ARGOCD and K8S

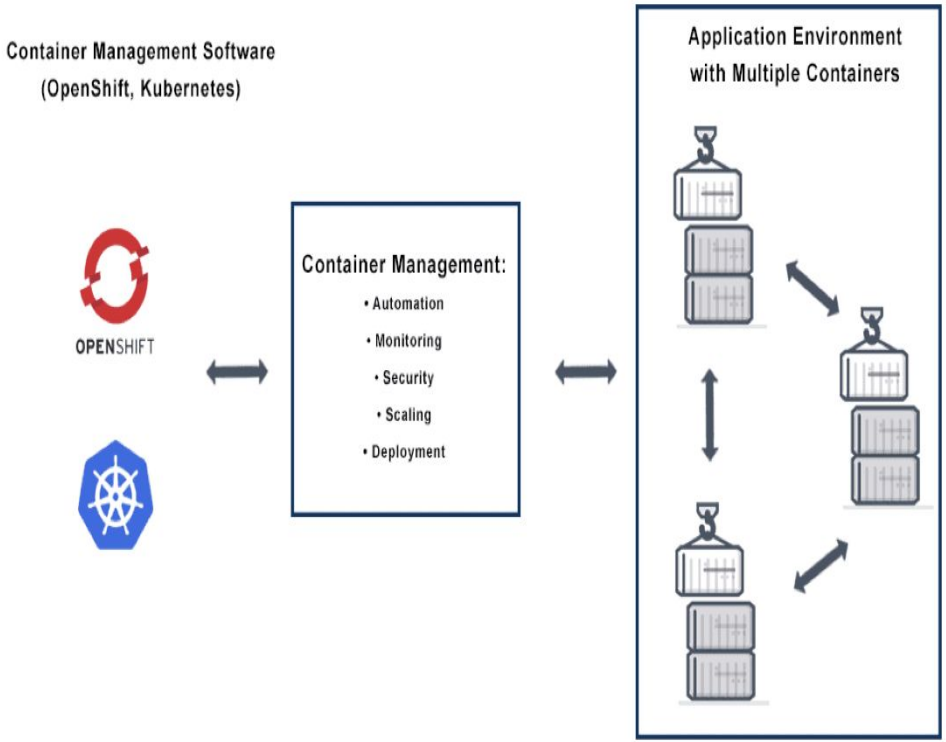
Containerizing and Deploying Application using K8s and ArgoCD

What is Kubernetes and Why?

Containers are short lived, fragile and ephemeral.

They need someone to manage them based on health checks, scheduling, scalability requirements.

Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. It was originally developed by Google and later donated to the Cloud Native Computing Foundation (CNCF). Kubernetes is derived from their internal Borg system, which was used to manage their massive-scale containerized applications.



Key benefits of Kubernetes

Container Orchestration: Kubernetes automates the deployment, scaling, and management of containerized applications.

Scalability: Kubernetes enables horizontal scaling of applications by adding or removing replicas based on demand.

High Availability: Kubernetes ensures high availability by automatically distributing containers across multiple nodes in the cluster. If a node fails, the system reschedules the affected containers to healthy nodes, minimizing downtime.

Portability: Ability to run applications consistently and seamlessly across various environments, such as on-premises data centers, public clouds, private clouds, or even on local development machines.

Self-Healing: Kubernetes continuously monitors the health of containers and nodes. If a container or node becomes unhealthy, Kubernetes automatically restarts or reschedules it on a healthy node to ensure the desired state is maintained.

Service Discovery and Load Balancing: Kubernetes automatically assigns a stable IP address and DNS name to services, allowing other applications to discover and communicate with them. It also performs load balancing across service replicas.

Declarative Configuration: Kubernetes allows users to declare the desired state of the system using YAML or JSON manifests.

Automated Rollouts and Rollbacks: Kubernetes supports rolling updates for application deployments. It allows you to update containers without downtime and provides the option to roll back to a previous version if issues arise.

Day “X" terminology-

Day 0: Day 0 refers to the initial setup and deployment phase of a Kubernetes cluster. This is the starting point, where you plan and prepare to create the cluster. Tasks in Day 0 might include:

Selecting the appropriate infrastructure, Installing Kubernetes, Network configuration, Security considerations, Installing add-ons

Day 1: Day 1 represents the cluster deployment and "go-live" phase. It includes activities involved in launching and making the cluster operational. Tasks in Day 1 might include:

Setting up namespaces, Deploying applications, Creating and managing persistent volumes, Testing and validation, Setting up monitoring and alerting

Day 2: Day 2 refers to the ongoing operations and maintenance phase of the Kubernetes cluster. It involves the day-to-day tasks of managing and ensuring the smooth functioning of the cluster. Tasks in Day 2 might include:

Scaling applications, Upgrading Kubernetes, Patching and maintenance, Backup and disaster recovery, Troubleshooting and debugging, Resource optimization

Deployment of an microservices application to Kubernetes cluster following GitOps Approach

The objective of this project is to deploy a scalable and secure application on Kubernetes Infrastructure using GitOps practices and to establish a Monitoring/Observability solution.

