

CKA Practice Exam Solutions

Complete solutions for Certified Kubernetes Administrator (CKA) practice questions.

Question 1: Kubeconfig Information Extraction

Task: Extract kubeconfig information from `/opt/course/1/kubeconfig` on `cka9412`

Solution

```
ssh cka9412

# Part 1: Extract all context names (one per line)
kubectl config get-contexts -o name --kubeconfig=/opt/course/1/kubeconfig > /opt/course/1/contexts

# Part 2: Get the current context
kubectl config current-context --kubeconfig=/opt/course/1/kubeconfig > /opt/course/1/current-context

# Part 3: Extract and decode the client-certificate for user account-0027
kubectl config view --kubeconfig=/opt/course/1/kubeconfig --raw -o jsonpath='{.users[?(@.name=="account-0027")].data}' > /opt/course/1/cert

# Verification
cat /opt/course/1/contexts
cat /opt/course/1/current-context
head /opt/course/1/cert # Should show -----BEGIN CERTIFICATE-----
```

Key Commands

- `kubectl config get-contexts -o name` - List all context names
 - `kubectl config current-context` - Show active context
 - `kubectl config view --raw` - Show actual base64-encoded data
 - `base64 -d` - Decode base64 data
-

Question 2: MinIO Operator with Helm

Task: Install MinIO Operator using Helm in namespace `minio` on `cka7968`

Solution

```
ssh cka7968

# Step 1: Create namespace
kubectl create namespace minio

# Step 2: Add MinIO Helm repository
helm repo add minio https://operator.min.io/
helm repo update

# Step 3: Install MinIO Operator
helm install minio-operator minio/operator --namespace minio

# Step 4: Wait for operator to be ready
kubectl wait --for=condition=ready pod -l app.kubernetes.io/name=operator -n minio --timeout=

# Step 5: Update Tenant YAML to add enableSFTP: true
sed -i '/features:/a\    enableSFTP: true' /opt/course/2/minio-tenant.yaml

# Alternative: Manual edit
# vi /opt/course/2/minio-tenant.yaml
# Add under features section:
#   enableSFTP: true

# Step 6: Create Tenant resource
kubectl apply -f /opt/course/2/minio-tenant.yaml

# Verification
helm list -n minio
kubectl get tenants -n minio
kubectl get pods -n minio
```

Key Points

- Helm release name: `minio-operator`
- Namespace: `minio`
- Feature to add: `enableSFTP: true` under `features` section
- MinIO operator manages MinIO tenants via CRDs

Question 3: Scale Down Pods

Task: Scale down `o3db-*` pods from 2 to 1 replica in namespace `project-h800` on `cka3962`

Solution

```
ssh cka3962

# Step 1: Check existing pods
kubectl get pods -n project-h800 | grep o3db

# Step 2: Identify the controller (Deployment, StatefulSet, etc.)
kubectl get deployments,statefulsets -n project-h800 | grep o3db

# Step 3: Scale down to 1 replica
# For Deployment:
kubectl scale deployment o3db -n project-h800 --replicas=1

# For StatefulSet:
kubectl scale statefulset o3db -n project-h800 --replicas=1

# Or find all matching resources and scale:
kubectl get deploy,sts -n project-h800 -o name | grep o3db | xargs -I {} kubectl scale {} -n project-h800 --replicas=1

# Verification
kubectl get pods -n project-h800 | grep o3db
# Should show only 1 pod running
```

Key Commands

- `kubectl scale` - Scale replicas of a resource
- `--replicas=N` - Set desired replica count
- Can scale: Deployment, StatefulSet, ReplicaSet, ReplicationController

Question 4: Identify Pods at Risk of Termination

Task: Find pods that would be terminated first under resource pressure in namespace `project-c13` on `cka2556`

