Service

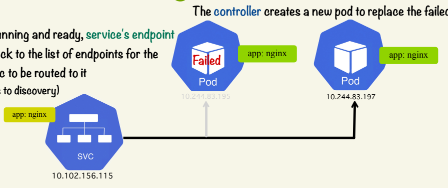
Services are a core component in Kubernetes that are used to manage networking and traffic flow within a cluster. They provide a stable IP address and DNS name for a set of pods and allow for communication between different components within and outside of the application. Services also enable load balancing , service discovery and traffic management, making them a critical component for building scalable and resilient applications in Kubernetes.

When a service is created, it is assigned a virtual IP address (known as a ClusterIP), which is used to route traffic to the pods that are part of the service. The service also has a DNS name, which can be used to access the service from within the cluster

1 If a pod fails or is removed from the service, controller will automatically remove it from the list of endpoints for the service. This ensures that traffic is not sent to a non-existent pod.

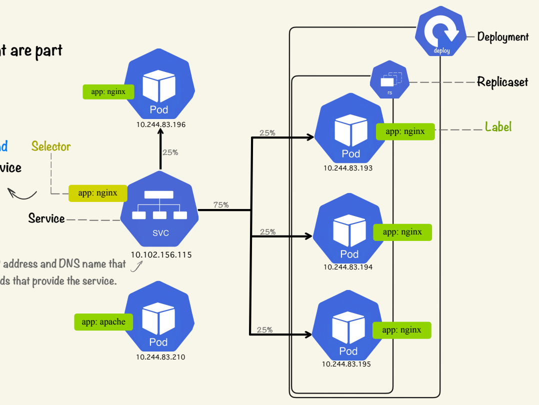
2 When a pod managed by a deployment fails,
The controller creates a new pod to replace the failed pod

3 Once the new pod is running and ready, service's endpoint controller will add it back to the list of endpoints for the service, allowing traffic to be routed to it
(used labels and selectors to discovery)



services use labels and selectors to discover and route traffic to the pods that are part of the service

Each service has a unique IP address and DNS name that can be used to access the pods that provide the service.



Service types in k8s

ClusterIP

This is the default type of service, which exposes the service on a cluster-internal IP address that is only accessible from within the cluster

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-internal
  namespace: dev
spec:
  type: ClusterIP
  ports:
  - targetPort: 80
    port: 80
    protocol: tcp
  selector:
    app: nginx
```

service will load balance traffic to any Pods that have the label app=nginx

The port is used to specify the port number that clients should use to access the service

NodePort

This type of service exposes the service on a static port on each node in the cluster, which can be accessed from outside the cluster using the node's IP address

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-ext
  namespace: dev
spec:
  type: NodePort
  ports:
  - targetPort: 8080
    port: 80
    nodePort:8080
  selector:
    app: nginx
```

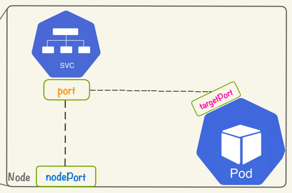
The targetPort is used to specify the port number on the Pods that the service should forward traffic to

The nodePort field is used to specify the high port number on each node in the cluster that can be used to access the service from outside the cluster

LoadBalancer

This type of service exposes the service using a cloud provider's load balancer, which distributes traffic to the different Pods

