Headlamp Multi-Cluster Setup Guide (MicroK8s + Headlamp)

This document explains how to integrate **multiple MicroK8s clusters** into **Headlamp**

Overview

Headlamp is a modern, extensible Kubernetes UI that allows you to monitor and manage multiple Kubernetes clusters from a single interface.

How Multi-Cluster Works

Headlamp connects to Kubernetes clusters using **kubeconfig files**. For **multi-cluster mode**, Headlamp reads a **single kubeconfig** that contains **multiple cluster entries**.

Flow:

1. Individual MicroK8s clusters generate kubeconfig files
2. Kubeconfigs are merged into a single multi-cluster config
3. Config is stored as a Kubernetes Secret
4. Headlamp reads the Secret and presents the cluster switcher UI

Architecture

Component Layout

Component	Location	Purpose
Headlamp	Cluster A	Web UI & cluster management
Merged Kubeconfig	Secret in Cluster A	Multi-cluster authentication
API Servers	All clusters	Kubernetes control plane endpoints

Ingress Controller	Cluster A	HTTPS access to Headlamp UI

Prerequisites

Component	**Installation**
MicroK8s	**sudo snap install microk8s --classic**
kubectl	**sudo snap install kubectl --classic**
Headlamp	**can download Helm or YAML Configuration**

Setup Paths

Choose your deployment approach:

Path A: Quick Test (Development)

Use when:

- Testing multi-cluster functionality
- Development/staging environments

Characteristics:

- Uses insecure-skip-tls-verify: true
- No certificate management
- Public IPs for API access
- NodePort for UI access
- Not production-ready

Path B: Production Secure

Use when:

- Production deployments

- Security compliance required
- Long-term installations

Characteristics:

- Full TLS certificate validation
- VPC peering or private network
- Ingress with Let's Encrypt
- RBAC and authentication
- Production-ready

Installation Steps

Step 1: Choose Headlamp Host Cluster

Decision: Which cluster will run Headlamp?

```
# On Cluster A (chosen as Headlamp host)
hostname
# Example: cluster-a-server
# Verify MicroK8s is running
microk8s status --wait-ready
```

Important: All subsequent Headlamp operations (merge, secret creation, deployment) happen on this cluster.

Step 2: Configure API Server Accessibility

Goal: Make each cluster's API server reachable from the Headlamp pod.

2.1 Determine Your Network Strategy

Option A: Private IPs with VPC Peering (Production)
Option B: Public IPs (Development/Quick Test)

2.2 Configure API Server (Run on EACH cluster)

```
# Stop MicroK8s
sudo microk8s stop
# Edit API server arguments
```

sudo nano /var/snap/microk8s/current/args/kube-apiserver

Add or modify these lines:

For all interfaces (development)
--advertise-address=0.0.0.0
OR for specific IP (production)
--advertise-address=172.31.7.94
Ensure secure port is set
--secure-port=16443

2.3 Configure Cloud Firewall/Security Groups (run for EACH cluster):

Navigate to EC2 → Security Groups → Select your cluster's SG

Add Inbound Rules:

Type	Protocol	Port	Source	Description
Custom TCP	**TCP**	**16443**	**Cluster A IP/32 or 0.0.0.0/0 for testing**	**Allow Headlamp to access API**
Custom TCP	**TCP**	**16443**	**Cluster B IP/32 or 0.0.0.0/0 for testing**	**Allow inter-cluster access**

2.4 Verify API Accessibility

From Cluster A, test reaching Cluster B
curl -k https://<cluster-b-ip>:16443/version
Expected: JSON response with version info
Error: Connection refused = firewall issue
Error: Connection timeout = routing issue

Step 3: TLS Strategy Selection
Choose your TLS approach:

Option A: Secure TLS (Production)
Pros:
Full certificate validation
Production-ready

Cons:
More complex setup
Requires certificate regeneration
When to use: Production, compliance-required environments

Option B: Insecure Skip TLS (Development)
Pros:
Faster setup
No certificate management

