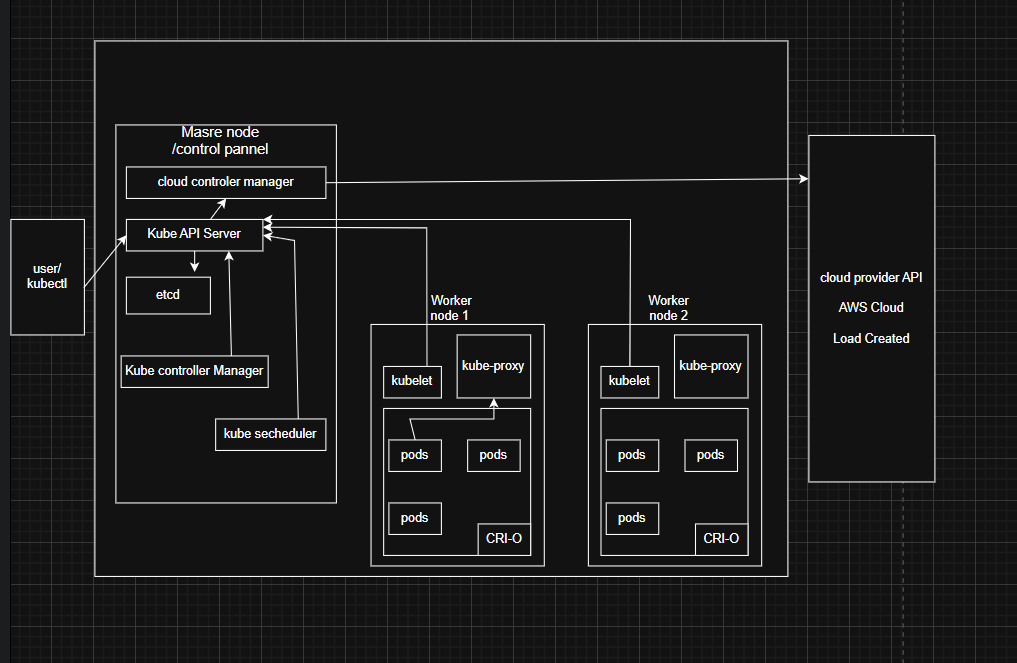
K8s document



**kube-apiserver :-**

The API server is a component of the Kubernetes [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

**Etcd :-**

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

**kube-scheduler:-**

Control plane component that watches for newly created [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) with no assigned [node](https://kubernetes.io/docs/concepts/architecture/nodes/), and selects a node for them to run on.

**kube-controller-manager :-**

Control plane component that runs [controller](https://kubernetes.io/docs/concepts/architecture/controller/) processes.

**cloud-controller-manager :-**

A Kubernetes [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

**Kubelet :-**

An agent that runs on each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in the cluster. It makes sure that [containers](https://kubernetes.io/docs/concepts/containers/) are running in a [Pod](https://kubernetes.io/docs/concepts/workloads/pods/).

**kube-proxy :-**

kube-proxy is a network proxy that runs on each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in your cluster, implementing part of the Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/) concept.

**Container runtime :-**

A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

1.PODS :-

*Pods* are the smallest deployable units of computing that you can create and manage in Kubernetes.A *Pod* is a group of one or more containers with shared storage and network resources, and a specification for how to run the containers.

Yaml:-

apiVersion: v1 #This tells Kubernetes which API version to use when interpreting this object

kind: Pod #Defines the type of Kubernetes object you want to create.

metadata: #metadata stores identifying information about the object.

name: nginx #normal naming

namespace : arjya #name space name

spec: #It describes how the Pod should run.

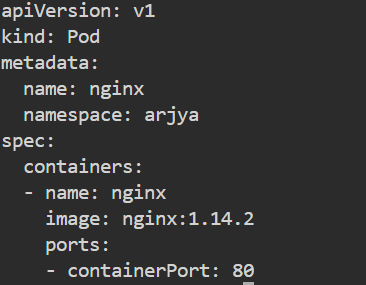
containers: #This field is a list, so you can define multiple containers inside the Pod

- name: nginx #Defines the name of the container inside the Pod.

