Kubernetes Networking 101

Shubham Sharma
Microsoft

@shubham1172

Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

What next?

Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

What next?

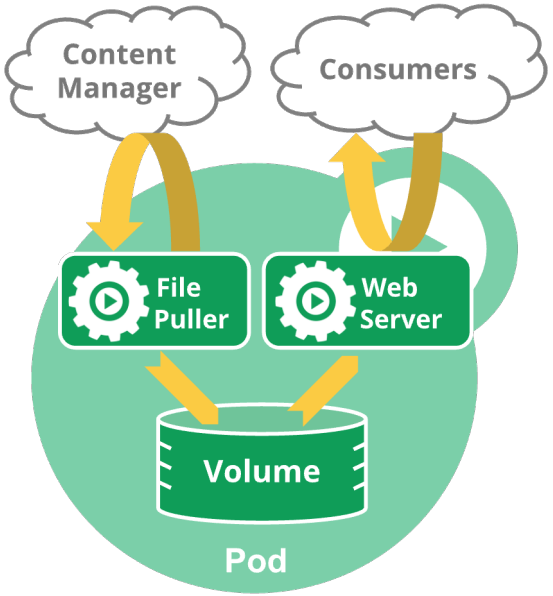
Pods

> Smallest deployable units of computing in Kubernetes

> Collection of one or more containers

> Unique IP address

> **Shared networking and storage**

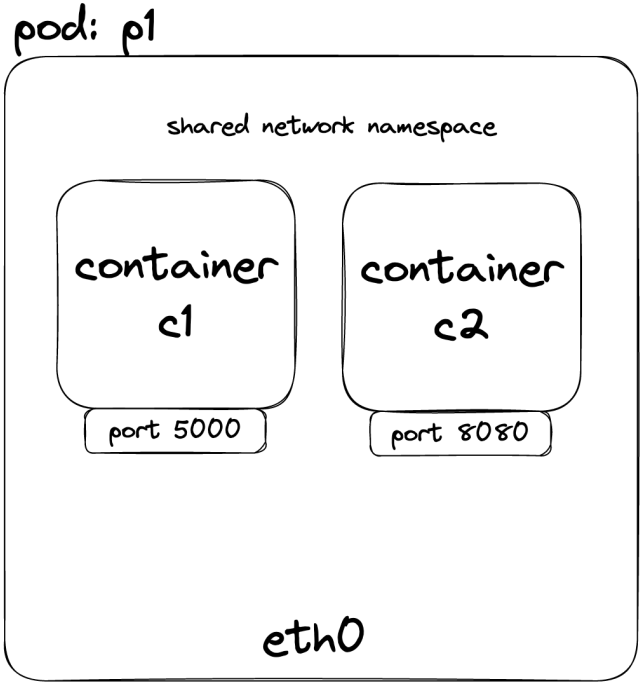


[1] How Pods manage multiple containers

Container networking

Every container in a Pod shares the network namespace, including the IP address and network ports.

Containers can communicate over
- localhost
- IPC (like POSIX shared memory)



Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

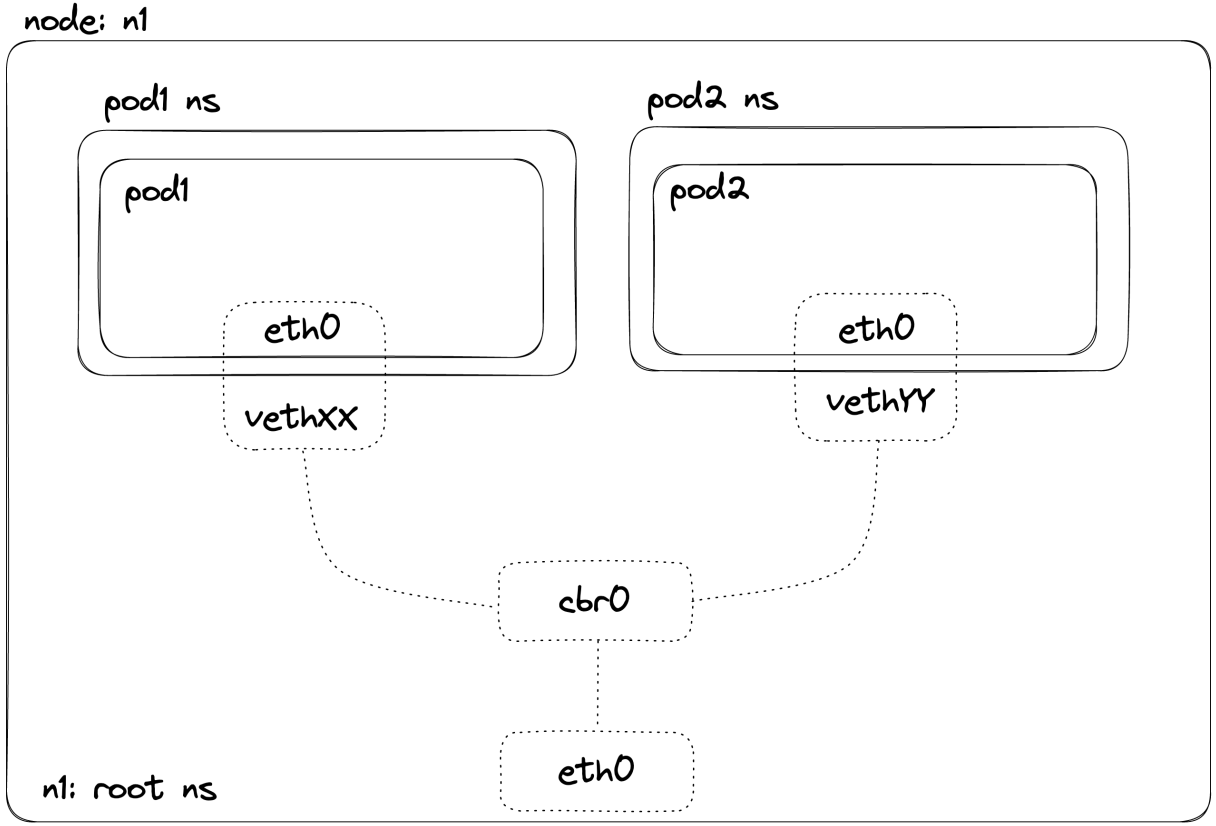
What next?

Pods on the same node

> Each pod has an IP address

> Virtual Ethernet Pairs

> Linux Ethernet Bridge (cbr0, docker0)



