Use case **1**
OSSM resources 2
caas-ekad-lab02 cluster 2
caas-ekad-ref01 cluster 4
RBAC: Authorization Policies 6
Conclusion 8

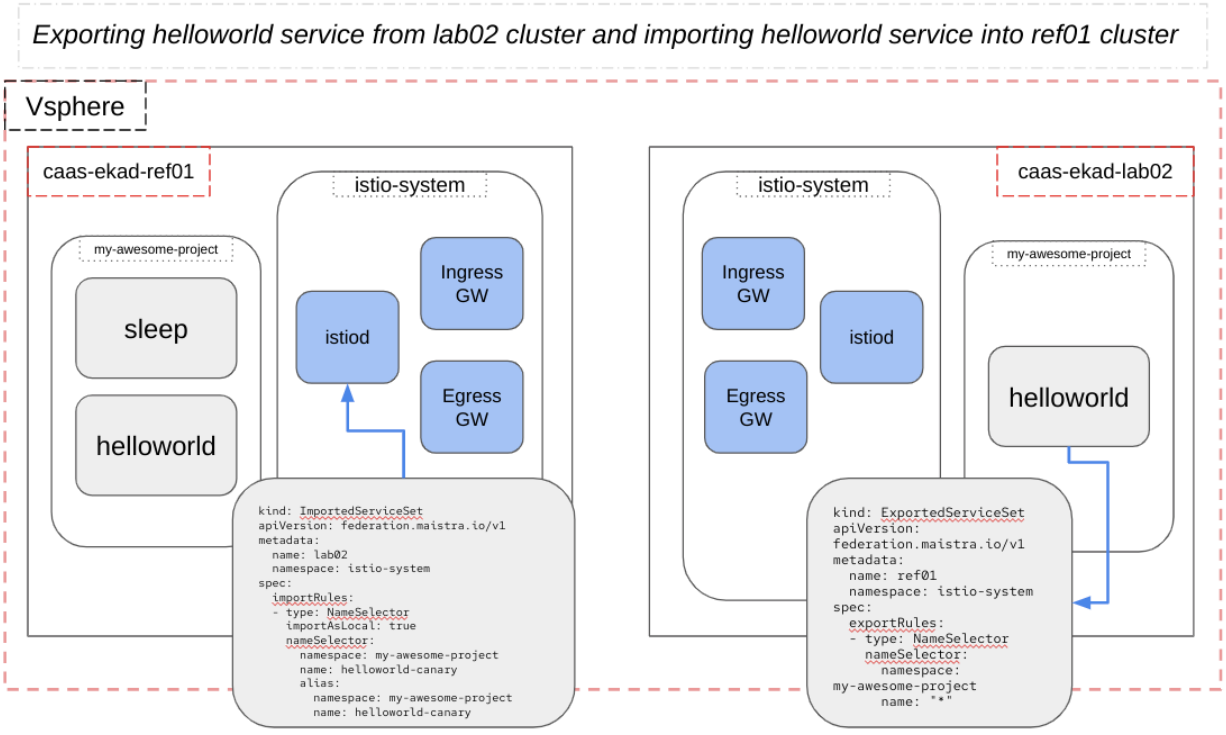
Use case

The customer's requirement is using RBAC - Authorization Policies in a multicluster setup environment with OSSM federation.

We are doing a PoC with two different clusters installed in vSphere. Both clusters are federated by creating a ServiceMeshPeer. So far, all works properly.

Clusters:

- caas-ekad-ref01
- caas-ekad-lab02



The scenario is exporting a service from *caas-ekad-lab02* cluster, and importing that service into *caas-ekad-ref01* cluster.

The sample application deployed in both clusters is *helloworld*, this is the application accessed in our PoC.

An additional application is used for accessing the *helloworld* application: *sleep*. Just a curl command is executed here to access both *helloworld* applications.

Requirement: The key requirement is ***applying an AuthorizationPolicy resource in both clusters for both helloworld applications. The AuthorizationPolicy will verify the identity of the source application (sleep in our PoC).***

OSSM resources

caas-ekad-lab02 cluster

ServiceMeshPeer

```
Unset
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
 name: caas-ekad-ref01
 namespace: istio-system
spec:
 remote:
  addresses:
  - 192.168.123.10
  discoveryPort: 8188
  servicePort: 15443
 gateways:
  ingress:
   name: caas-ekad-ref01-ingress
  egress:
   name: caas-ekad-ref01-egress
 security:
  trustDomain: caas-ekad-ref01.local
  clientID:
caas-ekad-ref01.local/ns/istio-system/sa/caas-ekad-lab02-egress-service-account
  certificateChain:
   kind: ConfigMap
```

```
    name: caas-ekad-ref01-ca-root-cert
```

