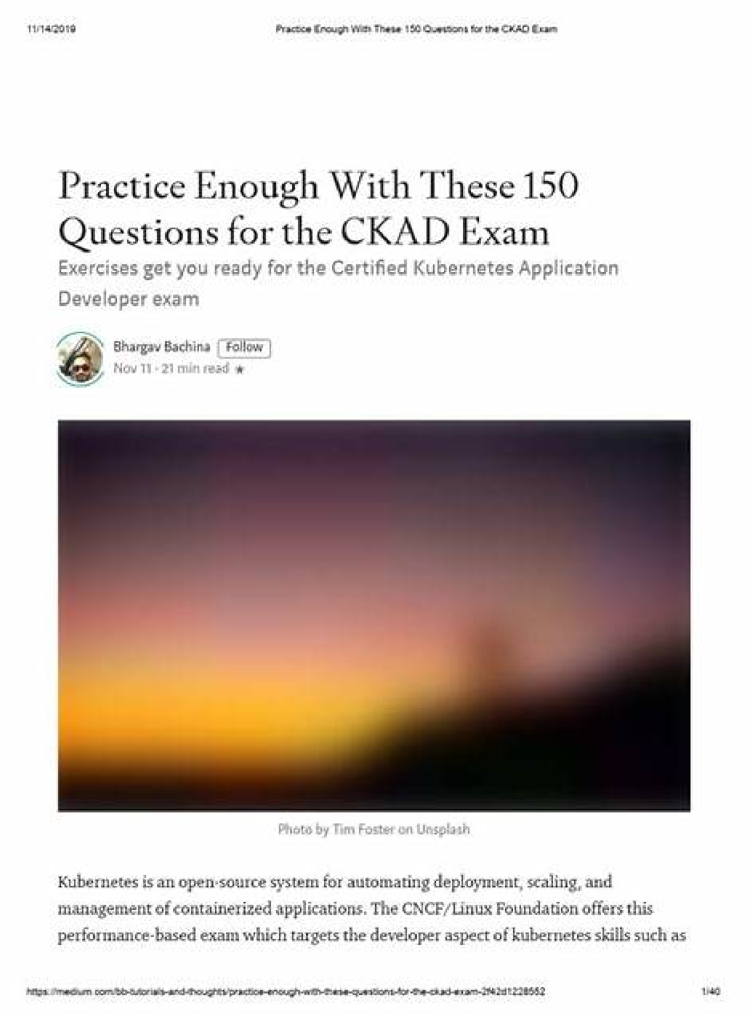
Real CKAD Questions - Remove Your Exam Fear



11/14/2019 Practice Enough With These 150 Questions for the CKAD Exam

Practice Enough With These 150 Questions for the CKAD Exam

Exercises get you ready for the Certified Kubernetes Application Developer exam

Bhargav Bachina Follow

Nov 11 · 21 min read ★

Photo by Tim Foster on Unsplash

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. The CNCF/Linux Foundation offers this performance-based exam which targets the developer aspect of kubernetes skills such as

https://medium.com/bb-tutorials-and-thoughts/practice-enough-with-these-questions-for-the-ckad-exam-2f42d1228552 1/40

BTW, DOWNLOAD part of Real4test CKAD dumps from Cloud Storage: https://drive.google.com/open?id=1OajZYCLnsZWpQsXf-duxg0qnufmoLAkW

The pressure is not terrible, and what is terrible is that you choose to evade it. You clearly have seen your own shortcomings, and you know that you really should change. Then, be determined to act! Buying our CKAD exam questions is the first step you need to take. And as long as you study with our CKAD Practice Guide, you will find that the exam is just a piece of cake and the certification is easy to get. With the certification, you will find your future is much brighter.

The CKAD Exam is aimed at developers who are already familiar with Kubernetes and have experience working with it. CKAD exam consists of a series of performance-based tasks that are designed to test the candidate's ability to use Kubernetes to deploy, manage, and scale containerized applications. The tasks are designed to simulate real-world scenarios that developers may encounter when working with Kubernetes. CKAD Exam is conducted online, and candidates have two hours to complete it. Upon successful completion of the exam, the candidate is awarded the CKAD certification, which is recognized by the industry as a standard for Kubernetes application development.

>> Reliable CKAD Dumps Ebook <<

Free PDF Quiz First-grade Linux Foundation CKAD - Reliable Linux Foundation Certified Kubernetes Application Developer Exam Dumps Ebook

Nowadays, using computer-aided software to pass the CKAD exam has become a new trend. Because the new technology enjoys a distinct advantage, that is convenient and comprehensive. In order to follow this trend, our company product such a CKAD exam questions that can bring you the combination of traditional and novel ways of studying. The passing rate of our study material is up to 99%. If you are not fortune enough to acquire the CKAD Certification at once, you can unlimitedly use our product at different discounts until you reach your goal and let your dream comes true.

