Git Operations: Argo CD vs Flux

Author: Qadeer Hussain   Course: Software Development Yr 4   Module: Cloud Data Centres Report
18/05/2025

***Abstract*—This report covers Git Operations, a modern approach to continuous deployment using Git as the source of truth. It compares two popular Git Ops tools Argo CD and Flux and demonstrates their use in Kubernetes environments. A hands on example with Argo CD is included to illustrate practical implementation.**

***Index Terms*—Git Ops, Argo CD, Flux, Continuous Deployment (CD), Kubernetes, Git Repositories**

I. Introduction

This report focuses on Git Operations, explaining its definition and relevance to cloud native computing. It also examines two widely used Git operation tools: Argo CD and Flux.

II. What is Git Operations

Git Operations is a way of implementing Continuous Deployment for Cloud Applications. The main idea of Git Ops is to have a Git repository that has declarative descriptions of the infrastructure currently in the production environment and an automated process to make the production environment match the described state in the repository. [1][2][3]

Git Ops has many benefits, such as efficiency, security, reduced costs, and faster deployments. Git Ops helps organisations manage containers and micro services more effectively by using Kubernetes, while maintaining consistency across all their infrastructure. [2][8]

III. Git Ops Workflow

The Git Ops workflow is built around four components each playing vital role in deployment and management of applications.

- **Git Repository:** Acts as the central source of truth for both applications code and configuration. Stores all critical information in the repository. [2]
- **Continuous Delivery (CD) Pipeline:** CD pipeline automates process of building, testing and deploying. Facilitating a smooth transition from development to production environments. [2]
- **Application Deployment Tool:** In charge of deploying the application to the desired environment. Orchestrates and manages the applications resources, making sure the application is deployed correctly and efficiently according to the configuration of the git repository. [2]
- **Monitoring System:** This is needed for maintaining the applications health, the system, keeps an eye on the applications performance. Gathers data and provides the development team with insight and feedback allowing them to make decisions based on any issues that arise. [2]

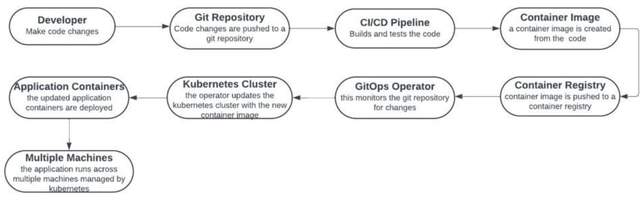

Developer
Make code changes
Git Repository
Code changes are pushed to a git repository
CI/CD Pipeline
Builds and tests the code
Container Image
a container image is created from the code
Container Registry
container image is pushed to a container registry
GitOps Operator
this monitors the git repository for changes
Kubernetes Cluster
the operator updates the kubernetes cluster with the new container image
Application Containers
the updated application containers are deployed
Multiple Machines
the application runs across multiple machines managed by kubernetes

Fig. 1. Git Ops Workflow

IV. Overview of Git Ops Tools

There are several tools used to support Git Ops workflows by automating deployment and synchronisation between Git repositories and Kubernetes clusters. Among them most widely adopted are Argo CD and Flux, both which allow for continuous Deployment using Git as the source truth. [2]

V. Argo CD

Fig. 2. Argo CD

A. Overview

Argo CD is a declarative open source, Git Ops continuous delivery tool for Kubernetes. It is a part of the Cloud Native Computing Foundation (CNCF). The aim of Argo CD is to automate deployment of applications into specific target environments inside of Kubernetes, using Git Repository as the source of truth for the desired state of applications. [6][7][8]

Continuously monitors Git repositories and compares the states of the application defined in the repository with the actual state in Kubernetes environment. Any differences noted trigger an automatic sync to ensure the actual state matches the desired state. By doing this, Argo CD provides a level of automation and control that dramatically simplifies management of deployments in Kubernetes. [6][7][8]

Argo CD has a comprehensive dashboard which provides visual representation of the applications status allowing for easy management. Also provides access control, audit logs and supports a range of configurations tools such as Kustomize, Helm. [6][7][8]

B. Argo Architecture

The core components of Argo CD include the API Server, Repository Server, and Application Controller:

- **API Server:** Is a gRPC/REST server which exposes API consumed by Web UI, CLI and CI/CD systems. It handles application management, status reporting, invoking of application operations, repository and cluster credential management, authentication and auth delegation, RBAC enforcement and listener/forwarder for Git web hook events. [4]
- **Repository Server:** Maintains a local cache of Git repositories and is responsible for generating Kubernetes manifests from the application's source, using inputs such as the repository URL, revision, application path and template specific settings. [4]
- **Application Controller:** Continuously monitors the state of deployed applications and compares it with the desired state from the repository. It detects OutOfSync application state and takes action. Responsible for invoking user-defined hooks for lifecycle events such as Pre Sync, Sync and Post Sync. [4]

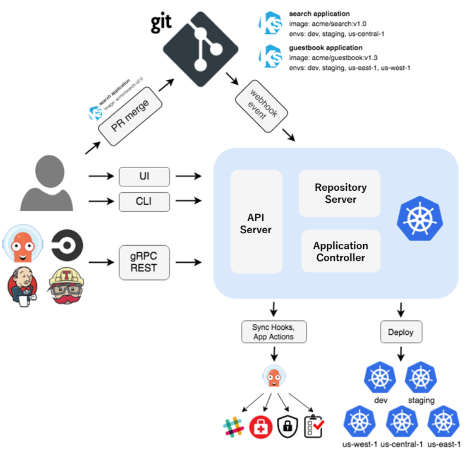


Fig. 3. Argo CD Architecture

VI. Flux

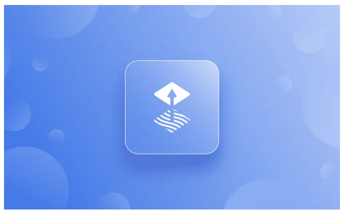


Fig. 4. Flux

A. Overview

Flux is another open source tool for Git Ops which is also designed to implement Git Ops workflows in Kubernetes. Just like Argo CD Flux is also apart of the Cloud Native Computing Foundation (CNCF). [6][7][8]

Flux operates in a very similar manner to Argo CD it also continuously monitors Git repositories and compares the states of the application defined in the repository with the actual state in Kubernetes environment. Any differences noted trigger an automatic sync to ensure the actual state matches the desired state. Flux also has image updates, automatically tracking and deploying containers images, this makes it useful for continuous delivery. [6][7][8]

Flux like Argo also supports similar configuration tools such as Kustomize, Helm. Key features include automated roll outs, rollbacks, and multi-environment support. [6][7][8]

B. Flux Architecture

Flux is designed to run within Kubernetes. It relies heavily on native resources and API's. This approach allows for more granular control and flexibility. However, this requires additional setup for a complete overview. This external tools like Weave Git Ops for visualisation. [7][8]

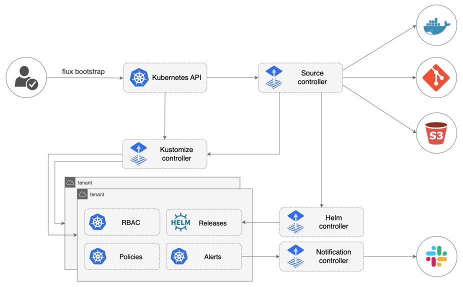


Fig. 5. Flux Architecture

VII. Comparison

A. Architecture

Both Argo CD and Flux are similar in there uses and function. However, Flux, is heavily dependent on Kubernetes resources and API's. This allows for more control and flexibility but requires additional setup for complete overview. Flux is an expandable toolkit, whereas, Argo CD is a complete application. Therefore, Argo is more opinionated then Flux. [7][8]

B. Development Process

Flux and Argo CD offer same deployment processes. Register the application using CLI or Kubernetes CRD's, stating the source to connect to such as the git repository. The tools get the state from the repository and position the application in the cluster. When an update is ready to be applied, push it to the repository. The tool will sync changes into the cluster, manually or automatically. Flux and Argo CD support Kustomize templates Helm charts. However, Argo does not use Helm internally, as Helm has some difficulties such as its awkwardness with CRD management. [8]

