KubeCon | CloudNativeCon

Europe 2023

KubeCon | CloudNativeCon

Europe 2023

SIG-Network: Intro and Updates

Antonio Ojea, Google

Shane Utt, Kong

Surya Seetharaman, Red Hat

Agenda

- Networking APIs
  - Service, EndpointSlice
  - Ingress, Gateway API, Service Mesh
  - NetworkPolicy, Admin Network Policy
- Networking Components (Kube-Proxy)
- Features in development
  - Incubation
  - Alpha
  - Beta
  - GA

Networking API - Service

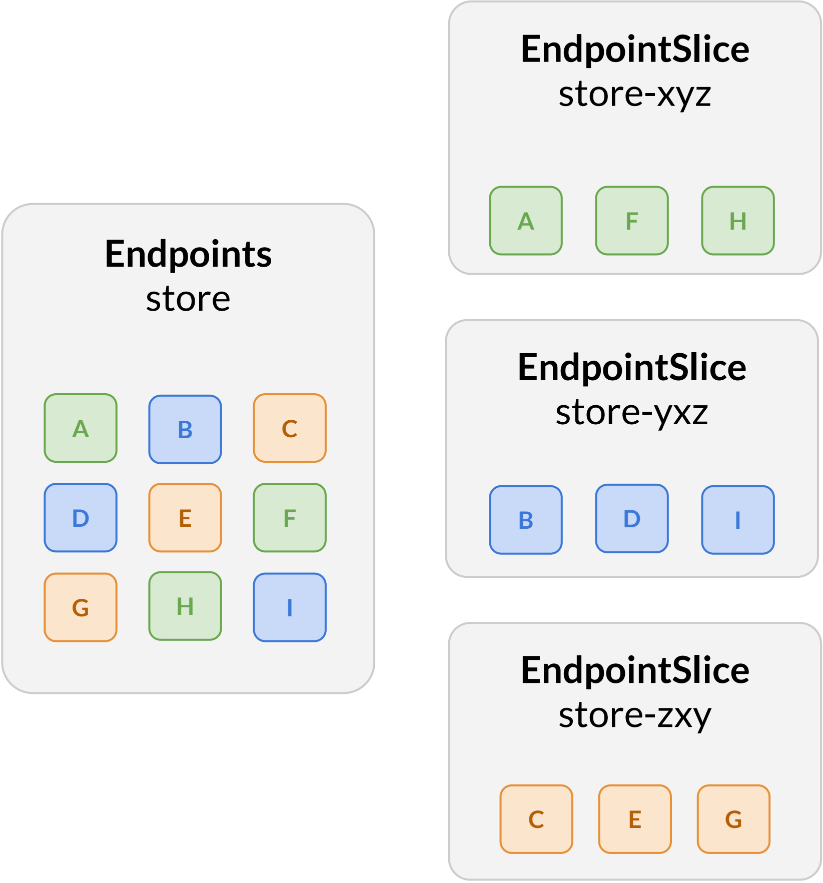
Service

- Enables grouping **Pods** together and exposing as a network Service
- Services are assigned **IP address(es)** they can be reached on
- Requests to those addresses will be **routed** to one of the associated Pods (via **endpoints**)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: store
spec:
  selector:
    app: store
  ports:
  - name: tcp
    protocol: TCP
    port: 80
    targetPort: tcp
```

Endpoints, EndpointSlice

- Track IPs and Ports for Pods backing a Service
- **Endpoints** was limited to 1000 Pods per Service
- The newer **EndpointSlices** are sharded Endpoints, much more scalable
- Features enabled by **EndpointSlices**:
  - Dual Stack
  - Topology
  - Terminating Endpoints



KubeCon | CloudNativeCon

Europe 2023

Networking API - Ingress

Ingress

- Host and Path Matching
- Forward to Service
- TLS Configuration
- Stable for 5+ years
- **Simple and broadly implementable**

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
spec:
  ingressClassName: acme
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Limitations

- Many **non-portable extensions** among 22+ implementations
  - Leading to **annotations everywhere**
- Insufficient permission model
- Mainly focused on **HTTP(S) traffic**
- Limited to North/South traffic

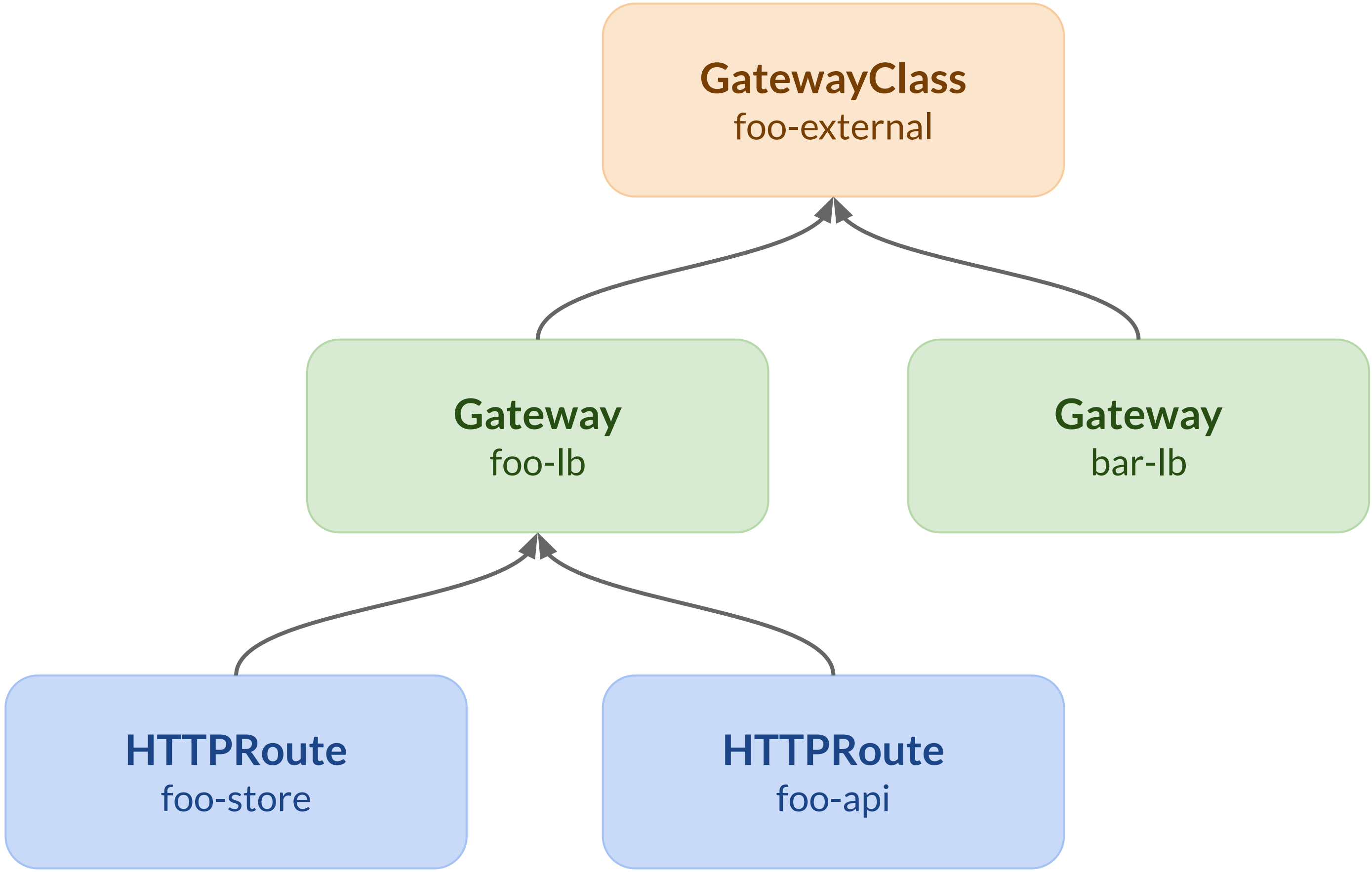
KubeCon | CloudNativeCon

Europe 2023

Networking API - Gateway

Gateway API

- Next generation of Kubernetes routing and load balancing APIs
- Designed to be **expressive** and **extensible**
- Role oriented resource model
- 20+ implementations
  - (and 3 integrations)
- Graduated to beta last year
- Aiming for **GA this year**!



API Types

Non-Routes

- GatewayClass
- Gateway
- ReferenceGrant

Routes

- HTTPRoute
- GRPCRoute
- TCPRoute
- UDPRoute
- TLSRoute

GatewayClass Example

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: GatewayClass
metadata:
  name: acme
spec:
  controllerName: kubernetes.io/acme
```

Gateway Example

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: Gateway
metadata:
  name: acme-gateway-1
spec:
  gatewayClassName: acme
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```

HTTPRoute Example

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: demo
spec:
  parentRefs:
  - name: acme-gateway-1
  rules:
  - backendRefs:
    - name: demo
      port: 80
```

Simple Path Match

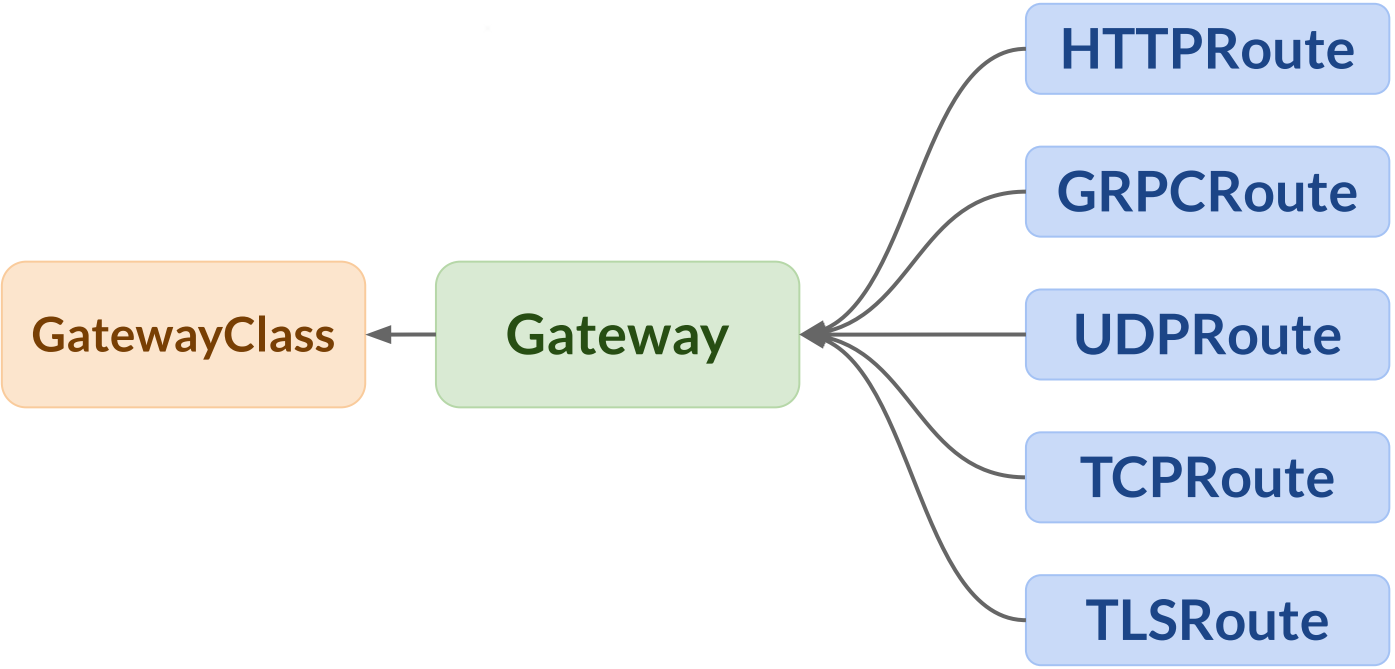
Ingress

