



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

# The Shared Service Blueprint: A Guide to Multi-Tenancy, Illustrated with KEDA

Aya Igarashi (@ladicle), Preferred Networks Inc.



# Agenda

## 1. Background

Why we need multi-tenancy for our AI compute platform

## 2. Multi-tenancy Patterns

How to isolate tenants on Kubernetes

## 3. Challenges & Solutions

What we had to solve after choosing a deployment model

## 4. Key Takeaways

# Our Multi-Tenancy Requirements



## ■ Full-Stack AI Company:

From research models down to chip design.  
We develop MN-Core, a custom AI processor.

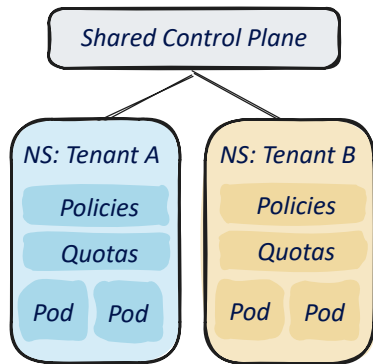
## ■ PFCP (Preferred Computing Platform):

A Kubernetes-based platform providing our  
compute to **internal & external** researchers



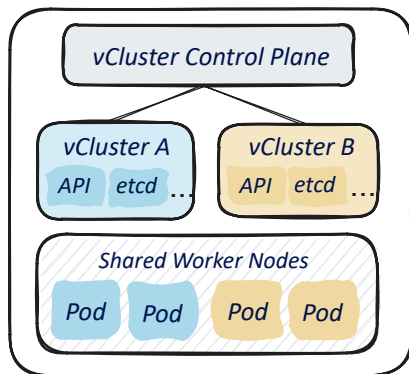
# Kubernetes Multi-Tenancy Patterns

## Namespace Isolation



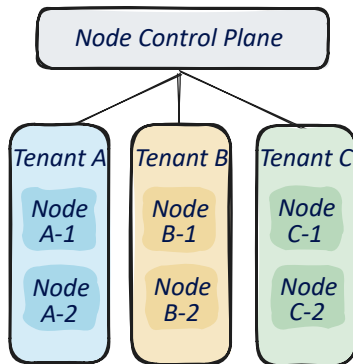
Logical Isolation  
Cost: Low

## Virtual Cluster



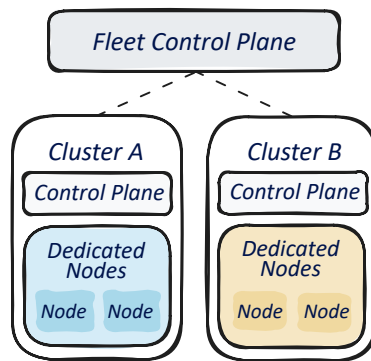
API Isolation  
Cost: Medium

## Dedicated Node



Kernel Isolation  
Cost: Medium

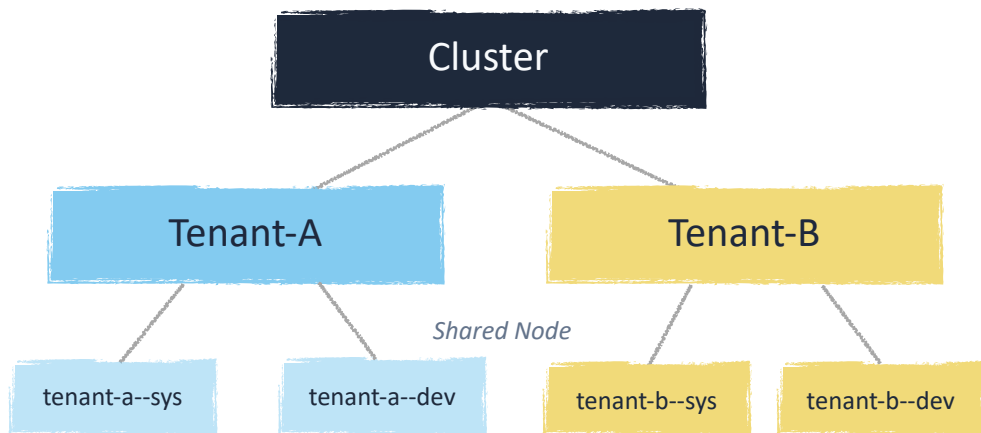
## Dedicated Cluster



Physical Isolation  
Cost: High

# Our Tenancy Model

## Hierarchical Namespaces (HNC) & Node-level Isolation



## Why this model

By combining namespace-based isolation with optional node-level isolation, you can dynamically adjust the cost and isolation level for multi-tenant environments.

## What is HNC?

Each tenant owns a root namespace and can manage child namespaces. It also manages ResourceQuotas per tenant.

*(HNC is archived upstream; we maintain a fork)*

- ▶ [github.com/kubernetes-retired/hierarchical-namespaces](https://github.com/kubernetes-retired/hierarchical-namespaces)
- ▶ [github.com/pfnet/hierarchical-namespaces](https://github.com/pfnet/hierarchical-namespaces)

# Why KEDA is a good case study

## Not built for our platform

In-house components can be multi-tenant by design.