image: nginx:1.14.2 #Specifies the container image to run

ports: #Lists the ports that this container expose

- containerPort: 80 #This container will listen on port 80

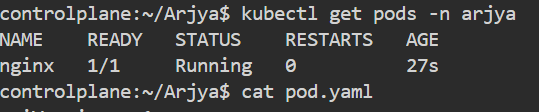


**Commands:-**

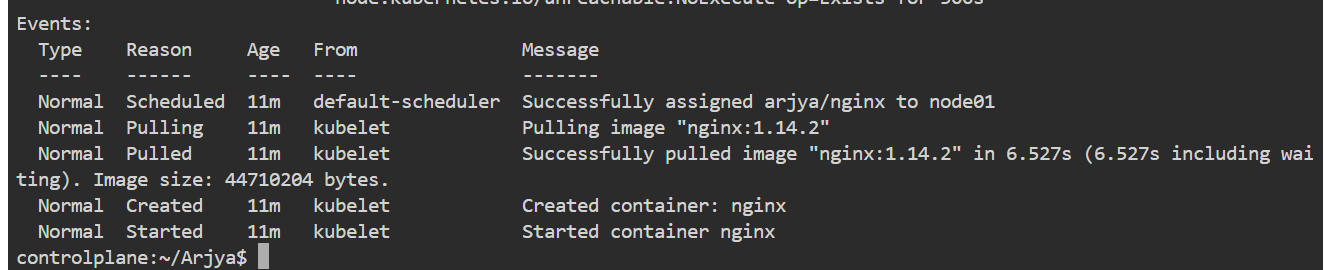
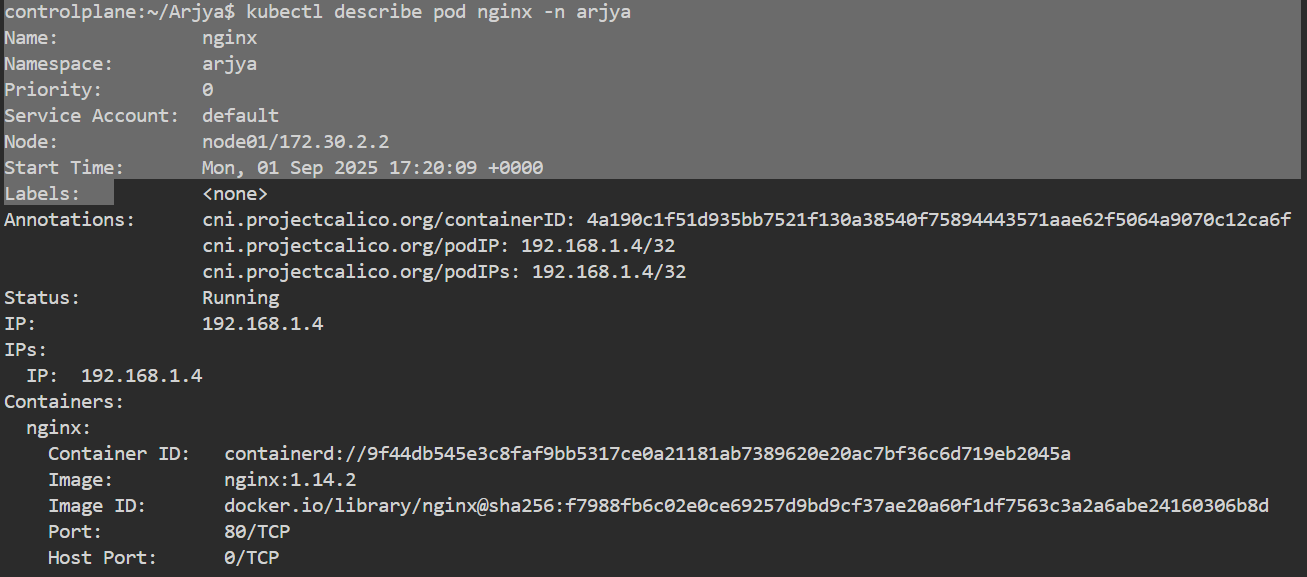
**1.Kubectl apply -f pod.yaml** : for creating pod



