CKAD Practice Exam: Configuration Solution Guide

Intermediate 20m Bookmark

Check 1: Potential Solution

```
# Create a secret named app-secret in the yqe Namespace that stores key-
value pair of password=abnaoieb2073xsj
kubectl -n yqe create secret generic --from-
literal=password=abnaoieb2073xsj app-secret
# Create a Pod that consumes the app-secret Secret using a Volume that
mounts the Secret in the /etc/app directory. The Pod should be named app
and run a memcached container.
cat << EOF | kubectl apply -n yqe -f -
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: memcached
    volumeMounts:
    - name: secret
      mountPath: "/etc/app"
  volumes:
  - name: secret
    secret:
      secretName: app-secret
EOF
```

Commentary

There are multiple types of Secrets but the most common is generic/opaque Secrets. This is the most likely type of Secret to appear on the exam. The Secrets documentation has a section linking to the different ways to create Secrets. Using `kubectl` or manifest files are both reasonable choices but using l

he

The Secrets documentation also has an example showing how to consume a Secret using a Volume. Customizing the example is the easiest way to achieve the result. Although the task doesn't specify which Namespace the Pod should be in, it must be in the same Namespace as the secret since secrets are a namespaced resource.

In the solution above, the file contents and `kubectl apply` command are combined into one. This approach allows you to use. the exam notepad feature without any need to use command-line editors. You can then paste. the command into the terminal to execute it. You can also create a file using `vim` and then paste in the example to modify. It is highly recommended to practice with `vim` before taking the exam.

Content to review

- Mastering Kubernetes Pod Configuration: Config Maps and Secrets - Storing and Accessing Sensitive Information Using Kubernetes Secrets

Suggested documentation bookmark(s)

- Secrets

Check 2: Potential Solution

```
# Create a new ServiceAccount named secure-svc in the same namespace
where the deployment currently exists
kubectl create serviceaccount -n app secure-svc
# Create a file named patch.yaml and configure the replicas
serviceAccountName to be secure-svc
spec:
  template:
    spec:
      serviceAccountName: secure-svc
# Patch the existing Deployment
kubectl patch deployment -n app secapp --patch "$(cat patch.yaml)"
```

Comm

serviceaccount command. Deployments can be configured either before or after with custom service accounts for the replicas declared within. Deployments which have already been created can be patched to use a different ServiceAccount to the one that they were originally created with.

Content to review

- Configure Service Accounts for Pods
- Deployments

Suggested documentation bookmark(s)

- Update API Objects in Place Using kubectl patch

Check 3: Potential Solution

```
# Start by generating a Pod manifest template specifying as many
parameters possible when using kubectl run - redirect and save the output
into a file named pod.yaml
(kubectl run --image=bash -n dnn secpod -l env=prod --dry-run=client -o
yaml -- /usr/local/bin/bash -c 'sleep 3600') > pod.yaml
# pod.yaml contains
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
  env: prod
 name: secpod
 namespace: dnn
spec:
 containers:
  - args:
  - /usr/local/bin/bash
  - -c
  - sleep 3600
  image: bash
  name: secpod
  resources: {}
 dnsPolicy: ClusterFirst
 restartPolicy: Always
status: {}
# Us
conta
requ
pod.yaml manifest.
```

the

```
  labels:
   env: prod
  name: secpod
  namespace: dnn
spec:
  securityContext:
   fsGroup: 3000
  containers:
  - name: c1
    securityContext:
     runAsUser: 1000
    args:
    - /usr/local/bin/bash
    - -c
    - sleep 3600
    image: bash
    resources: {}
  - name: c2
    securityContext:
     runAsUser: 2000
    args:
    - /usr/local/bin/bash
    - -c
    - sleep 3600
    image: bash
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
# Using kubectl apply - create the pod resource within the cluster
kubectl apply -f pod.yaml
# Confirm that the pod has been successfully created
kubectl get pods -n dnn
# Confirm the the pod contains 2 containers c1 and c2 and that they have
been configured with the required security context
kubectl exec -it -n dnn secpod -c c1 -- id
uid=1000 gid=0(root) groups=3000
kubectl exec -it -n dnn secpod -c c2 -- id
uid=2000 gid=0(root) groups=3000
```

Commentary

A Security Context defines privilege and access control settings either at the Pod layer or at the Container layer. To specify security settings for a Pod, include the `securityContext` field in the Pod spec section and/or in the Container section. The `runAsUser` field specifies the user id that all processes running within a container will run as.