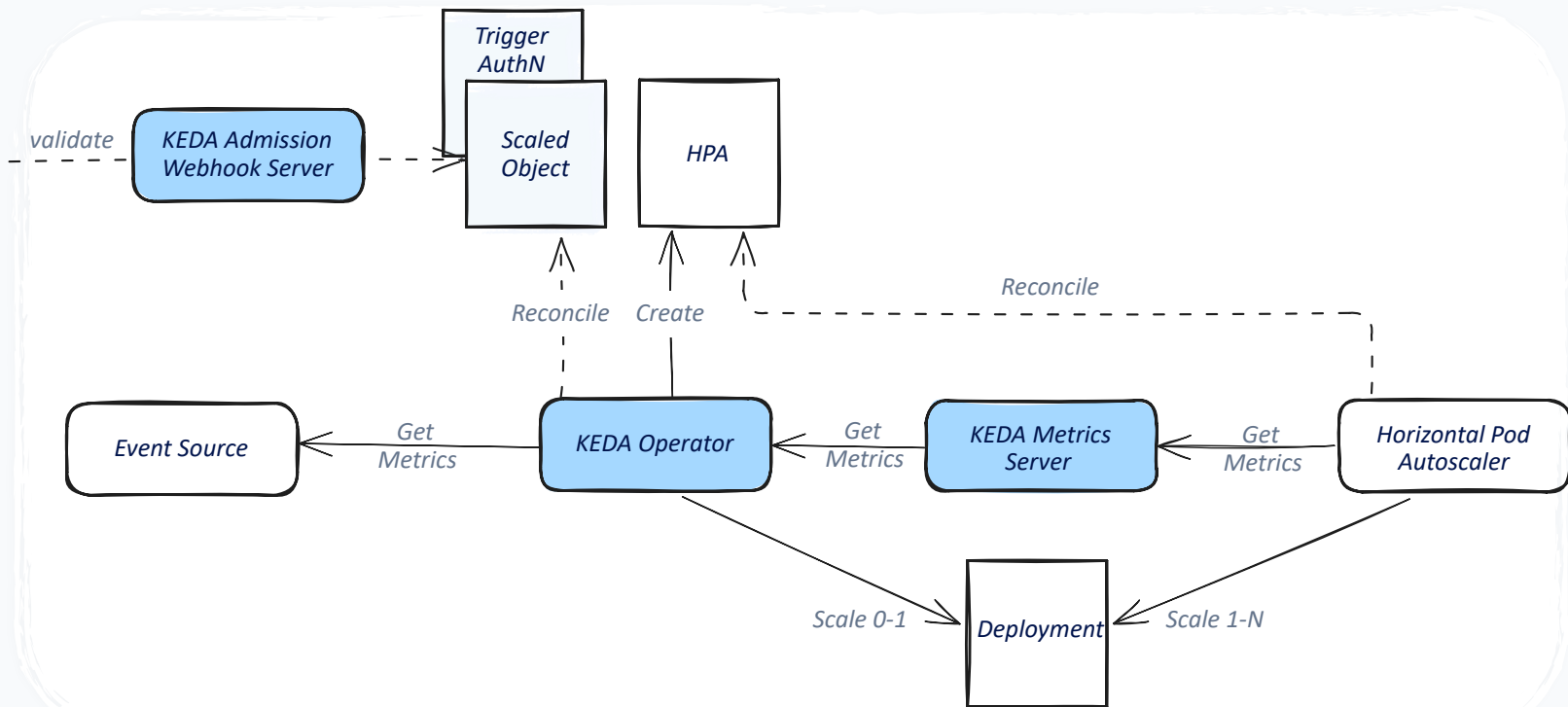
OSS components require adaptation.



## Touches many boundaries

Scaling, identity, metrics, RBAC, and cloud integration all meet here.

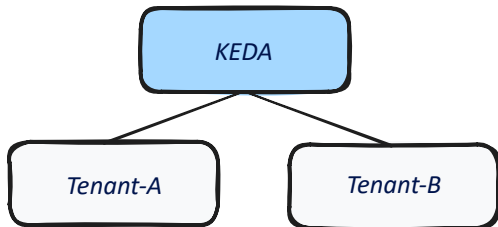
# Quick Recap: KEDA



# Shared or Per-Tenant KEDA?

We select a per-tenant deployment model.