Cons:
Man-in-the-middle vulnerability
Not production-ready
Security audit failures
When to use: Development, testing, PoCs only

Step 4: Configure TLS Certificates (Option A Only)
Skip this step if using Option B

4.1 Add Subject Alternative Names (SANs)
Run on EACH cluster:
# Edit certificate template
sudo nano /var/snap/microk8s/current/certs/csr.conf.template

Under [alt_names] section, add:

```
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
DNS.5 = cluster-a.yourdomain.com
# Add your DNS
IP.1 = 127.0.0.1
IP.2 = 172.31.7.94# Private IP
IP.3 = 13.239.17.107# Public IP (if used)
```

4.2 Regenerate Certificates

```
# Backup existing certificates
sudo cp -r /var/snap/microk8s/current/certs /var/snap/microk8s/current/certs.backup
# Stop MicroK8s
sudo microk8s stop
# Remove old certificates
sudo rm /var/snap/microk8s/current/certs/server.crt
sudo rm /var/snap/microk8s/current/certs/server.key
# Start MicroK8s (automatically regenerates certs)
sudo microk8s start
microk8s status --wait-ready
# Verify new certificate includes SANs
openssl x509 -in /var/snap/microk8s/current/certs/server.crt -text -noout | grep -A 10
"Subject Alternative Name"
```

Step 5: Export Kubeconfig from Each Cluster

5.1 Export from Cluster A

```
# Export kubeconfig
```

microk8s config > ~/cluster-a.yaml
 # Verify export
cat ~/cluster-a.yaml | grep "server:"
 # Should show: server: https://127.0.0.1:16443

5.2 Export from Cluster B

Export kubeconfig
microk8s config > ~/cluster-b.yaml
Verify export
cat ~/cluster-b.yaml | grep "server:"

5.3 Transfer Configs to Cluster A

From Cluster A, copy Cluster B's config
scp ubuntu@<cluster-b-ip>:~/cluster-b.yaml ~/cluster-b.yaml
Verify both files exist on Cluster A
ls -lh ~/cluster-*.yaml

Step 6: Edit Kubeconfig Files

Make each kubeconfig unique and accessible.

6.1 Edit cluster-a.yaml

nano ~/cluster-a.yaml
apiVersion: v1
kind: Config
clusters:
- cluster:
certificate-authority-data: LS0tLS1CRU... # Keep existing
For Option B (insecure), add:
insecure-skip-tls-verify: true
Change server from 127.0.0.1 to accessible IP:
server: https://x.x.x.x:16443 # Public IP or DNS

```
name: microk8s-cluster-a # Make unique

contexts:
- context:
cluster: microk8s-cluster-a  # Match cluster name
user: admin-cluster-a   # Make unique
name: cluster-a   # Make unique
current-context: cluster-a

users:
- name: admin-cluster-a   # Make unique
user:
client-certificate-data: LS0tLS1…   # Keep existing
client-key-data: LS0tLS1…     # Keep existing
```

6.2 Edit cluster-b.yaml

Repeat the same process with these unique names:
Cluster name: microk8s-cluster-b
Context name: cluster-b
User name: admin-cluster-b
Server: https://x.x.x.x:16443

Step 7: Install System kubectl on Cluster A

MicroK8s kubectl (microk8s kubectl) doesn't read ~/.kube/config by default. Install the system kubectl for kubeconfig management.

```
# Install system kubectl
sudo snap install kubectl --classic
# Remove any aliases that override kubectl
nano ~/.bashrc
# Remove line: alias kubectl='microk8s kubectl'
# Save and reload
source ~/.bashrc
# Verify you're using system kubectl
which kubectl
```

Should show: /snap/bin/kubectl (not microk8s)

Step 8: Merge Kubeconfigs

8.1 Create .kube Directory

mkdir -p ~/.kube

8.2 Move Config Files

Move to standard location
mv ~/cluster-a.yaml ~/.kube/cluster-a.yaml
mv ~/cluster-b.yaml ~/.kube/cluster-b.yaml

