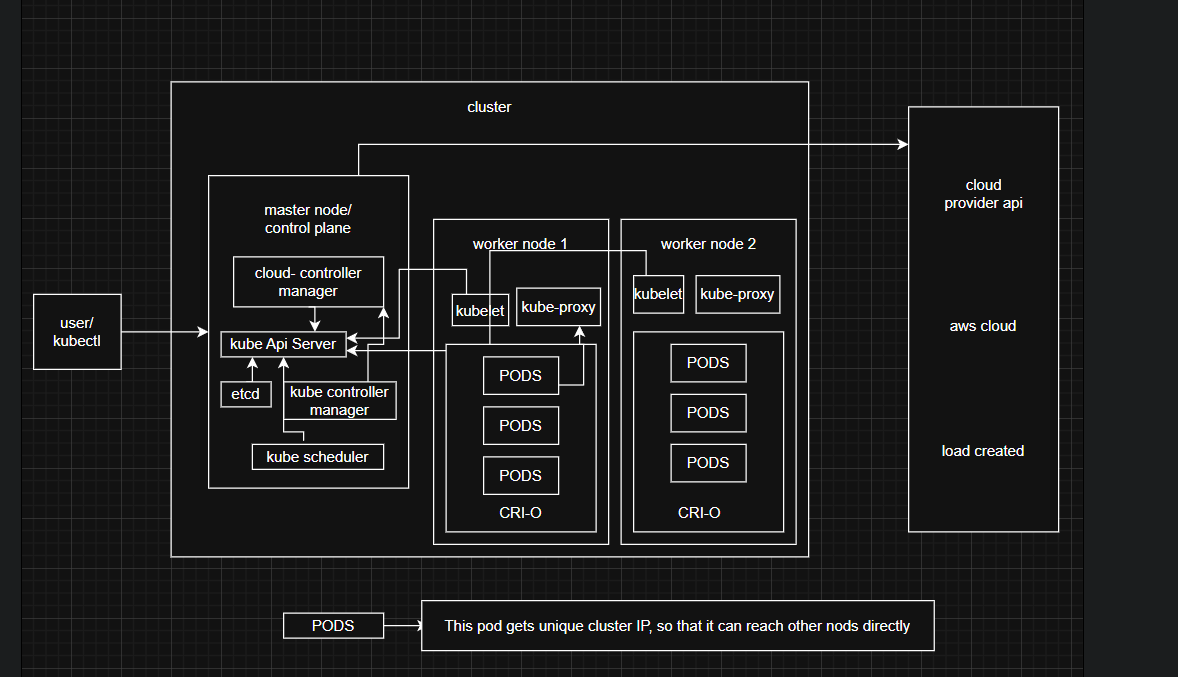
**\* K8s architecture:-**  
  


**1. Control Plane (Master Node):**

The control plane manages the overall cluster. It makes decisions about scheduling, scaling, and responding to events.  
It consists of:

**API Server:-**

- Acts as the entry point to Kubernetes.

- Exposes the Kubernetes API.

- All kubectl commands interact with it.

**Etcd:-**

- Key-value database.

- Stores cluster state and configuration.

- Highly available and consistent.

**Scheduler:-**

- Assigns Pods to nodes.

- Considers resource availability, policies, affinity, etc.

- Controller Manager

- Runs controllers that regulate the state of the cluster.

- Examples: Node Controller, Deployment Controller, ReplicaSet Controller.

**Cloud Controller Manager (optional):-**

- Integrates with cloud providers (AWS, GCP, Azure).

- Manages cloud-specific resources like load balancers, storage, etc.

**2. Worker Nodes (Data Plane):**

Nodes are machines (VMs or physical servers) where applications actually run.  
Each node has:

**Kubelet:-**

- Agent running on each node.

- Ensures Pods are running in containers.

- Communicates with the API server.

**Kube-Proxy:-**

- Handles networking for Pods.

- Manages routing and load-balancing.

**Container Runtime:-**

- Runs containers (Docker, containerd, CRI-O).

