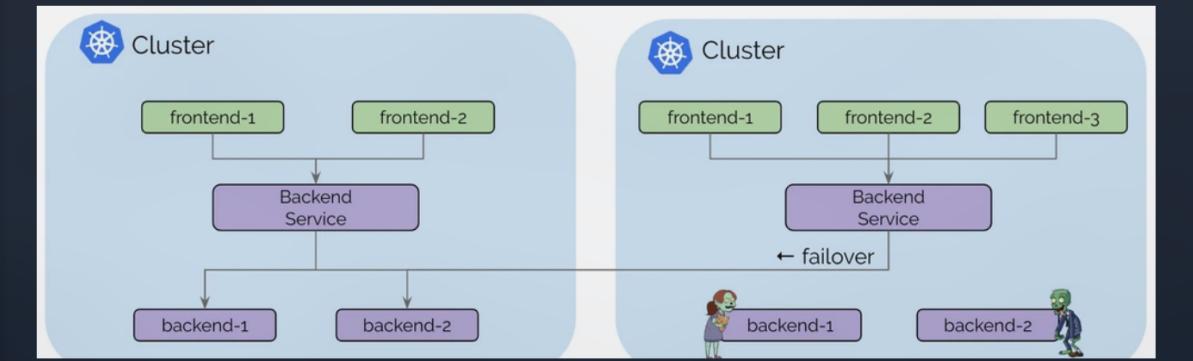


Installing the clusters

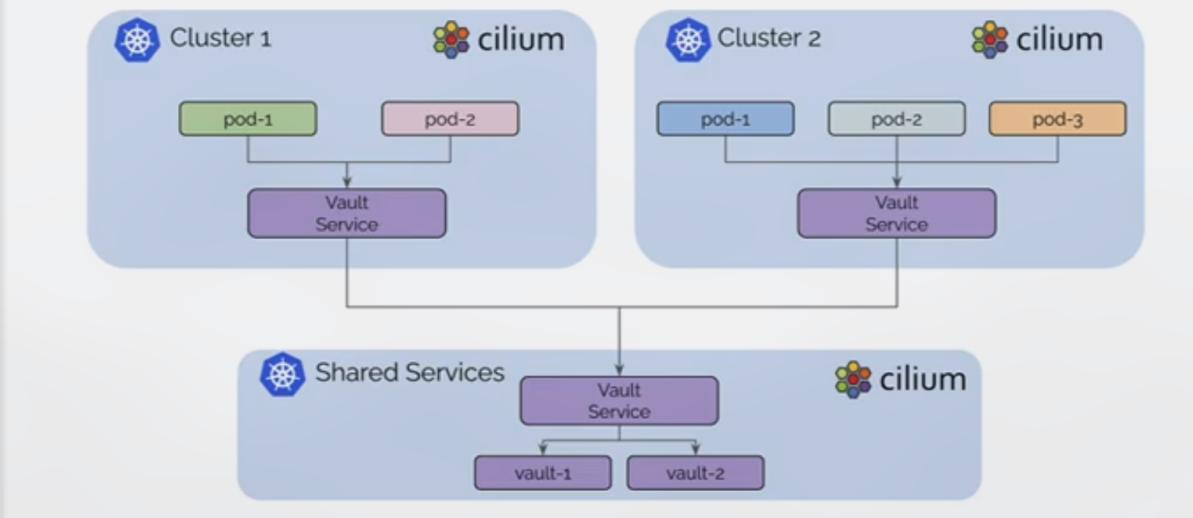
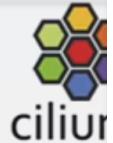
Fault Resilience

After the services are deployed and made global, we will see how this can be used to improve availability and fault tolerance of multi-cluster applications.



Cluster Mesh

Centralized Services Connectivity



Cluster Mesh



Requirements

- Non-conflicting Pod CIDR ranges and unique IP addresses between all clusters.
- IP Connectivity between clusters using e.g. VPN
- Load Balancer for Cluster Mesh API Server
- The network between clusters must allow the inter-cluster communication.