```yaml
ingressClassName: acme
rules:
- http:
    paths:
    - path: /login
      pathType: Prefix
      backend:
        service:
          name: demo
          port:
            number: 8080
```

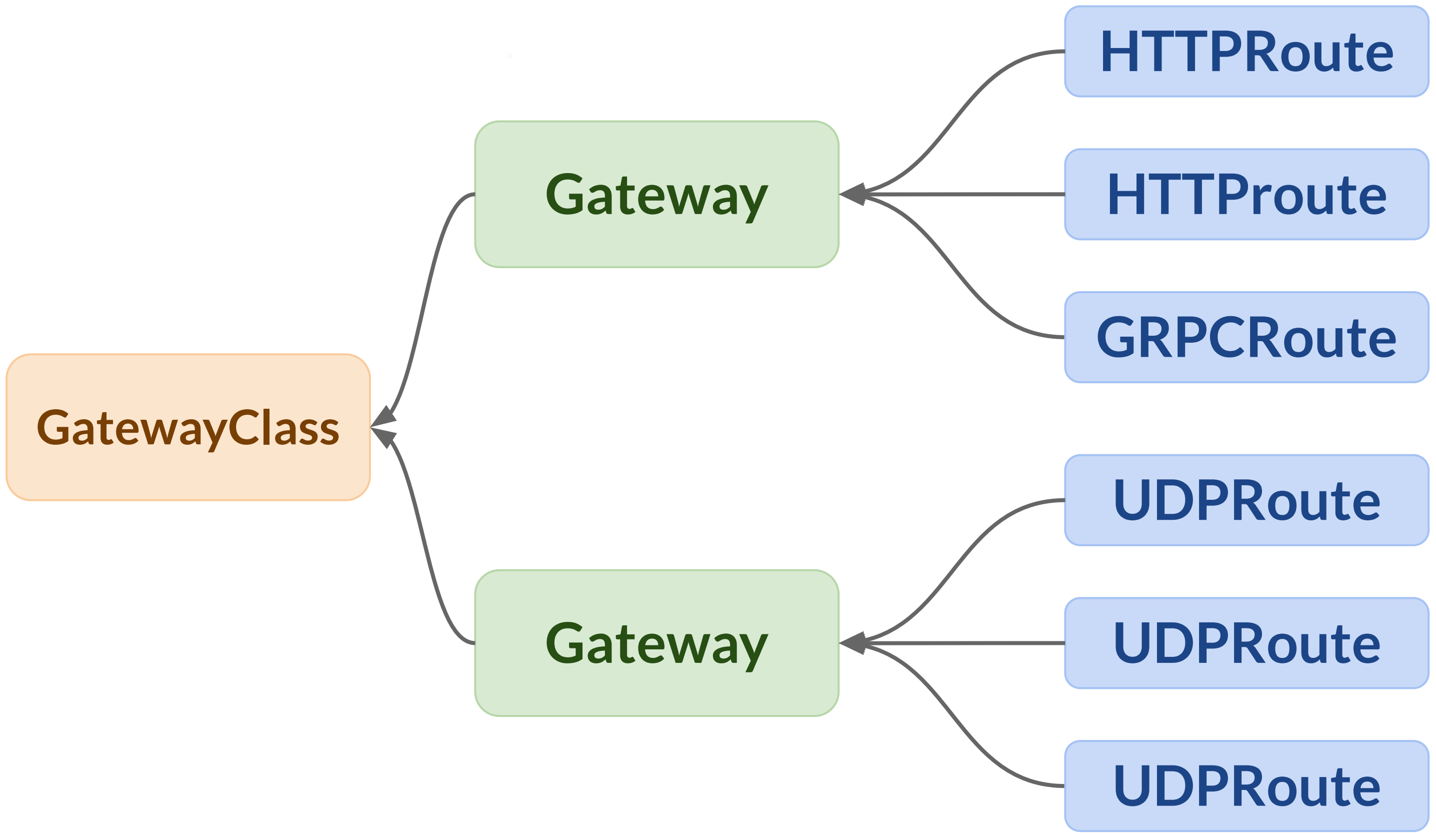
HTTPRoute

```yaml
parentRefs:
- name: acme-gateway-1
rules:
- matches:
  - path:
      type: PathPrefix
      value: /login
  backendRefs:
  - name: demo
    port: 8080
```

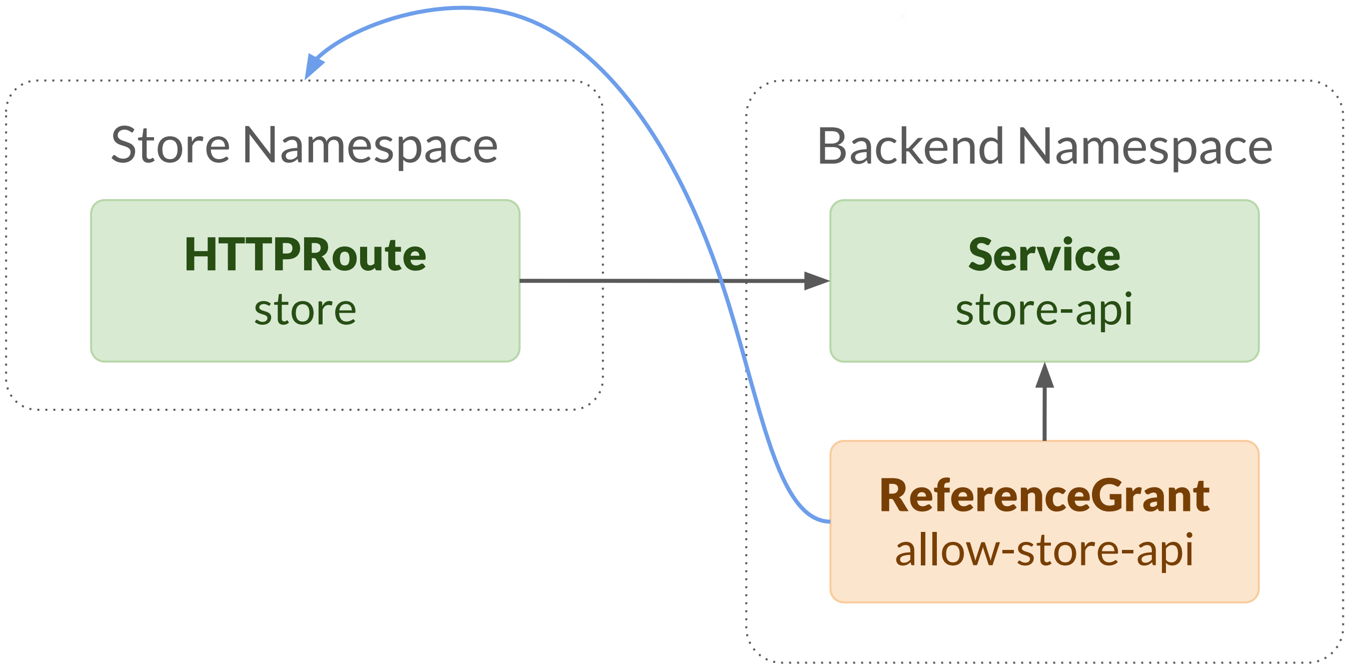
Attaching Routes



Attaching Routes (cont.)



ReferenceGrant



