Dimitar Zahariev

Technical Trainer

Kubernetes Network Policies

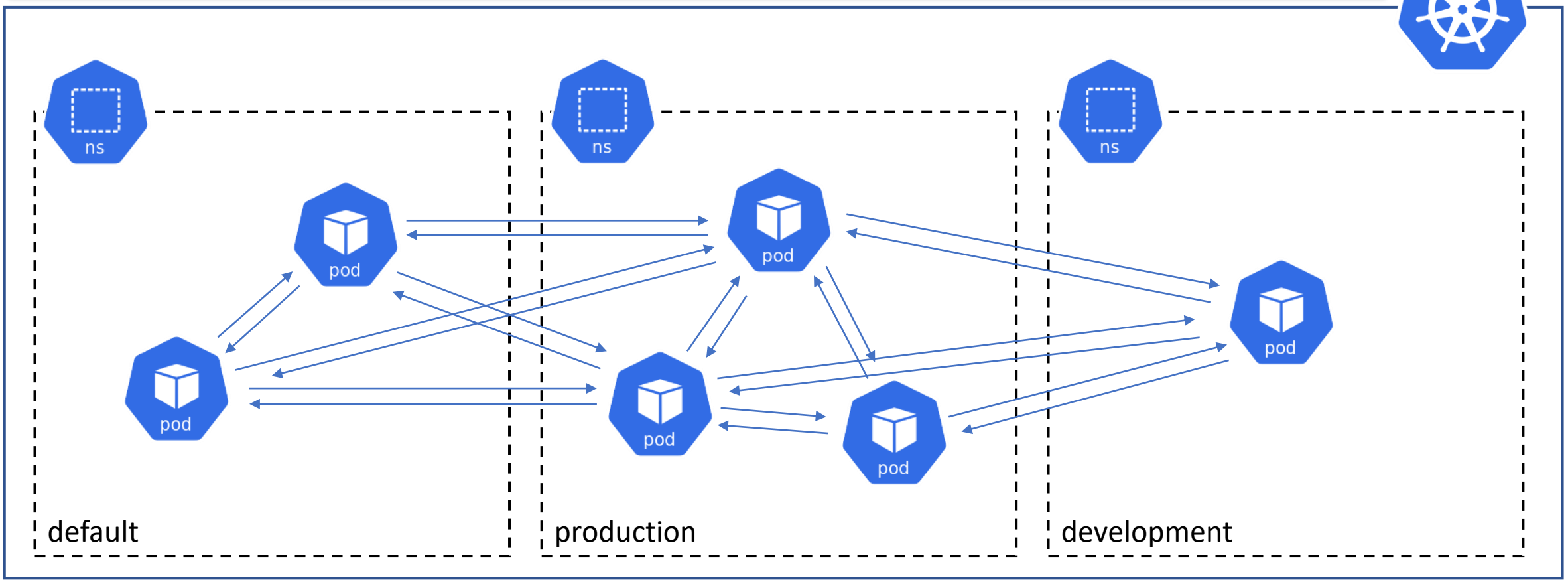
Agenda

- Kubernetes Network Policies
  - Introduction
  - Requirements
  - Components
- Demo
- Q&A Session