In this lab, we will create two Kind clusters and mesh them using Cilium.

We'll have two requirements for these clusters:

1. disable default CNI so we can easily install Cilium
2. use disjoint pods and services subnets

```
'''yaml
cat kind_koornacht.yaml
---
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
networking:
  disableDefaultCNI: true
  podSubnet: 10.1.0.0/16
  serviceSubnet: 172.20.1.0/24
nodes:
- role: control-plane
  extraPortMappings:
    # localhost.run proxy
    - containerPort: 32042
      hostPort: 32042
    # Hubble relay
    - containerPort: 31234
      hostPort: 31234
    # Hubble UI
    - containerPort: 31235
      hostPort: 31235
- role: worker
- role: worker
'''
```

```
```bash
cilium install --cluster-name abcd --cluster-id 1 --ipam kubernetes
cilium hubble enable
cilium status
````
```

```
```bash
cat kind_tion.yaml
````

apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
networking:
  disableDefaultCNI: true
  podSubnet: 10.2.0.0/16
  serviceSubnet: 172.20.2.0/24
nodes:
- role: control-plane
- role: worker
- role: worker
````
```

This Tion cluster will also feature one control-plane node and 2 worker nodes, but it will use 10.2.0.0/16 for the Pod network, and 172.20.2.0/24 for the Services.

```
cilium install --cluster-name tion --cluster-id 2 --ipam kubernetes
cilium status
```

The image shows a quiz interface with a dark blue background. On the left, there is a large rectangular area containing five questions, each with a green square icon and a checkbox. On the right, there is a vertical column with a title and a descriptive text.

| Question                                                    | Answer                              |
|-------------------------------------------------------------|-------------------------------------|
| --cluster-id must be set to different values                | <input checked="" type="checkbox"/> |
| Clusters must have the same number of nodes                 | <input type="checkbox"/>            |
| Clusters must use the same version of Kubernetes            | <input type="checkbox"/>            |
| Cluster CIDRs must not overlap                              | <input checked="" type="checkbox"/> |
| Using the Cilium CLI makes it easier to set up Cluster Mesh | <input checked="" type="checkbox"/> |

**? Cilium Install Quiz**

When installing Cilium on clusters that will need to be meshed (select all answers that apply)

# Cluster mesh adding

## Cluster Mesh Architecture

When activating Cluster Mesh on Cilium clusters, a new Control Plane is deployed to manage the mesh for this cluster, along with its etcd key-value store.

Agents of other clusters can then access this Cluster Mesh Control Plane in read-only mode, allowing them to access metadata about the cluster, such as Service names and corresponding IPs.

The diagram shows two separate Kubernetes clusters, Cluster 1 and Cluster 2, each with its own IP range (10.100.0.0/16 and 10.200.0.0/16 respectively). Within each cluster, there are multiple pods connected to a central 'clustermesh apiserver' via 'agent' components. These agents are also connected to a 'TLS' secured 'LB' (Load Balancer). A double-headed arrow labeled 'Watching (read-only)' connects the two clusters, indicating that the apiservers are monitoring each other's etcd stores for read-only access.

```
```bash
```

```
cilium clustermesh enable --context kind-koornacht --service-type NodePort  
cilium clustermesh enable --context kind-tion --service-type NodePort
```

```
cilium clustermesh status --context kind-koornacht --wait  
cilium clustermesh status --context kind-tion --wait
```

```
cilium clustermesh connect --context kind-koornacht --destination-context kind-tion
```

```
cilium clustermesh status --context kind-koornacht --wait
```

```
cilium clustermesh status --context kind-tion --wait
```

```
```
```

Clustermesh needs to be enabled on both clusters

It is recommended to use a `NodePort` setup for production

Clustermesh status can be checked using the Cilium CLI

The clustermesh connect command must be issued on both clusters

Multiple answers can be checked as correct.

**Meshing Clusters Quiz**

When meshing Kubernetes clusters (select all answers that apply)

