NO.1 Check the image version in pod without the describe command

Answer:

kubectl get po nginx -o
jsonpath='{.spec.containers[].image}{"\n"}'

NO.2 Create a nginx pod with label env=test in engineering namespace

Answer:

kubectl run nginx --image=nginx --restart=Never --labels=env=test --namespace=engineering --dry-run -o yaml > nginx-pod.yaml kubectl run nginx --image=nginx --restart=Never --labels=env=test --namespace=engineering --dry-run -o yaml | kubectl create -n engineering -f - YAML File:
apiVersion: v1
kind: Pod
metadata:
name: nginx
namespace: engineering
labels:
env: test
spec:
containers:
- name: nginx
image: nginx
imagePullPolicy: IfNotPresent
restartPolicy: Never
kubectl create -f nginx-pod.yaml

NO.3 Score: 4%

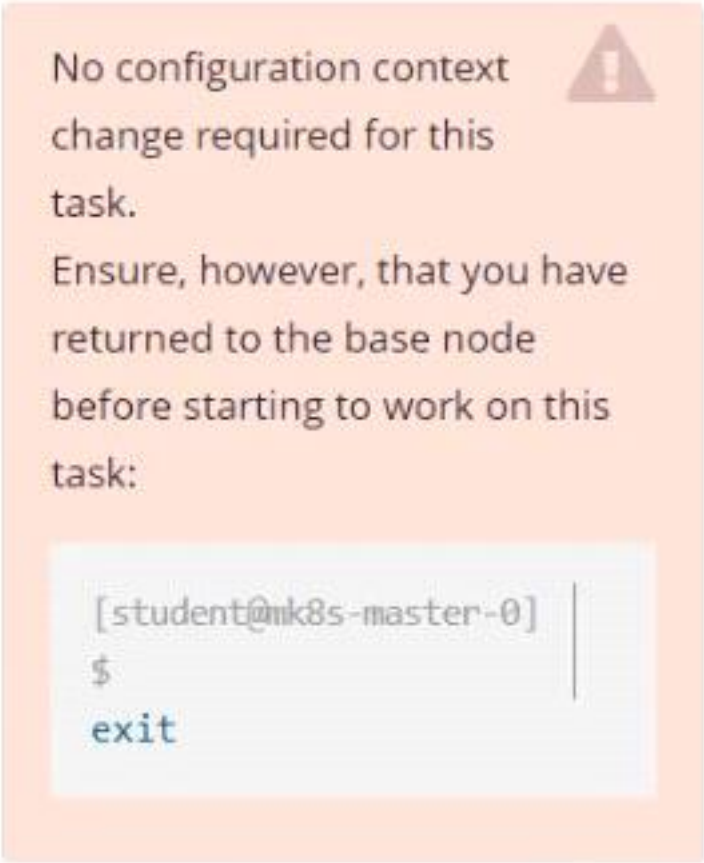
Task
Create a persistent volume with name app-data , of capacity 1Gi and access mode ReadOnlyMany. The type of volume is hostPath and its location is /srv/app-data .

Answer:

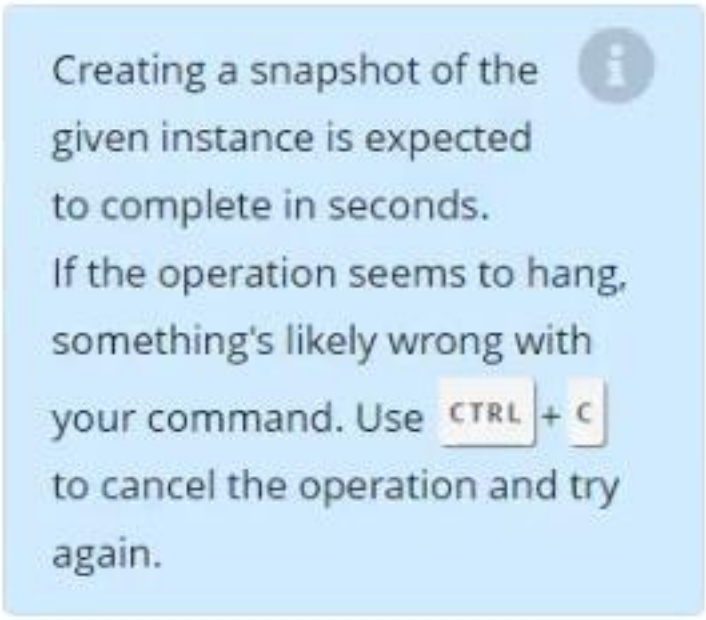
Solution:
#vi pv.yaml
apiVersion: v1
kind: PersistentVolume

metadata:
name: app-config
spec:
capacity:
storage: 1Gi
accessModes:
- ReadOnlyMany
hostPath:
path: /srv/app-config
#
kubectl create -f pv.yaml

NO.4 Score: 7%



Task
First, create a snapshot of the existing etcd instance running at https://127.0.0.1:2379, saving the snapshot to /srv/data/etcd-snapshot.db.