Day 0

- Setup Kubernetes Cluster and worker nodes following the industry best practices
- Understand key terms in Kubernetes
- Setup some additional tools with Kubernetes cluster
- Setup Default Kubernetes Dashboard
- Integrate with 3rd Party tools for cluster visibility

Day 1

- Deploy a complex Application that has database in microservice
- Understand why we need GitOps principles
- Follow the GitOps principles using Argo CD
- Implement blue-green and/or canary deployment strategies to minimize downtime during updates

Day 2

- Monitoring of Kubernetes Cluster using Prometheus and Grafana
- Explore commercial tools for monitoring Kubernetes clusters that offer free trial

Key Terminology and Architecture of Kubernetes

Control plane

Data plane (Worker Nodes)

Kube-apiserver

etcd

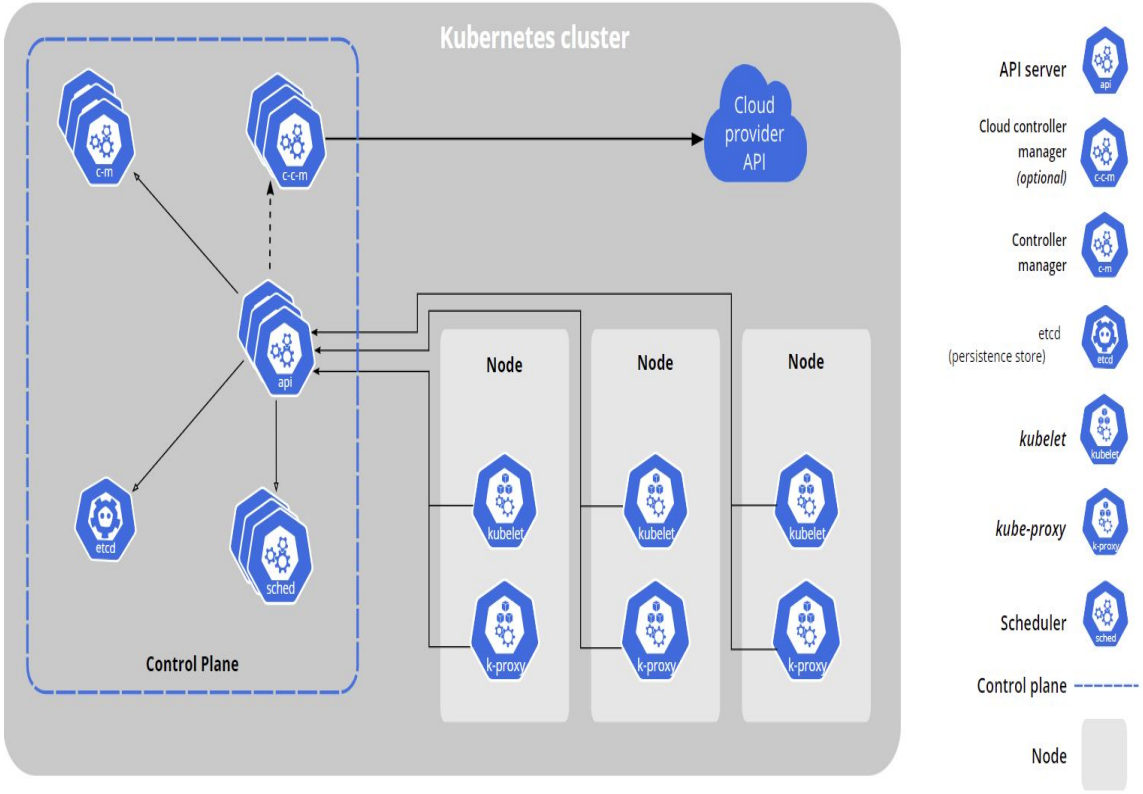
Kube-scheduler

Kube-controller-manager

Cloud-controller-manager

Kubelet

Kube-proxy



Control Plane

The control plane is responsible for managing the overall state of the Kubernetes cluster and making high-level decisions about its operation. It acts as the "brain" of the cluster, coordinating and orchestrating various tasks. The main components of the control plane are:

API Server: The API server is the front-end interface to the Kubernetes control plane. It exposes the Kubernetes API, which allows users and other components to interact with the cluster and manage resources.

etcd: etcd is a distributed key-value store that stores the entire state of the Kubernetes cluster. The control plane components use etcd as the central data store to store configuration data, current state, and metadata about the cluster.

Scheduler: The scheduler is responsible for placing newly created pods onto suitable nodes in the cluster. It considers various factors such as resource requirements, node availability, and affinity/anti-affinity rules to make intelligent placement decisions.