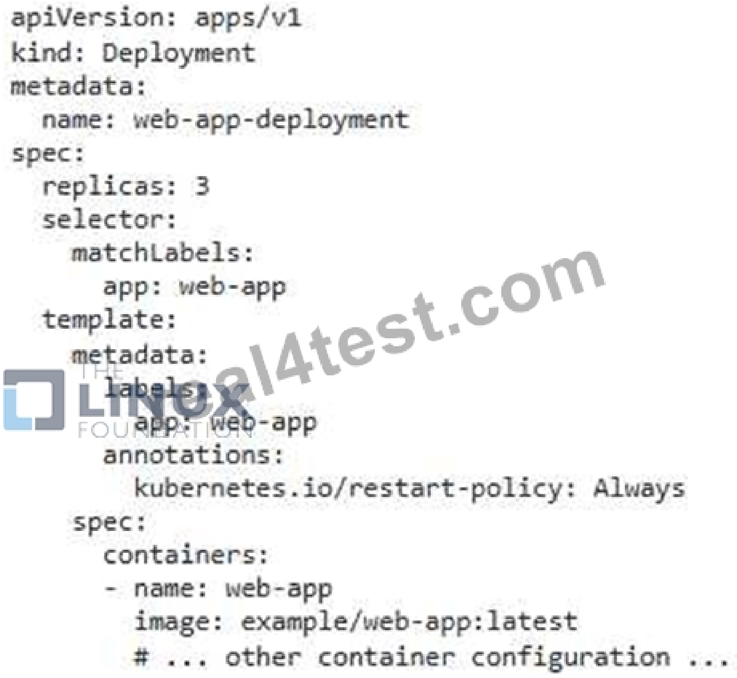
Linux Foundation CKAD certification exam is an online, proctored exam that can be taken from anywhere in the world. CKAD exam consists of a set of performance-based tasks that are designed to test the candidate's ability to use Kubernetes to deploy, manage and troubleshoot containerized applications. CKAD Exam is timed and candidates have 2 hours to complete the exam. To pass the exam, candidates must score 66% or higher.

Linux Foundation Certified Kubernetes Application Developer Exam Sample Questions (Q132-Q137):

NEW QUESTION # 132
You are running a web application that requires high availability and resilience. You have implemented a deployment using a Deployment object in Kubernetes, but you want to ensure that your application can automatically recover from pod failures. Design a strategy using annotations that will enable automatic pod restarts in case of application failures.

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Define an Annotation: Add an annotation called ' kubernetes-io/restart-policy' to your Deployment's 'spec-template-metadata' section, setting its value to 'Always'.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
      annotations:
        kubernetes.io/restart-policy: Always
    spec:
      containers:
      - name: web-app
        image: example/web-app:latest
        # ... other container configuration ...
```

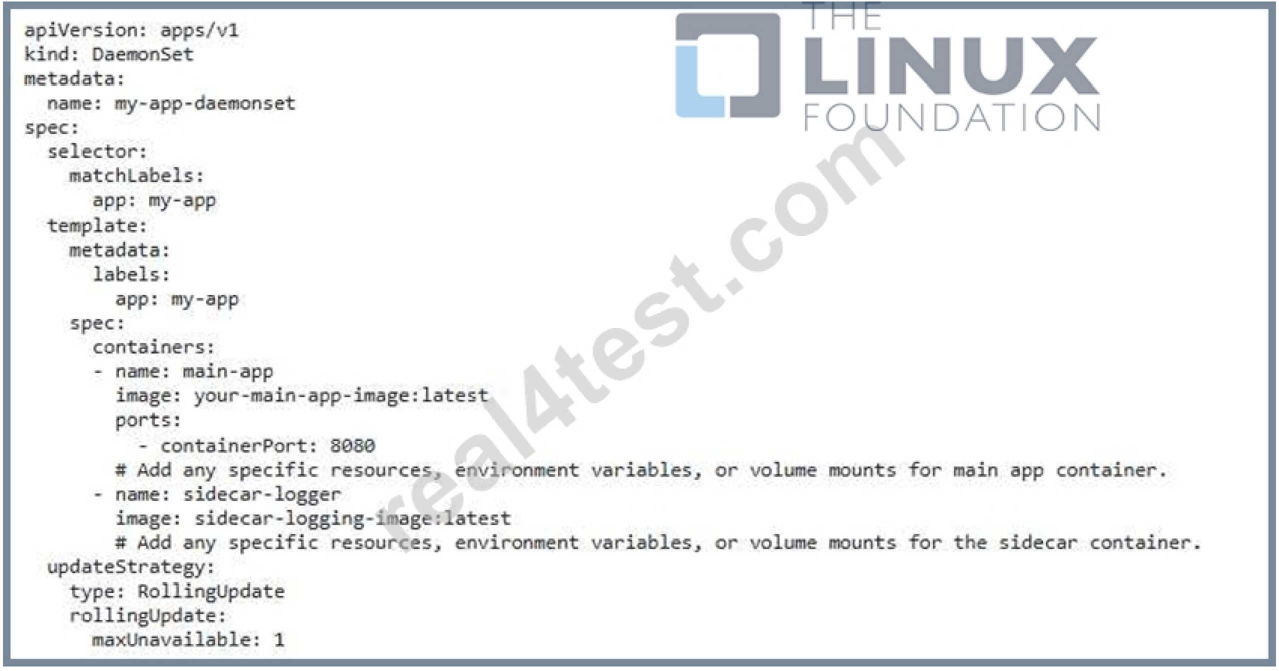
2. Trigger Application Failures: You can intentionally trigger failures in your pods to test the restart policy You can use 'kubectl exec' to run commands inside a pod and simulate an application failure using 'Pkill -f s. For example, you can run a command like 'Pkill -f web-app' to terminate the web app process. 3. Monitor Pod Restarts: Observe the pods in your deployment using 'kubectl get pods -l app=web-app' . You will see that Kubernetes automatically restarts pods where the application has failed, ensuring your application remains available. 4. Confirm Automatic Restart Verify the 'restartCount of the affected pods using 'kubectl describe pod This will show the number of times the pod has been restarted due to the application failure. 5. Alternative Restart Policies: While 'Always' is the default policy, you can also use other restart policies like 'onFailure' (restarts only it the pod exits due to an error) or 'Never' (doesn't restart tne pod regardless of the reason for failure). Use the ' kubernetes.wrestart-policy' annotation to set these alternative policies as needed for specific applications. ,

NEW QUESTION # 133
You have a web application that requires a specific sidecar container to perform certain tasks like logging and monitoring. You need to ensure that this sidecar container iS always running alongside your application pod, even it the main application pod restarts or iS

deleted and recreated. How would you achieve this using a DaemonSet in Kubernetes?

Answer:

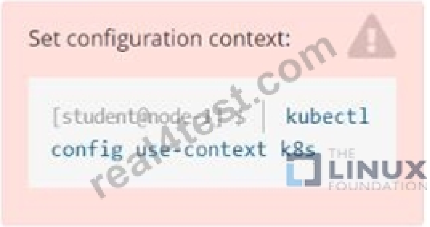
Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
I). Define the DaemonSet YAML: Create a YAML file that defines the DaemonSet configuration. This file will include the following key sections:
- Metadata Includes the name and labels for the DaemonSet.
- Spec: Defines the deployment details:
- Selector: Matches the labels of the pods that the DaemonSet should manage.
- Template: Contains the pod definition:
- Containers: Defines the main application container and the sidecar container.
- Ensure the sidecar container has appropriate resources and environment variables.
- Include any necessary ports or volume mounts for the sidecar container.
- UpdateStrategy: You might want to control the update strategy (RollingUpdate or Recreate) if you have multiple nodes.