**\* pod:-**  
- **Smallest deployable unit** in Kubernetes, not the container itself.  
- Can contain **one or more containers** that run together.  
- Containers inside a Pod **share the same IP, network, and storage**.  
- Pods are **ephemeral** – if one dies, Kubernetes creates a new one (with a new IP).  
- Higher-level controllers (ReplicaSet, Deployment) manage Pods for **scaling & self-healing.  
  
Pod yaml file:-**  
apiVersion: v1 # API version used for defining the Pod object

kind: Pod # The type of Kubernetes object here it's a Pod

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-pod # Name of the Pod

  namespace: karan # Namespace in which the Pod will be created

  labels: # Key-value pairs used for identification, selection, or grouping

    app: karan-nginx # Here app name is defined

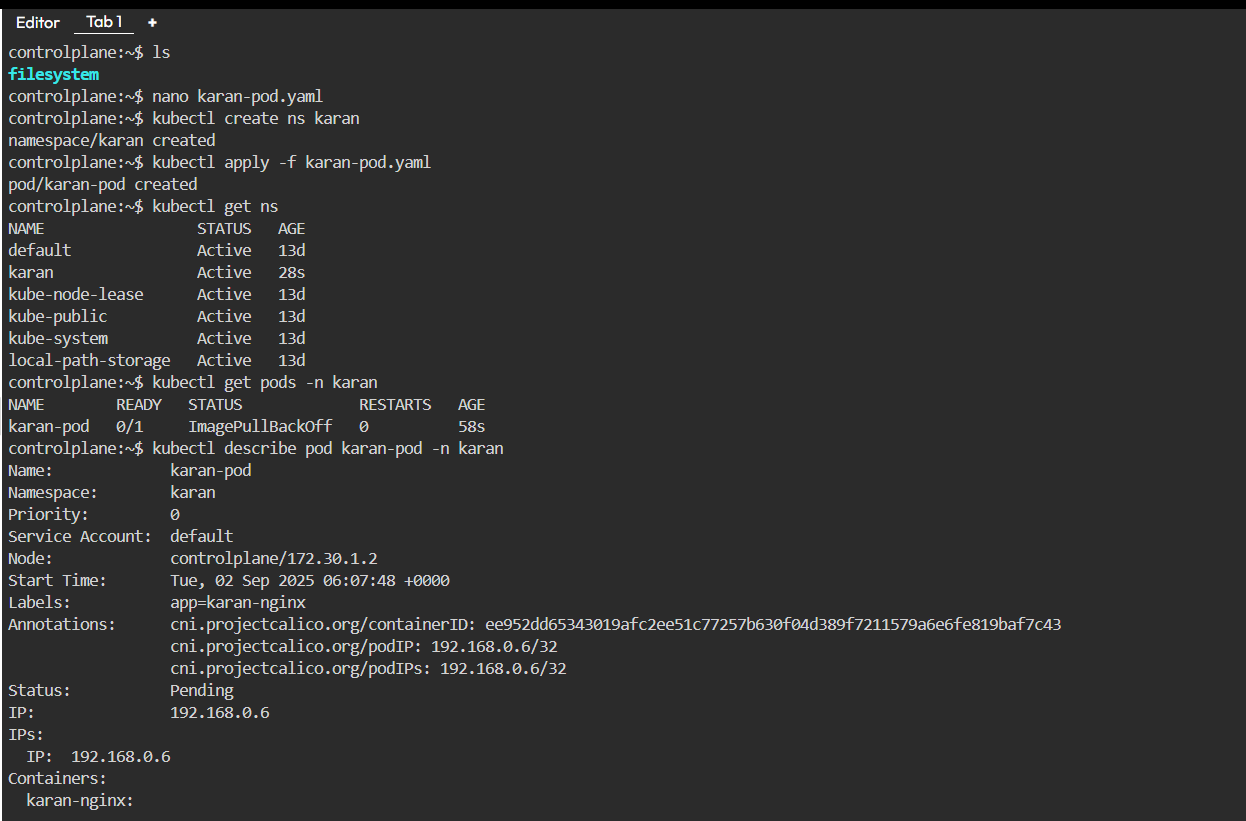
spec: # The specification of the desired state of the Pod

