CKA Practice Questions - Complete Solutions Guide

Comprehensive Solutions for Questions 1-17

CKA Practice Questions - Complete Solutions Guide

Question 1: ConfigMap DNS FQDN Updates

Server: ssh cka6016

Task: Update the ConfigMap used by the Deployment in Namespace lima-control with correct FQDN values for various Kubernetes DNS entries.

Solution

```
ssh cka6016

# Find the Deployment in lima-control namespace
kubectl -n lima-control get deployment

# Identify the ConfigMap used
kubectl -n lima-control get deployment <deployment-name> -o yaml | grep -A10 con
figMap

# List ConfigMaps
kubectl -n lima-control get cm

# Edit the ConfigMap
kubectl -n lima-control edit cm <configmap-name>
```

DNS FQDN Values

Update the ConfigMap with these values:

```
data:
  DNS_1: "kubernetes.default.svc.cluster.local"
  DNS_2: "department.lima-workload.svc.cluster.local"
  DNS_3: "section100.lima-workload.pod.cluster.local"
  DNS_4: "1-2-3-4.kube-system.pod.cluster.local"
```

Explanation

• DNS_1: Service FQDN format: <service-name>.<namespace>.svc.cluster.local
• DNS_2: Headless Service FQDN (same format as regular service)
• DNS_3: Pod FQDN for pods in headless service: <pod-name>.<namespace>.pod.cluster.local
• DNS_4: Pod by IP FQDN: Replace dots with dashes: <ip-with-dashes>.<namespace>.pod.cluster.local

Verification

```
# Get a pod from the deployment
kubectl -n lima-control get pods

# Test DNS resolution
kubectl -n lima-control exec -it <pod-name> -- nslookup kubernetes.default.svc.c
luster.local
kubectl -n lima-control exec -it <pod-name> -- nslookup department.lima-workload
.svc.cluster.local

# Restart deployment to apply ConfigMap changes
kubectl -n lima-control rollout restart deployment <deployment-name>
kubectl -n lima-control rollout status deployment <deployment-name>
```

Question 2: Static Pod + NodePort Service

Server: ssh cka2560

Task: Create a Static Pod on controlplane node and expose it with a NodePort Service.

Solution

```
ssh cka2560

# Create static pod manifest
cat <<EOF > /etc/kubernetes/manifests/my-static-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-static-pod
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:1-alpine
    resources:
      requests:
        cpu: 10m
        memory: 20Mi
EOF

# Wait for pod to be created (static pods get node name suffix)
watch kubectl get pods

# Check the actual pod name and labels
kubectl get pod -l component=my-static-pod -o yaml | grep -A5 labels
# Or
kubectl get pods | grep my-static-pod
POD_NAME=$(kubectl get pods --no-headers | grep my-static-pod | awk '{print $1}'
)
kubectl get pod $POD_NAME -o yaml | grep -A10 "labels:"
```

Create NodePort Service

Static pods typically get labels like tier: node and component name. Check actual labels first:

```
# Method 1: If static pod has predictable labels
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: static-pod-service
  namespace: default
spec:
  type: NodePort
  selector:
    component: my-static-pod
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
EOF
```

Alternative: Manual Endpoint Creation

If labels don't match, create service with manual endpoints:

```
# Get pod IP
POD_IP=$(kubectl get pod $POD_NAME -o jsonpath='{.status.podIP}')

# Create service without selector
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: static-pod-service
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: v1
kind: Endpoints
metadata:
  name: static-pod-service
subsets:
- addresses:
  - ip: $POD_IP
  ports:
  - port: 80
EOF
```

Verification

```
# Check service
kubectl get svc static-pod-service
kubectl get endpoints static-pod-service

# Get NodePort
NODE_PORT=$(kubectl get svc static-pod-service -o jsonpath='{.spec.ports[0].node
Port}')
echo "NodePort: $NODE_PORT"

# Test access
curl 192.168.100.31:$NODE_PORT
```

Question 3: Certificate Information

Server: ssh cka5248

Task: Find Issuer and Extended Key Usage values for Kubelet certificates on worker node.

