# GitOps Workflow using ArgoCD on Kubernetes

## Introduction

GitOps is a modern DevOps practice that leverages Git as the single source of truth for declarative infrastructure and application management. This project demonstrates the implementation of GitOps using ArgoCD to manage and synchronize the deployment of Kubernetes resources directly from a Git repository, ensuring automated and auditable deployments.

## Abstract

The project centers around deploying a GitOps pipeline where changes to Kubernetes manifests in a Git repository are automatically detected and applied to a Kubernetes cluster using ArgoCD. ArgoCD continuously monitors the Git repository and keeps the cluster state in sync with the declared manifests. The approach enhances deployment reliability, auditability, and rollback capabilities. Using a local Kubernetes environment (Minikube or K3s), ArgoCD was deployed, connected to a GitHub repository containing application manifests, and configured for automated sync. Version updates and configuration changes were managed entirely via Git commits.

## Tools Used

- **ArgoCD:** A decarative GitOps continuous delivery tool for Kubernetes.
- **K3s:** Lightweight Kubernetes distributions used to host the cluster locally.
- **GitHub:** Hosts the version-controlled Kubernetes manifests.
- **Docker:** Used to build and manage application container images.
- **Kubectl:** Command-line tool to interact with the Kubernetes cluster.

## Steps Involved in Building the Project

1. **Kubernetes Cluster Setup:** A local Kubernetes cluster was initialized using Minikube (or K3s). The cluster served as the environment for deploying both ArgoCD and the application workloads.

2. **ArgoCD Installation:** ArgoCD was installed within the cluster via manifests or Helm. The ArgoCD UI was exposed using a service, allowing web-based access for monitoring sync status and application health.
3. **Application Manifest Repository:** Kubernetes deployment YAML files were stored in a GitHub repository. These included configurations for the application deployment, service, and optional ingress.
4. **Connecting ArgoCD to GitHub:** ArgoCD was configured with the GitHub repository URL. A new Application resource was created in ArgoCD, pointing to the repo path and target namespace.
5. **GitOps in Action:** With auto-sync enabled, any change pushed to the Git repository (e.g., image version updates) triggered an automatic rollout. ArgoCD monitored the repository and applied changes to the cluster, eliminating the need for manual deployments.
6. **Testing & Verification:** Version updates were pushed via Git commits, and ArgoCD successfully detected and applied them. The UI reflected sync status, revision history, and deployment health.
7. **Documentation & Screenshots:** Screenshots and/or screen recordings were captured to demonstrate successful auto-sync, version upgrades, and rollback capabilities.

**Conclusion**

This project effectively demonstrates the power and simplicity of GitOps using ArgoCD. By using Git as the source of truth, all deployment changes are traceable and reversible. The pipeline ensures consistency across environments, enhances developer productivity, and enables secure, auditable delivery workflows. ArgoCD seamlessly integrates with Kubernetes and GitHub to bring modern DevOps practices into local or production clusters.