
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Microsegment Your Network Like Mastercard with AdminNetworkPolicy

KubeCon CloudNativeCon
North America 2024

Surya Seetharaman
Principal Software Engineer @ Red Hat
SIG-Network-Policy-API Contributor
@tssurya

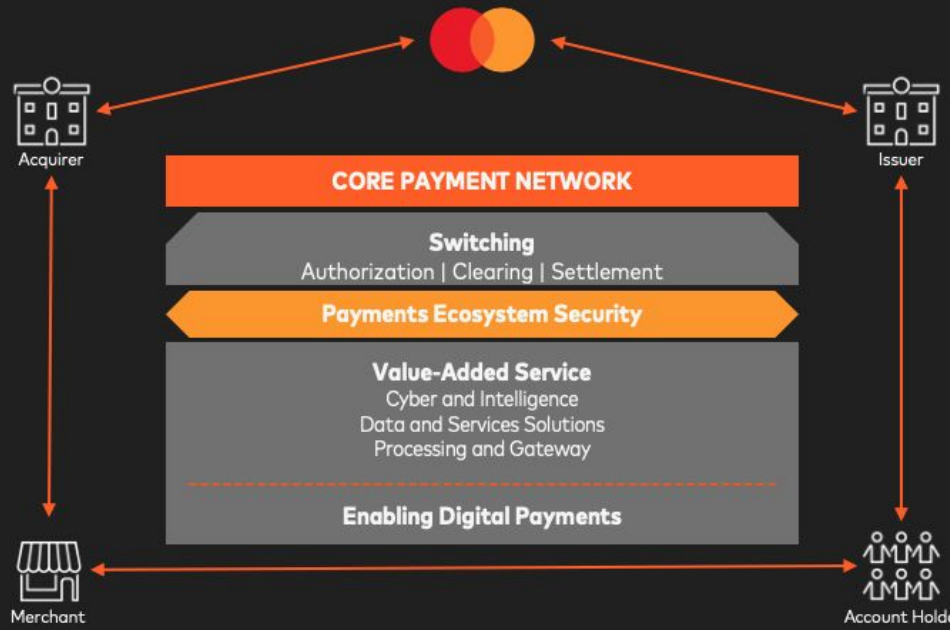
John Zaiss
Principal Engineer @ Mastercard
20+ years at Mastercard
6+ years working in Kubernetes

Daniel Ruggeri
Distinguished Engineer @ Mastercard
Web, Cloud, Network, Infra, Automation, and FOSS nerd

Mastercard - Overview

Mastercard - Overview

Mastercard is a technology company in the global payments industry. We connect consumers, financial institutions, merchants, governments, digital partners, businesses and other organizations worldwide by enabling electronic payments and making those payment transactions safe, simple, smart and accessible.



The Mastercard Journey to Kubernetes

The Mastercard private cloud ecosystem lacked a multifunctional enterprise Kubernetes platform essential for effectively managing and executing diverse container workloads at scale. We've been on this journey for quite some time and have designed a solution, which we have internally branded as the Mastercard Kubernetes Service (MKS).

