# GitOps using ArgoCD in AWS EKS

By:

Kavan Dalwadi

**Agenda :**

We are creating a fully functional CI/CD pipeline for AWS EKS multi cluster and will be using GitHub as SCM of our application code.

Github action will be used for Continuous Integration of the source code, i.e. it will build docker image from the repo and push that image to Docker Hub, with two tags latest and git commit.

For Continuous Delivery we'll be using ArgoCD which is a GitOps tool, for performing PULL based updates to our cluster. We are using GitLab as SCM for our infrastructure code. Any changes made in that GitLab repo will be reflected in our cluster.

On AWS EKS we have created two clusters one for dev environment and another for production environment. We have installed argoCD on the production cluster and it is connected to the dev cluster.

We have two folders named ‘dev’ and ‘prod’ in GitLab where all the config related to EKS cluster are saved. So if we change the deployment image in our dev folder, a new replicaSet with new pods having a new image will be created.

Lastly, we will configure a webhook in GitLab, to trigger argocd server on every commit pushed.

**Prerequisites :**

Install AWS cli , docker, argoCD, kubectl, eksctl

**Tools Used:**

GitHub Repo: Application SCM - having 3 branch dev, stage, master(prod)

GitHub actions: CI

GitLab Repo: EKS Infrastructure SCM - having two subfolder dev, prod

AWS EKS: K8s cluster - dev and prod

**What is GitOps ?**

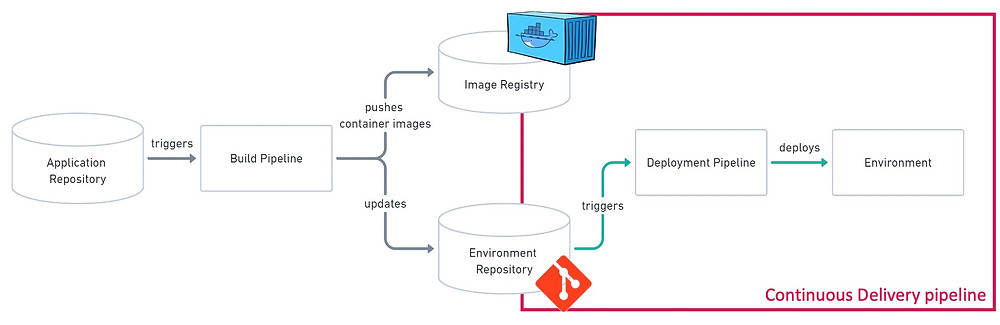
GitOps is an approach that enables developers/DevOps to automate infrastructure and manage it alongside their codebase.

GitOps involves using Git to manage infrastructure and application configurations. Git is an open source version control system that serves as a single source of truth for declarative infrastructure and applications.

**Before GitOps, a DevOps pipeline in Kubernetes involved the following steps:**

1. Developer commits code to a repo. This could be a container image specification or declarative configuration such as a YAML file or Helm chart.
2. The CI server performs the build, parsing the image specification and creating the image.
3. The CI server pushes the new image into an image repository.
4. An automated continuous delivery (CD) tool deploys the configuration directly into the Kubernetes cluster, usually via scripted kubectl commands.

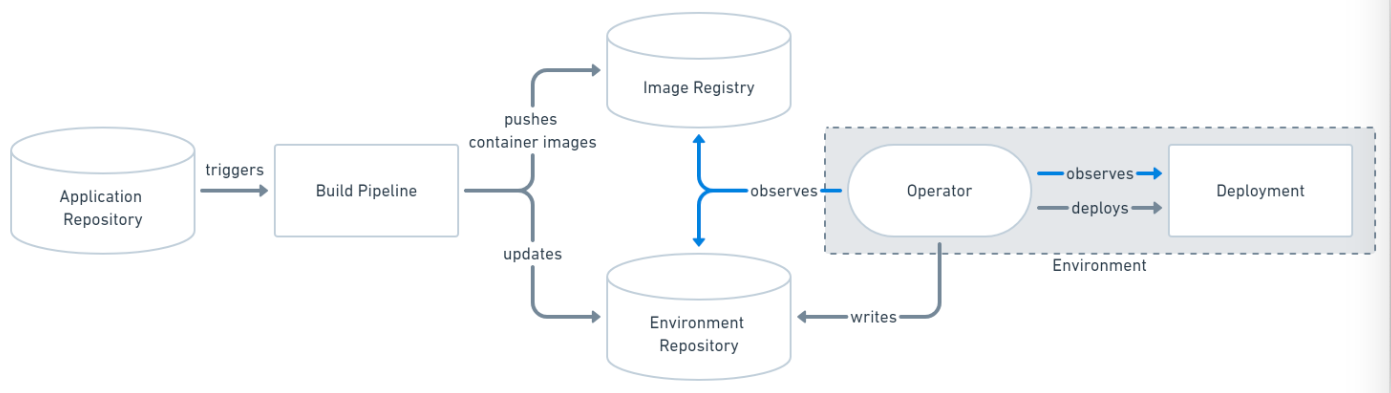
Push based Deployment:



How does GitOps work?

1. Developer commits code to repo
2. CI server builds the new version of the code and creates the image
3. CI server pushes the image to a repo  
   (up until now the process is the same as in a traditional environment)
4. A GitOps agent deployed in the cluster identifies the change and automatically makes the necessary changes in the cluster. There is no direct connection between the CI system and the Kubernetes cluster.

