Kubernetes CKA Exam

Example questions for CKA Exam

A step-by-step guide with practice lab examples to level up your Kubernetes skills

by Marcin Kujawski

Kubernetes

CKA Mock Exam

Preparation Guide for CKA Exam

Marcin Kujawski

Table of Contents

Prerequisites

All questions are assumed that you have:

1) Cluster configured with 3 nodes:
   - Control plane node named “master”
   - Worker nodes named “worker1” and “worker2”
2) Cluster was built with “kubeadm” tool.

CKA Tips & Tricks

I have prepared a separate GitHub repository to inform you what is important on K8s CKA exam. Please read it carefully and use that knowledge to pass the exam easily.

CKA Tips & Tricks

Question 1: Schedule Pod on Controlplane Nodes

Create a single Pod of image `nginx` in Namespace `default`. The Pod should be named `pod1` and the container should be named `pod1-container`. This Pod should only be scheduled on controlplane nodes. Do not add new labels to any nodes.

Answer:

1. Log into the control plane node.
2. Find the cluster nodes and their taints as well as labels:

```
$ kubectl get node
$ kubectl describe node master| grep Taint -A1
$ kubectl get node master --show-labels
```

3. Next we create the Pod template:

```
$ kubectl run pod1 --image=nginx -n default --dry-run=client
-o yaml > q1.yaml
$ vi q1.yaml
```

4. Perform the necessary changes manually. Use the Kubernetes docs and search for example for tolerations and nodeSelector to find examples:

```
# q1.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: pod1-container                                  # change
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  tolerations:                                            # add
  - effect: NoSchedule                                    # add
    key: node-role.kubernetes.io/control-plane            # add
  nodeSelector:                                           # add
    node-role.kubernetes.io/control-plane: ""             # add
status: {}
```

Important here to add the toleration for running on controlplane nodes, but also the nodeSelector to make sure it only runs on controlplane nodes. If we only specify a toleration the Pod can be scheduled on controlplane or worker nodes.

5. Create the Pod:

```
$ kubectl apply -f q1.yaml
```

6. Validate if the pod is scheduled properly:

```
$ kubectl get pod pod1 -o wide
```

Question 2: Scale down StatefulSet

There are five Pods named `psql-*` in Namespace `question2`. You are asked to scale the Pods down to one replica to save resources on cluster.

Answer:

1. Log into the control plane node.
2. Find the Pods running:

```
$ kubectl -n question2 get pod
$ kubectl describe node master
```

3. From their name it looks like these are managed by a StatefulSet. But if we're not sure we could also check for the most common resources which manage Pods:

```
$ kubectl -n question2 get sts,ds,deploy
```

4. Confirmed, we have to work with a StatefulSet.
5. To fulfil the task we simply run:

```
$ kubectl -n question2 scale psql --replicas 1
```

Question 3: Server Certificates

Check how long the `kube-apiserver` server certificate is valid on `master`. Do this with openssl or cfssl. Write the exipiration date into `/opt/exam/q3/expiration`. Also run the correct `kubeadm` command to list the expiration dates and confirm both methods show the same date. Write the correct `kubeadm` command that would renew the apiserver server certificate into `/opt/exam/q3/kubeadm-renew-certs.sh`.

Answer:

1. Log into the control plane node.
2. Find the cluster certificate:

```
$ find /etc/kubernetes/pki | grep -i apiserver
```

3. Next, use openssl command to find out the expiration date:

```
$ openssl x509 -noout -text -in
/etc/kubernetes/pki/apiserver.crt | grep Validity -A2
```

4. There we have it, so we write it in the required location:

```
$ vi /opt/exam/q3/expiration
```

5. Copy output from openssl command and paste it. Save and exit with the file:

```
:wq
```

6. And we use the feature from kubeadm to get the expiration too:

```
$ kubeadm certs check-expiration | grep apiserver
```

7. Looking good. And finally, we write the command that would renew all certificates into the requested location:

```
$ echo "kubeadm certs renew apiserver" >
/opt/exam/q3/kubeadm-renew-certs.sh
```

Question 4: Storage, PV, PVC, Pod volume

Create a new PersistentVolume named `db-pv`. It should have a capacity of `2Gi`, accessMode `ReadWriteOnce`, hostPath `/Volumes/Data` and no storageClassName defined. Next create a new PersistentVolumeClaim in Namespace `postgres-db` named `db-pvc`. It should request `2Gi` storage, accessMode `ReadWriteOnce` and should not define a storageClassName. The PVC should bound to the PV correctly. Finally create a new Deployment `postgres-dep` in Namespace `postgres-db` which mounts that volume at `/tmp/pg-data`. The Pods of that Deployment should be of image `postgres:12`.