```yaml
kind: ReferenceGrant
metadata:
  name: allow-store-api
  namespace: backend
spec:
  from:
  - kind: HTTPRoute
    namespace: store
  to:
  - group: ""
    kind: Service
    name: store-api
```

KubeCon | CloudNativeCon

Europe 2023

Networking API - Service Mesh

GAMMA Project

- **Gateway API** for **Mesh Management** and **Administration**
- Using Gateway API for east/west traffic in a service mesh context
  - **HTTPRoute** currently being experimented with
- 6+ implementations involved in the project
- Initial experimental conformance tests just landed (using **HTTPRoute**)

Follow up with Gateway API

gateway-api.sigs.k8s.io

#sig-network-gateway-api

kubernetes-sigs/gateway-api

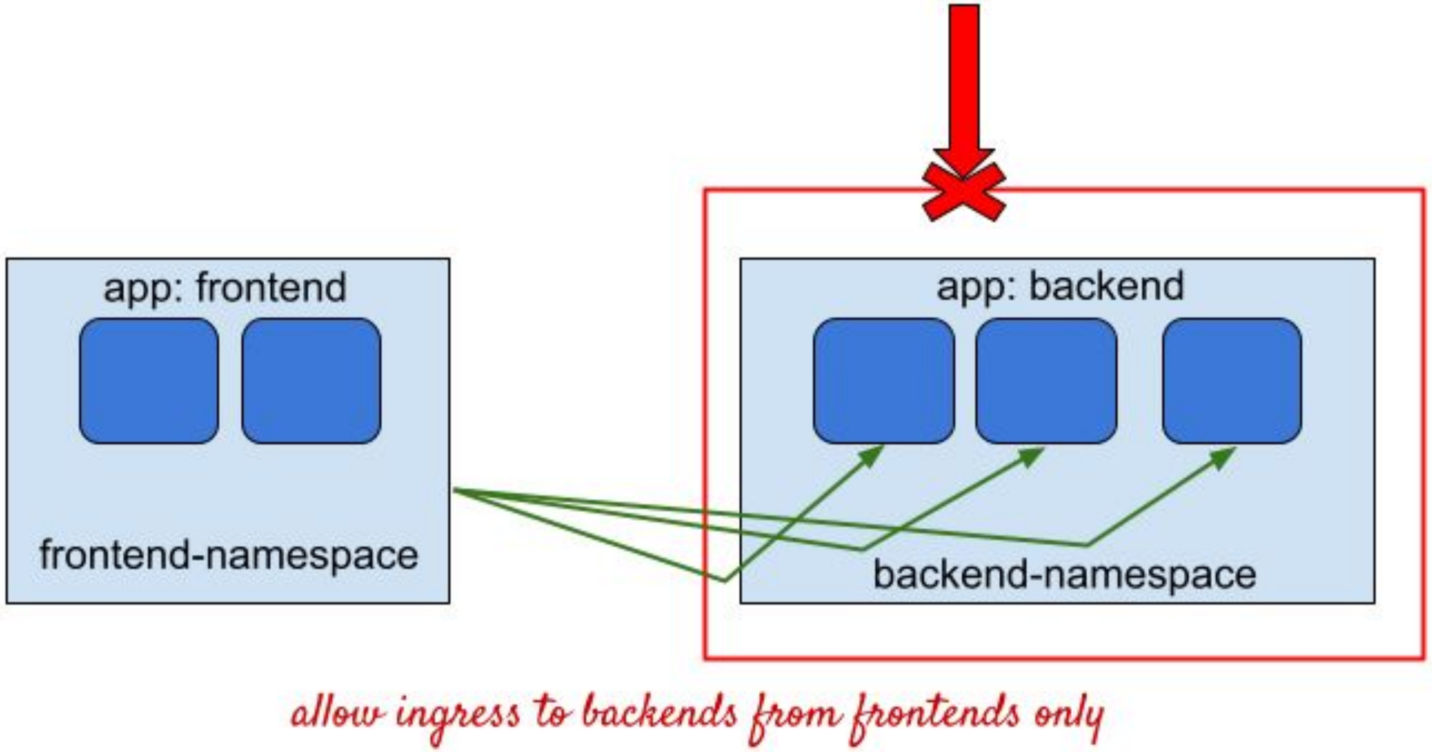
Network Policy

Contributions welcome! Stable for over 5 years.

network-policy-api && https://kubernetes.io/docs/concepts/services-networking/network-policies/

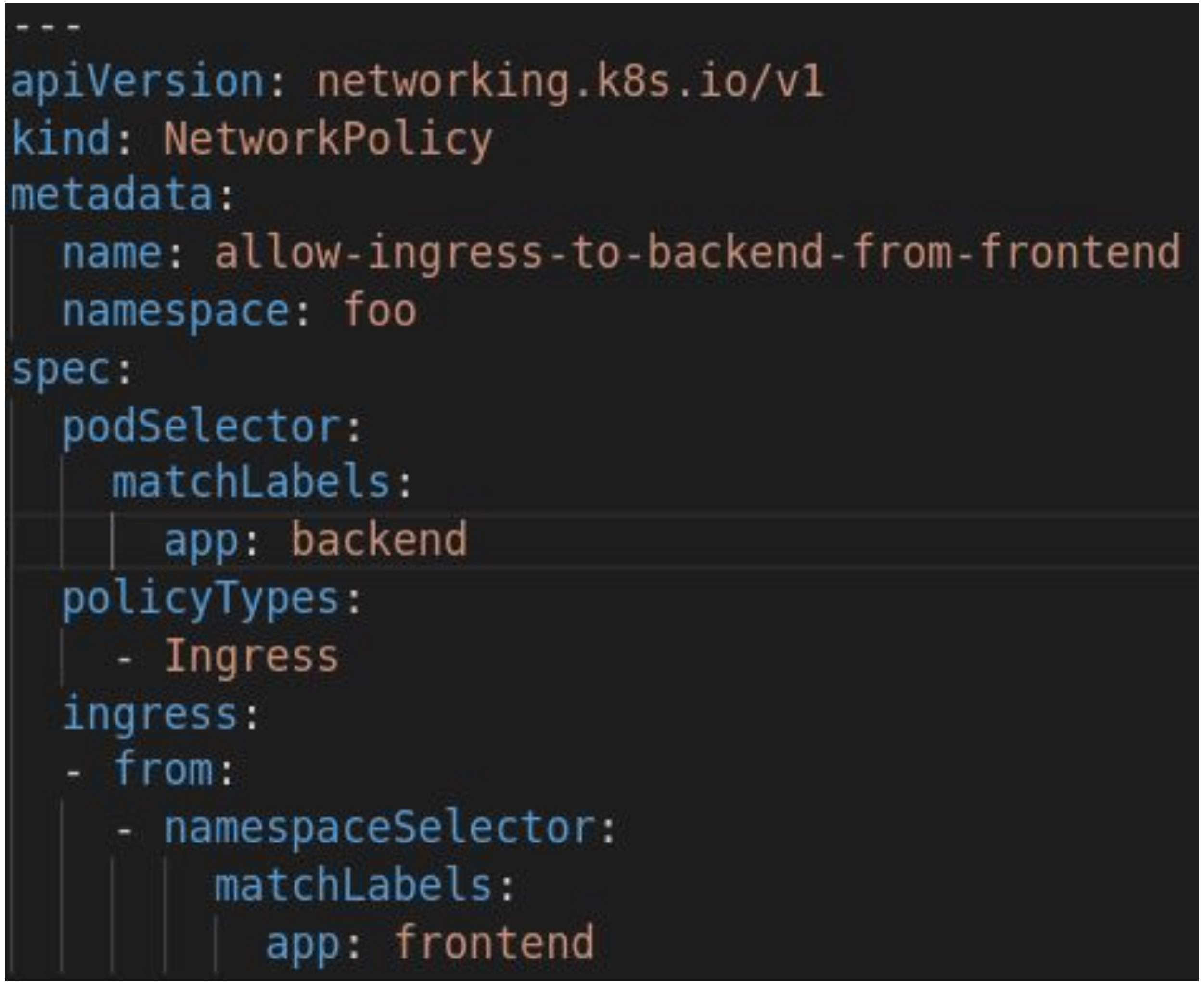
Network Policy API

- How can app owners control traffic to/from their workloads?
  - example; backends can get traffic only from frontends, databases can only get traffic from backends etc..



Network Policy API

- How can app owners control traffic to/from their workloads?
  - example; backends can get traffic only from frontends, databases can only get traffic from backends etc..
- An API that let’s users define simple ingress/egress rules
- API design is implicit in nature
- Network policy peers
  - pod, namespace, ipBlock



