**Kubernetes Network Policies**

**What are Network Policies?**

Kubernetes **NetworkPolicy** is a resource that controls **pod-level network access** — **who can talk to whom** over the network.

Think of it as a **firewall for pods**.

It applies only if your **CNI plugin supports it** (e.g., Calico, Cilium, WeaveNet).

**🧱 Structure of a NetworkPolicy**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: <policy-name>

namespace: <namespace>

spec:

podSelector: {} # Which pods this applies to

ingress: # Rules for incoming traffic

egress: # Rules for outgoing traffic

policyTypes: # [Ingress, Egress]

- Ingress

- Egress

**Scenario 1: Deny All Traffic by Default**

**What It Does:**

This policy **completely blocks all traffic** — both **incoming (ingress)** and **outgoing (egress)** — for all pods in the namespace.

**Why Use It:**

* To implement **zero trust** by default
* Only allow what’s necessary by explicitly writing policies
* Prevent unexpected pod communications (common in multi-tenant clusters)

**📜 YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: default-deny-all

namespace: webapps

spec:

podSelector: {} # Selects ALL pods in this namespace

policyTypes:

- Ingress # Block all incoming traffic

- Egress # Block all outgoing traffic

**How It Works:**

* The empty podSelector: {} means **every pod** is affected.
* policyTypes: [Ingress, Egress] says: “I want to manage both inbound and outbound.”
* Since there are no rules under ingress: or egress:, it means **everything is denied**.

**Outcome:**

* Pods **cannot reach each other**
* Pods **cannot talk to the internet**
* No one can talk **to** them either

**Scenario 2: Allow All Pods in the Same Namespace to Talk**

**What It Does:**

Allows **all pods in a namespace** to freely communicate **with each other**, but **not** with pods in other namespaces.

**Why Use It:**

* To enable internal microservice communication
* Still enforces namespace isolation (good for dev/staging/test)

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-same-namespace

namespace: webapps

spec:

podSelector: {} # Applies to all pods in this namespace

ingress:

- from:

- podSelector: {} # All pods in same ns can connect

policyTypes:

- Ingress

**How It Works:**

* The target is all pods (podSelector: {})
* We allow ingress **only** from **other pods in the same namespace**
* No access is allowed from **outside this namespace**

**Outcome:**

* Pod A ↔ Pod B in webapps: ✅
* Pod in monitoring → Pod in webapps: ❌

**Scenario 3: Allow Frontend Pods to Talk to Backend Pods**

**Use Case:**

Allow only **frontend** pods to send requests to **backend** pods — a typical microservice pattern.

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-frontend-to-backend

namespace: webapps

spec:

podSelector:

matchLabels:

app: backend # Target pods with this label

ingress:

- from:

- podSelector:

matchLabels:

app: frontend

policyTypes:

- Ingress

**How It Works:**

* Select pods labeled app: backend
* Allow only pods with app: frontend to access them
* Backend cannot initiate connections, and others (like logging or jobs) cannot access backend

**Outcome:**

* frontend → backend: ✅
* monitoring → backend: ❌
* backend → frontend: ❌

**Scenario 4: Allow Traffic from Another Namespace**

**What It Does:**

Allows pods in another namespace (like monitoring) to access your pods. Useful for **Prometheus scraping metrics** from app pods.

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-from-monitoring

namespace: webapps

spec:

podSelector: {}

ingress:

- from:

- namespaceSelector:

matchLabels:

team: monitoring

policyTypes:

- Ingress

**How It Works:**

* podSelector: {} = all pods in webapps
* Allow traffic from **any pod** in **namespaces** labeled team=monitoring

**Pre-requisite:**

kubectl label namespace monitoring team=monitoring

**Outcome:**

* monitoring/prometheus → webapps/backend: ✅
* Any other namespace → webapps: ❌

**Scenario 5: Allow Port-Specific Access (e.g., MySQL 3306)**

**Use Case:**

Only allow role=app pods to talk to mysql pods on port **3306**, blocking all other ports.

**📜 YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-mysql-access

namespace: webapps

spec:

podSelector:

matchLabels:

app: mysql

ingress:

- from:

- podSelector:

matchLabels:

role: app

ports:

- protocol: TCP

port: 3306

policyTypes:

- Ingress

**How It Works:**

* Only pods labeled role=app are allowed
* Access is **only granted on port 3306**
* No SSH, metrics ports, or others will work

**Scenario 6: Allow Pods to Access Internet**

**What It Does:**

Allows pods to **make external calls** (e.g., API requests) to any IP, but doesn’t allow inbound traffic.

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-egress-global

namespace: webapps

spec:

podSelector: {}

egress:

- to:

- ipBlock:

cidr: 0.0.0.0/0

policyTypes:

- Egress

**How It Works:**

* ipBlock: 0.0.0.0/0 = all IPs (the entire internet)
* Allows all outbound traffic
* Ingress remains blocked unless explicitly allowed

**Scenario 7: Allow Access to One External IP Only**

**Use Case:**

Let pods reach out to **one IP** like 10.10.20.5:443 — useful for calling an internal API or external DB securely.

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-external-api

namespace: webapps

spec:

podSelector:

matchLabels:

app: api-client

egress:

- to:

- ipBlock:

cidr: 10.10.20.5/32

ports:

- protocol: TCP

port: 443

policyTypes:

- Egress

**How It Works:**

* Only TCP:443 to one IP is allowed
* Everything else is blocked

**Scenario 8: Isolate a Sensitive Pod (e.g., Vault)**

**What It Does:**

Make sure vault pod:

* Can only **receive** traffic from approved sources
* Cannot make any outbound traffic (zero egress)

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: isolate-vault

namespace: security

spec:

podSelector:

matchLabels:

app: vault

ingress:

- from:

- podSelector:

matchLabels:

app: backend

egress: [] # block all outbound

policyTypes:

- Ingress

- Egress

**Outcome:**

* Only backend can reach Vault
* Vault can't reach internet or any other service

**Scenario 9: Allow DNS Egress (Required for Internet Access)**

**Why?**

If you block egress and **don’t allow DNS**, your pods **can’t resolve github.com, google.com, etc.**

**YAML:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-dns

namespace: webapps

spec:

podSelector: {}

egress:

- to:

- namespaceSelector: {}

podSelector:

matchLabels:

k8s-app: kube-dns

ports:

- port: 53

protocol: UDP

policyTypes:

- Egress

Confirm the label on CoreDNS:

kubectl get pods -n kube-system --show-labels

**🧠 Summary**

| **Scenario** | **Type** | **Direction** | **Description** |
| --- | --- | --- | --- |
| 1 | Default Deny | Ingress + Egress | Deny all traffic |
| 2 | Internal Access | Ingress | Allow pod ↔ pod in same ns |
| 3 | Microservice Access | Ingress | frontend → backend only |
| 4 | Cross-Namespace | Ingress | Allow from monitoring ns |
| 5 | Port-Level Control | Ingress | Allow access on 3306 only |
| 6 | Internet Access | Egress | Allow all outbound |
| 7 | IP-Specific | Egress | Allow 1 external IP only |
| 8 | Isolate Sensitive Pod | Ingress + Egress | Vault = no outbound |
| 9 | Allow DNS | Egress | Allow CoreDNS only |

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-all

namespace: your-namespace

spec:

podSelector: {} # Apply to all pods in this namespace

ingress:

- {} # Allow all sources

egress:

- {} # Allow all destinations

policyTypes:

- Ingress

- Egress