Security

Scalable

Reliability

Developer Experience

Unified Cloud Platform

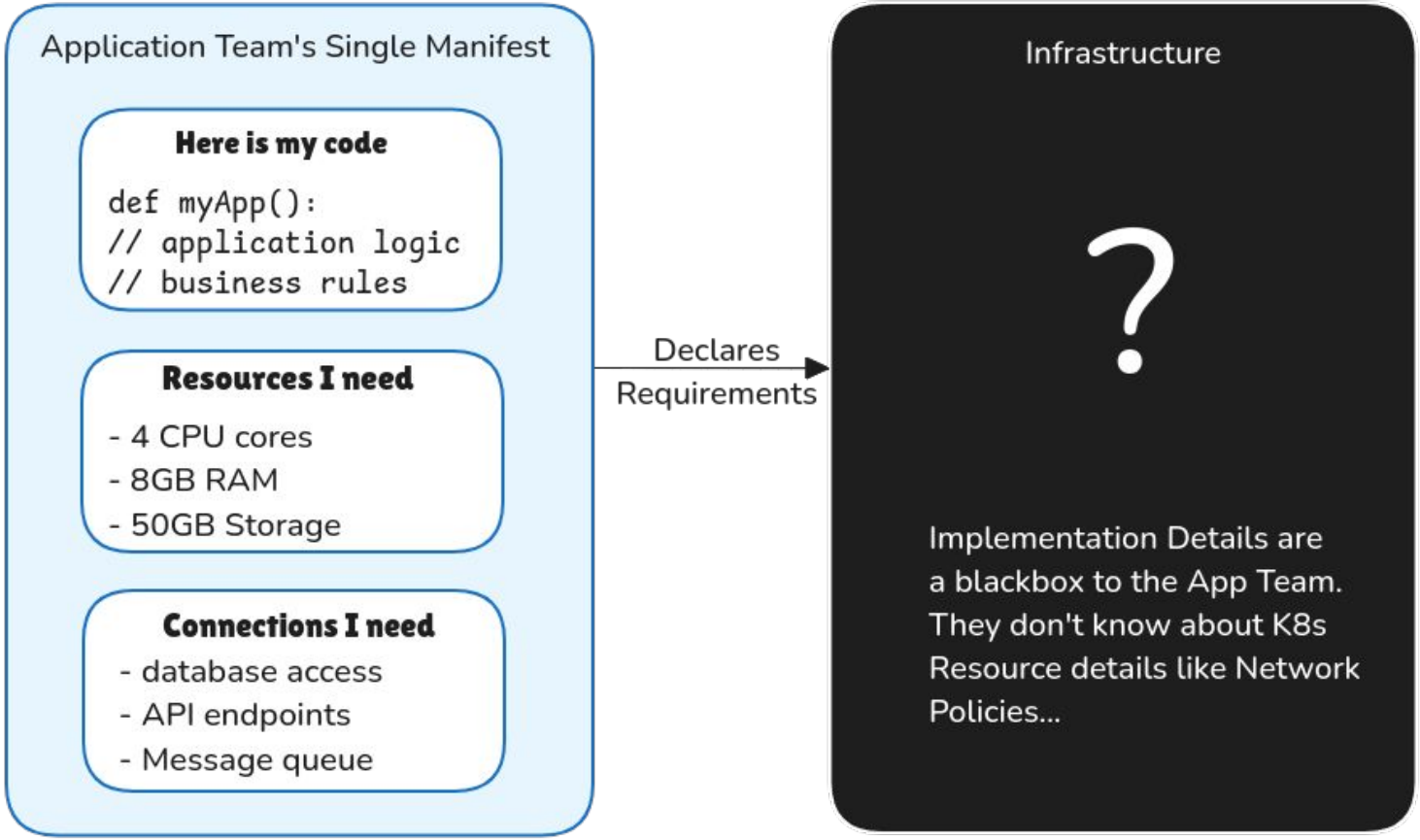
Enabling M&A

KubeCon | CloudNativeCon

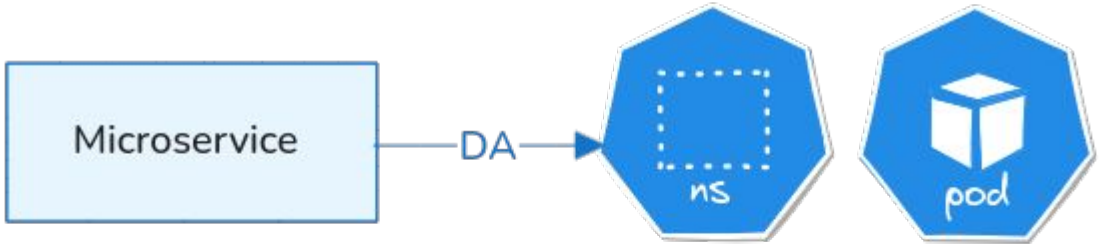
North America 2024

Microsegmentation in Mastercard

The Declarative Vision



Converting declarative request to Kubernetes resources

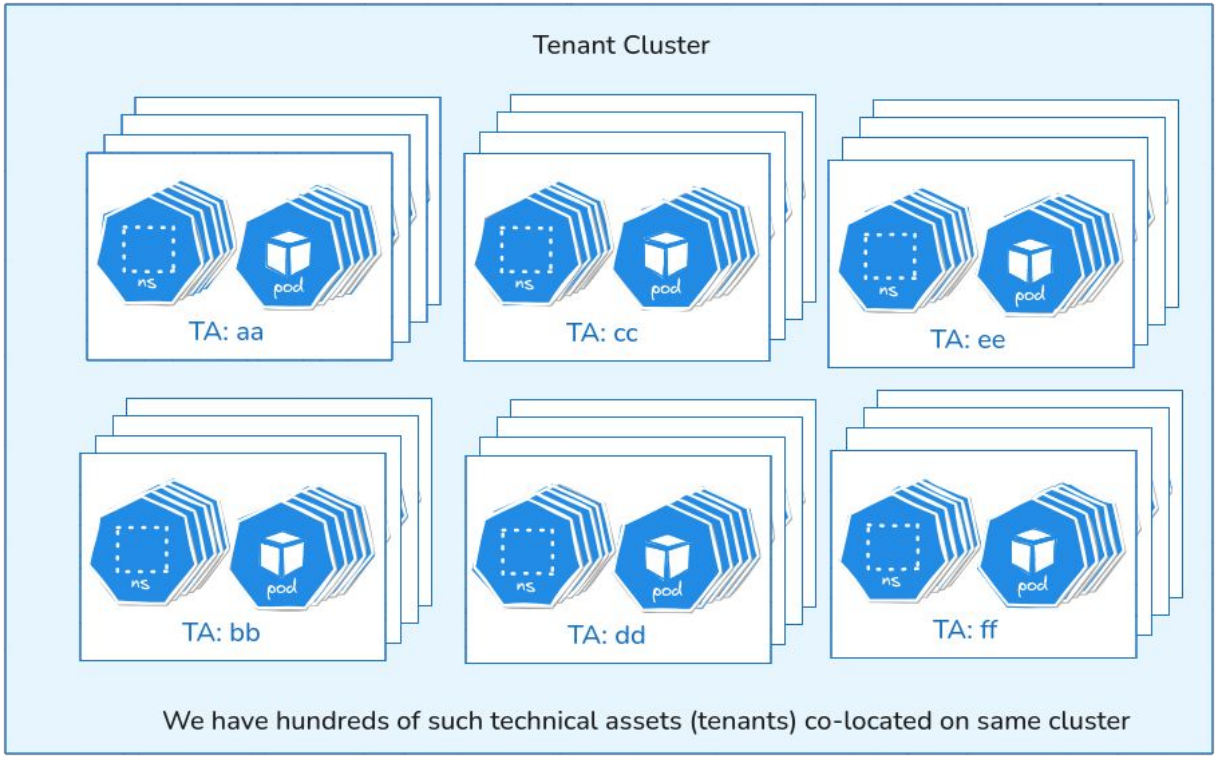


Each microservice (deployable asset) is 1 pod in 1 namespace

