KUBERNETES SERVICE

Problem-1: Pods are ephemeral in nature — their IP addresses change whenever a pod restarts or is recreated. Therefore, accessing a pod directly using its IP address is not reliable in the long run.

Problem-2: Multiple users may try to access the application simultaneously, but there is no such mechanism to distribute this traffic across pods by default.

Problem-3: Pods running in K8s cluster are accessible only to those who have access to Kubernetes cluster.

Kubernetes service can resolve this issues:

→ Kubernetes Services use **labels** to track pods, enabling them to route traffic even if pod IPs change — a process known as **Service Discovery**.

→ Kubernetes Services also provide **load balancing**, distributing incoming traffic evenly across all healthy pods to ensure reliability and high availability.

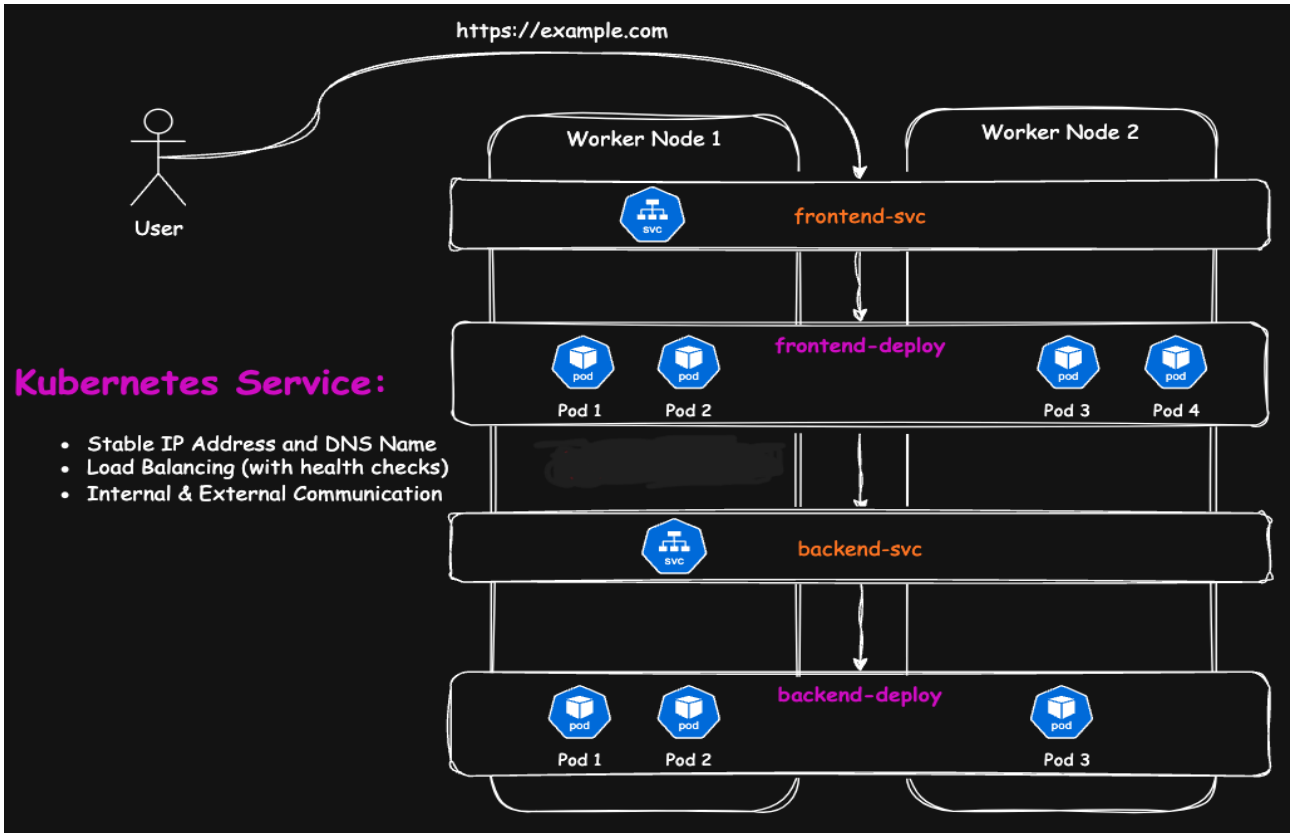
→ Kubernetes service exposes the application to the external world.