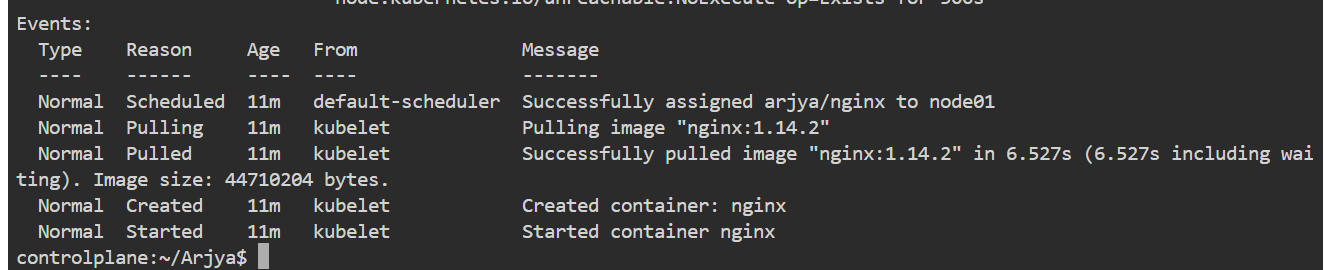
**2.kubectl get pods -n arjya** : for checking pods



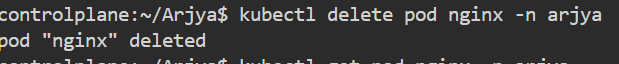
**3.kubectl get pod nginx -n arjya** : to get description about pod



4. **kubectl exec -it nginx -n arjya -- /bin/bash** :- Execute a command inside the Pod



5. **kubectl delete pod nginx -n arjya** : delete pod



2.Replicaset :-

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods

Yaml :-

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: Arjya-replica

  namespace: arjya

  labels: # key-value pairs attached to this ReplicaSet. They help identify/select resources.

    app: arjya-nginx-rs

spec: # spec defines the desired state.

  replicas: 3 # means the ReplicaSet will maintain 3 Pods

  selector: # Defines how the ReplicaSet knows which Pods belong to it.

    matchLabels:

      app: arjya-nginx-rs # This must match the labels in the Pod template below.

  template: # This pod templete Any new Pod created by this ReplicaSet will use this template.

    metadata:

      labels:

        app: arjya-nginx-rs

    spec:

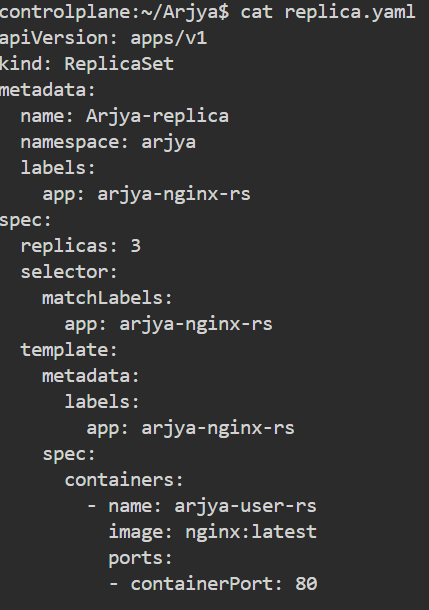
      containers:

        - name: arjya-user-rs

          image: nginx:latest

          ports:

          - containerPort: 80 # the Nginx container listens on port 80

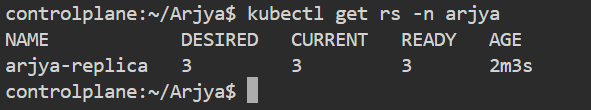


**Commands:-**

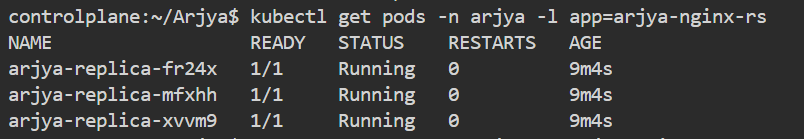
1. **kubectl apply -f replica.yaml :-**create replica