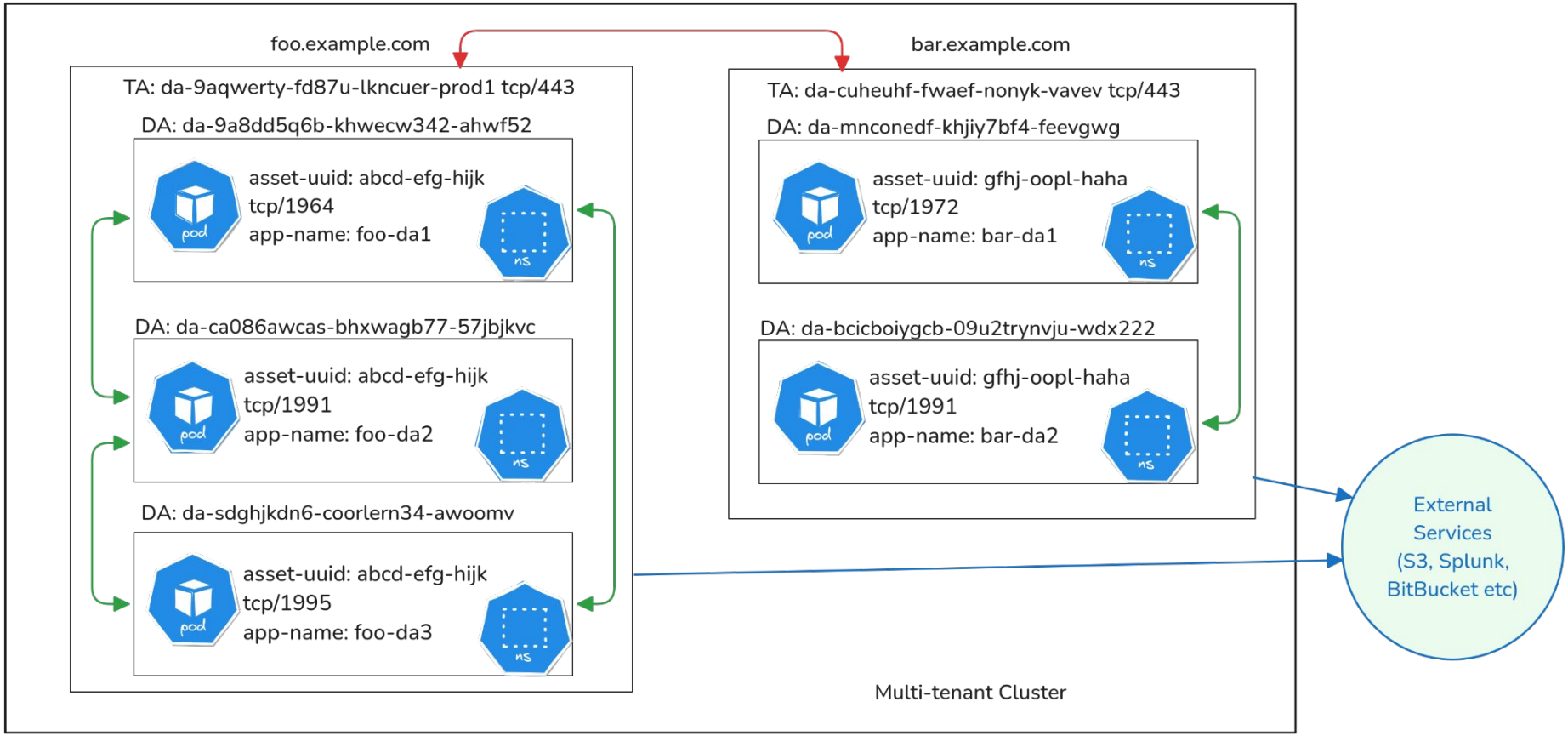
Each tenant (technical asset) has approximately 5 microservices (deployable assets)

Requirement for Microsegmentation

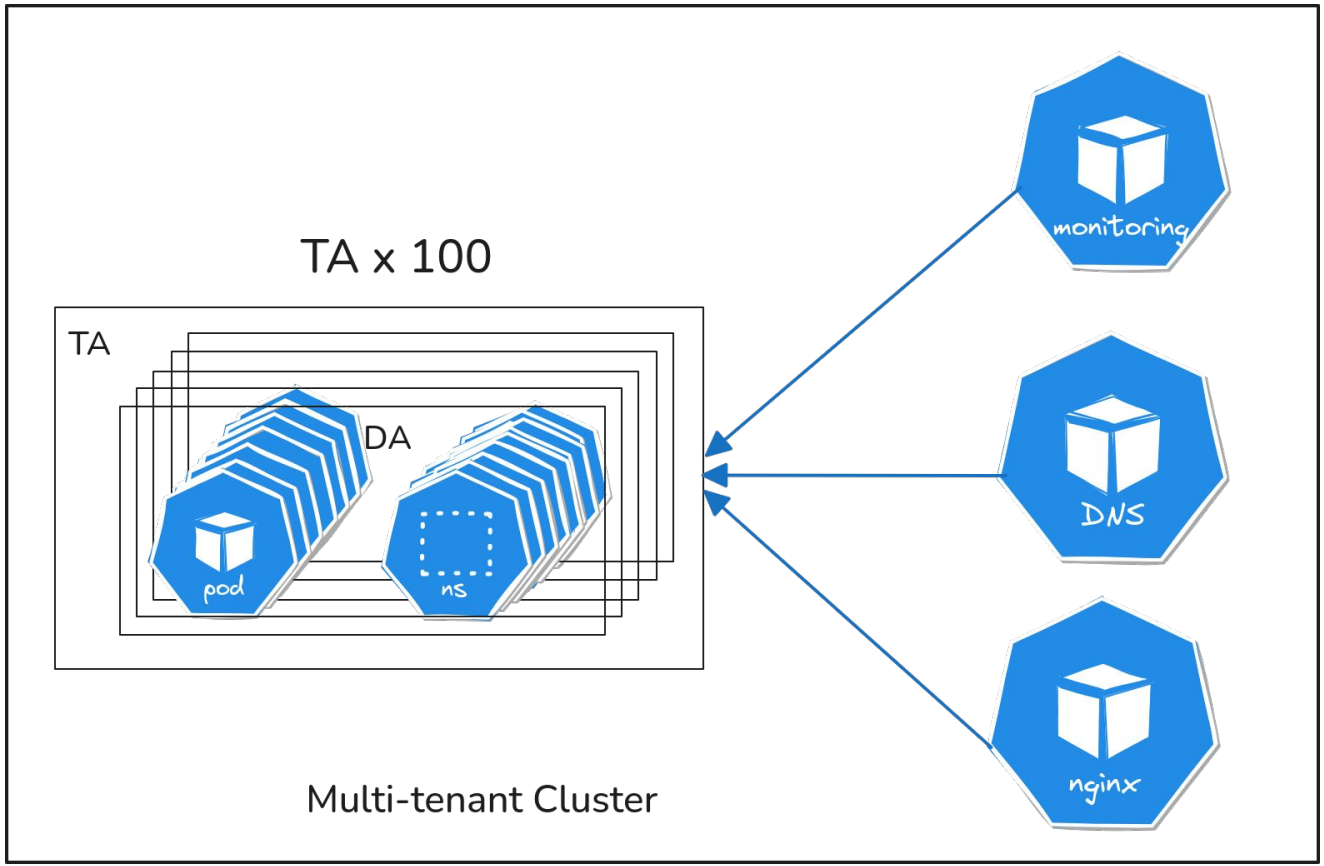
- Mastercard's default policy: "If it can be microsegmented, it SHOULD be... but if it is in the application data plane, it MUST be."
- Need to adhere to PCI Data Security Standard