C. Declarative Configuration

Flux and Argo CD both support declarative configuration. Flux favours Git Repository and Kustomisation. While Argo favours high abstraction with applications and applications sets. Declarative Configuration is used by flux to manage installation in the cluster. When Flux is installed it must specify or create a git repository that will store config manifests used by Flux. One can reproduce the same environment with this process. [8]

D. Ease of use

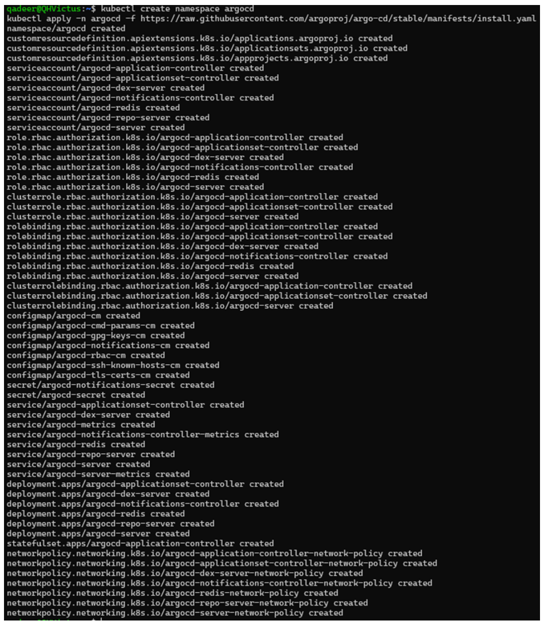
Flux and Argo CD are easy to implement use as they are supported by robust CLI and clear documentation. However, Flux has lower-level config mechanism and a steeper learning curve then Argo CD. Argo is more accessible with in built web interface. [8]

E. Access Management

Flux does not have its own access management system. Kubernetes identify, and RBAC controls must be used to configure who can interact with Flux resources. This standardises all controls in Kubernetes RBAC mechanism, however it makes it difficult to enforce granular policies. Argo allows the user to decouple access control from cluster level Kubernetes users. Argo users can be created by adding 'local' users to config map. Once the user is added, actions can be performed with Argo's RBAC layer. [8]

VIII. Implementation: Argo CD Hands-On Deployment

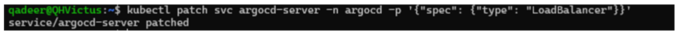
A. Install Argo CD



```
qadeer@QHVictus:~$ kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
namespace/argocd created
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-redis created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-server created
configmap/argocd-cm created
configmap/argocd-cmd-params-cm created
configmap/argocd-gpg-keys-cm created
configmap/argocd-notifications-cm created
configmap/argocd-rbac-cm created
configmap/argocd-ssh-known-hosts-cm created
configmap/argocd-tls-certs-cm created
secret/argocd-notifications-secret created
secret/argocd-secret created
service/argocd-applicationset-controller created
service/argocd-dex-server created
service/argocd-metrics created
service/argocd-notifications-controller-metrics created
service/argocd-redis created
service/argocd-repo-server created
service/argocd-server created
service/argocd-server-metrics created
deployment.apps/argocd-applicationset-controller created
deployment.apps/argocd-dex-server created
deployment.apps/argocd-notifications-controller created
deployment.apps/argocd-redis created
deployment.apps/argocd-repo-server created
deployment.apps/argocd-server created
statefulset.apps/argocd-application-controller created
networkpolicy.networking.k8s.io/argocd-application-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-applicationset-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-dex-server-network-policy created
networkpolicy.networking.k8s.io/argocd-notifications-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-redis-network-policy created
networkpolicy.networking.k8s.io/argocd-repo-server-network-policy created
networkpolicy.networking.k8s.io/argocd-server-network-policy created
```

Fig. 6. Install Argo

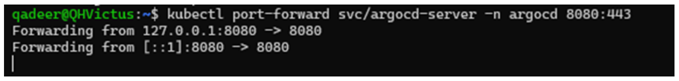
B. Service Type Load Balancer



```
qadeer@QHVictus:~$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
```

