

**Introducing Services**

A Kubernetes service is a resource you create to make a single, constant point of entry to a group of pods providing the same service. Each service has an IP address and port that never change while the service exists. Clients can open connections to that IP and Port and those connections are then routed to ones on the pod backing the service. This way, clients of service don't need to know the location of individual pods providing the service, allowing those pods to be moved around the cluster at any time.

**Example**

Let’s take an example of a web app where you have a front-end web server and a back-end database server. There may be multiple pods that all act as the front end, but there may only be a single backend database pod. You need to solve two problems to make the system function:

External clients need to connect to the frontend pods without caring if there’s only a single web server or hundreds.

The frontend pods need to connect to the backend database. Because the database runs inside a pod, it may be moved around the cluster over time, causing its IP address to change.

You don’t want to reconfigure the frontend pods every time the backend database is moved.

By creating a service for the frontend pods and configuring it to be accessible from outside the cluster, you expose a single, constant IP address through which external clients can connect to the pods. Similarly, by also creating a service for the backend pod, you create a stable address for the backend pod.

The service address doesn’t change even if the pod’s IP address changes. Additionally, by creating the service, you also enable the frontend pods to easily find the backend service by its name through either environment variables or DNS. All the components of your system (the two services, the two sets of pods backing those services, and the interdependencies between them)

**Types of services**

**ClusterIP**

A ClusterIP Service in Kubernetes is the most basic type of service and the default option when creating a new service.

A ClusterIP service provides a stable IP address (known as the “ClusterIP”) that routes traffic within the cluster to a set of pods.

The ClusterIP address is only accessible from within the cluster network, so it cannot be directly accessed from the external network.

ClusterIP services are useful for communication between pods within the same cluster, for example, for distributing network load between multiple replicas of a single service.

When a client within the cluster makes a request to the ClusterIP, it is automatically load balanced to one of the available pods. This way, you can access your service using the ClusterIP, and Kubernetes will automatically route the traffic to the appropriate pods.

In summary, ClusterIP services are designed for internal communication within a cluster and provide a simple way to access a set of pods behind a stable IP address.

The ClusterIP exposes the Service on a cluster-internal IP, which makes it reachable from within the cluster only.

The ClusterIP provides a load-balanced IP address. One or more pods that match a label selector can forward traffic to the IP address. The ClusterIP service must define one or more ports to listen on with target ports to forward TCP/UDP traffic to containers.

**Creating a ClusterIP**

First, let's create a ClusterIP using a YAML file

apiVersion: v1  
kind: Service  
metadata:  
 name: example-service  
spec:  
 selector:  
 app: example-app  
 ports:  
 - name: http  
 port: 80  
 targetPort: 8080  
 type: ClusterIP

In this example, the Service is named example-service and is of type ClusterIP.

The selector app: example-app is used to select all Pods that app: example-app will be part of this service.

The Service will be available on port 80. Target port 8080 is used to forward traffic to the container.

Once you have created the definition file, you can create the Service by running the following command:

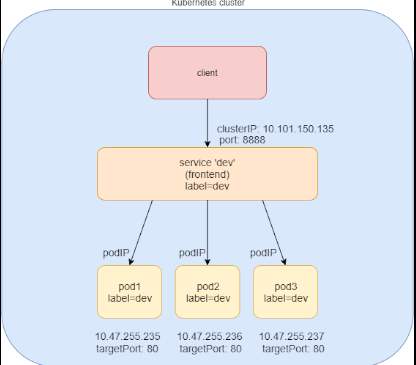
kubectl create -f cluster-ip-service.yaml

This will create a new Service in the cluster, which can be reached from within the cluster using the Service’s IP address.

You can view services by:

kubectl get svc

ClusterIP services should be used when you need to provide a stable IP address for pods within the cluster to communicate with each other.



Some common use cases for ClusterIP services are:

1. **Internal communication between pods:** If you have multiple pods that need to communicate with each other, you can use a ClusterIP service to provide a stable IP address for them to use.
2. **Load balancing between pods:** ClusterIP services provide built-in load balancing capabilities, so you can use them to distribute network traffic between multiple replicas of a single service.
3. **Service discovery:** ClusterIP services provide a stable IP address that can be used by other pods to discover and access a specific service.
4. **Maintenance:** By using a ClusterIP service, you can perform maintenance on individual pods without affecting the overall service. For example, you can update or roll back a single pod without affecting the others, as the ClusterIP service will continue to route traffic to the available pods.

It’s important to note that ClusterIP services are not intended for external access, as they are only accessible within the cluster network. If you need to provide external access to your service, you should consider using another type of Kubernetes service, such as a LoadBalancer service or a NodePort service.

**Node Port Service**

A NodePort service in Kubernetes is a type of service that exposes a service on a static port on each node in the cluster. This means that the service can be accessed from outside the cluster by sending traffic to any of the nodes on the specified NodePort.

A NodePort service provides a way to expose a service to the external network, but it does not provide the same level of load balancing and service discovery as a LoadBalancer service.

When you create a NodePort service, Kubernetes automatically assigns a static port on each node in the cluster and maps it to the service’s target pods.

Traffic to the specified NodePort on any node in the cluster is then automatically forwarded to the pods.

NodePort services are useful in situations where you need to expose a service to the external network but do not require the advanced features of a LoadBalancer service.

However, it is important to note that NodePort services do not provide the same level of network security as LoadBalancer services, as they expose the service directly to the node.

To create a NodePort service in Kubernetes, you can use the kubectl command-line tool and specify the type of service as "NodePort". Here's an example of how you can create a NodePort service for a deployment:

**Create a deployment:**

kubectl create deployment my-deployment --image=nginx

This will create a deployment with the name my-deployment and the image will be nginx

**Create a NodePort service:**

kubectl expose deployment my-deployment --type=NodePort --port=80

This creates a NodePort service that maps to the my-deployment deployment and exposes it on port 80.