Wiring up the declarative connectivity requirements

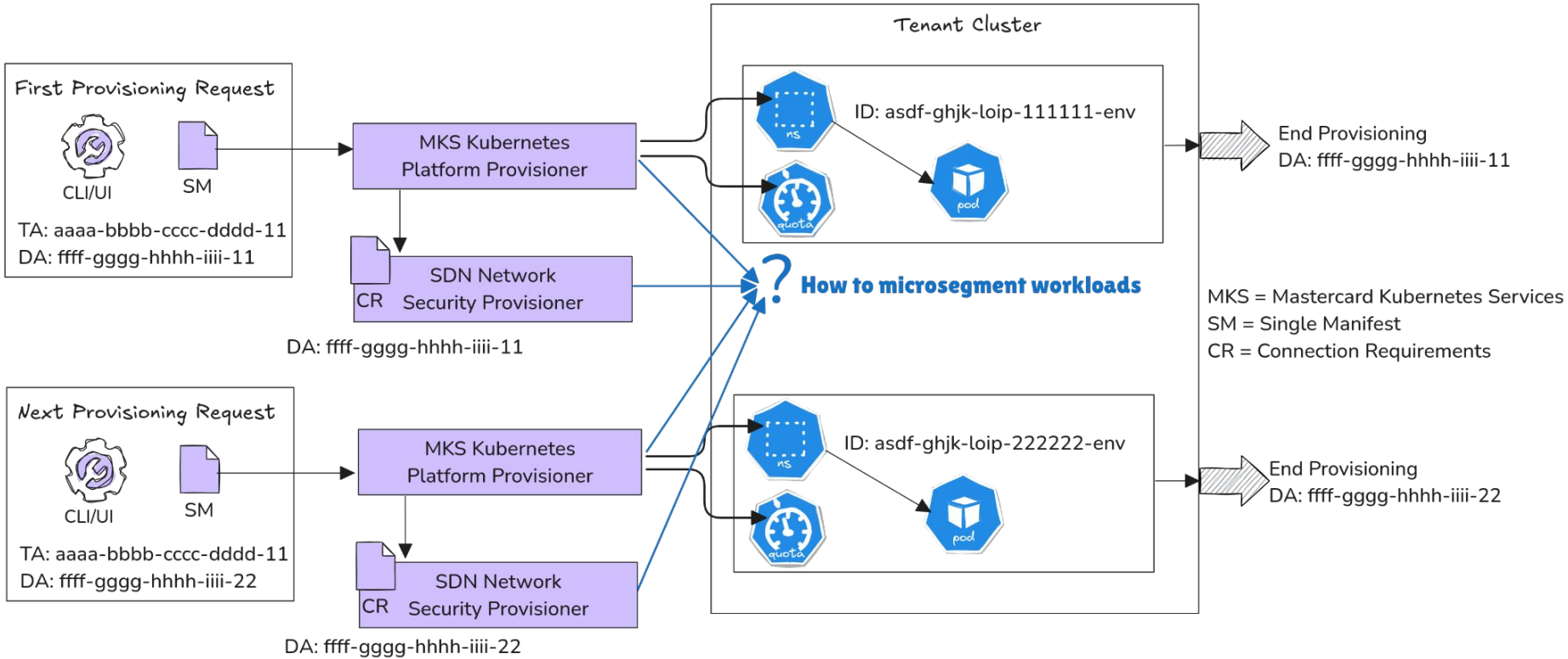


Wiring up the infrastructure connectivity requirements



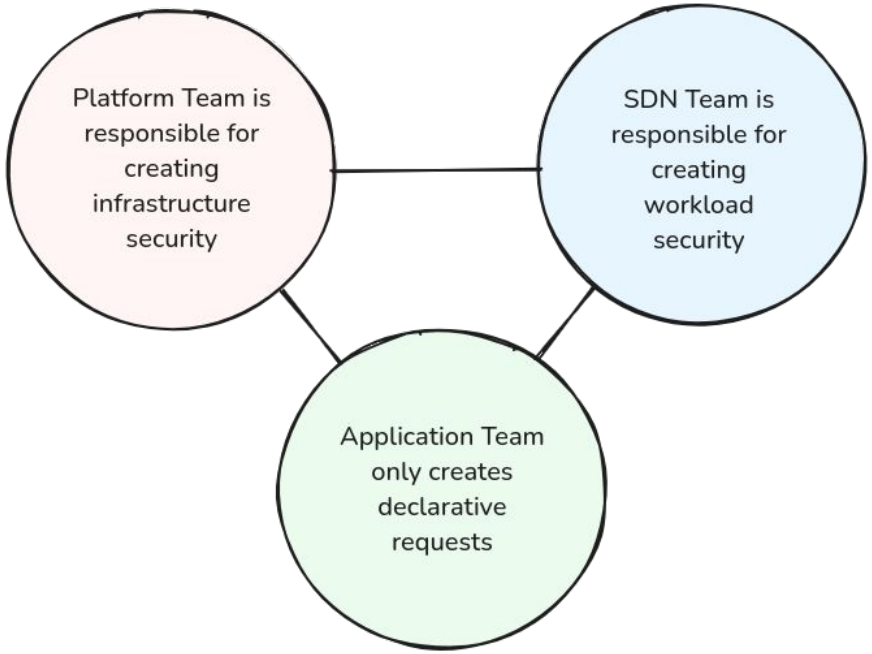
The platform team has a set of common connections we want to enforce on all TAs