```
Cluster Pods: clustermesh-apiserver Running: 1
Image versions cilium
 clustermesh-apiserver
 clustermesh-apiserver
 cilium-operator
 quay.io/cilium/cilium:v1.12.2@sha256:986f8b04cfdb35cf71470
 quay.io/coreos/etcd:v3.4.13: 1
 quay.io/cilium/clustermesh-apiserver:v1.12.2: 1
 quay.io/cilium/operator-generic:v1.12.2@sha256:00508f78dae

*kind-tion in ~
❯ cilium clustermesh connect --context kind-koornacht --destination-context kind-tion
💡 Extracting access information of cluster tion...
🔑 Extracting secrets from cluster tion...
⚠ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ Found ClusterMesh service IPs: [172.18.0.5]
💡 Extracting access information of cluster koornacht...
🔑 Extracting secrets from cluster koornacht...
⚠ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ Found ClusterMesh service IPs: [172.18.0.4]
💡 Connecting cluster kind-koornacht -> kind-tion...
🔑 Secret cilium-clustermesh does not exist yet, creating it...
🔑 Patching existing secret cilium-clustermesh...
💡 Patching DaemonSet with IP aliases cilium-clustermesh...
💡 Connecting cluster kind-tion -> kind-koornacht...
🔑 Secret cilium-clustermesh does not exist yet, creating it...
🔑 Patching existing secret cilium-clustermesh...
💡 Patching DaemonSet with IP aliases cilium-clustermesh...
✅ Connected cluster kind-koornacht and kind-tion!

*kind-tion in ~
❯ cilium clustermesh status --context kind-koornacht --wait
⚠ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
✅ Cluster access information is available:
- 172.18.0.4:30160
✅ Service "clustermesh-apiserver" of type "NodePort" found
⌚ [kind-koornacht] Waiting for deployment clustermesh-apiserver to become ready...
✅ All 3 nodes are connected to all clusters [min:1 / avg:1.0 / max:1]
👉 Cluster Connections:
- tion: 3/3 configured, 3/3 connected
✗ Global services: [min:3 / avg:3.0 / max:3]

*kind-tion in ~
❯ cilium clustermesh status --context kind-tion --wait
⚠ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
✅ Cluster access information is available:
- 172.18.0.5:32198
✅ Service "clustermesh-apiserver" of type "NodePort" found
⌚ [kind-tion] Waiting for deployment clustermesh-apiserver to become ready...
✅ All 3 nodes are connected to all clusters [min:1 / avg:1.0 / max:1]
👉 Cluster Connections:
- koornacht: 3/3 configured, 3/3 connected
✗ Global services: [min:5 / avg:5.0 / max:5]

*kind-tion in ~
❯
```

# Deploy the applications

## Deploying an application

We will now deploy a sample application on both Kubernetes clusters.

This application will contain two deployments:

- a simple HTTP application called `rebel-base`, which will return a static JSON document
- an `x-wing` pod which we will use to make requests to the `rebel-base` service from within the cluster

The only difference between the two deployments will be the `ConfigMap` resource deployed, which will contain the static JSON document served by `rebel-base`, and whose content will depend on the cluster.

Are you ready? Let's go!

