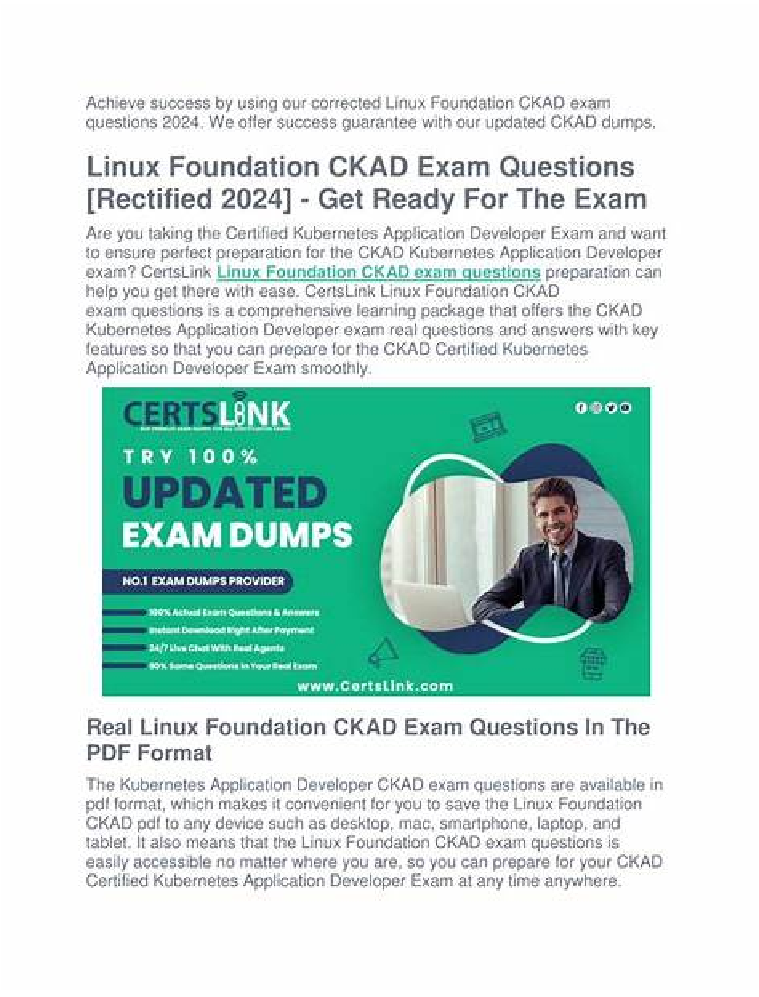
Pass Guaranteed Quiz CKAD - Authoritative Linux Foundation Certified Kubernetes Application Developer Exam Certification Torrent



Achieve success by using our corrected Linux Foundation CKAD exam questions 2024. We offer success guarantee with our updated CKAD dumps.

Linux Foundation CKAD Exam Questions [Rectified 2024] - Get Ready For The Exam

Are you taking the Certified Kubernetes Application Developer Exam and want to ensure perfect preparation for the CKAD Kubernetes Application Developer exam? CertsLink Linux Foundation CKAD exam questions preparation can help you get there with ease. CertsLink Linux Foundation CKAD exam questions is a comprehensive learning package that offers the CKAD Kubernetes Application Developer exam real questions and answers with key features so that you can prepare for the CKAD Certified Kubernetes Application Developer Exam smoothly.

CERTSLINK
TRY 100%
UPDATED
EXAM DUMPS
NO.1 EXAM DUMPS PROVIDER
100% Actual Exam Questions & Answers
Instant Download Right After Payment
24/7 Live Chat With Real Agents
90% Same Questions In Your Real Exam
www.CertsLink.com

Real Linux Foundation CKAD Exam Questions In The PDF Format

The Kubernetes Application Developer CKAD exam questions are available in pdf format, which makes it convenient for you to save the Linux Foundation CKAD pdf to any device such as desktop, mac, smartphone, laptop, and tablet. It also means that the Linux Foundation CKAD exam questions is easily accessible no matter where you are, so you can prepare for your CKAD Certified Kubernetes Application Developer Exam at any time anywhere.

What's more, part of that ActualPDF CKAD dumps now are free: https://drive.google.com/open?id=1Jpqg68zB_FsNIYF7xogB-ogPmA5-O9j1

In modern society, innovation is of great significance to the survival of a company. The new technology of the CKAD practice prep is developing so fast. So the competitiveness among companies about the study materials is fierce. Luckily, our company masters the core technology of developing the CKAD Exam Questions. On one hand, our professional experts can apply the most information technology to compile the content of the CKAD learning materials. On the other hand, they also design the displays according to the newest display technology.

The CKAD Certification Exam is a hands-on, performance-based exam, which means that the candidate needs to complete a set of tasks within a specified time limit. CKAD exam is conducted online and requires the candidate to have access to a Kubernetes cluster. Linux Foundation Certified Kubernetes Application Developer Exam certification exam tests the candidate's ability to understand Kubernetes architecture, deploy and manage applications, configure and run services, and troubleshoot common issues. Linux Foundation Certified Kubernetes Application Developer Exam certification exam is a testament to the candidate's practical skills and knowledge of Kubernetes, and it is recognized globally by organizations looking to hire Kubernetes professionals.

>> CKAD Certification Torrent <<

Linux Foundation CKAD Exam Study Material of ActualPDF in 3 Formats

Thanks to modern technology, learning online gives people access to a wider range of knowledge, and people have got used to convenience of electronic equipment. As you can see, we are selling our CKAD learning guide in the international market, thus there are three different versions of our CKAD exam materials which are prepared to cater the different demands of various people. It is worth mentioning that, the simulation test is available in our software version. With the simulation test, all of our customers will get accustomed to the CKAD Exam easily, and get rid of bad habits, which may influence your performance in the real CKAD exam. In addition, the mode of CKAD learning guide questions and answers is the most effective for you to remember the key points. During your practice process, the CKAD test questions would be absorbed, which is time-saving and high-efficient.

