07: Services

- Services provides stable and reliable networking for a set of unreliable Pods

Failures, scaling-ups, rollbacks, rollouts introduces new Pods with new IP address. **This creates massiva IP churn and demostrates why never connect directly to any Pod.**

- FIRST: K8s **Services** is an object in Kubernetes that provid3es stable **networking for Pods**. Like Deployment, Pod and ReplicaSet is a REST object in the API that is **defined in a manifest file and post to the API server.**

- SECOND: Every Service gets its own

  - **stable IP address**
  - **stable DNS name**
  - **stable port**
  - also load balanced traffic to Pods

- THIRD: Servces use **labels** and **selectos** to select the Pods to send traffic.

With Service, clients can access to the service without interruption due to Pods.

Labels

Services match with Pods via labels and selectors. The logic is an logic AND between the Service labels and Pod labels.

```
apiVersion: v1
kind: service
metadata:
  name: hello-svc
  spec:
    ports:
    - port: 8080
    selector:
      app: hello-world # Label 1
      env: tkb
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 10
  <snip>
  template:
    metadata:
      labels:
        app: hello-world
        env: tkb
    spec:
      containers:
<Snip>
```

The Service is selecting on Pods with both labels. It makes no difference if the Pods have additional labels.

If the Deployment has a Pods template with the same labels, any Pods it deploys will match the Service's selector and receive traffic from it.

- When a Service is created -> Kubernetes automaticaly creates associated **Endpoint** object than stores dunamic list of healthy Pods matching the Service's label selector.

LEER EndopointSlices que estan reemplazando a los Endpoints.

Dual Stack Networking

Pods and Services can have IPv4 and IPv6 in dual stack where IoT apps are supported.

Accesing Services from inside te cluster

- Default cluster: **ClusterIP**

ClusterIp Service has a stable virtual IP address that **only accessible from inside the cluster**. It's programmed into the internal network fabric and guaranteed to be stable for the life of the service.

Eveyr Service created gets a ClusterIP that's registered in the cluster's internal DNS service. ALl Pods in the cluster are pre-programmed to use the cluster's DNS service.

So if a Pod knows the name of a Service it can resolve it to a ClusterIP address and connect to the Pods behind it. It does not work outside of the cluster.

Accessing from outside the cluster

Two types of Service for requests from outside the cluster

NodePort

NodePort Service build on top of the clusterIP -> Allow external clients to hit a dedicated port on every cluster node and reach the Service. This type of service add to ClusterIP an additional NodePort that can be used to reach the Service from outside the cluster.

```
apiVErsion: v1
kind: Service
metadata:
  name: skippy
spec:
  type: NodePort
  ports:
  - port: 8080
```

```
    nodePort: 30050
  selector:
    app: hello-world
```

- Pods on the cluster can access this Service by hit skippy:8080.
- Clients can send traffic on port 30050.

LoadBalancer

LoadBalancer Srvices make external acccesswith a high-performance highly-available public IP or DNS name. Only work on clouds that support them.

Service Discovery

- DNS (Preferred)
- Env Variables (not preferred)

Kub clusters run on an internal DNS service that is the centre of service discovery. Every Pod/container can reslve every Service name to a ClusterIP and connect to the Pods behind it.

Hands-on

Dual Stack Networking Primer

- All cluster nodes need a routable IPv4 and IPv6 addresses
- CNI network plugin must support dual stacks

```
# dual-stack-svc.yml
kind: Service
metadata:
  name: dual-stack-svc
spec:
  ipFamilyPolicy: PreferDualStack
  type: ClusterIP
  ports:
  - port: 8080
  protocol: TCP
selector:
  app: hello-world
```

- ipFamilyPolicy
  - SingleStack
  - PreferDualStack
  - RequireDualStack

Prefer and Require dual stack tell Kub to allocate the Service an IPv4 and IPV6. The last one will fail if the cluster does not support dual stack networking.

```
kubectl apply -f dual-stack-svc.yml
kubectl get svc
kubectl describe svc dual-stack-svc
```

- output values:
  - Selector: list of labels
  - ipFamilyPoliy: single or dual stack
  - IP: permanent internal ClusterIP
  - Port: port the Service listens inside the cluster
  - TargetPort: port the app is listening
  - NodePort: cluster-wide port for external access
  - Endpoints: dynamic list of healthy Pod IPs

To access to the app from the web browser hit an IP address of at least one of the cluster nodes on the NodePort port.

- The web app deployed listen on 8080
- Kubernetes Service was config to listen on port 30013 on every cluster node
- **NodePorts are between 30.000 and 32.767**.

declarative way

```
apiVersion: v1
kind: Service
metadata:
  name: svc-test
spec:
  type: NodePort
  ports:
  - port: 8080
  nodePort: 30001
  targetPort: 8080
  protocol: TCP
selector:
  chapter: servic es
```

- Services are objects defined in v1 core API group
- Service Object
- in metadata block you can define name, labels and annotations -> labels are for identify Service, and are no related to selecting Pods
- selector tells Service to send traffic to all healthy Pods on the cluster with the chapter=services label.

```
kubectl apply -f svc.yml

kubectl get svc svc-test
```

ensure any firewall and security rules allow the traffic to flow. on local cluster, use localhost:30001

Endpoints Services

Endpoint object (same name as the Service) hods a list of all the Pods in the Service.

```
kubectl get endpointslices
kubectl describe endpointslice
```

LoadBalancer Services

- Easiest and the best type of Service
- `type=LoadBalancer` internet-facing load-balancer provider to the Service

```
# lb.yml
## listening on 9000
## forwarding to 8080 Pods
### label chapter=services

apiVersion: v1
kind: Service
metadata:
  name: cloud-lb
spec:
  type: LoadBalancer
  ports:
  - port: 9000
    targetPort: 8080
  selector:
    chapter: services
```

```
kubectl apply -f lb.yml
kubectl get svc --watch
```

- EXTERNAL-IP shows public address assigned to the Service by the cloud
- It may be a DNS name instead

```
kubectl delete -f deploy.yml -f svc.yml -f lb.yml
```