In the Below diagram, the user's request goes to the **frontend Service**, which routes it to a healthy frontend pod. The frontend pod forwards the request to the **backend Service**, which sends it to a healthy backend pod. The backend responds to the frontend, allowing the user to interact with the application.

There are 3 types of services:

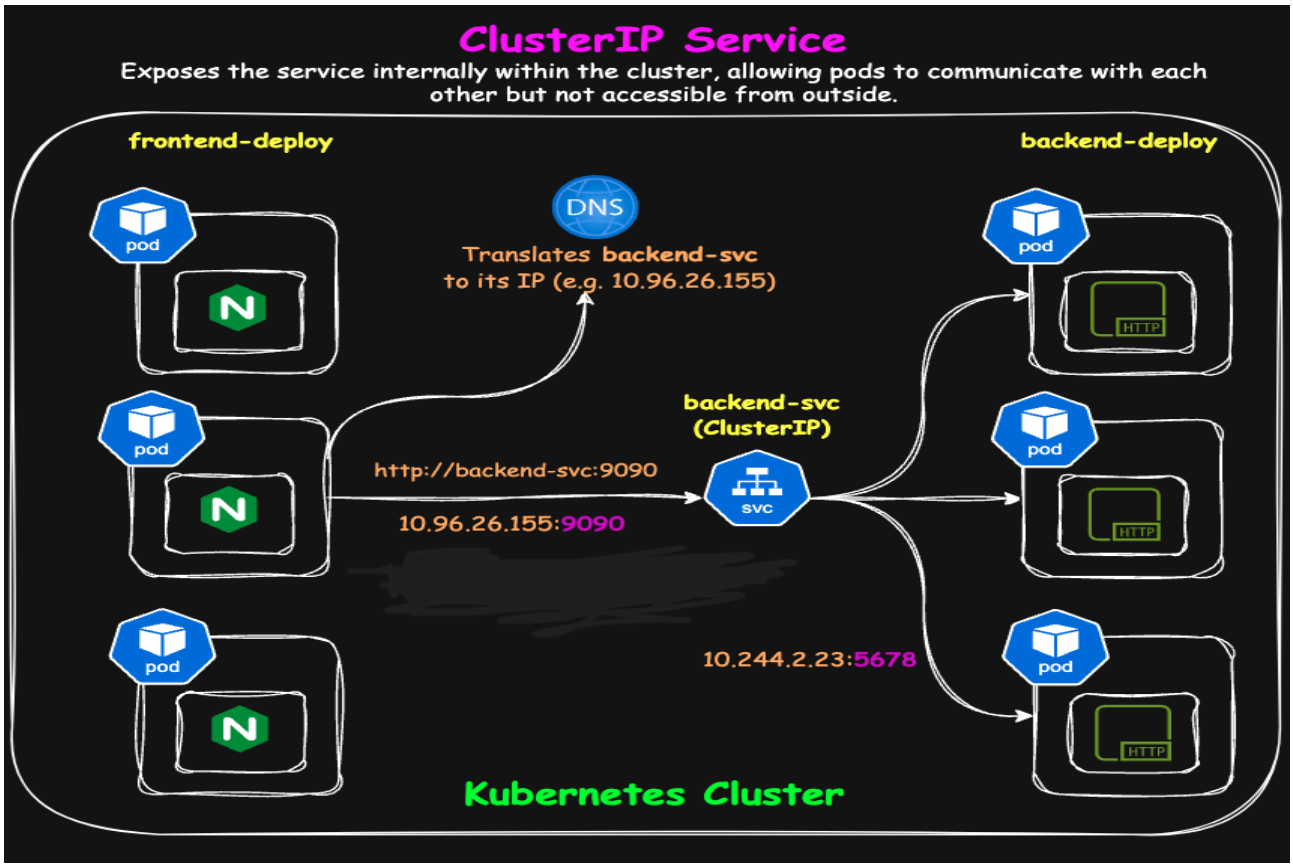
1) Service of ClusterIP Type
2) Servcice of NodePort Type
3) Service of Loadbalancer Type.