## Shared



One operator + one metrics server shared across tenants

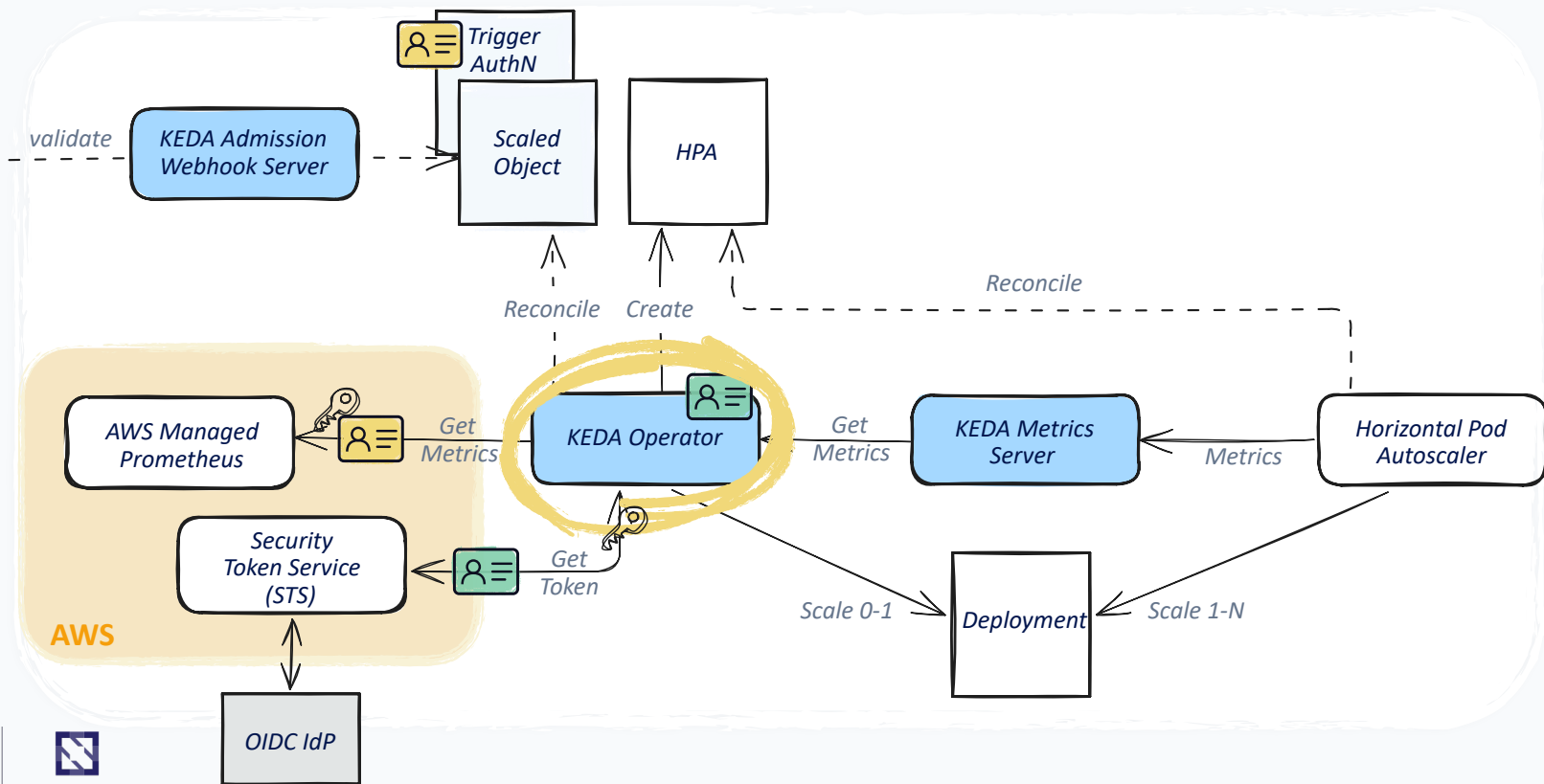
## Per-Tenant



Each tenant gets its own operator + metrics server

|                      | Shared                                        | Per-Tenant                           |
|----------------------|-----------------------------------------------|--------------------------------------|
| Operation            | One bug or load spike can affect every tenant | N installations to manage per tenant |
| Efficiency           | Best resource efficiency                      | Idle overhead per tenant             |
| Blast Radius         | One bug or load spike can affect every tenant | Failure stays inside one tenant      |
| Secret Scope         | May still require broad Secret read           | Secret read can stay namespaced      |
| Auth Bootstrap (AWS) | Identity that initiates cloud auth is shared  | Each tenant gets its own auth origin |

# Quick Recap: KEDA x AWS AMP Integration



# Where the Trust Boundary Actually Lives

## What looks tenant-friendly

- ✓ Auth can be configured per namespace.
- ✓ TriggerAuthentication lives with the workload.
- ✓ RBAC can isolate the KEDA objects.

## What still stays shared

- ! The operator process still owns the cloud identity.
- ! Some event sources run credential exchange under that identity.
- ! A shared operator becomes a shared trust boundary

**In our case, shared operator = shared trust boundary**  
→ We chose Per-Tenant KEDA

# Per-Tenant KEDA Still Left Four Constraints



## 1 Cluster-wide Singleton

Only one external.metrics API backend can front the cluster.



## 2 Aggregated API auth

The router can accidentally erase the caller identity.



## 3 Namespace scope

Tenant namespaces change, but the controller expects a static watch list.

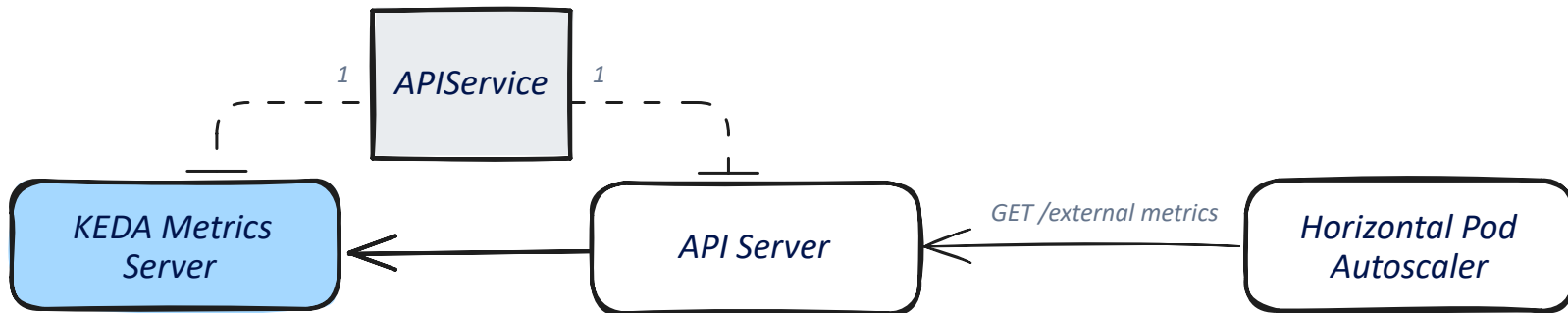


## 4 Traffic boundaries

Per-tenant installs still need explicit path isolation.

MULTIPLE APISERVICE CAN BE CREATED, BUT ONLY ONE CAN BE REGISTERED AS AN EXTERNAL METRIC

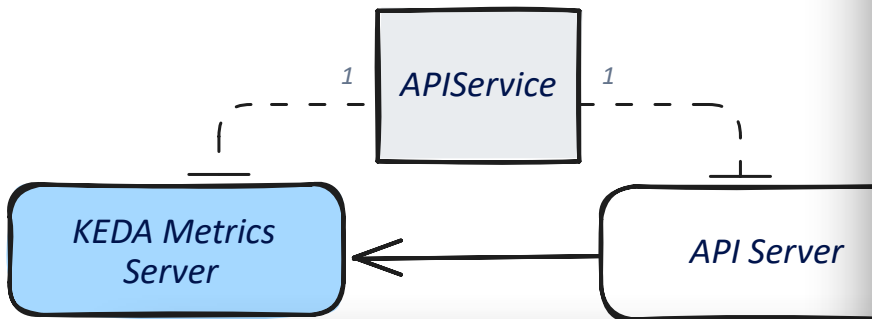
## Challenge 1: External Metrics Is a Singleton



Multiple KEDA installations still need a single cluster-wide entry point for `external.metrics.k8s.io`.

