DUMPSBOSS.

Certified Kubernetes Application Developer (CKAD) Program

Linux Foundation CKAD

Version Demo

Total Demo Questions: 5

Total Premium Questions: 57

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Application Design and Build	9
Topic 2, Application Deployment	14
Topic 3, Application Observability and Maintenance	9
Topic 4, Application Environment, Configuration and Security	14
Topic 5, Services and Networking	10
Topic 6, Mix Questions	1
Total	57

QUESTION NO: 1 - (SIMULATION)

Context

As a Kubernetes application developer you will often find yourself needing to update a running application.

Task

Please complete the following:

• Update the app deployment in the kdpd00202 namespace with a maxSurge of 5% and a maxUnavailable of 2%

• Perform a rolling update of the web1 deployment, changing the lfccncf/ngmx image version to 1.13

• Roll back the app deployment to the previous version

ANSWER: See the explanation for the answer

Explanation:

First select the required cluster context:

```
kubectl config use-context k8s
```

Set the rolling-update parameters on the `app` Deployment in namespace `kdpd00202`:

```
kubectl -n kdpd00202 patch deployment app --type=merge -p
'{"spec":{"strategy":{"type":"RollingUpdate","rollingUpdate":{"maxSurge":"5%","maxUnavailab
le":"2%"}}}}'
```

Perform the rolling image update for `web1`. Set its container image to the requested version (use the actual container name from the Deployment if it is not named `nginx`):

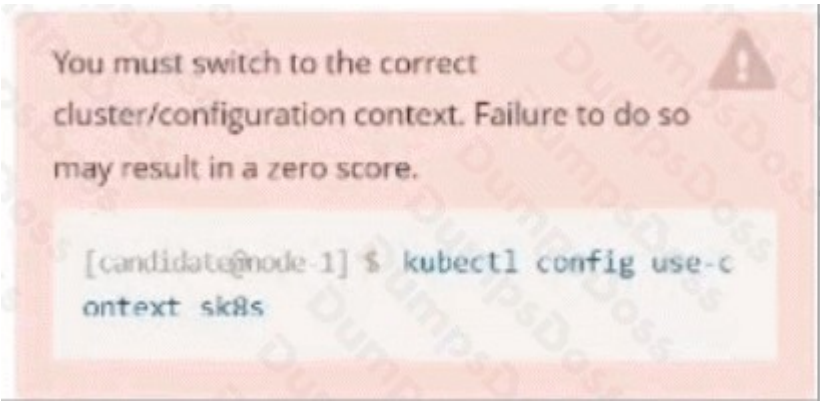
```
kubectl -n kdpd00202 set image deployment/web1 nginx=lfccncf/nginx:1.13
kubectl -n kdpd00202 rollout status deployment/web1
```

Finally, roll the `app` Deployment back to its immediately previous revision:

```
kubectl -n kdpd00202 rollout undo deployment/app
kubectl -n kdpd00202 rollout status deployment/app
```

If the image repository is displayed in the task environment with a different exact spelling (for example, `lfccncf/ngmx`), preserve that exact repository name and only set its tag to `:1.13`.

QUESTION NO: 2 - (SIMULATION)



Task:

Update the Deployment app-1 in the frontend namespace to use the existing ServiceAccount app.

ANSWER: See the explanation for the answer

Explanation:

Switch to the required cluster context, then set the Deployment pod template service account to the existing `app` ServiceAccount.

```
kubectl config use-context sk8s
kubectl -n frontend patch deployment app-1 \
  -p '{"spec":{"template":{"spec":{"serviceAccountName":"app"}}}}'
```

Verify the update:

```
kubectl -n frontend get deployment app-1 -o
jsonpath='{.spec.template.spec.serviceAccountName}{"\n"}'
```

The output must be `app`. Updating `spec.template.spec.serviceAccountName` changes the pod template and causes the Deployment to roll out replacement Pods using that ServiceAccount.