Haseeb Ullah



Types of services:

1) ClusterIP Service:



Haseeb Ullah

→ A ClusterIP Service is the default Kubernetes Service type that exposes applications internally within the cluster, enabling pod-to-pod communication via an internal IP.

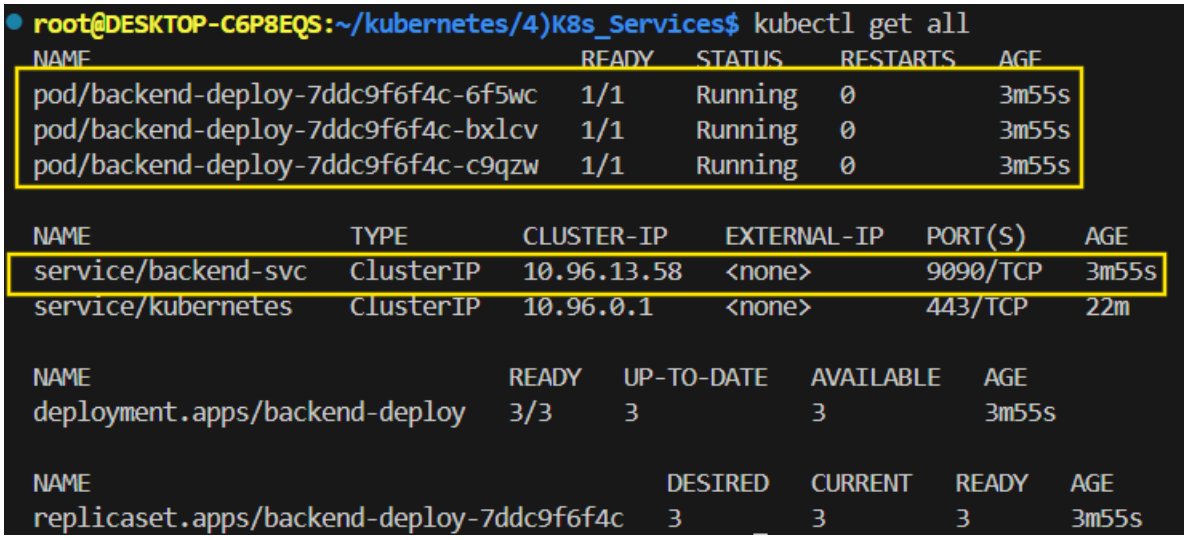
backend.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-container
          image: hashicorp/http-echo
          args:
            - "-text=Hello From Backend"