  containers: # List of containers in this Pod

    - name: karan-nginx # Name of the container

      image: niginx:latest # Docker image to run in the container

      ports: # List of ports that the container exposes

        - containerPort: 80 # The container will listen on port 80 **Commands:-**  
kubectl create namespace karan  
kubectl apply -f karan-pod.yaml  
kubectl get pods -n karan  
kubectl describe pod karan-pod -n karan   
  
**Image:-**  


**\* replicaset:-**  
- **Ensures a specified number of Pods** are running at all times (self-healing if Pods crash or nodes fail).  
- Uses a **label selector** to identify which Pods it manages.  
- Can **scale Pods up or down** by changing the replica count.  
- Usually not used directly—**Deployments manage ReplicaSets** for rollouts and rollbacks.  
- Provides **high availability** by spreading Pods across nodes.

**replicaset yaml file:-**  
apiVersion: apps/v1 # API version used for defining the ReplicaSet

kind: ReplicaSet # The type of Kubernetes object here it's a Replicaset

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-replica # Name of the Replica

  namespace: karan # Namespace in which the replica will be created

  labels: # Key-value pairs used for identification, selection, or grouping

    app: karan-nginx-rs # Here app name is defined

spec: # The specification of the desired state of the Replica

replicas: 3 # Number of Pod replicas to maintain

selector: #Defines how this ReplicaSet chooses which Pods to manage

matchLabels:

app: karan-nginx-rs #Only manage Pods that have this label

template: #Template for creating new Pods

metadata:

labels:

app: karan-nginx-rs # Must match the selector label

spec:

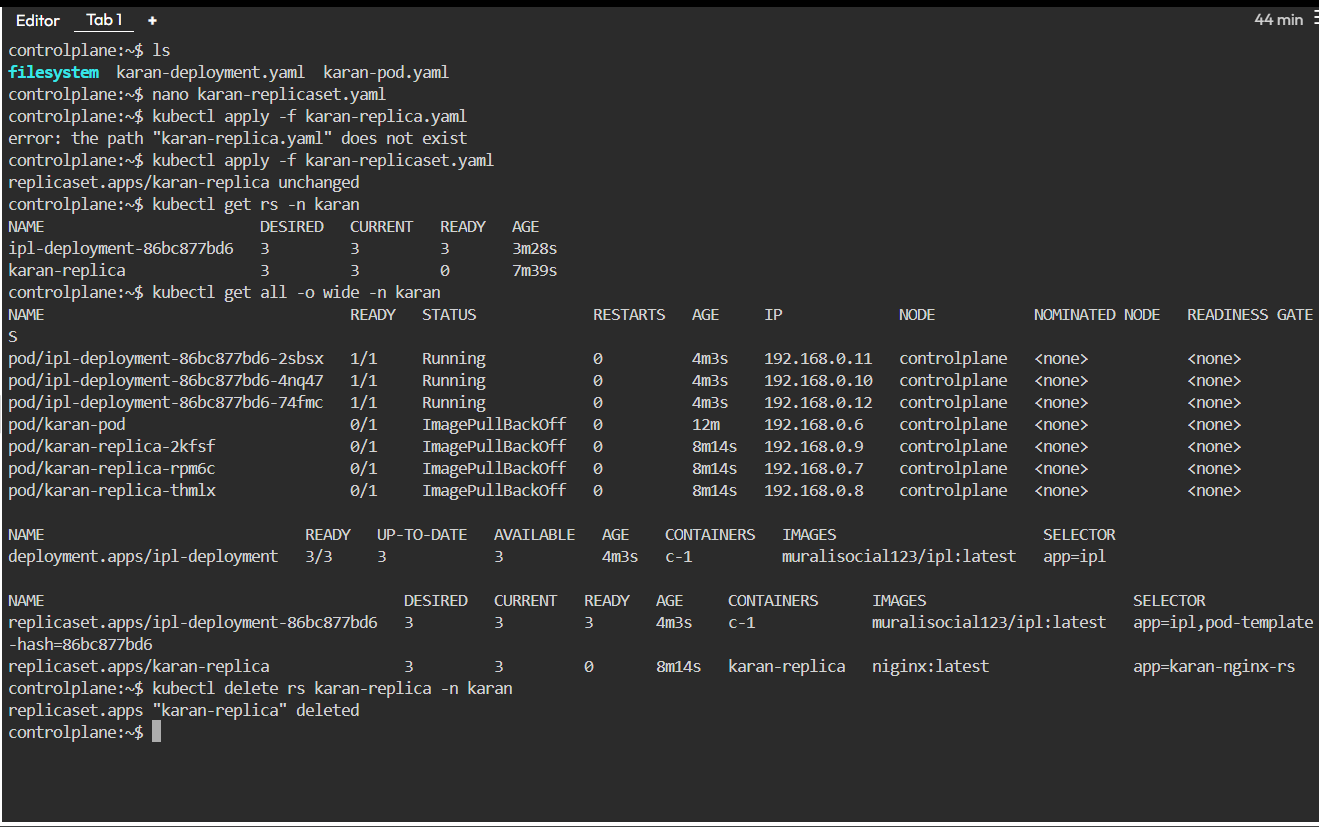
  containers: # List of containers in this replica