```
```yaml
```

```
kubectl config use kind-koornacht
```

```
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rebel-base
spec:
  selector:
    matchLabels:
      name: rebel-base
  replicas: 2
  template:
    metadata:
      labels:
        name: rebel-base
    spec:
      containers:
        - name: rebel-base
          image: docker.io/nginx:1.15.8
          volumeMounts:
            - name: html
              mountPath: /usr/share/nginx/html/
      livenessProbe:
        httpGet:
          path: /
          port: 80
      periodSeconds: 1
      readinessProbe:
        httpGet:
```

```

    path: /
    port: 80
  volumes:
    - name: html
      configMap:
        name: rebel-base-response
      items:
        - key: message
          path: index.html
  ---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: x-wing
spec:
  selector:
    matchLabels:
      name: x-wing
  replicas: 2
  template:
    metadata:
      labels:
        name: x-wing
    spec:
      containers:
        - name: x-wing-container
          image: docker.io/cilium/json-mock:1.2
      livenessProbe:
        exec:
          command:
            - curl
            - -sS
            - -o
            - /dev/null
            - localhost
      readinessProbe:
        exec:
          command:
            - curl
            - -sS
            - -o
            - /dev/null
            - localhost

```

```

kubectl apply -f configmap_koornacht.yaml -o yaml
apiVersion: v1
data:

```

```

message: |
  {"Cluster": "Koornacht", "Planet": "N'Zoth"}
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"message":"{\"Cluster\": \"Koornacht\", \"Planet\": \"N'Zoth\"\\n\"},\"kind\":\"ConfigMap\",\"metadata\":{\"annotations\":{},\"name\":\"rebel-base-response\",\"namespace\":\"default\"}}}

---

```

```

apiVersion: v1
kind: Service
metadata:
  name: rebel-base
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    name: rebel-base
```

```

```

@kind-koornacht in ~
❯ kubectl exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
{"Cluster": "Koornacht", "Planet": "N'Zoth"}

```

```  
kubectl config use kind-tion

```

apiVersion: v1
data:
  message: |
    {"Cluster": "Tion", "Planet": "Foran Tutha"}
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

```

```
{"apiVersion": "v1", "data": {"message": "{\"Cluster\": \"Tion\", \"Planet\": \"Foran\nTuthal\"}\n"}, "kind": "ConfigMap", "metadata": {"annotations": {}, "name": "rebel-base-res\nponse", "namespace": "default"}\n}
```

Making services global



Making Services Global

When two or more clusters are meshed, Cilium allows you to set services as global in one or more clusters, by adding an annotation to them:

```
io.cilium/global-service: "true"
```

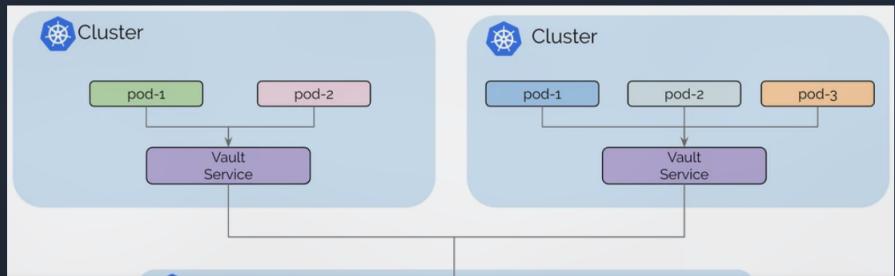
When this annotation is set, requests to this service will load-balance to all available services with the same name and namespace in all meshed clusters.



Shared services

Another use case of global services is shared services.

This is particularly useful when sharing stateful services between multiple Kubernetes clusters. If all clusters are meshed, the stateless applications spread among multiple clusters can all access the stateful services located on a single shared cluster.



```

❸ kind-tion in ~
❯ kubectl --context kind-koornacht annotate service rebel-base io.cilium/global-service="true"
service/rebel-base annotated

❸ kind-tion in ~
❯ kubectl --context kind-tion exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
{"Cluster": "Tion", "Planet": "Foran Tutha"}

❸ kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}

❸ kind-tion in ~
❯ kubectl --context kind-tion annotate service rebel-base io.cilium/global-service="true"
service/rebel-base annotated

❸ kind-tion in ~
❯ kubectl --context kind-tion exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}

❸ kind-tion in ~