Kubernetes Network Policies

Before Network Policies

Quite messy. And don’t to mention the security implications

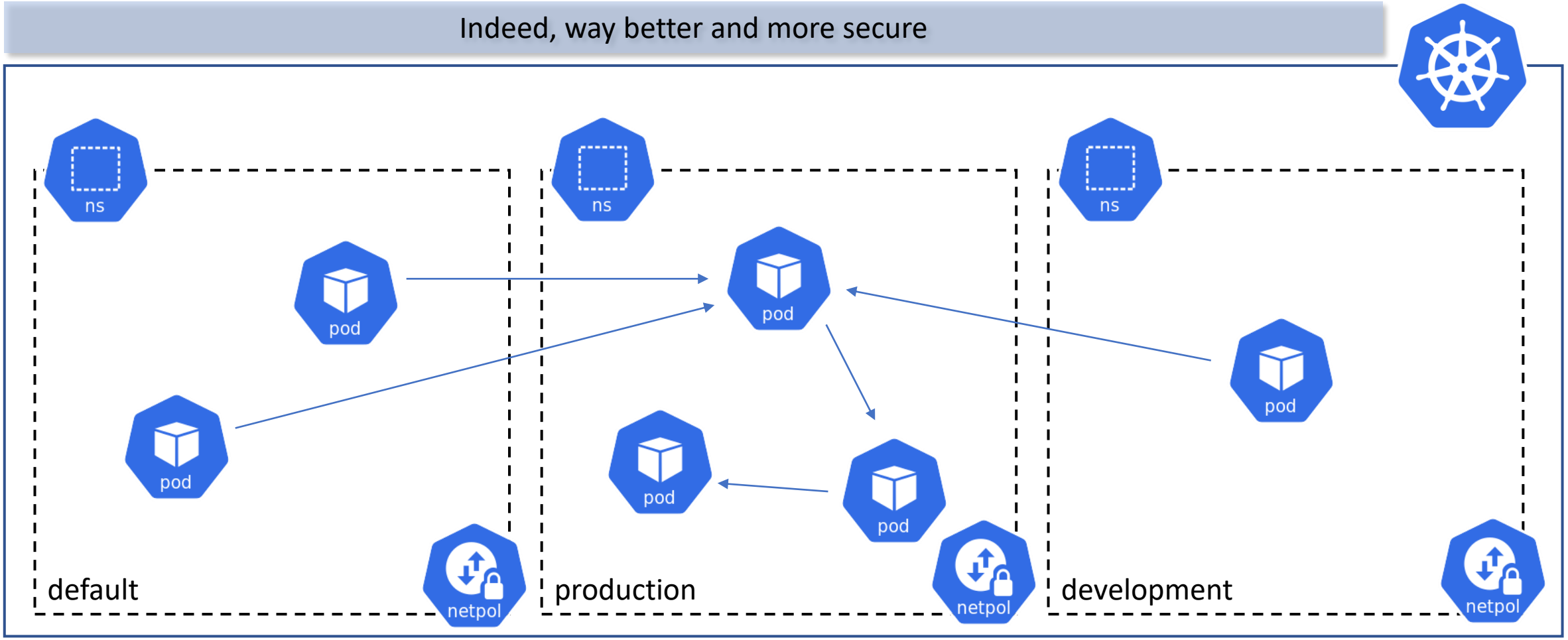


Kubernetes Network Policies

- Control traffic flow at the IP address or port level for apps in the cluster
- They are application centric and working on OSI Layer 3 and 4
- Allow control if and how a pod is allowed to communicate with various network entities over the network
- The entities are identified by a combination of the following identifiers
  - Other pods that are allowed
  - Namespaces that are allowed
  - IP blocks that are allowed