Solution

```
ssh cka5248

# Connect to worker node
ssh cka5248-node1

# Find kubelet certificates
sudo ls -la /var/lib/kubelet/pki/

# Expected files:
# - kubelet-client-current.pem (or kubelet-client-*.pem)
# - kubelet.crt (server certificate)
```

Examine Kubelet Client Certificate

```
# Client certificate - used for outgoing connections to API server
sudo openssl x509 -in /var/lib/kubelet/pki/kubelet-client-current.pem -text -noo
ut | grep "Issuer:"
sudo openssl x509 -in /var/lib/kubelet/pki/kubelet-client-current.pem -text -noo
ut | grep -A2 "X509v3 Extended Key Usage"
```

Examine Kubelet Server Certificate

```
# Server certificate - used for incoming connections from API server
sudo openssl x509 -in /var/lib/kubelet/pki/kubelet.crt -text -noout | grep "Issu
er:"
sudo openssl x509 -in /var/lib/kubelet/pki/kubelet.crt -text -noout | grep -A2 "
X509v3 Extended Key Usage"
```

Write Results

```
# Exit back to control plane
exit

# Create the output file
cat <<EOF > /opt/course/3/certificate-info.txt
Kubelet Client Certificate:
Issuer: CN=kubernetes
Extended Key Usage: TLS Web Client Authentication

Kubelet Server Certificate:
Issuer: CN=cka5248-node1@<timestamp>
Extended Key Usage: TLS Web Server Authentication
EOF
```

Expected Values

• Client Certificate:
- Issuer: Typically CN=kubernetes or the cluster CA

- Extended Key Usage: TLS Web Client Authentication

• Server Certificate:
- Issuer: Usually self-signed by kubelet or CN=<node-name>@<timestamp>

- Extended Key Usage: TLS Web Server Authentication

Question 4: Readiness/Liveness Probes

Server: ssh cka3200

Task: Create pods with probes where one pod's readiness depends on a service endpoint.

Solution

```
ssh cka3200
```

Create First Pod with Probes

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: ready-if-service-ready
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:1-alpine
    livenessProbe:
      exec:
        command:
        - "true"
      initialDelaySeconds: 5
      periodSeconds: 10
    readinessProbe:
      exec:
        command:
        - sh
        - -c
        - "wget -T2 -O- http://service-am-i-ready:80"
      initialDelaySeconds: 5
      periodSeconds: 5
EOF
```

Verify Pod is Not Ready

```
kubectl get pod ready-if-service-ready
# Should show 0/1 READY because service doesn't exist yet

kubectl describe pod ready-if-service-ready
# Check events - should see readiness probe failing
```

Create Second Pod with Label

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: am-i-ready
  namespace: default
  labels:
    id: cross-server-ready
spec:
  containers:
  - name: nginx
    image: nginx:1-alpine
EOF
```

Verify Service and Endpoints

```
# Check if service exists
kubectl get svc service-am-i-ready

# Check endpoints - should now have the second pod
kubectl get endpoints service-am-i-ready

# Verify first pod becomes ready
kubectl get pod ready-if-service-ready
# Should now show 1/1 READY
```

Complete Verification

```
# Both pods should be running and ready
kubectl get pods ready-if-service-ready,am-i-ready

# Check service has endpoint
kubectl describe svc service-am-i-ready

# Test the connection
kubectl exec ready-if-service-ready -- wget -O- http://service-am-i-ready:80
```

Question 5: Sorting Scripts

Server: ssh cka8448

Task: Create scripts that sort Kubernetes resources using kubectl.

Solution

```
ssh cka8448

# Create directory
mkdir -p /opt/course/5
```

Script 1: Sort Pods by Age

```
cat <<'EOF' > /opt/course/5/find_pods.sh
#!/bin/bash
kubectl get pods -A --sort-by=.metadata.creationTimestamp
EOF

chmod +x /opt/course/5/find_pods.sh
```

Script 2: Sort Pods by UID

```
cat <<'EOF' > /opt/course/5/find_pods_uid.sh
#!/bin/bash
kubectl get pods -A --sort-by=.metadata.uid
EOF

chmod +x /opt/course/5/find_pods_uid.sh
```

Verification

```
# Test the scripts
/opt/course/5/find_pods.sh
/opt/course/5/find_pods_uid.sh
```

Alternative with Output Format

For more detailed output:

```
# With custom columns
cat <<'EOF' > /opt/course/5/find_pods.sh
#!/bin/bash
kubectl get pods -A --sort-by=.metadata.creationTimestamp \
  -o custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,AGE:.metad
ata.creationTimestamp
EOF

cat <<'EOF' > /opt/course/5/find_pods_uid.sh
#!/bin/bash
kubectl get pods -A --sort-by=.metadata.uid \
  -o custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,UID:.metad
ata.uid
EOF
```

Question 6: Fix Kubelet

Server: ssh cka1024

Task: Fix kubelet issue on controlplane node and create a test pod.

