Linux Foundation CKAD Practice Questions

Shared by Booth on 18-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

SIMULATION

Context

You are asked to set resource requests and limits for a running workload to ensure fair resource management.

''Do not delete the existing Deployment . Failure to do so will result in a reduced score.''

Next, ensure that the total amount of resources in the namespace matches the maximum resources the Pods from the nginx-resources Deployment can request.

Failure to do so will result in the updated Deployment failing to roll out successfully.

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

Below are the exact steps/commands you can run.

1) Locate the Deployment and its namespace

kubectl get deploy -A | grep nginx-resources

You should see output like:

<namespace> nginx-resources ...

Set a variable (replace <NS> with what you see):

NS=<NS>

Confirm replicas:

kubectl -n $NS get deploy nginx-resources -o jsonpath='{.spec.replicas}{'\n'}'

2) Check if there is a ResourceQuota in that namespace

kubectl -n $NS get resourcequota

kubectl -n $NS describe resourcequota

If there is a quota, note these fields (common ones):

requests.cpu

requests.memory

limits.cpu

limits.memory

3) Decide requests/limits for the Deployment (example values)

If the question (in your environment) provides specific values, use those.

If it doesn't, a typical safe pair is:

requests: cpu: 100m, memory: 128Mi

limits: cpu: 200m, memory: 256Mi

I'll proceed with these example values. If your lab specifies different numbers, just swap them in.

4) Update the existing Deployment (DO NOT DELETE)

Option A (fastest): kubectl set resources

Assuming the container name is the first container (we'll detect it):

kubectl -n $NS get deploy nginx-resources -o jsonpath='{.spec.template.spec.containers[*].name}{'\n'}'

If it prints a single container name, set it like this:

kubectl -n $NS set resources deploy nginx-resources \

--requests=cpu=100m,memory=128Mi \

--limits=cpu=200m,memory=256Mi

Verify the Deployment now has resources

kubectl -n $NS get deploy nginx-resources -o jsonpath='{.spec.template.spec.containers[0].resources}{'\n'}'

5) Compute the total resources requested by the Deployment

Get replicas:

REPLICAS=$(kubectl -n $NS get deploy nginx-resources -o jsonpath='{.spec.replicas}')

echo $REPLICAS

6) Ensure the namespace quota matches (or exceeds) those totals

This is the part the question warns about: if the quota is too low, the Deployment update will fail to roll out.

6.1 If a ResourceQuota already exists

Patch it to allow at least the totals you calculated.

First, identify quota name:

RQ=$(kubectl -n $NS get resourcequota -o jsonpath='{.items[0].metadata.name}')

echo $RQ

Then patch (example: replicas=2 requests.cpu=200m, requests.memory=256Mi):

kubectl -n $NS patch resourcequota $RQ --type='merge' -p '{

'spec': {

'hard': {

'requests.cpu': '200m',

'requests.memory': '256Mi',

'limits.cpu': '400m',

'limits.memory': '512Mi'

}

}

}'

Adjust those numbers to your replicas request, and replicas limit (if your quota also enforces limits).

6.2 If there is NO ResourceQuota

Create one that matches the Deployment max request totals.

Example for replicas=2 with our sample requests/limits:

cat <<EOF | kubectl apply -n $NS -f -

apiVersion: v1

kind: ResourceQuota

metadata:

name: nginx-resources-quota

spec:

hard:

requests.cpu: '200m'

requests.memory: '256Mi'

limits.cpu: '400m'

limits.memory: '512Mi'

EOF

7) Verify rollout succeeds

kubectl -n $NS rollout status deploy nginx-resources

kubectl -n $NS get pods

Verify the running pods actually have the requests/limits:

kubectl -n $NS get pod -l app=nginx-resources -o jsonpath='{range .items[*]}{.metadata.name}{' '}{.spec.containers[0].resources}{'\n'}{end}'

(If the label selector app=nginx-resources doesn't exist, just pick a pod name from kubectl get pods and run:)

kubectl -n $NS describe pod | sed -n '/Limits:/,/Requests:/p'

Common reasons this fails (and the fix)

Rollout stuck / pods pending with ''exceeded quota''

Check:

kubectl -n $NS describe pod

kubectl -n $NS describe resourcequota

Fix: increase ResourceQuota hard values to match required totals.

You set requests higher than quota allows

Fix: either reduce requests or raise quota.

kubectl get deploy -A | grep nginx-resources

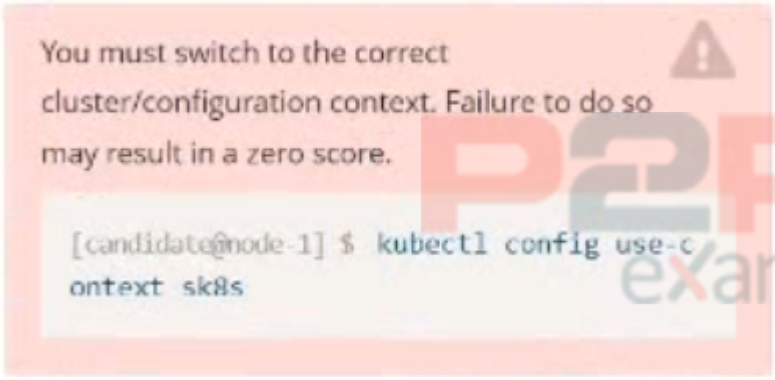
kubectl -n <NS> get deploy nginx-resources -o jsonpath='{.spec.replicas}{'\n'}{.spec.template.spec.containers[0].name}{'\n'}'

kubectl -n <NS> describe resourcequota

Question 2

Question Type: MultipleChoice

Refer to Exhibit.