Provision->Build->Deploy Workflow in Infrastructure



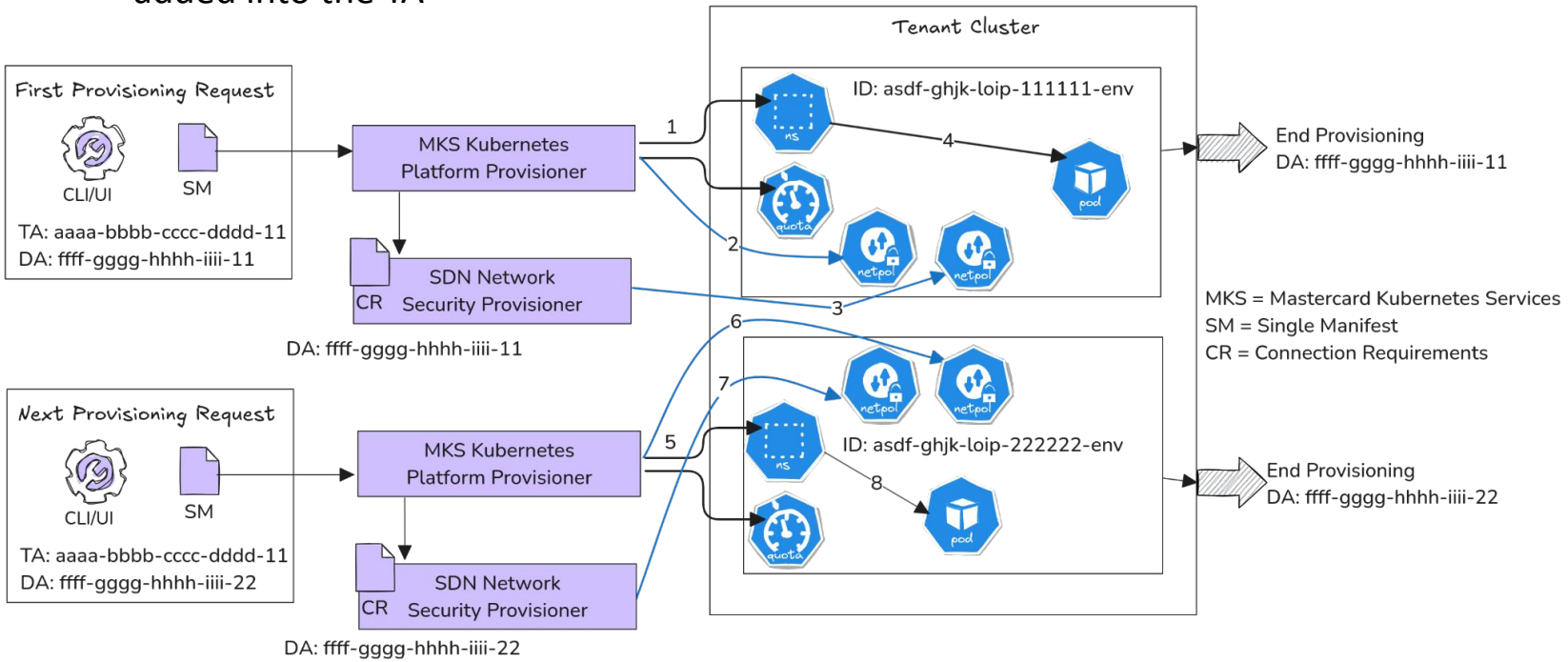
Why not use Network Policies?

- Need for a simple placeholder policy that met compliance regulations even before the workloads (ns+pod) are created
- SDN team needs to be proactive to App team requests (not reactive)
- Avoid duplication of Network Policy in every namespace



Why not use Network Policies?

- Dynamic nature of DAs with SMs
  - Multiple updates or multiple policies would need to be created when additional DAs get added into the TA



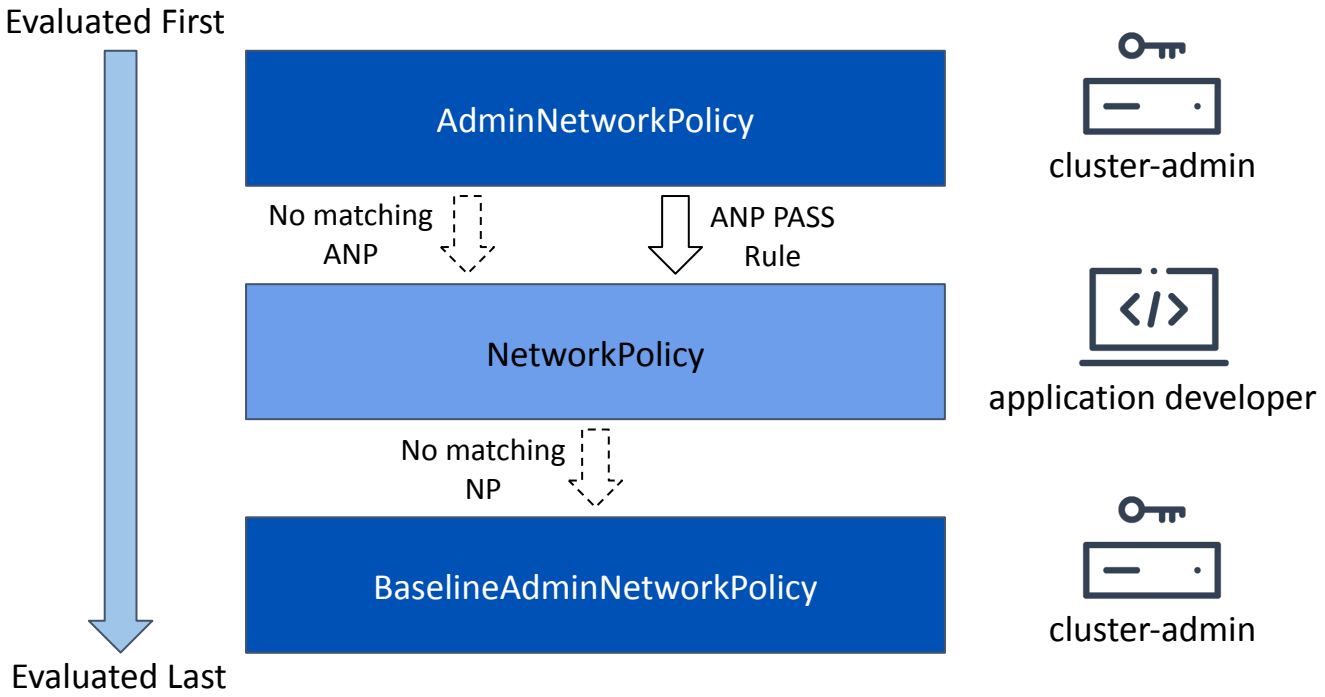
We wanted a Kubernetes upstream solution for this