Controller Manager: The controller manager runs various controllers that monitor the cluster state and drive it towards the desired state. Examples of controllers include the Replication Controller, Deployment Controller, and Node Controller.

Cloud Controller Manager (optional): If running on a cloud provider, the cloud controller manager interfaces with the cloud provider's APIs to manage cloud-specific resources such as load balancers, storage volumes, and virtual machines.

Key Terminology and Architecture of Kubernetes…

Namespace

Pod

Deployment

RepilcaSet

Secret

Kubeconfig

StatefulSet

PersistentVolume

PersistentVolumeClaim

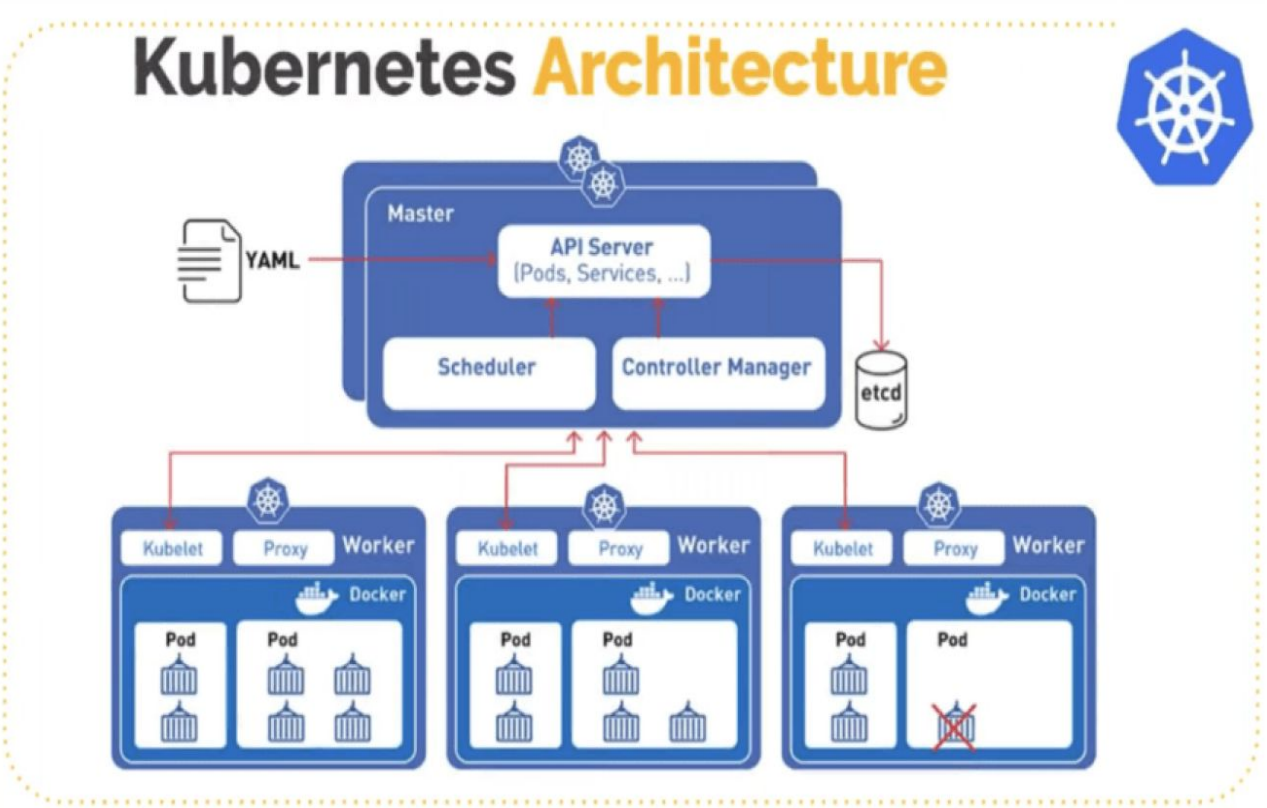
Kubernetes taints

Config Map

Service

Internal Service (ClusterIP)