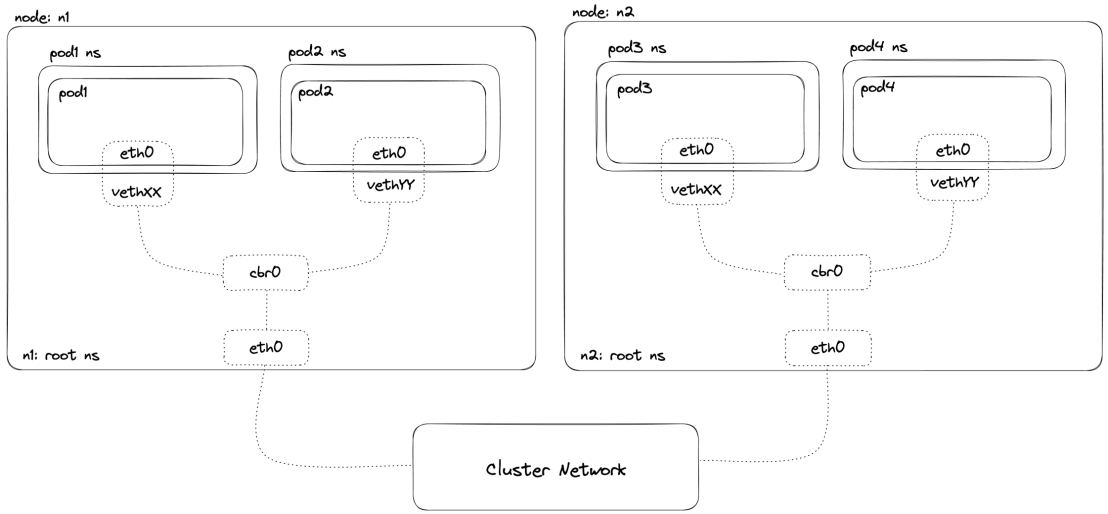
Pods across multiple nodes

> Bridge fails to resolve the MAC address

> Container Network Interface (CNI)

> CNI allocates IPs to Pods

> Different implementations (layer 2, layer 3, overlay, vxlan)



Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

What next?

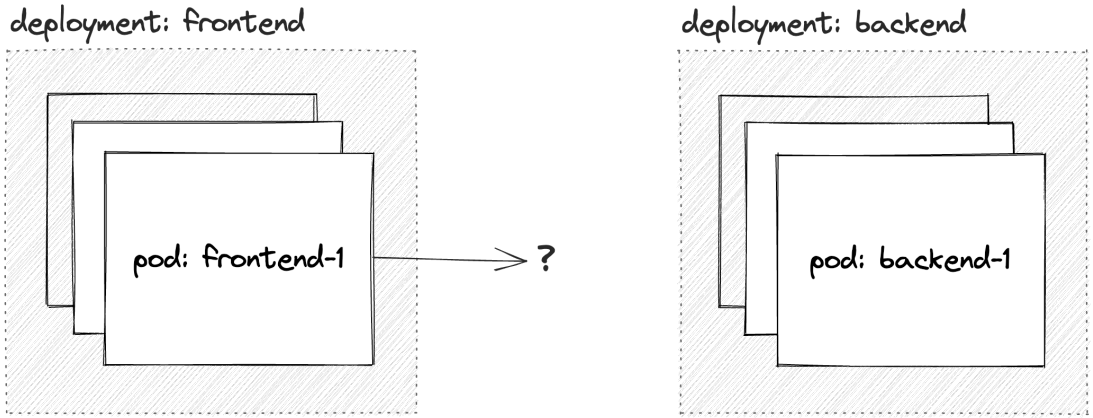
Service Discovery 101

> Virtual IPs are dynamic.

> Pods can be created/destroyed dynamically

> Which Pods are healthy?

How can a frontend Pods keep track of which IP addresses to connect to?

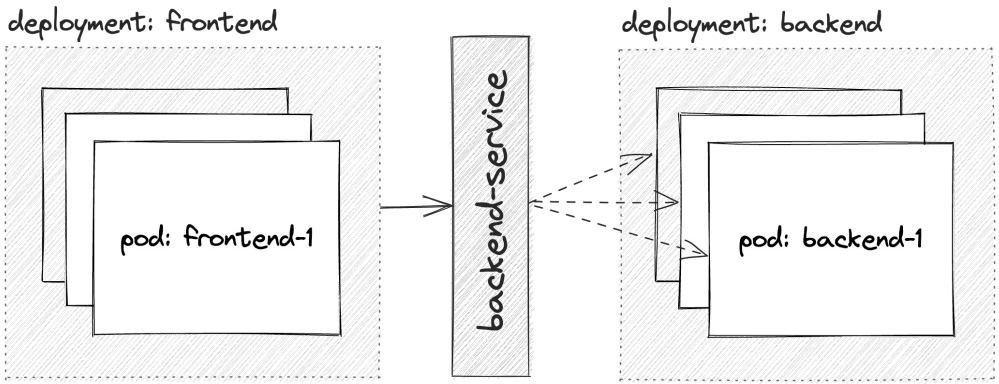


Service Object

> Service “backend-service” targets TCP **port 8080** on any Pod **with label** “app.kubernetes.io/name: backend”