- Around the same time in Kubernetes the sig-network-policy-API subproject was working on Admin Network Policy API
- Two cluster scoped CRDs:
  - **AdminNetworkPolicy**: Cannot be overridden by developer network policies
  - **BaselineAdminNetworkPolicy**: CAN be overridden by developer network policies
- Lives in kubernetes-sigs repo: Small, healthy community which participates in building new features around these APIs and maintains them
  - Current status: Alpha, on the journey to Beta


KubeCon
CloudNativeCon
North America 2024
AdminNetworkPolicy

Admin Network Policy

For the first iteration Mastercard does not plan to include network policies



Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

namespaces on which this policy applies

Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

namespaces on which this policy applies

first rule; holds highest precedence

Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

namespaces on which this policy applies

first rule; holds highest precedence

explicit action to perform (allow/deny/pass)

Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

namespaces on which this policy applies

first rule; holds highest precedence

explicit action to perform (allow/deny/pass)

namespaces from which connections are allowed (L3 match conditions)

Admin Network Policy API


```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
 name: sensitive-workload-control
spec:
 priority: 34
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring" # rule0
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         kubernetes.io/metadata.name: monitoring
   ports:
   - portNumber:
       protocol: TCP
       port: 7564
   - namedPort: "scrape"
<.....>
```

0 means highest priority and 1000 is the lowest

namespaces on which this policy applies

first rule; holds highest precedence

explicit action to perform (allow/deny/pass)

namespaces from which connections are allowed (L3 match conditions)

List of L4 match conditions (caution: this is always applied to the destination so to the subject pods in case of ingress)

Admin Network Policy API


```
<.....>
 - name: "pass-from-restricted-tenants" # rule1
   action: "Pass"
   from:
   - namespaces: # secure tenants
       matchLabels:
         tenant: secure
 - name: "default-deny" # rule2
   action: "Deny"
   from:
   - namespaces: {}
<.....>
```

second rule; holds second precedence

delegate decision making to network policies or baseline admin network policies

namespaces from which connections are denied (L3 match conditions)

Admin Network Policy API Nodes + Networks Peer


```
<.....>
egress:
 - name: "allow-to-kapi-server" # rule0
   action: "Allow"
   to:
   - nodes:
       matchExpressions:
       - key: node-role.kubernetes.io/control-plane
         operator: Exists
   ports:
   - portNumber:
       protocol: TCP
       port: 6443
- name: "default-deny" # rule1
   action: "Deny"
   to:
   - networks:
     - 0.0.0.0/0
<.....>
```

nodes to which connections are allowed (L3 match conditions)

networks to which connections are denied (L3 match conditions)

Baseline Admin Network Policy API

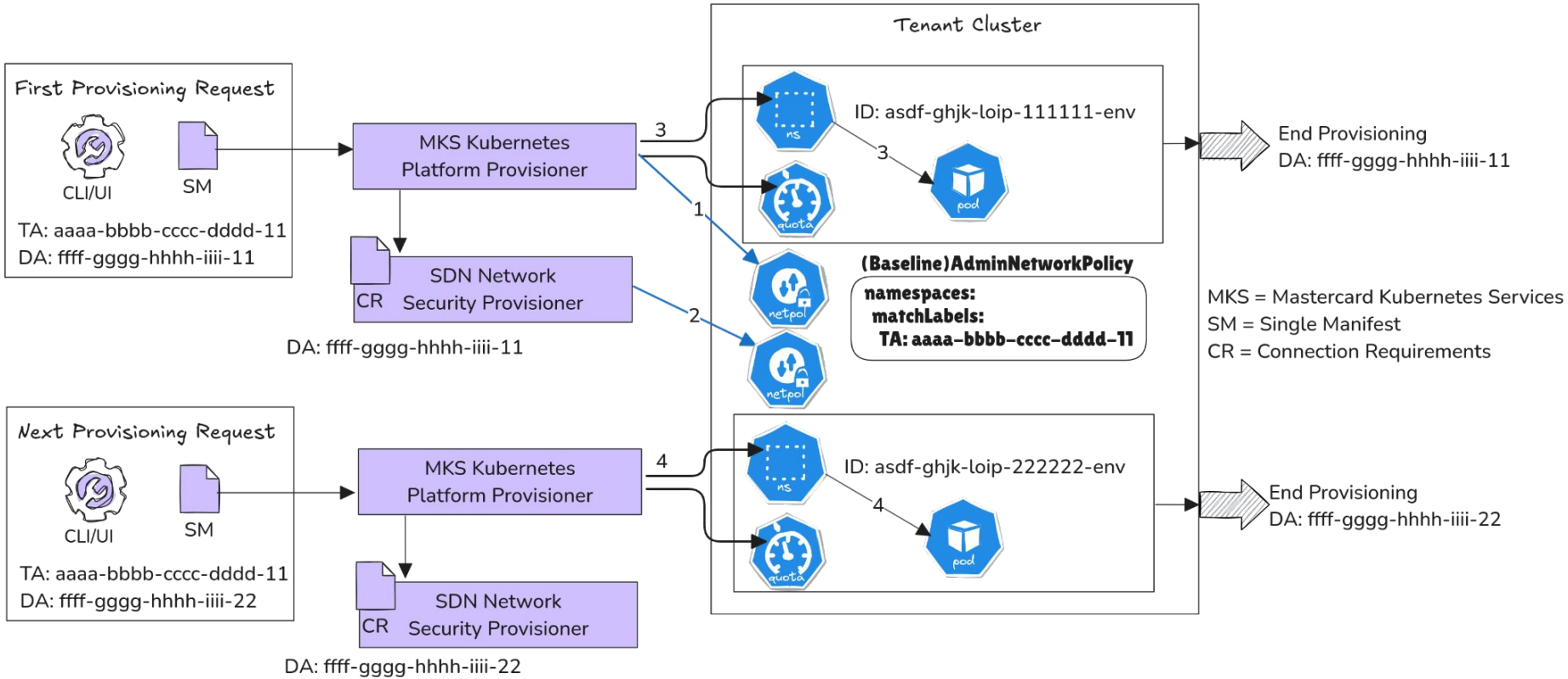

```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
 name: default
spec:
 subject:
   namespaces:
     matchLabels:
         tenant: true
 ingress:
 - name: "default-deny"
   action: "Deny"
   from:
   - namespaces: {}
 egress:
 - name: "default-deny"
   action: "Deny"
   to:
   - namespaces: {}
```

namespaces from which connections are denied (L3 match conditions)

namespaces to which connections are denied (L3 match conditions)

