TEL4VN

Configuring and managing application access with services

THAO LUONG
06/2020

Content

- ❑ Kube proxy
- ❑ Understanding services
- ❑ Type of service
- ❑ Service network internals
- ❑ Service discovery

Kube-proxy

Kuber-proxy reflects service as define in the Kubernetes API on
each node and can do simple TCP,UDP and SCTP stream forwarding or round

robin TCP, UDP fowarding across a set of backends.

Kube-proxy

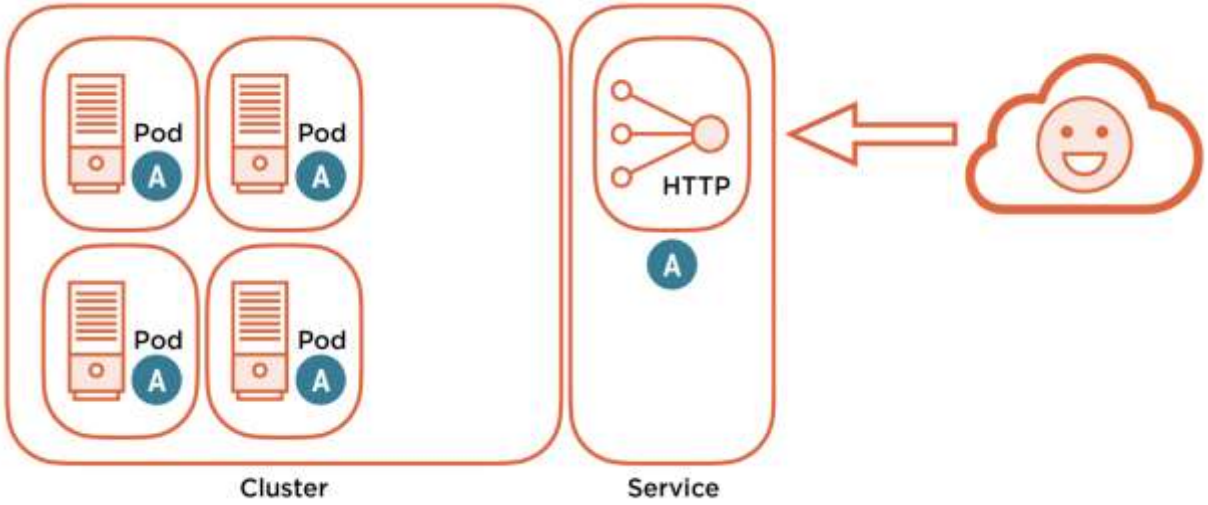
- Iptable mode: Iptable mode from within the kernel, directly

  forwards the connection to the pod, Fast and efficient but harder to debug

- Ipvs modewhen the Client Pod request reaches the kernel
  space, it is directly distributed to each Pod according to the ipvs