> Service has an IP address (**ClusterIP**)

> Provides load-balancing across Pods




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: backend
  labels:
    app.kubernetes.io/name: backend
spec:
  containers:
  - name: mybackend
    image: contoso/mybackend
    ports:
      - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app.kubernetes.io/name: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```


Service DNS

> Cluster IP is virtual and stable

> Kubernetes DNS service

> Search domains

```
# DNS A record of K8s service
service.namespace.svc.cluster.local

# Contacting a service in the same namespace
other-service

# Contacting a service in another namespace
other-service.other-namespace
```


Example

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app.kubernetes.io/name: backend
  template:
    metadata:
      labels:
        app.kubernetes.io/name: backend
    spec:
      containers:
      - name: mybackend
        image: contoso/mybackend
        ports:
            - containerPort: 8080
```

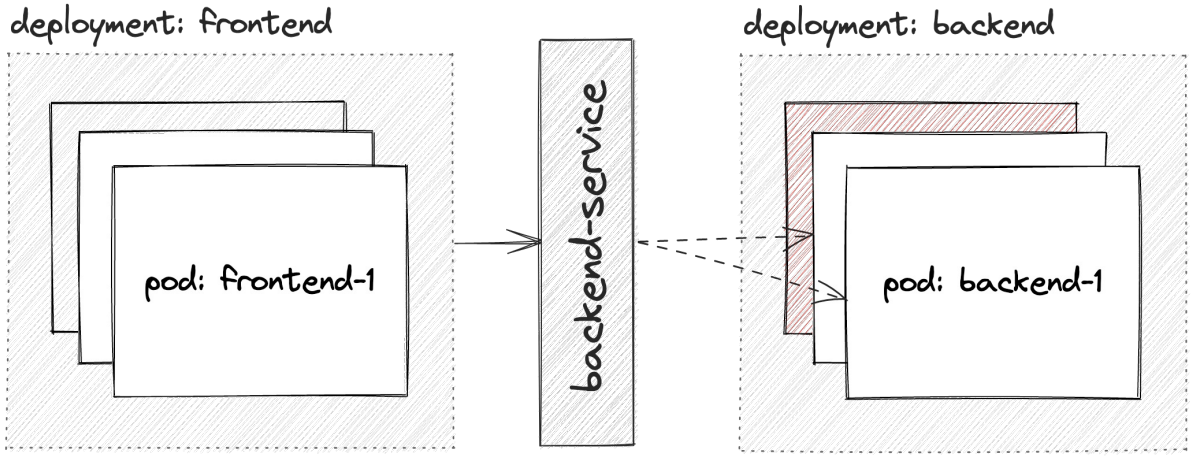
```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app.kubernetes.io/name: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app.kubernetes.io/name: frontend
  template:
    metadata:
      labels:
        app.kubernetes.io/name: frontend
    spec:
      containers:
      - name: myfrontend
        image: contoso/myfrontend
        env:
          - name: BACKEND_SERVICE_NAME
            value: backend-service
```


Readiness checks

Service objects can track which Pods are ready via a readiness check.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  ...
  template:
    ...
    spec:
      containers:
      - name: mybackend
        image: consoto/mybackend
        readinessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 10
          failureThreshold: 3
         ...
```



Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

What next?

Incoming traffic?

> ClusterIPs are virtual IPs **within** the cluster

> How to allow incoming traffic to Pods?