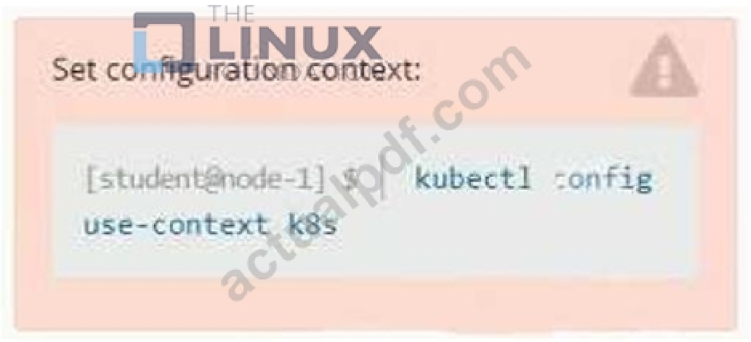
Exam Topics for CNCF Certified Kubernetes Application Developer

Our **CNCF CKAD Dumps** covers the following objectives of the CNCF Certified Kubernetes Application Developer Exam.

- Pod Design 20%
- Core Concepts 13%
- Services & Networking 13%
- State Persistence 8%

Linux Foundation Certified Kubernetes Application Developer Exam Sample Questions (Q94-Q99):

NEW QUESTION # 94
Refer to Exhibit.


Set configuration context:

```
[student@node-1] $  kubectl config
use-context k8s
```

Set Configuration Context:
[student@node-1] $ | kubectl
Config use-context k8s
Context
A pod is running on the cluster but it is not responding.
Task
The desired behavior is to have Kubernetes restart the pod when an endpoint returns an HTTP 500 on the /healthz endpoint. The service, probe-pod, should never send traffic to the pod while it is failing. Please complete the following:
* The application has an endpoint, /started, that will indicate if it can accept traffic by returning an HTTP 200. If the endpoint returns an HTTP 500, the application has not yet finished initialization.
* The application has another endpoint /healthz that will indicate if the application is still working as expected by returning an HTTP 200. If the endpoint returns an HTTP 500 the application is no longer responsive.
* Configure the probe-pod pod provided to use these endpoints
* The probes should use port 8080

Answer:

Explanation:
Solution:
To have Kubernetes automatically restart a pod when an endpoint returns an HTTP 500 on the /healthz endpoint, you will need to configure liveness and readiness probes on the pod.
First, you will need to create a livenessProbe and a readinessProbe in the pod's definition yaml file. The livenessProbe will check the /healthz endpoint, and if it returns an HTTP 500, the pod will be restarted. The readinessProbe will check the /started endpoint, and if it returns an HTTP 500, the pod will not receive traffic.
Here's an example of how you can configure the liveness and readiness probes in the pod definition yaml file:
apiVersion: v1
kind: Pod
metadata:

```
name: probe-pod
spec:
containers:
- name: probe-pod
image: <image-name>
ports:
- containerPort: 8080
livenessProbe:
httpGet:
path: /healthz
port: 8080
initialDelaySeconds: 15
periodSeconds: 10
failureThreshold: 3
readinessProbe:
httpGet:
path: /started
port: 8080
initialDelaySeconds: 15
periodSeconds: 10
failureThreshold: 3
```

The httpGet specifies the endpoint to check and the port to use. The initialDelaySeconds is the amount of time the pod will wait before starting the probe. periodSeconds is the amount of time between each probe check, and the failureThreshold is the number of failed probes before the pod is considered unresponsive.
You can use kubectl to create the pod by running the following command:

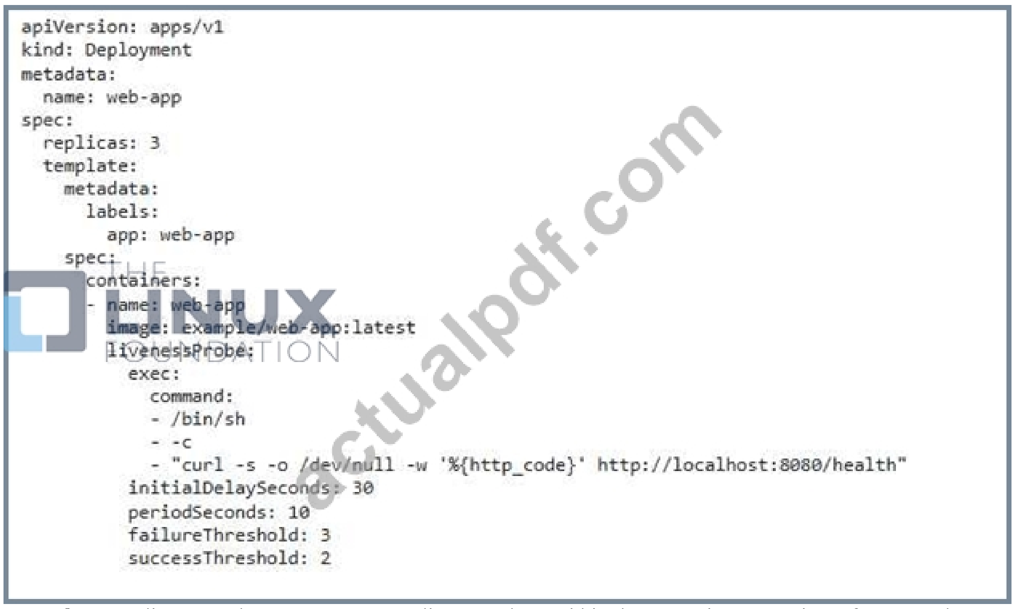
```
kubectl apply -f <filename>.yaml
```

Once the pod is created, Kubernetes will start monitoring it using the configured liveness and readiness probes. If the /healthz endpoint returns an HTTP 500, the pod will be restarted. If the /started endpoint returns an HTTP 500, the pod will not receive traffic.
Please note that if the failure threshold is set to 3, it means that if the probe fails 3 times consecutively it will be considered as a failure.
The above configuration assumes that the application is running on port 8080 and the endpoints are available on the same port.