    - name: karan-replica # Name of the container

      image: niginx:latest # Docker image to run in the container

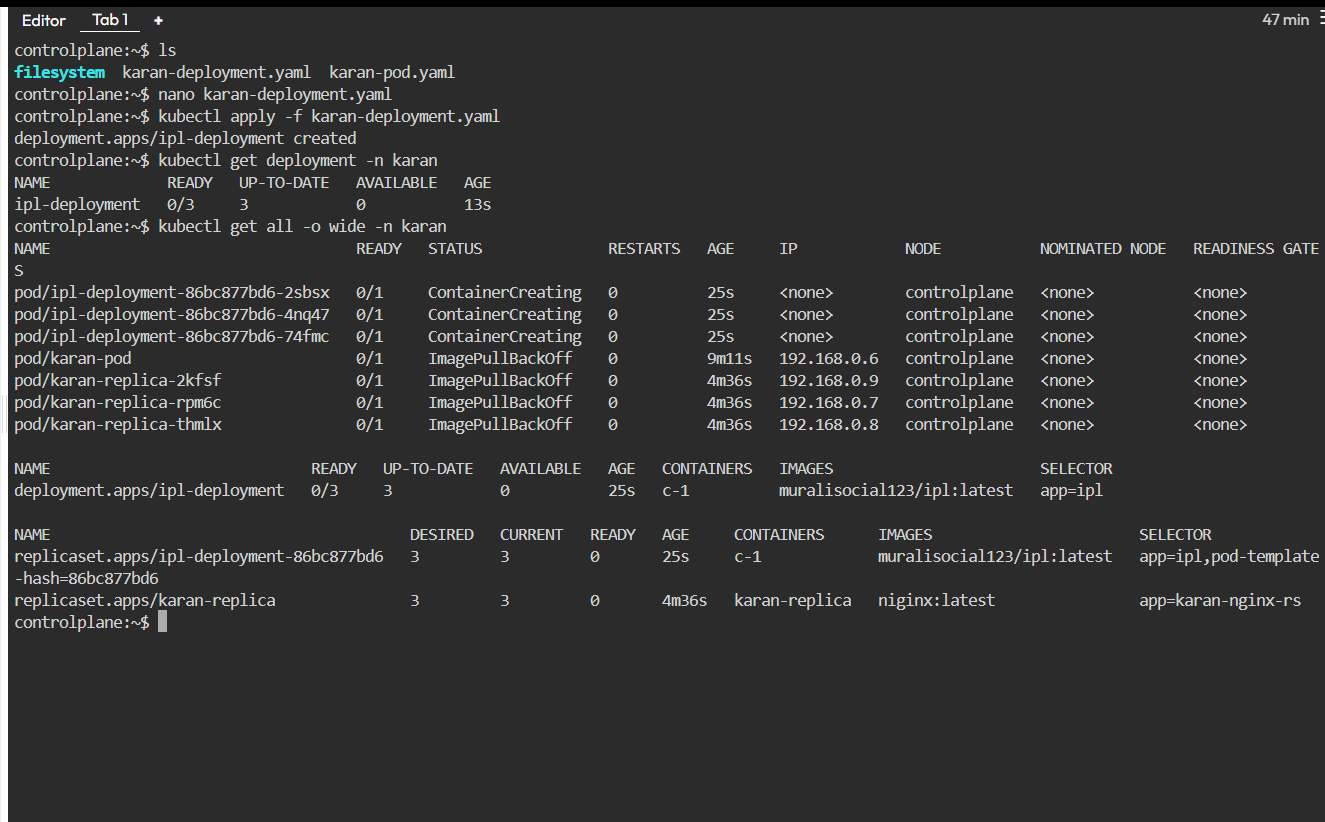
      ports: # List of ports that the container exposes