```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-app-daemonset
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: main-app
        image: your-main-app-image:latest
        ports:
          - containerPort: 8080
        # Add any specific resources, environment variables, or volume mounts for main app container.
      - name: sidecar-logger
        image: sidecar-logging-image:latest
        # Add any specific resources, environment variables, or volume mounts for the sidecar container.
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
```

2. Create the Daemonset Apply the Daemonset YAML file to your Kubernetes cluster using 'kubectl apply -f daemonset.yamr. This will create the DaemonSet and stan deploying tne pods on each node. 3. Verify Deployment: Use 'kubectl get daemonsetS to check the status of the DaemonSet. Verify that the pods are running on each node. 4. Testing and Monitoring: - Restart or Delete the Main App Pod: Observe how the sidecar container continues running alongside the main app pod, even when the main pod is restarted or deleted and recreated. - Check Logs If your sidecar container is responsible for logging, use 'kubectl logs to check the logs from the sidecar container This approach ensures that the sidecar container remains in a ready state on each node and is always available to support your application pod, fulfilling the requirements for logging and monitoring even when the main pod restarts or is recreated.

NEW QUESTION # 134
Exhibit:



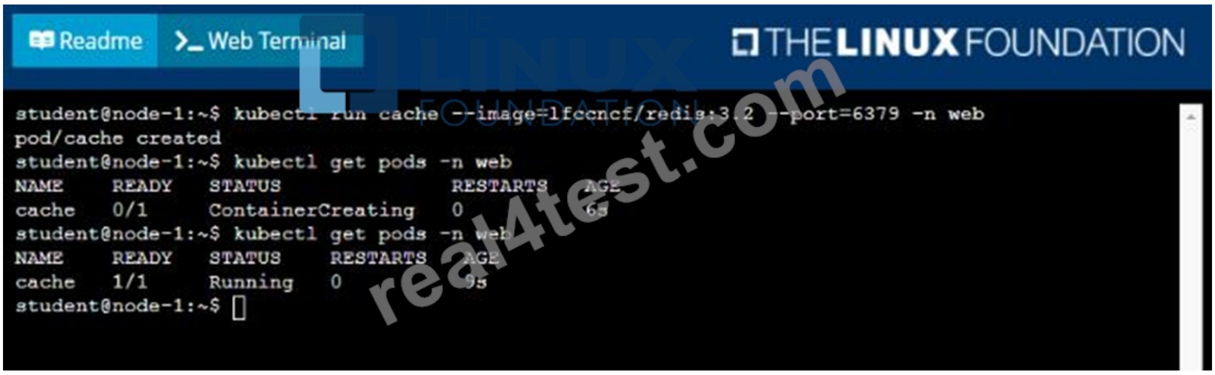
Set configuration context:

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

Context
A web application requires a specific version of redis to be used as a cache.
Task

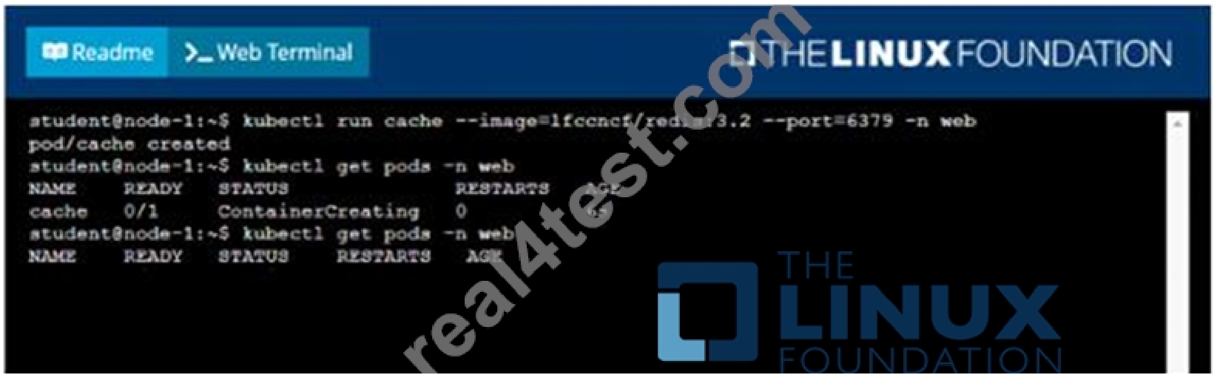
Create a pod with the following characteristics, and leave it running when complete:
* The pod must run in the web namespace.
The namespace has already been created
* The name of the pod should be cache
* Use the Ifccncf/redis image with the 3.2 tag
* Expose port 6379

- A. Solution:



```
student@node-1:~$ kubectl run cache --image=lfccncf/redis:3.2 --port=6379 -n web
pod/cache created
student@node-1:~$ kubectl get pods -n web
NAME    READY   STATUS              RESTARTS   AGE
cache   0/1     ContainerCreating   0          6s
student@node-1:~$ kubectl get pods -n web
NAME    READY   STATUS    RESTARTS   AGE
cache   1/1     Running   0          9s
student@node-1:~$
```

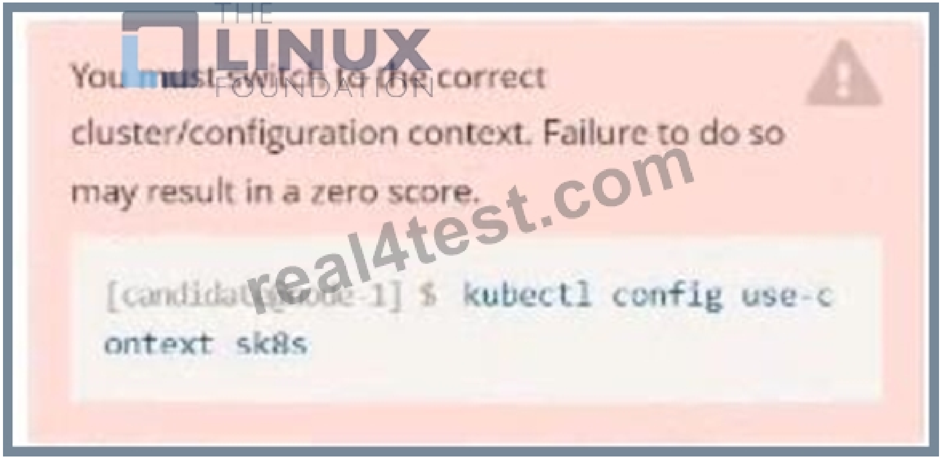
- B. Solution:



```
student@node-1:~$ kubectl run cache --image=lfccncf/redis:3.2 --port=6379 -n web
pod/cache created
student@node-1:~$ kubectl get pods -n web
NAME    READY   STATUS              RESTARTS   AGE
cache   0/1     ContainerCreating   0          6s
student@node-1:~$ kubectl get pods -n web
NAME    READY   STATUS    RESTARTS   AGE
```

Answer: A

NEW QUESTION # 135



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

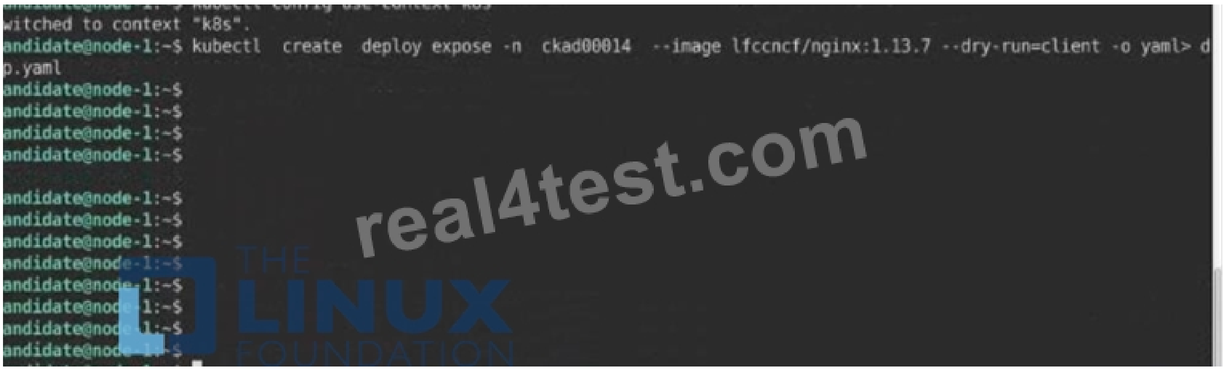
```
[candidate@node-1] $ kubectl config use-context sk8s
```

Task:
Create a Deployment named expose in the existing ckad00014 namespace running 6 replicas of a Pod. Specify a single container using the ifccncf/nginx: 1.13.7 image Add an environment variable named NGINX_PORT with the value 8001 to the container then expose port
8001

Answer:

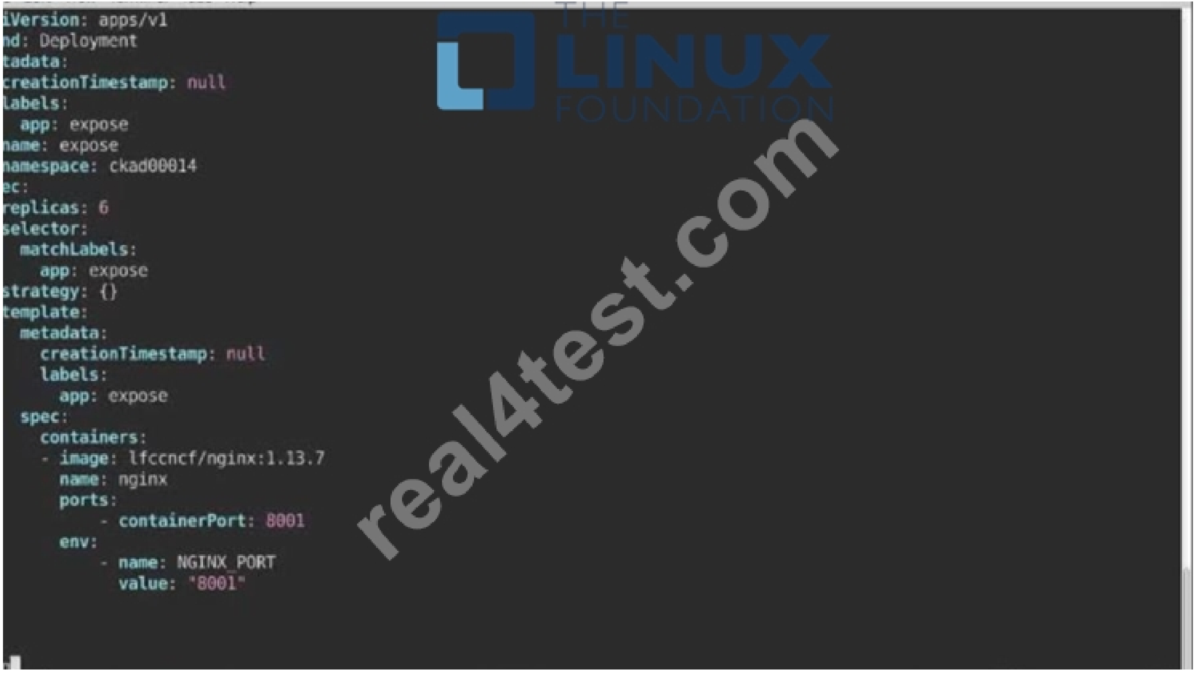
Explanation:

See the solution below.
Explanation
Solution:



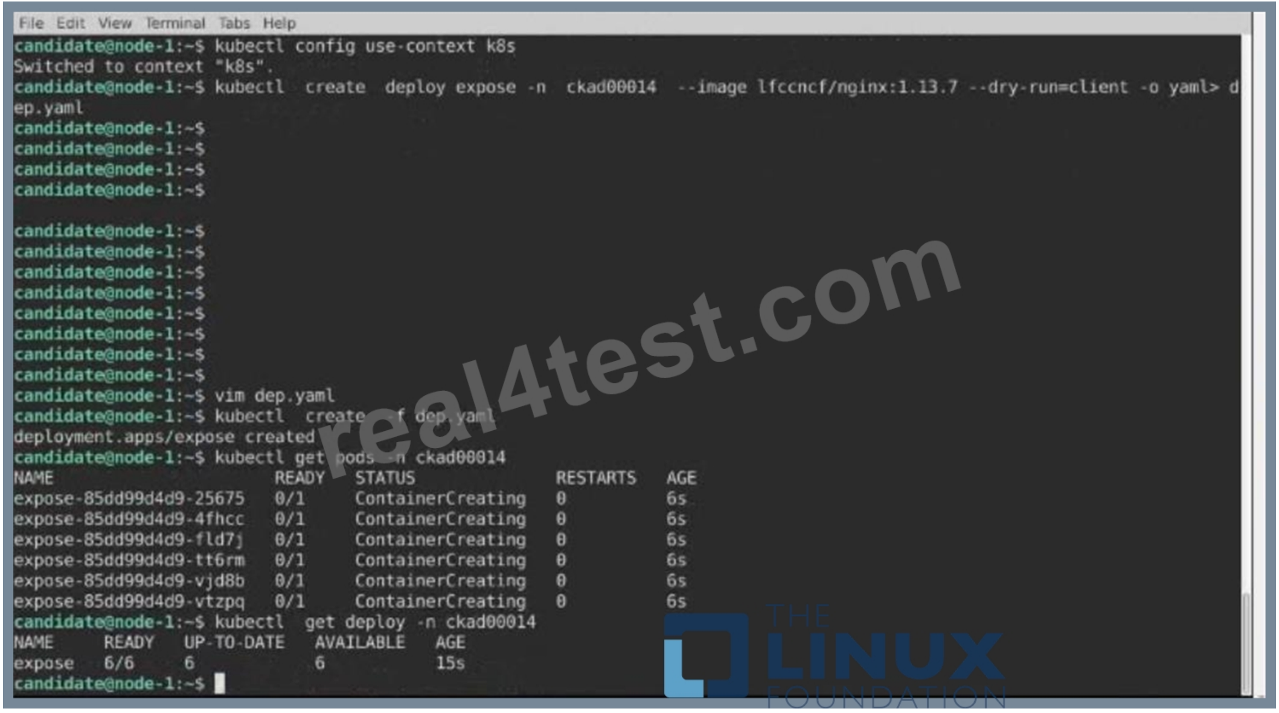
```
witched to context "k8s".
andidate@node-1:~$ kubectl  create  deploy expose -n  ckad00014  --image lfccncf/nginx:1.13.7 --dry-run=client -o yaml> d
p.yaml
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$

andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
andidate@node-1:~$
```

Text Description automatically generated



```
iVersion: apps/v1
nd: Deployment
tadata:
creationTimestamp: null
labels:
  app: expose
name: expose
namespace: ckad00014
ec:
replicas: 6
selector:
  matchLabels:
    app: expose
strategy: {}
template:
  metadata:
    creationTimestamp: null
    labels:
      app: expose
  spec:
    containers:
    - image: lfccncf/nginx:1.13.7
      name: nginx
      ports:
          - containerPort: 8001
      env:
          - name: NGINX_PORT
            value: "8001"
```

Text Description automatically generated



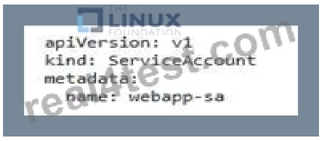
```
candidate@node-1:~$ kubectl config use-context k8s
Switched to context "k8s".
candidate@node-1:~$ kubectl  create  deploy expose -n  ckad00014  --image lfccncf/nginx:1.13.7 --dry-run=client -o yaml> d
ep.yaml
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$

candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$
candidate@node-1:~$ vim dep.yaml
candidate@node-1:~$ kubectl  create  -f dep.yaml
deployment.apps/expose created
candidate@node-1:~$ kubectl get pods -n ckad00014
NAME                     READY   STATUS              RESTARTS   AGE
expose-85dd99d4d9-25675  0/1     ContainerCreating   0          6s
