Kubernetes Networking : Network Policies

Part-2

Kubernetes **Network Policies** allow you to control the flow of traffic between pods in a cluster. These policies are an important part of security and resource management, as they let you define which pods can communicate with each other and under what circumstances. A **NetworkPolicy** object is used to define rules about what kind of traffic is allowed to and from a pod, helping isolate applications and enforce security policies within your Kubernetes environment.

In this document, we'll go through several examples of **NetworkPolicy** configurations, explaining how each one controls ingress and egress traffic for pods.

Network Policy Basics

Each **NetworkPolicy** is defined using the following key components:

- **Pod Selector**: Defines which pods the network policy applies to, typically by matching pod labels.
- **Policy Types**: Specifies whether the policy applies to **Ingress** (incoming traffic), **Egress** (outgoing traffic), or both.
- **Rules**: Defines the conditions under which traffic is allowed. This can be based on factors like pod labels, IP blocks, or specific ports.

Now, let’s explore various examples of **NetworkPolicy** configurations.

---

1. Network Policy: Allow Ingress from Specific Pods

This **NetworkPolicy** allows incoming traffic to the **frontend** pods only from **backend** pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-ingress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: frontend    # Applies this policy to pods with the label "app=frontend"
  policyTypes:
  - Ingress          # This policy only applies to incoming traffic
  ingress:
```

```
 - from:
   - podSelector:
       matchLabels:
         app: backend  # Allows traffic only from pods with label "app=backend"
```

Explanation:

- **Pod Selector**: This policy applies to pods labeled app=frontend.
- **Policy Type**: It only controls **Ingress** traffic.
- **Ingress Rule**: Only allows traffic from pods labeled app=backend in the same namespace.

Use Case:

In a microservices architecture, you might have a **frontend** pod that should only receive traffic from the **backend** pod (such as a REST API). This ensures that the frontend is not exposed to any other pods in the cluster, improving security.

---

2. Network Policy: Deny All Ingress

This **NetworkPolicy** blocks all incoming traffic to the **database** pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: database    # Applies to pods labeled "app=database"
  policyTypes:
  - Ingress          # Only controls incoming traffic
  ingress: []
```

Explanation:

- **Pod Selector**: This policy applies to pods labeled app=database.
- **Policy Type**: It only controls **Ingress** traffic.
- **Ingress Rule**: The empty ingress rule (ingress: []) means that no incoming traffic is allowed to the app=database pods.

Use Case:

This type of policy can be used to completely isolate sensitive database pods, preventing any incoming traffic except for what is explicitly allowed by other policies.

3. Network Policy: Allow Egress to a Specific Pod

This **NetworkPolicy** allows the **frontend** pods to send outgoing traffic only to the **database** pods on port **3306** (used by MySQL).

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress-database
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: frontend    # Applies to pods labeled "app=frontend"
  policyTypes:
  - Egress           # This policy only controls outgoing traffic
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database  # Allows outgoing traffic to pods labeled "app=database"
    ports:
    - protocol: TCP
      port: 3306       # Only allow outgoing traffic to port 3306 (MySQL)
```

Explanation:

- **Pod Selector**: This policy applies to pods labeled app=frontend.
- **Policy Type**: It only controls **Egress** (outgoing) traffic.
- **Egress Rule**: Allows outgoing traffic to pods labeled app=database, but only on TCP port **3306** (the default port for MySQL).

Use Case:

This policy is useful when you want to restrict the **frontend** pods so that they can only communicate with the **database** service on a specific port (in this case, the MySQL port). It limits the communication paths between services, enforcing stronger security boundaries.

4. Network Policy: Allow Egress to an External IP

This **NetworkPolicy** allows outgoing traffic from **backend** pods only to the external IP **8.8.8.8** (Google DNS).

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress-external
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend      # Applies to pods labeled "app=backend"
  policyTypes:
  - Egress            # Controls outgoing traffic
  egress:
  - to:
    - ipBlock:
        cidr: 8.8.8.8/32  # Allows outgoing traffic only to the external IP 8.8.8.8
```

Explanation:

- **Pod Selector**: This policy applies to pods labeled app=backend.
- **Policy Type**: It only controls **Egress** traffic.
- **Egress Rule**: Allows outgoing traffic to the external IP **8.8.8.8** (Google's DNS server).

Use Case:

This policy is useful when you want to restrict **backend** pods to only communicate with a specific external service (e.g., a DNS server). In this case, the pod can only send requests to **8.8.8.8**, which improves security by limiting access to external networks.

---

5. Network Policy: Allow Both Ingress and Egress

This **NetworkPolicy** allows both incoming traffic to **backend** pods from **api** pods and outgoing traffic from **backend** pods to **db** pods on port **5432** (PostgreSQL).

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress           # Controls incoming traffic
  - Egress            # Controls outgoing traffic
  ingress:
  - from:
```

```
  - podSelector:
      matchLabels:
        app: api      # Allows incoming traffic from pods labeled "app=api"
egress:
- to:
  - podSelector:
      matchLabels:
        app: db       # Allows outgoing traffic to pods labeled "app=db"
  ports:
  - protocol: TCP
    port: 5432        # Only allow outgoing traffic to port 5432 (PostgreSQL)
```

Explanation:

- **Pod Selector**: This policy applies to pods labeled app=backend.
- **Policy Type**: It controls both **Ingress** (incoming) and **Egress** (outgoing) traffic.
- **Ingress Rule**: Allows traffic from pods labeled app=api.
- **Egress Rule**: Allows traffic to pods labeled app=db, but only on TCP port **5432** (the default port for PostgreSQL).

Use Case:

This policy is useful in a microservices architecture where the **backend** service should accept requests from the **API** service but should only be allowed to connect to the **database** service on a specific port (e.g., PostgreSQL on port 5432).

Network Policy Deep Dive: Understanding Policy Behavior

1. Policy Types: Ingress and Egress

- **Ingress**: Controls incoming traffic to a pod. If a policy exists that controls ingress, no traffic will be allowed unless explicitly permitted by the policy.
- **Egress**: Controls outgoing traffic from a pod. Similarly, if an egress policy is defined, no traffic will be allowed to leave the pod unless explicitly permitted.

2. Pod Selectors

A **podSelector** defines which pods the network policy applies to. For example, using a label selector like matchLabels: { app: frontend } ensures that the policy only affects the pods with the label app=frontend.

3. Policy Enforcement

- If no network policies are applied to a pod, by default, all traffic is allowed both in and out of the pod.
- Once a network policy is applied to a pod, the default becomes to **deny all traffic**, unless explicitly allowed by the policy. For example, if a policy restricts ingress but doesn't define any rules for egress, all incoming traffic will be blocked by default.

Conclusion: Importance of Network Policies in Kubernetes

Network policies play a crucial role in securing Kubernetes clusters by controlling how pods communicate with each other and the outside world. By using Kubernetes network policies, you can:

- Limit access to sensitive services like databases.
- Isolate application components to reduce the attack surface.
- Ensure that microservices can only communicate with authorized services.

In a microservices environment where applications are split across multiple pods and services, network policies are essential for maintaining security, reducing the risk of lateral movement in case of a breach, and enforcing application-level access control.