```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-to-backend-from-frontend
  namespace: foo
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          app: frontend
```

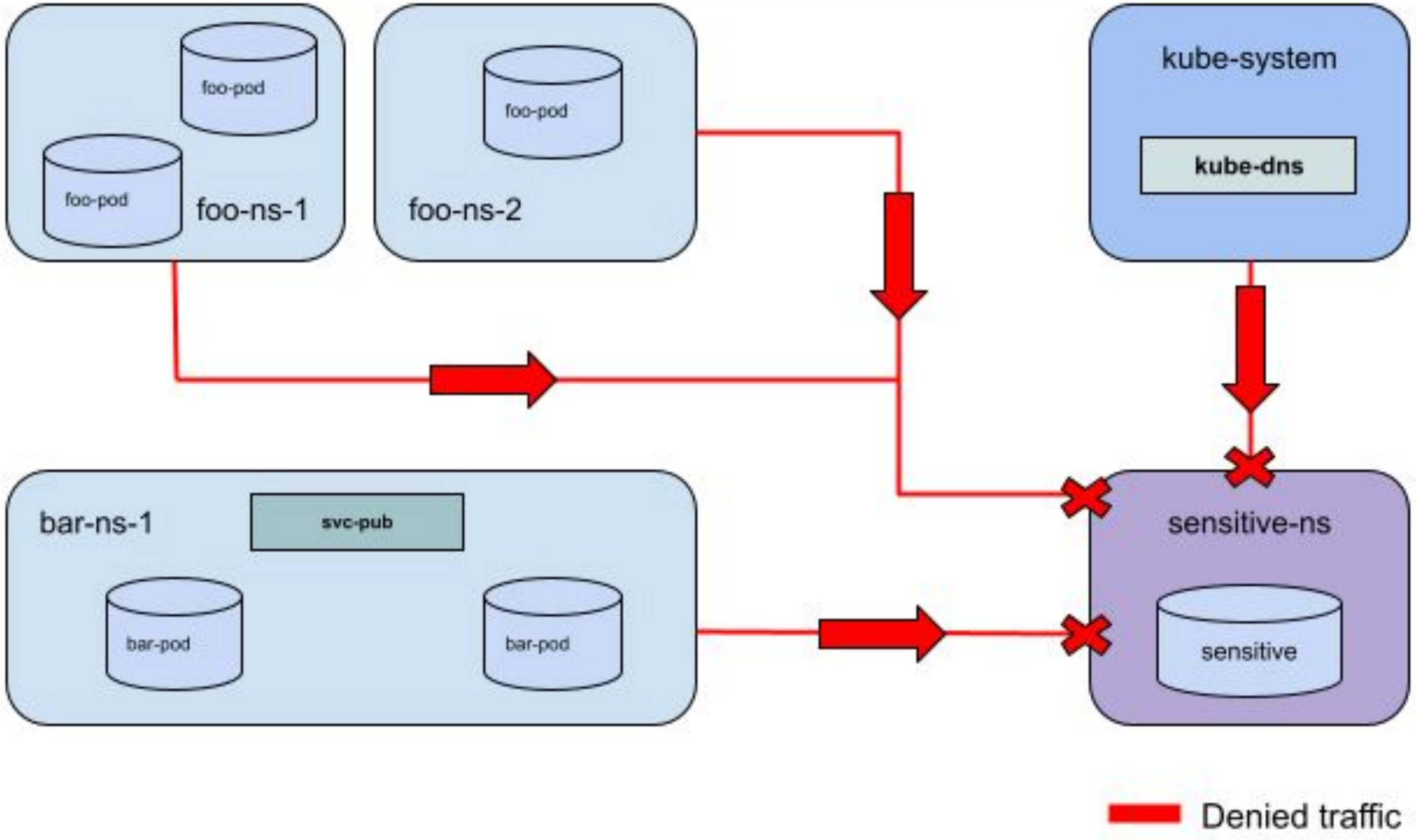
Admin Network Policy

Contributions welcome! Under active development!

https://github.com/kubernetes-sigs/network-policy-api && https://network-policy-api.sigs.k8s.io/

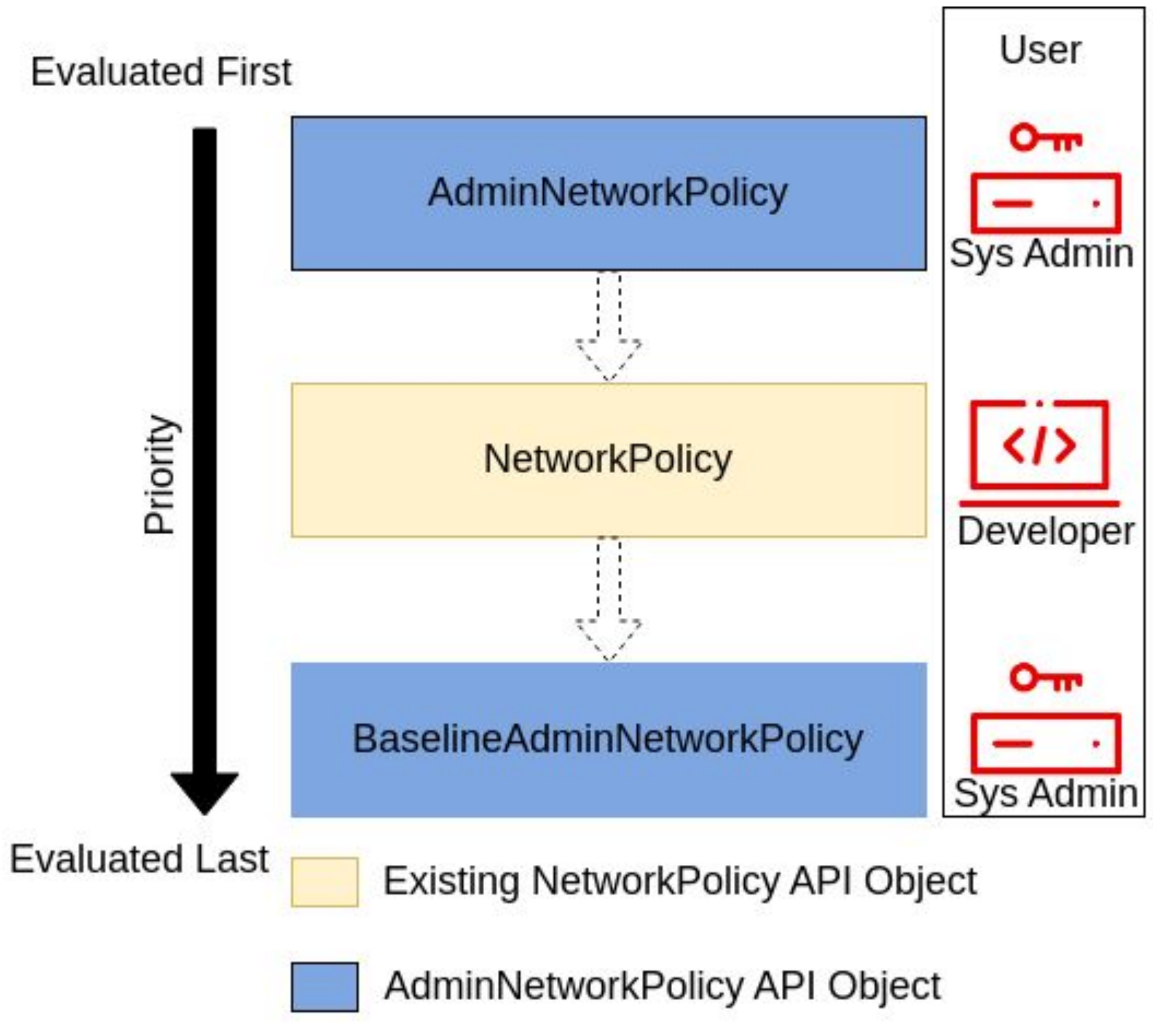
Admin Network Policy API

- Network Policies were designed for app owners…
- How can admins enforce policies cluster-wide??



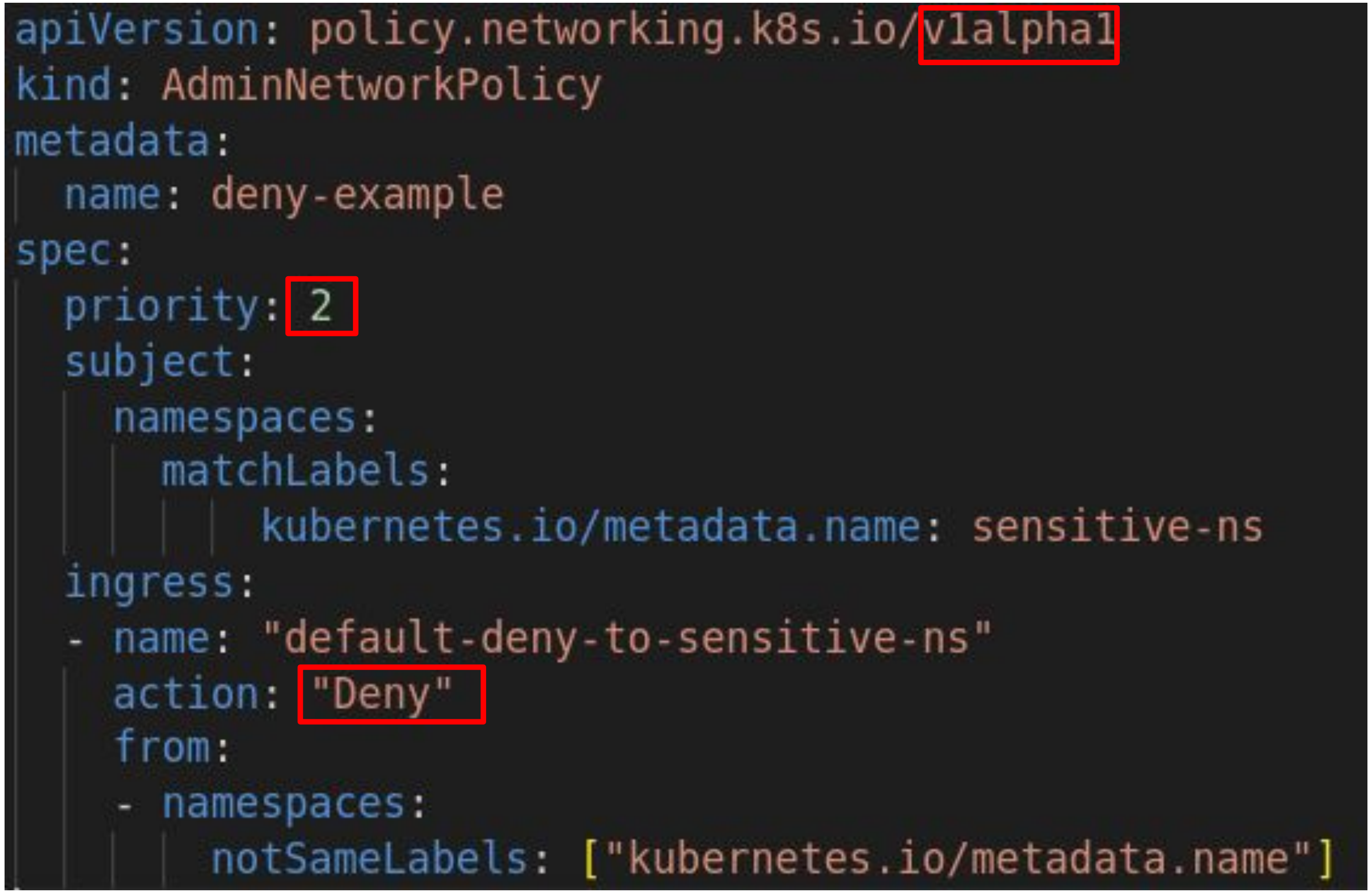
Admin Network Policy API

- Network Policies were designed for app owners…
- How can admins enforce policies cluster-wide??
- Cluster-scoped policy API
  - AdminNetworkPolicy
  - BaselineAdminNetworkPolicy
- API design is explicit in nature



Admin Network Policy API

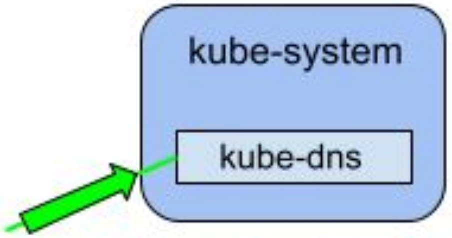
- Network Policies were designed for app owners...
- How can admins enforce policies cluster-wide??
- Cluster-scoped policy API
  - AdminNetworkPolicy
  - BaselineAdminNetworkPolicy
- API design is explicit in nature
- v1alpha1 supports east-west traffic
- Network policy peers
  - Pods, namespaces



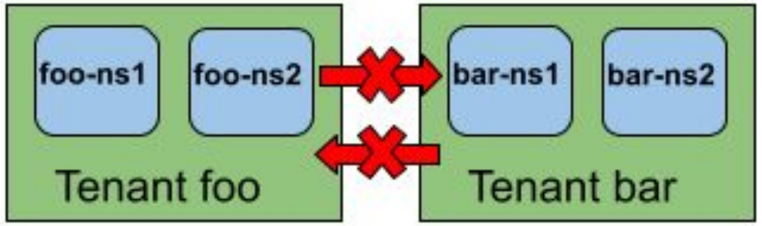
```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: deny-example
spec:
  priority: 2
  subject:
    namespaces:
      matchLabels:
          kubernetes.io/metadata.name: sensitive-ns
  ingress:
  - name: "default-deny-to-sensitive-ns"
    action: "Deny"
    from:
    - namespaces:
        notSameLabels: ["kubernetes.io/metadata.name"]
```

Admin Network Policy API

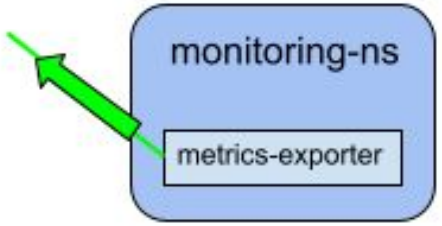
- Aiming for Beta this year!
- WIP: north-south traffic support