Solution

```
ssh cka1024

# Check kubelet status
systemctl status kubelet
```

Common Issues and Fixes

Issue 1: Kubelet Not Running

```
# Check logs
journalctl -u kubelet -n 50 --no-pager

# Start kubelet
sudo systemctl start kubelet
sudo systemctl enable kubelet
```

Issue 2: Configuration File Error

```
# Check kubelet config
sudo cat /var/lib/kubelet/config.yaml

# Common issues:
# - Wrong path to staticPodPath
# - Invalid YAML syntax
# - Wrong certificate paths

# Edit if needed
sudo vi /var/lib/kubelet/config.yaml

# Restart kubelet
sudo systemctl restart kubelet
```

Issue 3: Certificate Issues

```
# Check certificates
ls -la /var/lib/kubelet/pki/

# If certificates are missing or expired
sudo rm -rf /var/lib/kubelet/pki/*
sudo systemctl restart kubelet
```

Issue 4: Wrong Kubelet Service File

```
# Check service file
sudo cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

# Or
sudo cat /usr/lib/systemd/system/kubelet.service

# If modified, reload daemon
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

Verify Node is Ready

```
# Check node status
kubectl get nodes

# Should show:
# NAME       STATUS   ROLES           AGE   VERSION
# cka1024    Ready    control-plane   ...   ...

# Check node details
kubectl describe node cka1024 | grep -A5 Conditions
```

Create Test Pod

```
kubectl run success --image=nginx:1-alpine

# Verify pod is running
kubectl get pod success
kubectl describe pod success
```

Question 7: ETCD Operations

Server: ssh cka2560

Task: Get etcd version and create a snapshot.

Solution

```
ssh cka2560

# Create directory
mkdir -p /opt/course/7
```

Get ETCD Version

```
# Method 1: Direct etcd command (if etcd is in PATH)
ETCDCTL_API=3 etcd --version > /opt/course/7/etcd-version

# Method 2: From container (if etcd runs as static pod)
kubectl -n kube-system exec etcd-cka2560 -- etcd --version > /opt/course/7/etcd-
version

# Method 3: Check the binary
sudo /usr/local/bin/etcd --version > /opt/course/7/etcd-version
```

Create ETCD Snapshot

```
# Find etcd certificate locations
sudo cat /etc/kubernetes/manifests/etcd.yaml | grep -E "cert|key|ca"

# Create snapshot
ETCDCTL_API=3 etcdctl snapshot save /opt/course/7/etcd-snapshot.db \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key

# Alternative: Using etcdctl from etcd pod
kubectl -n kube-system exec etcd-cka2560 -- sh -c \
  "ETCDCTL_API=3 etcdctl snapshot save /tmp/snapshot.db \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key"

# Copy from pod
kubectl -n kube-system cp etcd-cka2560:/tmp/snapshot.db /opt/course/7/etcd-snaps
hot.db
```

Verify Snapshot

```
ETCDCTL_API=3 etcdctl snapshot status /opt/course/7/etcd-snapshot.db --write-out
=table

# Should show output like:
# +----------+----------+------------+------------+
# |   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE |
# +----------+----------+------------+------------+
# | 12345678 |    1234  |       567  |     8.9 MB |
# +----------+----------+------------+------------+
```

Verification

```
ls -lh /opt/course/7/
cat /opt/course/7/etcd-version
```

Question 8: Component Analysis

Server: ssh cka8448

Task: Determine how each control plane component is installed.

Solution

```
ssh cka8448

mkdir -p /opt/course/8
```

Check Each Component

Kubelet

```
# Check if kubelet is a system service
systemctl status kubelet
# Result: process (systemd service)
```

Kube-apiserver

```
# Check if running as static pod
ls /etc/kubernetes/manifests/ | grep apiserver

# Or check for process
ps aux | grep kube-apiserver

# Check as pod
kubectl -n kube-system get pod | grep apiserver

# Result: static-pod (if in /etc/kubernetes/manifests/)
```

Kube-scheduler

```
# Check for static pod
ls /etc/kubernetes/manifests/ | grep scheduler

kubectl -n kube-system get pod | grep scheduler

# Result: static-pod
```

Kube-controller-manager

```
# Check for static pod
ls /etc/kubernetes/manifests/ | grep controller

kubectl -n kube-system get pod | grep controller

# Result: static-pod
```

ETCD

```
# Check for static pod
ls /etc/kubernetes/manifests/ | grep etcd

kubectl -n kube-system get pod | grep etcd

# Result: static-pod
```

DNS

```
# Check DNS pods
kubectl -n kube-system get deploy,pod | grep -E "coredns|kube-dns"

# Result: pod coredns (or pod kube-dns)
```

Write Results

```
cat <<EOF > /opt/course/8/controlplane-components.txt
# /opt/course/8/controlplane-components.txt
kubelet: process
kube-apiserver: static-pod
kube-scheduler: static-pod
kube-controller-manager: static-pod
```

