Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2019, 6(1):247-251**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Continuous Integration/Continuous Deployment (CI/CD) for Cloud Native Infrastructure and Applications: Embracing GitOps

Savitha Raghunathan

saveetha13@gmail.com

Abstract This whitepaper delves into integrating Continuous Integration and Continuous Deployment (CI/CD) methodologies with GitOps principles for cloud-native applications. As organizations accelerate their adoption of cloud-native technologies, the need for more efficient, reliable, and scalable software delivery methods becomes paramount. GitOps offers a paradigm shift in managing infrastructure and applications using Git as the single source of truth. This paper explores the fundamentals of GitOps, its integration into CI/CD pipelines, the architectural considerations, advantages, challenges, and future directions of adopting GitOps practices.

Keywords GitOps, Continuous Integration (CI), Continuous deployment (CD), Cloud Native, Kubernetes, Application deployment

1. Introduction

The evolution of cloud native technologies has warranted a transformation in how software is developed, deployed, and managed at scale. The CI/CD pipeline, a cornerstone of modern DevOps practices, automates the steps from code integration to deployment. GitOps extends these practices by leveraging Git as the central mechanism for version control, collaboration, declarative infrastructure, and application management [5]. This method boosts both the efficiency and reliability of delivering software, conforming to the principles of Infrastructure as Code (IaC) and immutable infrastructure.

2. What is GitOps?

GitOps is a term that emerged in the cloud native ecosystem as a novel approach to software deployment and operations, drawing heavily on the principles and practices of DevOps but with a specific emphasis on using Git as the core tool for managing infrastructure and applications. This methodology adopts DevOps principles such as version control, collaboration, compliance, and continuous integration/deployment from application development to automate infrastructure. By leveraging Git as the central repository for declarative infrastructure and applications, GitOps empowers teams to handle their infrastructure using the identical tools and methodologies employed in code development [6]. It includes everything from provisioning new infrastructure, deploying applications, and updating configurations.

2.1 Definition and key concepts

GitOps is defined as using Git as a single source of truth for declarative infrastructure and applications. With Git at the center of the CI/CD pipeline, every change to the system is codified and versioncontrolled, allowing for automated management, monitoring, and rollback of systems based on the Git repository's state [6].

GitOps is based on several key ideas, including declarative system configuration [5][8], where every aspect needed to operate a system is precisely defined and stored in Git, allowing for exact system replication from the repository. It leverages version control for a detailed and immutable record of all modifications, supporting audits, rollbacks, and traceability. Automation plays a critical role, with CI/CD pipelines ensuring that updates

are seamlessly applied to match the intended state in Git, extending to system recovery and self-healing. Additionally, GitOps utilizes software agents within environments like Kubernetes clusters to continuously verify and align the system state with the specified configurations in Git [9].

2.2 GitOps and Kubernetes

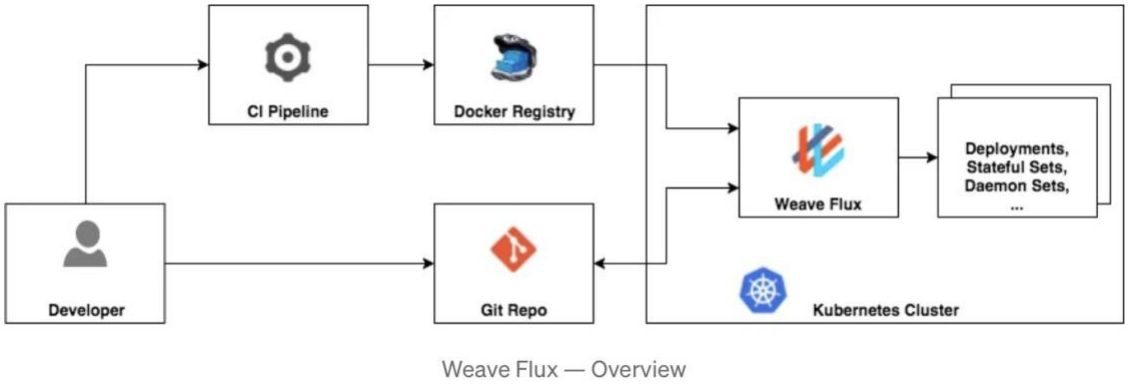

Figure 1: Flux integration with Kubernetes [1]

GitOps practices have been particularly effective in Kubernetes environments. Kubernetes' declarative nature and configuration through YAML files align perfectly with GitOps principles. Tools developed to facilitate GitOps workflows, such as Weaveworks' Flux [2] as shown in Fig 1, provide mechanisms to automatically apply updates to Kubernetes clusters based on changes in a Git repository. This allows for seamless synchronization between the source code repository and the running state of applications in Kubernetes, facilitating features like automated deployments, rollbacks, and configuration changes.