    - Support for ANP around northbound traffic
    - Support for ANP around host-networked backends
- Implementations in progress…
    - End user?
    - Have use cases?
    - Want to contribute?
    - Have feedback?
    - Join us!



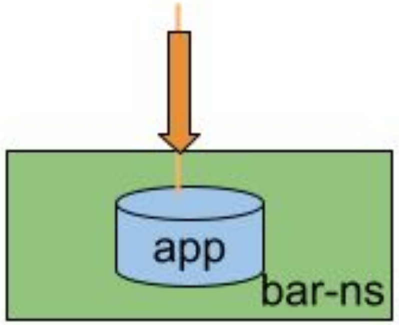
always ingress to dns namespace



isolate multi-tenants



always egress from monitoring namespace



explicitly delegate to network policy rules in an app namespace

Get Involved!

- **Focus Areas:**
  - **Network Policies,**
  - **Admin Network Policies**
- Bi-Weekly community meetings
  - Tuesday’s 6PM CET/12noon ET/9AM PT
- We welcome all kinds of contributions from all backgrounds
  - we’re **especially looking for more end-users and feedback**!

network-policy-api.sigs.k8s.io/

#sig-network-policy-api

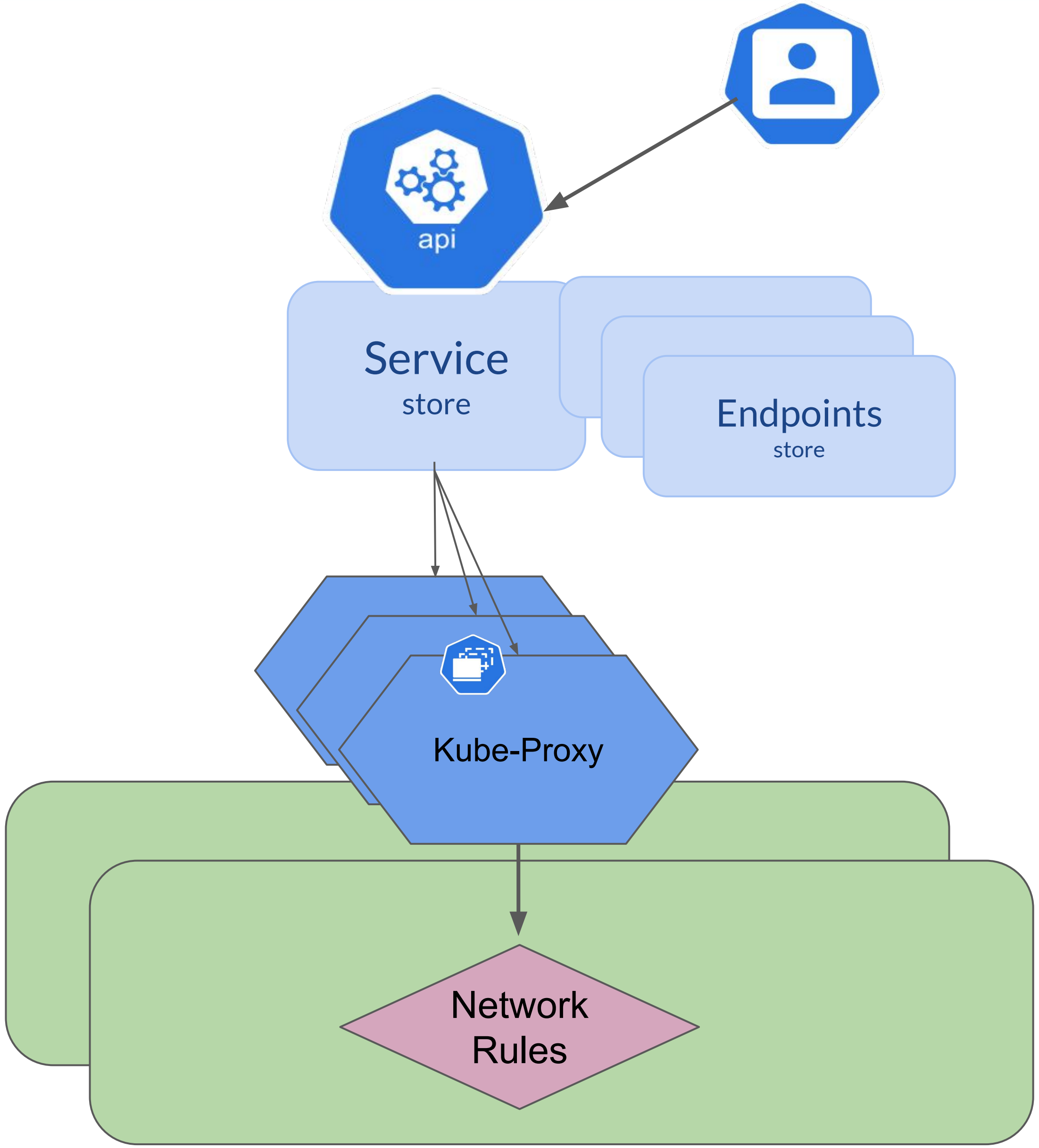
kubernetes-sigs/network-policy-api

Networking Components

Kube Proxy

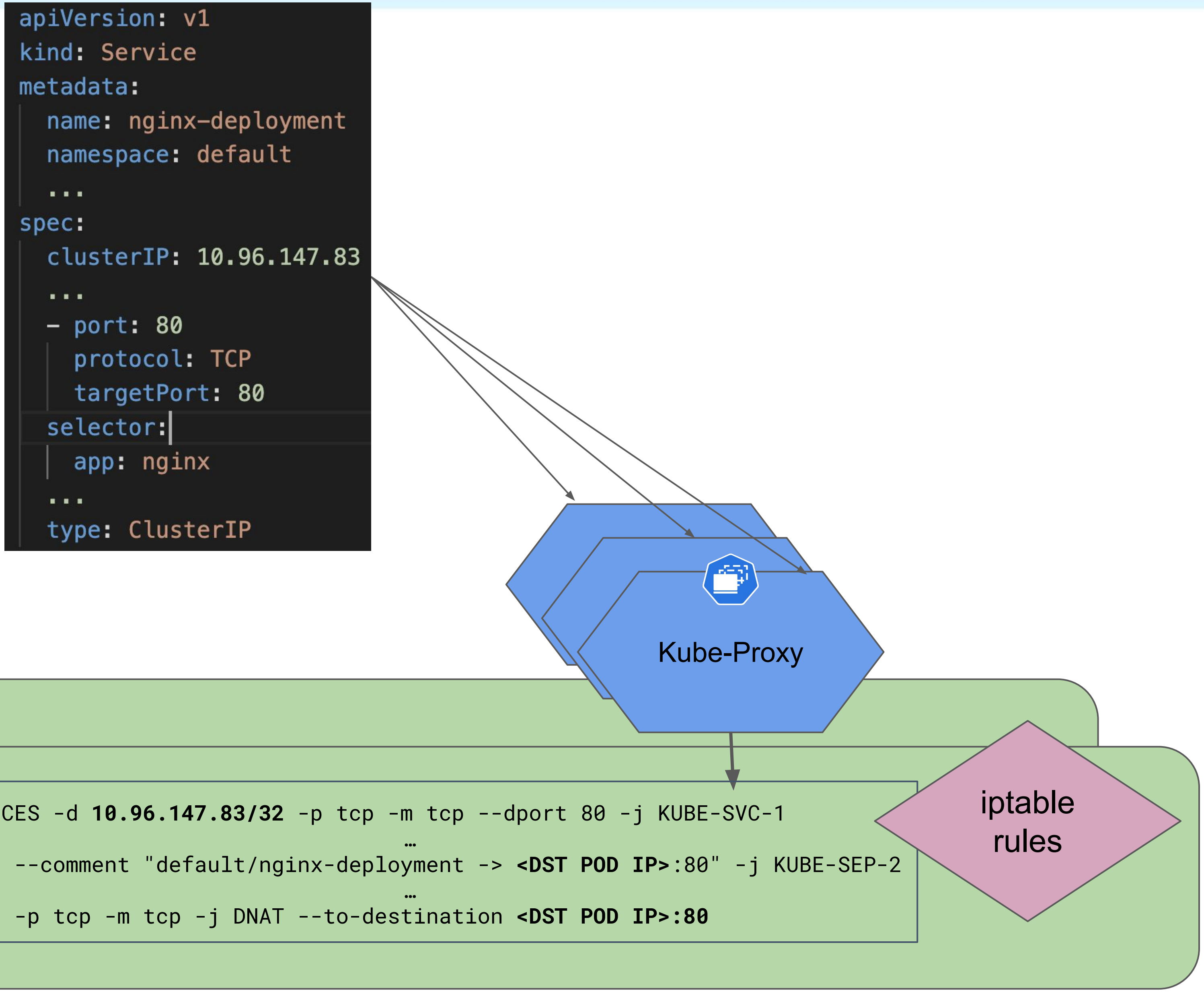
Kube Proxy

- We went over some core K8s Networking APIs: **Services** and **EndpointSlices**
- Kube-Proxy is the default implementation of service proxying in K8s
- It converts K8s Networking objects into rules



Kube Proxy

- Implemented in core K8s
- Can program rules using two modes/backends
  - Iptables (default)
  - ipvs

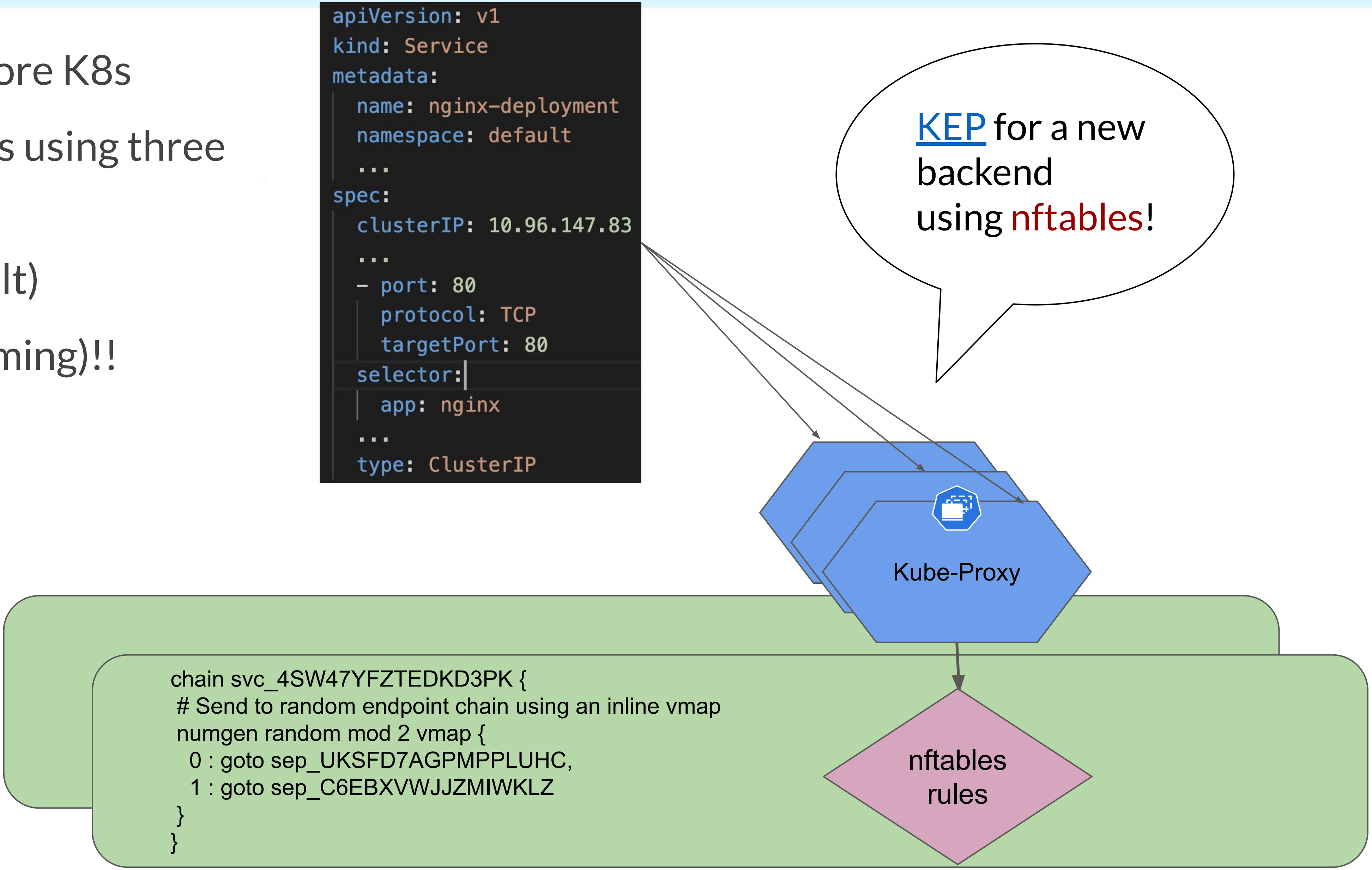


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  namespace: default
  ...
spec:
  clusterIP: 10.96.147.83
  ...
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  ...
  type: ClusterIP
```

```
-A KUBE-SERVICES -d 10.96.147.83/32 -p tcp -m tcp --dport 80 -j KUBE-SVC-1
…
-A KUBE-SVC-1 --comment "default/nginx-deployment -> <DST POD IP>:80" -j KUBE-SEP-2
…
-A KUBE-SEP-2 -p tcp -m tcp -j DNAT --to-destination <DST POD IP>:80
```