```

```

kind-tion in ~
❯ kubectl --context kind-tion exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
{"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}

❸ kind-tion in ~
❯ kubectl --context kind-tion annotate service rebel-base io.cilium/global-service="true"
service/rebel-base annotated

❸ kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}}
 {"Cluster": "Koornacht", "Planet": "N'Zoth"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}

❸ kind-tion in ~
❯ kubectl --context kind-koornacht scale deployment rebel-base --replicas 0
deployment.apps/rebel-base scaled

❸ kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}

❸ kind-tion in ~
❯ kubectl --context kind-tion exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl rebel-base; done'
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}
 {"Cluster": "Tion", "Planet": "Foran Tutha"}}


```

Now check the replies when querying from the Koornacht cluster:

```

❯ kubectl --context kind-koornacht exec -ti d
[REDACTED]

```

And from the Tion cluster:

```

❯ kubectl --context kind-tion exec -ti deploy
[REDACTED]

```

You should see only entries like:

```

{"Cluster": "Tion", "Planet": "Foran Tutha"}}
[REDACTED]

```

You can see that requesting the service on both clusters now only yields answers from the Tion cluster, effectively making up for the missing pods on the Koornacht cluster.

We've now seen how clusters can access all instances of an identical service across meshed cluster. What if we want to remove one specific instance of the service from the global service? We'll see how to do this in the next challenge!

Allows cluster 'mycluster' to access all instances of service 'mysvc' in all meshed clusters

Allows cluster 'mycluster' to access all instances of service 'mysvc' in all meshed clusters

Allows all meshed clusters to access service 'mysvc' from cluster 'mycluster'

Load balances requests to service 'mysvc' made from 'mycluster' to both 'mycluster' and other meshed clusters

Load balances requests from all clusters to all instances of service 'mysvc'

Multiple answers can be checked as correct.

?

Global Services Quiz

Setting up a global service named `mysvc` in cluster `mycluster` (select all answers that apply)

Global and shared service



Global vs Shared

We've seen how the `io.cilium/global-service` annotation allows for a cluster to load-balance requests to a service to all meshed clusters with the same service.

What if you want to remove the service of a specific cluster from the global service?

The `io.cilium/shared-service` annotation can be used for this.



Disabling Global Service

By default, a service marked as global is considered shared as well, so the value of `io.cilium/shared-service` is `true` for all clusters when the service is marked as global in at least one cluster in the mesh.

Setting it to `false` in a cluster removes that specific service from the global service:

```
io.cilium/shared-service: "false"
```

Prevents cluster 'mycluster' from accessing service 'mysvc' on other clusters in the mesh

Prevents other clusters in the mesh from accessing service 'mysvc' on cluster 'mycluster'

Allows cluster 'mycluster' to access service 'mysvc' on all clusters in the mesh

Allows other clusters in the mesh to access service 'mysvc' on cluster 'mycluster'

Multiple answers can be checked as correct.

?

Unshared Service Quiz

Unsharing service `mysvc` on cluster `mycluster`
(select all answers that apply)

Global services & latency



Global services & latency

Global services allow to load-balance traffic across multiple clusters.

As we have seen, this is very useful to implement a fallback policy for redundant services.

Most of the time however, it would be useful to limit latency by only using remote services when local ones are not available.

This is the objective of service affinity.



① Note:

The opposite effect can be obtained by using `remote` as the annotation value.

...

```
kubectl --context kind-koornacht scale deployment rebel-base --replicas 2  
kubectl --context kind-koornacht rollout status deployment/rebel-base  
kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10);  
do curl rebel-base; done'  
...  
...
```

A service with a local affinity always sends traffic locally

A service with a local affinity never sends traffic locally

A service with a remote affinity only sends traffic locally if there are no remote pods

A service with a local affinity only sends traffic locally if there are no remote pods

Multiple answers can be checked as correct.