---
apiVersion: v1
kind: Service
metadata:
  name: backend-svc
spec:
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 9090
      targetPort: 5678
  selector:
```

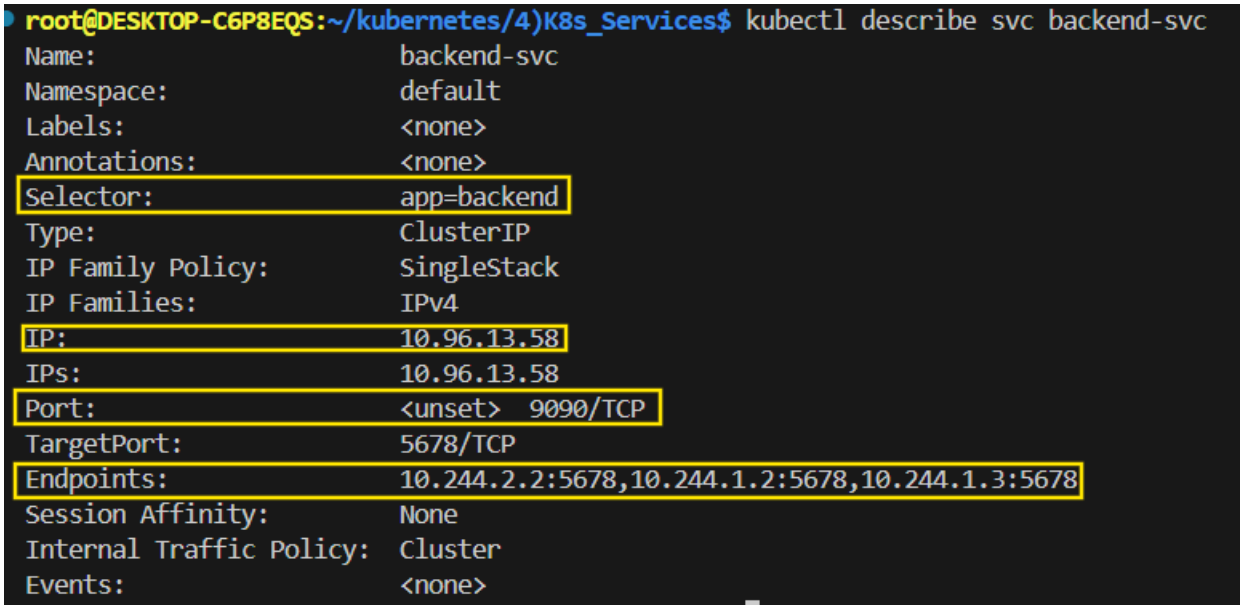
```
    app: backend
```

Lets apply this backend.yaml and explore the ClusterIP service:

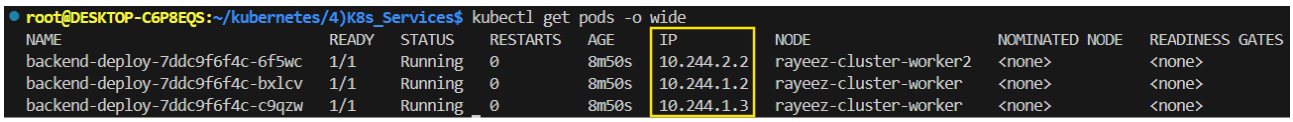
```
kubectl apply -f backend.yaml
```