Check the NodePort assigned to the service:

kubectl describe service my-deployment

This will show the NodePort assigned to the service, along with other information such as the ClusterIP and Endpoints.

Access the service from outside the cluster:

You can access the service by sending traffic to any of the nodes in the cluster on the specified NodePort. For example, if the NodePort is 30080, you can access the service by sending a request to <node-ip>:30080 where <node-ip> is the IP address of any node in the cluster.

Creating NodePort using a YAML file

apiVersion: v1  
kind: Service  
metadata:  
 name: my-service  
spec:  
 selector:  
 app: my-app  
 type: NodePort  
 ports:  
 - name: http  
 port: 80  
 targetPort: 80  
 nodePort: 30080

kubectl create my-service.yaml

This creates a NodePort service that maps to the my-deployment deployment and exposes it on port 80, with a NodePort of 30080.

Check the NodePort assigned to the service:

kubectl describe service my-service

This will show the NodePort assigned to the service, along with other information such as the ClusterIP and Endpoints.

NodePort is one of the ways to expose a service in Kubernetes. It is a good choice when:

1. **You need to expose a service to the external network:** NodePort allows you to expose a service to the external network by sending traffic to any of the nodes in the cluster on a specified port.
2. **You don’t need the advanced features of a LoadBalancer:** NodePort is a simpler solution compared to LoadBalancer, as it does not provide the same level of load balancing and service discovery. If you don’t need these advanced features, NodePort is a good choice.
3. **You have a limited budget:** LoadBalancer is often more expensive than NodePort, as it requires a load balancer in the cloud provider. If you have a limited budget, NodePort is a more cost-effective solution.
4. **You need to access services from on-premise applications:** NodePort is a good choice when you need to access services from on-premise applications, as it does not require a LoadBalancer, which is often not available on-premise.

It’s important to note that NodePort does not provide the same level of network security as LoadBalancer, as it exposes the service directly on the node. So, if security is a concern, you may want to consider LoadBalancer instead.

**Load balancer**

A LoadBalancer service in Kubernetes is a type of service that exposes a service to the external network by distributing incoming traffic across multiple pods. It provides advanced features such as load balancing and service discovery.

When you create a LoadBalancer service, Kubernetes creates a load balancer in your cloud provider and configures it to forward traffic to the pods in your service.

The load balancer uses IP addresses from the cloud provider to route traffic to the service.

A LoadBalancer service is a good choice when you need to:

1. **Expose a service to the external network:** LoadBalancer allows you to expose a service to the external network by routing traffic to the pods in the service.
2. **Balance the load between multiple pods:** LoadBalancer distributes incoming traffic across multiple pods, allowing you to scale your service and balance the load.
3. **Ensure high availability:** LoadBalancer provides automatic failover, ensuring high availability even in the event of a pod failure.
4. **Discover services:** LoadBalancer provides service discovery, allowing you to easily discover and access other services in the cluster.

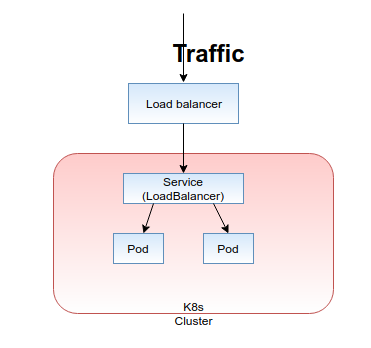
To create a LoadBalancer service, you simply need to specify the type as LoadBalancer in the service definition. When you create the service, Kubernetes will automatically create a load balancer in your cloud provider and configure it to forward traffic to the pods in your service.

Here is an example of a LoadBalancer service definition in YAML format

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

In this example, the LoadBalancer service named my-service will expose a service running on port 80 and target port 8080. The service will balance the load between pods labeled with app: my-app.

Once you create this service, Kubernetes will automatically create a load balancer in your cloud provider and configure it to forward traffic to the pods in your service. The service can then be accessed from the external network using the IP address of the load balancer.



**Ingress**

Ingress is a Kubernetes resource that allows you to expose multiple services under a single IP address. An Ingress resource defines rules that determine how traffic from the external network should be forwarded to the services in your cluster.

With Ingress, you can expose multiple services under a single IP address, and use rules to route traffic to the appropriate service based on the URL path or hostname. This allows you to expose multiple services with a single LoadBalancer or NodePort, reducing the cost and complexity of managing multiple load balancers or node ports.

Ingress also provides additional features such as SSL termination, URL rewrites, and authentication, allowing you to implement complex routing rules and secure access to your services.

To use Ingress in your cluster, you need to have an Ingress controller installed. The Ingress controller is responsible for interpreting the Ingress resource and configuring the appropriate load balancer or reverse proxy to implement the rules defined in the Ingress resource.

Here is an example of an Ingress resource definition in YAML format:

apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
 name: my-ingress  
spec:  
 rules:  
 - host: example.com  
 http:  
 paths:  
 - path: /app1  
 pathType: Prefix  
 backend:  
 service:  
 name: app1  
 port:  
 name: http  
 - host: example.

The example above provided is a YAML definition of a Kubernetes Ingress resource. This Ingress resource defines two routing rules:

1. For requests to example.com/app1, the traffic should be forwarded to the app1 service, on port http.
2. For requests to example.com, the traffic should be forwarded to an unnamed service, on an unnamed port.

The Ingress resource will work with an Ingress controller to route incoming traffic from the external network to the appropriate service based on the rules defined in the Ingress resource.

Note that the example is incomplete and some details are missing. To create a functional Ingress resource, you would need to define the services that the Ingress resource references, and also have an Ingress controller installed in your cluster.

