Kubernetes Taints and Tolerations Lab

Overview

This lab will teach you how to control pod scheduling in Kubernetes using taints and tolerations. You'll learn to restrict which pods can be scheduled on specific nodes and create specialized node pools.

Prerequisites

- Kubernetes cluster (minikube, kind, or cloud cluster)
- kubectl configured and connected to your cluster
- Basic understanding of Kubernetes pods and nodes

Learning Objectives

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

- Understand the difference between taints and tolerations
- Apply taints to nodes to repel pods
- Configure tolerations in pods to overcome taints
- Use different taint effects (NoSchedule, PreferNoSchedule, NoExecute)
- Implement real-world scheduling scenarios

Part 1: Understanding Taints and Tolerations

Theory

- **Taints**: Applied to nodes to repel pods that don't tolerate the taint
- **Tolerations**: Applied to pods to allow them to be scheduled on tainted nodes
- **Taint Effects**:
  - `NoSchedule`: Pods won't be scheduled unless they tolerate the taint
  - `PreferNoSchedule`: Kubernetes tries to avoid scheduling pods but it's not guaranteed
  - `NoExecute`: Existing pods without tolerance will be evicted

Key-Value Structure

```
key=value:effect
```

Part 2: Lab Setup

Step 1: Check Your Cluster

```bash
# Verify cluster connection
kubectl get nodes

# Check current node taints
kubectl describe nodes | grep -i taint
```

Step 2: Create a Multi-Node Environment (if using minikube)

```bash
# If using minikube, start with multiple nodes
minikube start --nodes=3

# Or add nodes to existing cluster
minikube node add --name worker-2
```

Part 3: Basic Taint Operations

Step 3: Apply Taints to Nodes

```bash
# Get node names
kubectl get nodes

# Apply a taint to a node (replace NODE_NAME with actual node name)
kubectl taint nodes NODE_NAME environment=production:NoSchedule

# Apply multiple taints
kubectl taint nodes NODE_NAME dedicated=database:NoSchedule
kubectl taint nodes NODE_NAME disk=ssd:PreferNoSchedule

# Verify taints
kubectl describe node NODE_NAME | grep -i taint
```

Step 4: Test Default Pod Behavior

Create a simple pod without tolerations:

```yaml
# basic-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: basic-pod
  labels:
    app: test
spec:
  containers:
  - name: nginx
    image: nginx:alpine
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
```

```bash
# Apply the pod
kubectl apply -f basic-pod.yaml

# Check pod status - it should be pending if scheduled on tainted node
kubectl get pods -o wide

# Check events to see scheduling issues
kubectl describe pod basic-pod
```

Part 4: Working with Tolerations

Step 5: Pod with Tolerations

Create pods that can tolerate specific taints:

```yaml
```

```yaml
# tolerated-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: production-pod
  labels:
    app: production
spec:
  tolerations:
  - key: "environment"
    operator: "Equal"
    value: "production"
    effect: "NoSchedule"
  containers:
  - name: nginx
    image: nginx:alpine
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
```

bash

```bash
# Apply the tolerated pod
kubectl apply -f tolerated-pod.yaml

# Verify it's scheduled on the tainted node
kubectl get pods -o wide
```

Step 6: Different Toleration Operators

yaml

```yaml
# flexible-toleration-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: flexible-pod
spec:
  tolerations:
  # Exists operator - tolerates any value for this key
  - key: "environment"
    operator: "Exists"
    effect: "NoSchedule"
  # Equal operator with specific value
  - key: "dedicated"
    operator: "Equal"
    value: "database"
    effect: "NoSchedule"
  # Tolerate all taints on a node
  - operator: "Exists"
    effect: "PreferNoSchedule"
  containers:
  - name: busybox
    image: busybox
    command: ["sleep", "3600"]
```

Part 5: Advanced Taint Effects

Step 7: NoExecute Effect

bash

```bash
# Apply NoExecute taint (evicts existing pods)
kubectl taint nodes NODE_NAME maintenance=true:NoExecute

# Create a pod with tolerance for NoExecute
```

yaml

```
# noexecute-tolerant-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: maintenance-tolerant-pod
spec:
  tolerations:
  - key: "maintenance"
    operator: "Equal"
    value: "true"
    effect: "NoExecute"
    tolerationSeconds: 300  # Tolerate for 5 minutes
  containers:
  - name: nginx
    image: nginx:alpine
```