NEW QUESTION # 95
You have a Deployment named 'web-app' running a containerized application with a complex startup sequence. The application relies on a database service that might be Slow to respond on startup. How would you implement Liveness and Readiness probes to ensure the application iS healthy and available to users, even during startup?

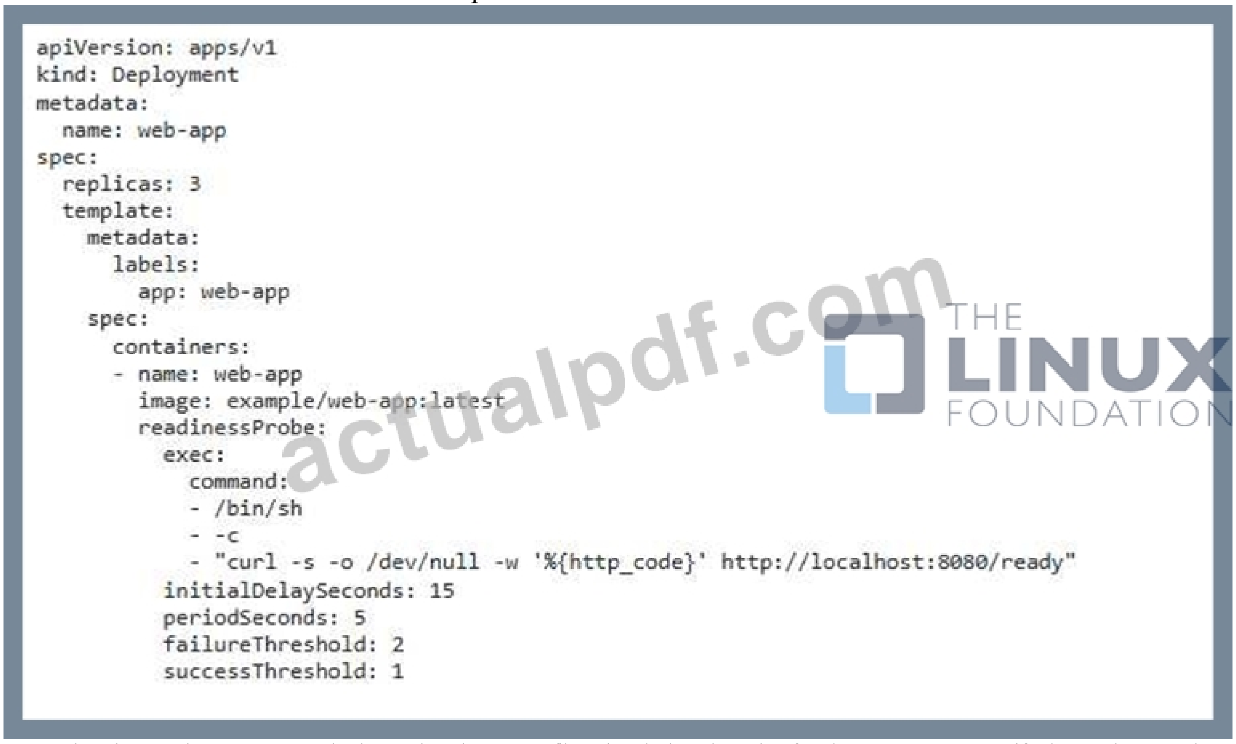
Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Define Liveness Probe:
- Create a 'livenessProbe' within the 'containers' section of your 'web-app' Deployment YAML-
- Choose a probe type appropriate tor your application. In this case, since the startup is complex, use an 'exec' probe.
- Specify the command to execute. This should be a simple command that checks if the application is up and ready to handle requests.
- Set 'initialDelaySecondS and 'periodSeconds' to provide sufficient time for the application to start.
- Configure 'failureThreshold' and 'successThreshold' to define how many tailed or successful probes trigger a pod restart.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: example/web-app:latest
        livenessProbe:
          exec:
            command:
            - /bin/sh
            - -c
            - "curl -s -o /dev/null -w '%{http_code}' http://localhost:8080/health"
          initialDelaySeconds: 30
          periodSeconds: 10
          failureThreshold: 3
          successThreshold: 2
```

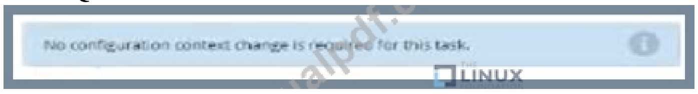
2. Define Readiness Probe: - Create a 'readinessProbe' Within the 'containers' section of your 'web-apps Deployment YAML. - Use the same 'exec' probe type as for the liveness probe. - Specify a command that checks it the application is ready to serve traffic. - Set 'initialDelaySeconds' and 'periodSeconds' to control the frequency and delay of the probe. - Configure 'failureThreshold' and 'successThreshold' to handle failed or successful probe results.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: example/web-app:latest
        readinessProbe:
          exec:
            command:
            - /bin/sh
            - -c
            - "curl -s -o /dev/null -w '%{http_code}' http://localhost:8080/ready"
          initialDelaySeconds: 15
          periodSeconds: 5
          failureThreshold: 2
          successThreshold: 1
```

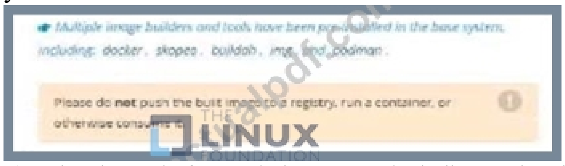
3. Deploy the Deployment: - Apply the updated YAML file using 'kubectl apply -f web-app.yamr 4. Verify the Probes: - Observe the pod logs using 'kubectl logs to see when liveness and readiness probes are executed. - Use 'kubectl get pods -I app=web-app' to check the status of pods and see how liveness and readiness probes affect the pod's health and availability. 5. Test the Application: - Send requests to the application to verify that it is healthy and responsive, even during startup. - Liveness Probe: The ' livenessProbe' checks if the application is still healthy and running. If the probe fails repeatedly, the Kubernetes will restart the pod to fix the issue. This ensures that unhealthy pods are removed and replaced with healthy ones. - Readiness Probe: The 'readinessproa' cnecks it the application iS ready to receive traffic. This allows Kubernetes to delay sending traffic to a pod until it is fully initialized and prepared to serve requests. It helps prevent users from encountering errors during startup. By using both liveness and readiness probes, you can ensure your application is healthy and available to users, even during complex startup sequences.,

NEW QUESTION # 96



No configuration context change is required for this task.