BANP has same API fields as ANP except its singleton thus has no priority; no PASS action

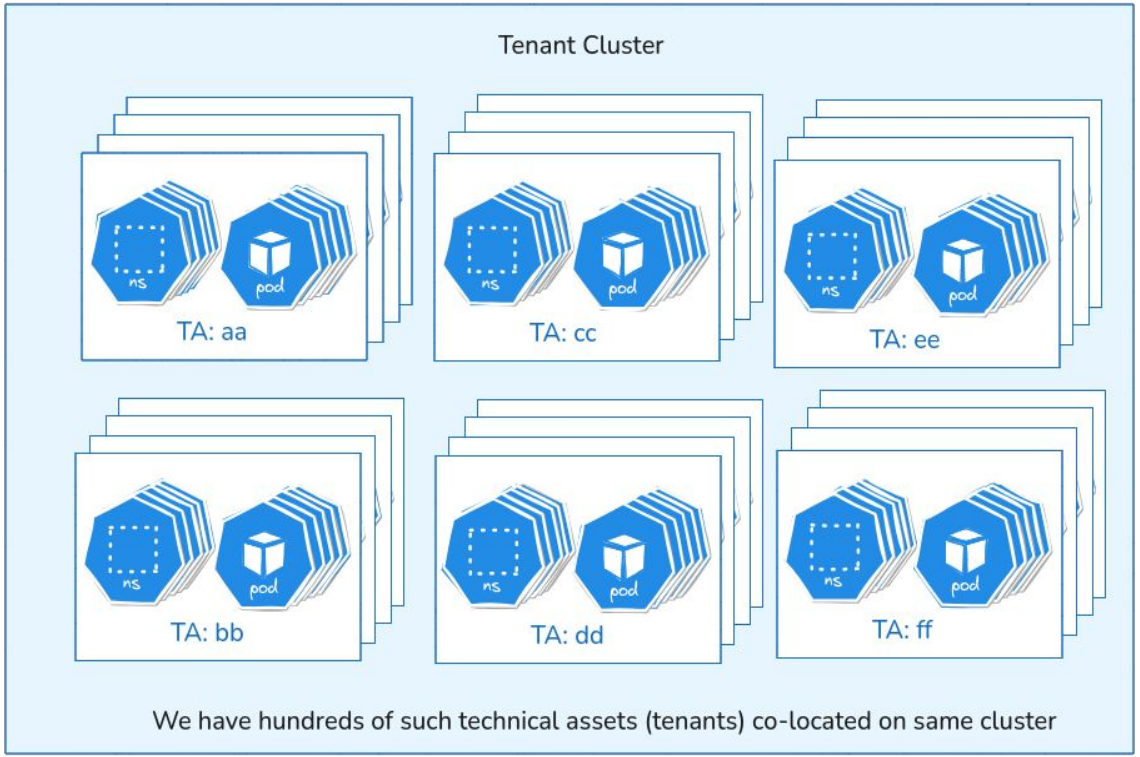
Pre Declaring ANPs that select Technical Assets




KubeCon
CloudNativeCon
North America 2024
DEMO Time!

DEMO’s Architecture Diagram: Setup

- 1 - Multi-Tenant Cluster
- 10 - Tenants (TAs)
- 5 - Deployable Assets (DA) per tenant
- 1 - Namespace per DA
- 1 - BANP deployed
- 1 - Platform ANP (DNS)
- 10 - Tenant Scoped ANP’s




KubeCon
CloudNativeCon
North America 2024
ANP Caveats

AdminNetworkPolicy API Usage Caveats

One has to explicitly DENY the traffic, there are no implicit DENY's like with Network Policies

```
- name: "default-deny" # rule2
  action: "Deny"
  from:
    ...
```

For ingress, at max a "deny-all-traffic" means denying all traffic from all namespaces in your cluster. Unlike network policies there is no way to ask for "deny-everything"

```
- name: "default-deny" # rule2
  action: "Deny"
  from:
  - namespaces: {}
```

AdminNetworkPolicy API Usage Caveats

Caution when using empty selectors, specially in your subject - so you can potentially lock the cluster up

```
subject:
   namespaces: {}
```

Specifying namespace and pod selectors to match on subjects and peers does NOT include host-networked pods

```
to:
- pods:
    namespaceSelector:
      matchLabels:
        tenant: true
    podSelector:
      matchLabels:
        env: prod
```

AdminNetworkPolicy API Usage Caveats

nodes and networks peers are only available in Egress rules (subject to change in future depending on this NPEP)

```
to:
   - nodes:
...
   - networks:
...
```

Hence no need to worry about allowing ingress traffic from host network like is required for kubelet health checks when network policies are applied

AdminNetworkPolicy API Usage Caveats

Use “allow” and “deny” when you are sure you want to at any cost make that happen. Once you put these you cannot have finer controls using NetworkPolicies

```yaml
metadata:
 name: sensitive-workload-control
spec:
 priority: 30
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "pass-from-intra-cluster"
   action: "Pass"
   from:
   - namespaces:
       matchLabels:
         name: production
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-from-production
 namespace: tenant-A
spec:
 podSelector: {}
 policyTypes:
   - Ingress
 ingress:
 - from:
   - namespaceSelector:
       matchLabels:
         name: production
```

AdminNetworkPolicy API Usage Caveats

Don’t create multiple admin network policies at the same priority that have overlapping rule definitions - in this case only of those rules will apply - which one gets applied is “implementation defined”

```yaml
metadata:
 name: platform-team-controls
spec:
 priority: 44
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "allow-from-monitoring"
   action: "Allow"
   from:
   - namespaces:
       matchLabels:
         name: monitoring
```

```yaml
metadata:
 name: sdn-team-controls
spec:
 priority: 44
 subject:
   namespaces:
     matchLabels:
       tenant: true
 ingress:
 - name: "deny-from-monitoring"
   action: "Deny"
   from:
   - namespaces:
       matchLabels:
         name: monitoring
```

