DUMPSBOSS.

Certified Kubernetes Administrator (CKA) Program

Linux Foundation CKA

Version Demo

Total Demo Questions: 18

Total Premium Questions: 186

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Cluster Architecture, Installation & Configuration	21
Topic 2, Workloads & Scheduling	106
Topic 3, Services & Networking	20
Topic 4, Storage	22
Topic 5, Troubleshooting	16
Topic 6, Mix Questions	1
Total	186

QUESTION NO: 1 - (SIMULATION)

List all the pods sorted by name

ANSWER: See the explanation for the answer

Explanation:

List Pods sorted alphabetically by their metadata name:

```
kubectl get pods --sort-by=.metadata.name
```

If Pods across every namespace are required, add `-A`:

```
kubectl get pods -A --sort-by=.metadata.name
```

QUESTION NO: 2 - (SIMULATION)

List “nginx-dev” and “nginx-prod” pod and delete those pods

ANSWER: See the explanation for the answer

Explanation:

List the two pods, then delete both by name:

```
kubectl get pods nginx-dev nginx-prod
kubectl delete pod nginx-dev nginx-prod
```

If the pods are in a namespace other than the current namespace, add the namespace flag to both commands, for example:

```
kubectl get pods -n <namespace> nginx-dev nginx-prod
kubectl delete pod -n <namespace> nginx-dev nginx-prod
```

QUESTION NO: 3 - (SIMULATION)

Create the nginx pod with version 1.17.4 and expose it on port 80

ANSWER: See the explanation for the answer

Explanation:

Create a standalone Pod named `nginx` using the `nginx:1.17.4` image and declare container port `80`:

```
kubectl run nginx --image=nginx:1.17.4 --restart=Never --port=80
```

`--restart=Never` ensures that `kubectl run` creates a Pod rather than a Deployment. The `--port=80` option sets the Pod container port metadata to 80; it does not create a Service.

QUESTION NO: 4 - (SIMULATION)

Create a pod that having 3 containers in it? (Multi-Container)

ANSWER: See the explanation for the answer

Explanation:

Create a Pod named `multi-container` with the three requested containers.

```
cat <<'EOF' > multi-container.yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container
spec:
  containers:
    - name: nginx-container
      image: nginx
    - name: redis-container
      image: redis
    - name: consul-container
      image: consul
EOF

kubectl apply -f multi-container.yaml
kubectl get pod multi-container
kubectl describe pod multi-container
```

The manifest defines a single Pod whose `spec.containers` list contains all three containers: `nginx-container`, `redis-container`, and `consul-container`.

QUESTION NO: 5 - (SIMULATION)

Create an nginx pod with containerPort 80 and it should check the pod running at endpoint / healthz on port 80 and verify and delete the pod.

ANSWER: See the explanation for the answer

Explanation:

Create the Pod with port `80` exposed and an HTTP `livenessProbe` against `/healthz` on that port:

```
cat <<'EOF' > nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    livenessProbe:
      httpGet:
        path: /healthz
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 10
EOF

kubectl apply -f nginx-pod.yaml
```

Verify that the probe is configured:

```
kubectl describe pod nginx
```

Check the `Liveness` section and Events in the output. The stock nginx image does not normally provide `/healthz`, so probe failures and container restarts may appear; this confirms Kubernetes is performing the requested health check.

Delete the Pod when finished:

```
kubectl delete pod nginx
```

QUESTION NO: 6 - (SIMULATION)

Create an nginx pod and list the pod with different levels of verbosity

ANSWER: See the explanation for the answer

Explanation:

Create the Pod imperatively, then use kubectl client verbosity flags while listing it:

```
kubectl run nginx --image=nginx --restart=Never --port=80

kubectl get pod nginx --v=7
kubectl get pod nginx --v=8
kubectl get pod nginx --v=9
```

`--restart=Never` ensures that `kubectl run` creates a standalone Pod rather than a Deployment. The `--v` flag changes kubectl's log verbosity; higher values provide more detailed request/debug output.

QUESTION NO: 7 - (SIMULATION)

You must connect to the correct host.

Failure to do so may result in a zero score.