Answer:

1. Log into the control plane node.
2. Find an example of PersistenVolme manifest file from official Kubernetes documentation and alter it:

```
# db-pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: db-pv                                    # change
  namespace: postgres-db                         # change
spec:
  capacity:
    storage: 2Gi                                 # change
```

```
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/Volumes/Data"                                  # change
```

3. Create the PV:

```
$ kubectl apply -f db-pv.yaml
```

4. The same we need to do with PersistentVolumeClaim. Navigate to proper documentation page and use example with adjustments:

```
# db-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: db-pvc                                             # change
  namespace: postgres-db                                   # change
spec:
  resources:
    requests:
      storage: 2Gi                                         # change
  accessModes:
    - ReadWriteOnce
```

5. Create the PVC:

```
$ kubectl apply -f db-pvc.yaml
```

6. Let's check the status of both. They should have status "Bound":

```
$ kubectl -n postgres-db get pv,pvc
```

7. Next, we create a Deployment and mount that volume inside:

```
$ kubectl -n postgres-db create deploy postgres-dep
--image=postgres:12 --dry-run=client -o yaml > postgres-
dep.yaml
```

8. Alter the YAML manifest file to mount the volume:

```
# postgres-dep.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  name: postgres-dep
  namespace: postgres-db
  labels:
    app: postgres-dep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-dep
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
```

```
        app: postgres-dep
    spec:
      volumes:                                          # add
      - name: data                                      # add
        persistentVolumeClaim:                          # add
          claimName: db-pvc                             # add
      containers:
      - image: postgres:12
        name: postgres
        resources: {}
        volumeMounts:                                   # add
        - name: data                                    # add
          mountPath: /tmp/pg-data                       # add
```

9. Create the Deployment:

```
$ kubectl apply -f postgres-dep.yaml
```

10. Check if the PV is functioning properly and mounted correctly:

```
$ kubectl -n postgres-db describe pod postgres-dep-
64579b656c-7dgqt | grep -A2 Mounts:
```

11. You should see the output similar to that:

```
    Mounts:
      /tmp/pg-data from data (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from
kube-api-access-5m59r (ro)
```

Question 5: Node and Pod Resource Usage

The `metrics-server` has been installed in the cluster. Your college would like to know the `kubectl` commands to:

- show Nodes resource usage
- show Pods and their containers resource usage

Please write the commands into `/opt/exam/q5/node.sh` and `/opt/exam/q5/pod.sh`

Answer:

1. Log into the control plane node.
2. The command we need to use here is top:

```
$ kubectl top -h
```

3. We see that the metrics server provides information about resource usage:

```
$ kubectl top node
```

4. We create the first file:

```
$ echo "kubectl top node" > /opt/exam/q5/node.sh
```

5. For the second file we might need to check the docs again:

```
$ kubectl top pod -h
```

6. With this we can finish this task:

```
$ echo "kubectl top pod --containers=true" >
/opt/exam/q5/pod.sh
```

Note:

If command `kubectl` top will not work it means that you have to install metric-server upfront.

Question 6: RBAC ServiceAccount Role RoleBinding

Create a new `ServiceAccount` named `config-admin` in namespace `web`. Create a `Role` and `RoleBinding`, both named `config-admin` as well. These should allow the new SA to only create `Secrets` and `ConfigMaps` in that Namespace. Validate the permissions with `auth can-i` command.

Answer:

1. Log into the control plane node.
2. Let's first create the ServiceAccount:

```
$ kubectl -n web create sa config-admin
```

3. Next, let's create the Role:

```
$ kubectl -n web create role config-admin --verb=create
--resource=secret --resource=configmap
```

4. And finally let's bind the Role with ServiceAccount and create the RoleBinding:

```
$ kubectl -n web create rolebinding config-admin --role
config-admin --serviceaccount=web:config-admin
```

5. Last step is to test if we create all above correctly. To test our RBAC setup we can use kubectl auth can-i command:

```
$ kubectl -n web auth can-i create secret --as
system:serviceaccount:web:config-admin
yes
$ kubectl -n web auth can-i create configmap --as
system:serviceaccount:web:config-admin
yes
$ kubectl -n web auth can-i create pod --as
system:serviceaccount:web:config-admin
no
$ kubectl -n web auth can-i delete secret --as
system:serviceaccount:web:config-admin
no
$ kubectl -n web auth can-i get configmap --as
system:serviceaccount:web:config-admin
no
```

Question 7: DaemonSet on all Nodes

Use Namespace `web` for the following. Create a `DaemonSet` named `ds-web-metric` with image `httpd:2.4-alpine` and labels `env=prod` and `uuid=18426a0b-5f59-4e10-923f-c0e078e82462`. Name of the Pod should be `metric-pod`. The Pods it creates should request 10 millicore cpu and 10 mebibyte of memory. The Pods of that DaemonSet should run on all nodes, also control planes.

