**GitOps with Kubernetes for Multi-Cloud Using Argo CD**

**Overview**

Implementing GitOps with Argo CD enables seamless deployment and management of applications across multiple cloud environments using Kubernetes. This guide provides a step-by-step process for setting up Argo CD and deploying applications in a multi-cloud setup.

**Prerequisites**

* Kubernetes clusters set up in multiple cloud environments (AWS EKS, Azure AKS, GCP GKE, etc.)
* Argo CD installed and configured
* Git repository for storing Kubernetes manifests
* Helm and Terraform for infrastructure as code (optional)

**Step 1: Install Argo CD**

1. Deploy Argo CD in a dedicated namespace:
2. kubectl create namespace argocd

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

After installation check these  
kubectl get svc argocd-server -n argocd

kubectl get pod -n argocd

1. Access Argo CD UI:

kubectl port-forward svc/argocd-server -n argocd 8080:443

If you are installing argocd on cluster so you can also use loadbalancer for that external ip

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

1. Retrieve the initial admin password:

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

1. Login to Argo CD CLI:

We need to install the argo cd cli

curl -sSL -o argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64

chmod +x argocd

sudo mv argocd /usr/local/bin/

argocd login localhost:8080 --username admin --password <retrieved-password>

**Step 2: Connect Kubernetes Clusters**

1. First check the cluster details  
   kubectl config current-context

kubectl config get-contexts

Eg: gke\_idp-project-deploy\_us-central1-a\_first-cluster

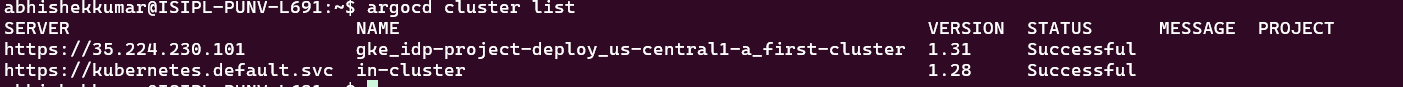
1. Register your clusters with Argo CD:

argocd cluster add <context-name>

Example for an AWS EKS cluster:

argocd cluster add arn:aws:eks:<region>:<account-id>:cluster/<cluster-name>

argocd cluster add gke\_idp-project-deploy\_us-central1-a\_first-cluster



**Step 3: Define Application in Git Repository**

1. Create a Git repository with Kubernetes manifests or Helm charts.
2. Define an Argo CD Application manifest (application.yaml):

|  |
| --- |
| # Application for default cluster (minikube)  apiVersion: argoproj.io/v1alpha1  kind: Application  metadata:  name: test-argo-application  namespace: argocd  spec:  project: test  source:  repoURL: <https://github.com/ABHISHEK-KUMAR-14022001/argocd-app-config.git>  targetRevision: HEAD  path: test  destination:  server: <https://kubernetes.default.svc>  namespace: myapp  syncPolicy:  syncOptions:  - CreateNamespace=true  automated:  selfHeal: true  prune: true  ---  # Application for new cluster (idp-project-deploy)  apiVersion: argoproj.io/v1alpha1  kind: Application  metadata:  name: deploy-argo-application  namespace: argocd  spec:  project: prod  source:  repoURL: <https://github.com/ABHISHEK-KUMAR-14022001/argocd-app-config.git>  targetRevision: HEAD  path: prod  destination:  server: <https://35.224.230.101> # External API of new cluster  namespace: deployapp  syncPolicy:  syncOptions:  - CreateNamespace=true  automated:  selfHeal: true  prune: true |

1. Ensure Projects Exist in ArgoCD

argocd proj list

If `test` and `deploy` are missing, create them:

argocd proj create test --description "Test Project"

argocd proj create prod --description "Deploy Project"

Ensure the Source Repositories Are Allowed

argocd proj add-source test https://github.com/ABHISHEK-KUMAR-14022001/argocd-app-config.git

argocd proj add-source prod https://github.com/ABHISHEK-KUMAR-14022001/argocd-app-config.git

Ensure Destinations Are Correct

argocd proj add-destination test https://kubernetes.default.svc myapp

argocd proj add-destination prod https://35.224.230.101 deployapp

1. Apply the manifest:

kubectl apply -f application.yaml -n argocd

1. you should see both applications **syncing successfully** in the ArgoCD UI or via CLI:

argocd app list



**Step 4: Sync and Deploy Applications**

1. Manually sync the application (if not automated):

argocd app sync my-multi-cloud-app

1. Monitor application status:

argocd app get my-multi-cloud-app

**Step 5: Automate Infrastructure with Terraform (Optional)**

1. Define cloud resources in Terraform files (main.tf).
2. Apply Terraform configuration:
3. terraform init

terraform apply -auto-approve

1. Store Terraform state in a shared backend for multi-cloud management.

**Step 6: Use Flux CD for Complementary Deployments (Optional)**

* For additional GitOps capabilities, integrate Flux CD to manage cluster configurations across clouds.
* Example Flux installation:

kubectl apply -f https://raw.githubusercontent.com/fluxcd/flux2/main/install.yaml

**Benefits**

* **Simplified Multi-Cloud Deployment**: Centralized GitOps approach for managing applications across AWS, Azure, and GCP.
* **Better Consistency**: Ensures uniform configurations across multiple clusters.
* **Automated Rollbacks**: Argo CD enables quick rollbacks in case of failures.
* **Declarative Infrastructure**: Manifests stored in Git enable version-controlled infrastructure management