```
kubectl describe svc backend-svc
```




```
kubectl get pods -o wide
```



Let’s create a test pod and try accessing the backend pods from inside it after logging in.

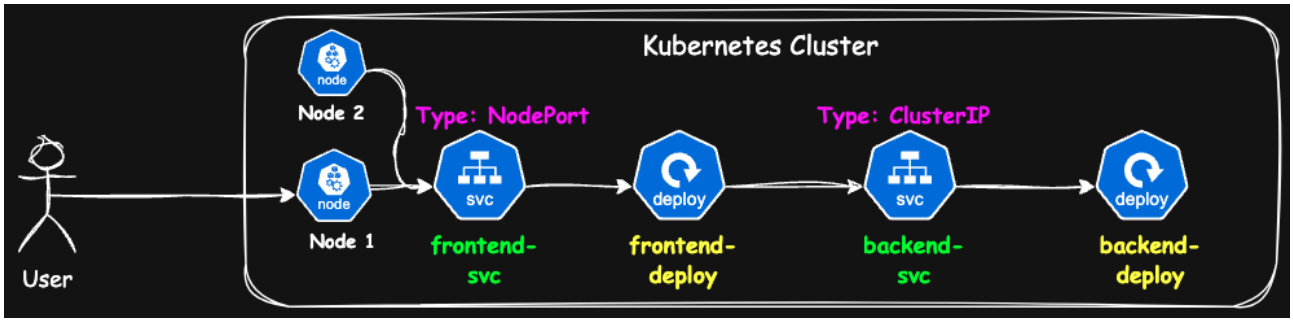
```
kubectl run -it mynginx --image=nginx --
/bin/bash
```

Pods were accessibe through service:

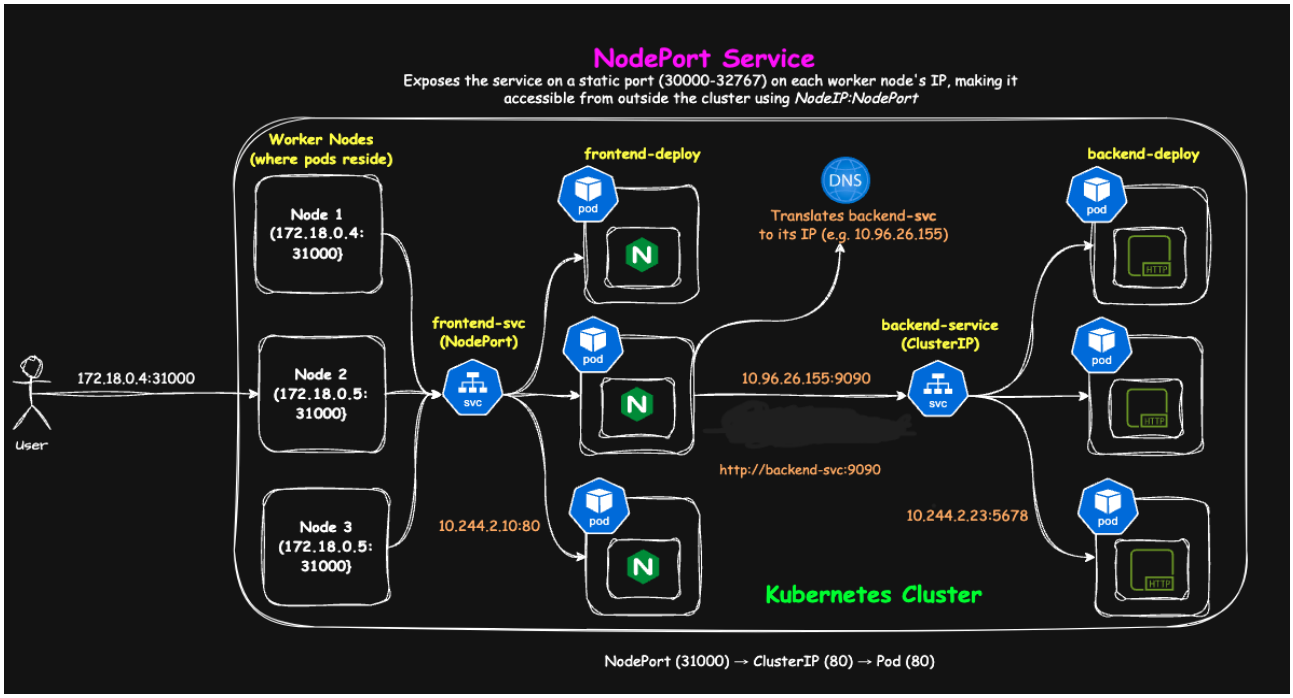
```
curl backend-svc:9090
```

```
root@mynginx:/# curl backend-svc:9090
Hello From Backend
```

2)NodePort Service: It is build on top of ClusterIP Service.



→ A **NodePort** Service exposes pods externally using a node's IP and a fixed port, unlike **ClusterIP**, which is accessible only within the cluster.



→ The user accesses the app via a node's IP and port. The **frontend Service** (**NodePort**) routes the request to the **frontend Deployment**,

which then forwards it to the **backend Service** (**ClusterIP**) and finally to the **backend Deployment**.

Note: When a **NodePort** Service is created, a corresponding **ClusterIP** Service is automatically created.

frontend.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend-container
          image: nginx
          ports:
            - containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - protocol: TCP
```

```
      port: 80          # ClusterIP service port
      targetPort: 80    # Container's port
inside the pod
      nodePort: 31000   # Exposed externally
(must be in 30000-32767)
```

Lets apply frontend.yaml file and explore NodePort service:

```
kubectl apply -f frontend.yaml
```

