Project Report – GitOps using ArgoCD and Kubernetes

Submitted by: *Anandakrishnan B*
Role: *DevOps*

Table of Contents

1. Abstract

This project demonstrates the implementation of a GitOps-based Continuous Deployment workflow using ArgoCD and Kubernetes. The goal of the project is to automate the delivery, synchronization, and lifecycle management of containerized applications by using Git as the single source of truth. Kubernetes is used as the orchestration platform, while ArgoCD continuously monitors the Git repository and ensures that the cluster state always matches the desired configuration. This approach reduces configuration drift, eliminates manual deployment steps, and improves reliability, consistency, and deployment speed. The implementation showcases Minikube for a local Kubernetes environment, Docker for image creation, and GitHub for version-controlled configuration management.

---

2. Introduction

2.1 Overview of DevOps

DevOps is a collaborative culture and methodology that bridges the gap between software development and IT operations. It focuses on automation, continuous integration, continuous delivery, faster releases, and improved system reliability. DevOps emphasizes tools and workflows that ensure rapid development, frequent deployments, and consistent environments. By adopting DevOps practices, organizations can reduce operational bottlenecks, enhance productivity, and accelerate innovation.

2.2 Introduction to GitOps

GitOps is an operational model for Kubernetes that uses Git repositories as the single source of truth for system configuration and application deployment. With GitOps, every cluster change is performed through Git commits instead of manual kubectl commands. A GitOps controller such as ArgoCD continuously monitors the repository and automatically applies changes to the cluster. This ensures version-controlled, auditable, and rollback-friendly deployments.

2.3 Benefits of GitOps

- Declarative and version-controlled deployments
- Automated synchronization and rollback
- Increased reliability and repeatability
- Enhanced security and audit trails
- Eliminates configuration drift
- Faster recovery from failures

---

3. Problem Statement

Traditional Kubernetes deployments often rely on manual steps using CLI tools, which can lead to configuration drift, human errors, and inconsistent environments during updates. There is a lack of automated synchronization between desired and actual cluster state, which affects stability and slows down deployments. A reliable, automated, and version-controlled deployment process is required.

---

4. Objective

The objective of this project is to implement a fully automated GitOps pipeline using ArgoCD and Kubernetes. The solution must ensure that application deployments are triggered by Git commits, synchronized automatically to the cluster, and continuously monitored for drift and rollback needs.

---

5. Scope of the Project

The project scope includes:

- Setting up a Kubernetes environment using Minikube
- Building and pushing Docker images
- Creating Kubernetes manifests for deployment and service
- Hosting manifests on GitHub
- Installing and configuring ArgoCD
- Automating deployments via GitOps
- Demonstrating auto-sync, rollback capability, and UI-based visibility

---

6. Tools and Technologies Used

6.1 Kubernetes

Kubernetes is an open-source container orchestration platform used to automate deployment, scaling, and management of containerized applications. In this project, Kubernetes provides the cluster environment where the application runs and is controlled using declarative manifests.

6.2 ArgoCD

ArgoCD is a declarative GitOps continuous delivery tool for Kubernetes. It automatically pulls changes from a Git repository and synchronizes them with the Kubernetes cluster. It also provides UI, CLI, and automatic drift detection.

6.3 Docker

Docker is a containerization platform used to package applications into lightweight, portable containers. In this project, Docker is used to build and push the application image to DockerHub.

6.4 Git & GitHub

Git is a distributed version control system. GitHub acts as the remote repository for storing Kubernetes manifests. The Git repository serves as the single source of truth in the GitOps model.

6.5 Minikube

Minikube is a local Kubernetes cluster used for testing and development. It is lightweight and allows GitOps implementation on a local machine without requiring cloud infrastructure.

6.6 Kubectl

Kubectl is the command-line tool used to interact with the Kubernetes cluster. It is used to view cluster resources, get logs, and verify ArgoCD deployments.