MULTIPLE APISERVICE CAN BE CREATED, BUT ONLY ONE CAN BE REGISTERED AS AN EXTERNAL METRIC

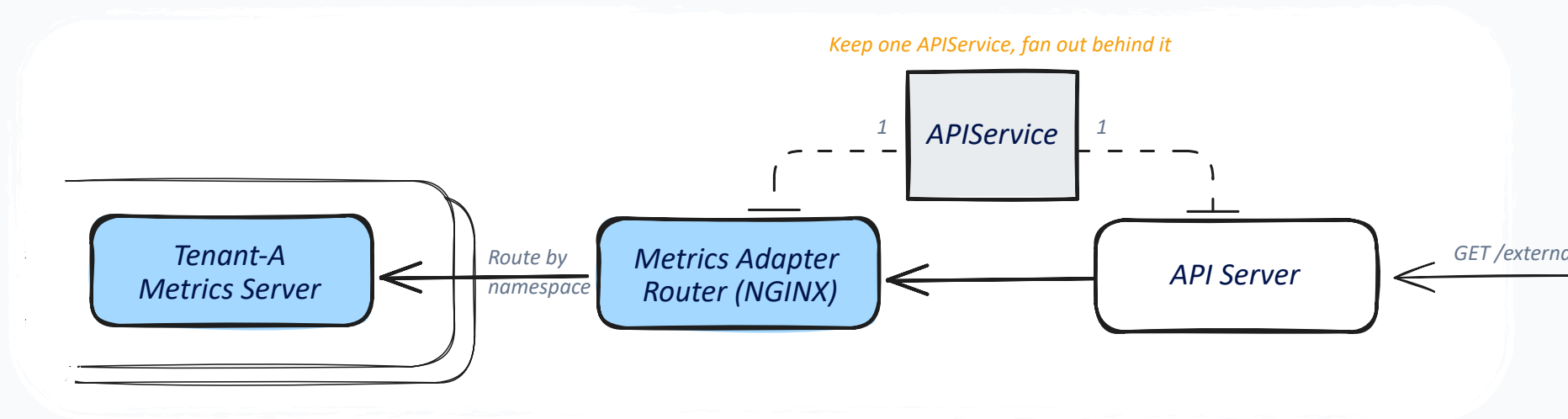
# Challenge 1: External Metrics Is a Singleton



```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1beta1.external.metrics.k8s.io
spec:
  version: v1beta1
  group: external.metrics.k8s.io
  service: external.metrics.k8s.io
    name: keda-external-metrics-apiserver
    namespace: keda-syste
```

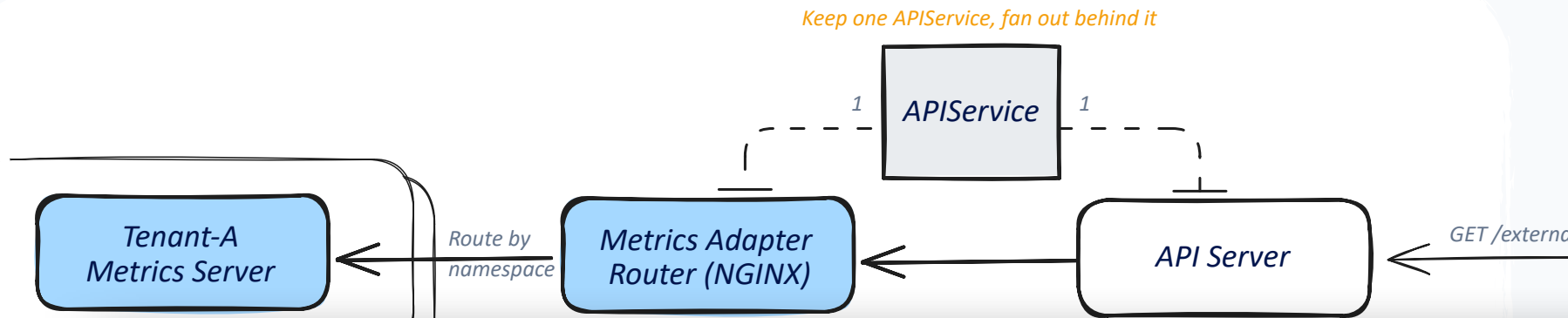
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: keda-hpa-app-scaler
spec:
  minReplicas: 1
  maxReplicas: 3
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  metrics:
    - type: External
      external: external
        metric:
          name: "s0-prometheus"
          selector:
            matchLabels:
              scaledobject.keda.sh/name: app-scaler
        target:
          type: AverageValue
          averageValue: "50"
```

# Solution 1: Metrics Adapter Router



Route requests by namespace  
GET /apis/external.metrics.k8s.io/v1beta1/namespaces/{ns}/...

# Solution 1: Metrics Adapter Router



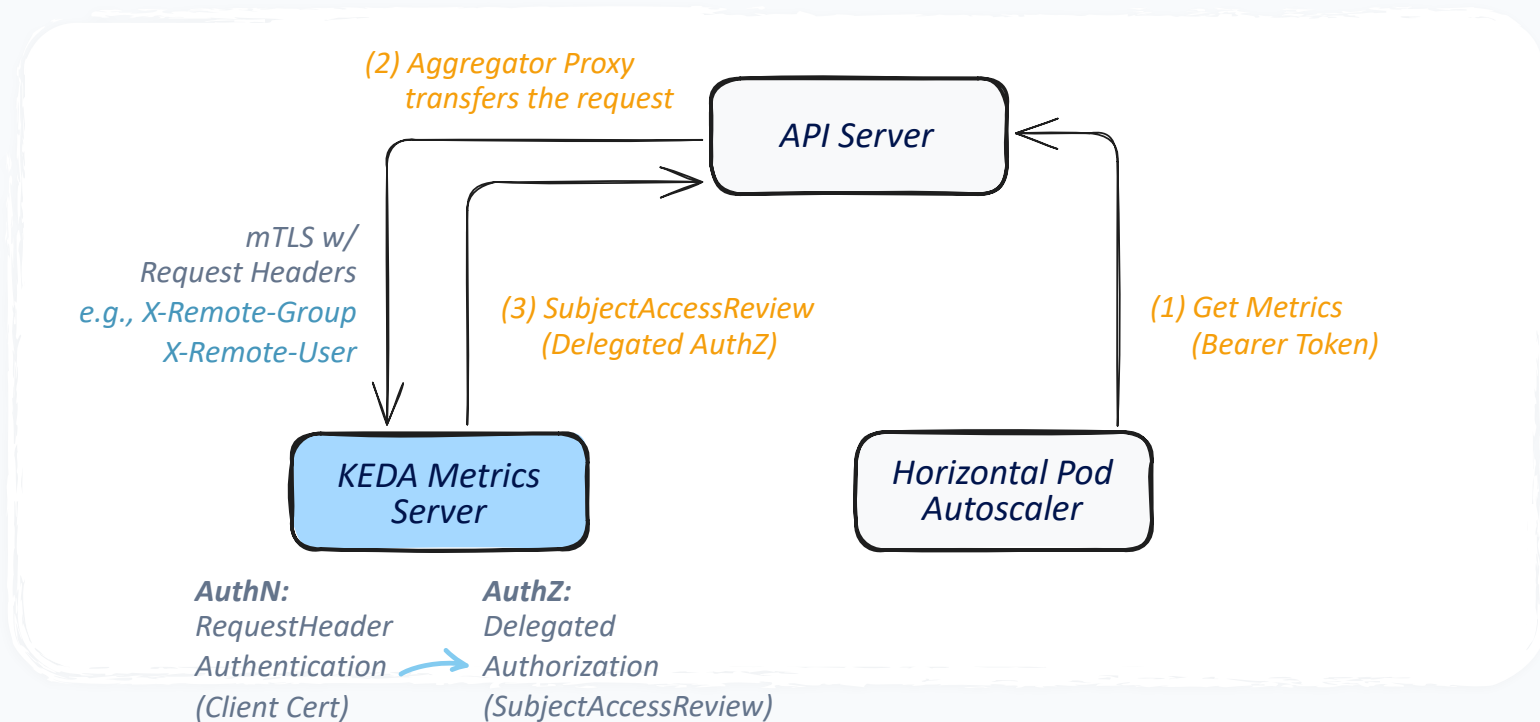
...