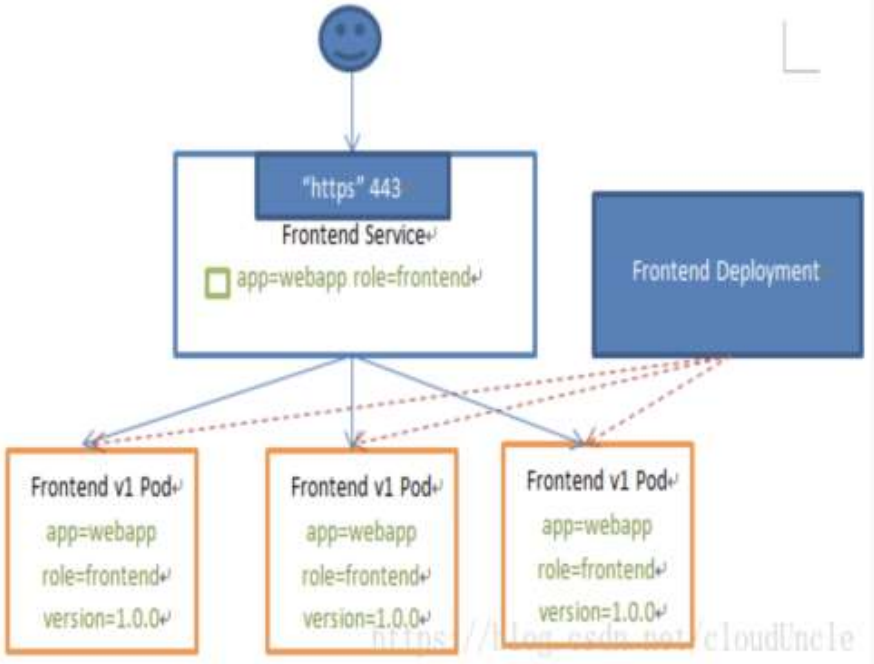
  rule

Services



Understanding services

- Service in K8s is an object, similar to a Pod
- A Service can be defined as a logical set of pods.



How services work

Services match Pods using Labels and Selectors

Creates and registers Endpoints in the Service (Pod IP and Port pair)

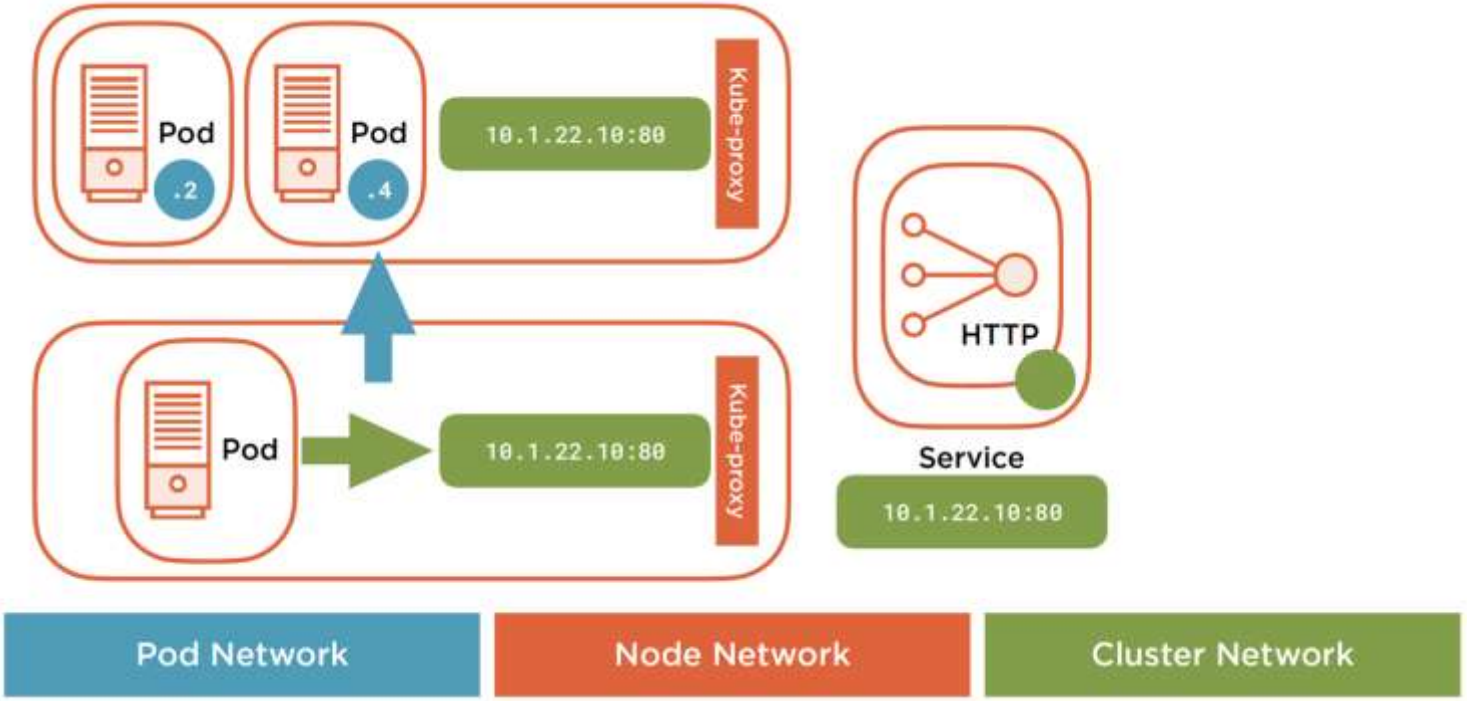
Implemented in the `kube-proxy` on the Node in iptables

