Kubernetes Services — Detailed Tutorial

Pods in Kubernetes are **ephemeral**, meaning they:

- Get new IPs on restart
- Can be recreated by ReplicaSets
- Can scale up/down dynamically

You can't reliably communicate with pods directly.

This is why **Services** exist — they provide a **stable virtual IP (cluster-wide)** and **load balancing** across pod replicas.

---

1. What is a Kubernetes Service?

A Service is an abstraction that provides:

✔ A stable virtual IP (ClusterIP)

✔ A stable DNS name

✔ Load-balancing across backend pods

✔ Traffic routing through kube-proxy (iptables/ipvs)

✔ Access from inside or outside the cluster

---

2. How a Service Works Internally

1. You create a Service with a pod selector:

   ```
   selector:
     app: myapp
   ```

2. Kube-proxy creates load-balancing rules using:
   - iptables (older, common)
   - IPVS (newer, faster)
3. Service gets a **stable ClusterIP**.
4. Any Pod contacting that ClusterIP goes through **kube-proxy**, which forwards traffic

to matching pods.

---

3. Types of Kubernetes Services

There are 3 main service types:

Service Type	Purpose	Accessible From
ClusterIP (default)	Internal communication	Inside the cluster
NodePort	Expose service on each node's IP	External (nodeIP:nodePort)
LoadBalancer	Expose service via cloud LB (AWS/ALB/ELB, GCP, Azure)	Internet or VPC

Now let us go deep into each service type

ClusterIP Service (Default)

This is the **most common** service type, used for **internal service-to-service communication**.

Use cases:

- Microservices calling each other
- Internal databases
- Backends not exposed to internet
- Communication between frontend → backend

Example:

```
frontend → backend
backend → database
```

ClusterIP YAML Example

```
apiVersion: v1
kind: Service
metadata:
  name: backend-svc
spec:
```

```
  type: ClusterIP
  selector:
    app: backend
  ports:
  - port: 80          # Service port
    targetPort: 8080  # Pod containerPort
```

Breakdown:

- `port`: Port exposed by the service
- `targetPort`: Pod's container port
- `ClusterIP`: Assigned automatically

How to access:

Inside cluster:

```
curl http://backend-svc
```

Or using namespace:

```
curl http://backend-svc.default.svc.cluster.local
```

NodePort Service

This exposes a Service on **every** node's IP at a static port (NodePort).
The port range for NodePort is: **30000–32767**

Traffic flow:

```
External Client → NodeIP:NodePort → Service → Pods
```

Use cases:

- Non-cloud environments (bare-metal)
- Local clusters (kubeadm, kind, minikube)
- Debugging external access

NodePort YAML Example

```
apiVersion: v1
kind: Service
metadata:
  name: backend-nodeport
```

```
spec:
  type: NodePort
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 32080      # Optional; can be auto-assigned
```

Access externally:

`http://<NodeIP>:32080`

You can hit **any** worker node IP — Kubernetes forwards it to the right pods.

NodePort Limitations:

- Exposes service on *all* nodes → not ideal for security
- Fixed high port range
- No smart routing (no CDN, health checks)
- Not production-grade for public internet

LoadBalancer Service (Cloud Only)

Used to expose service **publicly** with a cloud load balancer.

Supported in:

- AWS (ELB, NLB, ALB)
- GCP
- Azure
- DigitalOcean

Traffic flow:

`Internet → Cloud Load Balancer → NodePort → Service → Pods`

Yes — internally LoadBalancer still uses **NodePort**.

LoadBalancer YAML Example

```
apiVersion: v1
kind: Service
metadata:
  name: backend-lb
spec:
  type: LoadBalancer
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 8080
```

After creation:

```
kubectl get svc backend-lb
```

You will see:

- `EXTERNAL-IP`: Public IP assigned by cloud provider

Access:

```
http://<EXTERNAL-IP>
```

Additional Concepts

Selectorless Services (Headless Services)

Set:

```
clusterIP: None
```

Used for:

- StatefulSets
- Databases
- DNS-based discovery

Multi-Port Services

A single service can expose multiple ports:

```
ports:
- name: http
  port: 80
```

```
  targetPort: 8080
- name: https
  port: 443
  targetPort: 8443
```

Service DNS Resolution

Every service gets DNS:

`<servicename>.<namespace>.svc.cluster.local`

Example:

`redis.master.svc.cluster.local`

Service Without Selector

Used when:

- Manually managing endpoints
- ExternalName services

Example:

```
kind: Service
spec:
  type: ExternalName
  externalName: database.company.com
```

Which Service Type to Use?

Need	Service Type
Internal-only microservices	ClusterIP
Expose to local network without cloud LB	NodePort
Production external access	LoadBalancer
StatefulSets / DBs	Headless
Access external DB	ExternalName