```
location ~ ^/apis/external\.metrics\.k8s\.io/v1beta1/namespaces/tenant-(?<tenant>[a-zA-Z0-9]+(?:-|_)+[a-zA-Z0-9])$ {
    set $upstream_service "keda-operator-metrics-apiserver.$tenant.svc.cluster.local";
    proxy_pass https://$upstream_service$request_uri;
```

```
    proxy_ssl_verify on;
    proxy_ssl_trusted_certificate /certs/ca.crt;
    proxy_ssl_certificate /certs/tls.crt;
    proxy_ssl_certificate_key /certs/tls.key;
```

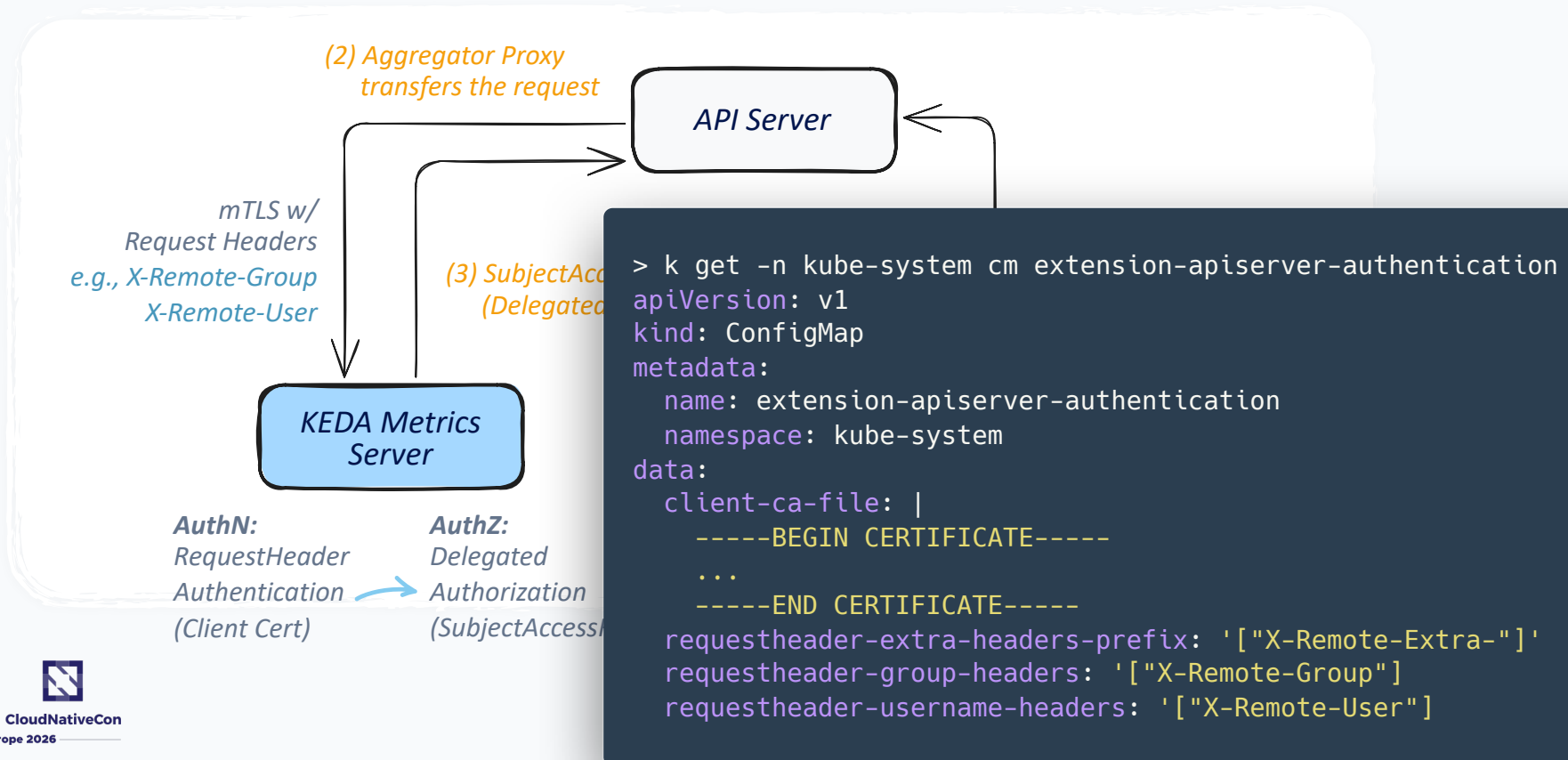
```
}
```

## Challenge 2: Normal Aggregated API Auth Flow



TRUST THE PROXY, NOT THE HEADER ITSELF

## Challenge 2: Normal Aggregated API Auth Flow



KubeCon

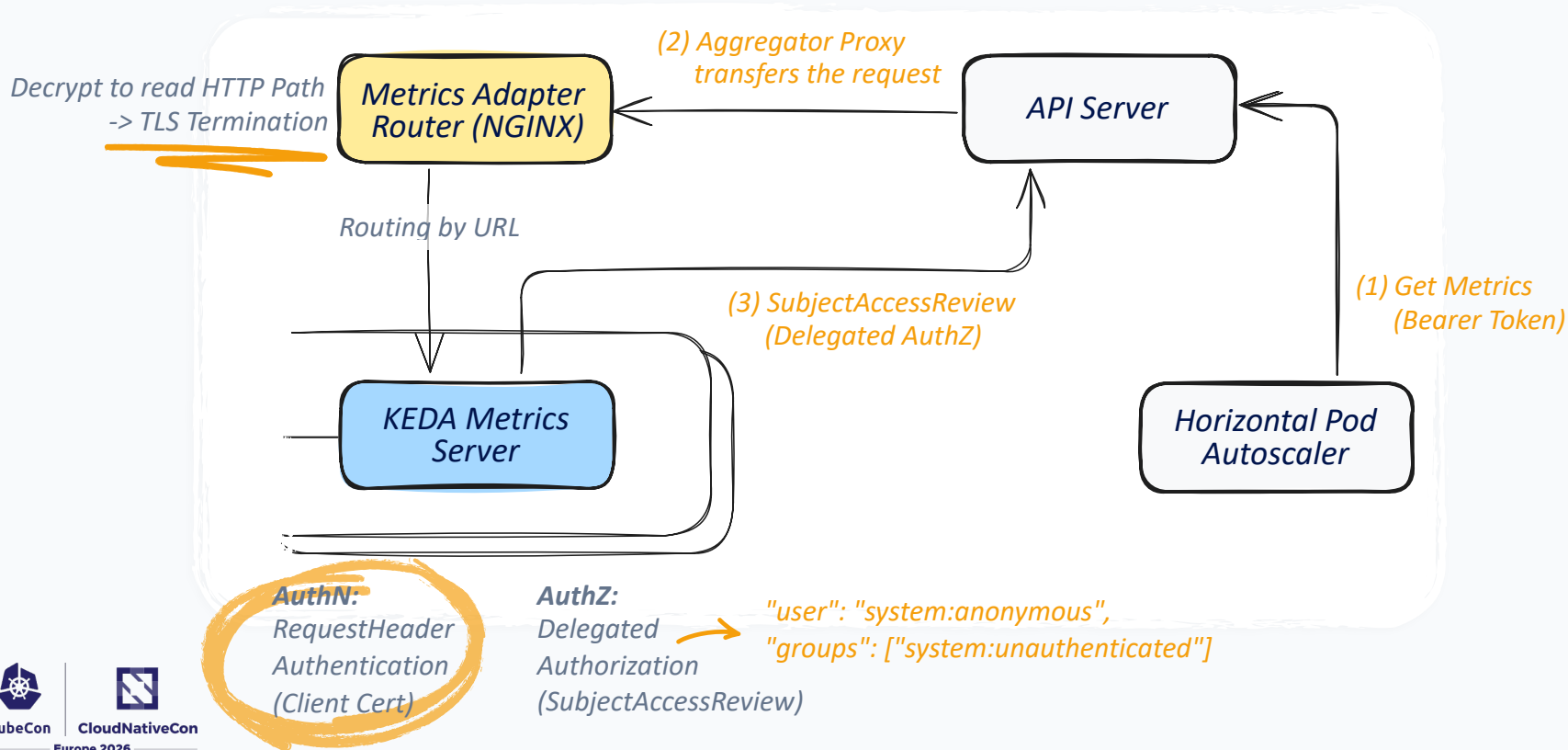


CloudNativeCon

Europe 2026

IF THE BACKEND CANNOT VERIFY THE PROXY, THE CALLER BECOMES ANONYMOUS

## Challenge 2: TLS Termination Can Break Caller Identity



## Solution 2: Rebuild the Trust Chain

### mTLS router -> backend

The router must present a client certificate when it connects upstream.

