Pass Guaranteed 2026 High Hit-Rate CKAD: Linux Foundation Certified Kubernetes Application Developer Exam Dumps

BTW, DOWNLOAD part of Dumpexams CKAD dumps from Cloud Storage: https://drive.google.com/open?id=1oEfj7mMiKsVLqQGIDopPCzAsrOHkxxTe

With these real CKAD Questions, you can prepare for the test while sitting on a couch in your lounge. Whether you are at home or traveling anywhere, you can do CKAD exam preparation with our Linux Foundation CKAD dumps. CKAD test candidates with different learning needs can use our three formats to meet their needs and prepare for the Linux Foundation CKAD test successfully in one go. Read on to check out the features of these three formats.

The CKAD Exam is specifically designed for developers who are already familiar with Kubernetes and have experience in developing and deploying applications on Kubernetes clusters. CKAD exam is designed to test the practical skills of developers in designing, deploying, and managing Kubernetes-based applications, and it requires candidates to demonstrate their proficiency in using Kubernetes APIs, tools, and resources.

>> CKAD Dumps <<

Web-Based Linux Foundation CKAD Practice Test - Compatible with All Major Browsers

It is understandable that different people have different preference in terms of CKAD study guide. Taking this into consideration, and in order to cater to the different requirements of people from different countries in the international market, we have prepared three kinds of versions of our CKAD Preparation questions in this website, namely, PDF version, online engine and software version, and you can choose any one of them as you like. No matter you buy any version of our CKAD exam questions, you will get success on your exam!

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%
- Services & Networking 13%
- State Persistence 8%
- Multi-Container Pods 10%

Linux Foundation Certified Kubernetes Application Developer Exam Sample Questions (Q115-Q120):

NEW QUESTION # 115
You are building a microservices architecture for a web application. One of your services handles user authentication. To ensure the service remains available even if one of the pods fails, you need to implement a high-availability solution. Design a deployment strategy for the authentication service that utilizes Kubernetes features to achieve high availability and fault tolerance.

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Deploy as a StatefulSet:
- Use a StatefulSet to deploy your authentication service. StatefulSets maintain persistent storage and unique identities for each pod, ensuring that data is preserved and the service can recover from failures without losing state.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: auth-service-statefulset
spec:
  serviceName: auth-service
  replicas: 3
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
    spec:
      containers:
      - name: auth-service
        image: your-image-repo:latest
        ports:
        - containerPort: 8080
      volumeClaimTemplates:
      - metadata:
          name: auth-data
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 1Gi
```


2. I-Ise Persistent Volumes: - Provision persistent volumes for each pod in the StatefulSet to store sensitive data like user credentials or session information. This ensures that the data persists even if a pod iS restarted or replaced. 3. Configure a Service with Load Balancing: - Create a Service that uses a load balancer (like a Kubernetes Ingress or external load balancer) to distribute traffic across the replicas of your authentication service. This ensures that requests are evenly distributed, even if some pods are down.

```
apiVersion: v1
kind: Service
metadata:
  name: auth-service
spec:
  type: LoadBalancer
  selector:
    app: auth-service
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

4. Implement Health Checks: - Set up liveness and readiness probes for the authentication service. Liveness probes ensure that unhealthy pods are restarted, while readiness probes ensure that only nealtny pods receive traffic. 5. Enable TLS/SSL: - Secure your authentication service with TLS/SSL to protect sensitive user data during communication. You can use certificates issued by a certificate authority (CA) or self-signed certificates for development environments. 6. Consider a Distributed Cache: - For improved performance and scalability, consider using a distributed cache like Redis or Memcached to store frequently accessed data, such as user authentication tokens. This can reduce the load on the authentication service and improve user response times.

NEW QUESTION # 116
You are managing a Kubernetes cluster running a highly-available application that uses a custom resource called 'Orders. The 'orders resource is created and managed by a custom controller that ensures the order processing workflow runs flawlessly. However, the 'order' resource's validation rules have changed, requiring a new schema to be applied. How can you ensure that the existing 'Order' resources conform to the new schema without disrupting the application's functionality?

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
I). Define the New Schema:
- Create a new CustomResourceDefinition (CRD) file with the updated schema for the 'Order' resource.
- Ensure that the CRD's 'spec-validation.openAPIV3Schema' field includes all the new validation rules.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: orders.example.com
spec:
  group: example.com
  version: v1
  names:
    kind: Order
    plural: orders
  scope: Namespaced
  validation:
    openAPIV3Schema:
      type: object
      properties:
        # Updated validation rules for the 'Order' resource
        ...
```

2. Update the CRD: - Apply the new CRD definition using 'kubectl apply -f order-crd.yaml'. 3. Create a Webhook for Validation: - Define a webhook in your Kubernetes cluster that will be responsible for validating the 'order' resources against the new schema. - Configure the webhook to be invoked during resource creation and update operations.

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: order-validation-webhook
webhooks:
- name: order-validation
  rules:
  - apiGroups: ["example.com"]
    apiVersions: ["v1"]
    resources: ["orders"]
    operations: ["CREATE", "UPDATE"]
  failurePolicy: Fail
  admissionReviewVersions: ["v1", "v1beta1"]
  clientConfig:
    service:
      name: order-validation-service
      namespace: default
      path: /validate
```