Task:
A Dockerfile has been prepared at -/human-stork/build/Dockerfile
1) Using the prepared Dockerfile, build a container image with the name macque and lag 3.0. You may install and use the tool of your choice.



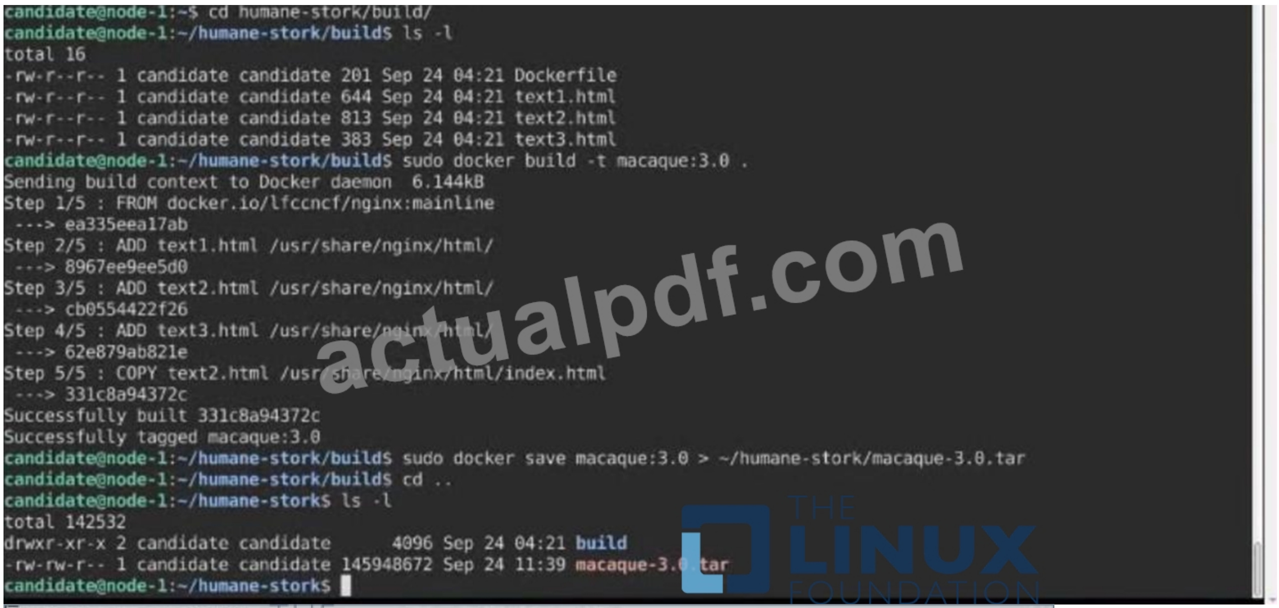
Multiple image builders and tools have been pre-installed in the base system, including: docker, skopeo, buildah, img, and podman.

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

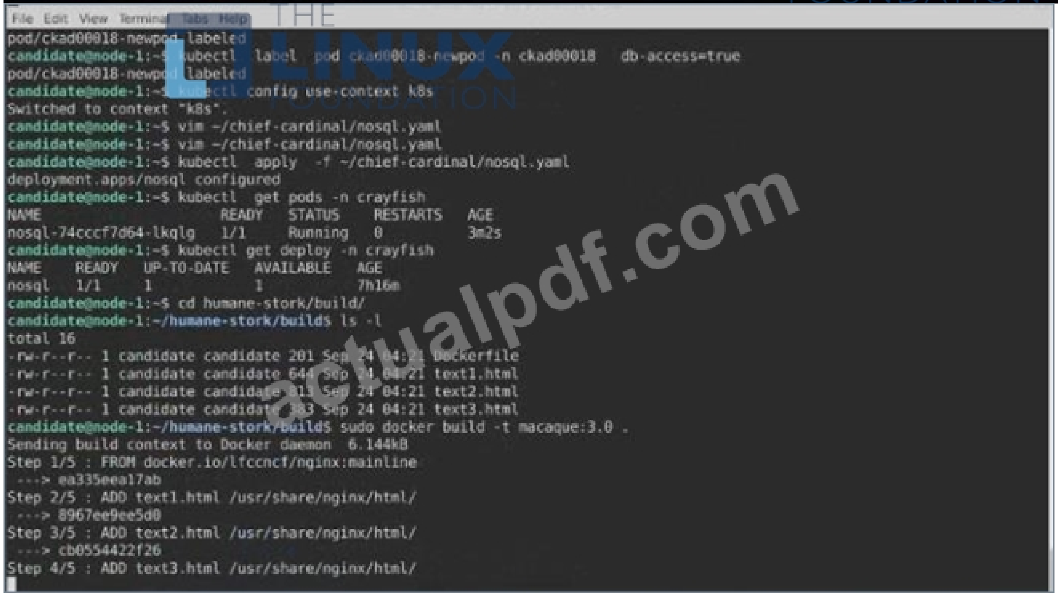
2) Using the tool of your choice export the built container image in OC-format and store it at -/human stork/macque 3.0 tar See the solution below.