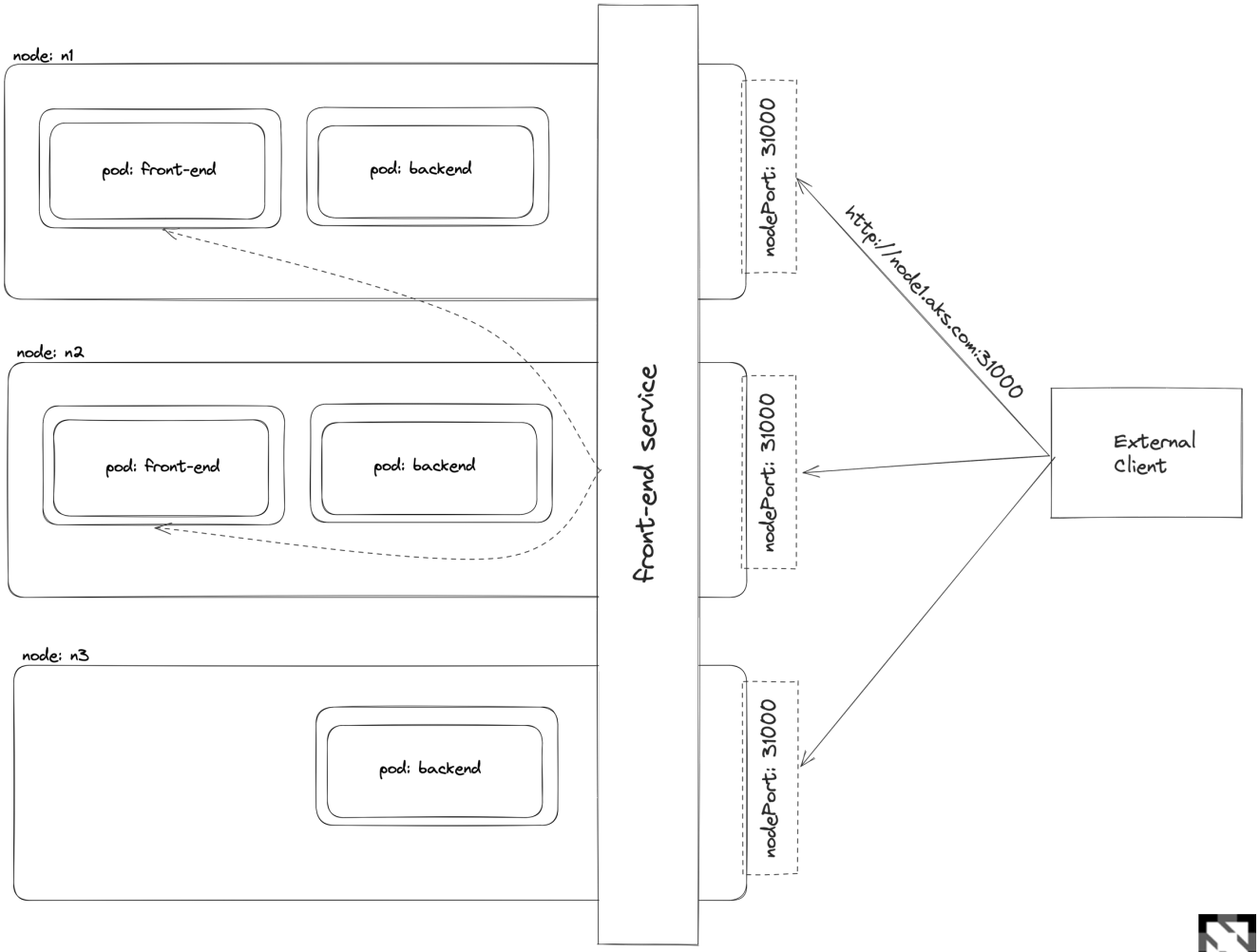
NodePort

Allow access from external clients via a universal port – **NodePort**

> Builds on top of Cluster IP internally

> Every node listens on a port and proxies it into a Service

> Port is reserved cluster-wide



NodePort

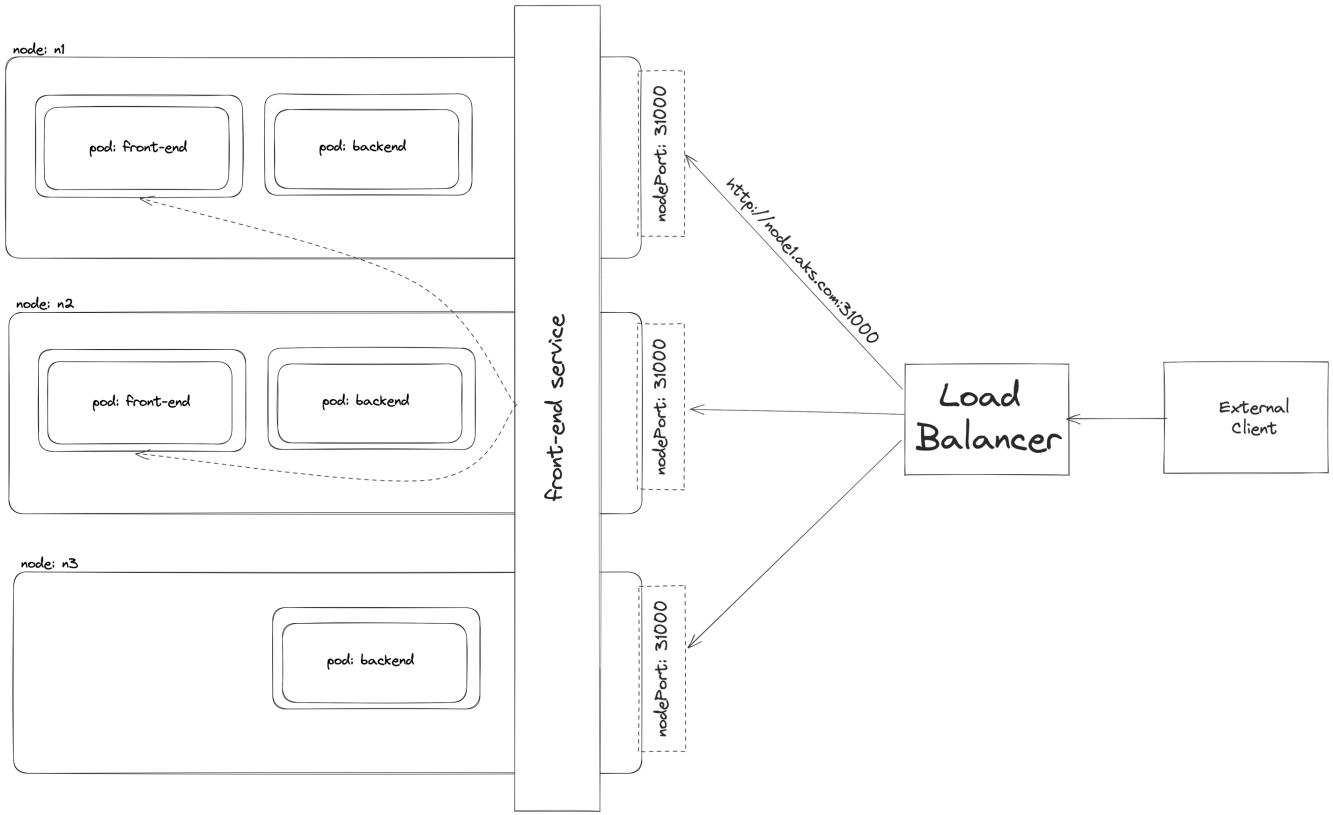
```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: NodePort
  selector:
    app.kubernetes.io/name: frontend
  ports:
    - port: 80
      targetPort: 8080
      # nodePort is exposed on all nodes in the cluster
      nodePort: 31000
```


LoadBalancer

Provisions an **external** load balancer for your service (based on the cloud provider)

> Builds on top of NodePort to expose pods internally

> Administrators can configure scaling, firewall, routing, etc. on the external load balancer



LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app.kubernetes.io/name: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```


Services without selector

How to access services that run outside the cluster?