External Service



Setup Kubernetes Cluster

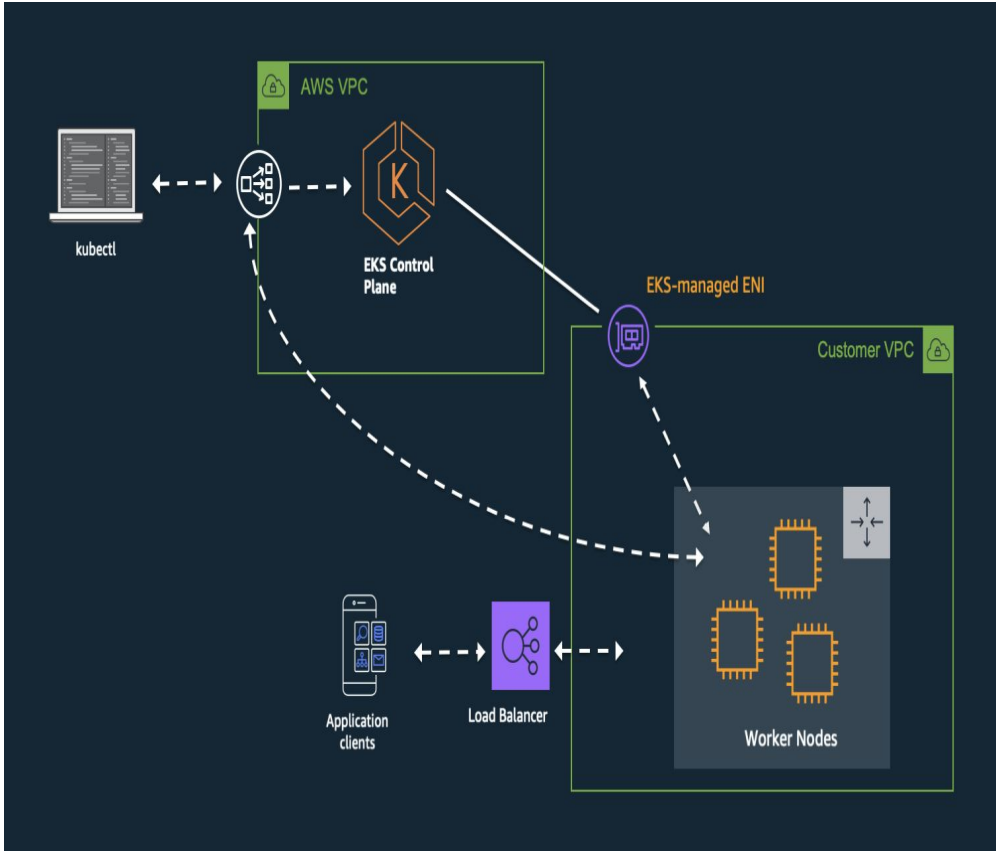
There are multiple options to setup a Kubernetes cluster

Preferred Options:

- EKS
- AKS
- GKE

Free options but not production grade:

- Minikube (https://minikube.sigs.k8s.io/docs/start/)
- Cevo
- kind
- K3S



EKS Setup

Create Secret key and Access key
export AWS_ACCESS_KEY_ID=AKIATPONSXRFFLZO5FEZ
export AWS_SECRET_ACCESS_KEY=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
export AWS_DEFAULT_REGION="ap-southeast-2"

Install ekctl cli and excute the below command (https://eksctl.io/installation/)
Without node group-
eksctl create cluster --name devops-eks --region=ap-south-1 --zones=ap-south-1a,ap-south-1b
--without-nodegroup
Add node group-
eksctl create nodegroup --cluster devops-eks --region ap-south-1 --name devops-eks-nodegroup
--node-type t3.small --managed --nodes 2 --nodes-min 1 --nodes-max 3 --ssh-access --ssh-public-key
mykeypair1 --asg-access
Delete the cluster-
eksctl delete nodegroup --cluster devops-eks --region ap-southeast-2 --name devops-eks-nodegroup

Output-

```
-bash: 00~eksctl: command not found
Ankits-MacBook-Air:Downloads ankitshukla$ eksctl create cluster --name devops-eks --region=ap-south-1 --zones=ap-south-1a,ap-south-1b
2025-01-23 21:06:02 [ℹ]  eksctl version 0.193.0
2025-01-23 21:06:02 [ℹ]  using region ap-south-1
2025-01-23 21:06:02 [ℹ]  subnets for ap-south-1a - public:192.168.0.0/19 private:192.168.64.0/19
2025-01-23 21:06:02 [ℹ]  subnets for ap-south-1b - public:192.168.32.0/19 private:192.168.96.0/19
2025-01-23 21:06:02 [ℹ]  nodegroup "ng-2e1d8e42" will use "" [AmazonLinux2/1.30]
2025-01-23 21:06:02 [ℹ]  using Kubernetes version 1.30
2025-01-23 21:06:02 [ℹ]  creating EKS cluster "devops-eks" in "ap-south-1" region with managed nodes
2025-01-23 21:06:02 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2025-01-23 21:06:02 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-south-1 --cluster=devops-eks'
2025-01-23 21:06:02 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "devops-eks" in "ap-south-1"
2025-01-23 21:06:02 [ℹ]  CloudWatch logging will not be enabled for cluster "devops-eks" in "ap-south-1"
2025-01-23 21:06:02 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-south-1 --cluster=devops-
eks'
2025-01-23 21:06:02 [ℹ]  default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2025-01-23 21:06:02 [ℹ]
2 sequential tasks: { create cluster control plane "devops-eks",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "ng-2e1d8e42",
    }

2025-01-23 21:06:02 [ℹ]  building cluster stack "eksctl-devops-eks-cluster"
2025-01-23 21:06:03 [ℹ]  deploying stack "eksctl-devops-eks-cluster"
2025-01-23 21:06:33 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:07:03 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:08:03 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:09:03 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:10:04 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"

2025-01-23 21:11:04 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:12:04 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:13:04 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:14:05 [ℹ]  waiting for CloudFormation stack "eksctl-devops-eks-cluster"
2025-01-23 21:14:06 [ℹ]  recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the requested permissions; the re
```