4. Deploy the Validation Service: - Create a deployment for the validation service that implements the logic for validating the 'Order' resources against the new schema. - The service should expose an endpoint that the webhook can communicate with.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-validation-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-validation
  template:
    metadata:
      labels:
        app: order-validation
    spec:
      containers:
      - name: order-validation
        image: your-validation-image:latest
        ports:
        - containerPort: 8443
```

5. Reconcile Existing Resources: - Once the validation webhook and service are deployed, create a temporary job that iterates through all existing 'Order resources. - The job snould validate each resource against tne new schema and automatically update any resources that do not comply.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: order-reconciliation-job
spec:
  template:
    spec:
      containers:
      - name: order-reconciliation
        image: your-reconciliation-image:latest
        command: ["/bin/sh", "-c", "kubectl get orders --all-namespaces -o jsonpath='{.items[].metadata.name}' | xargs -I {} kubectl get order {} -o jsonpath='{.spec}' | jq -r '. | . + { \"metadata\": { \"annotations\": { \"order.example.com/schema-version\": \"2\" } } }' | kubectl apply -f -"]
```

By following these steps, you can ensure that your 'order' resources conform to the new schema without disrupting the application's functionality The validation webhook prevents new invalid resources from being created, and the reconciliation job ensures that existing resources are updated to meet the new schema requirements. This approach allows for smooth schema evolution and maintains the consistency of your data.,

NEW QUESTION # 117

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.

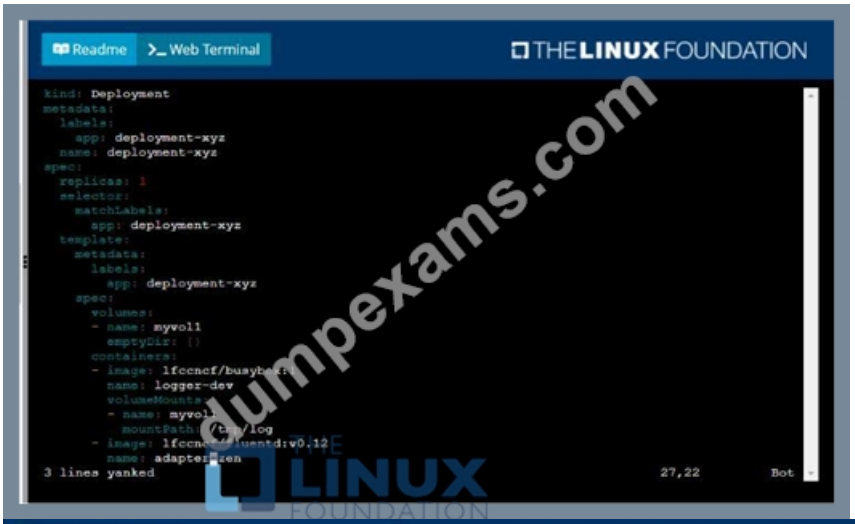
Task:
* Create a deployment named deployment-xyz in the default namespace, that:
*Includes a primary
lfccncf/busybox:1 container, named logger-dev
*includes a sidecar Ifccncf/fluentd:v0.12 container, named adapter-zen
*Mounts a shared volume /tmp/log on both containers, which does not persist when the pod is deleted
*Instructs the logger-dev
container to run the command

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

* 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 See the solution below.

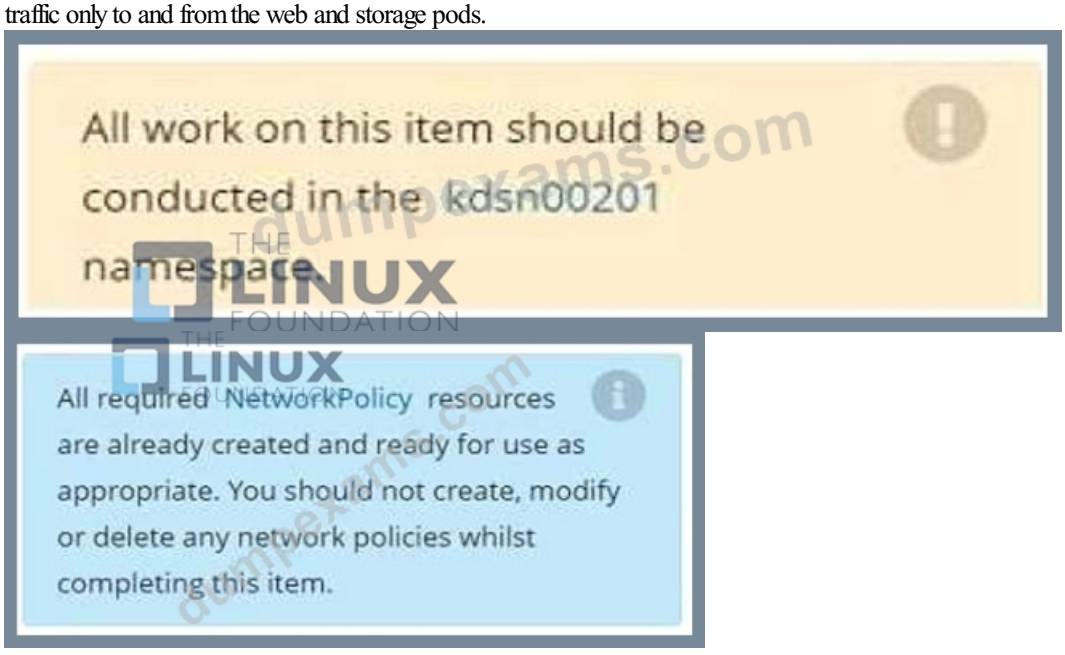
Answer:

Explanation:
Solution:



NEW QUESTION # 118
Exhibit:

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.



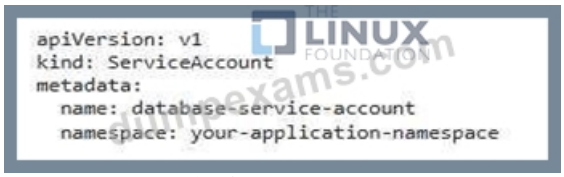
- A. Pending

Answer: A

NEW QUESTION # 119
You're building a containerized application that needs access to a database running outside of the Kubernetes cluster You need to implement a service account With specific permissions to access tne external database using an API key.

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 YAML file named 'database-service-account.yamr with the following contents:

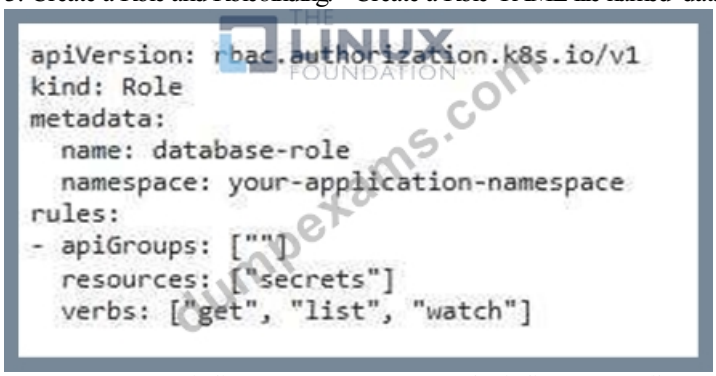


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: database-service-account
  namespace: your-application-namespace
```

2. Create a Secret for the API Key: - Create a secret YAML file named 'database-api-key.yaml with the following contents:

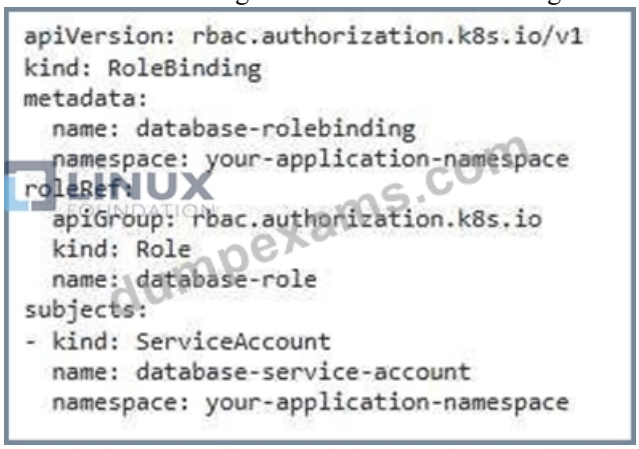
```
apiVersion: v1
kind: Secret
metadata:
  name: database-api-key
  namespace: your-application-namespace
type: Opaque
data:
  api_key:
```

3. Create a Role and Role8inding: - Create a Role YAML file named 'database-role.yaml with the following contents:



```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: database-role
  namespace: your-application-namespace
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "list", "watch"]
```

4. Create a ROIeBinding YAML: - Create a RoleBinding YAML file named 'database-rolebinding.yamr with the following contents:



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: database-rolebinding
  namespace: your-application-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: database-role
subjects:
- kind: ServiceAccount
  name: database-service-account
  namespace: your-application-namespace
```

5. Apply the YAML Files: - Apply the created YAML files using 'kubectl apply -f database-service-account.yamr , 'kubectl apply -f database-api-key.yamr, ' kubectl apply -f database-role.yamr , and 'kubectl apply -f database-rolebinding.yamr 6. Update your Deployment: - Update your application deployment to use the 'database-service-account and mount the secret containing the API key.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: your-application-deployment
  namespace: your-application-namespace
spec:
  template:
    spec:
      serviceAccountName: database-service-account
      containers:
      - name: your-application
        image: your-application-image
        env:
        - name: DATABASE_API_KEY
          valueFrom:
            secretKeyRef:
              name: database-api-key
              key: api_key
```

7. Access the External Database: - Your application container should now be able to access the external database using the API key provided in the secret.

NEW QUESTION # 120

.....