3. Benefits of GitOps

The adoption of GitOps offers several benefits, including:

- **Improved Developer Productivity:** Developers can manage infrastructure and deploy applications using familiar Git operations, reducing the learning curve and speeding up development cycles.
- **Enhanced Visibility and Control:** The entire state of the system is visible in Git, and changes can only be made through Git commits, providing a clear audit trail and simplifying compliance and governance.
- **Increased Operational Efficiency:** Automating deployment and management tasks reduces manual errors and frees up operational teams to focus on value-adding activities.
- **Enhanced Security:** Using Git as the control mechanism allows for the integration of security practices into the deployment process [9].

4. How to Architect a GitOps Pipeline

Architecting a GitOps pipeline requires a strategic integration of tools and practices designed to automate and manage the deployment and operation of cloud-native applications efficiently. The key components of a GitOps pipeline form an ecosystem that supports the principles of immutability, automation, and declarative configurations. Expanding on each of these components:

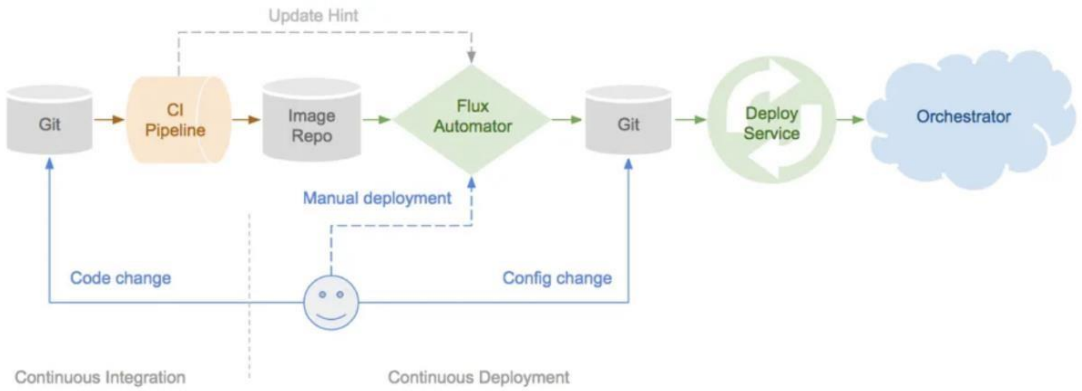

Figure 2: Example Gitops pipeline using Flux [4]

4.1 Source Code Repository

The repository is the heart of the GitOps workflow, storing not just application code but also the entire desired state of the infrastructure and configurations in a declarative manner [4]. This includes Kubernetes manifests, Helm charts, and any other infrastructure as code (IaC) templates.

- **Best Practices:** To manage deployments across various stages, use separate branches for different environments (e.g., development, staging, production). Implementing pull request reviews and integrating automated testing within the Git workflow enhances reliability and collaboration.

4.2 Continuous Integration (CI) Tools

Continuous Integration (CI) tools are responsible for automatically building, testing, and validating the code in response to changes committed to the repository [4]. This tooling and process ensures that only high-quality code that passes all tests is considered for deployment.

- **Integration with GitOps:** CI tools can be configured to automatically trigger builds based on Git events, such as a merge into the main branch. Successful builds can trigger the CI tool to update the Docker image in the container registry and push the new image version to the source code repository as part of the deployment specification.

4.3 Continuous Deployment (CD) Tools with GitOps Capabilities

Continuous Deployment (CD) tools designed explicitly for GitOps, like Argo CD or Flux, monitor the source code repository for changes to the deployment specifications. These tools automatically apply changes to the Kubernetes environment, ensuring that the actual state matches the desired state declared in Git [4].

- **Key Features:** These tools provide automated rollbacks, detailed auditing, and synchronization checks. They can also manage complex deployments across multiple environments and support canary or blue-green deployment strategies to minimize risks.

4.4 Container Orchestration Platforms

Kubernetes serves as the operational backbone, orchestrating containerized applications' deployment, scaling, and management. It interprets and enforces the desired state defined in the Git repository.

- **Integration with GitOps:** Kubernetes' declarative API and its ability to self-heal align perfectly with GitOps principles. Tools like Argo CD or Flux extend Kubernetes' capabilities by adding a layer of automation and monitoring to ensure the desired state of the application/Infrastructure in Git is accurately reflected in the Kubernetes environment.