6.7 DockerHub

DockerHub is a container registry used to store the Docker image. ArgoCD pulls this image through Kubernetes manifests during application deployment.

---

7. System Requirements

7.1 Hardware Requirements

Component	
Processor	: Dual Core or higher
RAM	: 8 GB (recommended for Minikube)
Storage	: 20 GB free space
System	: Windows / Linux / macOS

7.2 Software Requirements

Software version	
OS	: Windows 10 / Ubuntu / macOS
Docker	: Latest
Minikube	: Latest
Kubectl	: Latest
Git	: Latest
Browser	: Chrome / Firefox
Internet	: Required for DockerHub & GitHub

---

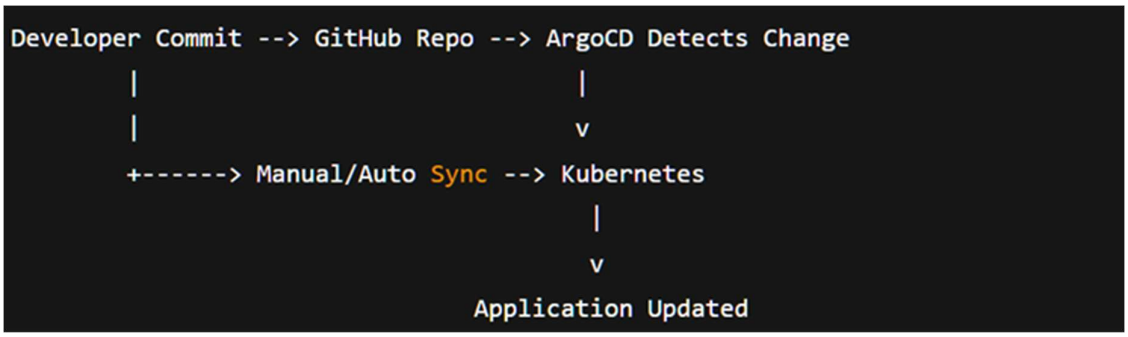
8. Architecture and Workflow

8.1 📌 Architecture Diagram (ASCII – D1)

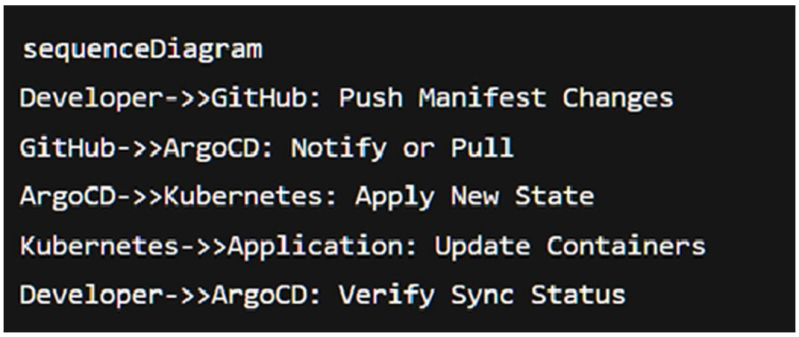
8.1 📌 Architecture Diagram (Mermaid – D2)

```
flowchart TD
A[Developer] --> B[GitHub Repo <br> Source of Truth]
B --> C[ArgoCD <br> Sync and Monitor]
C --> D[Kubernetes Cluster <br> Deployment + Service]
D --> E[Application Running]
```

8.2 📌 GitOps Workflow Diagram (ASCII – D1)



8.2 📌 GitOps Workflow Diagram (Mermaid – D2)



---

9. Implementation / Methodology

9.1 Environment Setup (Minikube + ArgoCD Install)

minikube start

kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

kubectl port-forward svc/argocd-server -n argocd 8081:443

Retrieve password:

kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d

---

9.2 Create Deployment & Service Manifests

Example deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gitops-demo
spec:
  replicas: 1
```

Example service:

```
apiVersion: v1
kind: Service
metadata:
  name: gitops-demo
```

9.3 Push Manifests to GitHub

