Services

- A Service in Kubernetes abstracts and exposes an application running on a set of pods, enabling network access.
- It provides stable IP addresses, load balancing, and service discovery, ensuring consistent and reliable communication.
- Services can be of various types, such as ClusterIP, NodePort, LoadBalancer, and ExternalName.

Key types of services include:

- **ClusterIP:** Exposes the service on an internal IP within the cluster, accessible only within the cluster.
- **NodePort:** Exposes the service on each Node's IP at a static port, making it accessible from outside the cluster.
- **LoadBalancer:** Exposes the service externally using a cloud provider's load balancer.
- **ExternalName:** Maps the service to a DNS name, allowing access to external services by a consistent name.

Cluster IP to inside the node (pod to pod)

- ClusterIP in Kubernetes enables communication between pods within the same cluster by providing an internal IP address.
- This allows pods to access each other using a stable IP and DNS name, facilitating seamless internal service discovery and communication.
- It ensures that services are accessible only within the cluster.

Step:1

Create the Vi cluster.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new1deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: httpd-container
        image: httpd
        ports:
          - containerPort: 80

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myservice
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

Step:2

To verify the pods

```
controlplane $ kubectl create -f cluster.yaml
deployment.apps/new1deploy created
service/myservice created
```

Step: 3

To check the status kubectl describe service/myservice

```
controlplane $ kubectl describe service/myservice
Name:              myservice
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=my-app
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.104.224.252
IPs:               10.104.224.252
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.4:80
Session Affinity:  None
Events:            <none>
```

Step:4

Going to port forward methods

```
controlplane $ kubectl port-forward services/myservice 80
Forwarding from 127.0.0.1:80 -> 80
Forwarding from [::1]:80 -> 80
```

Step:5

To verify the services

```
^Ccontrolplane $ kubectl describe service
Name:              kubernetes
Namespace:         default
Labels:            component=apiserver
                   provider=kubernetes
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.96.0.1
IPs:               10.96.0.1
Port:              https  443/TCP
TargetPort:        6443/TCP
Endpoints:         172.30.1.2:6443
Session Affinity:  None
Events:            <none>
```

```
Name:              myservice
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=my-app
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.104.224.252
IPs:               10.104.224.252
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.4:80
Session Affinity:  None
Events:            <none>
```

Step:5

To verify and check the command is curl http:// and ipaddress

```
controlplane $ curl http://192.168.1.4
<html><body><h1>It works!</h1></body></html>
```

Nodeport Methods

- NodePort in Kubernetes exposes a service on a specific port on each node in the cluster, making it accessible externally via NodeIP:NodePort.
- It enables direct access to the service from outside the cluster by routing traffic to the appropriate pods.
- NodePort is often used in conjunction with a load balancer or ingress controller for better traffic management.

Step: 1

Create the Yaml file

vi Nodeport.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: newd2deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: httpd-container
        image: httpd
        ports:
          - containerPort: 80
---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nodeport
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30100
```

Step: 2

To Create the Pods

```
controlplane $ kubectl create -f Nodeport.yaml
deployment.apps/newd2deploy created
service/nodeport created
```

Step: 3

To check the status kubectl describe service/nodeport

```
controlplane $ kubectl describe service/nodeport
Name:                     nodeport
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=my-app
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.103.87.246
IPs:                      10.103.87.246
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30100/TCP
Endpoints:                192.168.1.4:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Step: 4

To check the service

```
controlplane $ kubectl describe service
Name:              kubernetes
Namespace:         default
Labels:            component=apiserver
                   provider=kubernetes
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.96.0.1
IPs:               10.96.0.1
Port:              https  443/TCP
TargetPort:        6443/TCP
Endpoints:         172.30.1.2:6443
Session Affinity:  None
Events:            <none>
```

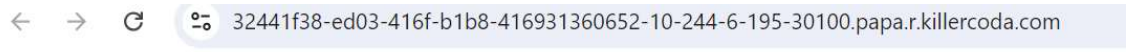
```
Name:                     nodeport
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=my-app
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.103.87.246
IPs:                      10.103.87.246
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30100/TCP
Endpoints:                192.168.1.4:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Step: 5

To enter the port number and verify it

Step: 6

To enter the port no and access the images



It works!

Loadbalancer Methods:

- LoadBalancer in Kubernetes creates an external load balancer that routes traffic to a service, making it accessible from outside the cluster.
- It automatically provisions a load balancer from the cloud provider, distributing incoming requests across the pods.
- This service type is ideal for exposing applications to external users with built-in scalability and high availability.

Step: 1

Create the Yaml file

vi loadbalance.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new4deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: httpd-container
        image: httpd
        ports:
          - containerPort: 80

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: loadsbalancer
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
  type: LoadBalancer
```

Step: 2

To create the pods

```
controlplane $ kubectl create -f loadbalance.yaml
deployment.apps/new4deploy created
service/loadsbalancer created
```

Step: 3

To check the status kubectl describe service/loadbalancer

```
controlplane $ kubectl describe service/loadbalancer
Name:                     loadbalancer
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=my-app
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.98.114.91
IPs:                      10.98.114.91
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31455/TCP
Endpoints:                192.168.0.4:80,192.168.0.5:80,192.168.1.4:80 + 1 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Step: 4

To verify the service

```
controlplane $ kubectl describe service
Name:              kubernetes
Namespace:         default
Labels:            component=apiserver
                   provider=kubernetes
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.96.0.1
IPs:               10.96.0.1
Port:              https  443/TCP
TargetPort:        6443/TCP
Endpoints:         172.30.1.2:6443
Session Affinity:  None
Events:            <none>
```

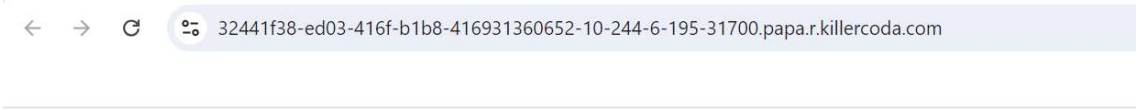
```
Name:                     loadsbalancer
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=my-app
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.109.254.39
IPs:                      10.109.254.39
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31700/TCP
Endpoints:                192.168.0.4:80,192.168.0.5:80,192.168.1.4:80 + 1 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Step: 5

To enter the port number and verify it

Step: 6

To enter the port no and access the images



It works!

Step: 7

To see the all services

```
controlplane $ kubectl get svc
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP      10.96.0.1       <none>        443/TCP        24d
loadbalancer   LoadBalancer   10.98.114.91    <pending>     80:31455/TCP   10m
loadsbalancer  LoadBalancer   10.109.254.39   <pending>     80:31700/TCP   9m35s
nodeport       NodePort       10.103.87.246   <none>        80:30100/TCP   27m
```