After Network Policies

Indeed, way better and more secure

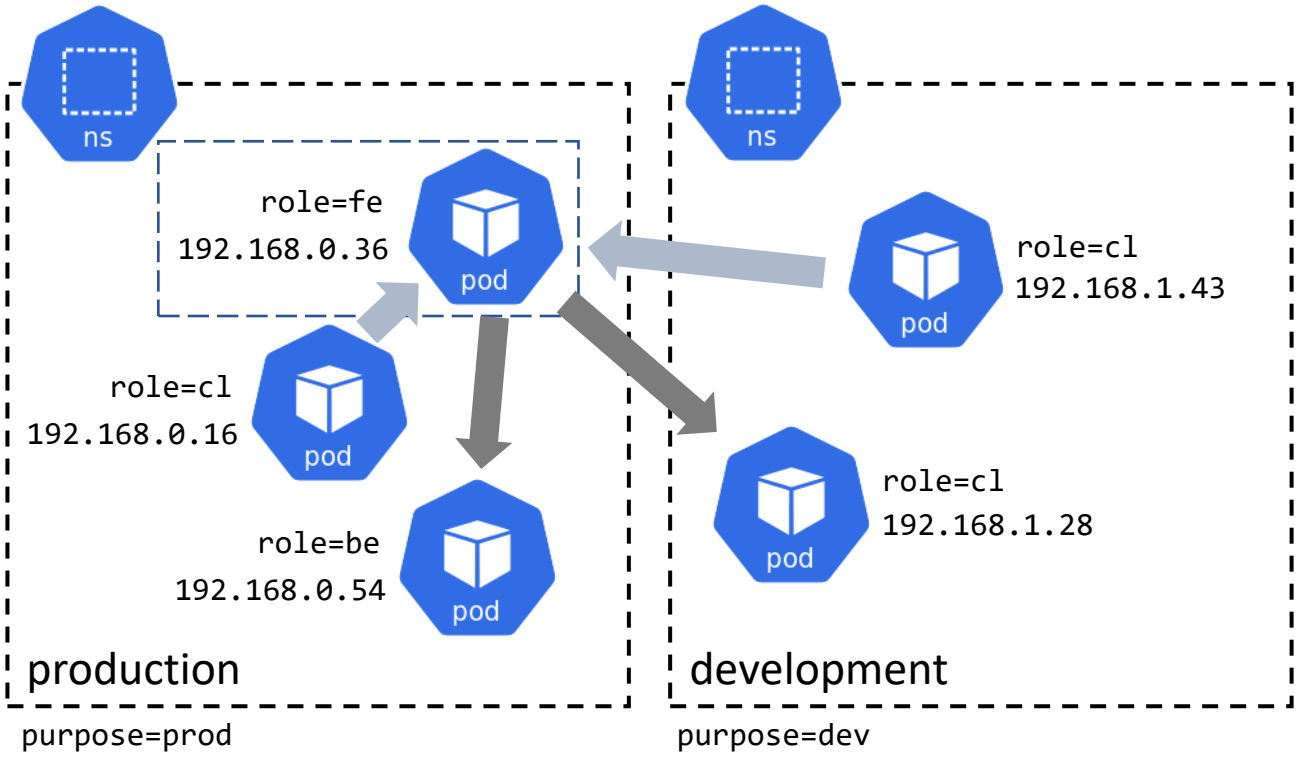


Requirements

- Network policies are **implemented by the network plugin**
- Creating a network policy without the right plugin won’t have any effect
- So, we should use one that provides support if we want to use them
- **Flannel** does not support network policies
- **Antrea**, **Calico**, **Cilium**, **Romana**, and **Weave Net** do support them
- Even **Kube-router** has support for network policies
- Some, like Calico, even offer an **improved** or **extended** policy object

Policy Types (based on direction)