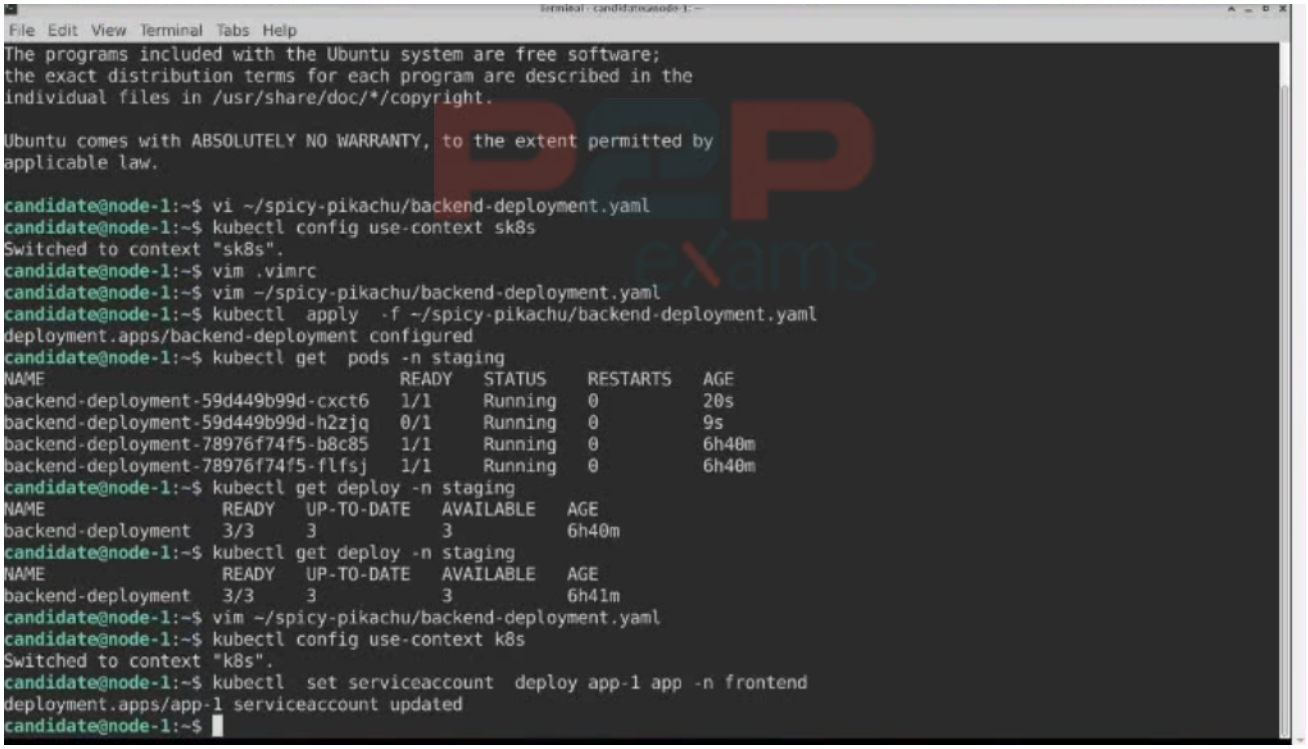


Task:

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

Options:

A- Explanation:
Solution:



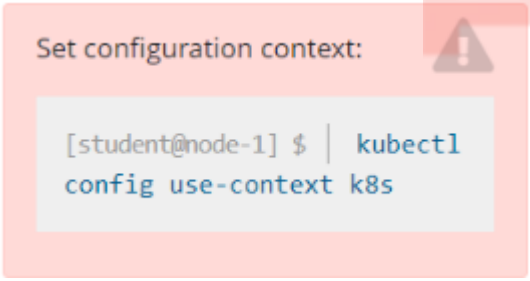
Answer:

A

Question 3

Question Type: MultipleChoice

Refer to Exhibit.



Given a container that writes a log file in format A and a container that converts log files from format A to format B, create a deployment that runs both containers such that the log files from the first container are converted by the second container, emitting logs in format B.

Task:

* Create a deployment named deployment-xyz in the default namespace, that:

* Includes a primary

lfccncf/busybox:1 container, named logger-dev

* includes a sidecar Ifccncf/fluentd:v0.12 container, named adapter-zen

* Mounts a shared volume /tmp/log on both containers, which does not persist when the pod is deleted

* Instructs the logger-dev

container to run the command

```
while true; do
echo "i luv cncf" >> /
tmp/log/input.log;
sleep 10;
done
```

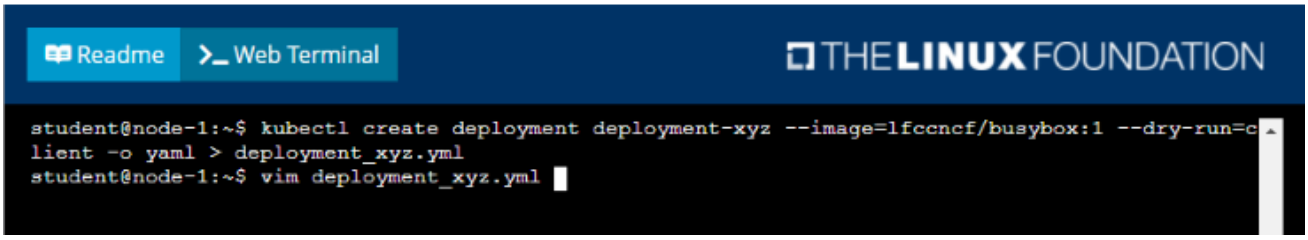
which should output logs to /tmp/log/input.log in plain text format, with example values:

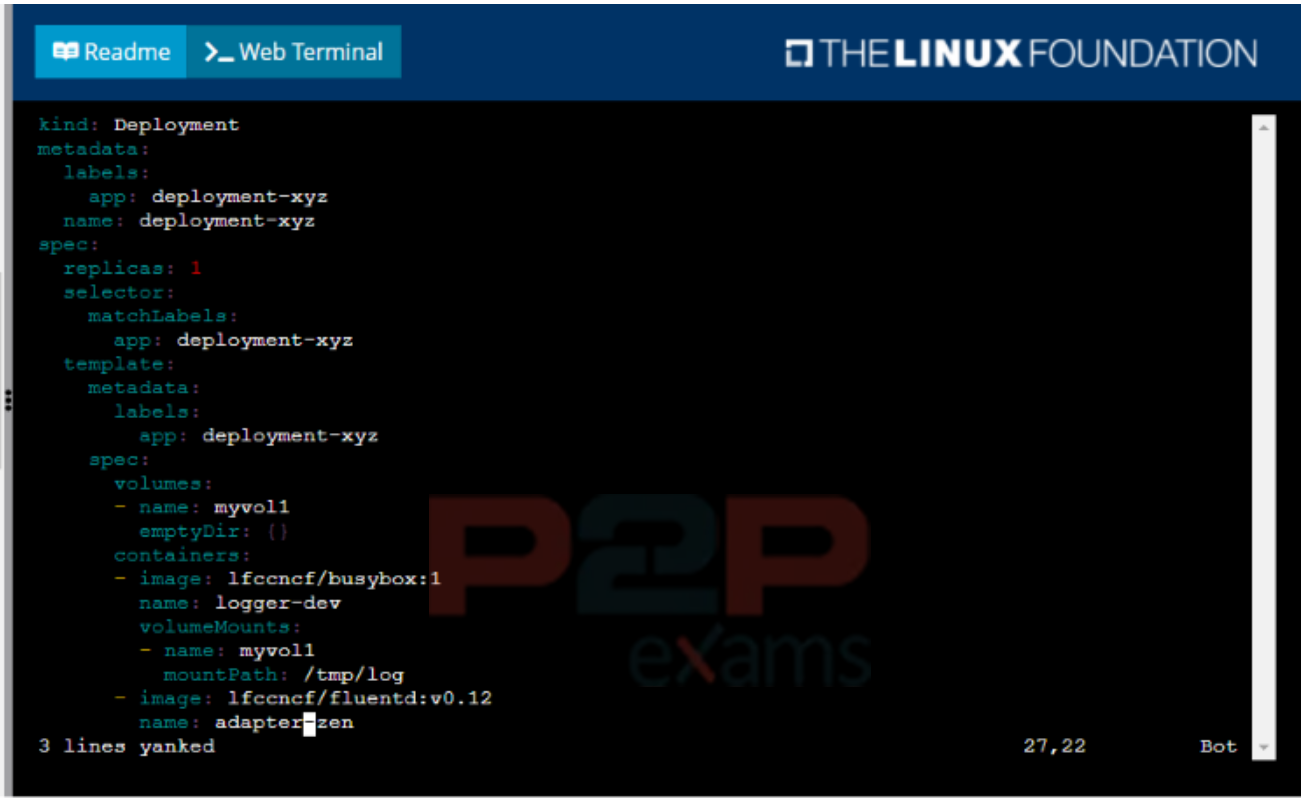
```
i luv cncf
i luv cncf
i luv cncf
```