```
git add .
git commit -m "initial commit"
git push
```

9.4 Configure ArgoCD App

```
kubectl apply -f argocd/application.yaml
```

9.5 ArgoCD Sync and Deployment

- Open UI → Sync
- ArgoCD deploys app to Kubernetes
- Status should show Synced + Healthy

9.6 Verify Application on NodePort

minikube service gitops-demo --url

curl $(minikube service gitops-demo --url)

9.7 Update Git and Observe Auto Sync (GitOps in Action)

docker build -t <user>/gitops-demo:0.2.0 .

docker push <user>/gitops-demo:0.2.0

Edit patch file → push → ArgoCD auto-sync → new version deployed.

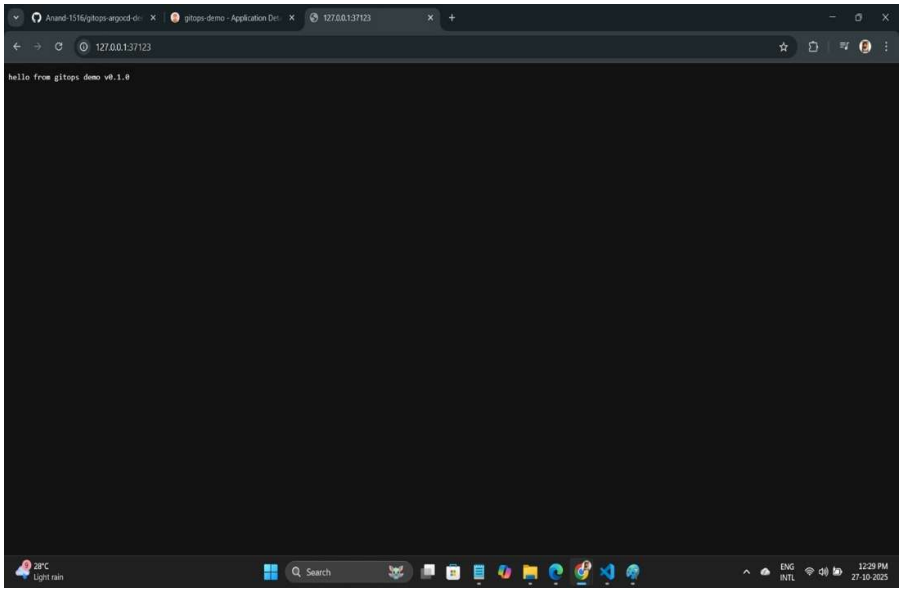
---

10. Results and Observations

10.1 Deployment Output Screenshots

Screenshots to be included:

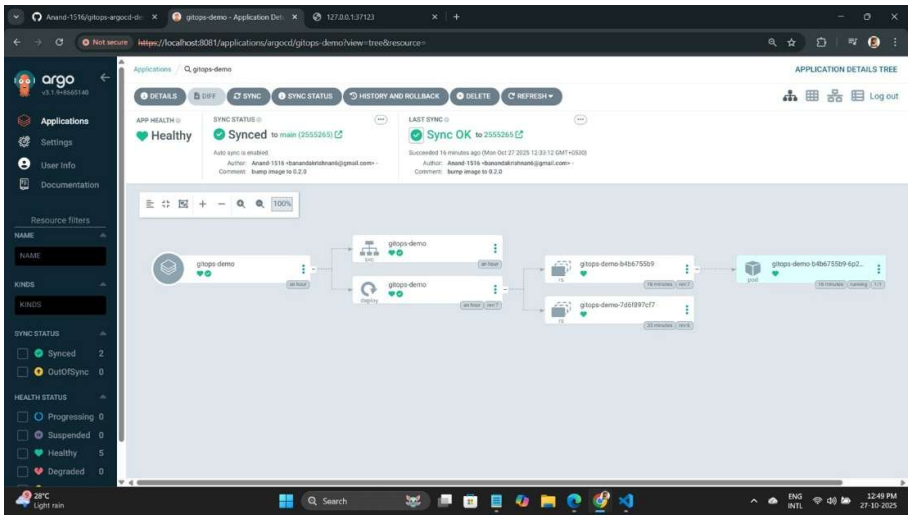
- Kubernetes resources via kubectl get all
- Application output:
- hello from gitops demo v0.1.0



10.2 ArgoCD UI Screenshots

Screenshots to be included:

- Application Synced + Healthy
- ArgoCD dashboard
- Auto-sync-trigger-event

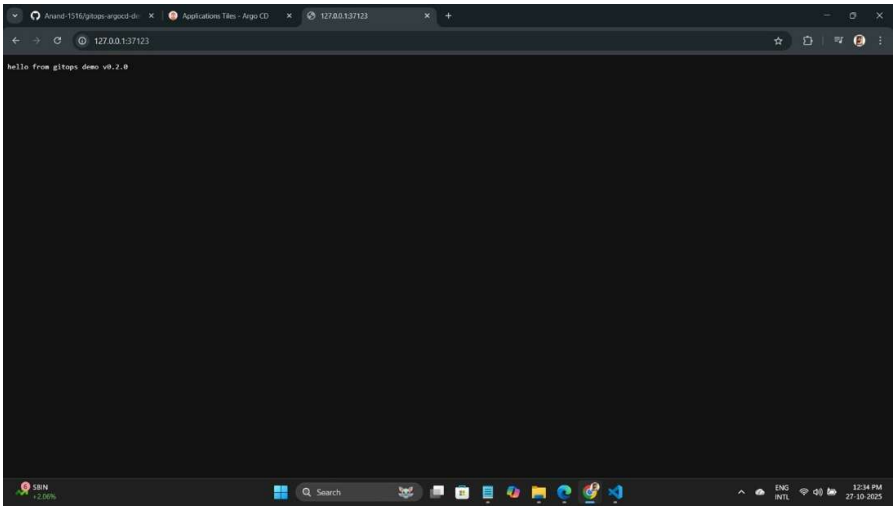


10.3 Auto Sync Result

When the new image version was pushed and Git was updated, ArgoCD automatically detected changes and deployed the new version.

Final output:

hello from gitops demo v0.2.0



ArgoCD confirmed:

- Git state = Cluster state
- Status = Synced
- Health = Healthy

---

11. Advantages of GitOps

- Centralized single source of truth
- Faster deployment with reduced manual effort
- Immutable history and easy rollback
- Increased security and compliance