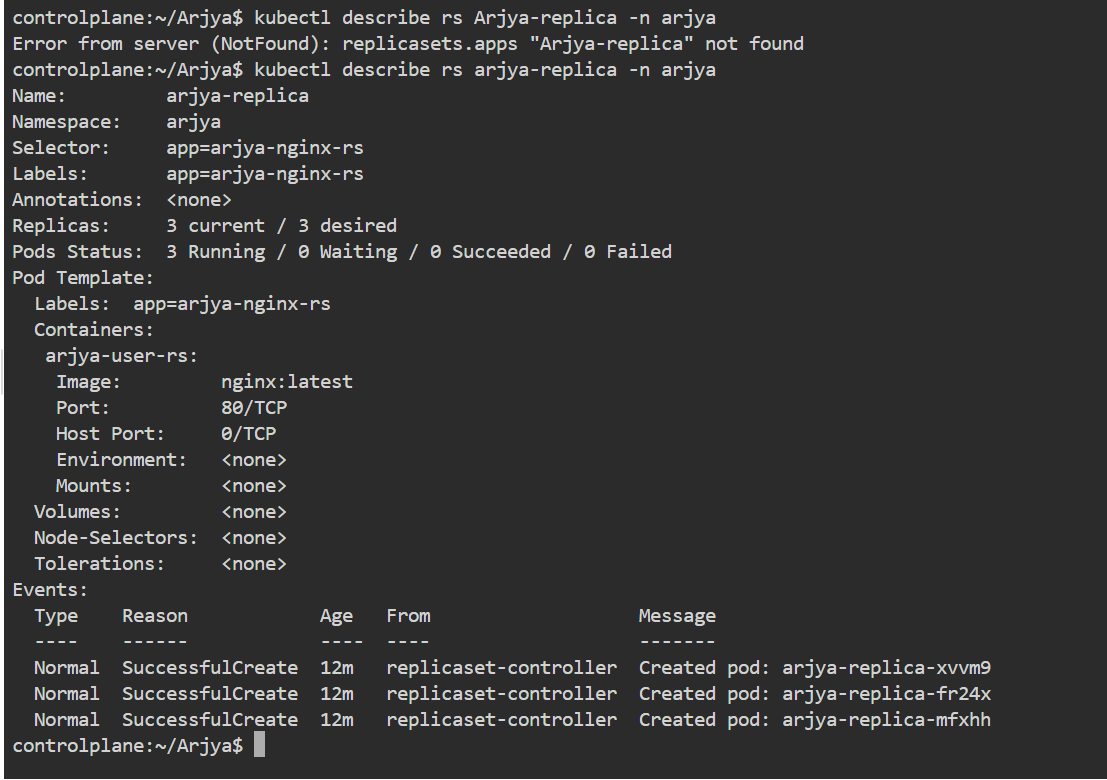
1. **kubectl get rs -n arjya:** check replicaset created or not



1. **ds -n arjya -l app=arjya-nginx-rs :** list all replicas

****

1. **kubectl describe rs Arjya-replica -n arjya :** describe replicaset

****

Deployment:-

A *Deployment* provides declarative updates for pods and replicaset. [Create a Deployment to rollout a ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment). The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.

Yaml :-

apiVersion: apps/v1

kind: Deployment

metadata:

  name: Arjya-dp

  namespace: arjya

  labels:

    app: arjya-nginx-dp

spec:

  replicas: 2

  selector:

    matchLabels:

      app: arjya-nginx-dp

  template:

    metadata:

      labels:

        app: arjya-nginx-dp

    spec:

      containers:

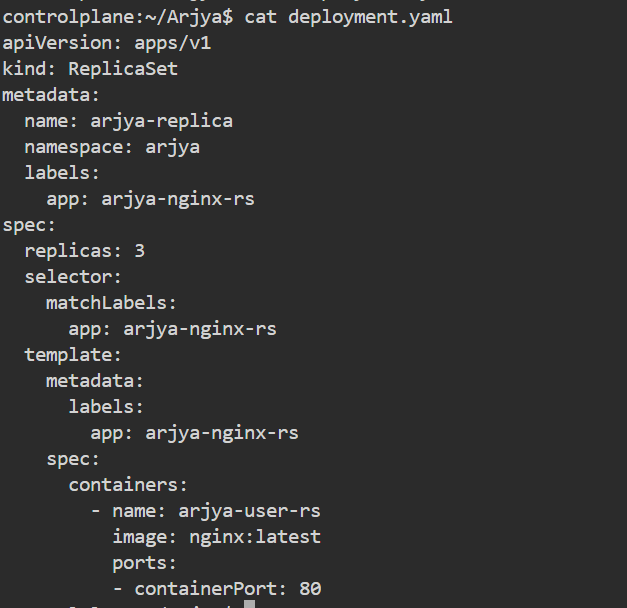
        - name: arjya-user-dp

          image: nginx:1.21.3

          ports:

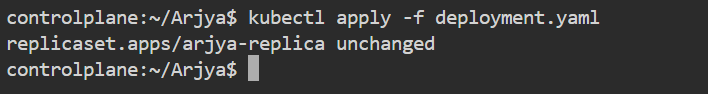
          - containerPort: 80

1.