Fig. 7. Service Type Load Balancer

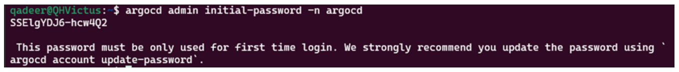
C. Port Forwarding



```
qadeer@QHVictus:~$ kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Fig. 8. Port Forwarding

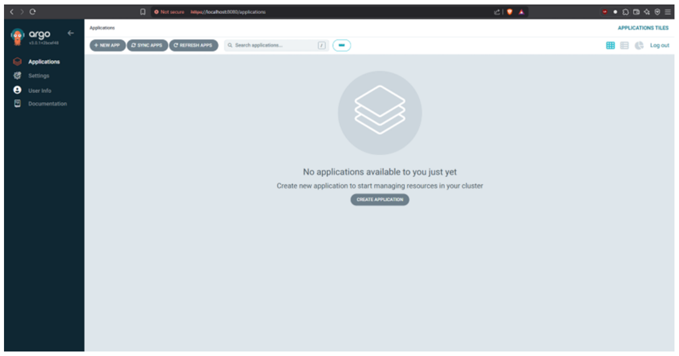
D. Login Using The CLI



```
qadeer@QHVictus:~$ argocd admin initial-password -n argocd
SSElgYDJ6-hcw4Q2

 This password must be only used for first time login. We strongly recommend you update the password using `
argocd account update-password`.
```

Fig. 9. Get Password



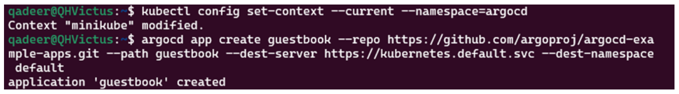
Not secure https://localhost:8080/applications

No applications available to you just yet

Create new application to start managing resources in your cluster

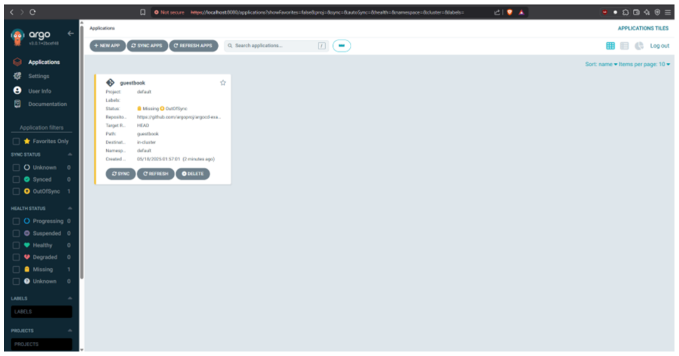
Fig. 10. Logged into Argo

E. Creating Apps via CLI