Answer:

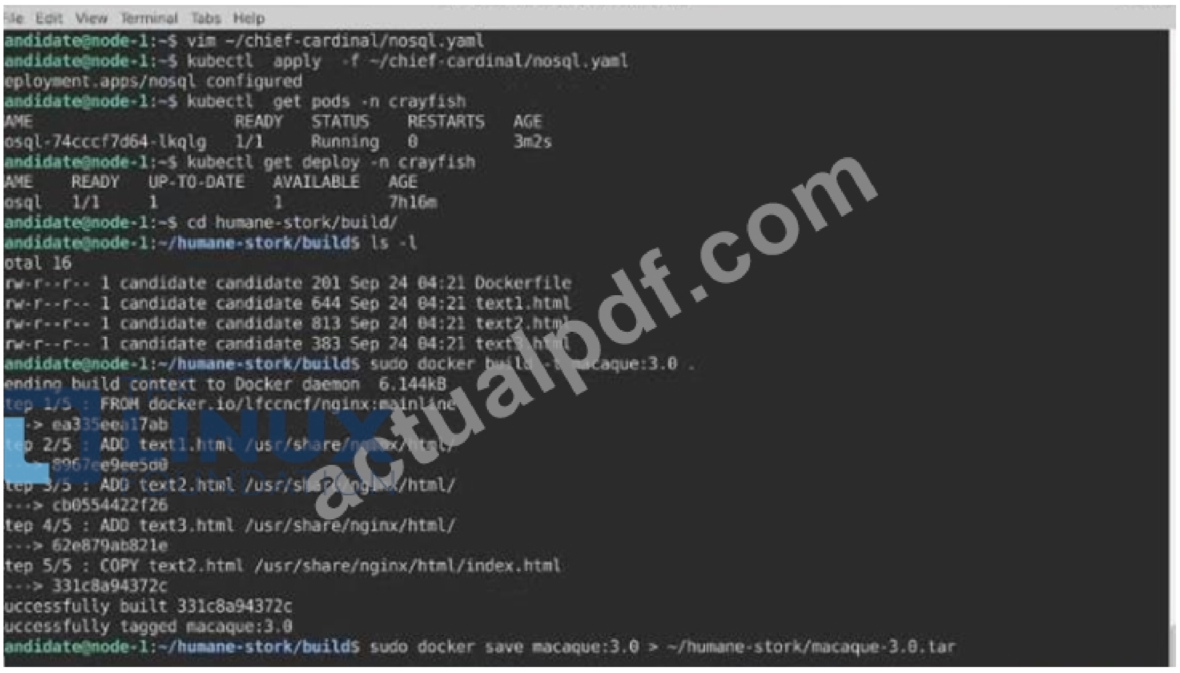
Explanation:
Explanation
Solution:



```
candidate@node-1:~$ cd humane-stork/build/
candidate@node-1:~/humane-stork/build$ ls -l
total 16
-rw-r--r-- 1 candidate candidate 201 Sep 24 04:21 Dockerfile
-rw-r--r-- 1 candidate candidate 644 Sep 24 04:21 text1.html
-rw-r--r-- 1 candidate candidate 813 Sep 24 04:21 text2.html
-rw-r--r-- 1 candidate candidate 383 Sep 24 04:21 text3.html
candidate@node-1:~/humane-stork/build$ sudo docker build -t macaque:3.0 .
Sending build context to Docker daemon  6.144kB
Step 1/5 : FROM docker.io/lfccncf/nginx:mainline
 ---> ea335eea17ab
Step 2/5 : ADD text1.html /usr/share/nginx/html/
 ---> 8967ee9ee5d0
Step 3/5 : ADD text2.html /usr/share/nginx/html/
 ---> cb0554422f26
Step 4/5 : ADD text3.html /usr/share/nginx/html/
 ---> 62e879ab821e
Step 5/5 : COPY text2.html /usr/share/nginx/html/index.html
 ---> 331c8a94372c
Successfully built 331c8a94372c
Successfully tagged macaque:3.0
candidate@node-1:~/humane-stork/build$ sudo docker save macaque:3.0 > ~/humane-stork/macaque-3.0.tar
candidate@node-1:~/humane-stork/build$ cd ..
candidate@node-1:~/humane-stork$ ls -l
total 142532
drwxr-xr-x 2 candidate candidate      4096 Sep 24 04:21 build
-rw-rw-r-- 1 candidate candidate 145948672 Sep 24 11:39 macaque-3.0.tar
candidate@node-1:~/humane-stork$
```