Deployment of an Application to Kubernetes Cluster using Argocd

We can use the following git repo for this project:

https://github.com/aws-containers/retail-store-sample-app

We used the PetClinic App in our Last month Project.

https://github.com/spring-projects/spring-petclinic

We will follow the Argo CD GitOps approach. Why?

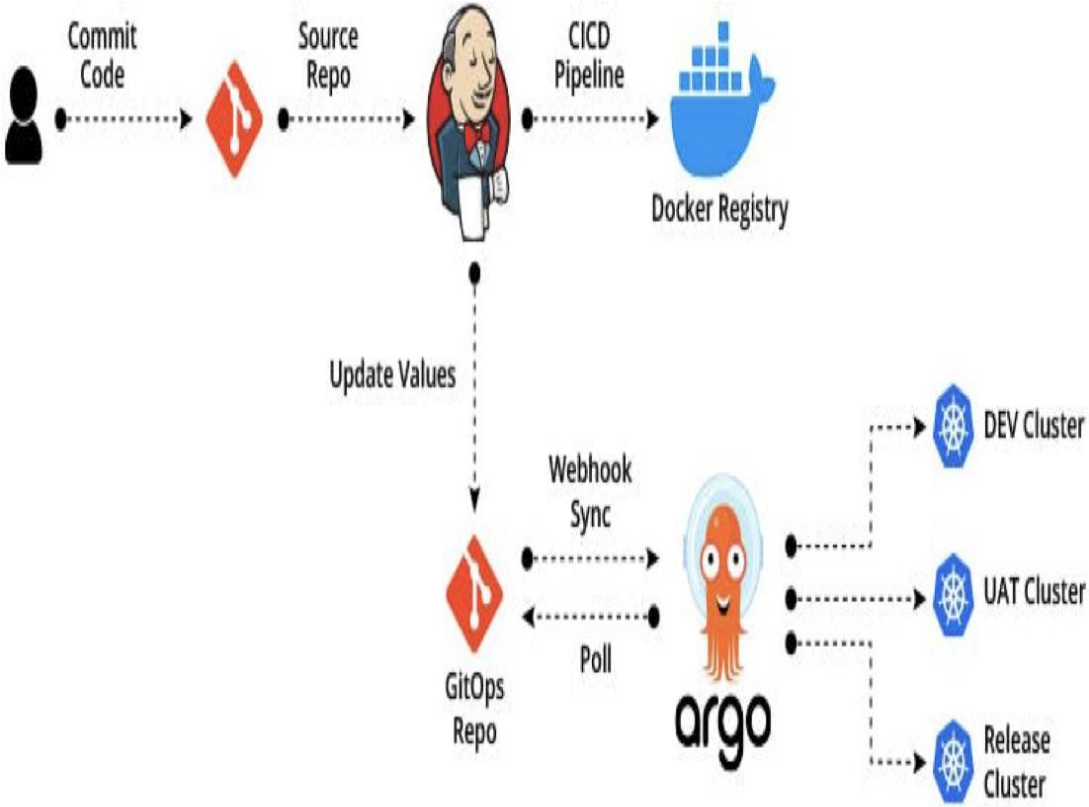


Image 1: CI/CD Flow with Argo CD

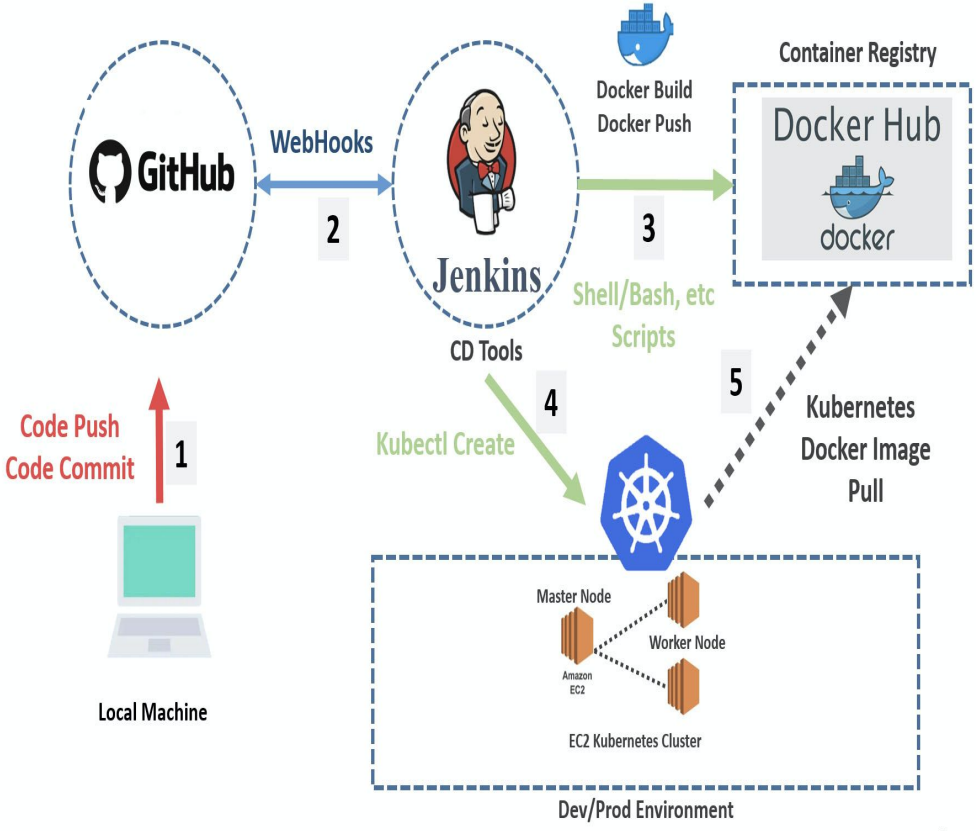
Issues with Jenkins ,Github Action and Other CI/CD

Install kubectl on Jenkins

Provide Jenkins/github access to Cloud and Kubernetes.

No visibility of the environment after deployment.

Rollbacks are manual.



How Argo CD Solves this?