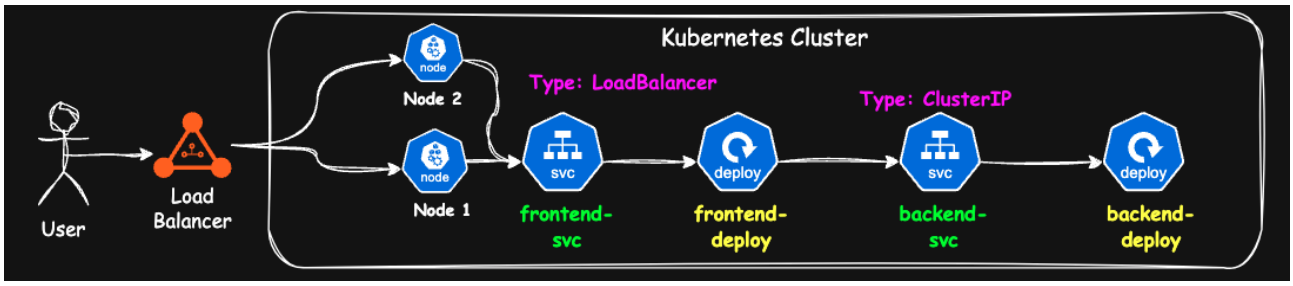
```
root@DESKTOP-C6P8EQS:~/kubernetes/4)K8s_Services$ kubectl get all
NAME                                    READY   STATUS    RESTARTS      AGE
pod/backend-deploy-7ddc9f6f4c-6f5wc     1/1     Running   0             96m
pod/backend-deploy-7ddc9f6f4c-bxlcv     1/1     Running   0             96m
pod/backend-deploy-7ddc9f6f4c-c9qzw     1/1     Running   0             96m
pod/frontend-deploy-578b67fb7b-599hb    1/1     Running   0             11m
pod/frontend-deploy-578b67fb7b-ds88n    1/1     Running   0             11m
pod/frontend-deploy-578b67fb7b-rl7l2    1/1     Running   0             11m
pod/mynginx                             1/1     Running   1 (77m ago)   84m

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/backend-svc     ClusterIP   10.96.13.58     <none>        9090/TCP       96m
service/frontend-svc    NodePort    10.96.253.233   <none>        80:31000/TCP   6m52s
service/kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        115m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend-deploy      3/3     3            3           96m
deployment.apps/frontend-deploy     3/3     3            3           14m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-deploy-7ddc9f6f4c      3         3         3       96m
replicaset.apps/frontend-deploy-578b67fb7b     3         3         3       11m
replicaset.apps/frontend-deploy-67447bb9ff     0         0         0       14m
```

→ For frontend deployment, frontend service of type ClusterIP is listening on port 80 and service type NodePort is listening on 31000.