BaselineAdminNetworkPolicy API Usage Caveats

Given BANP is a singleton use it wisely - setting default deny's for all your workload namespaces -> forces tenants to add explicit allows and think about security

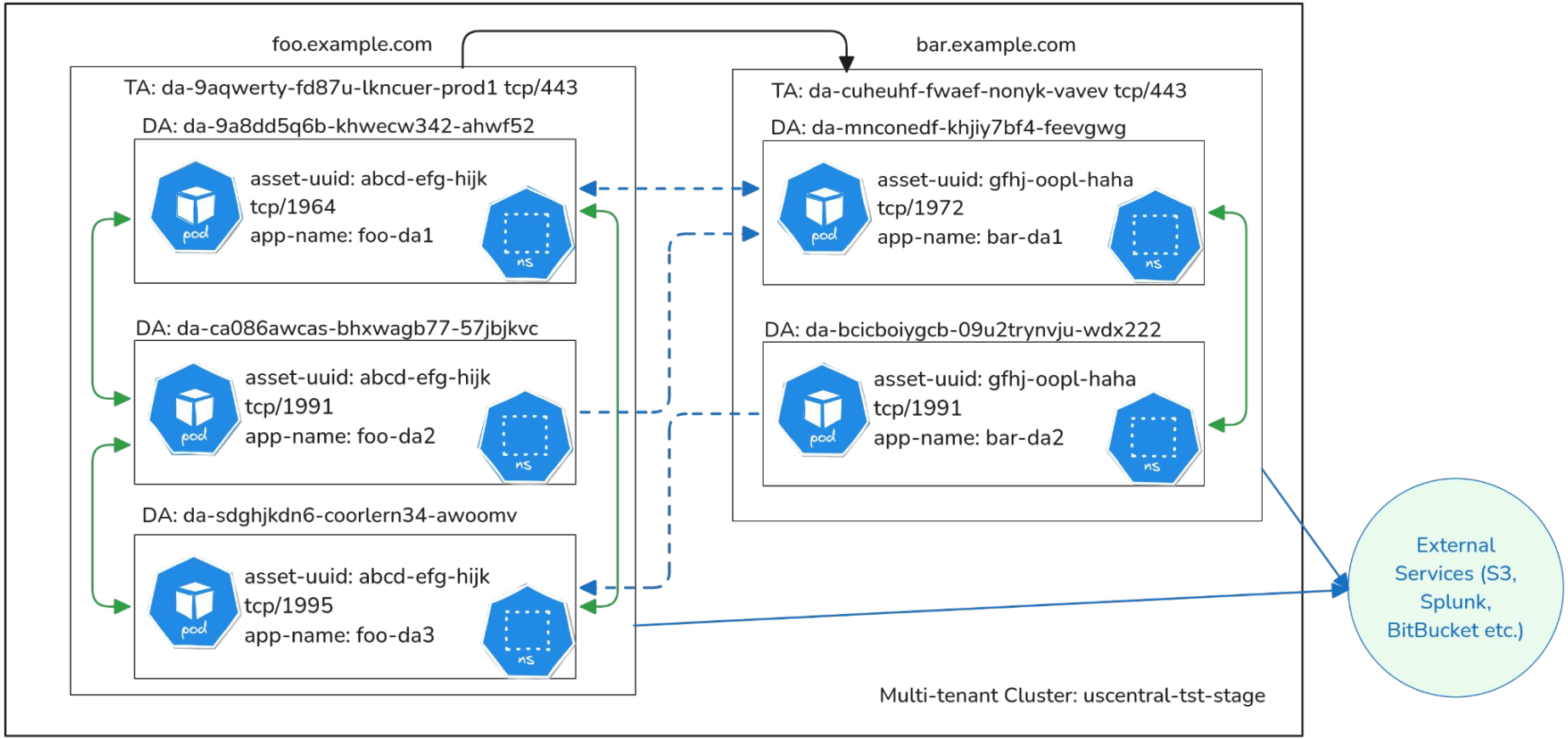
```yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
 name: default
spec:
 subject:
   namespaces:
     matchLabels:
         tenant: true
 ingress:
 - name: "default-deny"
   action: "Deny"
   from:
   - namespaces: {}
 egress:
 - name: "default-deny"
   action: "Deny"
   to:
   - namespaces: {}
```


KubeCon
CloudNativeCon
North America 2024
Lessons Learnt &&
What's Next?

Lessons Learned

- **Microsegmentation Controls**: Designing connections requires collaboration across multiple teams, including the platform team.
- **PCI DSS Compliance**: This is more than microsegmentation is about becoming secure by default.
- **Collaborative Design**: Adopting open standards and collaborating with the community, specifically the sig-network-policy-api WG.
- **Start Simple**: Begin with a minimalistic approach to reduce complexity and yet still leave room to build in finer controls as needed.

Future with AdminNetworkPolicy + NetworkPolicy



KubeCon | CloudNativeCon

North America 2024

Questions?

Feedback on the talk is welcome!->

KubeCon | CloudNativeCon

North America 2024

Leveraging Priority Field between Platform & SDN Teams

- Platform team creates the ANPs for defining connections to infrastructure components from the microservices
  - Can use a range of 20-40
- SDN team is responsible for defining the connections between microservices
  - Can use a range of 40-60
- This way even upon accidental overlap platform team’s rules will be evaluated first

DEMO

- We have 10 tenants each with 5 Deployable Assets
- Tenant AA & Tenant BB for the demo
- We need a BANP {} {} to isolate tenants
- After BANP no inter or intra tenant connectivity
  - TAAA -> TAAA Backend(s) fails

- Define ANP's for each Tenant (Allow) →
  - TAAA -> TAAA Backend(s) still fails, because DNS is missing
  - Deploy an Infra ANP for the cluster with DNS ANP
  - TAAA -> TAAA Backend(s) SUCCESS!
  - TAAA -> TABB Backend(s) BLOCKED!

- Kapi server still doesn't work because we have BANP denying it