8.3 Merge Configurations

Using KUBECONFIG environment variable
export KUBECONFIG=~/.kube/cluster-a.yaml:~/.kube/cluster-b.yaml
kubectl config view --merge --flatten > ~/.kube/multi-cluster.yaml

 # Unset environment variable
unset KUBECONFIG

8.4 Verify Merge

View merged config
kubectl --kubeconfig=~/.kube/multi-cluster.yaml config view
List clusters
kubectl --kubeconfig=~/.kube/multi-cluster.yaml config get-clusters
Expected output:
NAME
microk8s-cluster-a
microk8s-cluster-b
List contexts
kubectl --kubeconfig=~/.kube/multi-cluster.yaml config get-contexts

8.5 Test Connectivity

```
# Test Cluster A
kubectl --kubeconfig=~/.kube/multi-cluster.yaml --context=cluster-a get nodes
# Test Cluster B
kubectl --kubeconfig=~/.kube/multi-cluster.yaml --context=cluster-b get nodes
# Both should return node lists
```

Step 9: Create Headlamp Namespace

```
# Use system kubectl with merged config
cp ~/.kube/multi-cluster.yaml ~/.kube/config
# Switch to Cluster A context
kubectl config use-context cluster-a
# Verify you're on Cluster A
kubectl get nodes
# Create dedicated namespace for Headlamp
kubectl create namespace headlamp
# Verify namespace
kubectl get namespace headlamp
```

Why a dedicated namespace?

- Separation of concerns
- Easier RBAC management
- Cleaner than mixing with kube-system

Step 10: Create Kubeconfig Secret

```
# Create secret in headlamp namespace
kubectl create secret generic headlamp-kubeconfig \
--from-file=config=$HOME/.kube/multi-cluster.yaml \
-n headlamp
```

Verify secret exists
kubectl get secret headlamp-kubeconfig -n headlamp

Step 11: Deploy Headlamp
Option A: Using Helm
Option B: Using Manual YAML

Edit the headlamp deployment - Option B
args:
- "-kubeconfig=/etc/headlamp/config"
- "-plugins-dir=/headlamp/plugins"

volumeMounts: # Add the volume mount
- name: kubeconfig
mountPath: /etc/headlamp

volumes: # Add the volume
- name: kubeconfig
secret:
secretName: headlamp-kubeconfig

 # Apply the deployment
kubectl apply -f ~/headlamp.yaml

Step 12: Verify Headlamp Deployment
Check pod status
kubectl get pods -n headlamp

Verify kubeconfig is mounted
kubectl exec -n headlamp deployment/headlamp -- ls -la /etc/headlamp/

Test if pod can access both clusters
kubectl exec -n headlamp deployment/headlamp -- \
kubectl --kubeconfig=/etc/headlamp/config config get-contexts

Expected: Shows both cluster-a and cluster-b

Step 13: Configure Ingress with TLS

For deployments, use Ingress with HTTPS:

Enable MicroK8s ingress
microk8s enable ingress
Enable cert-manager for Let's Encrypt
microk8s enable cert-manager

 # Create ClusterIssuer

certificate-issuer.yaml

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
 name: lets-encrypt
spec:
 acme:
   email: your-email@example.com
   server: https://acme-v02.api.letsencrypt.org/directory
   privateKeySecretRef:
     name: lets-encrypt-private-key
   # Add a single challenge solver, HTTP01 using nginx
   solvers:
   - http01:
       ingress:
         class: public
```

kubectl apply -f ~/certificate-issuer.yaml

 # Create Ingress

ingress.yaml

```
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
```

```
  name: headlamp-ingress
  namespace: headlamp
  annotations:
    cert-manager.io/cluster-issuer: "lets-encrypt"
spec:
  tls:
  - secretName: headlamp-tls-secret
    hosts:
    - headlamp.yourdomain.com
  rules:
  - host: headlamp.yourdomain.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: headlamp
            port:
              number: 80
```

kubectl apply -f ~/ingress.yaml
Access via: https://headlamp.yourdomain.com