[candidate@base] $ ssh Cka000049

Task

Perform the following tasks:

Create a new PriorityClass named high-priority for user-workloads with a value that is one less

than the highest existing user-defined priority class value.

Patch the existing Deployment busybox-logger running in the priority namespace to use the high-priority priority class.

ANSWER: See the explanation for the answer

Explanation:

Connect to the required host first, then determine the actual highest non-system PriorityClass value. Do not use a guessed value.

```
ssh Cka000049

max=$(kubectl get priorityclass -o jsonpath='{range
.items[*]}{.metadata.name}{"\t"}{.value}{"\n"}{end}' | awk '$1 !~ /^system-/' | sort -
k2,2nr | head -n1 | awk '{print $2}')
value=$((max - 1))
echo "Highest user-defined value: $max; high-priority value: $value"
```

Create the required PriorityClass with the computed value:

```
cat > high-priority.yaml <<EOF
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: ${value}
globalDefault: false
description: High priority class for user workloads
EOF

kubectl apply -f high-priority.yaml
```

Patch the Deployment pod template in namespace `priority`. Changing the pod template causes the Deployment to roll out Pods using the new class.

```
kubectl patch deployment busybox-logger -n priority \
  --type=merge \
  -p '{"spec":{"template":{"spec":{"priorityClassName":"high-priority"}}}}'
```

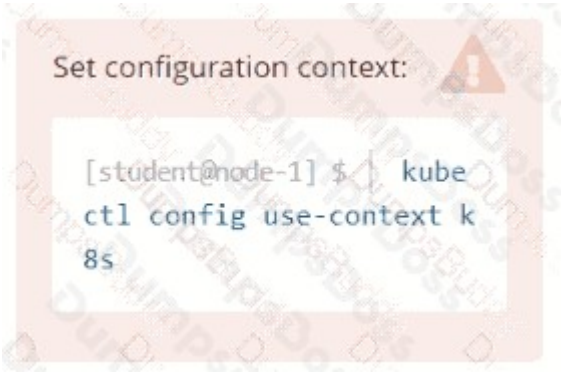
Verify both requirements:

```
kubectl get priorityclass high-priority
kubectl get deployment busybox-logger -n priority \
  -o jsonpath='{.spec.template.spec.priorityClassName}{"\n"}'
# expected: high-priority
```

The `awk '$1 !~ /^system-/'` filter excludes Kubernetes system PriorityClasses such as `system-cluster-critical` and `system-node-critical`; the new value is exactly one less than the greatest remaining user-defined class value.

QUESTION NO: 8 - (SIMULATION)

Score: 5%



Task

Monitor the logs of pod bar and:

• Extract log lines corresponding to error file-not-found

• Write them to /opt/KUTR00101/bar

ANSWER: See the explanation for the answer

Explanation:

Set the requested Kubernetes context, then stream the pod logs through `grep` and redirect matching lines to the required file:

```
kubectl config use-context k8s
kubectl logs bar | grep 'unable-to-access-website' > /opt/KUTR00101/bar
```

Optionally verify the result:

```
cat /opt/KUTR00101/bar
```

The `unable-to-access-website` log message is the error file-not-found entry to extract.

QUESTION NO: 9 - (SIMULATION)

Get the list of pods of webapp deployment

ANSWER: See the explanation for the answer

Explanation:

List the Pods belonging to the `webapp` Deployment by using its label selector:

```
kubectl get pods -l app=webapp
```

If the Deployment uses a different label, first inspect its labels and then use the matching selector:

```
kubectl get deployment webapp --show-labels
kubectl get pods -l <label-key>=<label-value>
```

QUESTION NO: 10 - (SIMULATION)

Perform the following tasks:

Add an init container to hungry-bear (which has been defined in spec file /opt/KUCC00108/pod-spec-KUCC00108.yaml)

The init container should create an empty file named/workdir/calm.txt

If /workdir/calm.txt is not detected, the pod should exit

Once the spec file has been updated with the init container definition, the pod should be created

ANSWER: See the explanation for the answer

Explanation:

Edit `/opt/KUCC00108/pod-spec-KUCC00108.yaml`. Add a shared `emptyDir` volume, mount it at `/workdir` in the existing application container, and add the init container below. The init container creates the file and explicitly fails if it cannot subsequently find it; a failed init container prevents the main container from starting.

```
apiVersion: v1
kind: Pod
metadata:
  name: hungry-bear
spec:
  volumes:
  - name: workdir
    emptyDir: {}

  initContainers:
  - name: create-calm-file
    image: busybox:1.36
    command:
    - sh
    - -c
    - |
      touch /workdir/calm.txt
      test -f /workdir/calm.txt || exit 1
```

```
    volumeMounts:
    - name: workdir
      mountPath: /workdir

  containers:
  - name: <keep-the-existing-container-name>
    # Keep the existing image and other settings from the supplied file.
    volumeMounts:
    - name: workdir
      mountPath: /workdir
```

Do not replace the existing container definition; merge the `volumeMounts` entry into it. Preserve any existing command, image, ports, and resource settings.

Create the pod after saving the file:

```
kubectl apply -f /opt/KUCC00108/pod-spec-KUCC00108.yaml
kubectl get pod hungry-bear
kubectl describe pod hungry-bear
```

The relevant behavior is that `test -f /workdir/calm.txt || exit 1` returns a non-zero status when the file is absent. Kubernetes then marks the init container as failed and does not start the regular container.

QUESTION NO: 11 - (SIMULATION)

Get list of all the pods showing name and namespace with a jsonpath expression.

ANSWER: See the explanation for the answer

Explanation:

Run:

```
kubectl get pods --all-namespaces -o=jsonpath='{range
.items[*]}{.metadata.name}{"\t"}{.metadata.namespace}{"\n"}{end}'
```

This retrieves Pods from every namespace and uses a JSONPath `range` to print each Pod's name and namespace on a separate line.

QUESTION NO: 12 - (SIMULATION)

Apply the autoscaling to this deployment with minimum 10 and

maximum 20 replicas and target CPU of 85% and verify hpa is

created and replicas are increased to 10 from 1

ANSWER: See the explanation for the answer

Explanation:

Create a HorizontalPodAutoscaler for the `webapp` Deployment with 10 minimum replicas, 20 maximum replicas, and an 85% CPU utilization target:

```
kubectl autoscale deployment webapp --min=10 --max=20 --cpu-percent=85
```

Verify that the HPA was created and that it has scaled the Deployment to the required minimum:

```
kubectl get hpa
kubectl get deployment webapp
kubectl get pods -l app=webapp
```

The HPA should show `MINPODS` as `10` and `MAXPODS` as `20`. The Deployment should scale from 1 replica to 10 replicas because the HPA enforces its configured minimum.

QUESTION NO: 13 - (SIMULATION)

Watch the job that runs 10 times one by one and verify 10 pods are created and delete those after it's completed

ANSWER: See the explanation for the answer

Explanation:

Watch the Job until it reports `COMPLETIONS 10/10`:

```
kubectl get job hello-job -w
```

In another terminal, watch the Pods created by that Job. Since the Job runs sequentially, Pods will be created one at a time; after completion there should be 10 Pods associated with the Job:

```
kubectl get pods -l job-name=hello-job -w
```

Optionally verify the final count:

```
kubectl get pods -l job-name=hello-job --no-headers | wc -l
```

After the Job has completed, delete it. This also deletes its Pods through the default foreground cascading deletion behavior:

```
kubectl delete job hello-job
```

QUESTION NO: 14 - (SIMULATION)

Create PersistentVolume named task-pv-volume with storage 10Gi, access modes ReadWriteMany, storageClassName manual, and volume at /mnt/data and Create a PersistentVolumeClaim of at least 3Gi storage and access mode ReadWriteOnce and verify

ANSWER: See the explanation for the answer

Explanation:

Create the PersistentVolume with the requested `ReadWriteMany` access mode and the claim with the requested `ReadWriteOnce` access mode.

```
cat <<'EOF' | kubectl apply -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  storageClassName: manual
  hostPath:
    path: /mnt/data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
```

```
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  resources:
    requests:
      storage: 3Gi
EOF

kubectl get pv task-pv-volume
kubectl get pvc task-pv-claim
```

The PV has `10Gi`, class `manual`, and path `/mnt/data`. The PVC requests `3Gi`, which satisfies “at least 3Gi.”