Answer:

1. Log into the control plane node.
2. Navigate to the official Kubernetes documentation and find example code for DaemonSet. Copy it and paste into manifest. Adjust the code to the task's requirements:

```
# ds-web-metric.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ds-web-metric                                  # change
  namespace: web                                       # change
  labels:                                              # add
    env: prod                                          # add
    uuid: 18426a0b-5f59-4e10-923f-c0e078e82462         # add
spec:
  selector:
    matchLabels:
      env: prod                                        # add
```

```
      uuid: 18426a0b-5f59-4e10-923f-c0e078e82462    # add
  template:
    metadata:
      labels:
        env: prod                                   # add
        uuid: 18426a0b-5f59-4e10-923f-c0e078e82462  # add
    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: metric-pod                            # change
        image: httpd:2.4-alpine                     # change
        resources:
          requests:
            cpu: 10m                                # change
            memory: 10Mi                            # change
```

3. Create the DaemonSet:

```
$ kubectl apply -f ds-web-metric.yaml
```

4. Validate that Pods are running on each node including master node. You should see all 3 Pods running within all cluster nodes:

```
$ kubectl -n web get pods -l env=prod -o wide
```

Question 8: Cluster Event Logging

Write a command into `/opt/exam/q8/cluster_events.sh` which shows the latest events in the whole cluster, ordered by time (`metadata.creationTimestamp`). Use `kubectl` for it. Now delete the `kube-proxy` Pod running on node `worker1` and write the events this caused into `/opt/exam/q8/pod_kill.log`. Finally kill the containerd container of the `kube-proxy` Pod on node `worker1` and write the events into `/opt/course/q8/container_kill.log`.

Answer:

1. Log into the control plane node.
2. Get the events for whole cluster and sort them as requested by creationTimestamp. Remember to use -A flag to capture all namespaces:

```
$ kubectl get events -A --sort-by=.metadata.creationTimestamp
```

3. Command should return the valid response. Let's save it to the script file:

```
$ echo "kubectl get events -A --sort-
by=.metadata.creationTimestamp" >
/opt/exam/q8/cluster_events.sh
```

4. Now, let's delete the kube-proxy Pod. To do that, we need to first check what is the Pod name and right after delete it. Use grep to filter the proxy Pod running only on worker1 node:

```
$ kubectl -n kube-system get pod -o wide | grep proxy | grep
worker1
```

5. Get the name of the Pod and simply delete it:

```
$ kubectl -n kube-system delete pod kube-proxy-qlsw4
```

6. Check the events now:

```
$ sh /opt/exam/q8/cluster_events.sh
```

7. Write the events the killing caused into file:

```
$ sh /opt/exam/q8/cluster_events.sh | tail -n7 >
/opt/exam/q8/pod_kill.log
```

8. Finally, let's kill the container belonging to the container of the kube-proxy Pod. In order to do this login directly via SSH to worker1 node and list the containers running:

```
$ ssh 'student@worker1'
$ sudo crictl ps | grep kube-proxy
```

9. Note down the container ID and delete it:

```
$ sudo crictl rm -f 28d7dbc9c9a68
```

10. You should see the information that container was deleted. However, let's validate if it is really gone:

```
$ sudo crictl ps | grep kube-proxy
```

11. The killed container (with ID 28d7dbc9c9a68) does not exist anymore, but also noticed that a new container (with new container ID) was directly created. Thanks Kubernetes!
12. Last step is to gather events again, select the lines that corresponds to the last deletion and write them into the second file:

```
$ vi /opt/exam/q8/cluster_events.sh
```

Comparing the events, you have seen when deleted the whole Pod there were more things to be done, hence more events. For example was the DaemonSet in the game to re-create the missing Pod. When you manually killed the main container of the Pod, the Pod would still exist but only its container needed to be re-created, hence less events.

Question 9: Network Policy

To secure your backend application create a NetworkPolicy called `backend-np` in Namespace `enigma`. It should allow the `backend-*` Pods only to:

- connect to `db1-*` Pods on port 1111
- connect to `db2-*` Pods on port 2222

Use the `app` label of Pods in your policy. After implementation, connections from `backend-*` Pods to `vault-*` Pods on port 3333 should for example no longer work.