### Trust the router CA

Configure request-header auth with the router CA, not just the default API server CA.

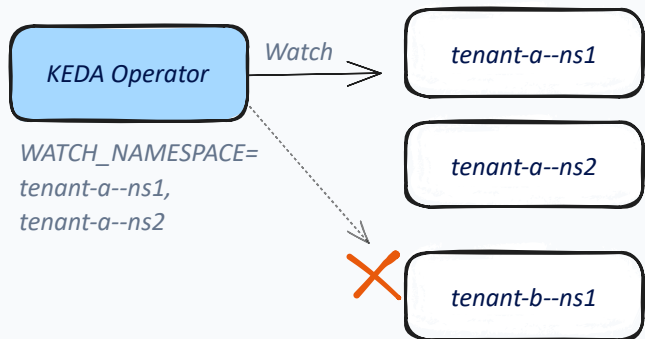
<https://github.com/kubernetes-sigs/custom-metrics-apiserver>

### Preserve X-Remote-\*

Do not let the proxy strip or normalize the identity headers.

**Result: Metrics Server still can see the original HPA identity, and delegated auth keeps working.**

## Challenge 3: Operator Namespace Scope



- **Security: operator watches secrets**  
Restrict watch scope to tenant namespaces only

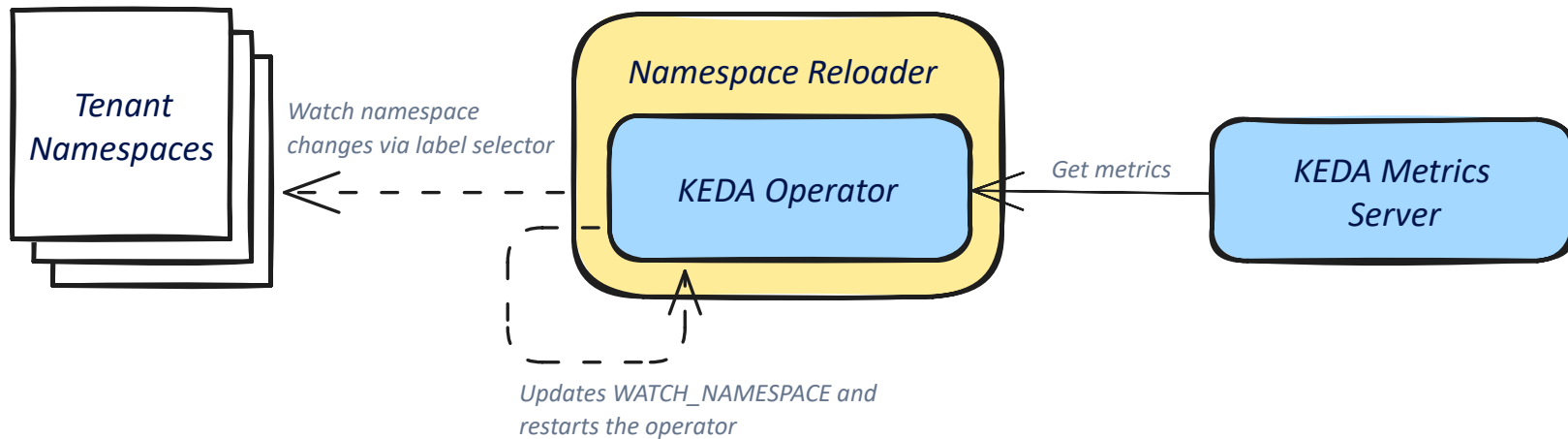
- **WATCH\_NAMESPACE is a static list**  
Users create/delete namespaces freely – static config can't keep up

- **controller-runtime can't change watch scope at runtime**  
Manager's cache namespace config is immutable after Start()

→ Need a way to dynamically update namespace scope

WRAP THE OPERATOR IN A PROCESS MANAGER THAT RESTARTS ON NAMESPACE CHANGES

## Solution 3: Operator Namespace Reloader

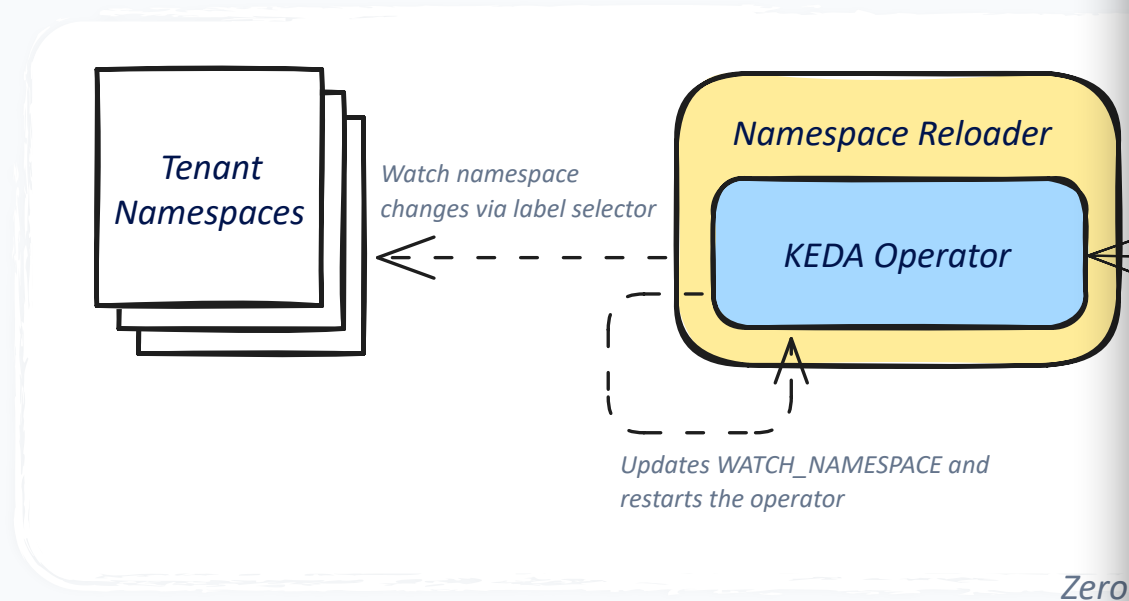