Solution

```
ssh cka2556

# Step 1: Get pods with QoS class and Priority
kubectl get pods -n project-c13 -o custom-columns=\
NAME:.metadata.name,\
QOS:.status.qosClass,\
PRIORITY:.spec.priority

# Step 2: Identify BestEffort pods (terminated first)
kubectl get pods -n project-c13 -o json | jq -r '.items[] |
  select(.status.qosClass == "BestEffort") |
  .metadata.name' > /opt/course/4/pods-terminated-first.txt

# Step 3: If no BestEffort, find lowest priority Burstable
if [ ! -s /opt/course/4/pods-terminated-first.txt ]; then
  kubectl get pods -n project-c13 -o json | jq -r '.items |
    sort_by(.spec.priority // 0,
      (if .status.qosClass == "BestEffort" then 0
        elif .status.qosClass == "Burstable" then 1
        else 2 end)) |
    .[].metadata.name' | head -n 2 > /opt/course/4/pods-terminated-first.txt
fi

# Verification
cat /opt/course/4/pods-terminated-first.txt
```

QoS Classes (Eviction Order)

1. **BestEffort** - No resource requests/limits (terminated FIRST)
2. **Burstable** - Partial requests/limits (terminated SECOND)
3. **Guaranteed** - Requests = Limits (terminated LAST)

Within same QoS class, lower priority pods are terminated first.

Question 5: Replace Autoscaler with HPA using Kustomize

Task: Replace external autoscaler with HPA in `api-gateway` using Kustomize on `cka5774`

Solution

```

ssh cka5774

cd /opt/course/5/api-gateway

# Step 1: Remove ConfigMap reference from base
cd base
sed -i '/horizontal-scaling-config/d' kustomization.yaml
rm -f horizontal-scaling-config.yaml 2>/dev/null

# Step 2: Create HPA in base
cat > hpa.yaml <<'EOF'
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api-gateway
  minReplicas: 2
  maxReplicas: 4
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
EOF

# Step 3: Add HPA to base kustomization
echo "  - hpa.yaml" >> kustomization.yaml

# Step 4: Create prod patch for max 6 replicas
cd ../prod
cat > hpa-patch.yaml <<'EOF'
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
spec:
  maxReplicas: 6
EOF

# Step 5: Add patch to prod kustomization
if ! grep -q "patchesStrategicMerge:" kustomization.yaml; then
  echo "patchesStrategicMerge:" >> kustomization.yaml
fi
echo "  - hpa-patch.yaml" >> kustomization.yaml

# Step 6: Preview changes
cd /opt/course/5/api-gateway
kubectl kustomize staging/
kubectl kustomize prod/

```

```
# Step 7: Apply changes
kubectl kustomize /opt/course/5/api-gateway/staging | kubectl apply -f -
kubectl kustomize /opt/course/5/api-gateway/prod | kubectl apply -f -

# Verification
kubectl get hpa api-gateway -n api-gateway-staging
kubectl get hpa api-gateway -n api-gateway-prod
```

Key Points

- **Staging HPA:** min=2, max=4, CPU=50%
- **Prod HPA:** min=2, max=6, CPU=50%
- ConfigMap `horizontal-scaling-config` removed
- Uses Kustomize strategic merge patches

Question 6: PersistentVolume, PVC, and Deployment

Task: Create PV, PVC, and Deployment with volume mount on `cka7968`

Solution

```
ssh cka7968
```

```
# Step 1: Create PersistentVolume
```

```
kubectl apply -f - <<EOF
```

```
apiVersion: v1
```

```
kind: PersistentVolume
```

```
metadata:
```

```
  name: safari-pv
```

```
spec:
```

```
  capacity:
```

```
    storage: 2Gi
```

```
  accessModes:
```

```
    - ReadWriteOnce
```

```
  hostPath:
```

```
    path: /Volumes/Data
```

```
EOF
```

```
# Step 2: Create namespace if needed
```

```
kubectl create namespace project-t230 --dry-run=client -o yaml | kubectl apply -f -
```

```
# Step 3: Create PersistentVolumeClaim
```

```
kubectl apply -f - <<EOF
```

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
```

```
metadata:
```

```
  name: safari-pvc
```

```
  namespace: project-t230
```

```
spec:
```

```
  accessModes:
```

```
    - ReadWriteOnce
```

```
  resources:
```

```
    requests:
```

```
      storage: 2Gi
```

```
EOF
```

```
# Step 4: Create Deployment with volume mount
```

```
kubectl apply -f - <<EOF
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: safari
```