        - containerPort: 80 # The container will listen on port 80  
  
**Commands:-**  
vi karan-replica.yml  
kubectl apply -f karan-replica.yml  
kubectl get rs -n karan  
kubectl get all -o wide -n karan  
kubectl delete rs karan-replica -n karan

**Images:-**  


**\* deployment:-**  
- **Manages ReplicaSets**, which in turn manage Pods — adds an extra layer of control.  
- Supports **rolling updates** and **rollbacks** for zero-downtime app upgrades.  
- Provides **scaling** (manual or autoscaling) of Pods easily.  
- Ensures **desired state management** — if Pods/ReplicaSets drift, Deployment fixes them.  
- Commonly used for **stateless applications** (web servers, APIs, etc.).  
  
**deployment yaml file:-**  
apiVersion: apps/v1 # API version used for defining the Deploying  
kind: Deployment # The type of Kubernetes object here it's a Deployment  
metadata: # Metadata about the object (name, namespace, labels)   
  name: ipl-deployment # Name of Deployment  
  namespace: karan # Namespace in which the Deployment will be created  
spec: # The specification of the desired state of the Deployment  
 replicas: 3 # Number of Pod replicas to maintain  
 selector: # How to select which Pods this Deployment will manage  
 matchLabels:  
 app: ipl # Matches Pods with this label  
 template: #Template for creating new Pods  
 metadata:  
 labels:  
 app: ipl # Must match the selector label  
 spec:   
  containers: # List of containers in this replica   
    - name: c-1 # Name of the container  
      image: muralisocial123/ipl:latest # Docker image to run in the container  
      ports: # List of ports that the container exposes  
        - containerPort: 80 # The container will listen on port 80

**Commands:-**  
kubectl create ns karan  
kubectl apply -f karan-deployment.yml  
kubectl get deployment -n karan  
kubectl get all -o wide -n karan  
kubectl scale deployment ipl-deployment --replicas=5 -n karan

**Images:-**  


**\* Services:-**  
- **Provide stable networking** to Pods (since Pod IPs are ephemeral).  
- Use **label selectors** to route traffic to the right Pods.  
- Support **4 main types**: ClusterIP (default), NodePort, LoadBalancer, ExternalName.  
- Offer **service discovery** DNS names(e.g., svc name.namespace.svc.cluster.local).  
- Can handle **load balancing** across multiple Pod replicas.  
  