kube-proxy watches the API Server and the Endpoints

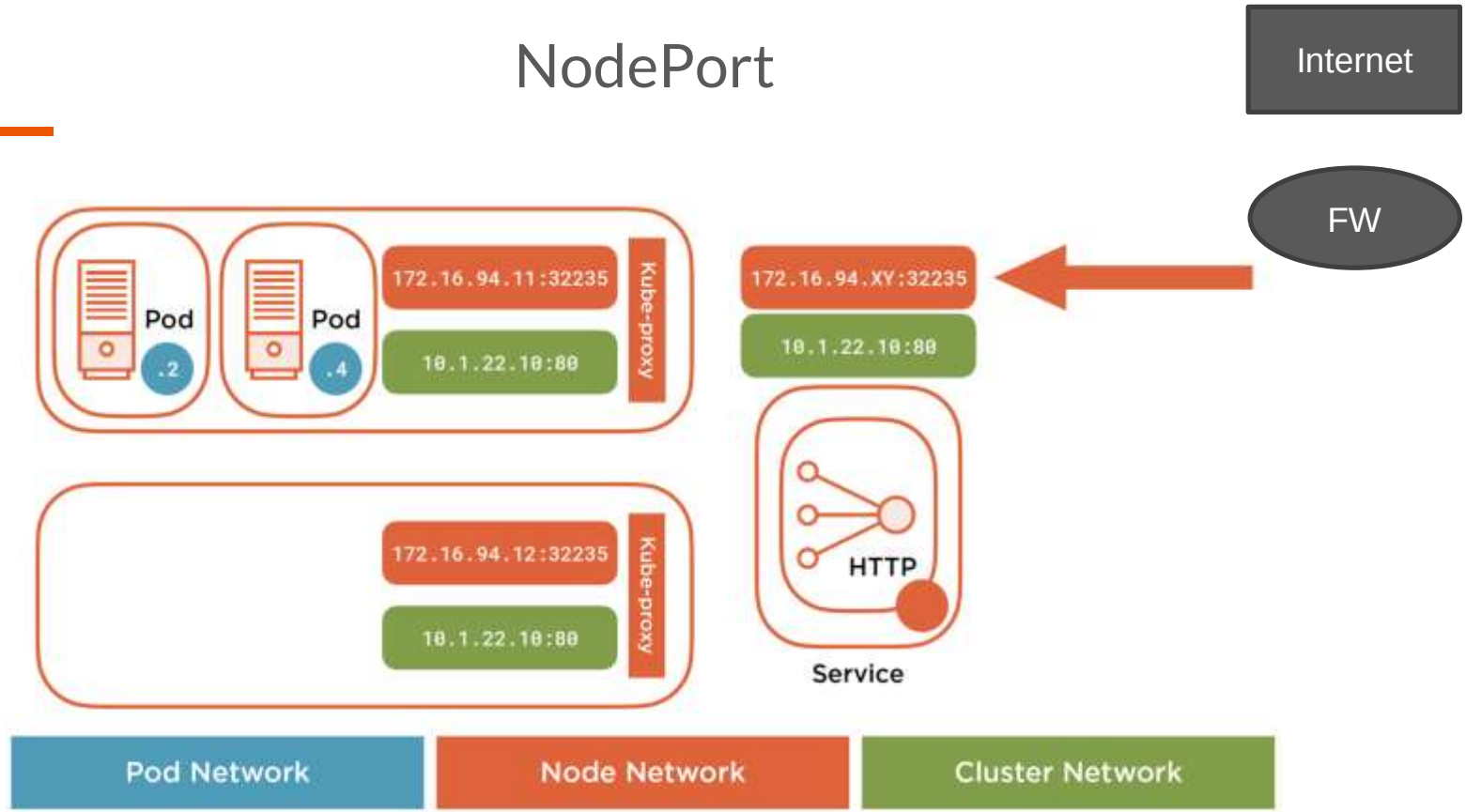
Service types

- ClusterIp: The default type, which automatically assigns a virtual IP that
  can be accessed only inside the Cluster.
- NodePort: Bind a port to each machine for Service on the basis of ClusterIP, so you can access the service through <NodeIP>: NodePort
- LoadBalancer: Based on the NodePort, create an external load balancer
  with the cloud provider and forward the request to <NodeIP>: NodePort