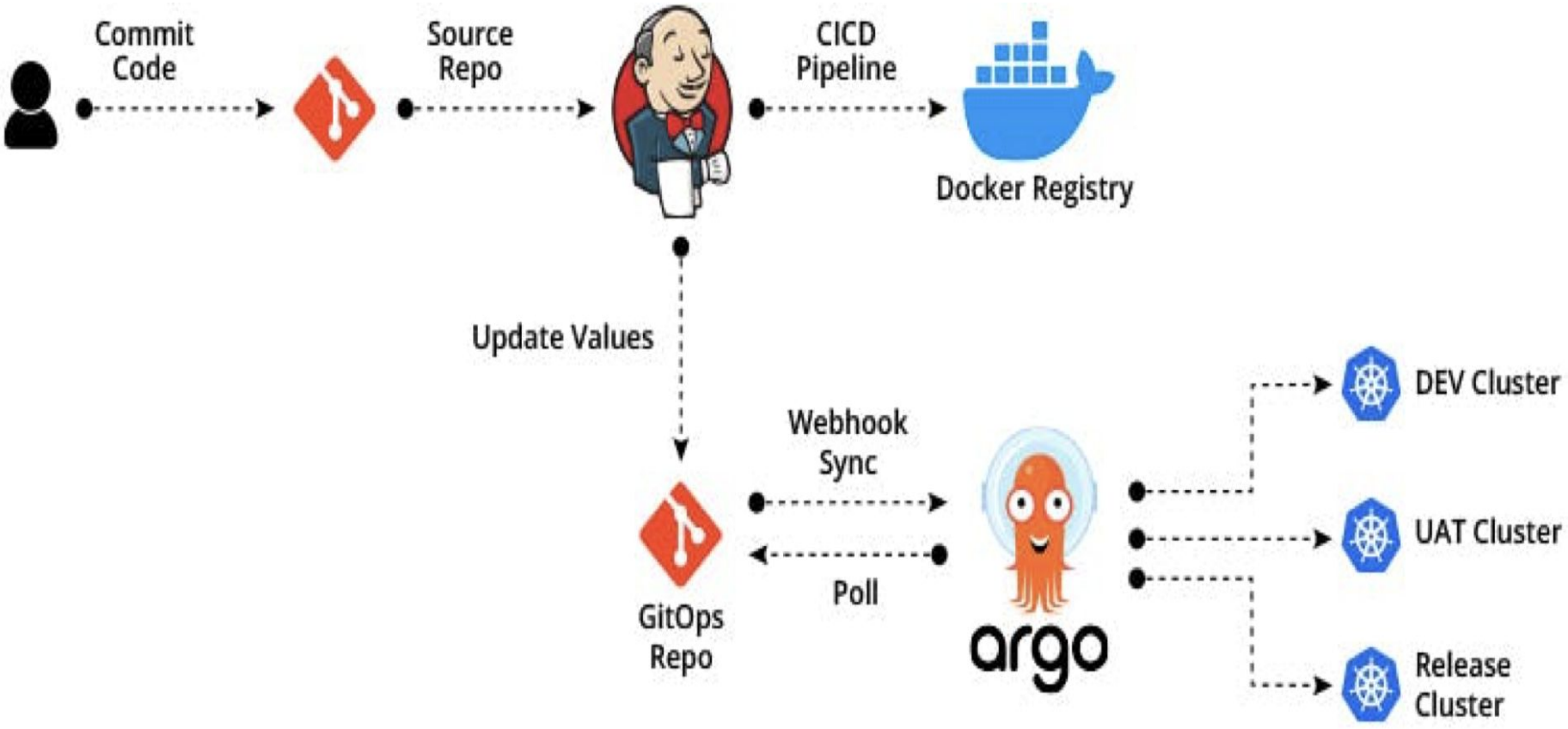