```
etcd: static-pod
dns: pod coredns
EOF
```

Verification

```
cat /opt/course/8/controlplane-components.txt
```

Question 9: Manual Scheduling

Server: ssh cka5248

Task: Stop scheduler, manually schedule a pod, then restart scheduler.

Solution

```
ssh cka5248
```

Stop Kube-Scheduler

```
# Method 1: Move static pod manifest
sudo mv /etc/kubernetes/manifests/kube-scheduler.yaml /tmp/

# Method 2: Rename the manifest
sudo mv /etc/kubernetes/manifests/kube-scheduler.yaml /etc/kubernetes/kube-sched
uler.yaml.disabled

# Verify scheduler is gone
kubectl -n kube-system get pods | grep scheduler
# Should show no results after ~30 seconds
```

Create Unscheduled Pod

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: manual-schedule
  namespace: default
spec:
  containers:
  - name: httpd
    image: httpd:2-alpine
EOF

# Verify pod is created but not scheduled
kubectl get pod manual-schedule
# Should show: Pending status

kubectl get pod manual-schedule -o yaml | grep nodeName
# Should show: nodeName is empty or not present
```

Manually Schedule the Pod

```
# Get the node name
kubectl get nodes

# Method 1: Using kubectl patch
kubectl patch pod manual-schedule -p '{"spec":{"nodeName":"cka5248"}}'

# Method 2: Edit the pod directly
kubectl edit pod manual-schedule
# Add under spec:
#   nodeName: cka5248

# Method 3: Create binding object
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Binding
metadata:
  name: manual-schedule
  namespace: default
target:
  apiVersion: v1
  kind: Node
  name: cka5248
EOF
```

Verify Pod is Running

```
kubectl get pod manual-schedule
```

```
# Should show: Running on cka5248

kubectl get pod manual-schedule -o wide
```

Restart Kube-Scheduler

```
# Move manifest back
sudo mv /tmp/kube-scheduler.yaml /etc/kubernetes/manifests/

# Verify scheduler is running
kubectl -n kube-system get pods | grep scheduler
```

Create Second Pod to Test Scheduler

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: manual-schedule2
  namespace: default
spec:
  containers:
  - name: httpd
    image: httpd:2-alpine
EOF

# Verify it gets scheduled automatically
kubectl get pod manual-schedule2 -o wide
# Should be scheduled on cka5248-node1 (worker node)
```

Question 10: Storage Class and PVC

Server: ssh cka6016

Task: Create StorageClass with local-path provisioner and adjust a Job to use PVC.

Solution

```
ssh cka6016
```

Create StorageClass

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-backup
provisioner: rancher.io/local-path
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain
EOF

# Verify
kubectl get sc local-backup
```

View Original Job

```
cat /opt/course/10/backup.yaml
```

Create PVC

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: backup-pvc
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: local-backup
  resources:
    requests:
      storage: 50Mi
EOF
```

Update Job to Use PVC

```
# Backup original
cp /opt/course/10/backup.yaml /opt/course/10/backup.yaml.bak

# Edit the Job
cat <<EOF > /opt/course/10/backup.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: backup-job
  namespace: default
spec:
  template:
    spec:
      containers:
      - name: backup
        image: busybox:1
        command:
        - sh
        - -c
        - "echo 'Backup data' > /backup/backup.txt && date >> /backup/backup.txt
"
        volumeMounts:
        - name: backup-volume
          mountPath: /backup
      restartPolicy: OnFailure
      volumes:
      - name: backup-volume
        persistentVolumeClaim:
          claimName: backup-pvc
EOF
```

Deploy and Verify

```
# Delete old job if exists
kubectl delete job backup-job --ignore-not-found

# Apply updated job
kubectl apply -f /opt/course/10/backup.yaml

# Check job status
kubectl get job backup-job
kubectl get pods -l job-name=backup-job

# Check PVC status
kubectl get pvc backup-pvc
# Should show: Bound status

# Check PV was created
kubectl get pv
# Should see a PV bound to backup-pvc

# Verify job completed
kubectl get job backup-job
# Should show: COMPLETIONS 1/1
```

Verification

```
# Check the created PV
kubectl describe pv <pv-name>

# Check PVC
kubectl describe pvc backup-pvc

# Check job logs
kubectl logs -l job-name=backup-job
```

Question 11: Secrets Management

Server: ssh cka2560

Task: Create namespace, mount existing Secret, and create new Secret with environment variables.

Solution

```
ssh cka2560
```

Create Namespace

```
kubectl create namespace secret
```

Create Pod with Sleep Command

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
  namespace: secret
spec:
  containers:
  - name: busybox
    image: busybox:1
    command:
    - sh
    - -c
    - "sleep 1d"
EOF
```