Conte

- kubectl Cheat Sheet (--dry-run)

Check 4: Potential Solution

```
# Start by generating a Pod manifest template specifying as many
parameters possible when using kubectl run - redirect and save the output
into a file named pod.yaml
(kubectl run web1 -n ca100 --image=nginx -l env=prod,type=processor --
port=80 --dry-run=client -o yaml) > pod.yaml
# pod.yaml contains
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    env: prod
    type: processor
  name: web1
  namespace: ca100
spec:
  containers:
  - image: nginx
    name: web1
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
# Using vim open up and modify the pod.yaml template with the memory
request of 100Mi and a memory limit at 200Mi:
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    env: prod
    type: processor
  name: web1
  namespace: ca100
spec:
  containers:
  - image: nginx
    name: web1
    ports:
    - containerPort: 80
```

```
# Using kubectl apply - create the pod resource within the cluster
kubectl apply -f pod.yaml
# Confirm that the pod has been successfully created
kubectl get pods -n ca100
# Confirm that the pod has the correct labels, port, limits, and requests
kubectl describe pod -n ca100 web1
```

Content to review

- Assign Memory Resources to Containers and Pods
- Pods

Suggested documentation bookmark(s)

- kubectl Cheat Sheet (run)

Check 5: Potential Solution

```
# Create a new ConfigMap using the kubectl create command with the
required key/value pairs
kubectl create configmap config1 -n ca200 --from-literal COLOUR=red --
from-literal SPEED=fast
# Next, generate a Pod manifest template specifying as many parameters
possible when using kubectl run - redirect and save the output into a
file named pod.yaml
(kubectl run -n ca200 redfastcar --image=busybox --dry-run=client -o yaml
-- /bin/sh -c "env | grep -E 'COLOUR|SPEED'; sleep 3600") > pod.yaml
# pod.yaml contains:
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   run: car1
 name: car1
 namespace: ca200
spec:
 containers:
 - args:
   - /bin/sh
   - -c
   - env | grep -E 'COLOUR|SPEED'; sleep 3600
   image: busybox
   name: redfastcar
   re
 dnsP
 res
status: {}
```

```
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   run: redfastcar
 name: redfastcar
 namespace: ca200
spec:
 containers:
 - args:
   - /bin/sh
   - -c
   - env | grep -E 'COLOUR|SPEED'; sleep 3600
   image: busybox
   name: redfastcar
   resources: {}
   envFrom:
   - configMapRef:
       name: config1
 dnsPolicy: ClusterFirst
 restartPolicy: Always
status: {}
# Using kubectl apply - create the pod resource within the cluster
kubectl apply -f pod.yaml
# Confirm that the pod has launched successfully with the ConfigMap
managed env vars
kubectl logs -n ca200 redfastcar
COLOUR=red
SPEED=fast
```

Commentary

ConfigMaps can be used to manage and maintain configuration key/value pairs which can then be later mounted into a Pod as environment variables.

Content to review

- ConfigMap as container environment variables
- Pods

Suggested documentation bookmark(s)

- kubectl Cheat Sheet (logs, run)

Mark as completed

Did you like this Resource? Report an issue

About the author

Cloud Academy

Instructor

Students	Labs	Courses	Learning paths
2,617	83	9	18

Digital skills are built at the intersection of knowledge, experience, and context. The fundamental building blocks of the training templates in our Library meet teams wherever they are along the cloud maturity curve, imparting the knowledge and experience needed to take them to the next level. Our training platform is intuitive and scalable. Most importantly, all training is easily customizable, which enables organizations to provide context and guidance for teams of any size. Teams leveraging Cloud Academy hit the ground running.

Covered topics

Deployment Compute DevOps Kubernetes

COMMUNITY

Join Discord Channel

HELP

Help Center