```
  namespace: project-t230
```

```
spec:
```

```
  replicas: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: safari
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: safari
```

```
    spec:
```

```
      containers:
```

```
        - name: safari
```

```
          image: httpd:2-alpine
```

```
    volumeMounts:
      - name: safari-volume
        mountPath: /tmp/safari-data
  volumes:
    - name: safari-volume
      persistentVolumeClaim:
        claimName: safari-pvc
EOF

# Verification
kubectl get pv safari-pv
kubectl get pvc safari-pvc -n project-t230
kubectl get deployment safari -n project-t230
kubectl get pods -n project-t230

# Test volume mount
POD=$(kubectl get pod -n project-t230 -l app=safari -o jsonpath='{.items[0].metadata.name}')
kubectl exec -n project-t230 $POD -- ls -la /tmp/safari-data
```

Key Points

- PV: 2Gi, ReadWriteOnce, hostPath `/Volumes/Data`, no storageClassName
- PVC: 2Gi, ReadWriteOnce, no storageClassName
- Deployment: mounts volume at `/tmp/safari-data`
- Image: `httpd:2-alpine`

Question 7: Metrics Server Scripts

Task: Create bash scripts for node and pod resource usage on `cka5774`

Solution

```
ssh cka5774

mkdir -p /opt/course/7

# Create node.sh
cat > /opt/course/7/node.sh <<'EOF'
#!/bin/bash
kubectl top nodes
EOF

# Create pod.sh
cat > /opt/course/7/pod.sh <<'EOF'
#!/bin/bash
kubectl top pods --all-namespaces --containers
EOF

# Make scripts executable
chmod +x /opt/course/7/node.sh
chmod +x /opt/course/7/pod.sh

# Test scripts
echo "=== Testing node.sh ==="
/opt/course/7/node.sh

echo ""
echo "=== Testing pod.sh ==="
/opt/course/7/pod.sh

# Verification
ls -la /opt/course/7/
```

Key Commands

- `kubectl top nodes` - Show node CPU/memory usage
- `kubectl top pods --all-namespaces --containers` - Show pod and container resource usage
- Requires metrics-server to be installed in cluster

Question 8: Add Worker Node to Cluster

Task: Update worker node Kubernetes version and join to cluster on `cka3962`

Solution

```
ssh cka3962

# Step 1: Get control plane version
KUBE_VERSION=$(kubectl version -o json | jq -r '.serverVersion.gitVersion' | sed 's/v//')
echo "Control plane version: $KUBE_VERSION"

# Step 2: Generate join command
JOIN_CMD=$(sudo kubeadm token create --print-join-command)
echo "Join command: $JOIN_CMD"

# Step 3: SSH to worker node
ssh cka3962-node1

# Step 4: Update package repositories
sudo apt-get update

# Step 5: Find matching package version
apt-cache madison kubeadm | grep ${KUBE_VERSION}

# Step 6: Upgrade kubeadm (adjust version as needed)
PACKAGE_VERSION="1.29.0-1.1" # Example, use actual version
sudo apt-mark unhold kubeadm
sudo apt-get install -y kubeadm=${PACKAGE_VERSION}
sudo apt-mark hold kubeadm

# Step 7: Upgrade kubelet and kubectl
sudo apt-mark unhold kubelet kubectl
sudo apt-get install -y kubelet=${PACKAGE_VERSION} kubectl=${PACKAGE_VERSION}
sudo apt-mark hold kubelet kubectl

# Step 8: Restart kubelet
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# Step 9: Join cluster (use actual command from step 2)
sudo kubeadm join <control-plane-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:

# Exit back to control plane
exit

# Step 10: Verify node joined
kubectl get nodes
kubectl wait --for=condition=Ready node/cka3962-node1 --timeout=300s
kubectl describe node cka3962-node1
```

Key Points

- Upgrade kubeadm, kubelet, and kubectl to match control plane version

- Use `kubeadm token create --print-join-command` to generate join command
 - Restart kubelet after upgrade
 - Verify node is Ready before considering task complete
-

Question 9: ServiceAccount API Access

Task: Create pod with ServiceAccount and query Kubernetes API on `cka9412`

Solution