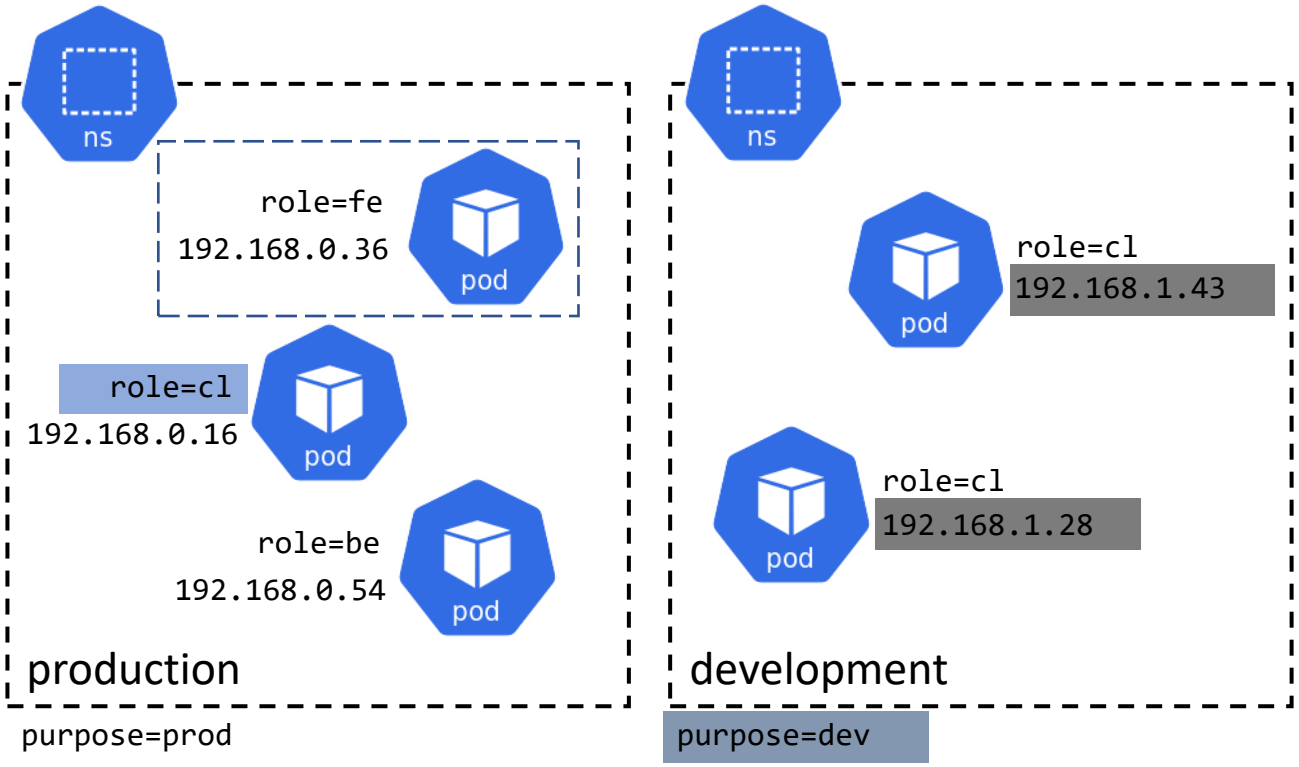
- Ingress
  - Allows incoming communication
- Egress
  - Allows outgoing communication



- Of course, we can mix them

Policy Types (based on criteria)

- Pod-based policy
  - Using a pod selector
- Namespace-based policy
  - Using a namespace selector
- IP-based policy
  - Using IP block/CIDR range
- Of course, we can mix them



Network Policy Resource (1)

- A sample policy (partial)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: fe-network-policy
  namespace: production
spec:
  podSelector:
    matchLabels:
      role: fe
  policyTypes:
  - Ingress
  - Egress
```

apiVersion, **kind**, and **metadata** are mandatory as with other Kubernetes objects

namespace can be omitted. In any case, network policies are namespaced objects. So, we must either set it this way or specify it during a deployment

spec contains all the information needed to define network policy

podSelector is part of every policy. It selects the group of pods to which a policy applies. If left empty, it will select all pods in the namespace

policyTypes includes either **Ingress**, **Egress**, or both. If not specified, the default is **Ingress**

Network Policy Resource (2)

- A sample policy (partial)

```
ingress:
 - from:
   - ipBlock:
       cidr: 192.168.0.0/16
       except:
       - 192.168.1.0/24
   - namespaceSelector:
       matchLabels:
         project: development
   - podSelector:
       matchLabels:
         role: cl
   ports:
   - protocol: TCP
     port: 6379
```

Each policy may include a list of allowed **ingress** rules

Each rule allows the traffic that matches **both** the **from** and **ports** sections

On the left, we can see a **single rule**

It matches the traffic on a single port (**6379/tcp**) that is coming from one of three sources – **ipBlock**, **namespaceSelector**, and **podSelector**

Network Policy Resource (3)

- A sample policy (partial)