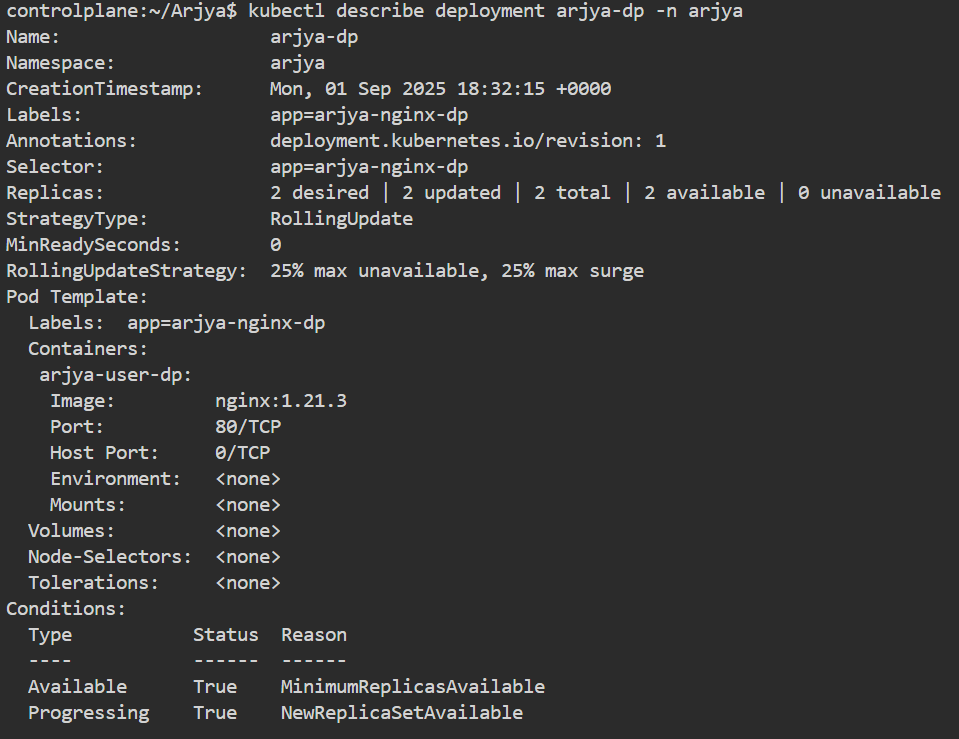


**Commands:-**

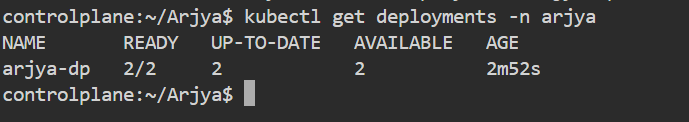
**1.ubectl apply -f deployment.yaml :- to create deployment**

****

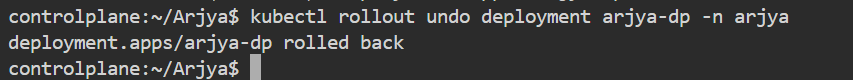
**2. kubectl describe deployment Arjya-dp -n arjya : describe deployment**

****

1. **kubectl get deployments -n arjya : to see deployment**

****

1. **kubectl rollout undo deployment arjya-dp -n arjya : rollout the task (image)**

****

Service:-

A Service in Kubernetes is an abstractionlayer that provides a stablenetworkendpoint to access a set of Pods.