QUESTION NO: 3 - (SIMULATION)

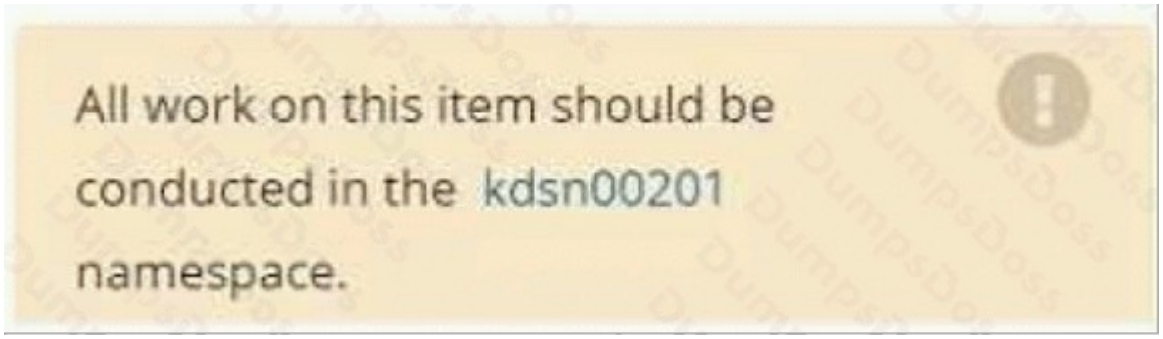
Set Configuration Context:

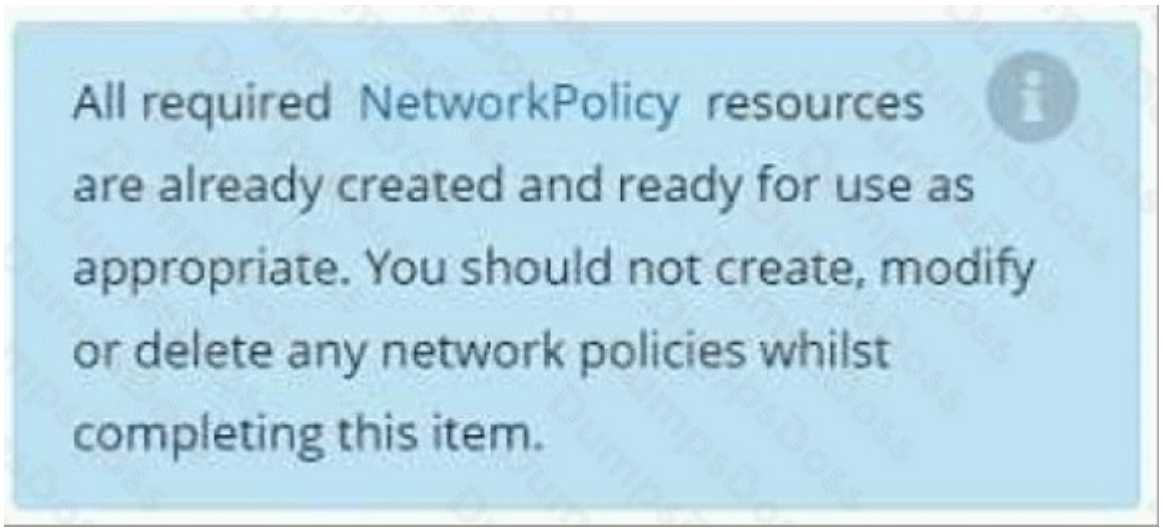
[student@node-1] $ | kubectl

Config use-context k8s

Task

You have rolled out a new pod to your infrastructure and now you need to allow it to communicate with the web and storage pods but nothing else. Given the running pod kdsn00201 -newpod edit it to use a network policy that will allow it to send and receive traffic only to and from the web and storage pods.