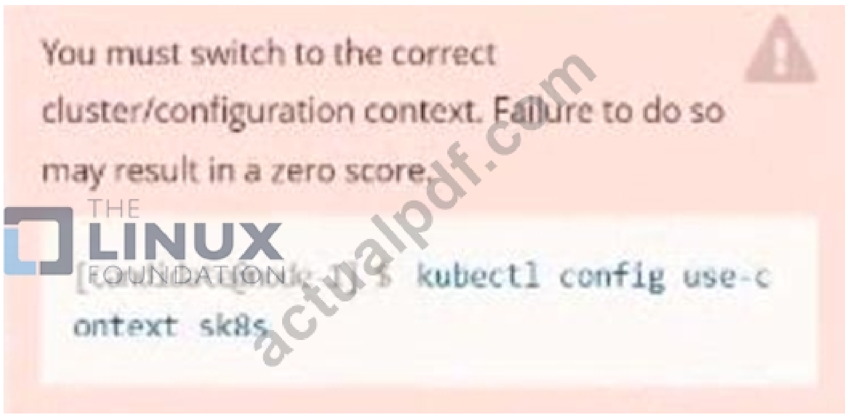
```
pod/ckad00018-newpod labeled
candidate@node-1:~$ kubectl  label  pod ckad00018-newpod -n ckad00018   db-access=true
pod/ckad00018-newpod labeled
candidate@node-1:~$ kubectl config use-context k8s
Switched to context "k8s".
candidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
candidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
candidate@node-1:~$ kubectl  apply  -f ~/chief-cardinal/nosql.yaml
deployment.apps/nosql configured
candidate@node-1:~$ kubectl  get pods -n crayfish
NAME                    READY   STATUS    RESTARTS   AGE
nosql-74cccf7d64-lkqlg   1/1     Running   0          3m2s
candidate@node-1:~$ kubectl get deploy -n crayfish
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nosql   1/1     1            1           7h16m
candidate@node-1:~$ cd humane-stork/build/
candidate@node-1:~/humane-stork/build$ ls -l
total 16
-rw-r--r-- 1 candidate candidate 201 Sep 24 04:21 Dockerfile
-rw-r--r-- 1 candidate candidate 644 Sep 24 04:21 text1.html
-rw-r--r-- 1 candidate candidate 813 Sep 24 04:21 text2.html
-rw-r--r-- 1 candidate candidate 383 Sep 24 04:21 text3.html
candidate@node-1:~/humane-stork/build$ sudo docker build -t macaque:3.0 .
Sending build context to Docker daemon  6.144kB
Step 1/5 : FROM docker.io/lfccncf/nginx:mainline
 ---> ea335eea17ab
Step 2/5 : ADD text1.html /usr/share/nginx/html/
 ---> 8967ee9ee5d0
Step 3/5 : ADD text2.html /usr/share/nginx/html/
 ---> cb0554422f26
Step 4/5 : ADD text3.html /usr/share/nginx/html/
```



```
andidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
andidate@node-1:~$ kubectl  apply  -f ~/chief-cardinal/nosql.yaml
eployment.apps/nosql configured
andidate@node-1:~$ kubectl  get pods -n crayfish
AME                    READY   STATUS    RESTARTS   AGE
osql-74cccf7d64-lkqlg   1/1     Running   0          3m2s
andidate@node-1:~$ kubectl get deploy -n crayfish
AME    READY   UP-TO-DATE   AVAILABLE   AGE
osql   1/1     1            1           7h16m
andidate@node-1:~$ cd humane-stork/build/
andidate@node-1:~/humane-stork/build$ ls -l
otal 16
rw-r--r-- 1 candidate candidate 201 Sep 24 04:21 Dockerfile
rw-r--r-- 1 candidate candidate 644 Sep 24 04:21 text1.html
rw-r--r-- 1 candidate candidate 813 Sep 24 04:21 text2.html
rw-r--r-- 1 candidate candidate 383 Sep 24 04:21 text3.html
andidate@node-1:~/humane-stork/build$ sudo docker build -t macaque:3.0 .
ending build context to Docker daemon  6.144kB
tep 1/5 : FROM docker.io/lfccncf/nginx:mainline
 --> ea335eea17ab
tep 2/5 : ADD text1.html /usr/share/nginx/html/
 --> 8967ee9ee5d0
tep 3/5 : ADD text2.html /usr/share/nginx/html/
 ---> cb0554422f26
tep 4/5 : ADD text3.html /usr/share/nginx/html/
 ---> 62e879ab821e
tep 5/5 : COPY text2.html /usr/share/nginx/html/index.html
 ---> 331c8a94372c
uccessfully built 331c8a94372c
uccessfully tagged macaque:3.0
andidate@node-1:~/humane-stork/build$ sudo docker save macaque:3.0 > ~/humane-stork/macaque-3.0.tar
```

NEW QUESTION # 97

Context



You must switch to the correct cluster/configuration context. Failure to do so may result in a zero score.

```
[candidate@node-1] $ kubectl config use-context sk8s
```

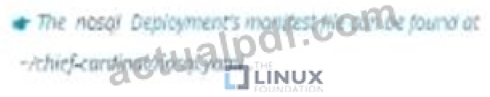
Task:

The pod for the Deployment named nosql in the craytisn namespace fails to start because its container runs out of resources.

Update the nosol Deployment so that the Pod:

1) Request 160M of memory for its Container

2) Limits the memory to half the maximum memory constraint set for the crayfah name space.

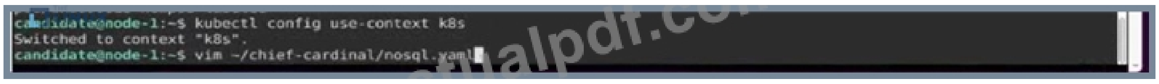


The nosql Deployment's manifest file can be found at ~/chief-cardinal/nosql.yaml

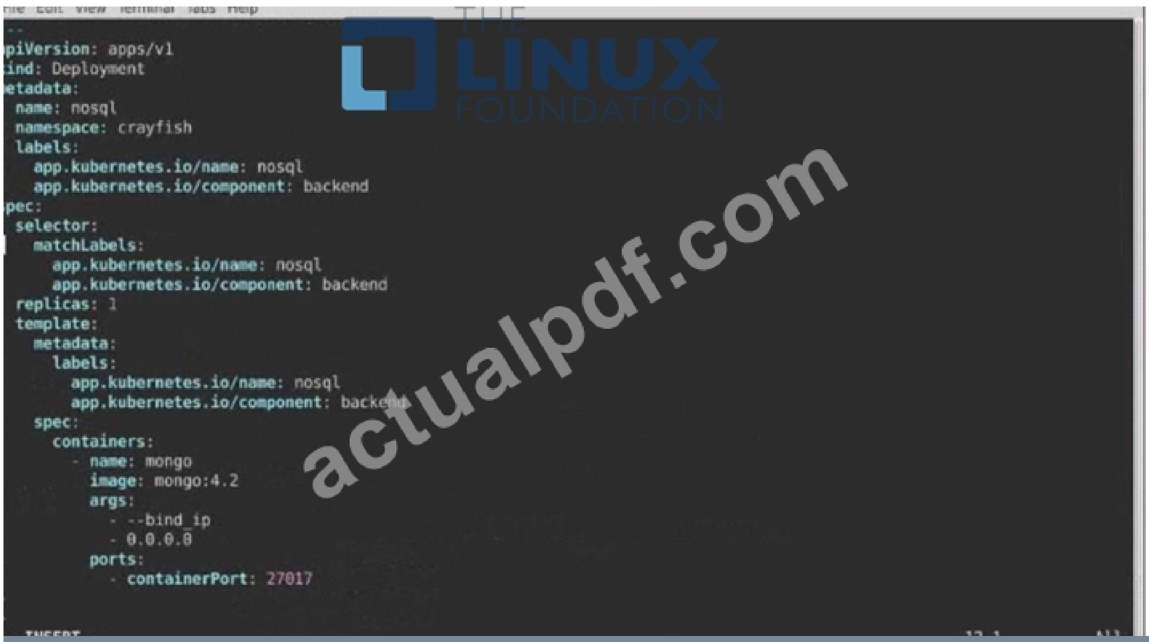
Answer:

Explanation:

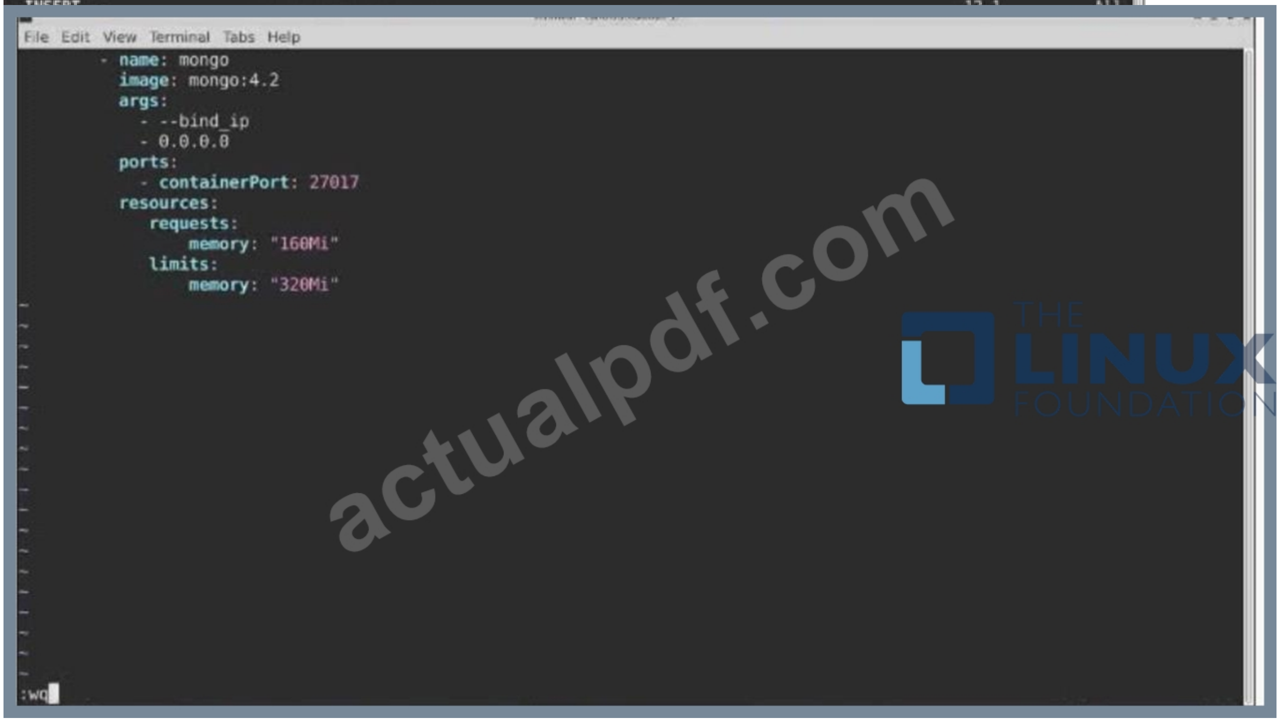
Solution:



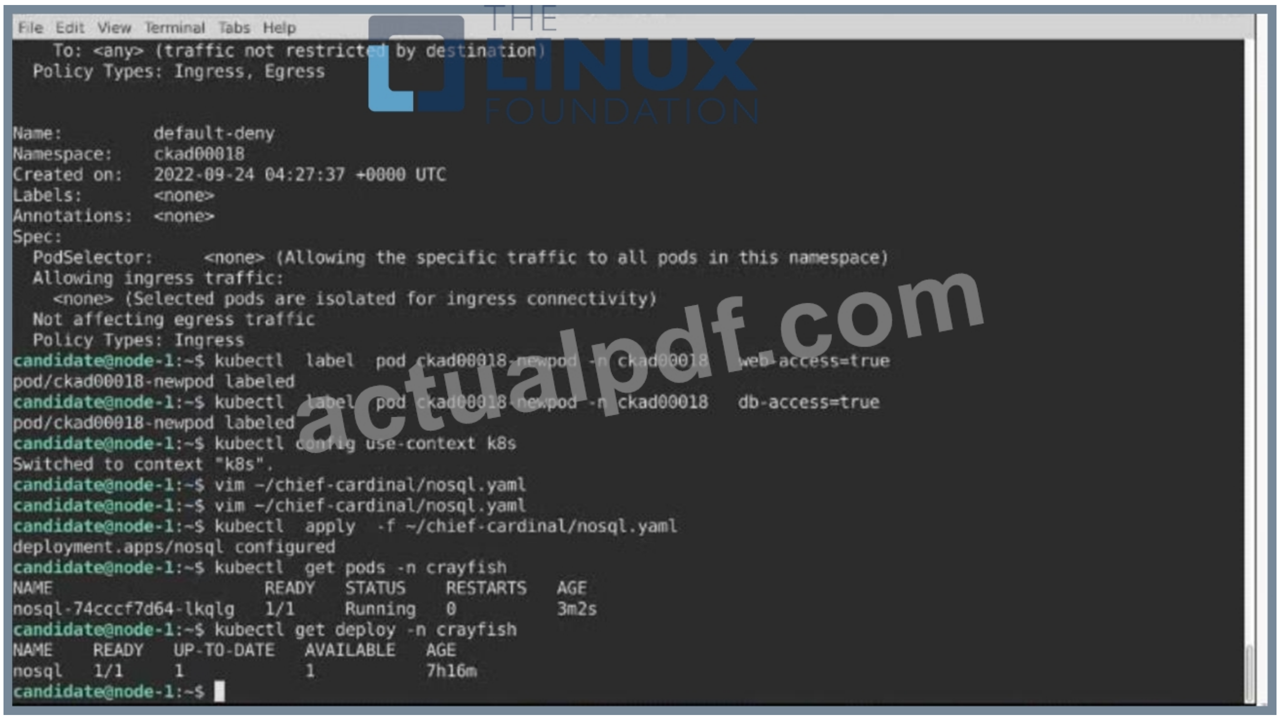
```
candidate@node-1:~$ kubectl config use-context k8s
Switched to context "k8s".
candidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
```



```
--
piVersion: apps/v1
ind: Deployment
etadata:
 name: nosql
 namespace: crayfish
 labels:
   app.kubernetes.io/name: nosql
   app.kubernetes.io/component: backend
pec:
 selector:
   matchLabels:
     app.kubernetes.io/name: nosql
     app.kubernetes.io/component: backend
 replicas: 1
 template:
   metadata:
     labels:
       app.kubernetes.io/name: nosql
       app.kubernetes.io/component: backend
   spec:
     containers:
       - name: mongo
         image: mongo:4.2
         args:
           - --bind_ip
           - 0.0.0.0
         ports:
           - containerPort: 27017

 INSERT
```



```
          - name: mongo
            image: mongo:4.2
            args:
              - --bind_ip
              - 0.0.0.0
            ports:
              - containerPort: 27017
            resources:
               requests:
                   memory: "160Mi"
               limits:
                   memory: "320Mi"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```



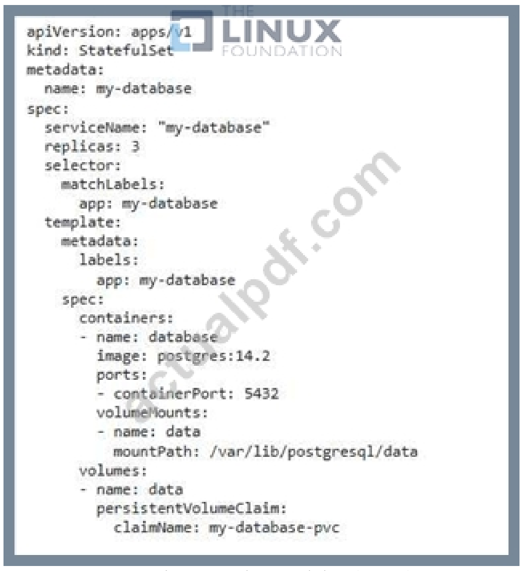
```
File  Edit  View  Terminal  Tabs  Help
    To: <any> (traffic not restricted by destination)
  Policy Types: Ingress, Egress