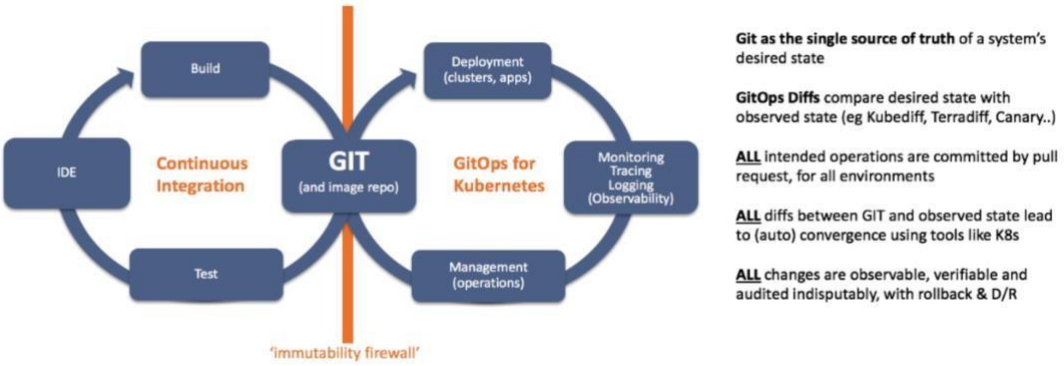

Figure 3: Operating model for Kubernetes [3]

4.5 Monitoring and Alerting

Monitoring and alerting are crucial for maintaining application reliability and performance. Tools like Prometheus for monitoring and Grafana for visualization provide insights into application and infrastructure health, performance metrics, and operational status

- **Integration with GitOps:** These tools can be configured to automatically alert teams about discrepancies between the declared state in Git and the actual state of the environment [8]. They support the GitOps principle of correcting divergence, enabling teams to respond quickly to issues.

5. Challenges

- **Learning Curve:** Teams may face challenges in adopting new tools and shifting to a GitOps mindset [7].
- **Complexity in Large Environments:** Managing numerous Git repositories and ensuring consistency across large, distributed systems can be complex.
- **Security Concerns:** The central role of Git necessitates implementing strict security practices to prevent unauthorized access and changes.

6. Future Direction

The future of GitOps is likely to evolve towards greater automation, with AI and machine learning playing roles in predictive modeling and anomaly detection. Integration with serverless architectures and adopting GitOps across multi-cloud environments will further expand its applicability and efficiency.

7. Conclusion

Embracing GitOps within CI/CD pipelines for cloud native applications represents a significant leap toward a more efficient, reliable, and scalable software delivery system. By leveraging Git as the single source of truth and automating the deployment and management of infrastructure and applications, organizations can achieve unprecedented levels of operational excellence. The future of GitOps is bright, with ongoing innovations expected to unlock new capabilities in cloud native application delivery.

References

[1]. M. Nguyen, "Continuous Deployment to Kubernetes with GitOps (Weave Flux)," *Medium*, Jul. 25, 2018. https://medium.com/@m.k.joerg/gitops-weave-flux-in-detail-77ce36945646

[2]. "Weave Flux," *GitHub*, Aug. 01, 2018. https://github.com/fluxcd/flux *priority_high* Webpage author

[3]. A. Richardson and Weaveworks, "What Is GitOps Really?," *Medium*, Oct. 02, 2018. https://medium.com/weaveworks/what-is-gitops-really-e77329f23416

[4]. A. Richardson and Weaveworks, "The GitOps Pipeline — Part 2," *Weaveworks Blog*, Sep. 28, 2018. https://medium.com/weaveworks/the-gitops-pipeline-part-2-c53fbc79960d

[5]. A. Williams, "GitOps for Kubernetes: A DevOps Iteration Focused on Declarative Infrastructure," *The New Stack*, Feb. 27, 2018. https://thenewstack.io/gitops-kubernetesdevops-iteration-focused-declarative-infrastructure/

[6]. D. Bryant, "'GitOps': Weaveworks Explain Their Model for Using Developer Tooling to Implement CI/CD," *InfoQ*, Sep. 06, 2018. https://www.infoq.com/news/2018/09/gitopsweaveworks/

[7]. T. A. Limoncelli, "GitOps: A Path to More Self-service IT," *ACM Queue*, vol. 16, no. 3, pp. 13– 26, Jun. 2018, doi: https://doi.org/10.1145/3236386.3237207.

[8]. A. Richardson and Weaveworks, "GitOps Part 3 — Observability," *Medium*, Sep. 28, 2018. https://medium.com/weaveworks/gitops-part-3-observability-b6b2825e29f

[9]. Weaveworks, "Hands-On Gitops," *Fractal Lambda*, Dec. 2018. https://fractallambda.com/assets/2018-12-11-kubecon-gitops-tutorial.pdf