Kube Proxy

- Implemented in core K8s
- Can program rules using three modes/backends
  - Iptables (default)
  - nftables (upcoming)!!
  - ipvs



```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  namespace: default
  ...
spec:
  clusterIP: 10.96.147.83
  ...
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  ...
  type: ClusterIP
```

KEP for a new backend using nftables!

```
chain svc_4SW47YFZTEDKD3PK {
 # Send to random endpoint chain using an inline vmap
 numgen random mod 2 vmap {
   0 : goto sep_UKSFD7AGPMPPLUHC,
   1 : goto sep_C6EBXVWJJZMIWKLZ
 }
}
```

Updates / New KEPs - Antonio

SIG-NETWORK KEPs

https://github.com/orgs/kubernetes/projects/10

Inception

MultiNetwork

Kubernetes Pod network evolution

KubeProxy NFTables

KubeProxy improved ingress connectivity reliability (LoadBalancers)
Collaboration with SIG-Cloud-Provider

Alpha

Multiple Cluster-CIDRs

Assign multiple PodCIDRs to Nodes

Multiple Service-CIDRs

Reserve Service IP Ranges For Dynamic and Static NodePort Allocation

Admin network policy

Beta

Topology Aware Routing

Cleaning up iptables chain ownership (kubelet-kubeproxy)

Improve Performance Kube-proxy iptables

Expanded DNS configuration

GA

Gateway API*

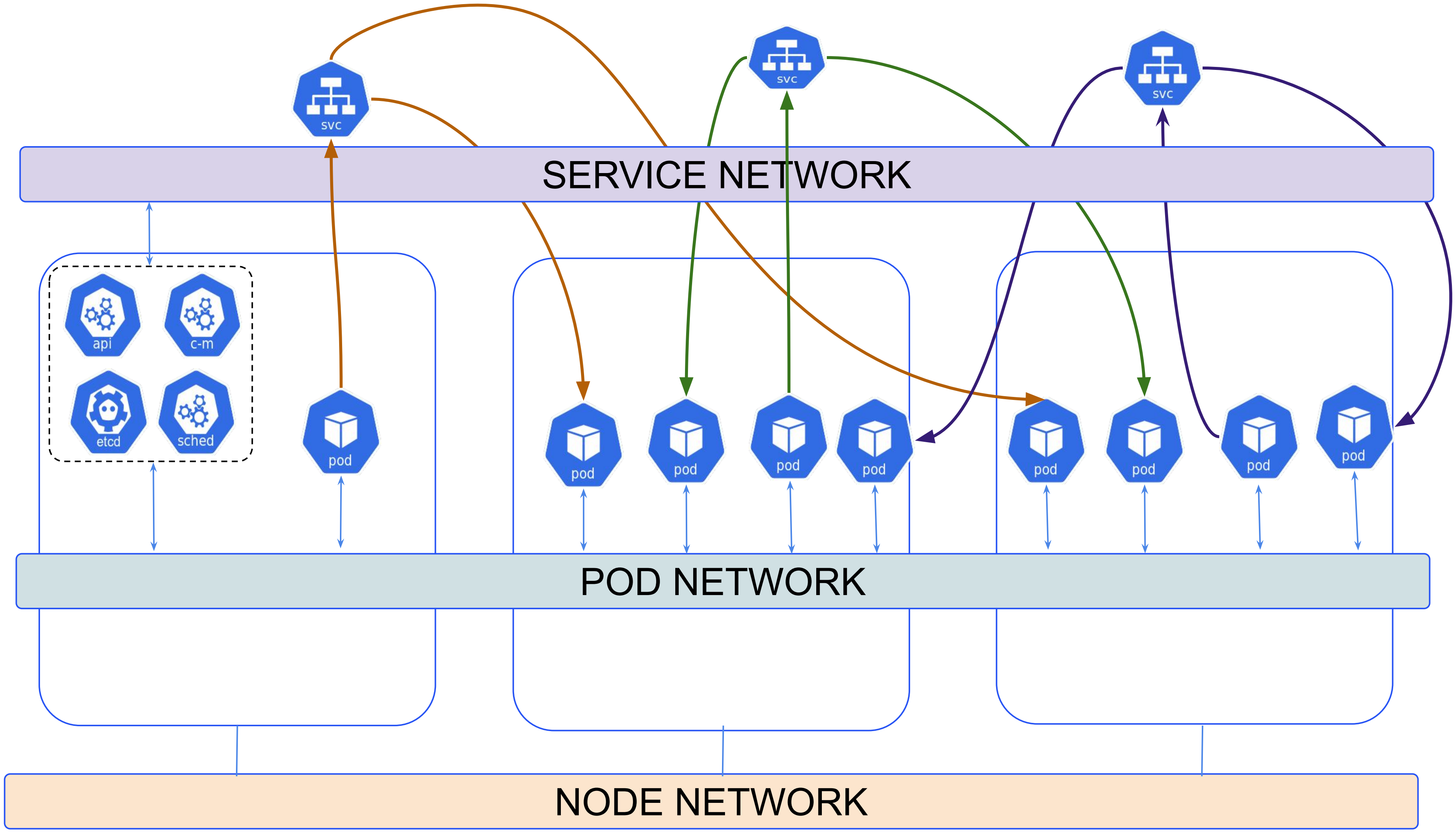
Kubernetes Service/Ingress evolution

Service Internal Traffic Policy

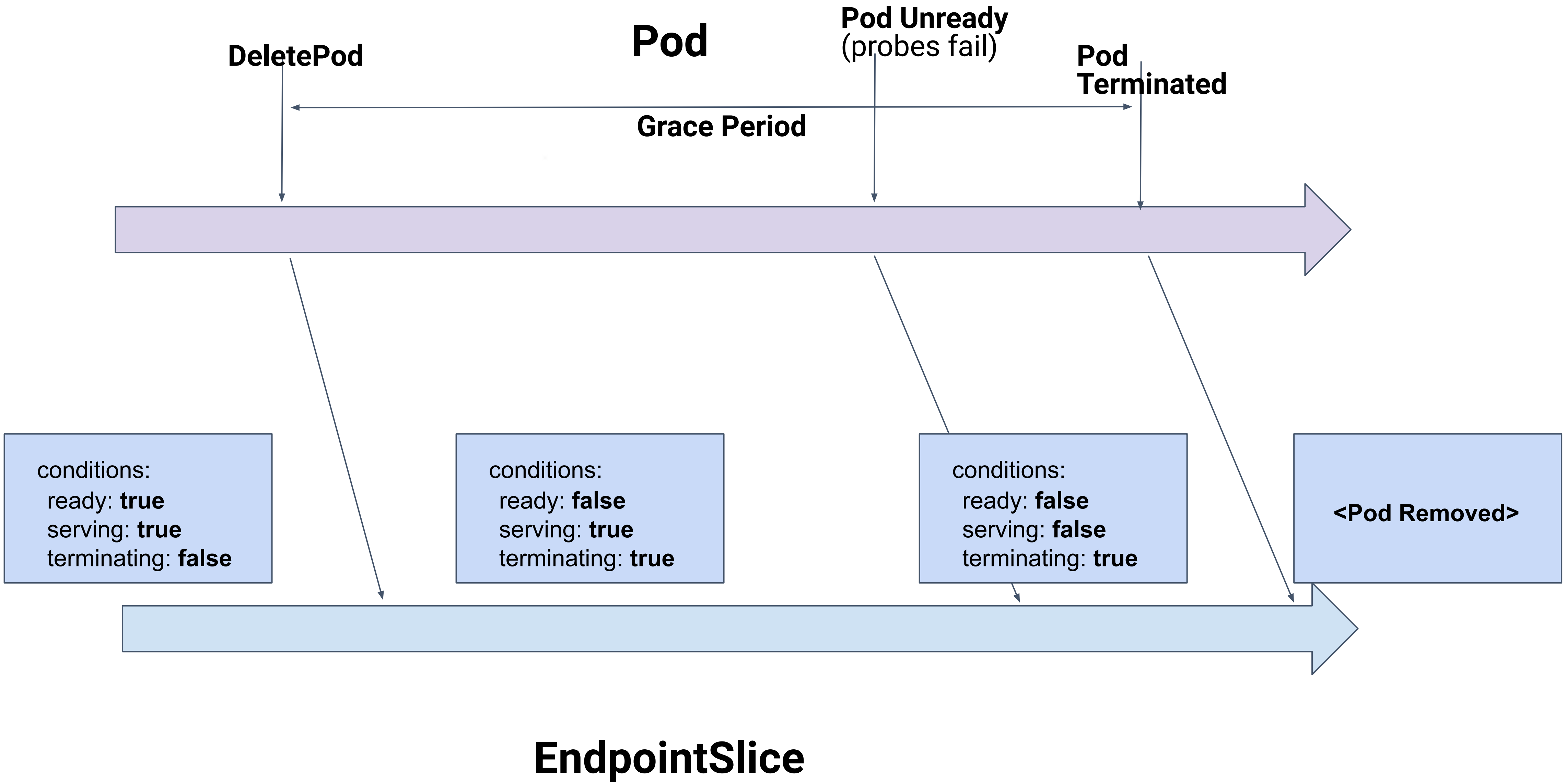
Reserve Service IP Ranges For Dynamic and Static IP Allocation