Verification note: the PVC will normally remain `Pending` with only this PV, because a claim requesting `ReadWriteOnce` requires a PV that explicitly lists `ReadWriteOnce`; a PV listing only `ReadWriteMany` is not an access-mode match. This is expected given the two different access modes explicitly required by the question.

QUESTION NO: 15 - (SIMULATION)

Create a ETCD backup of kubernetes cluster

Note : You don’t need to memorize command, refer - https://kubernetes.io/docs/tasks/administer-cluster/configureupgrade-etcd/ during exam See the solution below.

ANSWER: See the explanation for the answer

Explanation:

Create an etcd v3 snapshot from a control-plane node. For a kubeadm-managed local etcd, the required TLS files are normally under `/etc/kubernetes/pki/etcd/`.

```
ETCDCTL_API=3 etcdctl \
  --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 \
  snapshot save /root/etcd-snapshot.db
```

Verify that the snapshot was created and is valid:

```
ETCDCTL_API=3 etcdctl --write-out=table snapshot status /root/etcd-snapshot.db
```

If the cluster uses a different endpoint or certificate locations, obtain the exact values from the etcd static Pod manifest:

```
kubectl -n kube-system describe pod etcd-<control-plane-node>
# or inspect directly:
grep -E -- '--advertise-client-urls|--trusted-ca-file|--trusted-cafile|--cert-file|--
certfile|--key-file|--keyfile' /etc/kubernetes/manifests/etcd.yaml
```

Use the manifest's `--advertise-client-urls` value for `--endpoints`, its CA file for `--cacert`, server certificate for `--cert`, and server key for `--key`. Replace `/root/etcd-snapshot.db` with the backup filename/path required by the task.

Reference: Kubernetes: Operating etcd clusters for Kubernetes

QUESTION NO: 16 - (SIMULATION)

Create a Pod nginx and specify both CPU, memory requests and limits together and verify.

ANSWER: See the explanation for the answer

Explanation:

Create the Pod named `nginx` with CPU and memory requests and limits in its container specification:

```
cat <<'EOF' | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests:
        cpu: "500m"
        memory: "100Mi"
      limits:
        cpu: "1"
        memory: "200Mi"
EOF
```

Verify that the Pod is running and that the configured resource values are present:

```
kubectl get pod nginx
kubectl describe pod nginx
```

In the `kubectl describe pod nginx` output, under the nginx container, verify:

```
Requests: cpu: 500m, memory: 100Mi
Limits: cpu: 1, memory: 200Mi
```

Alternatively, display only the resource configuration:

```
kubectl get pod nginx -o jsonpath='{.spec.containers[0].resources}' ; echo
```

QUESTION NO: 17 - (SIMULATION)

Create a pod named kucc8 with a single app container for each of the

following images running inside (there may be between 1 and 4 images specified):

nginx + redis + memcached.

ANSWER: See the explanation for the answer

Explanation:

Create the `kucc8` Pod with three regular containers, one for each requested image:

```
cat <<'EOF' | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: kucc8
spec:
  containers:
  - name: nginx
    image: nginx
  - name: redis
    image: redis
  - name: memcached
    image: memcached
EOF
```

Verify it is running:

```
kubectl get pod kucc8
kubectl describe pod kucc8
```

The Pod is named `kucc8` and has exactly one container using each of the `nginx`, `redis`, and `memcached` images.

QUESTION NO: 18 - (SIMULATION)

Task Weight: 4%

Task

Schedule a Pod as follows:

• Name: kucc1

• App Containers: 2

• Container Name/Images:

o nginx

o consul

ANSWER: See the explanation for the answer

Explanation:

Create a Pod named `kucc1` with two application containers, named `nginx` and `consul`, using their respective images:

```
cat <<'EOF' > /tmp/kucc1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kucc1
spec:
  containers:
  - name: nginx
    image: nginx
  - name: consul
    image: consul
EOF
kubectl apply -f /tmp/kucc1.yaml
```

Verify that both containers are running:

```
kubectl get pod kucc1
kubectl describe pod kucc1
```

The required configuration is the two entries under `spec.containers`; do not create a Deployment, as the task specifically requests a Pod.