Zero changes to KEDA itself — image + env only

<https://github.com/pfnet/ns-reloader>

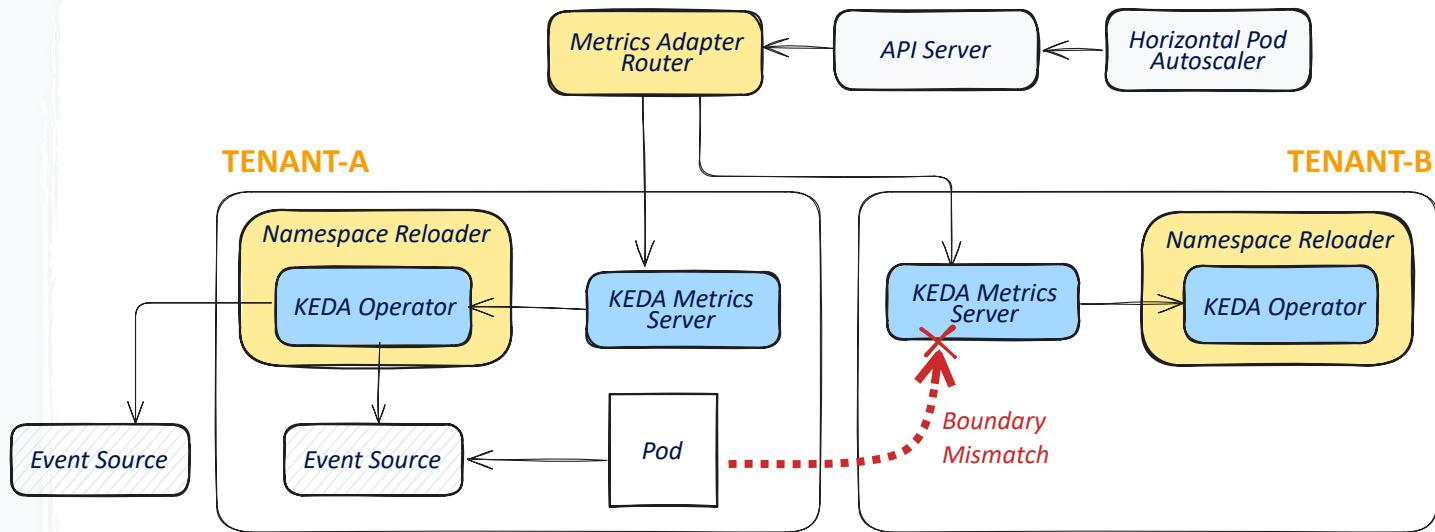
WRAP THE OPERATOR IN A PROCESS MANAGER THAT RESTARTS ON NAMESPACE CHANGES

## Solution 3: Operator Namespace Reloader



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: keda-operator-with-reloader
  namespace: keda
spec:
  replicas: 1
  selector:
    matchLabels:
      app: keda-operator
  template:
    metadata:
      labels:
        app: keda-operator
    spec:
      serviceAccountName: keda-operator
      containers:
        - name: keda-operator
          image: <wapped_keda_operator_image>
          env:
            - name: RELOADER_NAMESPACE_SELECTOR
              value: "keda.sh/enabled=true"
            - name: RELOADER_TARGET_ENV_KEY
              value: "WATCH_NAMESPACE"
          command: ["/reloader"]
          args:
            - "/keda"
            - "--leader-elect"
            - "--zap-log-level=info"
```

