Kubernetes Network Policy and Istio Service Mesh - Student Guide

Overview

This lesson demonstrates two important Kubernetes networking concepts:

1. **Network Policies** - How to control network traffic between pods
2. **Istio Service Mesh** - Advanced traffic management with sidecar pattern

Learning Objectives

By the end of this lesson, you will be able to:

- Configure Kubernetes Network Policies to control pod-to-pod communication
- Set up Istio service mesh in a Kubernetes cluster
- Understand the sidecar pattern and its benefits
- Deploy and access applications through Istio ingress gateway

---

Part 1: Network Policy Demo

Objective

Block frontend pods from connecting to MySQL database while allowing backend pods to maintain connectivity.

Prerequisites

- Minikube installed
- kubectl configured
- Basic understanding of Kubernetes pods and services

Step 1: Set up Minikube with Calico CNI

First, clean up any existing Minikube cluster:

```bash
minikube delete
```

Start Minikube with Calico CNI (required for Network Policy support):

```bash
minikube start --network-plugin=cni --cni=calico
```

Why Calico? Network Policies require a CNI plugin that supports them. Calico is one of the most popular choices.

Verify Calico is running:

```bash
kubectl get pods -l k8s-app=calico-node -n kube-system
```

Step 2: Deploy the Application

Apply the demo application manifest:

```bash
kubectl apply -f network-policy-demo.yaml
```

This creates:

- **Frontend pod** - Simulates a web frontend
- **Backend pod** - Simulates an API backend
- **MySQL pod** - Database server
- **Services** - ClusterIP services for each component

Verify all pods are running:

```bash
kubectl get pods
```

Expected output:

```
NAME     READY   STATUS    RESTARTS   AGE
backend   1/1    Running   0          56s
frontend  1/1    Running   0          56s
mysql     1/1    Running   0          56s
```

Step 3: Test Initial Connectivity

Check that frontend can connect to database (this should work initially):

```bash
kubectl exec -it frontend -- bash
```

Inside the frontend pod:

```bash
apt update && apt install telnet -y
telnet db 3306
```

You should see:

```
Trying 10.105.255.201...
Connected to db.
```

Exit the telnet session and pod:

```bash
exit
exit
```

Step 4: Apply Network Policy

Now apply the network policy to block frontend access:

```bash
kubectl apply -f db-netpol.yaml
```

Step 5: Verify the Policy Works

Test that frontend is now blocked:

```bash
kubectl exec -it frontend -- bash
telnet db 3306
```

This should hang (connection blocked). Use Ctrl+C to exit.

Test that backend still works:

```bash
kubectl exec -it backend -- bash
apt update && apt install telnet -y
telnet db 3306
```

This should still connect successfully.

Understanding the Network Policy

The `db-netpol.yaml` file contains rules that:

- Apply to pods with specific labels (the database)
- Allow ingress traffic only from pods with certain labels (backend)
- Block all other traffic by default

Part 2: Istio Service Mesh Demo

Objective

Deploy Istio service mesh and demonstrate the sidecar pattern with the Bookinfo application.

Step 1: Prepare Minikube for Istio

Start Minikube with sufficient memory:

```bash
minikube start --memory=4096
```

Why more memory? Istio components require additional resources to run properly.

Step 2: Download and Install Istio

Download Istio:

```bash
curl -L https://istio.io/downloadIstio | sh -
```

Navigate to the Istio directory:

```bash
cd istio-1.26.2/bin
```

Install Istio with demo profile:

```bash
./istioctl install --set profile=demo -y
```

The demo profile includes:

- Istio core components
- Ingress and egress gateways
- Additional features for learning

Step 3: Configure Automatic Sidecar Injection

Enable automatic sidecar injection for the default namespace:

```bash
kubectl label namespace default istio-injection=enabled
```

What is sidecar injection? Istio automatically injects a proxy container (Envoy) alongside each application container to handle network traffic.

Step 4: Deploy the Bookinfo Application

Deploy the sample Bookinfo application:

```bash
kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

This creates a microservices application with:

- **Product page** - Main application frontend
- **Details** - Book details service
- **Reviews** - Book reviews service (3 versions)
- **Ratings** - Star ratings service

Step 5: Verify Sidecar Injection

Check that pods have 2/2 containers (app + sidecar):

```bash
kubectl get pods
```

Expected output:

```
NAME                       READY   STATUS    RESTARTS   AGE
details-v1-766844796b-jkbrg      2/2     Running   0          77s
productpage-v1-54bb874995-q4c6g  2/2     Running   0          77s
ratings-v1-5dc79b6bcd-55fkn      2/2     Running   0          77s
reviews-v1-598b896c9d-85n8t      2/2     Running   0          77s
reviews-v2-556d6457d-l7ktl       2/2     Running   0          77s
reviews-v3-564544b4d6-9lr8t      2/2     Running   0          77s
```

Notice: Each pod shows `2/2` ready, indicating the application container plus the Istio sidecar.

Step 6: Configure Ingress Gateway

Apply the gateway configuration:

```bash
kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

This creates:

- **Gateway** - Configures the ingress gateway
- **VirtualService** - Routes traffic to the product page

Step 7: Access the Application

Get the ingress gateway URL:

```bash
minikube service istio-ingressgateway -n istio-system --url
```

This returns multiple URLs. Use the second one (typically on port 80) to access:

```
http://192.168.49.2:31171/productpage
```

Open this URL in your browser to see the Bookinfo application.

Key Concepts Explained

Network Policies

- **Purpose**: Control traffic flow between pods at the network level
- **Default behavior**: Without policies, all pods can communicate
- **Label-based**: Use selectors to define which pods the policy applies to
- **Ingress/Egress**: Control incoming and outgoing traffic separately

Istio Service Mesh

- **Sidecar Pattern**: Each pod gets a proxy container that handles network traffic
- **Traffic Management**: Control routing, load balancing, and failover
- **Security**: Mutual TLS, authentication, and authorization
- **Observability**: Metrics, logging, and tracing out of the box

Benefits of Service Mesh

- **Decoupled**: Network logic separated from application code
- **Consistent**: Same features across all services
- **Secure**: Automatic encryption and authentication
- **Observable**: Built-in monitoring and tracing

Troubleshooting Tips

Network Policy Issues

- Ensure you're using a CNI that supports Network Policies (like Calico)
- Check pod labels match the policy selectors
- Remember: policies are namespace-scoped

Istio Issues

- Verify sufficient memory allocation to Minikube
- Check that sidecar injection is enabled for the namespace

- Ensure all pods show 2/2 ready status
- Use `istioctl analyze` to diagnose configuration issues

Common Commands

```bash
# Check network policies
kubectl get networkpolicies

# Describe a network policy
kubectl describe networkpolicy <policy-name>

# Check Istio configuration
istioctl analyze

# View Istio proxy configuration
istioctl proxy-config cluster <pod-name>
```

Exercise Questions

1. What happens if you remove the network policy? Test it.
2. How would you modify the policy to allow frontend access on a different port?
3. What's the difference between pods with and without Istio sidecars?
4. How can you verify that traffic is being encrypted by Istio?

Additional Resources

- Kubernetes Network Policy Documentation
- Istio Documentation
- Calico Network Policy Guide
- Envoy Proxy Documentation