Prerequisites:

Before start doing the lab, login to master node and execute the following:

```
$ ssh 'student@master'
$ wget -O - https://raw.githubusercontent.com/mariano-italiano/cka-prep/master/q9.sh | bash
```

Answer:

1. Log into the control plane node.
2. First we look at the existing Pods and their labels:

```
$ kubectl -n enigma get pod
$ kubectl -n enigma get pod -L app
```

3. We need to test the current connection situation and see nothing is restricted. All tests are done from backend Pod:

```
$ kubectl -n enigma get pod -o wide
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.5:1111
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.8:2222
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.9:3333
```

4. All above connections should be allowed and working.
5. Now we create the NetworkPolicy by copying and changing an example from the K8s docs:

```
# backend-np.yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: backend-np                                    # change
  namespace: enigma                                   # change
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Egress                  # policy is only about Egress
  egress:
    -                         # first rule
      to:                         # first condition "to"
      - podSelector:
          matchLabels:
            app: db1
      ports:                      # second condition "port"
      - protocol: TCP
        port: 1111
    -                         # second rule
      to:                         # first condition "to"
```

```
    - podSelector:
        matchLabels:
          app: db2
    ports:                    # second condition "port"
    - protocol: TCP
      port: 2222
```

6. The NetworkPolicy above has two rules with two conditions each, it can be read as:

```
allow outgoing traffic if:
  (destination pod has label app=db1 AND port is 1111)
  OR
  (destination pod has label app=db2 AND port is 2222)
```

7. Create the NetworkPolicy:

```
$ kubectl apply -f backend-np.yaml
```

8. Now let's test again if all is working as expected. We can use exactly same commands as before:

```
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.5:1111
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.8:2222
$ kubectl -n enigma exec backend-pod -- curl -s 10.0.1.9:3333
```

9. As a result, first two connections should work, but last not.
10. Great, looking more secure. Task done.

Question 10: Fix Worker Node

There seems to be an issue with the `kubelet` not running on one worker node. Identify the problematic node and fix it. Please confirm that cluster has node available in Ready state afterwards. You should be able to schedule a Pod on that node. Write the reason of the issue into `/opt/exam/q10/kubelet-issue`.

Prerequisites:

Before start doing the lab, login to worker1 node and execute the following:

```
$ ssh 'student@worker1'
$ wget -O - https://raw.githubusercontent.com/mariano-italiano/cka-prep/master/q10.sh | bash
```

Answer:

The procedure on tasks like these should be to check if the kubelet is running, if not start it, then check its logs and correct errors if there are some.

Always helpful to check if other clusters already have some of the components defined and running, so you can copy and use existing config files. Though in this case it might not need to be necessary.

1. Log into the control plane node.
2. Find the cluster node that has a problem or is NotReady:

```
$ kubectl get nodes
```

3. Notice which worker node is faulty. SSH to this node and check if the kubelet is running:

```
$ ssh worker1
$ systemctl status kubelet
```

4. You should notice that the service is not running. Let's try to start it:

```
$ systemctl start kubelet
```

5. Unfortunately, service is not starting up. Take a closer look on service configuration file located at /etc/systemd/system/kubelet.service.d/10-kubeadm.conf. We see it's trying to execute /usr/local/bin/kubelet with some parameters defined in its service config file. A good way to find errors and see the extended logging is to use journalctl tool or directly check system logs i.e. messages/syslog files.

```
$ journalctl -u kubelet
$ less /var/log/messages
$ less /var/log/syslog
```

6. You should see now what the problem is:

```
kubelet.service: Failed to locate executable
/usr/local/bin/kubelet: No such file or directory
```

7. We need to find correct path to kubelet binary:

```
$ which kubelet
/usr/bin/kubelet
```

8. Correct the binary path (change /usr/local/bin/kubelet to /usr/bin/kubelet) to the kubelet configuration file /etc/systemd/system/kubelet.service.d/10-kubeadm.conf:

```
$ sudo vi
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

9. Once corrected, reload systemctl daemon and start service again:

```
$ systemctl daemon-reload
$ systemctl restart kubelet
```

10. Let's verify the status of the kubelet service now:

```
$ systemctl status kubelet
```

11. The node should become available for the api server, give it a bit of time though. After a while final check for node status:

```
$ kubectl get nodes
```

12. Finally, write the reason into the file:

```
$ echo  "wrong path to kubelet binary specified in service
config" > /opt/exam/q10/kubelet-issue
```