Terminating Endpoints

Services and Cluster CIDRs



Terminating Endpoints



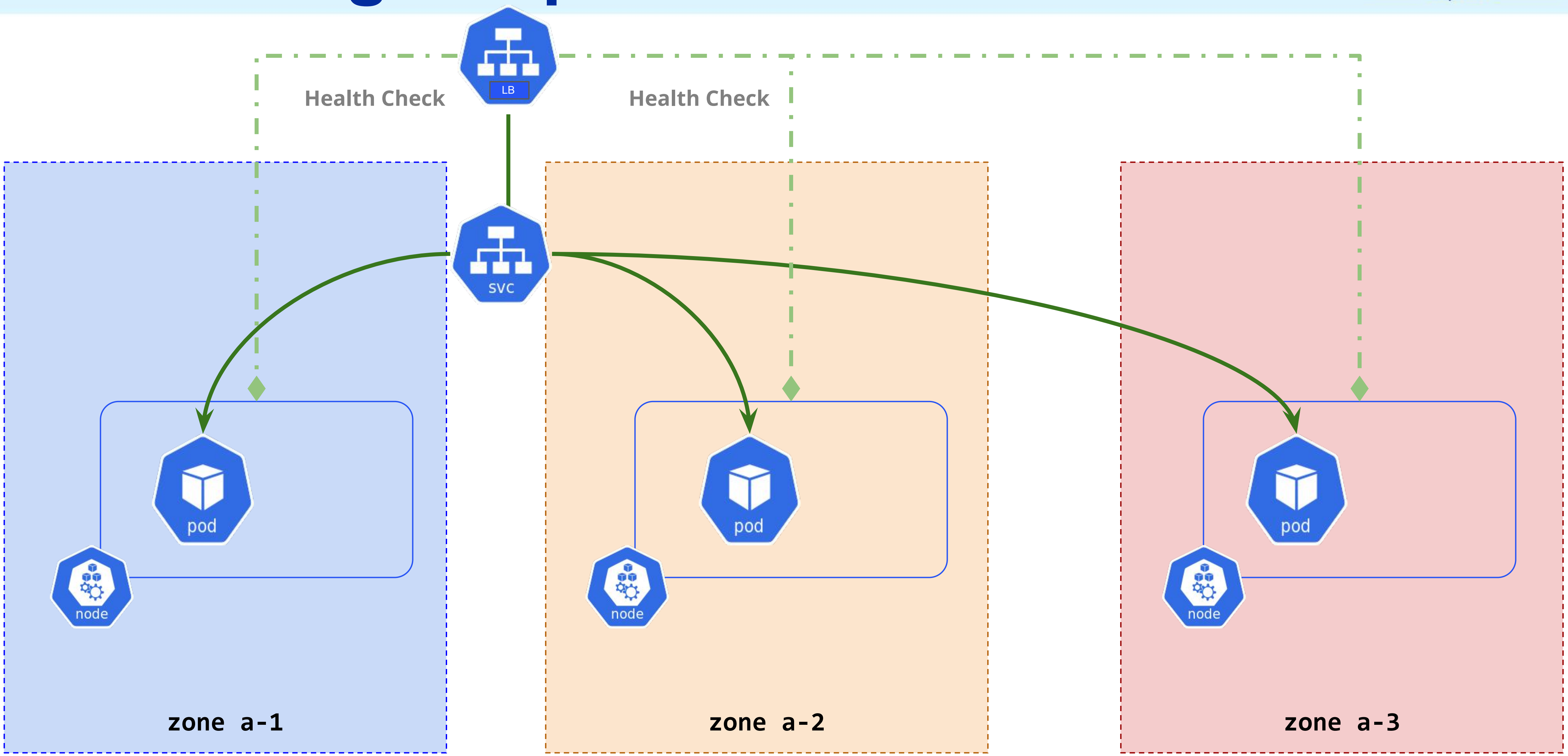
conditions:
ready: **true**
serving: **true**
terminating: **false**

conditions:
ready: **false**
serving: **true**
terminating: **true**

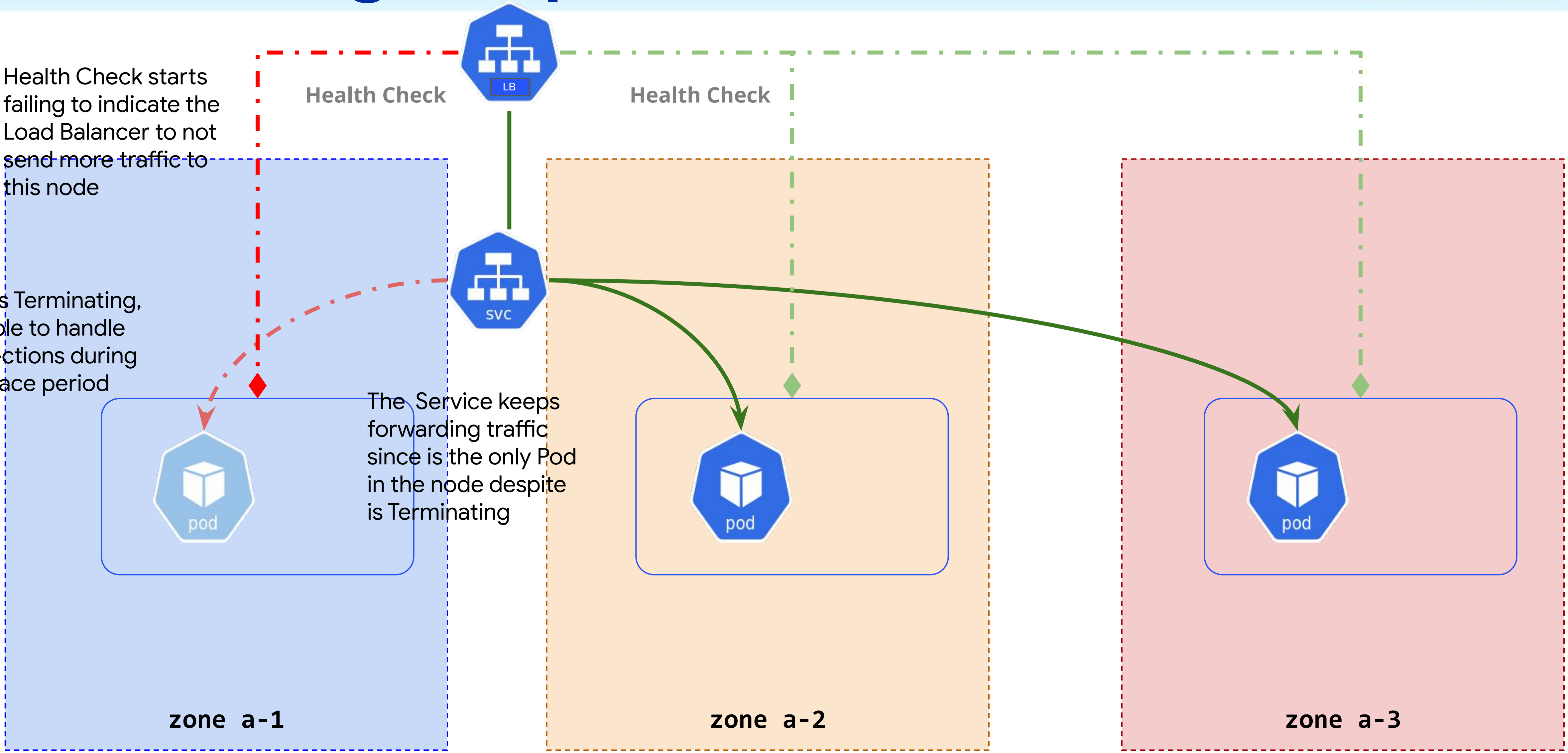
conditions:
ready: **false**
serving: **false**
terminating: **true**

<Pod Removed>

Terminating Endpoints



Terminating Endpoints



Health Check starts failing to indicate the Load Balancer to not send more traffic to this node

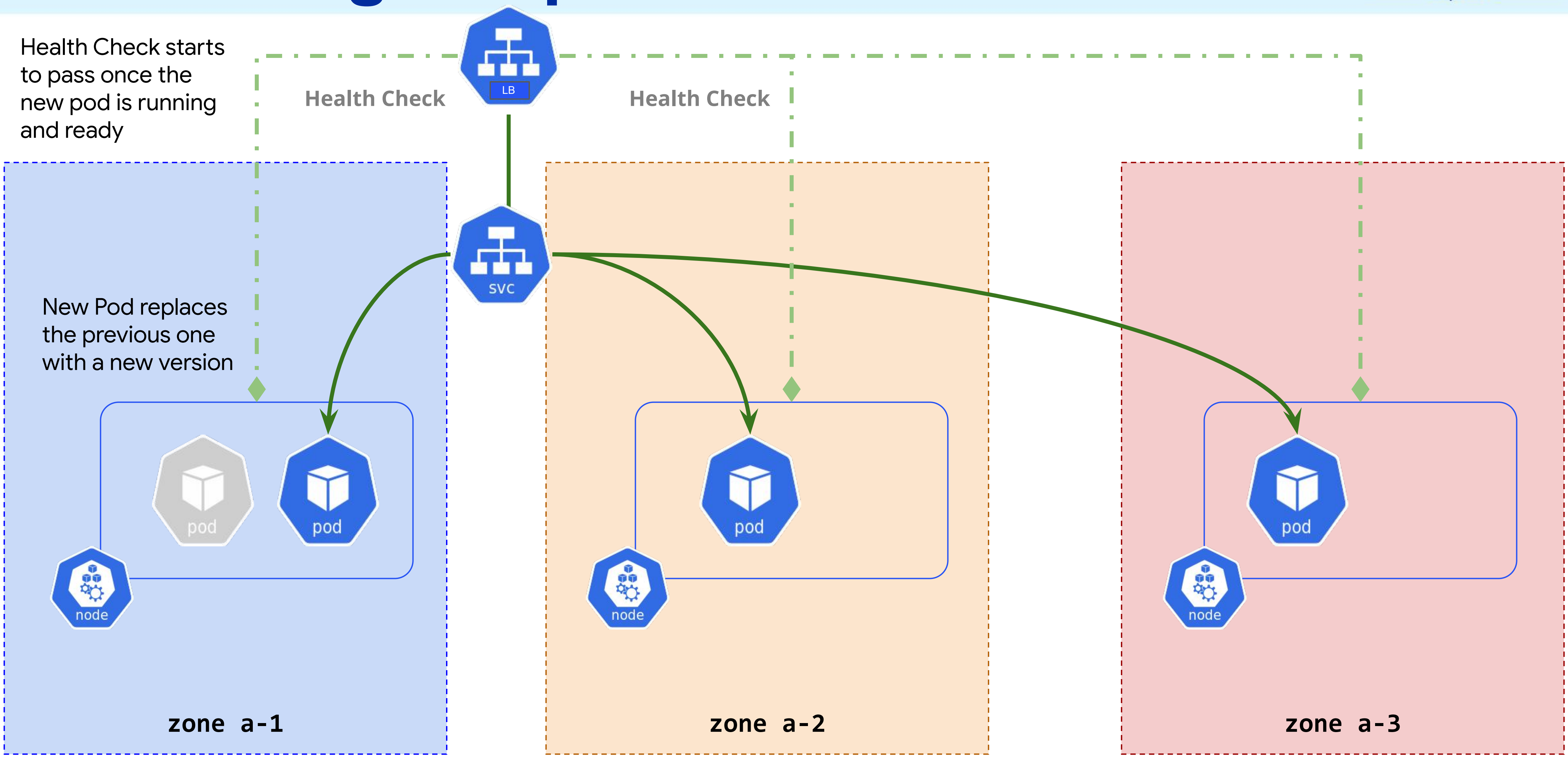
Pods is Terminating, is is able to handle connections during the grace period

The Service keeps forwarding traffic since is the only Pod in the node despite is Terminating