## Challenge 4: Network boundary mismatch



→ A malicious or misconfigured Pod in tenant A can directly reach KEDA components and user Pods in tenant B

# Solution 4: Match Policy to Traffic Scope



## Ingress:

Allow only known sources (Router, Metrics Server, etc.)



## Egress:

Scope to own tenant namespaces + Required services

tenant-system local

shared control-plane

KEDA Metrics Server

Metrics Adapter Router :443

gRPC :9666

KEDA Operator

shared control-plane

Kube API Server

KEDA Operator

Tenant Event Source

tenant-wide

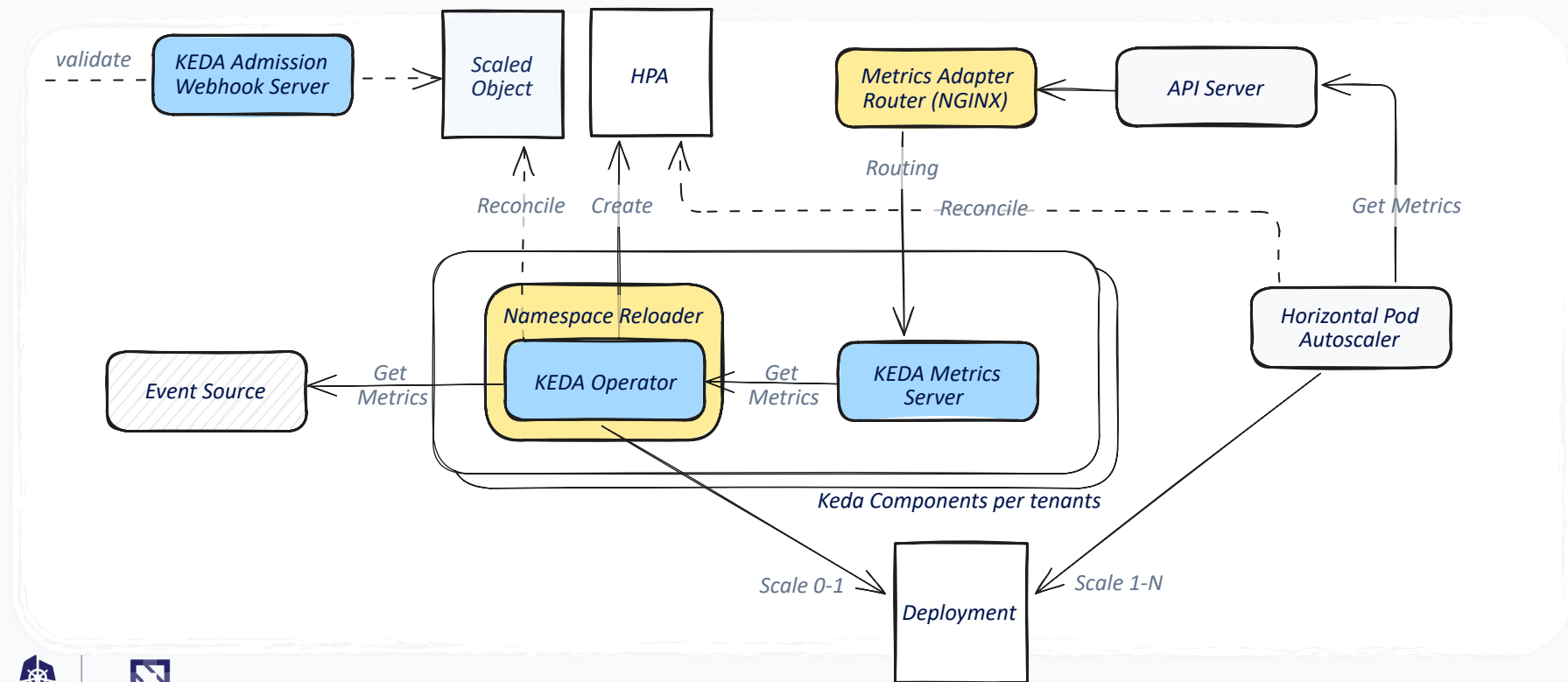
Cross-tenant = deny

**NetworkPolicy design steps: classify the paths, then define the scope.**

*We used HNC-propagated namespace labels. In Cilium, matching those labels in toEndpoints meant we needed a clusterwide policy.*



# Full Architecture



# Key Takeaways



## The Challenge

- **OSS Assumptions** – 1 cluster = 1 instance (Isolation rarely considered)



## What to Evaluate

- **Operations** – Manageability, resource efficiency
- **Reliability** – Blast radius of failures, noisy neighbor risk
- **Security** – Permissions, Network paths & Auth



## No Silver Bullet

- **Security** – Permissions, Network paths & Auth
- **Embrace Trade-offs** – Decide based on actual requirements



GIVE ME FEEDBACK!

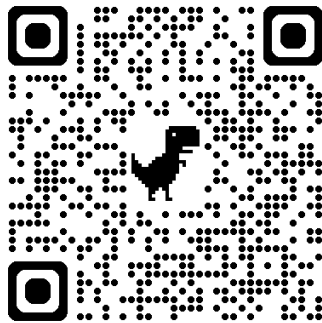
# Thank you!



**Aya Igarashi**

GitHub: @Ladicle

Email: [ladiclexxx@gmail.com](mailto:ladiclexxx@gmail.com)



<https://sched.co/2CW6n>

## Resources

Forked Hierarchical Namespaces:

<https://github.com/pfnet/hierarchical-namespaces>

Namespace Reloader:

<https://github.com/pfnet/ns-reloader>



KubeCon



CloudNativeCon

Europe 2026