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-ext-lb
  namespace: dev
spec:
  type: LoadBalancer
  ports:
  - targetPort: 8080
    port: 8080
    nodePort:31090
  selector:
    app: nginx
```



The selector field specifies the label selector that the service will use to find the Pods that it should load balance traffic to

if you don't specify a nodePort value , Kubernetes will automatically allocate a random high port number (30000-32767)for the service

ExternalName

This type of service maps the service to an external DNS name, allowing the service to be accessed from within the cluster using a consistent name

To expose a service to the outside world in k8s, you can use one of the following methods

LoadBalancer Service
This method requires that your cloud provider supports LoadBalancer services, and it can incur additional costs

NodePort Service
This type of service exposes the service on a static port on each node in the cluster, which can be accessed from outside the cluster using the node's IP address

Ingress
An Ingress is a Kubernetes resource that defines a set of rules for routing external HTTP(S) traffic to a service. Ingress resources require an Ingress controller to be deployed in the cluster, which is responsible for implementing the routing rules. Ingress controllers are available for many popular web servers, such as Nginx, Traefik

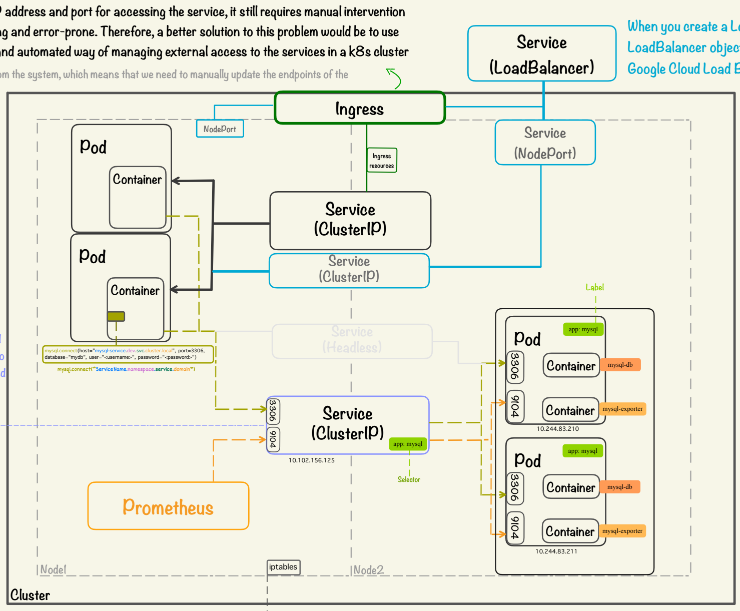
While a load balancer service can provide a stable IP address and port for accessing the service, it still requires manual intervention to update the endpoints, which can be time-consuming and error-prone. Therefore, a better solution to this problem would be to use Kubernetes Ingress, which provides a more flexible and automated way of managing external access to the services in a k8s cluster

it is possible that a node might be added or removed from the system, which means that we need to manually update the endpoints of the load balancer or recreate the service.

When you create a LoadBalancer service , it creates a cloud provider-specific LoadBalancer object, such as an Amazon Elastic Load Balancer (ELB) or Google Cloud Load Balancer, which is external to the Kubernetes cluster

When you create a Service of type LoadBalancer, k8s will automatically create a NodePort and ClusterIP for the Service.

Headless services are used for direct access to pods Go to page 36



For services that we do not want to expose to the outside world (such as database clusters like mysql), we set the service type to ClusterIP. If we do not specify the service type, it will be selected as ClusterIP by default

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    matchLabels:
      app: mysql
  ports:
  - name: mysql
    port: 3306
    targetPort: 3306
  - name: mysql-exporter
    port: 9104
    targetPort: 9104
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  namespace: dev
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:latest
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: password
            - name: MYSQL_DATABASE
              value: mydb
        - name: mysql-exporter
          image: prom/mysqld-exporter:v0.12.1
          ports:
            - containerPort: 9104
          env:
            - name: DATA_SOURCE_NAME
              value: "root:$MYSQL_ROOT_PASSWORD@(localhost:3306)/
$MYSQL_DATABASE?tls=false"
```

Go to page 36

In the real world, StatefulSets used instead of Deployments for stateful applications

When a service is created, the kube-proxy on each node in the cluster automatically creates iptables rules to forward traffic to the service endpoints.

```
iptables -A KUBE-SERVICES -d 10.102.156.125/32 -p tcp -m comment --comment "/* dev/mysql: cluster IP" -m tcp --dport 3306 -j KUBE-SVC-ABC123
```

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE   SELECTOR      NODE-PORT   ENDPOINTS
mysql-service  ClusterIP   10.102.156.125   <none>        3306/TCP,9104/TCP   1d    app=mysql     <none>      10.244.83.210:3306, 10.244.83.211:3306
```

```
iptables -A KUBE-SVC-<service-uid> -m comment --comment "dev/mysql-service:" -j KUBE-SEP-<endpoint-uid-1>
iptables -A KUBE-SVC-<service-uid> -m comment --comment "dev/mysql-service:" -j KUBE-SEP-<endpoint-uid-2>
```