Check Existing Secret File

```
cat /opt/course/11/secret1.yaml
```

Create Secret from File

```
kubectl apply -f /opt/course/11/secret1.yaml -n secret

# Verify secret was created
kubectl -n secret get secret
```

Update Pod to Mount Secret

```
# Delete the pod
kubectl -n secret delete pod secret-pod

# Recreate with secret mounted
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
  namespace: secret
spec:
  containers:
  - name: busybox
    image: busybox:1
    command:
    - sh
    - -c
    - "sleep 1d"
    volumeMounts:
    - name: secret1
      mountPath: /tmp/secret1
      readOnly: true
  volumes:
  - name: secret1
    secret:
      secretName: secret1
EOF
```

Create Second Secret

```
kubectl -n secret create secret generic secret2 \
  --from-literal=user=user1 \
  --from-literal=pass=1234

# Verify
kubectl -n secret get secret secret2
kubectl -n secret describe secret secret2
```

Update Pod with Environment Variables

```
# Delete pod again
kubectl -n secret delete pod secret-pod

# Recreate with both secret volume and env vars
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
  namespace: secret
spec:
  containers:
  - name: busybox
    image: busybox:1
    command:
    - sh
    - -c
    - "sleep 1d"
    env:
    - name: APP_USER
      valueFrom:
        secretKeyRef:
          name: secret2
          key: user
    - name: APP_PASS
      valueFrom:
        secretKeyRef:
          name: secret2
          key: pass
    volumeMounts:
    - name: secret1
      mountPath: /tmp/secret1
      readOnly: true
  volumes:
  - name: secret1
    secret:
      secretName: secret1
EOF
```

Verification

```
# Check pod is running
kubectl -n secret get pod secret-pod

# Verify secret1 is mounted
kubectl -n secret exec secret-pod -- ls -la /tmp/secret1
kubectl -n secret exec secret-pod -- cat /tmp/secret1/<key-name>

# Verify environment variables
kubectl -n secret exec secret-pod -- env | grep APP_
# Should show:
# APP_USER=user1
# APP_PASS=1234
```

Question 12: Node Selector Scheduling

Server: ssh cka5248

Task: Schedule a pod only on controlplane nodes without adding new labels.

Solution

```
ssh cka5248
```

Check Existing Node Labels

```
# View all node labels
kubectl get nodes --show-labels

# Check controlplane node specifically
kubectl describe node cka5248 | grep -A10 Labels

# Common controlplane labels:
# - node-role.kubernetes.io/control-plane=""
# - node-role.kubernetes.io/master="" (older versions)
```

Create Pod with Node Selector

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: default
spec:
  containers:
  - name: pod1-container
    image: httpd:2-alpine
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
EOF
```

Alternative for Older Kubernetes Versions

If the cluster uses master label:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: default
spec:
  containers:
  - name: pod1-container
    image: httpd:2-alpine
  nodeSelector:
    node-role.kubernetes.io/master: ""
EOF
```

Handle Taints (if necessary)

```
# Check if controlplane has taints
kubectl describe node cka5248 | grep -A5 Taints

# If there's a NoSchedule taint, add toleration
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: default
spec:
  containers:
  - name: pod1-container
    image: httpd:2-alpine
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
  tolerations:
  - key: node-role.kubernetes.io/control-plane
    operator: Exists
    effect: NoSchedule
EOF
```

Verification

```
# Check pod is running on controlplane
kubectl get pod pod1 -o wide

# Verify it's on the control plane node
kubectl get pod pod1 -o jsonpath='{.spec.nodeName}'
# Should output: cka5248

# Check pod details
kubectl describe pod pod1
```

Question 13: Multi-Container Pod

Server: ssh cka3200

Task: Create a pod with three containers sharing a volume with specific functionality.

Solution

```
ssh cka3200
```

Create Multi-Container Pod

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-playground
  namespace: default
spec:
  containers:
  # Container 1: nginx with node name env var
  - name: c1
    image: nginx:1-alpine
    env:
    - name: MY_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    volumeMounts:
    - name: shared-volume
      mountPath: /vol

  # Container 2: writes date to shared volume
  - name: c2
    image: busybox:1
    command:
    - sh
    - -c
    - "while true; do date >> /vol/date.log; sleep 1; done"
    volumeMounts:
    - name: shared-volume
      mountPath: /vol

