GitOps Workflow using Argo CD on Kubernetes
Project Report

Introduction

This project demonstrates the implementation of GitOps using Argo CD, Kubernetes (Minikube), and GitHub. GitOps is a modern approach to continuous deployment where the entire system state is version-controlled and automatically synchronized from a Git repository to the target environment. This ensures consistent, reproducible, and auditable deployments.

Abstract

The main objective of this project is to deploy and manage a sample Nginx application on a Kubernetes cluster using Argo CD. The application's desired state is defined in GitHub as manifests and is automatically synchronized with the Kubernetes cluster. Features like Auto-Sync, Auto-Prune, and Self-Heal ensure that the cluster remains aligned with the declared state, even when manual changes are introduced or unwanted drift occurs. This project highlights the ease, reliability, and efficiency of GitOps-driven deployments.

Tools Used

- **Minikube** – Local Kubernetes cluster for testing and development
- **Argo CD** – Continuous delivery tool that follows GitOps principles
- **GitHub** – Repository to store Kubernetes manifests and configuration files
- **kubectl / argocd CLI** – Tools to interact with the cluster and manage deployments
- **Docker** – Used as the driver for running Minikube locally

Steps Involved in Building the Project

1. **Setup Minikube**
    - Installed and started Minikube with Docker as the driver.
    - Verified the cluster using kubectl cluster-info.
2. **Install Argo CD**
    - Created a namespace and installed Argo CD using the official manifest.
    - Verified pods and exposed the UI using port-forwarding.
3. **Configure Argo CD**

- Retrieved the initial admin password and accessed the UI.
- Optionally installed the argocd CLI for enhanced control.

4. **Prepare Application in GitHub**
    - Created a repository with Kubernetes manifests including deployment.yaml, service.yaml, and kustomization.yaml.
    - Defined a Nginx deployment with 2 replicas and exposed it through a LoadBalancer.
5. **Create Application in Argo CD**
    - Configured the application in Argo CD with the repository URL and automated sync settings.
6. **Verify GitOps Workflow**
    - **Auto-Sync**: Changes in Git are automatically reflected in the cluster.
    - **Auto-Prune**: Removing a resource from Git leads to its removal from the cluster.
    - **Self-Heal**: Manual changes are automatically corrected to match the Git state.

Conclusion

This project successfully demonstrated how GitOps enables reliable and automated deployment workflows using Argo CD and Kubernetes. By leveraging Git as the single source of truth, the deployment process becomes more traceable, consistent, and resistant to unintended changes. The use of Auto-Sync, Auto-Prune, and Self-Heal ensures that the cluster's state remains aligned with the desired configuration, even in dynamic or error-prone environments. This foundational experience will be beneficial for future projects involving continuous delivery, infrastructure as code, and cloud-native deployments.