```
egress:
 - to:
   - ipBlock:
       cidr: 10.0.0.0/24
   ports:
   - protocol: TCP
     port: 5978
```

Each policy may include a list of allowed **egress** rules

Each rule allows the traffic that matches **both** the **to** and **ports** sections

On the left, we can see a **single rule**

It matches the traffic on a single port (**5978/tcp**) to any destination in **10.0.0.0/24** specified via **ipBlock**

Of course, here we can use also a **namespaceSelector** and **podSelector**

Default Policies

Default deny ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Default deny egress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

Default deny all

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Default allow all ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

Default allow all egress

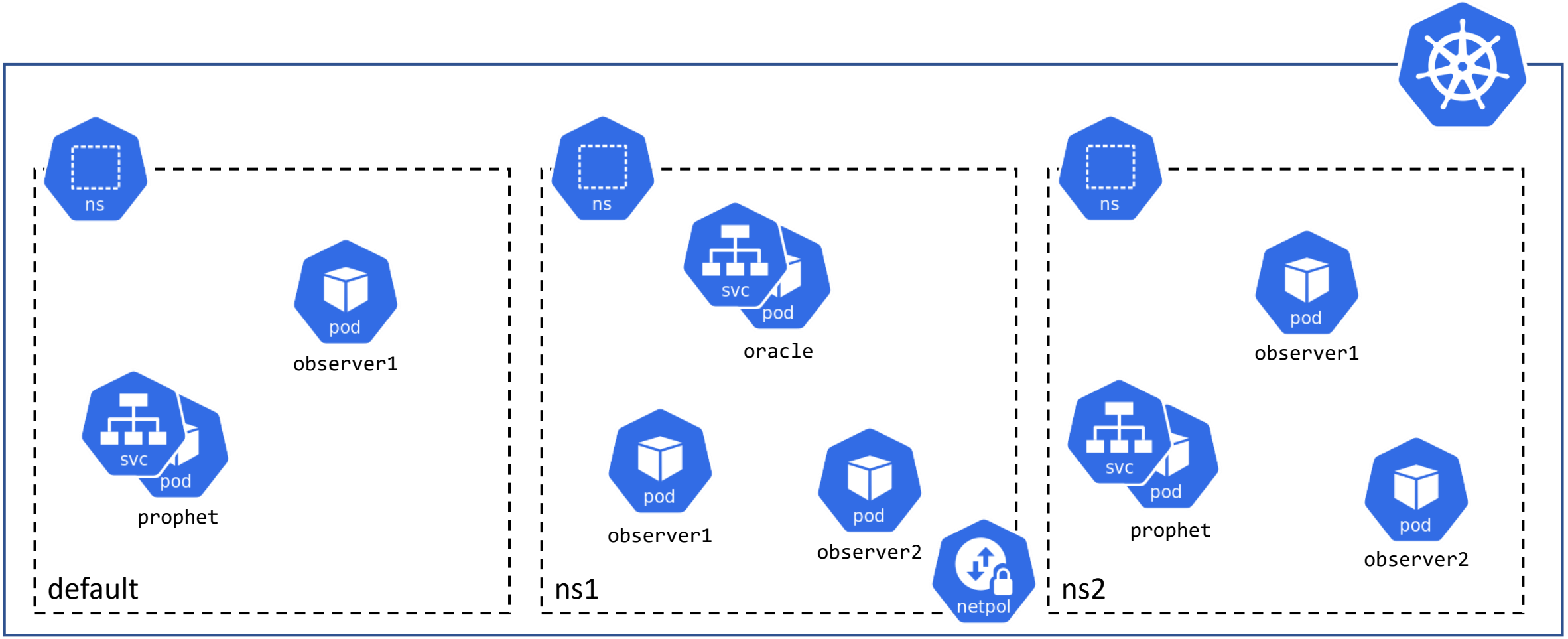
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

A Few More Things

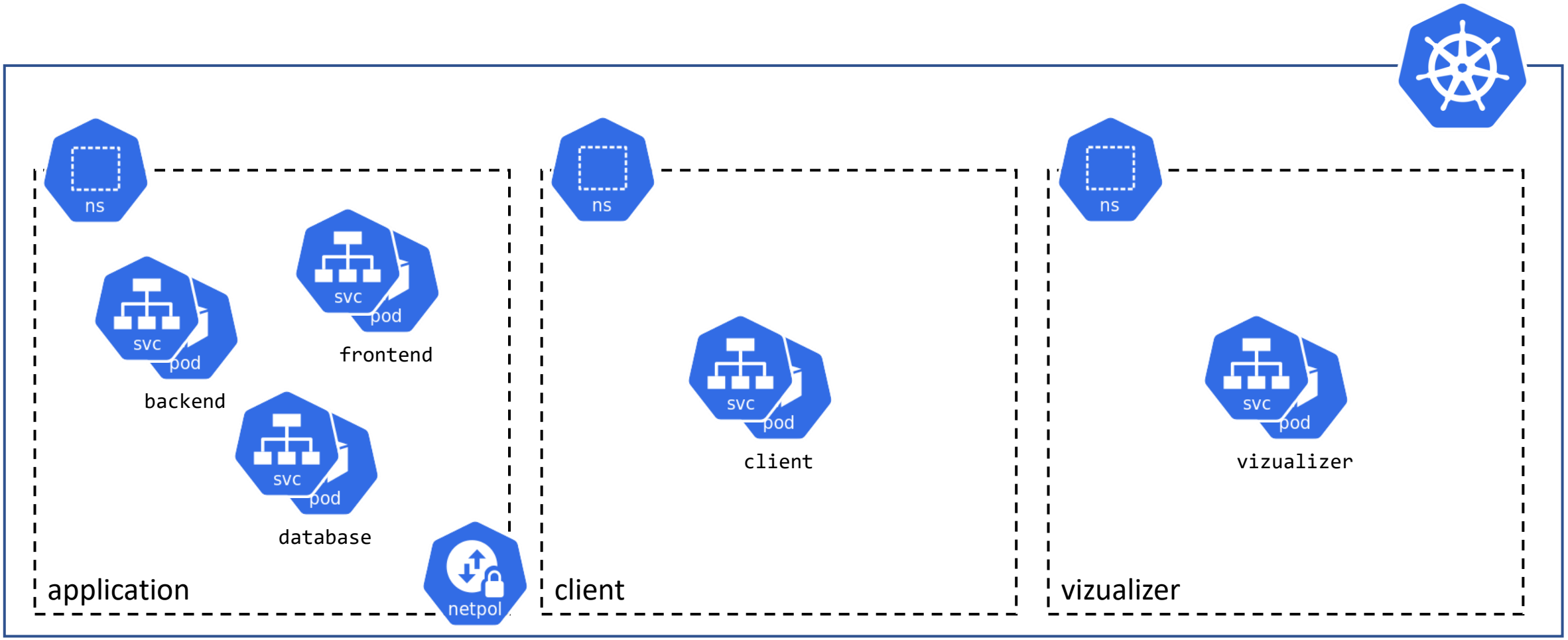
- Additivity
  - **Ingress** lists defined in multiple policies for a single pod or group of pods are **combined additively**. The same applies for **egress** lists
  - Network policies do not conflict, they are **additive**
  - The **order** of evaluation **does not affect** the result
- Combination
  - Usually, we use the **namespaceSelector** and **podSelector** individually (**OR**)
  - We can use them combined in a single entry (**AND**)

Demo

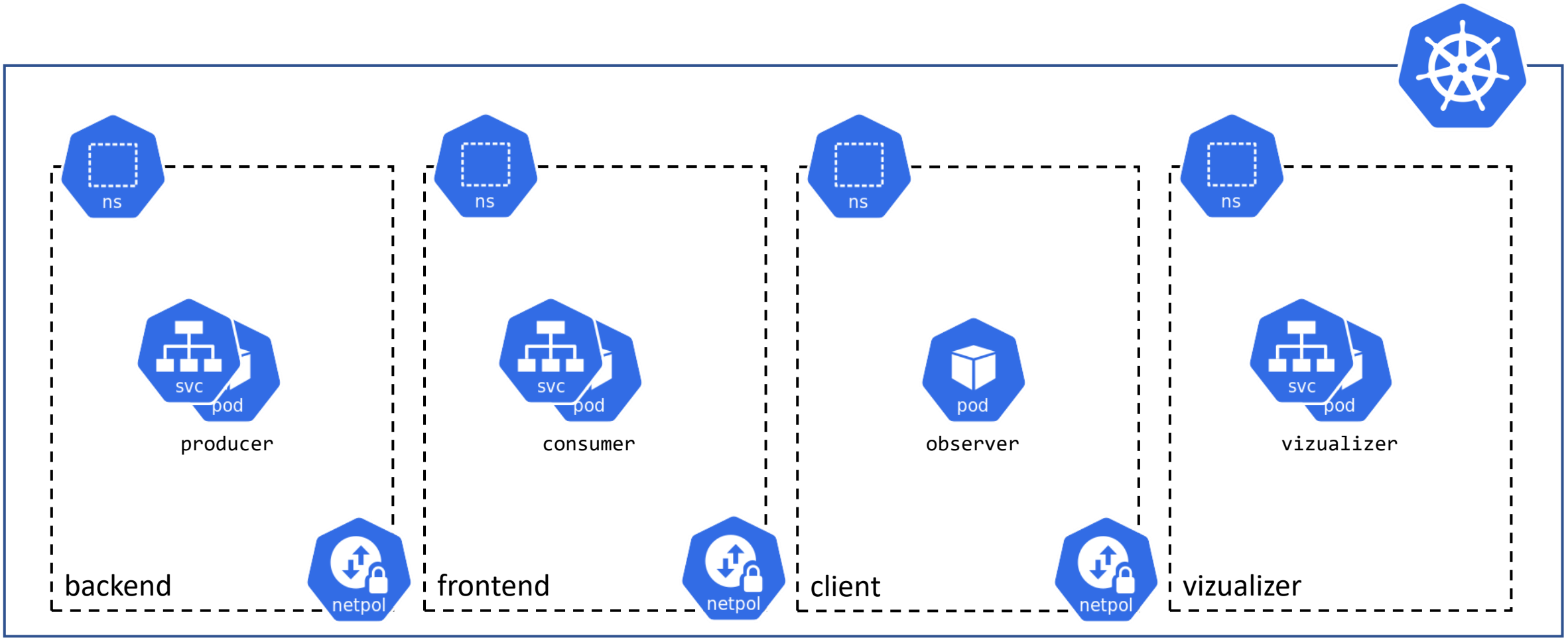
Scenario #1


ns
pod
observer1
svc
pod
prophet
default
ns
svc
pod
oracle
pod
observer1
pod
observer2
netpol
ns1
ns
pod
observer1
svc
pod
prophet
pod
observer2
ns2

Scenario #2 *


ns
svc
pod
backend
svc
pod
frontend
svc
pod
database
netpol
application
ns
svc
pod
client
client
ns
svc
pod
vizualizer
vizualizer

Scenario #3 *



Q&A Session

Thank you!

Contacts:

https://www.linkedin.com/in/dzahariev/

https://github.com/shekeriev

Resources

- Network Policies
  - https://kubernetes.io/docs/concepts/services-networking/network-policies/
- Declare Network Policy
  - https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/
- Network Policy Recipes
  - https://github.com/ahmetb/kubernetes-network-policy-recipes
- Calico Network Policy
  - https://projectcalico.docs.tigera.io/security/calico-network-policy