for Kubernetes Doesn't Exist
- The full history of GitOps 2017-2020

Automated delivery pipelines roll out changes to your infrastructure when changes are made to Git. But the idea of GitOps goes further than that – GitOps uses tools to compare the actual production state of your whole application with what's under source control and then it tells you when your cluster doesn't match the real world.

By applying GitOps best practices, there is a 'source of truth' for both your infrastructure and application code, allowing development teams to increase velocity and improve system reliability.

The benefits of applying GitOps best practices are far reaching and provide:

1. Increased Productivity
   Continuous deployment automation with an integrated feedback control loop speeds up Mean Time to Deployment. Your team can ship 30-100 times more changes per day, increasing overall development output 2-3 times.
2. Enhanced Developer Experience
   Push code and not containers. Developers can use familiar tools like Git to manage updates and features to Kubernetes more rapidly without having to know the internal of Kubernetes. Newly on-boarded developers can get quickly up to speed and be productive within days instead of months.
3. Improved Stability
   When you use Git workflows to manage your cluster, you

automatically gain a convenient audit log of all cluster changes outside of Kubernetes. An audit trail of who did what, and when to your cluster can be used to meet SOC 2 compliance and ensure stability.

4. Higher Reliability
   With Git's capability to revert/rollback and fork, you gain stable and reproducible rollbacks. Because your entire system is described in Git, you also have a single source of truth from which to recover after a meltdown, reducing your meantime to recovery (MTTR) from hours to minutes.
5. Consistency and Standardization
   Because GitOps provides one model for making infrastructure, apps and Kubernetes add-on changes, 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.
6. 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 is key to a secure definition of the desired state of the cluster.