# GitOps Workflow using ArgoCD on Kubernetes

## Loom video:

https://www.loom.com/share/f2a737a2a30d4e5c9c54bb504842f11b?sid=16e7217d-d104-457d-bd14-bd112eb3a298

## Github repo: vommidapuchinni/argocd-k8s

**Abstract:**

This project demonstrates the implementation of GitOps practices using ArgoCD on a local Kubernetes cluster (Minikube). The goal was to automate Kubernetes deployments by storing application manifests in GitHub and enabling continuous synchronization with ArgoCD. This ensures reliable, consistent, and automated deployments for modern cloud-native applications.

**Introduction:**

GitOps is an operational model for Kubernetes where Git acts as the single source of truth. ArgoCD, a declarative GitOps tool, continuously monitors Git repositories and keeps Kubernetes clusters synchronized. This project highlights how GitOps simplifies deployment, improves automation, and provides version control for infrastructure.

**Tools Used:**

- Docker (container runtime for Minikube)
- Minikube (local Kubernetes cluster)
- kubectl (Kubernetes CLI)
- Git & GitHub (version control & repo for manifests)
- ArgoCD (GitOps continuous delivery tool)

**Steps Involved in Building the Project:**

1. Installed prerequisites: Docker, Minikube, kubectl, and Git.
2. Started a Kubernetes cluster with Minikube and verified node status.
3. Installed ArgoCD in the 'argocd' namespace and verified pods and services.
4. Exposed ArgoCD UI using kubectl port-forward and logged in with admin credentials.
5. Created Kubernetes manifests (Deployment & Service) and pushed them to GitHub.
6. Connected GitHub repo to ArgoCD and enabled Auto-Sync.
7. Tested GitOps by scaling Nginx replicas from 1 → 2 via Git commit, observed ArgoCD auto-sync update the cluster.

**Conclusion:**

The project successfully implemented GitOps using ArgoCD and Kubernetes. By storing application manifests in GitHub and enabling automated synchronization, we achieved continuous delivery, version control, and simplified deployment management. This demonstrates how GitOps enhances reliability and consistency in Kubernetes environments

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ minikube start --driver=docker
* minikube v1.33.1 on Microsoft Windows 11 Home Single Language 10.0.26100.4946 Build 26100.4946
* Using the docker driver based on user configuration
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   28s   v1.30.0

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured


umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$
```