Secure cross cluster communication



Securing Cross-Cluster Communication

In the previous examples, we have used a pod (`x-wing`) as a curl client to access another set of pods (`rebel-base`) either in a local or a remote cluster.

Cilium Network Policies are Kubernetes resources which allow to restrict access between pods, by labels.



Network Policies Lab

ⓘ Note:

You can learn more about Cilium Network Policies by taking the Isovalent Network Policy Lifecycle lab.



Cross-Cluster Network Policies

When using Cilium Cluster Mesh, it is possible to add Cilium Network Policies to filter traffic between clusters.

At the moment, the Koornacht service load-balances to both Koornacht and Tion clusters, with a local affinity.

In the context of a Zero-Trust security policy, we would like to block all traffic, and then allow only what is necessary.

...

--

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "default-deny"
spec:
  description: "Default Deny"
  endpointSelector: {}
  ingress:
    - {}
  egress:
    - toEndpoints:
        - matchLabels:
            io.kubernetes.pod.namespace: kube-system
            k8s-app: kube-dns
  toPorts:
```

- ports:
 - port: "53"
 - protocol: UDP
- rules:
 - dns:
 - matchPattern: "*"

...

```

❯ vi default-deny.yaml
❸kind-tion in ~ took 4s
❯ kubectl --context kind-koornacht apply -f default-deny.yaml
kubectl --context kind-tion apply -f default-deny.yaml
ciliumnetworkpolicy.cilium.io/default-deny created
ciliumnetworkpolicy.cilium.io/default-deny created

❹kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl --max-time 2 rebel-base; done'
curl: (28) Connection timed out after 2001 milliseconds
curl: (28) Connection timed out after 2001 milliseconds
curl: (28) Connection timed out after 2000 milliseconds
curl: (28) Connection timed out after 2001 milliseconds
curl: (28) Connection timed out after 2001 milliseconds
curl: (28) Connection timed out after 2001 milliseconds
curl: (28) Connection timed out after 2000 milliseconds
command terminated with exit code 28

❺kind-tion in ~ took 20s
❯

curl: (28) Connection timed out after 2000 milliseconds
command terminated with exit code 28

❻kind-tion in ~ took 28s
❯ hubctl observe --verdict DROPPED
Dec 26 13:03:24.881 [hubble-relay-7c6c883688-nmvxs]: 3 nodes are unavailable: tlon/tion-control-plane, tlon/tion-worker (and 1 more)
Dec 26 12:56:59.587: fe80::30ff:ffff%fe80:3d4e (ID:75699) <-> ff02::2 (unknown) Unsupported L3 protocol DROPPED (ICMPv6 RouterSolicitation)
Dec 26 12:58:13.255: fe80::30bb:bfff%fe80:3d4e (ID:53077) <-> ff02::2 (unknown) Unsupported L3 protocol DROPPED (ICMPv6 RouterSolicitation)
Dec 26 12:58:13.255: fe80::30bb:bfff%fe80:3d4e (ID:53077) <-> ff02::2 (unknown) Unsupported L3 protocol DROPPED (ICMPv6 RouterSolicitation)
Dec 26 13:00:16.115: fe80::30bb:bfff%fe80:3d4e (ID:53077) <-> ff02::2 (unknown) Unsupported L3 protocol DROPPED (ICMPv6 RouterSolicitation)
Dec 26 13:02:53.328: default/x-wing->9fc0cd0b-6228-4678 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:53.328: default/x-wing->9fc0cd0b-6228-4678 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:53.328: default/x-wing->9fc0cd0b-6228-4678 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:54.331: default/x-wing->9fc0cd0b-6228-4678 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:55.339: default/x-wing->9fc0cd0b-6228-4484 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:55.339: default/x-wing->9fc0cd0b-6228-4484 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:56.347: default/x-wing->9fc0cd0b-6228-4484 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:56.347: default/x-wing->9fc0cd0b-6228-4484 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:57.356: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:57.356: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:58.367: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:58.367: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:02:59.368: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:02:59.368: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:03:00.378: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:03:00.378: default/x-wing->9fc0cd0b-6228-4485 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:03:01.387: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:03:01.387: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:03:02.395: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:03:02.395: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)
Dec 26 13:03:02.395: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) policy-verdict:none DROPPED (TCP Flags: SYN)
Dec 26 13:03:02.395: default/x-wing->9fc0cd0b-6228-4486 (ID:58602) <-> default/rebel-base-7ddfb0c5d4-7c86 (ID:53077) Policy denied DROPPED (TCP Flags: SYN)