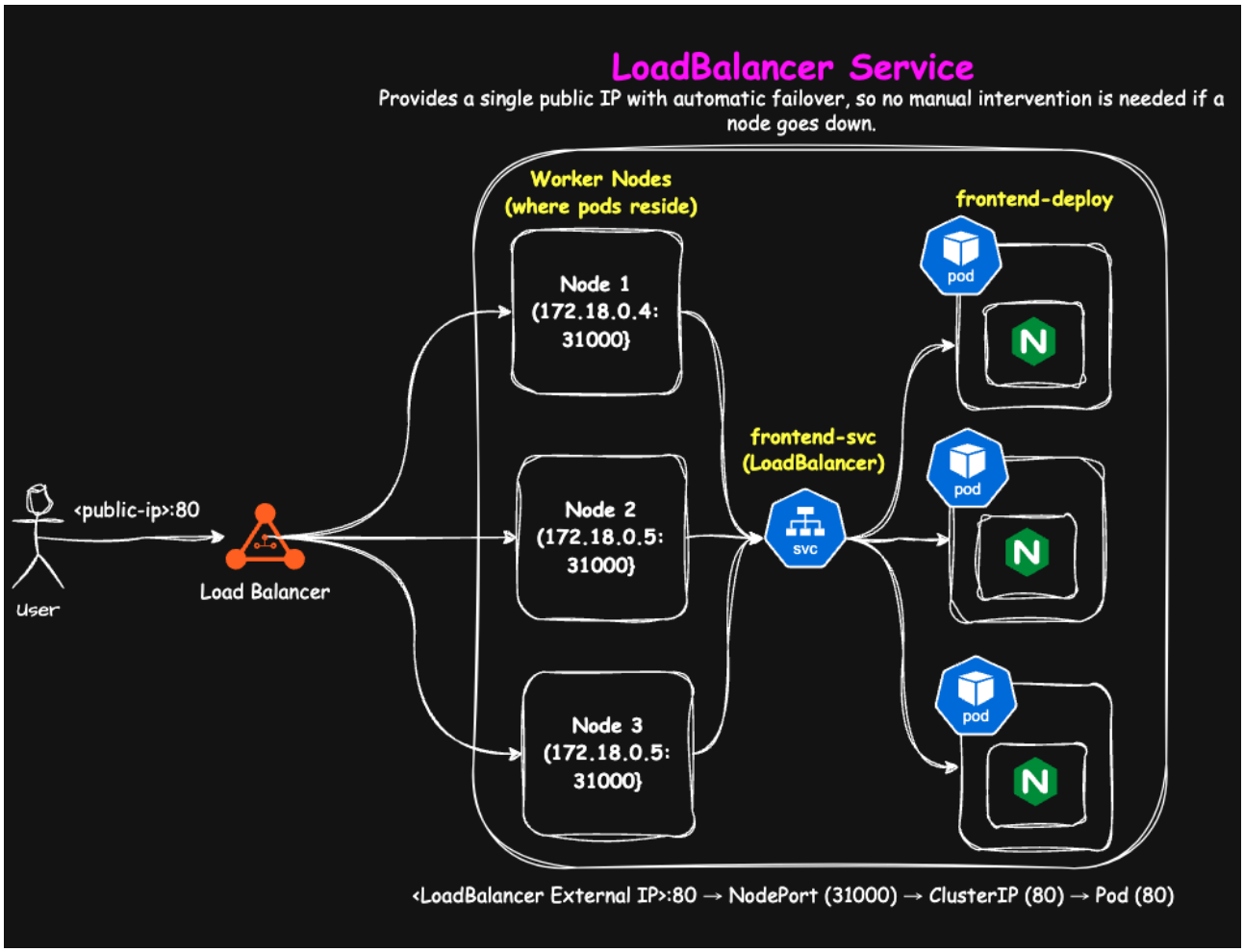
Lets Access the frontend app from browser:

```
http://<NodeIP>:<NodePort>
```

3)Load Balancer Service: It is build on top of NodePort service and ClusterIP service.



Remembering the IP addresses of each node is difficult. So when a **LoadBalancer** service is implemented, it automatically provisions a load balancer on the cloud platform, allowing the application to be accessible from the internet (external world).



→ When you create a **LoadBalancer service**, Kubernetes requests a **public IP** from the **cloud provider** (e.g., AWS, GCP, Azure).

→ This **public IP** is linked to the **LoadBalancer**, which then **routes traffic** to a **NodePort service** within the cluster.

→ The **NodePort service** internally uses a **ClusterIP service** to **distribute traffic** to the **pods**.

Service.yaml for **LoadBalancer** mode:

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80          # ClusterIP service port
      targetPort: 80    # Container's port
inside the pod
      nodePort: 31000   # Exposed externally
(must be in 30000-32767)
```

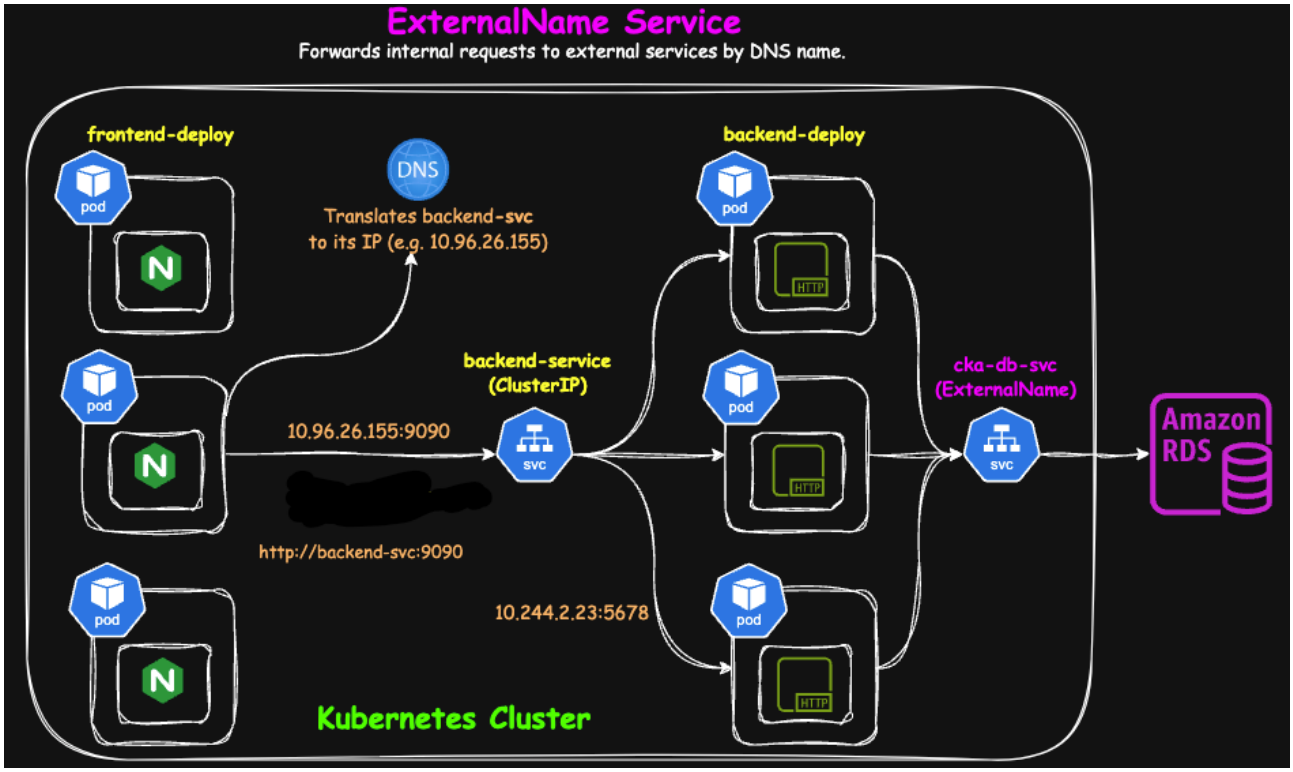
```
root@DESKTOP-C6P8EQS:~/kubernetes/4)K8s_Services$ kubectl get svc frontend-svc
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
frontend-svc   LoadBalancer   10.96.253.233   <pending>     80:31000/TCP   40m
```

If it is executed on a cloud platform, It will create a Loadbalancer having public IP address with the help of cloud control manager.

```
kubectl describe svc frontend-svc
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/4)K8s_Services$ kubectl describe svc frontend-svc
Name:                     frontend-svc
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=frontend
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.96.253.233
IPs:                      10.96.253.233
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31000/TCP
Endpoints:                10.244.1.6:80,10.244.1.5:80,10.244.2.6:80
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

4)ExternalName Service:



→ Offers a **simple alias** to connect **internal services to external resources** using **DNS names**.

→ When using **ExternalName**, applications **connect to services using internal DNS names**, and **Kubernetes handles the external redirection.**

→ This **separates configuration from application logic**, ensuring **easier maintenance** and **flexibility**.

Service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: cka-db-svc
spec:
  type: ExternalName
  externalName: cka-db-khjukjij.us-east-
1.rds.amazonaws.com #DNS name of the Amazon RDS
Instance
```