What are Kubernetes Network Policies?

Kubernetes Network Policies are like **firewall rules** for your **Pods**.
They control **who can talk to whom** in the cluster at the **network level (Layer 3/4)**.

Why use Network Policies?

By default, **all pods can talk to all other pods** in a Kubernetes cluster.
This is fine for small apps, but risky in production.

Network Policies help you:

- **Isolate sensitive pods**
- **Block unnecessary traffic**
- **Enforce security and compliance**

Components of a Network Policy

A Network Policy defines:

Component	What it does
`podSelector`	Selects **which pods** the policy applies to
`ingress`	Controls **who can send traffic into** the selected pods
`egress`	Controls **where the selected pods can send traffic to**
`namespaceSelector`	Allows traffic from **specific namespaces**
`ports`	You can allow/block traffic to specific **ports**

✅ Step-by-step Setup

1. Start Kind cluster

Make sure your kind cluster is created with a CNI that supports `NetworkPolicy`, such as **Calico** (Kind's default CNI doesn't support network policies).

You can create a kind cluster with Calico using:

```
bash
----------
kind create cluster --config=kind-config.yaml
```

Then apply Calico:

```
bash

kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.
yaml
```

---

2. Create 3 Pods: Frontend, Backend, and Random

```
yaml
----------
# pods.yaml
apiVersion: v1
kind: Pod
metadata:
  name: backend
  labels:
    app: backend
spec:
  containers:
  - name: backend
    image: busybox
    command: ["sh", "-c", "while true; do nc -lk -p 8080 -e echo hello; done"]
    ports:
    - containerPort: 8080

---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  containers:
  - name: frontend
    image: busybox
    command: ["sleep", "3600"]

---
apiVersion: v1
kind: Pod
metadata:
  name: random
```

```
  labels:
    app: random
spec:
  containers:
  - name: random
    image: busybox
    command: ["sleep", "3600"]
```

Apply this:

```
bash
----------
kubectl apply -f pods.yaml
```

---

3. Apply NetworkPolicy

This is your `allow-frontend-to-backend.yaml`:

```
yaml
----------
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: frontend
```

Apply it:

```
bash
----------
kubectl apply -f allow-frontend-to-backend.yaml
```

---

4. Test connectivity

✅ From `frontend` (should succeed):

```
bash
----------
kubectl exec frontend -- sh -c "wget -qO- http://backend:8080"
```

Should print: `hello`

❌ From `random` (should hang/fail):

```
bash
----------
kubectl exec random -- sh -c "wget -qO- http://backend:8080"
```

Should hang or return an error, depending on the network plugin.

🎯 Summary

- Frontend ✅ can access Backend
- Random ❌ **cannot** access Backend
- NetworkPolicy controls **Ingress** to Backend, allowing only pods labeled `app: frontend`

By default, Kubernetes allows:

A. Only frontend pods to talk to backend pods
B. All pods to talk to all other pods
C. No pod communication unless a policy is set
D. Communication only within the same namespace

✅ **Correct Answer:** B
📝 *Explanation: In a fresh Kubernetes cluster with no Network Policies, all pods can talk to each other.*

Why are Network Policies important in production?

A. They reduce RAM usage
B. They help isolate sensitive pods and block unnecessary traffic
C. They improve pod restart speed
D. They control YAML file access

✅ **Correct Answer:** B
📝 *Explanation: Network Policies are used to enforce security and limit unnecessary communication.*

Which component controls incoming traffic in a Network Policy?

A. egress
B. nodeSelector
C. ingress
D. Entrypoint

✅ **Correct Answer:** C

📝 *Explanation:* `ingress` *defines who can send traffic into selected pods.*