```
ssh cka9412

mkdir -p /opt/course/9

# Step 1: Create Pod with ServiceAccount
kubectl run api-contact \
  --image=nginx:1-alpine \
  --namespace=project-swan \
  --overrides='
{
  "spec": {
    "serviceAccountName": "secret-reader",
    "containers": [{
      "name": "api-contact",
      "image": "nginx:1-alpine",
      "command": ["sleep", "3600"]
    }]
  }
}'

# Wait for Pod
kubectl wait --for=condition=ready pod/api-contact -n project-swan --timeout=60s

# Step 2: Execute API query and save result
kubectl exec -n project-swan api-contact -- sh -c '
# Install curl if not present
command -v curl >/dev/null 2>&1 || apk add --no-cache curl >/dev/null 2>&1

# Query Kubernetes API
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
CACERT=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt

curl -s --cacert $CACERT \
  -H "Authorization: Bearer $TOKEN" \
  https://kubernetes.default.svc/api/v1/secrets
' > /opt/course/9/result.json

# Verification
ls -lh /opt/course/9/result.json
head -10 /opt/course/9/result.json
```

Key Points

- Pod uses ServiceAccount `secret-reader`
- ServiceAccount token at `/var/run/secrets/kubernetes.io/serviceaccount/token`
- CA cert at `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`
- Query API: `https://kubernetes.default.svc/api/v1/secrets`

- Result saved to `/opt/course/9/result.json`

Question 10: ServiceAccount with RBAC

Task: Create ServiceAccount, Role, and RoleBinding in namespace `project-hamster` on `cka3962`

Solution

```
ssh cka3962

# Create ServiceAccount
kubectl create serviceaccount processor -n project-hamster

# Create Role with create permissions for secrets and configmaps
kubectl create role processor \
  --verb=create \
  --resource=secrets,configmaps \
  -n project-hamster

# Create RoleBinding
kubectl create rolebinding processor \
  --role=processor \
  --serviceaccount=project-hamster:processor \
  -n project-hamster

# Verification
kubectl get sa processor -n project-hamster
kubectl describe role processor -n project-hamster
kubectl describe rolebinding processor -n project-hamster

# Test permissions
kubectl auth can-i create secrets \
  --as=system:serviceaccount:project-hamster:processor \
  -n project-hamster

kubectl auth can-i create configmaps \
  --as=system:serviceaccount:project-hamster:processor \
  -n project-hamster
```

Key Points

- ServiceAccount: `processor`
- Role: `processor` with `create` verb on `secrets,configmaps`

- RoleBinding: `processor` linking Role to ServiceAccount
 - Permissions limited to namespace `project-hamster`
-

Question 11: DaemonSet on All Nodes

Task: Create DaemonSet that runs on all nodes including control plane on `cka2556`

Solution

```
ssh cka2556

# Create DaemonSet
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ds-important
  namespace: project-tiger
  labels:
    id: ds-important
    uuid: 18426a0b-5f59-4e10-923f-c0e078e82462
spec:
  selector:
    matchLabels:
      id: ds-important
      uuid: 18426a0b-5f59-4e10-923f-c0e078e82462
  template:
    metadata:
      labels:
        id: ds-important
        uuid: 18426a0b-5f59-4e10-923f-c0e078e82462
    spec:
      tolerations:
        - key: node-role.kubernetes.io/control-plane
          operator: Exists
          effect: NoSchedule
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
      containers:
        - name: httpd
          image: httpd:2-alpine
          resources:
            requests:
              cpu: 10m
              memory: 10Mi

EOF

# Verification
kubectl get daemonset ds-important -n project-tiger
kubectl get pods -n project-tiger -l id=ds-important -o wide
```

Key Points

- Name: `ds-important`
- Labels: `id=ds-important`, `uuid=18426a0b-5f59-4e10-923f-c0e078e82462`

- Image: `httpd:2-alpine`
 - Resources: 10m CPU, 10Mi memory
 - Tolerations for control plane nodes
-

Question 12: Deployment with Pod Anti-Affinity

Task: Create Deployment with anti-affinity ensuring one pod per node on `cka2556`

Solution