* The adapter-zen sidecar container should read /tmp/log/input.log and output the data to /tmp/log/output.* in Fluentd JSON format. Note that no knowledge of Fluentd is required to complete this task: all you will need to achieve this is to create the ConfigMap from the spec file provided at /opt/KDMC00102/fluentd-configma p.yaml , and mount that ConfigMap to /fluentd/etc in the adapter-zen sidecar container

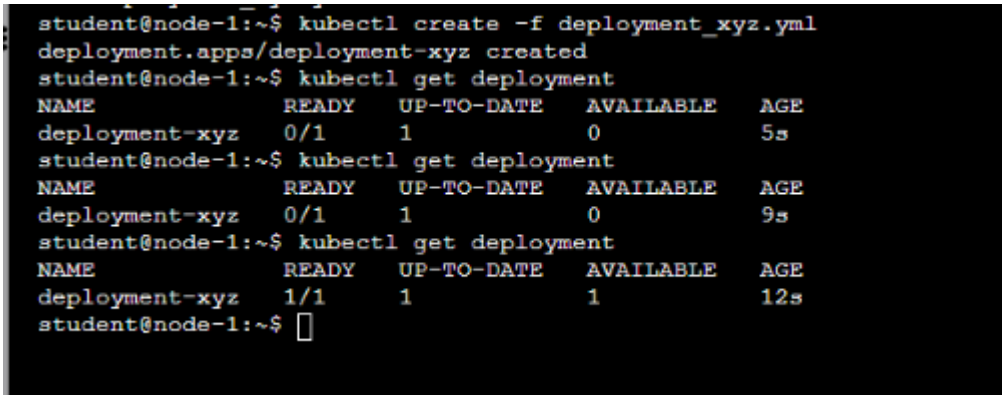
Options:

A- Explanation:
Solution:




Readme
Web Terminal
THE LINUX FOUNDATION
kind: Deployment
metadata:
labels:
app: deployment-xyz
name: deployment-xyz
spec:
replicas: 1
selector:
matchLabels:
app: deployment-xyz
template:
metadata:
labels:
app: deployment-xyz
spec:
volumes:
- name: myvol1
emptyDir: {}
containers:
- image: lfccncf/busybox:1
name: logger-dev
volumeMounts:
- name: myvol1
mountPath: /tmp/log
- image: lfccncf/fluentd:v0.12
name: adapter-zen
3 lines yanked
27,22
Bot


Readme
Web Terminal
THE LINUX FOUNDATION
replicas: 1
selector:
matchLabels:
app: deployment-xyz
template:
metadata:
labels:
app: deployment-xyz
spec:
volumes:
- name: myvol1
emptyDir: {}
containers:
- image: lfccncf/busybox:1
name: logger-dev
command: ["/bin/sh","-c","while [true]; do echo 'i luv cncf' >> /tmp/log/input.log; sl
eep 10; done"]
volumeMounts:
- name: myvol1
mountPath: /tmp/log
- image: lfccncf/fluentd:v0.12
name: adapter-zen
command: ["/bin/sh","-c","tail -f /tmp/log/input.log >> /tmp/log/output.log"]
volumeMounts:
- name: myvol1
mountPath: /tmp/log
29,83
Bot



Answer:

A

Question 4

Question Type: MultipleChoice

SIMULATION

Context

You are tasked to create a secret and consume the secret in a pod using environment variables as follow:

Task

* Create a secret named another-secret with a key/value pair; key1/value4

* Start an nginx pod named nginx-secret using container image nginx, and add an environment variable exposing the value of the secret key key 1, using COOL_VARIABLE as the name for the environment variable inside the pod

Options:

A- See the solution below

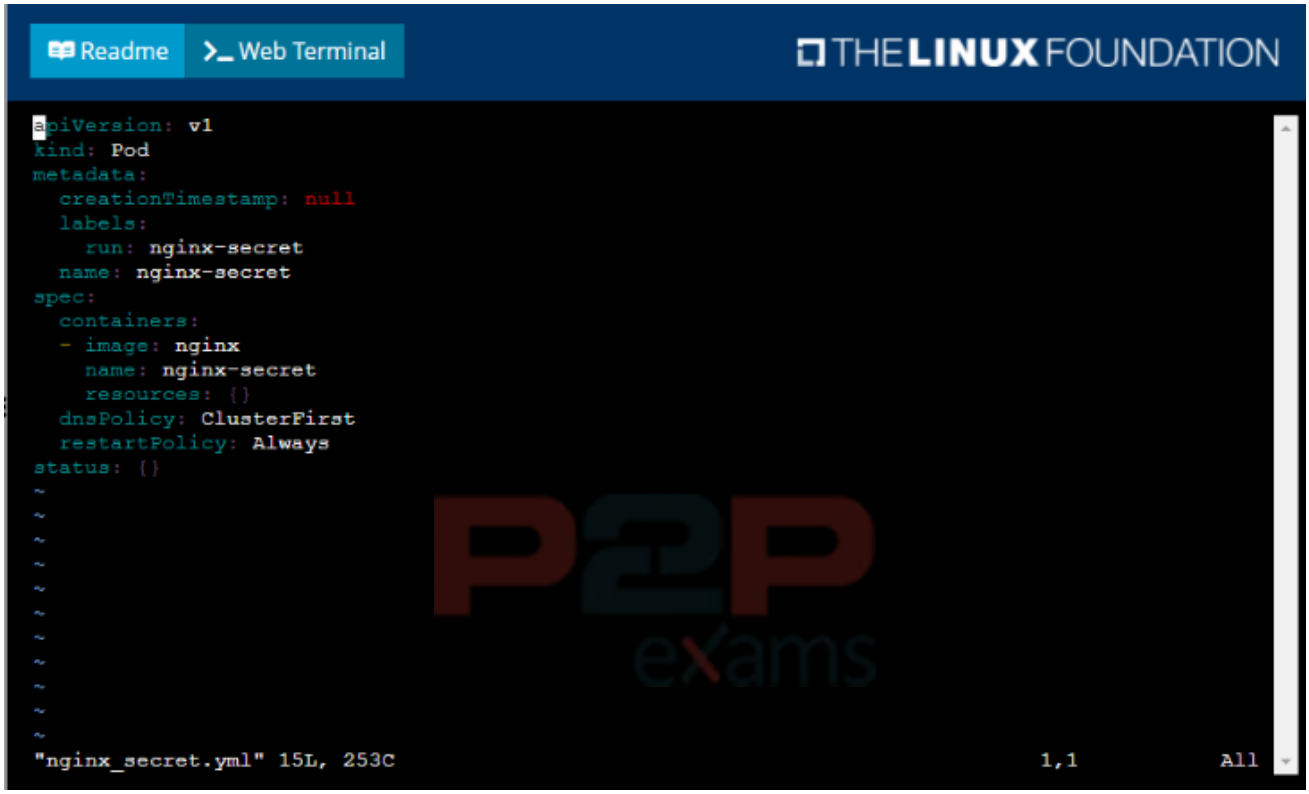
Answer:

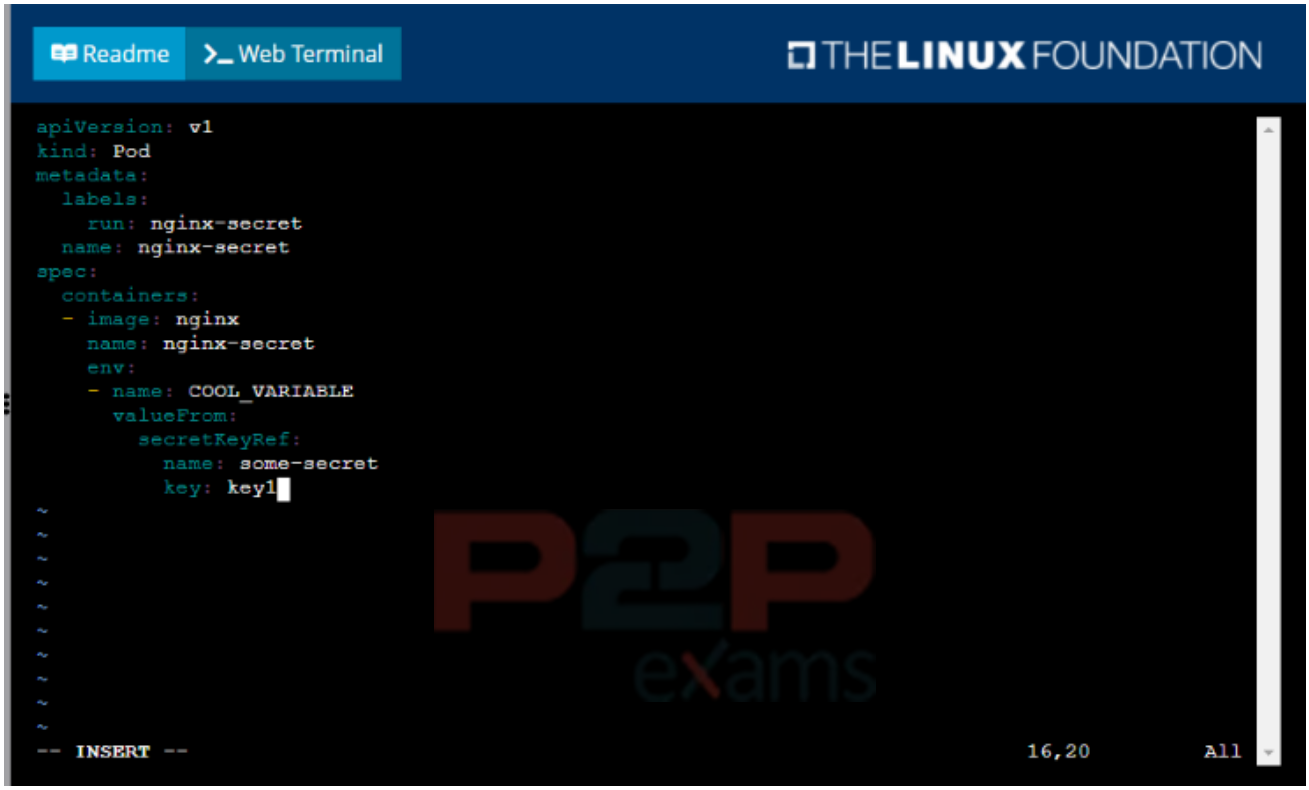
A

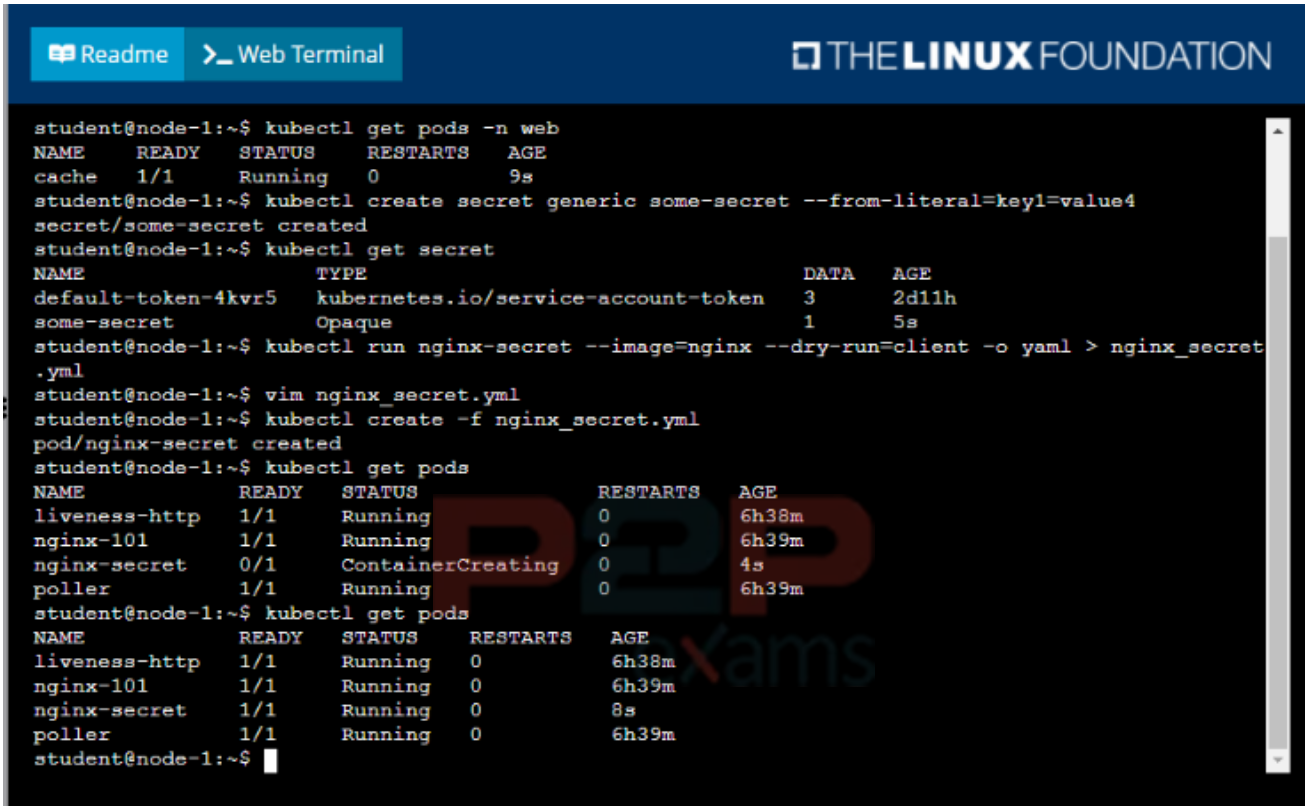
Explanation:

