GitOps Workflow Using ArgoCD on Kubernetes

Introduction

In modern DevOps practices, **GitOps** has emerged as a powerful methodology to automate and manage infrastructure and application deployments using Git as the single source of truth. GitOps ensures that any changes to the application or infrastructure are version-controlled, auditable, and automatically deployed to the Kubernetes cluster. This project demonstrates the implementation of a GitOps pipeline using **ArgoCD** to automate the deployment of Kubernetes applications from a GitHub repository.

Abstract

The project implements a fully automated workflow where Kubernetes deployment and service manifests are stored in a GitHub repository. ArgoCD continuously monitors the repository and ensures that the cluster state always matches the desired state defined in Git. Any updates, such as version changes in container images, are automatically applied, providing a reliable, self-healing, and reproducible deployment process. This approach simplifies application lifecycle management, reduces manual errors, and accelerates deployment cycles.

Tools Used

- **Kubernetes** (Minikube/K3s) – Local cluster for testing deployments
- **ArgoCD** – GitOps tool to manage application sync and automation
- **GitHub** – Repository to store Kubernetes manifests
- **Docker** – Container platform to host application images
- **kubectl** – Command-line tool for Kubernetes management

Steps Involved in Building the Project

1. **Setup Kubernetes Cluster**
    - Installed Minikube to create a local Kubernetes environment.
2. **Deploy ArgoCD**
    - Deployed ArgoCD on the cluster in the argocd namespace.

- Verified ArgoCD pods (argocd-server, argocd-application-controller) were running and healthy.

3. **Create Kubernetes Manifests**
   - Wrote deployment.yaml and service.yaml under base/ folder for the application.
   - Configured deployments with replicas, container images, and service exposure.
4. **Configure ArgoCD Application**
   - Created an ArgoCD Application manifest (app.yaml) pointing to the GitHub repository and base/ path.
   - Enabled **auto-sync** with self-healing to ensure the cluster always matches the repository state.
5. **Version Control and Deployment**
   - Pushed Kubernetes manifests to GitHub.
   - ArgoCD automatically detected changes and deployed resources to the cluster.
   - Updated the application version by modifying the image tag in deployment.yaml and observing ArgoCD roll out the new version seamlessly.
6. **Verification and Monitoring**
   - Used kubectl get pods and kubectl describe deployment to verify pods were running correctly.
   - ArgoCD UI showed applications as Synced and Healthy, confirming successful deployment and synchronization.

Conclusion

This project successfully implements a **GitOps pipeline** using ArgoCD on Kubernetes. By storing Kubernetes manifests in GitHub and enabling automatic synchronization, it demonstrates the core principles of GitOps: **version control, automated deployment, and self-healing infrastructure**. This workflow reduces manual intervention, ensures reproducibility, and provides a professional foundation for scalable application deployment in a Kubernetes environment.