```
ssh cka2556

# Create Deployment
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-important
  namespace: project-tiger
  labels:
    id: very-important
spec:
  replicas: 3
  selector:
    matchLabels:
      id: very-important
  template:
    metadata:
      labels:
        id: very-important
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchLabels:
                  id: very-important
              topologyKey: kubernetes.io/hostname
      containers:
        - name: container1
          image: nginx:1-alpine
        - name: container2
          image: google/pause
EOF

# Verification
kubectl get deployment deploy-important -n project-tiger
kubectl get pods -n project-tiger -l id=very-important -o wide

# Check pod distribution
kubectl get pods -n project-tiger -l id=very-important -o wide --no-headers | awk '{print $7}'
```

Key Points

- 3 replicas with anti-affinity
- Only one pod per node (hostname topology)
- With 2 worker nodes: 2 pods Running, 1 Pending

- Two containers: nginx:1-alpine and google/pause
-

Question 13: Gateway API HTTPRoute

Task: Create HTTPRoute to replace Ingress with custom routing on `cka7968`

Solution

```
ssh cka7968
```

```
# Get Gateway name
```

```
GATEWAY_NAME=$(kubectl get gateway -n project-r500 -o jsonpath='{.items[0].metadata.name}')
```

```
# Create HTTPRoute
```

```
kubectl apply -f - <<EOF
```

```
apiVersion: gateway.networking.k8s.io/v1
```

```
kind: HTTPRoute
```

```
metadata:
```

```
  name: traffic-director
```

```
  namespace: project-r500
```

```
spec:
```

```
  parentRefs:
```

```
  - name: ${GATEWAY_NAME}
```

```
  hostnames:
```

```
  - "r500.gateway"
```

```
  rules:
```

```
    # Route /desktop
```

```
    - matches:
```

```
      - path:
```

```
        type: PathPrefix
```

```
        value: /desktop
```

```
    backendRefs:
```

```
    - name: desktop
```

```
      port: 80
```

```
    # Route /mobile
```

```
    - matches:
```

```
      - path:
```

```
        type: PathPrefix
```

```
        value: /mobile
```

```
    backendRefs:
```

```
    - name: mobile
```

```
      port: 80
```

```
    # Route /auto with User-Agent: mobile -> mobile service
```

```
    - matches:
```

```
      - path:
```

```
        type: PathPrefix
```

```
        value: /auto
```

```
      headers:
```

```
      - name: User-Agent
```

```
        value: mobile
```

```
        type: Exact
```

```
    backendRefs:
```

```
    - name: mobile
```

```
      port: 80
```

```
    # Route /auto (default) -> desktop service
```

```
    - matches:
```

```
      - path:
```

```
        type: PathPrefix
```

```
        value: /auto
```

```
    backendRefs:
```

```
- name: desktop
  port: 80
EOF

# Test routes
curl http://r500.gateway:30080/desktop
curl http://r500.gateway:30080/mobile
curl http://r500.gateway:30080/auto -H "User-Agent: mobile"
curl http://r500.gateway:30080/auto
```

Key Points

- HTTPRoute name: `traffic-director`
- Replicates Ingress routes for `/desktop` and `/mobile`
- New `/auto` path with User-Agent based routing
- Rules evaluated top to bottom

Question 14: Certificate Management

Task: Check certificate expiration and create renewal script on `cka9412`

Solution

```
ssh cka9412

mkdir -p /opt/course/14

# Step 1: Check certificate expiration with openssl
sudo openssl x509 -in /etc/kubernetes/pki/apiserver.crt -noout -enddate | cut -d= -f2 | sudo

# Step 2: Verify with kubeadm
sudo kubeadm certs check-expiration

# Step 3: Create renewal script
sudo tee /opt/course/14/kubeadm-renew-certs.sh > /dev/null <<'EOF'
#!/bin/bash
kubeadm certs renew apiserver
EOF

sudo chmod +x /opt/course/14/kubeadm-renew-certs.sh

# Verification
cat /opt/course/14/expiration
cat /opt/course/14/kubeadm-renew-certs.sh
sudo kubeadm certs check-expiration | grep apiserver
```

Key Points

- Certificate location: `/etc/kubernetes/pki/apiserver.crt`
- Expiration saved to `/opt/course/14/expiration`
- Renewal command: `kubeadm certs renew apiserver`
- Both openssl and kubeadm should show same expiration date

Question 15: NetworkPolicy for Security

Task: Create NetworkPolicy to restrict backend pod connections on `cka7968`

Solution