**there are 4 type of services:-**  
- ClusterIp  
- NodePort  
- LoadBalancer  
- ExternalName  
  
**1) ClusterIp yaml file:-**  
apiVersion: v1 # API version used for defining the Service

kind: Service # The type of Kubernetes object here it's a Service

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-service # Name of Service

namespace: karan # Namespace in which the Service will be created

spec: # The specification of the desired state of the Service

  selector: #Selects the Pods this Service will send traffic to

    app: ipl #Must match the label in your Deployment.

  ports: # List of ports that the container exposes

    - protocol: TCP #Communication protocol

      port: 80

      targetPort: 3000 #Port on the container inside the Pod

  type: ClusterIP #Internal access within the cluster only

**2) NodePort yaml file:-**apiVersion: v1 # API version used for defining the Service

kind: Service # The type of Kubernetes object here it's a Service

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-service # Name of Service

namespace: karan # Namespace in which the Service will be created

spec: # The specification of the desired state of the Service

  selector: #Selects the Pods this Service will send traffic to

    app: ipl #Must match the label in your Deployment.

  ports: # List of ports that the container exposes

    - protocol: TCP #Communication protocol

      port: 80

      targetPort: 3000 #Port on the container inside the Pod

nodePort: 30080 # Fixed external port exposed on each Node

type: NodePort # Exposes the app externally on each Node IP

**3) LoadBalancer yaml file:-**  
apiVersion: v1 # API version used for defining the Service

kind: Service # The type of Kubernetes object here it's a Service

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-service # Name of Service

namespace: karan # Namespace in which the Service will be created

spec: # The specification of the desired state of the Service

  selector: #Selects the Pods this Service will send traffic to

    app: ipl #Must match the label in your Deployment.

  ports: # List of ports that the container exposes

    - protocol: TCP #Communication protocol

      port: 80

      targetPort: 3000 #Port on the container inside the Pod

type: LoadBalancer # Exposes the app outside the cluster  
  
**4) ExternalName yaml file:-**   
apiVersion: v1 # API version used for defining the Service

kind: Service # The type of Kubernetes object here it's a Service

metadata: # Metadata about the object (name, namespace, labels)

  name: karan-service # Name of Service

namespace: karan # Namespace in which the Service will be created

spec: # The specification of the desired state of the Service

  selector: #Selects the Pods this Service will send traffic to

    app: ipl #Must match the label in your Deployment.

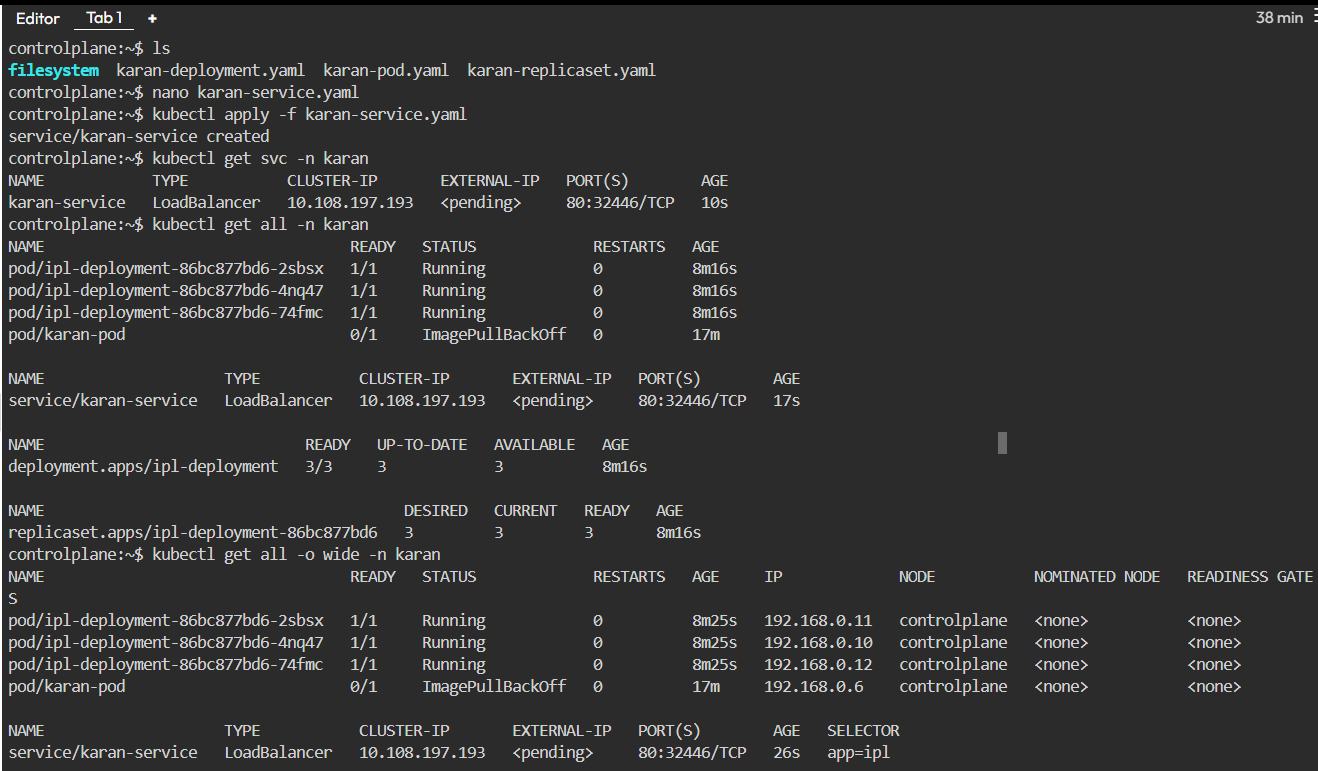
  ports: # List of ports that the container exposes

