DevOps Shack

Service Types in Kubernetes

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]

In Kubernetes, services enable communication between various components (pods) within a cluster and from outside the cluster. They provide an abstraction layer that decouples the logical definition of a service from its implementation, making it easier to manage and scale applications. There are mainly four types of services in Kubernetes:

ClusterIP Service

A ClusterIP service exposes the service on an internal IP within the Kubernetes cluster. These services are only reachable from within the cluster. They are often used for inter-component communication within the cluster.

Example Code:

Here's an example YAML manifest for creating a ClusterIP service named `my-service`:

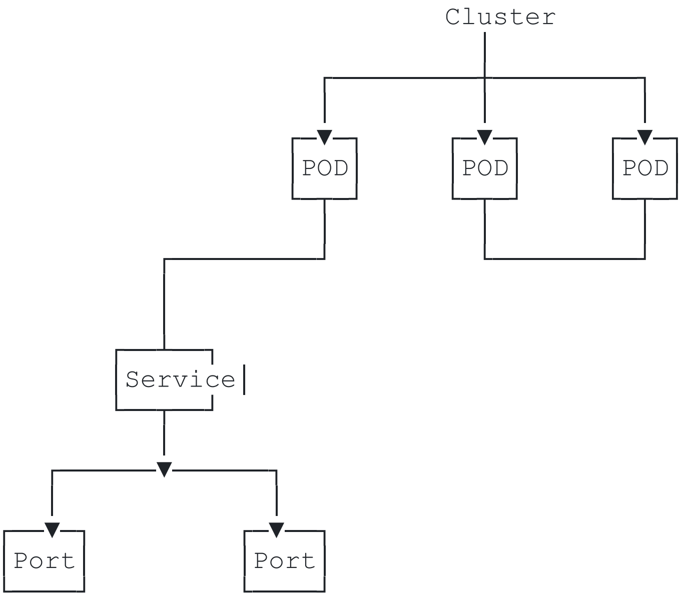
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

In this example:

- `apiVersion` specifies the Kubernetes API version being used.
- `kind` specifies the type of Kubernetes resource, in this case, a Service.

- `metadata` contains information about the service, including its name.
- `spec` specifies the desired state for the service.
  - `selector` specifies the labels that identify the pods the service will route traffic to.
  - `ports` specifies the ports that the service will listen on and forward traffic to.
    - `port` is the port on which the service will listen.
    - `targetPort` is the port on the pods to which the traffic will be forwarded.

Diagram:



In the diagram:

- Pods represent the application components running within the cluster.
- The Service acts as an internal load balancer, routing traffic to pods based on their labels.
- Ports define the communication channels between the Service and the Pods.

This setup allows other components within the cluster to access the `my-service` using its internal IP address, providing a stable way for inter-component communication.

NodePort Service

A NodePort service exposes the service on a static port on each node's IP. It allocates a port from a predefined range and forwards traffic to the service on the specified port. NodePort services make the service accessible from outside the cluster by using `<NodeIP>:<NodePort>`.

Example Code:

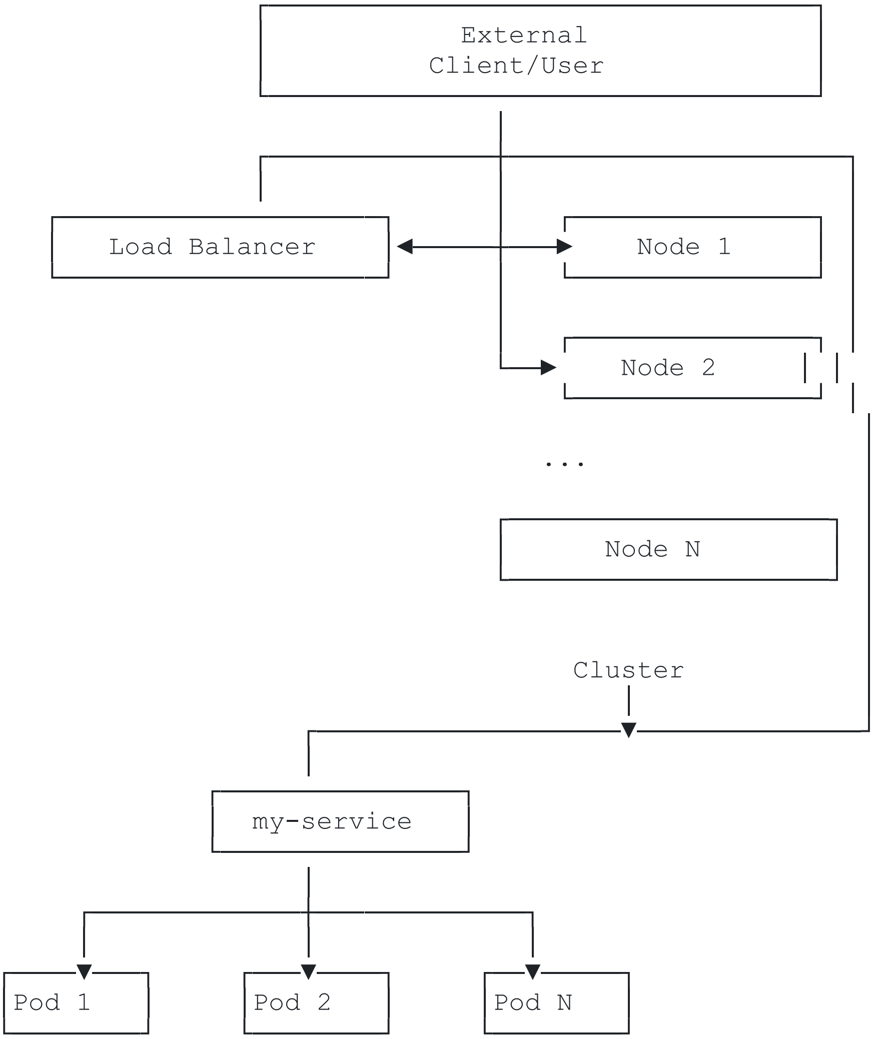
Here's an example YAML manifest for creating a NodePort service named `my-service`:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30000  # NodePort range: 30000-32767
```

In this example:

- `apiVersion`, `kind`, and `metadata` define the basic service metadata.
- `spec` specifies the desired state for the service.
    - `selector` specifies the labels that identify the pods the service will route traffic to.
    - `ports` specifies the ports that the service will listen on and forward traffic to.
        - `port` is the port on which the service will listen.
        - `targetPort` is the port on the pods to which the traffic will be forwarded.
        - `nodePort` is the port on each node through which the service will be accessible.

Diagram:



In the diagram:

- External clients/users communicate with the service through NodePorts, which are accessible on each node's IP.
- Each node forwards traffic received on the NodePort to the pods running the application (identified by the selector labels).
- Pods represent the application components running within the cluster.
- The service acts as a load balancer, distributing traffic across multiple pods if they exist.

This setup allows external clients to access the `my-service` using any of the cluster's node IPs and the assigned NodePort, providing a way to expose services to the outside world.

LoadBalancer Service

A LoadBalancer service exposes the service externally using a cloud provider's load balancer. This type of service is useful when you need to make your service accessible from outside the Kubernetes cluster, such as from the internet.

Example Code:

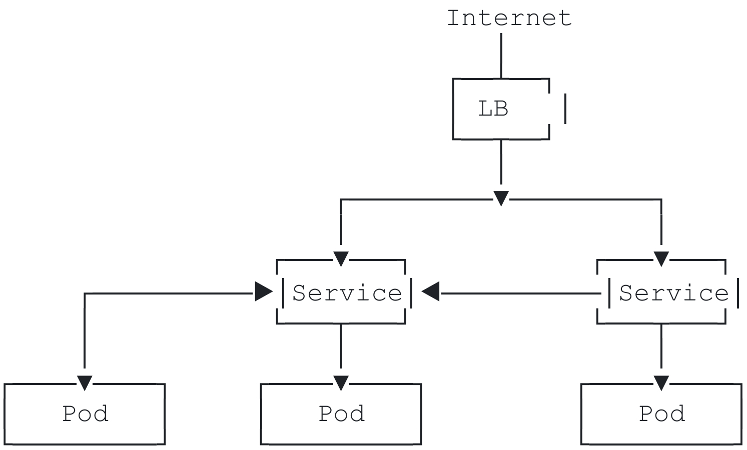
Below is an example YAML manifest for creating a LoadBalancer service named `my-loadbalancer-service`:

```
apiVersion: v1
kind: Service
metadata:
  name: my-loadbalancer-service
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

In this example:

- `apiVersion` specifies the Kubernetes API version being used.
- `kind` specifies the type of Kubernetes resource, in this case, a Service.
- `metadata` contains information about the service, including its name.
- `spec` specifies the desired state for the service.
  - `type` is set to `LoadBalancer` to indicate the service type.
  - `selector` specifies the labels that identify the pods the service will route traffic to.
  - `ports` specifies the ports that the service will listen on and forward traffic to.
    - `port` is the port on which the service will listen.
    - `targetPort` is the port on the pods to which the traffic will be forwarded.

Diagram:



In the diagram:

- The Load Balancer (LB) is a cloud provider's load balancer that distributes incoming traffic among the nodes in the Kubernetes cluster.
- The LoadBalancer service routes external traffic to the pods running within the cluster based on their labels.
- Pods represent the application components running within the cluster.

This setup enables external clients to access the `my-loadbalancer-service` from the internet through the cloud provider's load balancer, which then forwards the traffic to the pods running the application within the Kubernetes cluster.

ExternalName Service

An ExternalName service does not have any selectors or endpoints like other service types. Instead, it simply returns a CNAME record with the DNS name specified in its `externalName` field.

Example Code:

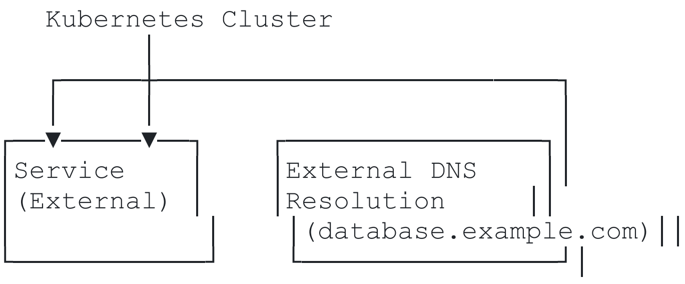
Here's an example YAML manifest for creating an ExternalName service named `external-svc`:

```
apiVersion: v1
kind: Service
metadata:
  name: external-svc
spec:
  type: ExternalName
  externalName: database.example.com
```

In this example:

- `apiVersion` specifies the Kubernetes API version being used.
- `kind` specifies the type of Kubernetes resource, in this case, a Service.
- `metadata` contains information about the service, including its name.
- `spec` specifies the desired state for the service.
    - o `type: ExternalName` specifies the service type as ExternalName.
    - o `externalName: database.example.com` specifies the DNS name to map the service to.

Diagram:



In the diagram:

- The Kubernetes Cluster contains the ExternalName service named `external-svc`.
- When services within the cluster attempt to access `external-svc`, Kubernetes resolves the DNS name specified in `externalName` field (`database.example.com`) through the External DNS resolution mechanism.
- External DNS resolves `database.example.com` to the corresponding IP address outside the Kubernetes cluster.

This setup allows components within the Kubernetes cluster to access external services using a DNS name rather than an IP address, providing flexibility and abstraction in managing external dependencies.