> Use service discovery via DNS for external resources

> External database in production, internal in testing

```yaml
apiVersion: v1
kind: Service
metadata:
  name: external-database
spec:
  # Remember, no selectors!
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```yaml
apiVersion: discovery.k8s.io/v1
kind: EndpointSlice
metadata:
  name: external-database-1 # by convention, use the name of the Service
                            # as a prefix for the name of the EndpointSlice
  labels:
    # You should set the "kubernetes.io/service-name" label.
    # Set its value to match the name of the Service
    kubernetes.io/service-name: external-database
addressType: IPv4
ports:
  - name: ''
    appProtocol: http
    protocol: TCP
    port: 9376
endpoints:
  - addresses:
      - "10.4.5.6" # the IP addresses in this list can appear in any order
      - "10.1.2.3"
```


Agenda

Container-to-Container communication

Pod-to-Pod communication

Service discovery

Looking beyond the cluster

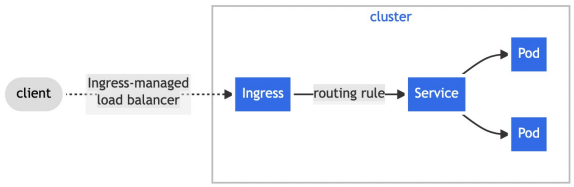
What next?

HTTP load balancing with Ingress

Service object operates at L4, how to route based on HTTP (L7)?

Ingress: Kubernetes native way to do “virtual hosting” – host many sites on a single IP.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - pathType: Prefix
        path: "/bar"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: "*.foo.com"
    http:
      paths:
      - pathType: Prefix
        path: "/foo"
        backend:
          service:
            name: service2
            port:
              number: 80
```



[2] What is ingress?

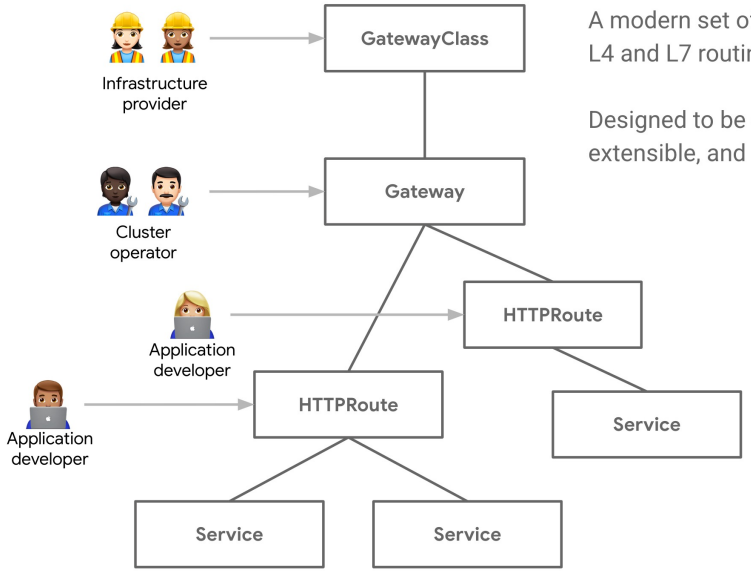
Gateway API

While Ingress is a useful abstraction, it has not scaled very well.

> Some features are underdefined, implementations may vary, reducing portability.

> Easy to misconfigure it.

Gateway API (being developed by SIG-network) is working on a modern API for L7 (and even L4) routing.



A modern set of APIs for deploying L4 and L7 routing in Kubernetes

Designed to be generic, expressive, extensible, and role-oriented

[3]: What is the Gateway API?

More topics

Kube-proxy

CNI types and implementations

Endpoints and EndpointSlices

Ingress Controllers

Service Meshes

References

[1] https://kubernetes.io/docs/concepts/workloads/pods/#pod-networking

[2] https://kubernetes.io/docs/concepts/services-networking/ingress/#what-is-ingress

[3] https://gateway-api.sigs.k8s.io/#what-is-the-gateway-api

Thank you!