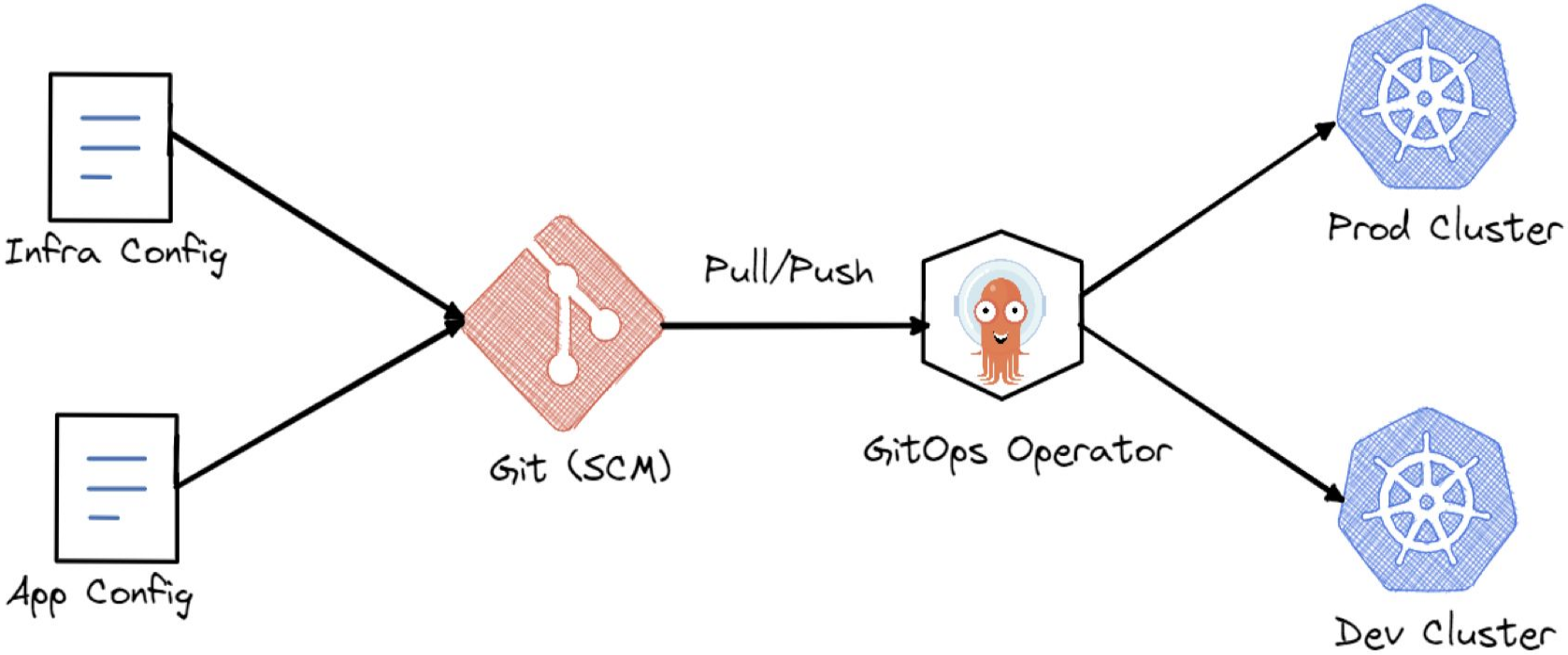