expose-85dd99d4d9-4fhcc  0/1     ContainerCreating   0          6s
expose-85dd99d4d9-fld7j  0/1     ContainerCreating   0          6s
expose-85dd99d4d9-tt6rm  0/1     ContainerCreating   0          6s
expose-85dd99d4d9-vjd8b  0/1     ContainerCreating   0          6s
expose-85dd99d4d9-vtzpq  0/1     ContainerCreating   0          6s
candidate@node-1:~$ kubectl  get deploy -n ckad00014
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
expose   6/6     6            6           15s
candidate@node-1:~$
```

NEW QUESTION # 136

You are tasked with setting up a secure Kubernetes cluster for a web application. The application has sensitive data that must be protected. You need to configure a mecnanism to restrict access to the application's pods based on user identities. Describe a method to achieve this using Kubernetes RBAC and Service Accounts, ensuring that only authorized users can access specific pods.

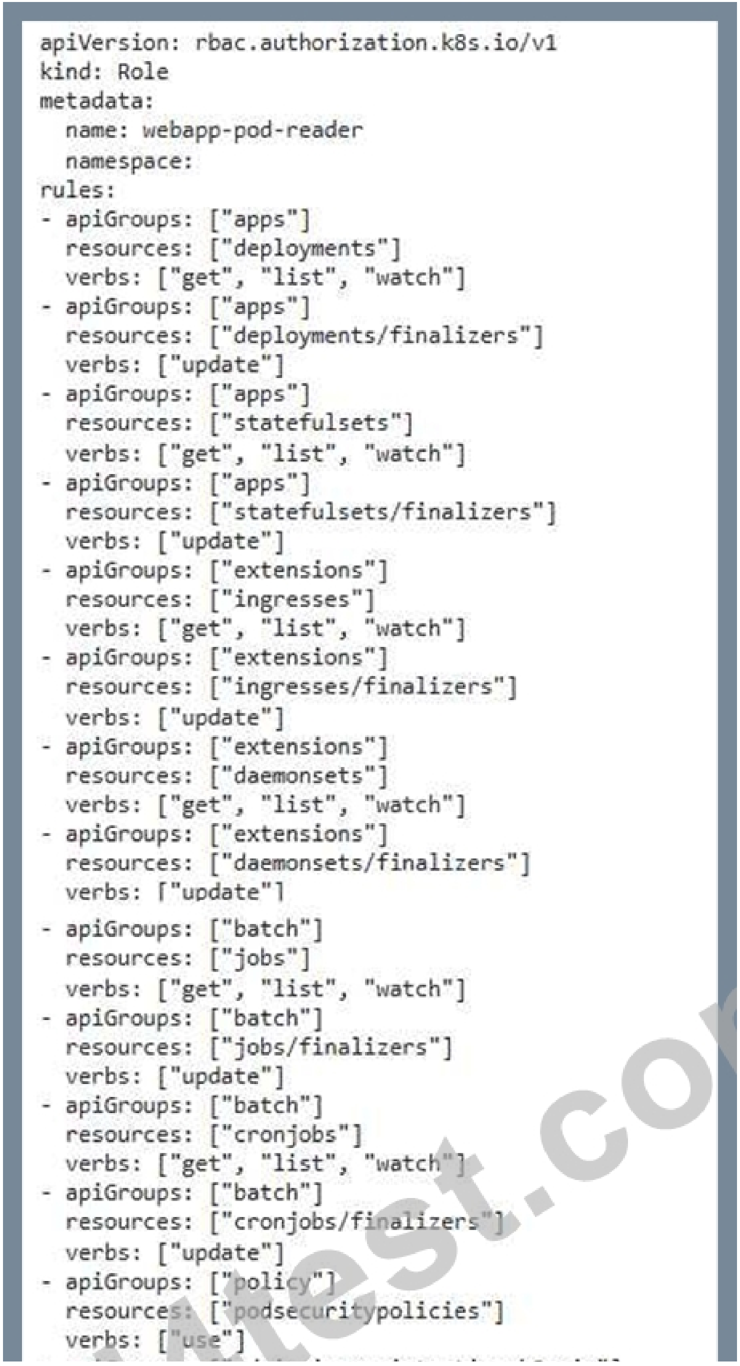
Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Create a Service Account
- Create a Service Account for the application:

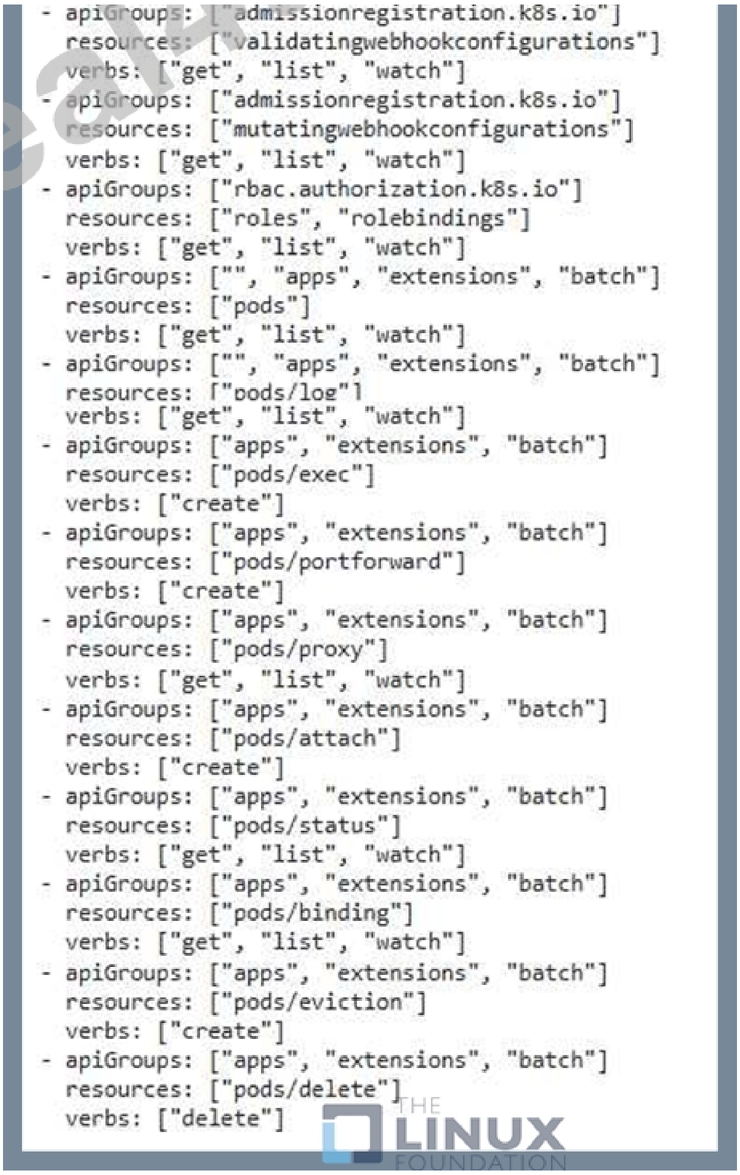


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: webapp-sa
```

- Apply the Service Account configuratiom basn kubectl apply -f webapp-sa.yaml 2. Create a Role: - Define a Role that grants access to the specific pods:

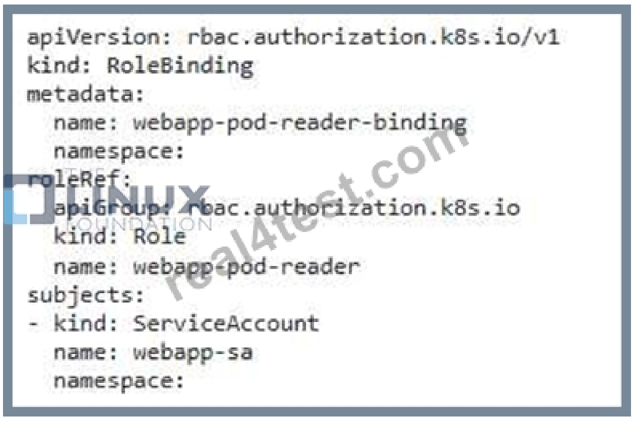


```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: webapp-pod-reader
  namespace:
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments/finalizers"]
  verbs: ["update"]
- apiGroups: ["apps"]
  resources: ["statefulsets"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["statefulsets/finalizers"]
  verbs: ["update"]
- apiGroups: ["extensions"]
  resources: ["ingresses"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["ingresses/finalizers"]
  verbs: ["update"]
- apiGroups: ["extensions"]
  resources: ["daemonsets"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["daemonsets/finalizers"]
  verbs: ["update"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch"]
  resources: ["jobs/finalizers"]
  verbs: ["update"]
- apiGroups: ["batch"]
  resources: ["cronjobs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs/finalizers"]
  verbs: ["update"]
- apiGroups: ["policy"]
  resources: ["podsecuritypolicies"]
  verbs: ["use"]
```



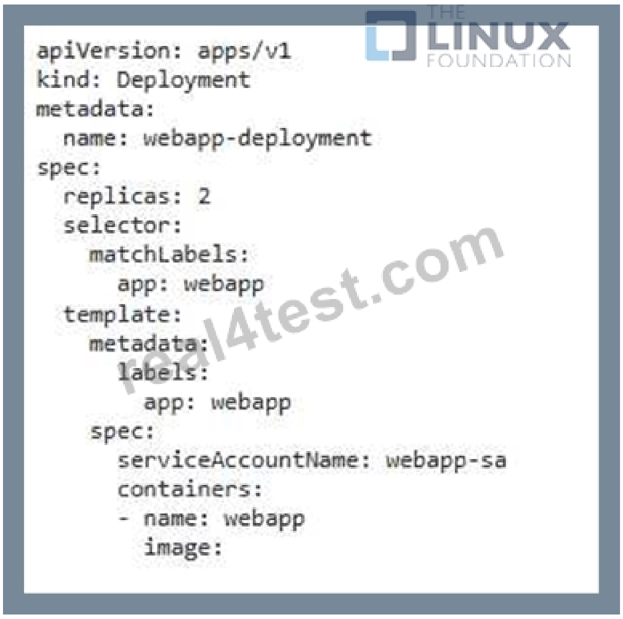
```
- apiGroups: ["admissionregistration.k8s.io"]
  resources: ["validatingwebhookconfigurations"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["admissionregistration.k8s.io"]
  resources: ["mutatingwebhookconfigurations"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["", "apps", "extensions", "batch"]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["", "apps", "extensions", "batch"]
  resources: ["pods/log"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/exec"]
  verbs: ["create"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/portforward"]
  verbs: ["create"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/proxy"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/attach"]
  verbs: ["create"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/status"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/binding"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/eviction"]
  verbs: ["create"]
- apiGroups: ["apps", "extensions", "batch"]
  resources: ["pods/delete"]
  verbs: ["delete"]
```

- Apply the Role configuratiom bash kubectl apply -f webapp-pod-reader.yaml 3. Create a RoleBinding: - Bind the Role to the Service Account



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: webapp-pod-reader-binding
  namespace:
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: webapp-pod-reader
subjects:
- kind: ServiceAccount
  name: webapp-sa
  namespace:
```

- Apply the RoleBinding configuration: bash kubectl apply -f webapp-pod-reader-binding_yaml 4. Configure the Application: - When deploying the application, specify the Service Account:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      serviceAccountName: webapp-sa
      containers:
      - name: webapp
        image:
```

5. Verify Access: - Use the 'kubectr command with the Service Account's credentials to verify that only authorized users can access the application's pods: bash kubectl -service-account=webapp-sa get pods -n This setup utilizes Kubernetes RBAC to control access to the application's pods. - The Service Account acts as an identity for the application. - The Role defines the permissions granted to the Service Account, specifically allowing access to the pods. - The RoleBinding associates the Role with the Service Account, linking the permissions to the identity. - When the application is deployed witn tne specified Service Account, it inherits the permissions defined in the RoleBinding. This ensures that only users with the necessary credentials (associated with the Service Account) can access and interact with the application's pods, safeguarding sensitive data.

NEW QUESTION # 137

......

New CKAD Test Tips: https://www.real4test.com/CKAD_real-exam.html search for ▶ CKAD ◀ for free download on www.prep4away.com Online CKAD Version

- study.stcs.edu.np, www.stes.tyc.edu.tw, pedulihati.yukcollab.com, daotao.wisebusiness.edu.vn, infodots.in, learnsphere.co.in, www.jamieholroydguitar.com, www.stes.tyc.edu.tw, omegaglobeacademy.com, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, Disposable vapes