Solution:

```
student@node-1:~$ kubectl create secret generic some-secret --from-literal=key1=value4
secret/some-secret created
student@node-1:~$ kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-4kvr5   kubernetes.io/service-account-token   3      2d11h
some-secret           Opaque                                1      5s
student@node-1:~$ kubectl run nginx-secret --image=nginx --dry-run=client -o yaml > nginx_secret
.yml
student@node-1:~$ vim nginx_secret.yml
```


Readme
Web Terminal
THE LINUX FOUNDATION
apiVersion: v1
kind: Pod
metadata:
creationTimestamp: null
labels:
run: nginx-secret
name: nginx-secret
spec:
containers:
- image: nginx
name: nginx-secret
resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}
"nginx_secret.yml" 15L, 253C
1,1
All


Readme
Web Terminal
THE LINUX FOUNDATION
apiVersion: v1
kind: Pod
metadata:
labels:
run: nginx-secret
name: nginx-secret
spec:
containers:
- image: nginx
name: nginx-secret
env:
- name: COOL_VARIABLE
valueFrom:
secretKeyRef:
name: some-secret
key: key1
-- INSERT --
16,20
All



Question 5

Question Type: MultipleChoice

SIMULATION

Context

You are asked to allow a Pod to communicate with two other Pods but nothing else.

You must connect to the correct host . Failure to do so may result

in a zero score.

!

[candidate@base] $ ssh ckad000

18

charming-macaw namespace to use a NetworkPolicy allowing the Pod to send and receive traffic only to and from the Pods front and db.

All required NetworkPolicies have already been created.

You must not create, modify or delete any NetworkPolicy while working on this task. You may

only use existing NetworkPolicies .

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

ssh ckad00018

You cannot create/modify/delete any NetworkPolicy.

So the only way to make the existing policies ''take effect'' is to ensure the right Pods have the labels/selectors those policies expect.

The task: in namespace charming-macaw, configure things so the target Pod can send + receive traffic ONLY to/from Pods front and db.

1) Inspect what NetworkPolicies already exist (don't change them)

kubectl -n charming-macaw get netpol

kubectl -n charming-macaw get netpol -o wide

Dump them to see the selectors they use:

kubectl -n charming-macaw get netpol -o yaml

You are looking for policies that:

select the restricted pod via spec.podSelector

and allow ingress/egress only with selectors that match front and db

often there's also a ''default deny'' policy.

2) Identify the Pods and their current labels

kubectl -n charming-macaw get pods -o wide

kubectl -n charming-macaw get pods --show-labels

Specifically inspect labels for front and db:

kubectl -n charming-macaw get pod front --show-labels

kubectl -n charming-macaw get pod db --show-labels

(If they're Deployments instead of single Pods, do:)

kubectl -n charming-macaw get deploy --show-labels

kubectl -n charming-macaw get pods -l app=front --show-labels

kubectl -n charming-macaw get pods -l app=db --show-labels

3) Figure out which pod is ''the Pod'' to restrict

Usually there's a third pod (e.g., backend, api, app) besides front and db.

List pods again and identify the ''other'' one:

kubectl -n charming-macaw get pods

Let's assume the pod to restrict is called app (replace as needed):

TARGET=

4) Match the existing NetworkPolicy selectors by labeling pods (allowed)

Because you can't edit NetworkPolicies, you must make labels on Pods (or their controllers) match the policies' selectors.

4.1 Determine the label required on the TARGET pod

From the YAML, find the policy that selects the restricted pod, e.g.:

spec:

podSelector:

matchLabels:

role: restricted

Extract podSelector from each policy quickly:

kubectl -n charming-macaw get netpol -o jsonpath='{range .items[*]}{.metadata.name}{' => '}{.spec.podSelector}{'\n'}{end}'

Pick the selector that is meant for the restricted pod, then apply it to the TARGET pod (example: role=restricted):

kubectl -n charming-macaw label pod $TARGET role=restricted --overwrite

Best practice (if the pod is managed by a Deployment): label the Deployment template instead, so it persists.

Find the owner:

kubectl -n charming-macaw get pod $TARGET -o

jsonpath='{.metadata.ownerReference[0].kind}{' '}{.metadata.ownerReference[0].name}{'\n'}'

If it's a ReplicaSet, find its Deployment:

RS=$(kubectl -n charming-macaw get pod $TARGET -o
jsonpath='{.metadata.ownerReference[0].name}')

kubectl -n charming-macaw get rs $RS -o jsonpath='{.metadata.ownerReference[0].kind}{'
'}{.metadata.ownerReference[0].name}{'\n'}'

Then label the Deployment (example):

kubectl -n charming-macaw label deploy <DEPLOYMENT_NAME> role=restricted --overwrite

4.2 Ensure front and db match what the allow-rules reference

Look inside the allow policy ingress.from / egress.to. You might see something like:

from:

- podSelector:

matchLabels:

name: front

- podSelector:

matchLabels:

name: db

So you must ensure:

front pod has name=front

db pod has name=db

Apply labels (examples---use what the policy expects):

kubectl -n charming-macaw label pod front name=front --overwrite

kubectl -n charming-macaw label pod db name=db --overwrite

Again, if they're Deployments, label the Deployment instead:

kubectl -n charming-macaw label deploy front name=front --overwrite

kubectl -n charming-macaw label deploy db name=db --overwrite

5) Verify the NetworkPolicies now ''select'' the right pods

Check which labels each pod has now:

kubectl -n charming-macaw get pods --show-labels

Confirm the restricted pod matches the NetPol podSelector:

kubectl -n charming-macaw get netpol <POLICY_NAME> -o jsonpath='{.spec.podSelector}{'\n'}'

kubectl -n charming-macaw get pod $TARGET --show-labels

6) Functional verification (quick network tests)