```
ssh cka7968
```

```
# Create NetworkPolicy
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: np-backend
  namespace: project-snake
```

```
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Egress
  egress:
  # Allow DNS
  - to:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          k8s-app: kube-dns
      ports:
        - protocol: UDP
          port: 53
  # Allow db1:1111
  - to:
    - podSelector:
        matchLabels:
          app: db1
      ports:
        - protocol: TCP
          port: 1111
  # Allow db2:2222
  - to:
    - podSelector:
        matchLabels:
          app: db2
      ports:
        - protocol: TCP
          port: 2222
```

```
EOF
```

```
# Test connectivity
```

```
BACKEND_POD=$(kubectl get pods -n project-snake -l app=backend -o jsonpath='{.items[0].metadata.name}')
DB1_IP=$(kubectl get pods -n project-snake -l app=db1 -o jsonpath='{.items[0].status.podIP}')
DB2_IP=$(kubectl get pods -n project-snake -l app=db2 -o jsonpath='{.items[0].status.podIP}')
```

```
kubectl -n project-snake exec $BACKEND_POD -- curl -m 2 $DB1_IP:1111
```

```
kubectl -n project-snake exec $BACKEND_POD -- curl -m 2 $DB2_IP:2222
```

Key Points

- NetworkPolicy name: `np-backend`
- Applies to pods with `app: backend` label
- Egress policy (controls outbound traffic)
- Allowed: db1:1111, db2:2222, DNS
- Blocked: All other connections (e.g., vault:3333)

Question 16: CoreDNS Custom Domain

Task: Update CoreDNS to support custom domain on `cka5774`

Solution

```
ssh cka5774

mkdir -p /opt/course/16

# Step 1: Backup CoreDNS configuration
kubectl get configmap coredns -n kube-system -o yaml > /opt/course/16/coredns_backup.yaml

# Step 2: Update CoreDNS configuration
kubectl get configmap coredns -n kube-system -o yaml > /tmp/coredns-update.yaml

# Add custom-domain to kubernetes line
sed -i '/kubernetes cluster\.local/s/cluster\.local/cluster\.local custom-domain/' /tmp/coredns-update.yaml

# Apply updated configuration
kubectl apply -f /tmp/coredns-update.yaml

# Step 3: Restart CoreDNS pods
kubectl delete pod -n kube-system -l k8s-app=kube-dns

# Wait for pods to be ready
kubectl wait --for=condition=ready pod -n kube-system -l k8s-app=kube-dns --timeout=60s

# Step 4: Test DNS resolution
kubectl run test-dns-1 --image=busybox:1 --rm -i --restart=Never -- nslookup kubernetes.default.svc

kubectl run test-dns-2 --image=busybox:1 --rm -i --restart=Never -- nslookup kubernetes.default.svc

# Verification
kubectl get configmap coredns -n kube-system -o jsonpath='{.data.Corefile}' | grep kubernetes
```

Key Configuration Change

```
# Before:
kubernetes cluster.local in-addr.arpa ip6.arpa {

# After:
kubernetes cluster.local custom-domain in-addr.arpa ip6.arpa {
```

Key Points

- Backup at `/opt/course/16/coredns_backup.yaml`
- Add `custom-domain` after `cluster.local` in Corefile
- Both `*.cluster.local` and `*.custom-domain` should resolve
- Restart CoreDNS pods to apply changes

Question 17: Container Runtime Investigation

Task: Find container using `crictl` and extract information on `cka2556`

Solution

```
ssh cka2556

# Step 1: Create Pod
kubectl run tigers-reunite \
  --image=httpd:2-alpine \
  --namespace=project-tiger \
  --labels=pod=container,container=pod

kubectl wait --for=condition=ready pod/tigers-reunite -n project-tiger --timeout=60s

# Step 2: Find node
NODE=$(kubectl get pod tigers-reunite -n project-tiger -o jsonpath='{.spec.nodeName}')
echo "Pod is on node: $NODE"