```
qadeer@QHVictus:~$ kubectl config set-context --current --namespace=argocd
Context "minikube" modified.
qadeer@QHVictus:~$ argocd app create guestbook --repo https://github.com/argoproj/argocd-exa
mple-apps.git --path guestbook --dest-server https://kubernetes.default.svc --dest-namespace
 default
application 'guestbook' created
```

Fig. 11. Creating Apps via CLI 1



guestbook

Project: default
Labels:
Status: Missing OutOfSync
Reposito... https://github.com/argoproj/argocd-exa...
Target R... HEAD
Path: guestbook
Destinat... in-cluster
Namesp... default
Created ... 05/18/2025 01:57:01 (2 minutes ago)

Fig. 12. Creating Apps via CLI 2

F. Creating Apps via UI

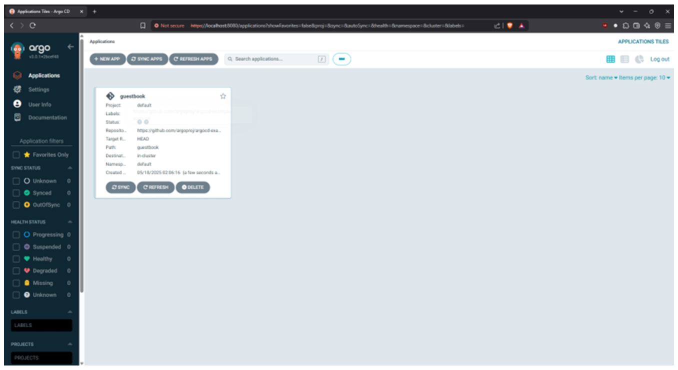
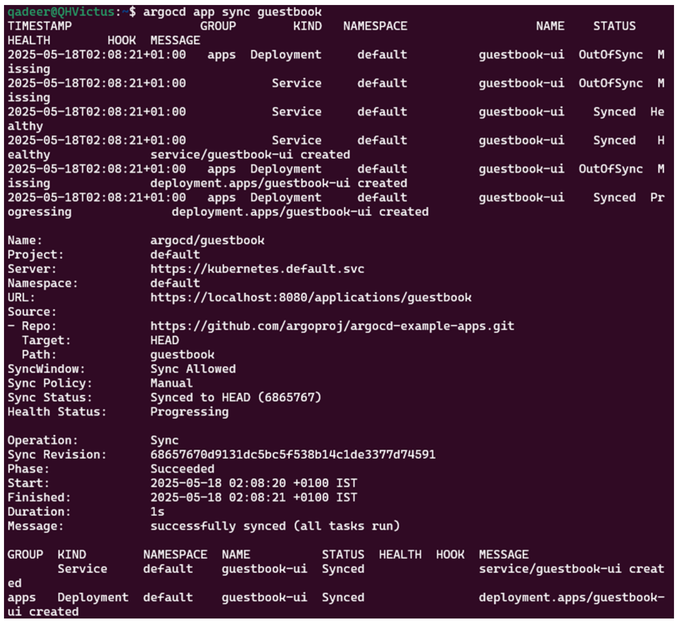


Fig. 13. Creating Apps Via UI

G. Sync (Deploy) The Application



```
qadeer@QHVictus:~$ argocd app sync guestbook
TIMESTAMP                  GROUP        KIND   NAMESPACE                  NAME    STATUS
HEALTH        HOOK  MESSAGE
2025-05-18T02:08:21+01:00   apps  Deployment     default         guestbook-ui  OutOfSync  M
issing
2025-05-18T02:08:21+01:00             Service     default         guestbook-ui  OutOfSync  M
issing
2025-05-18T02:08:21+01:00             Service     default         guestbook-ui    Synced  He
althy
2025-05-18T02:08:21+01:00             Service     default         guestbook-ui    Synced   H
ealthy              service/guestbook-ui created
2025-05-18T02:08:21+01:00   apps  Deployment     default         guestbook-ui  OutOfSync  M
issing              deployment.apps/guestbook-ui created
2025-05-18T02:08:21+01:00   apps  Deployment     default         guestbook-ui    Synced  Pr
ogressing              deployment.apps/guestbook-ui created

Name:               argocd/guestbook
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          default
URL:                https://localhost:8080/applications/guestbook
Source:
- Repo:             https://github.com/argoproj/argocd-example-apps.git
  Target:           HEAD
  Path:             guestbook
SyncWindow:         Sync Allowed
Sync Policy:        Manual
Sync Status:        Synced to HEAD (6865767)
Health Status:      Progressing

Operation:          Sync
Sync Revision:      68657670d9131dc5bc5f538b14c1de3377d74591
Phase:              Succeeded
Start:              2025-05-18 02:08:20 +0100 IST
Finished:           2025-05-18 02:08:21 +0100 IST
Duration:           1s
Message:            successfully synced (all tasks run)