Step 8: Time-bound Tolerations

yaml

```
# timed-toleration-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: timed-pod
spec:
  tolerations:
  - key: "maintenance"
    operator: "Equal"
    value: "true"
    effect: "NoExecute"
    tolerationSeconds: 60  # Pod will be evicted after 60 seconds
  containers:
  - name: busybox
    image: busybox
    command: ["sleep", "3600"]
```

Part 6: Real-World Scenarios

Step 9: GPU Node Pool Simulation

bash

```
# Taint nodes for GPU workloads
kubectl taint nodes NODE_NAME hardware=gpu:NoSchedule
```

yaml

```yaml
# gpu-workload.yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-workload
spec:
  tolerations:
  - key: "hardware"
    operator: "Equal"
    value: "gpu"
    effect: "NoSchedule"
  nodeSelector:
    hardware: "gpu"  # Ensure scheduling only on GPU nodes
  containers:
  - name: tensorflow
    image: tensorflow/tensorflow:latest-gpu
    command: ["sleep", "3600"]
```

Step 10: Database Node Dedication

yaml

```yaml
# database-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      tolerations:
      - key: "dedicated"
        operator: "Equal"
        value: "database"
        effect: "NoSchedule"
      - key: "dedicated"
        operator: "Equal"
        value: "database"
        effect: "NoExecute"
      containers:
      - name: postgres
        image: postgres:13
        env:
        - name: POSTGRES_PASSWORD
          value: "password123"
```

Part 7: Management and Cleanup

Step 11: Managing Taints

```
bash
```

```
# List all taints on all nodes
kubectl get nodes -o json | jq '.items[].spec.taints'

# Remove a specific taint (note the minus sign)
kubectl taint nodes NODE_NAME environment=production:NoSchedule-

# Remove all taints with a specific key
kubectl taint nodes NODE_NAME environment-

# Remove all taints from a node
kubectl taint nodes NODE_NAME --all-
```

Step 12: Monitoring and Troubleshooting

```bash
# Check pod events for taint-related issues
kubectl get events --field-selector reason=FailedScheduling

# View detailed scheduling decisions
kubectl describe pod POD_NAME

# Check node capacity and allocations
kubectl describe nodes

# View pods by node
kubectl get pods -o wide --all-namespaces
```

Part 8: Lab Exercises

Exercise 1: Three-Tier Application

Create a three-tier application with:

- Frontend pods that can run on any node
- Backend pods that prefer dedicated nodes but can run elsewhere
- Database pods that must run only on dedicated database nodes

Exercise 2: Maintenance Scenario

1. Taint a node for maintenance
2. Deploy pods with different toleration strategies

3. Observe eviction behavior
4. Gracefully drain and uncordon the node

Exercise 3: Multi-Environment Cluster

Set up a cluster with:

- Development environment (loose scheduling)
- Staging environment (preferred scheduling)
- Production environment (strict scheduling)

Part 9: Best Practices

Taint Naming Conventions

- Use descriptive keys: `environment`, `workload-type`, `hardware`
- Use consistent values: `production`, `staging`, `development`
- Document your taint strategy

Common Patterns

1. **Environment Isolation**: Separate prod/staging/dev workloads
2. **Hardware Specialization**: GPU, high-memory, SSD nodes
3. **Maintenance Windows**: Controlled pod eviction
4. **Cost Optimization**: Spot instances with tolerations

Security Considerations

- Use RBAC to control who can modify taints
- Regular auditing of node taints
- Document taint purposes and owners

Cleanup

```bash
```

```
# Remove all test pods
kubectl delete pod --all

# Remove all deployments created in this lab
kubectl delete deployment --all

# Remove taints from all nodes
kubectl get nodes -o name | xargs -I {} kubectl taint {} --all-

# Verify cleanup
kubectl describe nodes | grep -i taint
```

Summary

In this lab, you learned:

- How to apply and remove taints from nodes
- How to configure tolerations in pods
- Different taint effects and their behaviors
- Real-world use cases for taints and tolerations
- Best practices for managing node scheduling

Key Commands Reference

```bash
# Apply taint
kubectl taint nodes NODE_NAME key=value:effect

# Remove taint
kubectl taint nodes NODE_NAME key=value:effect-

# View taints
kubectl describe nodes | grep -i taint

# Check pod scheduling
kubectl get pods -o wide
kubectl describe pod POD_NAME
```

This foundation will help you implement sophisticated pod scheduling strategies in your Kubernetes clusters!