Now, if a change made to a container image or a manifest was undesirable—for example, because it contained a bug or was performed by a bad actor—it is possible to immediately revert to the last good configuration in the git repository. Git becomes the single source of truth for application and environment state.

Pull based Deployment:

* GitOps Agent installed in environment, e.g. AWS EKS cluster
* It Monitors and Compares desired state (Git Repo) with actual state (Cluster).
* Applies the changes necessary to get to desired state.

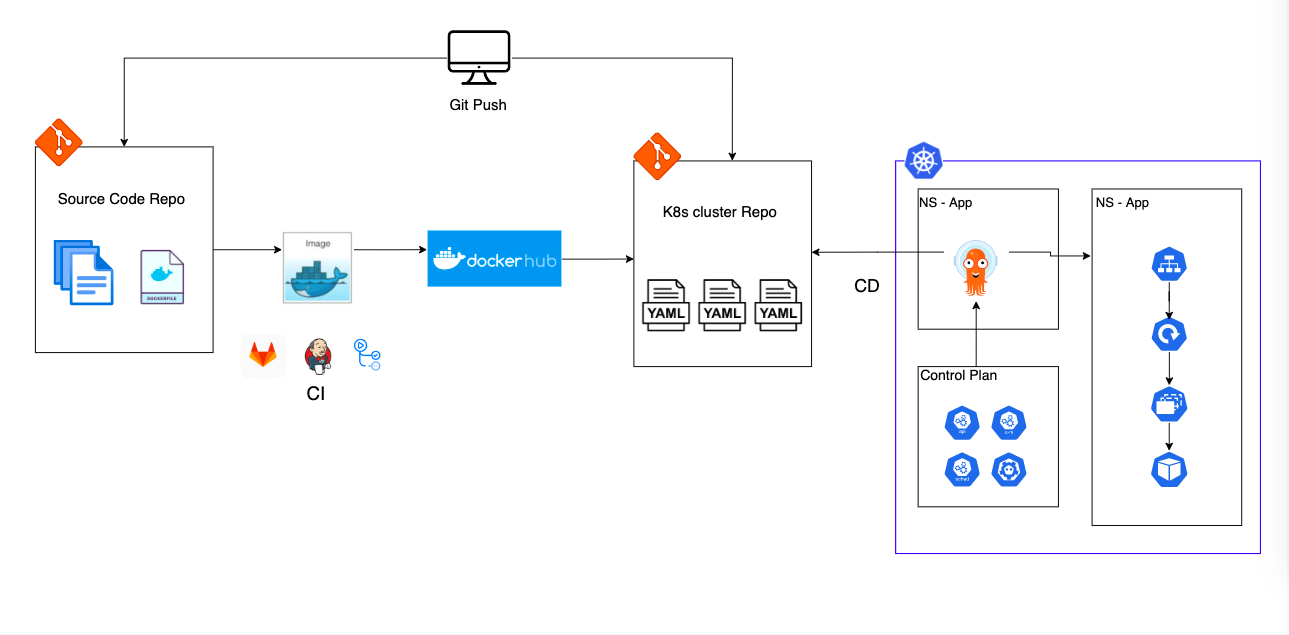
**ArgoCD**

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes.

Argo CD follows the **GitOps** pattern of using Git repositories as the source of truth for defining the desired application state. Kubernetes manifests can be specified in several ways:

* kustomize applications
* helm charts
* ksonnet applications
* jsonnet files
* Plain directory of YAML/json manifests
* Any custom config management tool configured as a config management plugin

Argo CD is implemented as a kubernetes controller which continuously monitors running applications and compares the current, live state against the desired target state (as specified in the Git repo). A deployed application whose live state deviates from the target state is considered OutOfSync. Argo CD reports & visualizes the differences, while providing facilities to automatically or manually sync the live state back to the desired target state. Any modifications made to the desired target state in the Git repo can be automatically applied and reflected in the specified target environments.



**Install ArgoCD:**

<https://argo-cd.readthedocs.io/en/stable/getting_started/#1-install-argo-cd>

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

**OR**

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

**Login ArgoCD:**

<https://argo-cd.readthedocs.io/en/stable/getting_started/#4-login-using-the-cli>

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

**ArgoCD Configuration:**

<https://argo-cd.readthedocs.io/en/stable/operator-manual/declarative-setup/>

**GitLab URL:**

<https://gitlab.com/Kavan-Dalwadi/gitops_argocd.git>

Create AWS EKS Cluster using aws cli:

eksctl create cluster --name <name> \

--version 1.22 \

--region <region> \

--nodegroup-name my-nodes \

--node-type t3.small \

--managed --nodes 2 \

--ssh-public-key <key\_name> \

--node-ami-family Ubuntu2004

**Webhook:**

​​https://<argocd\_server\_url>/api/webhook

**To add another cluster:**

Download argocd cli

argocd login --insecure <argocd\_serverIP:PORT>

argocd cluster add <cluster\_endpoint\_URL>

Application yaml:

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: myapp-argo

namespace: argocd

spec:

project: default

source:

repoURL: https://gitlab.com/Kavan-Dalwadi/gitops\_argocd.git

targetRevision: HEAD

path: dev

destination:

server: https://kubernetes.default.svc

namespace: default

syncPolicy:

syncOptions:

- CreateNamespace=true

automated:

selfHeal: true

prune: true