  # Container 3: tail the log file
  - name: c3
    image: busybox:1
    command:
    - sh
    - -c
    - "tail -f /vol/date.log"
    volumeMounts:
    - name: shared-volume
      mountPath: /vol

  volumes:
  - name: shared-volume
    emptyDir: {}
EOF
```

Verification

```
# Check all containers are running
kubectl get pod multi-container-playground

# Should show: 3/3 Running

# Check c1 has the node name env variable
kubectl exec multi-container-playground -c c1 -- env | grep MY_NODE_NAME

# Check c2 is writing to the file
kubectl exec multi-container-playground -c c2 -- cat /vol/date.log

# Check c3 logs (should show continuous date output)
```

```
kubectl logs multi-container-playground -c c3

# Follow logs in real-time
kubectl logs multi-container-playground -c c3 -f
```

Detailed Verification

```
# Verify shared volume works
kubectl exec multi-container-playground -c c1 -- ls -la /vol
kubectl exec multi-container-playground -c c2 -- ls -la /vol
kubectl exec multi-container-playground -c c3 -- ls -la /vol

# All should show date.log

# Check file is growing
kubectl exec multi-container-playground -c c1 -- wc -l /vol/date.log
sleep 5
kubectl exec multi-container-playground -c c1 -- wc -l /vol/date.log
# Line count should increase
```

Question 14: Cluster Information

Server: ssh cka8448

Task: Gather various information about the cluster configuration.

Solution

```
ssh cka8448

mkdir -p /opt/course/14
```

1. Count Controlplane Nodes

```
kubectl get nodes -l node-role.kubernetes.io/control-plane --no-headers | wc -l

# Or
kubectl get nodes --no-headers | grep -i "control-plane\|master" | wc -l
```

2. Count Worker Nodes

```
# Workers are nodes without control-plane role
kubectl get nodes --no-headers | grep -v "control-plane\|master" | wc -l

# Or use label selector
kubectl get nodes -l '!node-role.kubernetes.io/control-plane' --no-headers | wc
-l
```

3. Find Service CIDR

```
# Method 1: From kube-apiserver manifest
sudo cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep service-cluster-ip
-range

# Method 2: From kube-controller-manager
sudo cat /etc/kubernetes/manifests/kube-controller-manager.yaml | grep cluster-c
idr

# Method 3: Check existing services
kubectl cluster-info dump | grep -m 1 service-cluster-ip-range

# Method 4: From API server process
ps aux | grep kube-apiserver | grep service-cluster-ip-range
```

4. Find CNI Plugin and Config

```
# Check CNI plugin directory
ls /etc/cni/net.d/

# Read CNI config
cat /etc/cni/net.d/*.conf* | head -20

# Check running CNI pods
kubectl -n kube-system get pods | grep -E "calico|flannel|weave|cilium|canal"

# Check daemonsets
kubectl -n kube-system get ds

# Common CNI configs:
# - Calico: /etc/cni/net.d/10-calico.conflist
# - Flannel: /etc/cni/net.d/10-flannel.conflist
# - Weave: /etc/cni/net.d/10-weave.conflist
```

5. Static Pod Suffix

```
# Static pods get the node name as suffix
# Format: <pod-name>-<node-name>

# Check existing static pods
kubectl -n kube-system get pods -o wide | grep cka8448

# The suffix is: -cka8448
```

Write Results

```
cat <<EOF > /opt/course/14/cluster-info
# /opt/course/14/cluster-info
1: 1
2: 2
3: 10.96.0.0/12
4: Calico, /etc/cni/net.d/10-calico.conflist
5: -cka8448
EOF
```

Example Results Explanation

- 1: 1 controlplane node
- 2: 2 worker nodes (non-controlplane)
- 3: Service CIDR (e.g., 10.96.0.0/12 or 10.32.0.0/16)
- 4: CNI Plugin name and config file path
- 5: Static pod suffix (the node hostname with dash prefix)

Question 15: Events and Logging

Server: ssh cka6016

Task: Create scripts for viewing events and log events from pod/container deletions.

Solution

```
ssh cka6016

mkdir -p /opt/course/15
```

Create Event Viewing Script

```
cat <<'EOF' > /opt/course/15/cluster_events.sh
#!/bin/bash
kubectl get events -A --sort-by=.metadata.creationTimestamp
EOF

chmod +x /opt/course/15/cluster_events.sh

# Test it
/opt/course/15/cluster_events.sh
```

Delete Kube-Proxy Pod and Capture Events

```
# Get kube-proxy pod name
KUBE_PROXY_POD=$(kubectl -n kube-system get pod -l k8s-app=kube-proxy -o jsonpat
h='{.items[0].metadata.name}')

echo "Deleting pod: $KUBE_PROXY_POD"