- Automated reconciliation and drift detection
- Better stability and reliability at scale

---

12. Limitations

- Requires strong understanding of Git and YAML
- Initial setup complexity is high
- Frequent syncs may increase cluster API load
- Secret management needs extra tools (Vault/Sealed Secrets)
- Not ideal for very small or one-time deployments

---

13. Future Enhancements

- Add Argo Rollouts for Canary/Blue-Green deployments
- Integrate Sealed Secrets for secret encryption
- Implement Prometheus + Grafana for monitoring
- Use Helm or Kustomize overlays for multi-environment
- Enable multi-cluster GitOps using ArgoCD

---

14. Conclusion

This project successfully demonstrated a GitOps workflow using ArgoCD and Kubernetes. By storing application and infrastructure definitions in Git, deployments became more reliable, auditable, and automated. ArgoCD continuously synchronized the cluster with Git, ensuring that any change pushed to the repository was automatically reflected in the Kubernetes environment. This approach provides a modern, scalable, and efficient deployment strategy that aligns well with DevOps and cloud-native principles.

---

15. References

- https://argo-cd.readthedocs.io
- https://kubernetes.io
- https://docs.docker.com
- https://minikube.sigs.k8s.io
- https://git-scm.com/docs

Resume Line: Implemented GitOps pipeline using ArgoCD and Kubernetes to automate application deployment from Git commits with real-time sync and rollback capabilities

Thank you