Next, restore an existing, previous snapshot located at /var/lib/backup/etcd-snapshot-previo us.db

Answer:

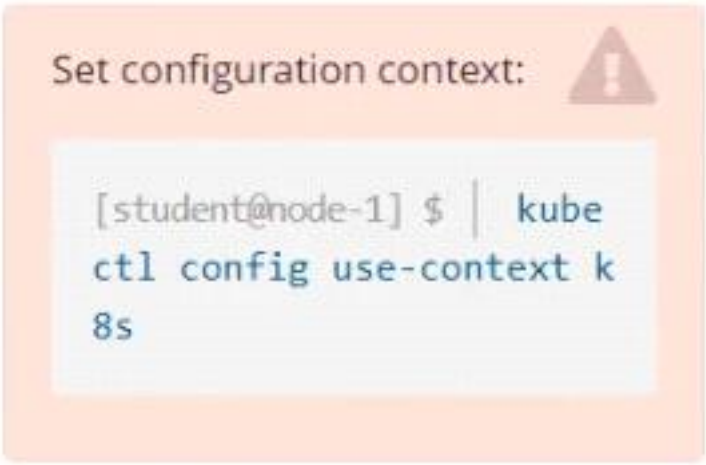
Solution:

#backup

ETCDCTL_API=3 etcdctl --endpoints="https://127.0.0.1:2379" --cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --key=/opt/KUIN000601/etcd-client.key snapshot save /etc/data/etcd-snapshot.db

#restore
ETCDCTL_API=3 etcdctl --endpoints="https://127.0.0.1:2379" --cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --key=/opt/KUIN000601/etcd-client.key snapshot restore /var/lib/backup/etcd-snapshot-previoys.db

NO.5 Score: 7%



Task
Create a new nginx Ingress resource as follows:
* Name: ping
* Namespace: ing-internal
* Exposing service hi on path /hi using service port 5678

Answer:
Solution:
vi ingress.yaml
#
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:

name: ping
namespace: ing-internal
spec:
rules:
- http:
paths:
- path: /hi
pathType: Prefix
backend:
service:
name: hi
port:
number: 5678
#
kubectl create -f ingress.yaml

NO.6 Create a persistent volume with name app-data, of capacity 2Gi and access mode ReadWriteMany. The type of volume is hostPath and its location is /srv/app-data.

Answer:

solution
Persistent Volume
A persistent volume is a piece of storage in a Kubernetes cluster. PersistentVolumes are a cluster-level resource like nodes, which don't belong to any namespace. It is provisioned by the administrator and has a particular file size. This way, a developer deploying their app on Kubernetes need not know the underlying infrastructure. When the developer needs a certain amount of persistent storage for their application, the system administrator configures the cluster so that they consume the PersistentVolume provisioned in an easy way.
Creating Persistent Volume
kind: PersistentVolume apiVersion: v1 metadata: name:app-data spec: capacity: # defines the capacity of PV we are creating storage: 2Gi #the amount of storage we are tying to claim accessModes: # defines the rights of the volume we are creating - ReadWriteMany hostPath: path: "/srv/app-data" # path to which we are creating the volume Challenge Create a Persistent Volume named app-data, with access mode ReadWriteMany, storage classname shared, 2Gi of storage capacity and the host path /srv/app-data.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-data
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
      path: /srv/app-data
  storageClassName: shared
~
~
~
~
~
~
~
~
~
~
~
~
"app-data.yaml" 12L, 194C
```

2. Save the file and create the persistent volume.

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl create -f pv.yaml
persistentvolume/pv created
```

3. View the persistent volume.

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
app-data   2Gi        RWX            Retain           Available           shared                  31s
```

Our persistent volume status is available meaning it is available and it has not been mounted yet. This status will change when we mount the persistentVolume to a persistentVolumeClaim.

PersistentVolumeClaim

In a real ecosystem, a system admin will create the PersistentVolume then a developer will create a PersistentVolumeClaim which will be referenced in a pod. A PersistentVolumeClaim is created by specifying the minimum size and the access mode they require from the persistentVolume.

Challenge

Create a Persistent Volume Claim that requests the Persistent Volume we had created above. The claim should request 2Gi. Ensure that the Persistent Volume Claim has the same storageClassName as the persistentVolume you had previously created.

kind: PersistentVolume apiVersion: v1 metadata: name:app-data

spec:

accessModes: - ReadWriteMany resources:

requests: storage: 2Gi

storageClassName: shared

2. Save and create the pvc

njerry191@cloudshell:~ (extreme-clone-2654111)$ kubect1 create -f app-data.yaml

persistentvolumeclaim/app-data created

3. View the pvc

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pvc
NAME    STATUS    VOLUME    CAPACITY    ACCESS MODES    STORAGECLASS
pv      Bound     pv        512m        RWX             shared
```

4. Let's see what has changed in the pv we had initially created.

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM        STORAGECLASS   REASON   AGE
pv      512m       RWX            Retain           Bound    default/pv   shared         16m
```

Our status has now changed from available to bound.

5. Create a new pod named myapp with image nginx that will be used to Mount the Persistent Volume Claim with the path /var/app/config.

Mounting a Claim

apiVersion: v1 kind: Pod metadata: creationTimestamp: null name: app-data spec: volumes: - name:congigpvc persistenVolumeClaim: claimName: app-data containers: - image: nginx name: app volumeMounts: - mountPath: "/srv/app-data " name: configpvc

NO.7 List "nginx-dev" and "nginx-prod" pod and delete those pods

Answer:

kubect1 get pods -o wide

kubectl delete po "nginx-dev" kubectl delete po "nginx-prod"