ANSWER: See the explanation for the answer

Explanation:

NetworkPolicies are applied by *pod labels*; they are not referenced from `spec` of a Pod. Do not create or change a NetworkPolicy. Instead, label `kdsn00201-newpod` so that it matches the selector of the pre-created policy which permits ingress and egress only for the `web` and `storage` pods.

```
kubectl config use-context k8s
kubectl -n kdsn00201 get networkpolicy -o yaml
kubectl -n kdsn00201 get pods --show-labels
```

From the NetworkPolicy output, locate the policy with both an `ingress` rule and an `egress` rule allowing the web and storage pod selectors. Copy every label under that policy's `spec.podSelector.matchLabels` onto `kdsn00201-newpod`.

For example, if the selected policy contains:

```
spec:
  podSelector:
    matchLabels:
      access: web-storage-only
```

apply that selector label to the running pod:

```
kubectl -n kdsn00201 label pod kdsn00201-newpod access=web-storage-only --overwrite
```

Alternatively, edit the Pod as requested:

```
kubectl -n kdsn00201 edit pod kdsn00201-newpod
```

and add the exact selector label(s) under `metadata.labels`, for example:

```
metadata:
  labels:
    access: web-storage-only
```

Verify that the Pod now matches the policy selector:

```
kubectl -n kdsn00201 get pod kdsn00201-newpod --show-labels
kubectl -n kdsn00201 describe networkpolicy <precreated-policy-name>
```

This makes the existing policy select the new pod, thereby restricting both incoming and outgoing traffic to the preconfigured web and storage pod peers.

QUESTION NO: 4 - (SIMULATION)

Context

Your application's namespace requires a specific service account to be used.

Task

Update the app-a deployment in the production namespace to run as the restrictedservice service account. The service account has already been created.

ANSWER: See the explanation for the answer

Explanation:

Set the required Kubernetes context, then patch the Deployment pod template to use the existing `restrictedservice` ServiceAccount:

```
kubectl config use-context k8s
kubectl -n production patch deployment app-a \
  -p '{"spec":{"template":{"spec":{"serviceAccountName":"restrictedservice"}}}}'
```

Verify that the Deployment template was updated and wait for its rollout if needed:

```
kubectl -n production get deployment app-a \
  -o jsonpath='{.spec.template.spec.serviceAccountName}{"\n"}'
kubectl -n production rollout status deployment/app-a
```

The expected ServiceAccount value is `restrictedservice`.

QUESTION NO: 5 - (SIMULATION)

Context

You have been tasked with scaling an existing deployment for availability, and creating a service to expose the deployment within your infrastructure.

Task

Start with the deployment named kdsn00101-deployment which has already been deployed to the namespace kdsn00101 . Edit it to:

• Add the func=webFrontEnd key/value label to the pod template metadata to identify the pod for the service definition

• Have 4 replicas

Next, create ana deploy in namespace kdsn00l01 a service that accomplishes the following:

• Exposes the service on TCP port 8080

• is mapped to me pods defined by the specification of kdsn00l01-deployment

• Is of type NodePort

• Has a name of cherry

ANSWER: See the explanation for the answer

Explanation:

Set the required context, update the Deployment pod-template label and replica count, then create the NodePort Service in namespace `kdsn00101`.

```
kubectl config use-context k8s

kubectl -n kdsn00101 patch deployment kdsn00101-deployment \
  --type merge \
  -p '{"spec":{"replicas":4,"template":{"metadata":{"labels":{"func":"webFrontEnd"}}}}}'
```

Create the Service using the new pod-template label as its selector:

```
cat <<'EOF' | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: cherry
  namespace: kdsn00101
spec:
```

```
  type: NodePort
  selector:
    func: webFrontEnd
  ports:
    - protocol: TCP
      port: 8080
EOF
```

The Service `cherry` is of type `NodePort`, exposes TCP port `8080`, and selects the Deployment's pods through `func=webFrontEnd`. Since no `targetPort` is specified, Kubernetes defaults it to the service port, `8080`.

Optionally verify:

```
kubectl -n kdsn00101 get deployment kdsn00101-deployment
kubectl -n kdsn00101 get service cherry
```