Exec into the restricted pod and try to reach:

front allowed

db allowed

anything else blocked

If busybox has wget:

kubectl -n charming-macaw exec -it $TARGET -- sh -c 'wget -qO- http://front 2>/dev/null || true'

kubectl -n charming-macaw exec -it $TARGET -- sh -c 'wget -qO- http://db 2>/dev/null || true'

Test something that should be blocked (example: kubernetes service DNS name):

kubectl -n charming-macaw exec -it $TARGET -- sh -c 'wget -qO- https://kubernetes.default.svc 2>/dev/null || echo 'blocked''

Also test inbound (from front to target, and from db to target) if the target listens on a port; otherwise inbound testing may be limited.

What you're doing conceptually

Existing NetPols are already correct.

Your job is to make pod labels match the NetPol selectors so:

default deny applies to the target

allow rules apply only between target front and target db

Question 6

Question Type: MultipleChoice

SIMULATION

You must connect to the correct host . Failure to do so may result in a zero score.

[candidate@base] $ ssh ckad00033

Task

A Dockerfile has been prepared at /home/candidate/build/Dockerfile.

Using the prepared Dockerfile, build a container image with the name macaque and tag 1.2. You may install and use the tool of your choice.

Multiple image builders and tools have been pre-installed in the base system,

including: docker, skopeo, buildah, img , and podman.

Please do not push the built image to a registry, run a container, or otherwise consume it.

Using the tool of your choice, export the built container image in OCI or Docker image format and store it at /home/candidate/macaque-1.2.tar.

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

Understood --- I'll redo Question 13 without using any (or similar) icons.

ssh ckad00033

This task is only about building and exporting a container image.

You must not push it, run it, or consume it in any way.

You may use any image builder. Below are clean, correct solutions. Use one of them.

Option A: Using Docker

1) Go to the Dockerfile location

cd /home/candidate/build

ls -l Dockerfile

2) Build the image

Image name: macaque

Tag: 1.2

docker build -t macaque:1.2 .

Verify:

docker images | grep macaque

3) Export the image to a tar file

Docker image format is acceptable.

docker save macaque:1.2 -o /home/candidate/macaque-1.2.tar

Verify the file exists:

ls -lh /home/candidate/macaque-1.2.tar

Do not load or run the image.

Option B: Using Podman (rootless alternative)

1) Build the image

cd /home/candidate/build

podman build -t macaque:1.2 .

Verify:

podman images | grep macaque

2) Export the image

podman save macaque:1.2 -o /home/candidate/macaque-1.2.tar

Verify:

ls -lh /home/candidate/macaque-1.2.tar

Option C: Using Buildah (OCI format)

cd /home/candidate/build

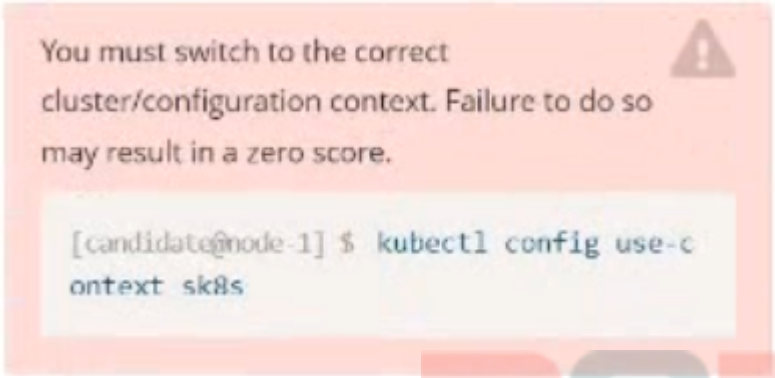
buildah bud -t macaque:1.2 .

buildah push macaque:1.2 oci-archive:/home/candidate/macaque-1.2.tar

Question 7

Question Type: MultipleChoice

SIMULATION



Task:

1- Update the Propertunel scaling configuration of the Deployment web1 in the ckad00015 namespace setting maxSurge to 2 and maxUnavailable to 59

2- Update the web1 Deployment to use version tag 1.13.7 for the Ifconf/nginx container image.

3- Perform a rollback of the web1 Deployment to its previous version

Options:

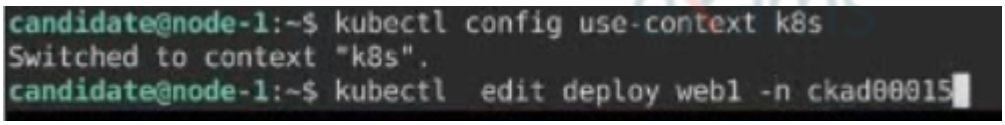
A- See the solution below

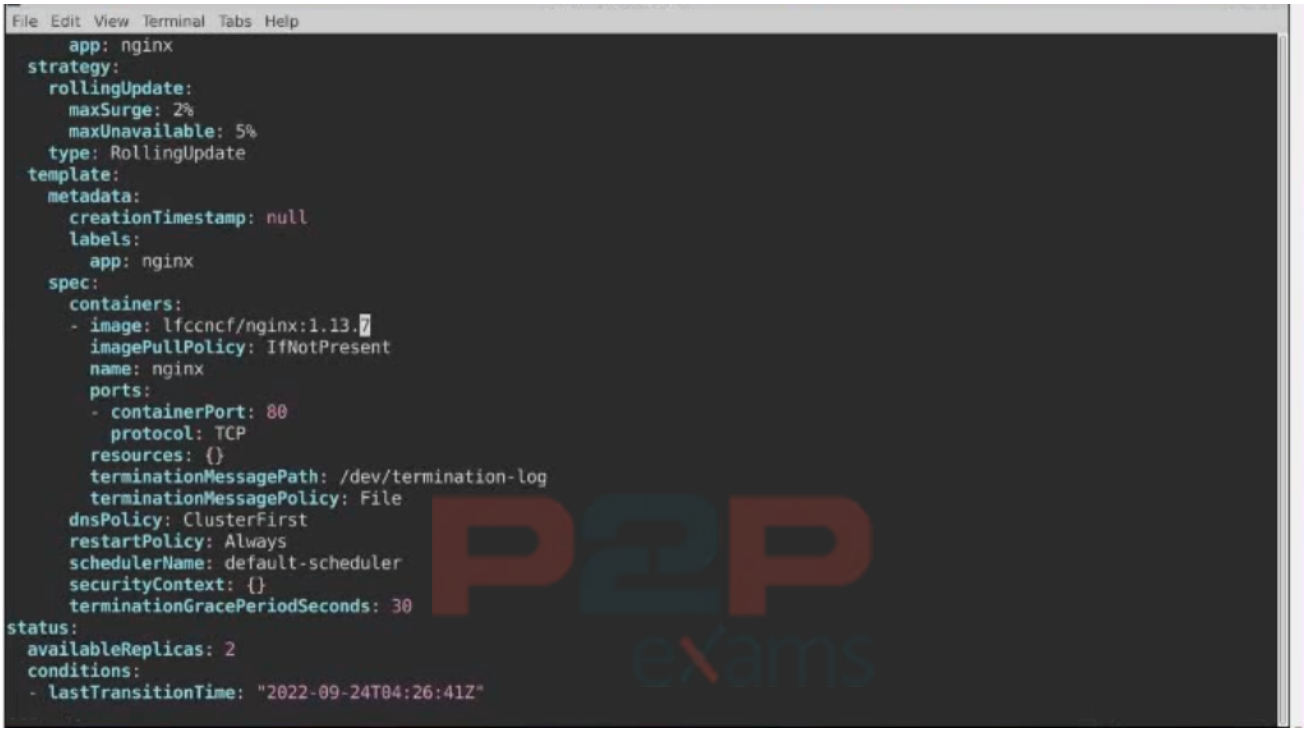
Answer:

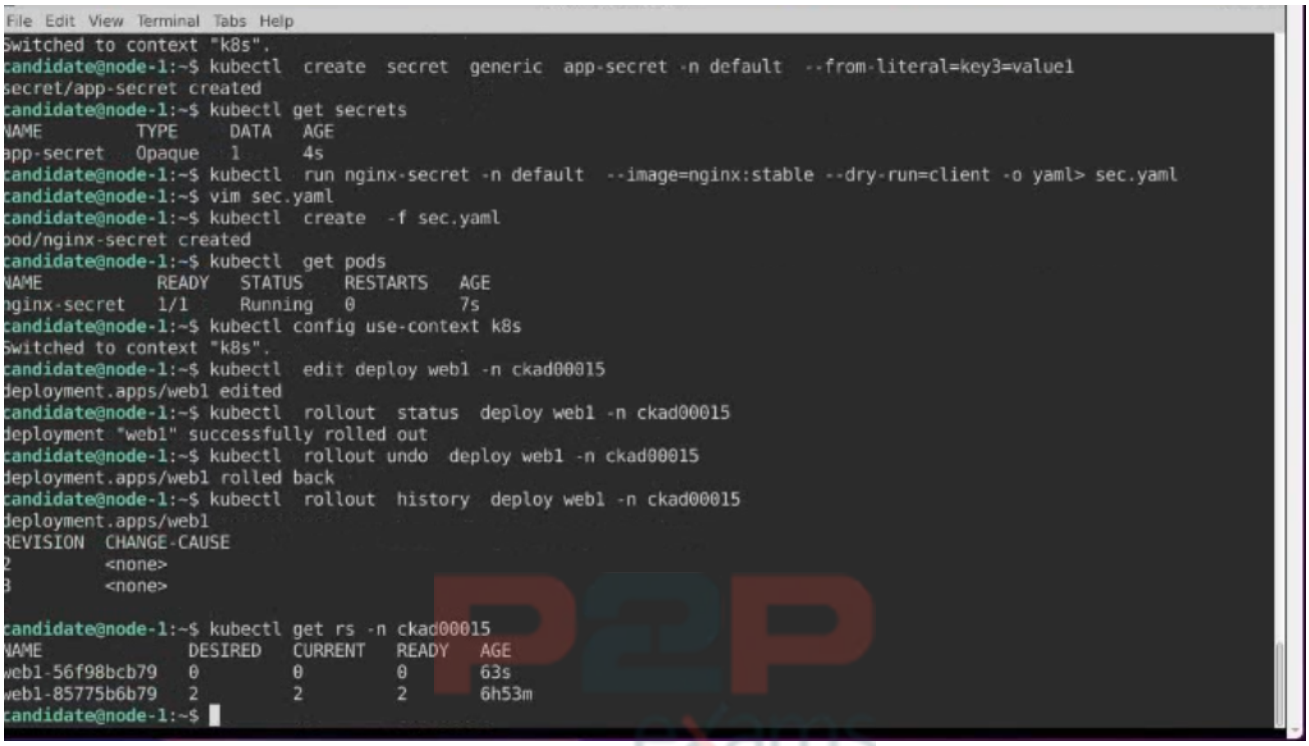
A

Explanation:

Solution:



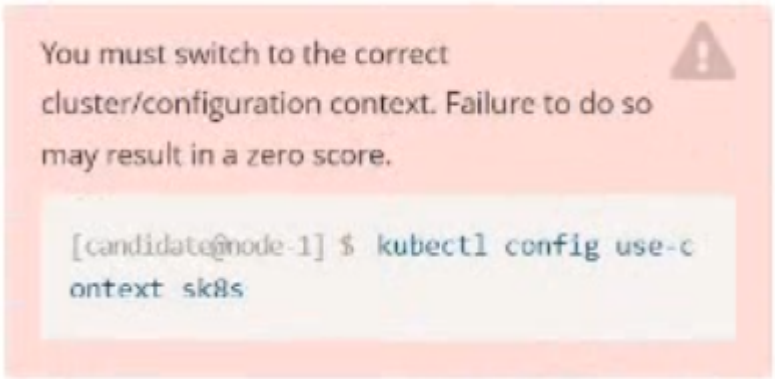




Question 8

Question Type: MultipleChoice

SIMULATION



Task:

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

Options:

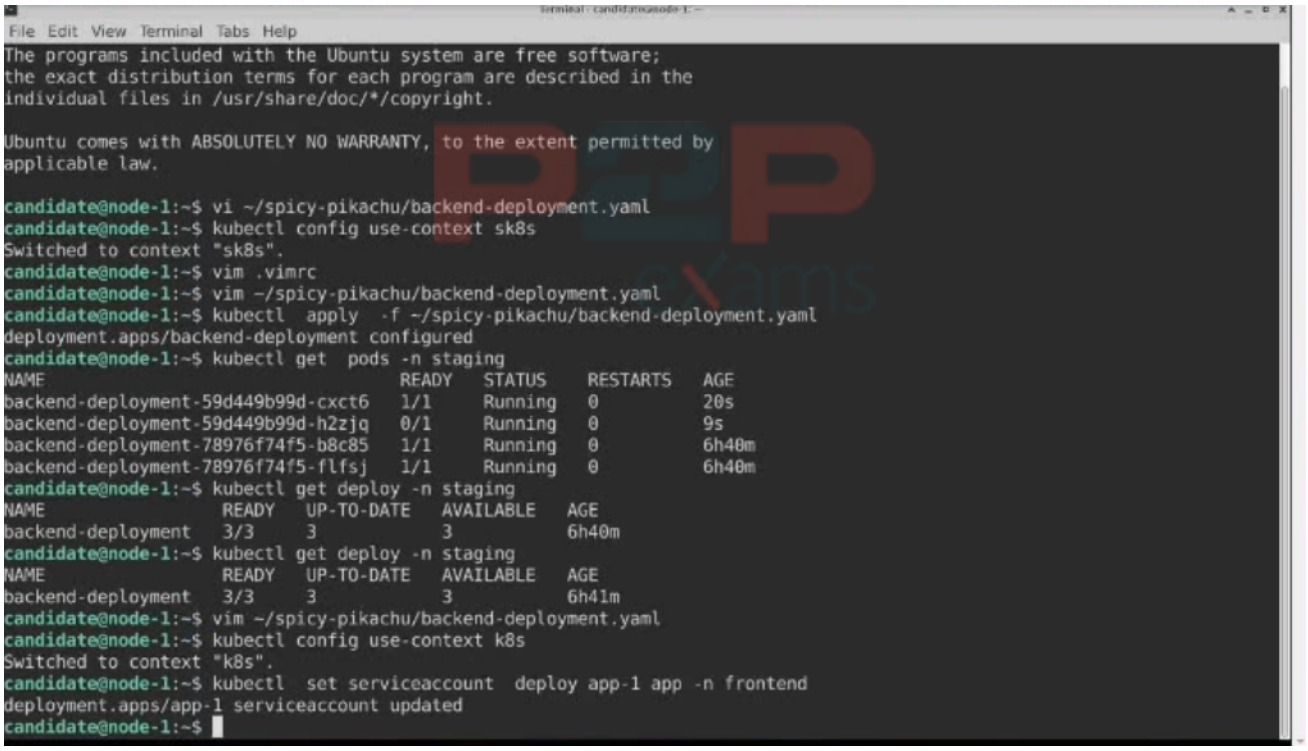
A- See the solution below

Answer:

A

Explanation:

Solution:



Question 9

Question Type: MultipleChoice

SIMULATION

Set Configuration Context:

[student@node-1] $ | kubectl

Config use-context k8s

Context

A user has reported an aopticauon is unteachable due to a failing livenessProbe .

Task

Perform the following tasks:

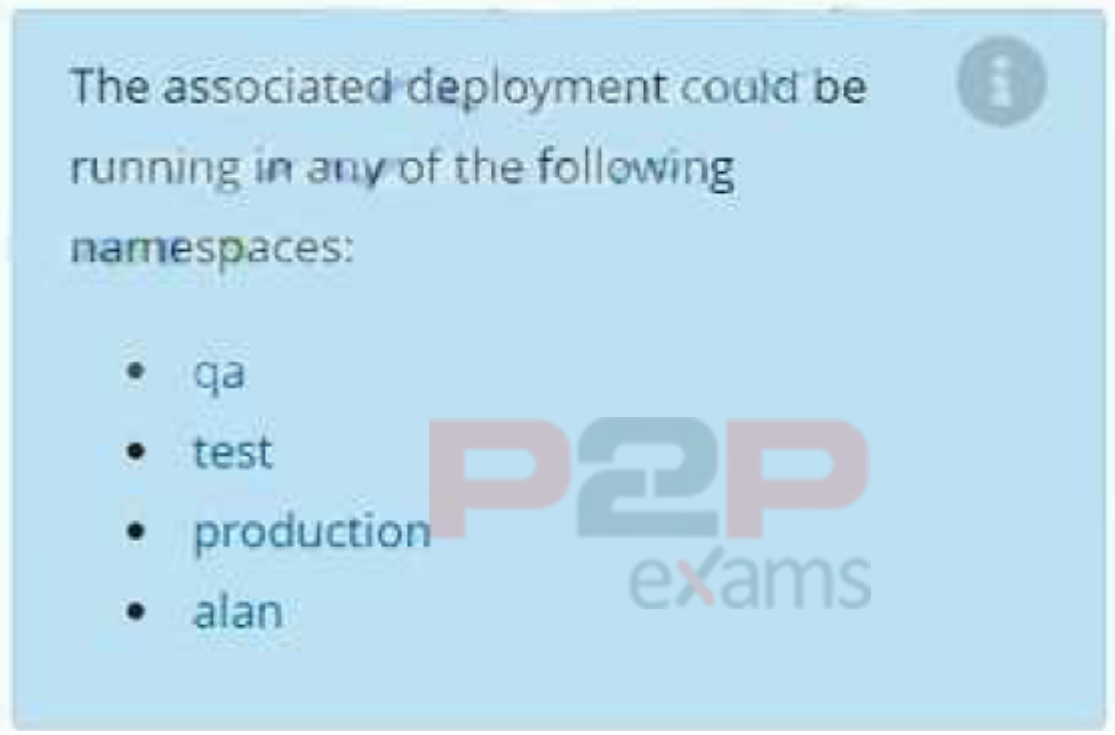
* Find the broken pod and store its name and namespace to /opt/KDOB00401/broken.txt in the format:

/

The output file has already been created

* Store the associated error events to a file /opt/KDOB00401/error.txt, The output file has already been created. You will need to use the -o wide output specifier with your command

* Fix the issue.



Options:

A- See the solution below

Answer:

A

Explanation:

To find the broken pod and store its name and namespace to /opt/KDOB00401/broken.txt, you can use the kubectl get pods command and filter the output by the status of the pod.

kubectl get pods --field-selector=status.phase=Failed -o jsonpath='{.items[*].metadata.namespace}/{.items[*].metadata.name}' > /opt/KDOB00401/broken.txt

This command will list all pods with a status of Failed and output their names and namespaces in the format <namespace>/. The output is then written to the /opt/KDOB00401/broken.txt file.

To store the associated error events to a file /opt/KDOB00401/error.txt, you can use the kubectl describe command to retrieve detailed information about the pod, and the grep command to filter the output for error events.

kubectl describe pods --namespace | grep -i error -B5 -A5 > /opt/KDOB00401/error.txt

Replace and with the name and namespace of the broken pod you found in the previous step.

This command will output detailed information about the pod, including error events. The grep command filters the output for lines containing 'error' and also prints 5 lines before and after the match.

To fix the issue, you need to analyze the error events and find the root cause of the issue.

It could be that the application inside the pod is not running, the container image is not available, the pod has not enough resources, or the liveness probe configuration is incorrect.

Once you have identified the cause, you can take appropriate action, such as restarting the application, updating the container image, increasing the resources, or modifying the liveness probe configuration.

After fixing the issue, you can use the kubectl get pods command to check the status of the pod and ensure

P2P exams