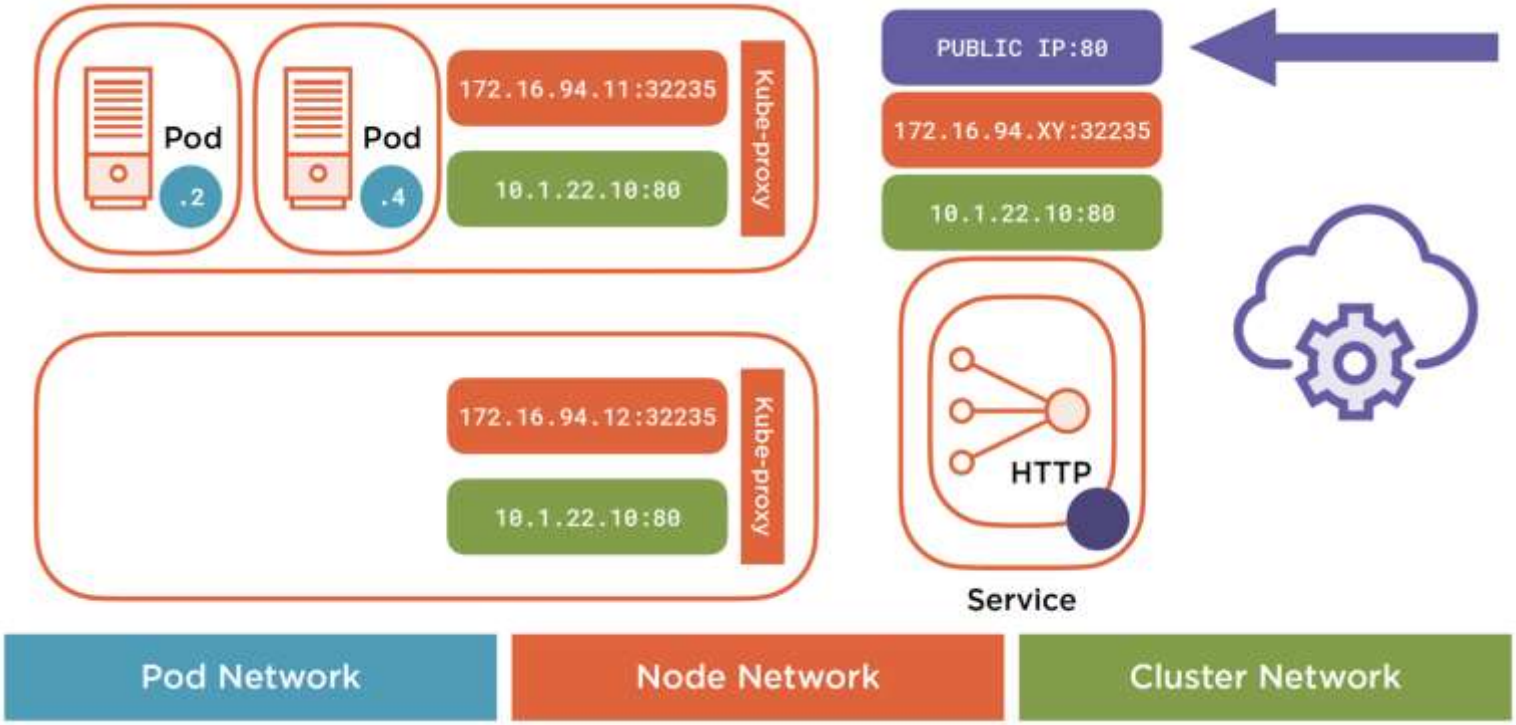
ClusterIP


Pod
.2
Pod
.4
10.1.22.10:80
Kube-proxy
Pod
10.1.22.10:80
Kube-proxy
HTTP
Service
10.1.22.10:80
Pod Network
Node Network
Cluster Network

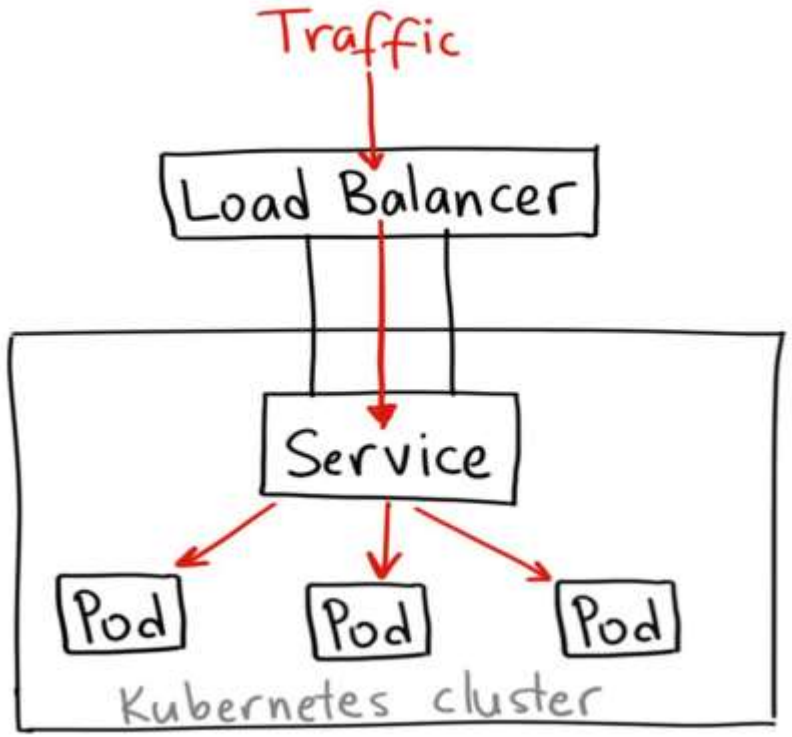
NodePort


Internet
FW
Pod
.2
Pod
.4
172.16.94.11:32235
10.1.22.10:80
Kube-proxy
172.16.94.12:32235
10.1.22.10:80
Kube-proxy
172.16.94.XY:32235
10.1.22.10:80
HTTP
Service
Pod Network
Node Network
Cluster Network

LoadBalancer



LoadBalancer



Defining deployments and services

```
kind: Deployment
...
  template:
    metadata:
      labels:
        run: hello-world
    spec:
        containers:
...
```

Match

```
kind: Service
...
spec:
  type: ClusterIP
  selector:
    run: hello-world
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
```

```
kubectl create deployment hello-world --image=gcr.io/google-samples/hello-app:1.0

kubectl expose deployment hello-world --port=80 --target-port=8080 --type NodePort
```

Demo 1

Expose and access app with services

- ClusterIP
- NodePort
- LoadBalancer


Q
A