# Step 3: SSH to node
ssh $NODE << 'ENDSSH'

# Find container
CONTAINER_ID=$(sudo crictl ps | grep tigers-reunite | grep httpd | awk '{print $1}')
echo "Container ID: $CONTAINER_ID"

# Get runtime type
RUNTIME_TYPE=$(sudo crictl inspect $CONTAINER_ID | jq -r '.info.runtimeType')
echo "Runtime Type: $RUNTIME_TYPE"

# Create directory
sudo mkdir -p /opt/course/17

# Write container ID and runtime type
echo "$CONTAINER_ID $RUNTIME_TYPE" | sudo tee /opt/course/17/pod-container.txt

# Get logs
sudo crictl logs $CONTAINER_ID | sudo tee /opt/course/17/pod-container.log

# Verify
echo "=== Results ==="
cat /opt/course/17/pod-container.txt
head -10 /opt/course/17/pod-container.log

ENDSSH
```

Key crictl Commands

- `crictl ps` - List containers
- `crictl pods` - List pods
- `crictl inspect <ID>` - Inspect container
- `crictl logs <ID>` - Get container logs

- `crictl exec -it <ID> sh` - Execute in container

Key Points

- Pod: `tigers-reunite` with labels `pod=container` and `container=pod`
 - File: `/opt/course/17/pod-container.txt` contains ID and runtimeType
 - File: `/opt/course/17/pod-container.log` contains container logs
 - Tool: `crictl` (CRI-compatible CLI)
-

Summary of Key Concepts

Resource Management

- **QoS Classes:** BestEffort → Burstable → Guaranteed (eviction order)
- **Resource requests/limits:** Define container resource requirements
- **HPA:** Horizontal Pod Autoscaler for automatic scaling

Networking

- **NetworkPolicy:** Control ingress/egress traffic
- **CoreDNS:** Cluster DNS service configuration
- **Gateway API:** Modern alternative to Ingress

Storage

- **PV/PVC:** Persistent volume and claims
- **StorageClass:** Dynamic provisioning
- **Volume types:** hostPath, NFS, cloud providers

Security

- **RBAC:** Role-Based Access Control
- **ServiceAccount:** Pod identity for API access
- **NetworkPolicy:** Pod-to-pod communication control
- **Certificate management:** kubeadm certs commands

Scheduling

- **Affinity/Anti-affinity**: Control pod placement
- **Taints/Tolerations**: Node restrictions
- **DaemonSet**: One pod per node
- **topologyKey**: Define scheduling domains

Tools

- **kubectl**: Primary CLI tool
 - **kubeadm**: Cluster management
 - **crictl**: Container runtime CLI
 - **Helm**: Package manager
 - **Kustomize**: Configuration management
-

Best Practices

1. **Always backup** before making changes
 2. **Verify changes** after applying
 3. **Use labels** for resource organization
 4. **Test permissions** with `kubectl auth can-i`
 5. **Check logs** when troubleshooting
 6. **Use dry-run** to preview changes
 7. **Document** custom configurations
 8. **Monitor** resource usage
 9. **Follow security** best practices
 10. **Keep cluster** components updated
-

Quick Reference Commands

```
# Resource Management
kubectl top nodes
kubectl top pods --all-namespaces --containers
kubectl scale deployment <name> --replicas=N

# RBAC
kubectl create serviceaccount <name>
kubectl create role <name> --verb=create --resource=secrets
kubectl create rolebinding <name> --role=<role> --serviceaccount=<ns>:<sa>
kubectl auth can-i <verb> <resource> --as=<user>

# Networking
kubectl get networkpolicies
kubectl describe networkpolicy <name>

# Certificates
sudo kubeadm certs check-expiration
sudo kubeadm certs renew <cert-name>
sudo openssl x509 -in <cert> -noout -enddate

# Container Runtime
sudo crictl ps
sudo crictl logs <container-id>
sudo crictl inspect <container-id>

# Debugging
kubectl describe <resource> <name>
kubectl logs <pod>
kubectl exec -it <pod> -- sh
kubectl get events --sort-by=.metadata.creationTimestamp
```

End of CKA Practice Solutions