Image 1: CI/CD Flow with Argo CD

Branching Strategy to follow



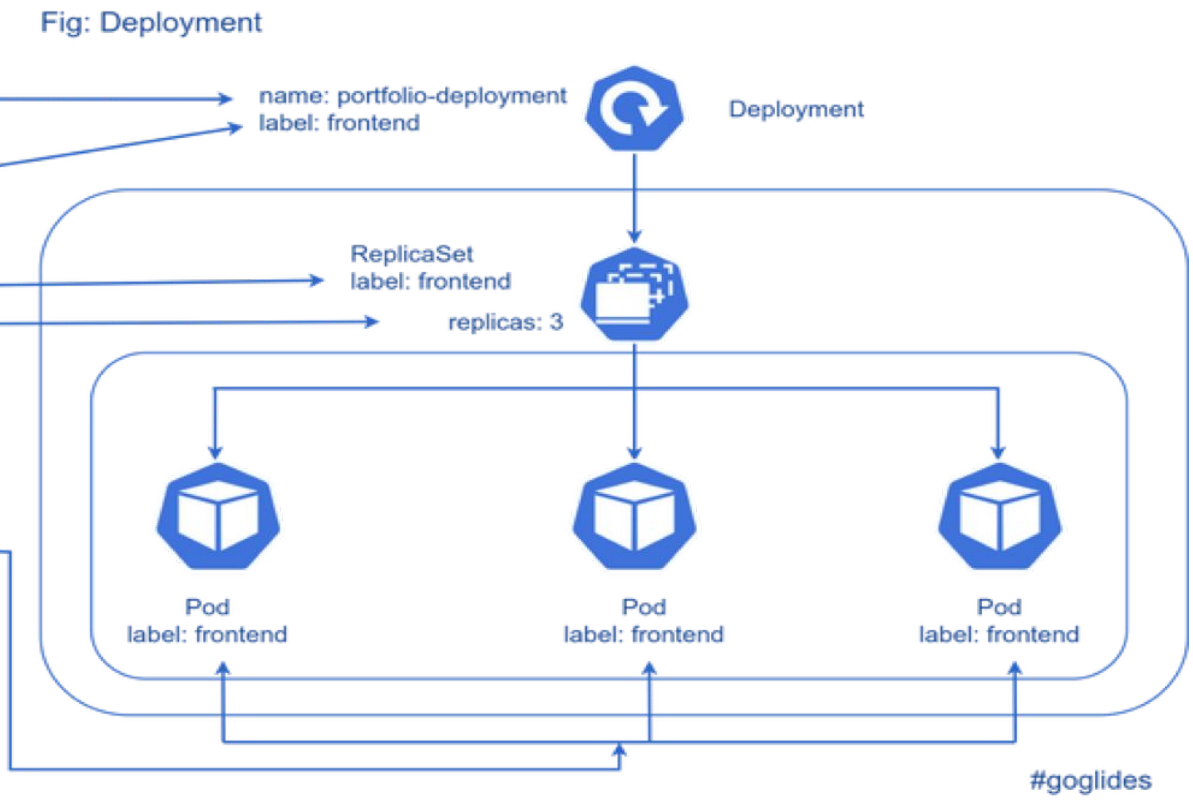
GitOps Architecture

Key requirements for Application Build and Deployment

1. Use any CI tool for build, we don’t care.
2. Produce the containerize docker image as final CI output.
3. We will not get into complexity of code quality checks. We already covered such concepts in Project # 1.
4. Make sure Argo CD is deployed inside the cluster.
5. Connect Argo CD with your GitHub repo deployment files.
6. Ensure container resource limits while deployment (CPU and memory requirements)
7. Ensure horizontal auto scalers are configured
8. All secrets to be managed properly using the default secrets management in Kubernetes.
9. Ensure that the application can withstand complete failure of worker nodes and still recover.
10. MySQL to be a microservice and use persistent storage for DB.
11. Ensure deployment to support rolling deployment strategy as well as recreate deployment strategy.
12. [Advanced use case: Support Canary deployment strategy]

How deployment looks like in Kubernetes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: portfolio-deployment
  labels:
    app: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: portfolio-container
        image: nirajpdn/react-app:v1
        ports:
        - containerPort: 80
```



ArgoCD installation Step by step

Step 1: Add the ArgoCD Helm Repository

```
helm repo add argo https://argoproj.github.io/argo-helm
```

```
helm repo update
```

Step2: Create a Namespace for ArgoCD

```
kubectl create namespace argocd
```

Step 3: Install ArgoCD Using Helm

```
helm install argocd argo/argo-cd --namespace argocd
```

Step 4: Verify the Installation

```
kubectl get pods -n argocd
```

Step 5: Expose the ArgoCD Server

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}' ## not in minikube case

kubectl get svc -n argocd argocd-server
```

Visit `https://<EXTERNAL-IP>`

Port-Forwarding (Temporary Local Access)

```
kubectl port-forward svc/argocd-server -n argocd 8080:443 ## in-case of minikube
```

Login to ArgoCD:

Retrieve the Initial Admin Password:

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

Username- admin

Pasword-
```

Create your First Application

```
argocd app create ingress-nginx \
  --repo https://github.com/your-username/your-repo.git \
  --path manifests \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace default \
  --sync-policy automated
```