# Delete the pod
kubectl -n kube-system delete pod $KUBE_PROXY_POD

# Wait a moment for events to be generated
sleep 3

# Capture events related to pod deletion
kubectl get events -n kube-system --sort-by=.metadata.creationTimestamp | grep -
i "$KUBE_PROXY_POD\|kube-proxy" > /opt/course/15/pod_kill.log

# Alternative: Get events for last few minutes
kubectl get events -n kube-system --sort-by=.metadata.creationTimestamp --field-
selector involvedObject.name=$KUBE_PROXY_POD > /opt/course/15/pod_kill.log
```

Kill Container and Capture Events

```
# Wait for new kube-proxy pod to be running
sleep 10
kubectl -n kube-system get pods -l k8s-app=kube-proxy

# Get new pod name
NEW_KUBE_PROXY_POD=$(kubectl -n kube-system get pod -l k8s-app=kube-proxy -o jso
npath='{.items[0].metadata.name}')

# Get the node where it's running
NODE=$(kubectl -n kube-system get pod $NEW_KUBE_PROXY_POD -o jsonpath='{.spec.no
deName}')

echo "Pod $NEW_KUBE_PROXY_POD is running on node $NODE"

# SSH to the node if needed, or use crictl on control plane
# Get container ID
CONTAINER_ID=$(kubectl -n kube-system get pod $NEW_KUBE_PROXY_POD -o jsonpath='{
.status.containerStatuses[0].containerID}' | cut -d'/' -f3)

echo "Container ID: $CONTAINER_ID"

# Kill the container using crictl
sudo crictl stop $CONTAINER_ID
# Or force kill
sudo crictl rm -f $CONTAINER_ID
```

```
# Wait for events
sleep 3

# Capture events
kubectl get events -n kube-system --sort-by=.metadata.creationTimestamp | grep -
i "$NEW_KUBE_PROXY_POD\|kube-proxy" > /opt/course/15/container_kill.log
```

Alternative Method (from node)

```
# If you need to SSH to worker node
NODE=$(kubectl -n kube-system get pod -l k8s-app=kube-proxy -o jsonpath='{.items
[0].spec.nodeName}')

ssh $NODE

# On the node, find and kill container
sudo crictl ps | grep kube-proxy
CONTAINER_ID=$(sudo crictl ps | grep kube-proxy | awk '{print $1}')
sudo crictl stop $CONTAINER_ID

exit

# Back on control plane, capture events
kubectl get events -n kube-system --sort-by=.metadata.creationTimestamp | tail -
20 > /opt/course/15/container_kill.log
```

Verification

```
# Check all files are created
ls -lh /opt/course/15/

# View contents
cat /opt/course/15/cluster_events.sh
head -20 /opt/course/15/pod_kill.log
head -20 /opt/course/15/container_kill.log
```

Question 16: Resources and Namespaces

Server: ssh cka3200

Task: List all namespaced resources and find namespace with most Roles.

Solution

```
ssh cka3200

mkdir -p /opt/course/16
```

List All Namespaced Resources

```
# Get all API resources that are namespaced
kubectl api-resources --namespaced=true -o name > /opt/course/16/resources.txt

# Verify
cat /opt/course/16/resources.txt
```

Find Namespace with Most Roles

```
# List all project-* namespaces
kubectl get namespaces | grep "^project-"

# Count roles in each project-* namespace
for ns in $(kubectl get namespaces -o jsonpath='{.items[?(@.metadata.name matche
s "^project-")].metadata.name}'); do
  count=$(kubectl -n $ns get roles --no-headers 2>/dev/null | wc -l)
  echo "$ns: $count"
done

# Alternative using a more direct approach
for ns in $(kubectl get ns --no-headers | grep "^project-" | awk '{print $1}');
do
  count=$(kubectl -n $ns get roles --no-headers 2>/dev/null | wc -l)
  echo "$ns $count"
done | sort -k2 -rn | head -1
```

Write Result

```
# Find the namespace with highest count
HIGHEST_NS=""
HIGHEST_COUNT=0

for ns in $(kubectl get ns --no-headers | grep "^project-" | awk '{print $1}');
do
  count=$(kubectl -n $ns get roles --no-headers 2>/dev/null | wc -l)
  if [ $count -gt $HIGHEST_COUNT ]; then
    HIGHEST_COUNT=$count
    HIGHEST_NS=$ns
  fi
done

# Write to file
echo "$HIGHEST_NS: $HIGHEST_COUNT" > /opt/course/16/crowded-namespace.txt

# Verify
cat /opt/course/16/crowded-namespace.txt
```

Alternative One-Liner

```
# Using kubectl and shell processing
kubectl get ns -o name | grep project- | cut -d'/' -f2 | \
  while read ns; do echo "$ns $(kubectl -n $ns get roles --no-headers 2>/dev/nul
l | wc -l)"; done | \
  sort -k2 -rn | head -1 > /opt/course/16/crowded-namespace.txt
```

Verification