Type:- ClusterIP, NodePort, LoadBalancer, ExternalName

Yaml :-

apiversion: v1

kind: Service

metadata:

name: arjya-service

namespace: arjya

spec:

selector:

app: arjya-app

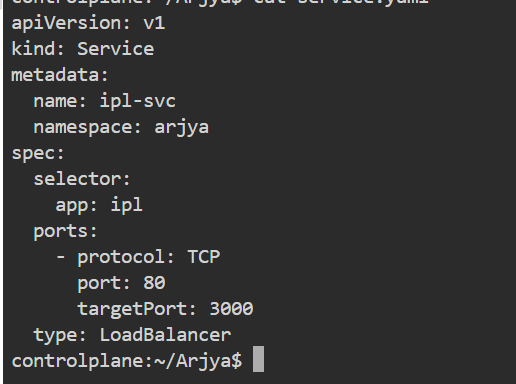
ports:

- protocol: TCP

port: 80

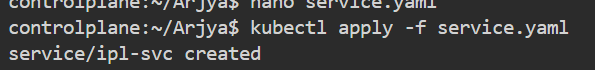
targetPort: 3000

type: LoadBalancer

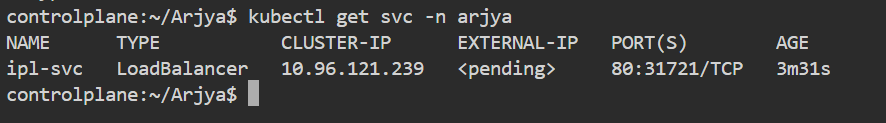


**Commands:-**

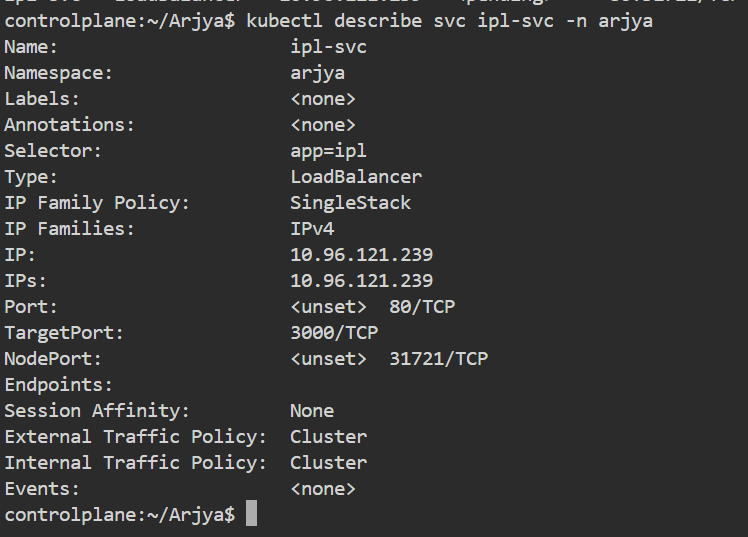
1. **kubectl apply -f service.yaml:** create sevice

****

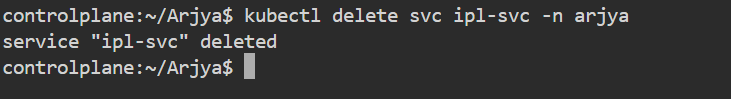
1. **kubectl get svc -n arjya :** check the service



1. **kubectl describe svc ipl-svc -n arjya**: describe service



1. **kubectl delete svc ipl-svc -n arjya :- delete service**

****

**ConfigMaps :-**

A ConfigMap is an API object used to store non-confidential data in key-value pairs.

**Yaml :-**

apiVersion: v1

kind: ConfigMap

metadata:

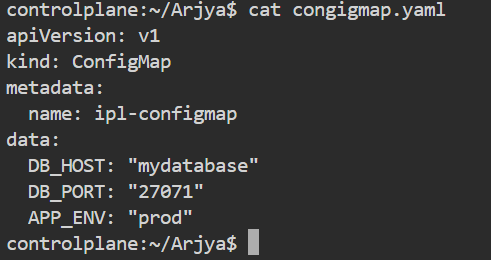
name: ipl-configmap

data:

DB\_HOST: "mydatabase" # Database host is set to mydatabase

DB\_PORT: "27071" # Database port is set to 27071.

APP\_ENV: "prod" # Environment is set to production.

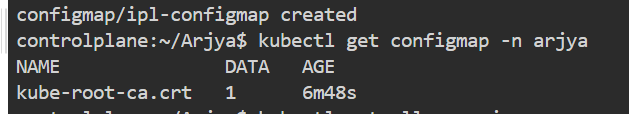


**Commands :-**

**1.kubectl apply -f congigmap.yaml** :-for creating configmap



**2.** **kubectl get configmap -n arjya :-** to check configmap created or not



3.