Kubernetes Security Hands-On Labs — Network Policies and Pod Security Standards (PSS)

Author : Sandeep Kumar Sharma

Introduction:

In this lab series, we'll focus on **network-level** and **pod-level** security in Kubernetes. Security in Kubernetes isn't just about users and RBAC; it's also about how Pods communicate and how they're restricted from performing dangerous operations.

We'll discuss common security problems and how to fix them using **Network Policies** and **Pod Security Standards (PSS)**.

Lab 1 — Network Policies

Problem Scenario:

By default, **all Pods in a namespace can communicate with each other**. This can be a serious security risk if one Pod gets compromised — it could send unauthorized traffic to other Pods.

Objective:

You'll learn how to restrict Pod-to-Pod communication using **NetworkPolicies**.

Step 1 — Create a Namespace for Testing

```
kubectl create namespace secure-apps
```

Step 2 — Deploy Two Simple Pods

We'll deploy two busybox Pods in the same namespace.

File: pods.yaml

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: pod-a
  namespace: secure-apps
  labels:
    app: pod-a
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-b
  namespace: secure-apps
  labels:
    app: pod-b
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
```

Apply it:

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

Step 3 — Verify Communication (Before Policy)

Try to ping one Pod from another:

```
kubectl exec -it pod-a -n secure-apps -- ping pod-b
```

You'll see that **Pod A can reach Pod B**. This is the default Kubernetes behavior — all Pods can talk to each other within a namespace.

Step 4 — Create a Network Policy to Restrict Communication

Now, we'll allow only **pod-a** to receive traffic and block all others.

File: deny-all.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: secure-apps
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Apply it:

```
kubectl apply -f deny-all.yaml
```

Now, this policy **denies all inbound and outbound communication** between Pods in the namespace.

Try ping again:

```
kubectl exec -it pod-a -n secure-apps -- ping pod-b
```

You'll notice it **fails** — communication is blocked. ✅

Step 5 — Allow Specific Communication (Solution)

Let's now allow **Pod A** to talk to **Pod B** using a new policy.

File: allow-specific.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-pod-a-to-b
  namespace: secure-apps
spec:
  podSelector:
    matchLabels:
      app: pod-b
  ingress:
  - from:
    - podSelector:
```

```
      matchLabels:
        app: pod-a
```

Apply it:

```
kubectl apply -f allow-specific.yaml
```

Now, only Pod A can reach Pod B, while all other communications remain blocked.

Test it again:

```
kubectl exec -it pod-a -n secure-apps -- ping pod-b
```

✅It works!

If you try the reverse direction (Pod B → Pod A):

```
kubectl exec -it pod-b -n secure-apps -- ping pod-a
```

❌It fails, as expected.

Problem Solved:

You've successfully **controlled inter-pod communication**, allowing only the required paths. This limits the blast radius in case of compromise and enforces **least privilege networking**.

Lab 2 — Pod Security Standards (PSS)

Problem Scenario:

By default, any user can deploy Pods that run as **root** or access host-level resources. This is risky because if such a Pod is compromised, it could take control of the node.

We'll use **Pod Security Standards (PSS)** to enforce security at the namespace level.

Pod Security Levels:

1. **Privileged** — No restrictions; full access (used for system pods).
2. **Baseline** — Some restrictions; disallows known risky behaviors.
3. **Restricted** — Most secure; applies strict limitations.

Objective:

Apply security controls to prevent Pods from running as root or mounting sensitive volumes.

Step 1 — Create a Namespace

```
kubectl create namespace secure-pss
```

Step 2 — Apply Pod Security Label

```
kubectl label namespace secure-pss pod-security.kubernetes.io/enforce=restricted
```

This enforces the **restricted** security profile in this namespace.

Step 3 — Try to Deploy an Insecure Pod

File: insecure-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: insecure-pod
  namespace: secure-pss
spec:
  containers:
  - name: root-container
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    securityContext:
      runAsUser: 0  # root user
```

Apply it:

```
kubectl apply -f insecure-pod.yaml
```

You'll get an error:

```
Error from server (Forbidden): pods "insecure-pod" is forbidden: violates
PodSecurity "restricted:latest"
```

✅Perfect! The restricted profile **blocked a root-level container**.

Step 4 — Deploy a Secure Pod (Solution)

File: secure-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
  namespace: secure-pss
spec:
  containers:
  - name: non-root-container
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    securityContext:
      runAsUser: 1000
      allowPrivilegeEscalation: false
```

Apply it:

```
kubectl apply -f secure-pod.yaml
```

It will successfully run because it complies with the **restricted** policy.

Problem Solved:

Before applying Pod Security Standards, any user could run privileged Pods and compromise the node. After applying **restricted** security mode, Kubernetes automatically blocks risky deployments.

Verification Summary

Lab	Problem	Solution	Key Concept
1	Pods can communicate freely (no isolation)	Apply NetworkPolicies	Restrict network access
2	Pods can run as root and access host	Enforce Pod Security Standards	Enforce least privilege execution

---

✅**Final Outcome:** After completing these labs, participants will understand: - How to **restrict Pod-to-Pod communication** using Network Policies. - How to **enforce security controls** at the namespace level using Pod Security Standards.

These skills are foundational for securing real-world Kubernetes clusters and ensuring that workloads follow the **principle of least privilege**.