GROUP  KIND        NAMESPACE  NAME          STATUS  HEALTH  HOOK  MESSAGE
       Service     default    guestbook-ui  Synced                service/guestbook-ui creat
ed
apps   Deployment  default    guestbook-ui  Synced                deployment.apps/guestbook-
ui created
```

Fig. 14. Syncing via CLI

H. Syncing via UI

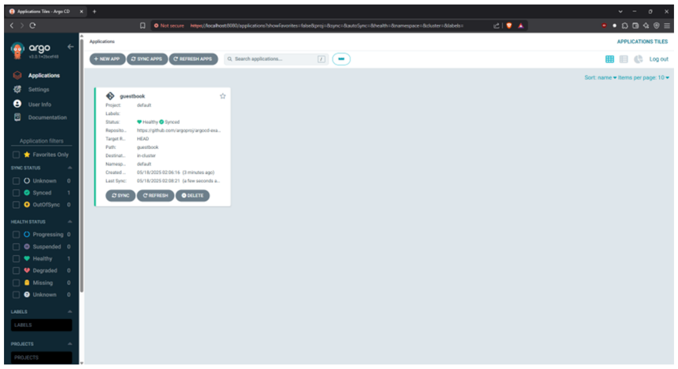


Fig. 15. Syncing via UI 1

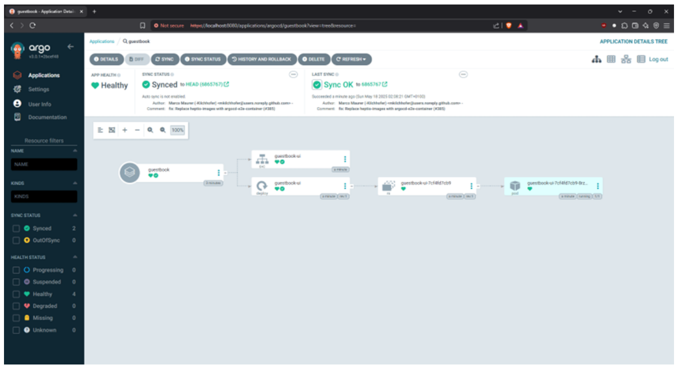


Fig. 16. Syncing via UI 2

IX. Challenges

The following challenges were encountered during the research and practical implementation:

- One of the main challenges was researching recent comparisons between Argo CD and Flux. Many online resources vary in depth and focus. Filtering through the documents and cross referencing sources.
- During the Argo CD setup, a few issues were encountered due to outdated commands found in third-party sources. To resolve this, the official Argo CD documentation was used to ensure command accuracy and version compatibility. The setup process was further streamlined by experience with Kubernetes through earlier labs.

X. Conclusion

In conclusion this report examined the basis of what Git Operations is and covered two popular Git Ops tools Argo CD and Flux. While both tools offer similar core functionality, they differ in architecture, ease of use, and flexibility. Argo CD provides a more user-friendly experience with a built-in UI. Flux, on the other hand, offers greater flexibility appealing to users who prefer a more Kubernetes-native and extensible setup.

Through a hands on use of Argo CD, this report illustrated how Git Ops can be effectively implemented in practice. As organisations continue to scale their cloud infrastructure, adopting Git Ops practices with the right tool can significantly improve deployment and management.

References

[1] A. K. S. H. Florian Beetz, “GitOps,” [Online]. Available: https://www.gitops.tech/. [Accessed May 2025].

[2] GitLab, ”What is GitOps?,” 2025. [Online]. Available: https://about.gitlab.com/topics/gitops/. [Accessed May 2025].

[3] Geekas For Geeks, ”Continuous Deployment to Kubernetes with GitOps,” July 2024. [Online]. Available: https://www.geeksforgeeks.org/continuous-deployment-to-kubernetes-with-gitops/. [Accessed May 2025]

[4] Argo, ”What Is Argo CD,” 2025. [Online]. Available: https://argo-cd.readthedocs.io/en/stable/. [Accessed May 2025].

[5] Flux, ”Flux,” 2025. [Online]. Available: https://fluxcd.io/. [Accessed May 2025].

[6] Geeks For Geeks, ”Top 20 Kubernetes Tools To Use in 2025,” 21 November 2024. [Online]. Available: https://www.geeksforgeeks.org/top-kubernetes-tools/. [Accessed May 2025].

[7] Code Fresh, ”Argo CD vs. Flux: 6 Key Differences and How to Choose,” 2025. [Online]. Available: https://codefresh.io/learn/argo-cd/argo-cd-vs-flux-6-key-differences-and-how-to-choose/. [Accessed May 2025]

[8] J. Walker, ”Flux CD vs. Argo CD: GitOps Tools Comparison,” 28 November 2024. [Online]. Available: https://spacelift.io/blog/flux-vs-argo-cd. [Accessed May 2025].

[9] Cloud Native Landscape, ”Cloud Native Landscape,” 2025. [Online]. Available: https://landscape.cncf.io/?group=projects-and-products. [Accessed May 2025].