Terminating Endpoints

Health Check starts to pass once the new pod is running and ready

New Pod replaces the previous one with a new version

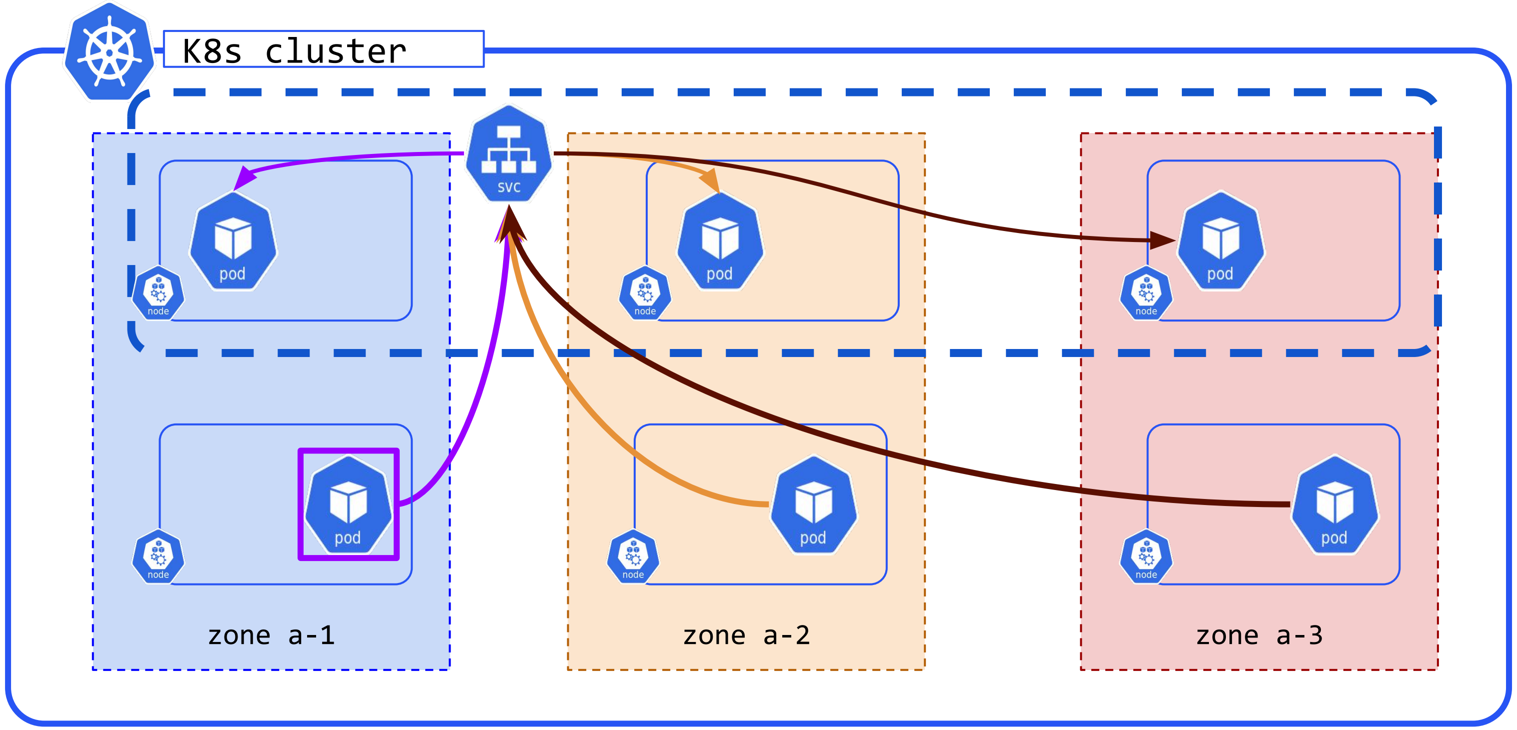


Topology Aware Routing: Prefer Zone

Symmetry is beautiful

Reasons:

- Economic
- Performance
- Latency



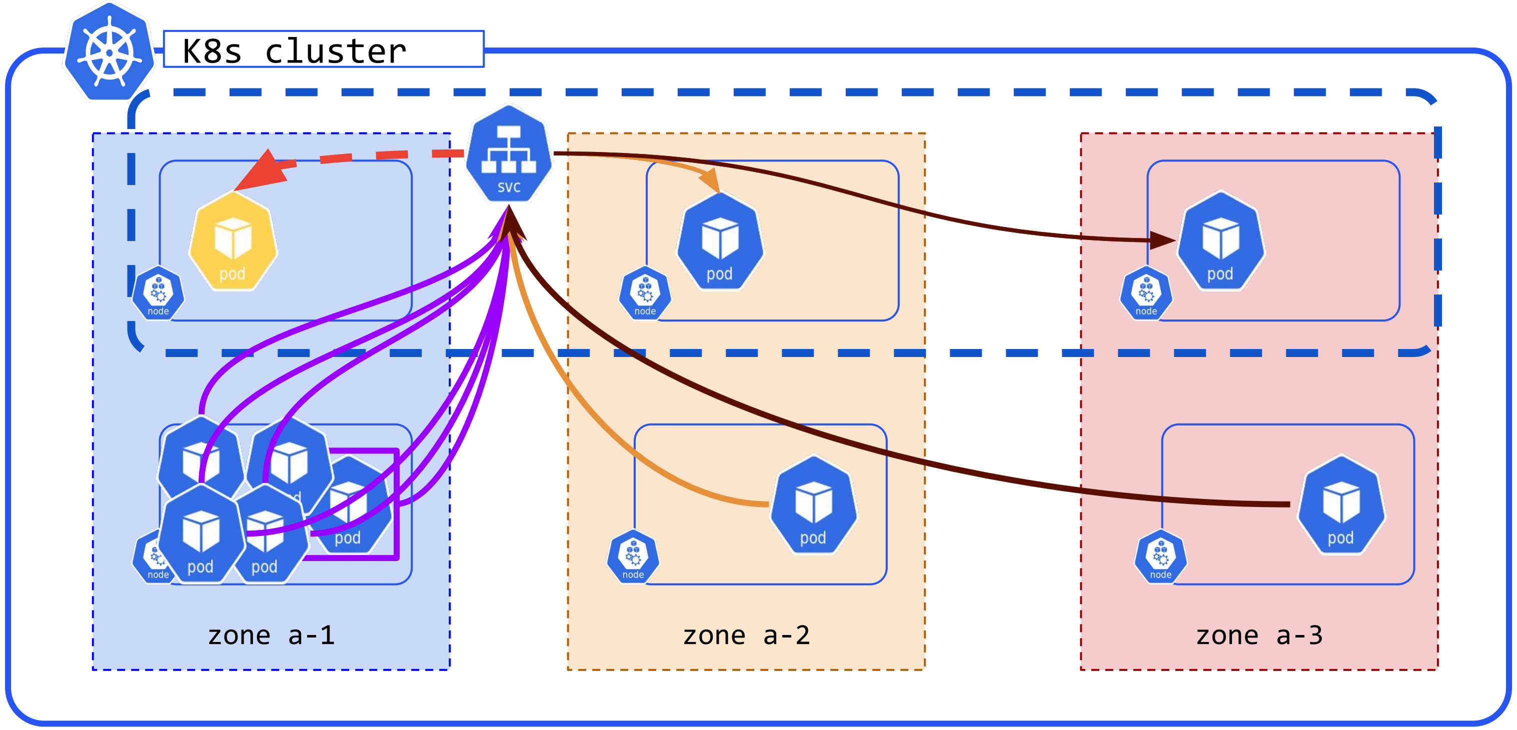
Topology Aware Routing: Problems

We can control the scheduling of the backends, can we do the same for the clients?

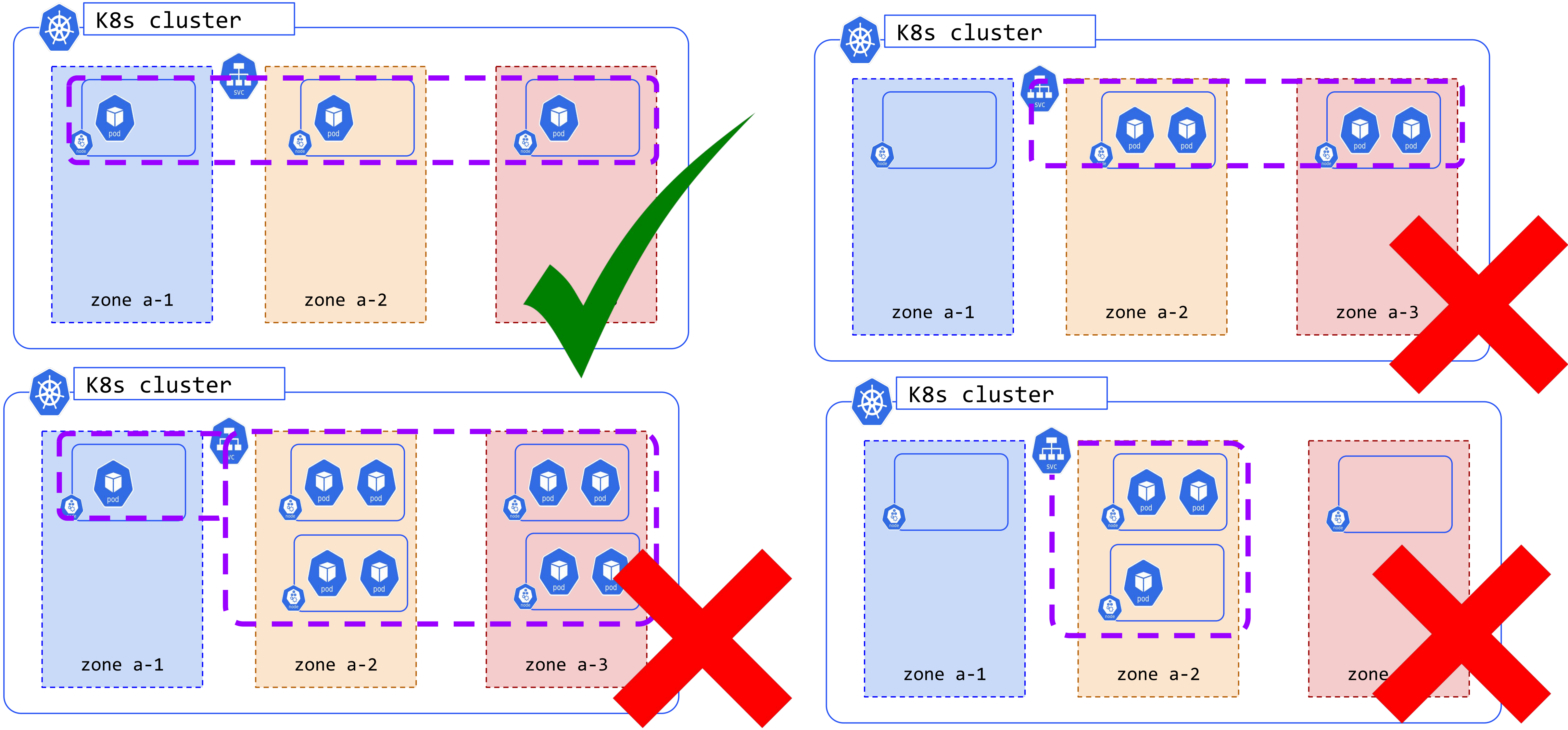
How the Service implementation knows an endpoint is saturated?

Symmetry is not the reality …

Anything that can go wrong will go wrong



Topology Aware Routing: scheduling



Join the community

[SIG Network README]()

#sig-network

Q & A

Please scan the QR Code above
to leave feedback on this session