Network Policy

Network Policy defines how the pods in the same namespace will communicate with each other and the network endpoint.

The Network Policy Spec

A network policy specification consists of four elements:

1. podSelector: the pods that will be subject to this policy (the policy target) - mandatory
2. policyTypes: specifies which types of policies are included in this policy, ingress and/or egress - this is optional but I recommend to always specify it explicitly.
3. ingress: allowed **inbound** traffic to the target pods - optional
4. egress: allowed **outbound** traffic from the target pods - optional

Note: The Default Policy is Allow

>Communications between namespaces are still allowed by default.

Case 1:

In this case, a good starting point is to allow all pods in the same namespace to talk to each other and explicitly whitelist communication across namespaces, since that is usually more rare.

You can use the following network policy to allow all pod-to-pod communication within a namespace:

apiVersion: networking.k8s.io/v1 **kind: NetworkPolicy** **metadata:** name: allow-same-namespace **spec:** podSelector: {} policyTypes: - Ingress ingress: - from: - podSelector: {}

If You Know The Sources and Sinks for Communication

For example, if your hub is a database pod and has an app=db label, you could allow access to the database only from pods that have a networking/allow-db-access=true label by applying the following policy:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-db-access
spec:
  podSelector:
    matchLabels:
      app: "db"
  policyTypes:
  - Ingress
  ingress:
```

```
 - from:
   - podSelector:
       matchLabels:
         networking/allow-db-access: "true"
```

Why Use a Kubernetes Secret?

A Kubernetes Secret is mainly designed to carry sensitive information that the web service needs to run. This includes information such as username and password, tokens for connecting with other pods, and certificate keys. Putting sensitive information in a Secret object allows for better security and tighter control over those details.

Secrets are also easy to integrate with existing services. You just have to tell the pods to use the custom Secrets you have created alongside the native Secrets created by Kubernetes. This means you can use Secrets to make deploying a web service across multiple clusters easier.

A standard Secret file looks something like this:

```
apiVersion: v1
kind: Secret
metadata:
name: test-secret
data:
username: bXktYXBw
password: Mzk1MjgkdmRnN0pi
```