Name:         default-deny
Namespace:    ckad00018
Created on:   2022-09-24 04:27:37 +0000 UTC
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    <none> (Selected pods are isolated for ingress connectivity)
  Not affecting egress traffic
  Policy Types: Ingress
candidate@node-1:~$ kubectl  label  pod ckad00018-newpod -n ckad00018  web-access=true
pod/ckad00018-newpod labeled
candidate@node-1:~$ kubectl  label  pod ckad00018-newpod -n ckad00018  db-access=true
pod/ckad00018-newpod labeled
candidate@node-1:~$ kubectl config use-context k8s
Switched to context "k8s".
candidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
candidate@node-1:~$ vim ~/chief-cardinal/nosql.yaml
candidate@node-1:~$ kubectl  apply  -f ~/chief-cardinal/nosql.yaml
deployment.apps/nosql configured
candidate@node-1:~$ kubectl  get pods -n crayfish
NAME                     READY   STATUS    RESTARTS   AGE
nosql-74cccf7d64-lkqlg   1/1     Running   0          3m2s
candidate@node-1:~$ kubectl get deploy -n crayfish
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nosql   1/1     1            1           7h16m
candidate@node-1:~$
```

NEW QUESTION # 98

You are tasked with deploying a stateful application, a distributed database, that requires persistent storage and consistent ordering of pods. The application's pods need to communicate With each other using a specific port (5432). How would you configure a StatefulSet to achieve this?

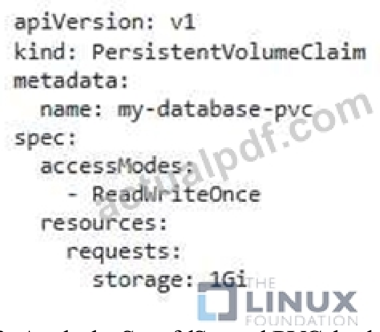
Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Create the StatefulSet YAML:



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-database
spec:
  serviceName: "my-database"
  replicas: 3
  selector:
    matchLabels:
      app: my-database
  template:
    metadata:
      labels:
        app: my-database
    spec:
      containers:
      - name: database
        image: postgres:14.2
        ports:
        - containerPort: 5432
        volumeMounts:
        - name: data
          mountPath: /var/lib/postgresql/data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: my-database-pvc
```

2. Create a PersistentVolumeClaim (PVC):



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-database-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

3. Apply the StatefulSet and PVC: bash kubectl apply -f statefulset.yaml kubectl apply -f pvc.yaml 4. Check the StatefulSet and Pods: bash kubectl get statefulsets my-database kubectl get pods -l app=my-database - StatefulSet This defines the desired state for the database pods, ensuring tneir order and persistent storage. - serviceName: This field defines the service name used to access the database instances. - replicas: Defines the desired number of database instances (3 in this example). - selector: Matches pods with the "app: my-database" label. - template: Defines the pod template to use for each instance. - containers: Contains the database container definition. - ports: Exposes the database's internal port (5432) to the outside world. - volumeMounts: Mounts the persistent volume claim to the container's storage directory. - volumes: Defines the volume to use, in this case, a persistent volume claim. - persistentVolumeClaim: Links the StatefulSet to the PVC- - PVC (my-database-pvc): Requests a persistent volume of 1 Gi for each database pod. This ensures data persistence between restarts. - accessM0des: ReadWriteOnce: Allows only one pod to access the volume at a time. - resources-requests-storage: Specifies the storage request for each PVC- This setup ensures that each database pod: - Has a unique name based on its ordinal position within the StatefulSet - Has persistent storage using the PVC. - Can communicate with otner pods through the defined service. - Maintains consistent ordering, essential for distributed database functionality

NEW QUESTION # 99
......

Exam CKAD Topic: https://www.actualpdf.com/CKAD_exam-dumps.html