❻kind-tion in ~

```

...

```

apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "x-wing-to-rebel-base"
spec:
  description: "Allow x-wing in Koornacht to contact rebel-base"
  endpointSelector:
    matchLabels:
      name: x-wing
      io.cilium.k8s.policy.cluster: koornacht
  egress:
    - toEndpoints:
        - matchLabels:
            name: rebel-base

```

ur default deny policy blocks both ingress and egress connections for all pods, but the new policy we've added only allows egress connectivity. We also need to allow ingress

connections to reach the rebel-base pods. Let's fix this with a new CiliumNetworkPolicy resource:

```
...
---
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "rebel-base-from-x-wing"
spec:
  description: "Allow rebel-base to be contacted by Koornacht's x-wing"
  endpointSelector:
    matchLabels:
      name: rebel-base
  ingress:
    - fromEndpoints:
        - matchLabels:
            name: x-wing
            io.cilium.k8s.policy.cluster: koornacht
...

```

```
⌚ kind-tion in ~ took 1m33s
❯ kubectl --context kind-koornacht apply -f rebel-base-from-x-wing.yaml
ciliumnetworkpolicy.cilium.io/rebel-base-from-x-wing created

⌚ kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl --max-time 2 rebel-base; done'
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
curl: (28) Connection timed out after 2000 milliseconds
command terminated with exit code 28
```

This is because we haven't applied any specific policies to the Tion cluster, where the default deny policy was also deployed.

We need to apply the rebel-base-from-x-wing Network Policy to the Tion cluster to allow the ingress connection:

```
⌚ {"Cluster": "Koornacht", "Planet": "N'Zoth"}
curl: (28) Connection timed out after 2000 milliseconds
command terminated with exit code 28

⌚ kind-tion in ~ took 12s
❯ kubectl --context kind-tion apply -f rebel-base-from-x-wing.yaml
ciliumnetworkpolicy.cilium.io/rebel-base-from-x-wing created

⌚ kind-tion in ~
❯ kubectl --context kind-koornacht exec -ti deployments/x-wing -- /bin/sh -c 'for i in $(seq 1 10); do curl --max-time 2 rebel-base; done'
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
{"Cluster": "Tion", "Planet": "Foran Tutha"}
{"Cluster": "Koornacht", "Planet": "N'Zoth"}
{"Cluster": "Tion", "Planet": "Foran Tutha"}
{"Cluster": "Tion", "Planet": "Foran Tutha"}
{"Cluster": "Tion", "Planet": "Foran Tutha"}
{"Cluster": "Koornacht", "Planet": "N'Zoth"}  
⌚ kind-tion in ~
```

- Cilium allows to filter network requests across clusters

- The io.cilium.k8s.policy.cluster can be used in an endpointSelector

- The io.cilium.k8s.policy.cluster can be used in an ingress rule

- The io.cilium.k8s.policy.cluster can be used in an egress rule

Multiple answers can be checked as correct.

Exam

When you want your service to be globally accessible and simultaneously not to be shared and also tries the local service first then only checks all other service in different clusters

```
kubectl annotate service <> io.cilium/service-affinity="local"  
kubectl annotate service <> io.cilium/global-service="true"  
kubectl annotate service <> io.cilium/shared-service="false"
```