ExportedServiceSet

```
Unset
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
 name: caas-ekad-ref01
 namespace: istio-system
spec:
 exportRules:
 - type: NameSelector
  nameSelector:
   namespace: my-awesome-project
   name: "*"
```

Services exported:

```
spec:
  exportRules:
  - nameSelector:
      name: '*'
      namespace: my-awesome-project
    type: NameSelector
status:
  exportedServices:
  - exportedName: helloworld.my-awesome-project.svc.caas-ekad-ref01-exports.local
    localService:
      hostname: helloworld.my-awesome-project.svc.cluster.local
      name: helloworld
      namespace: my-awesome-project
kind: List
```

caas-ekad-ref01 cluster

ServiceMeshPeer

```
Unset
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
 name: caas-ekad-lab02
 namespace: istio-system
spec:
 remote:
  addresses:
  - 192.168.160.10
  discoveryPort: 8188
  servicePort: 15443
 gateways:
  ingress:
   name: caas-ekad-lab02-ingress
  egress:
   name: caas-ekad-lab02-egress
 security:
  trustDomain: caas-ekad-lab02.local
  clientID:
caas-ekad-lab02.local/ns/istio-system/sa/caas-ekad-ref01-egress-service-account
  certificateChain:
   kind: ConfigMap
   name: caas-ekad-lab02-ca-root-cert
```

ImportedServiceSet with *importAsLocal* set to False

```
Unset
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
 name: caas-ekad-lab02
 namespace: istio-system
spec:
 importRules:
 - type: NameSelector
  importAsLocal: false
  nameSelector:
```

```
    namespace: my-awesome-project
    name: helloworld
    alias:
     namespace: global
     name: helloworld
  - type: NameSelector
   importAsLocal: false
   nameSelector:
    namespace: my-awesome-project
    name: "*"
  locality:
   region: caas-ekad-lab02
```

ImportedServiceSet with *importAsLocal* set to True

```
Unset
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
 name: caas-ekad-lab02
 namespace: istio-system
spec:
 importRules:
 - type: NameSelector
  importAsLocal: true
  nameSelector:
   namespace: my-awesome-project
   name: helloworld
   alias:
    namespace: my-awesome-project
    name: helloworld
 - type: NameSelector
  importAsLocal: false
  nameSelector:
   namespace: my-awesome-project
   name: "*"
 locality:
  region: caas-ekad-lab02
```

RBAC: Authorization Policies

The following Authorization Policy resources has been tested without success in the caas-ekad-lab02 cluster, where the service is exported:

AP using ServiceAccount:

```
Unset
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: helloworld
 namespace: my-awesome-project
spec:
 selector:
  matchLabels:
   app: helloworld
 action: AUDIT
 rules:
 - from:
  - source:
    principals:
    - caas-ekad-ref01.local/ns/my-awesome-project/sa/sleep
  to:
  - operation:
    methods:
    - GET
```

AP using xfcc header:

```
Unset
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: helloworld
 namespace: my-awesome-project
spec:
 selector:
  matchLabels:
   app: helloworld
 action: AUDIT
```

```
 rules:
 - to:
  - operation:
    methods:
    - GET
 - when:
  - key: request.regex.headers[x-forwarded-client-cert]
   values:
   -
.*caas-ekad-lab02.local/ns/istio-system/sa/caas-ekad-ref01-ingress-service-acco
unt*.*
```

As described in the official OCP documentation, the original spiffe is not propagated between clusters:

- For exported services, their target services will only see traffic from the ingress gateway, not the original requestor (that is, they won't see the client ID of either the other mesh's egress gateway or the workload originating the request)

Watching the istio-proxy's log, we can confirm that the SA analyzed by the rbac module is the ingressgateway one:

```
Unset
2023-10-18T10:33:00.434799Z  debug  envoy rbac  checking request:
requestedServerName:
outbound_.5000_._.helloworld.my-awesome-project.svc.cluster.local, sourceIP:
10.128.2.58:45322, directRemoteIP: 10.128.2.58:45322, remoteIP:
10.128.2.58:45322,localAddress: 10.131.0.61:5000, ssl: uriSanPeerCertificate: spif
fe://caas-ekad-lab02.local/ns/istio-system/sa/caas-ekad-ref01-ingress-service-a
ccount, dnsSanPeerCertificate: , subjectPeerCertificate: , headers: ':authority',
'helloworld.my-awesome-project.svc.cluster.local:5000'
```

Conclusion

We can confirm that the product has been designed to not propagate the original spiffe between federated clusters. Thus, the PoC result is unsuccessful.

At this point, how can we implement RBAC in a federated environment by applying Authorization Policies? The Authorization Policy can be set by trusting the Ingress Gateway's Service Account, but thereby we can't differentiate the original requester.