```
# Check both files
cat /opt/course/16/resources.txt | head -20
cat /opt/course/16/crowded-namespace.txt
```

```
# Verify the count manually
NAMESPACE=$(cat /opt/course/16/crowded-namespace.txt | awk '{print $1}' | tr -d
':')
kubectl -n $NAMESPACE get roles
```

Question 17: Kustomize and CRDs

Server: ssh cka6016

Task: Modify Kustomize config to fix operator permissions and add a new Student resource.

Solution

```
ssh cka6016
```

Check Current Operator Status

```
# Check operator pod
kubectl get pods -A | grep operator

# Check logs to find missing permissions
kubectl logs -n <operator-namespace> <operator-pod-name>

# Example error: "cannot list resource 'students' in API group 'example.com'"
```

Explore Kustomize Structure

```
cd /opt/course/17/operator

# View directory structure
tree
# Or
ls -la
find . -type f

# Check base directory
ls -la base/
cat base/kustomization.yaml
cat base/role.yaml
```

Find Required CRDs

```
# Check operator logs for missing permissions
kubectl logs <operator-pod-name> -n <operator-namespace> | grep -i "forbidden\|d
enied\|cannot"

# Check existing CRDs
kubectl get crd

# Common CRDs the operator might need:
# - students.example.com
# - teachers.example.com
# - courses.example.com
```

Update Role Permissions

```
# Edit the role in base directory
cd /opt/course/17/operator/base

# Backup original
cp role.yaml role.yaml.bak

# Edit role.yaml to add missing permissions
cat <<EOF >> role.yaml
- apiGroups: ["example.com"]
  resources: ["students"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["example.com"]
  resources: ["teachers"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["example.com"]
  resources: ["courses"]
  verbs: ["get", "list", "watch"]
EOF
```

Add New Student Resource

```
# Check if there are existing student resources
```

```
ls -la base/ | grep -i student
cat base/student*.yaml

# Create student4 resource
cat <<EOF > base/student4.yaml
apiVersion: example.com/v1
kind: Student
metadata:
  name: student4
spec:
  name: "John Doe"
  description: "Fourth year computer science student"
  grade: "A"
EOF

# Update kustomization.yaml to include the new resource
cat base/kustomization.yaml
```

Update Kustomization

```
# Edit base/kustomization.yaml
cd /opt/course/17/operator/base

# Backup
cp kustomization.yaml kustomization.yaml.bak

# Add student4 to resources list
vi kustomization.yaml

# Add under resources:
# - student4.yaml
```

Example kustomization.yaml:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- namespace.yaml
- role.yaml
- rolebinding.yaml
- serviceaccount.yaml
- deployment.yaml
- student1.yaml
- student2.yaml
- student3.yaml
- student4.yaml  # Add this line

namespace: operator-system
```

Deploy Changes

```
# Deploy to prod
cd /opt/course/17/operator

kubectl kustomize prod | kubectl apply -f -

# Alternative: Delete and recreate
kubectl kustomize prod | kubectl delete -f - --ignore-not-found
kubectl kustomize prod | kubectl apply -f -
```

Verification

```
# Check operator pod is running
kubectl get pods -n operator-system

# Check logs (should not show permission errors)
kubectl logs -n operator-system <operator-pod-name>

# Verify role has correct permissions
kubectl get role -n operator-system operator-role -o yaml

# Check students
kubectl get students -n operator-system

# Verify student4 exists
```

```
kubectl get student student4 -n operator-system -o yaml
```

Troubleshooting

```
# If changes didn't apply
kubectl kustomize prod

# Check for syntax errors
kubectl kustomize prod | kubectl apply -f - --dry-run=client

# Check role
kubectl describe role operator-role -n operator-system

# Check rolebinding
kubectl describe rolebinding operator-rolebinding -n operator-system
```

Summary

This guide covers all 17 CKA practice questions with detailed solutions including:

- DNS and networking configuration
- Static Pods and Services
- Certificate management
- Pod probes and health checks
- kubectl sorting and scripting
- Kubelet troubleshooting
- ETCD backup and restore
- Component analysis
- Manual scheduling
- Storage Classes and PVCs
- Secrets management
- Node scheduling with selectors
- Multi-container pods
- Cluster information gathering
- Event logging and monitoring
- Resource discovery
- Kustomize and CRD management

Each solution includes verification steps and common troubleshooting approaches. Good luck with your CKA exam!