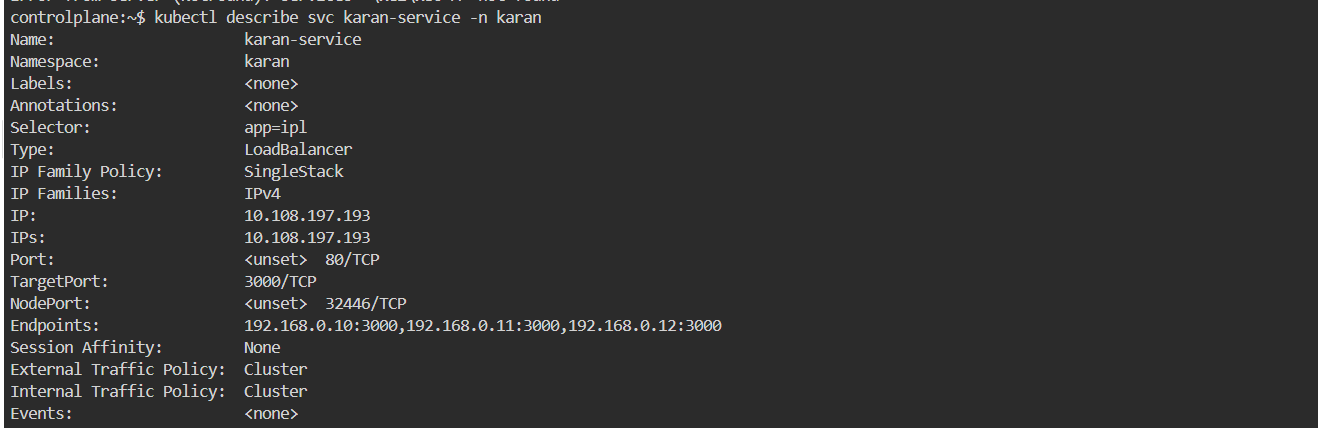
    - protocol: TCP #Communication protocol

      port: 80

     externalName: flipkart.com, amazon.com, etc # External DNS name

type: ExternalName # Maps service to external DNS name  
  
**commands:-**vi karan-service.yml  
kubectl apply -f karan-service.yaml  
kubectl get svc -n karan  
kubectl get all -n karan  
kubectl get all -o wide -n karan  
kubectl get svc –A  
kubectl describe svc karan-service -n karan  
kubectl delete svc karan-service –n karan

**Images:-**

****