Exam CKAD Forum - Valid CKAD Exam Test

P.S. Free 2026 Linux Foundation CKAD dumps are available on Google Drive shared by iPassleader: https://drive.google.com/open?id=1MQZkRYU2fPMD-YAahr6iQ4WXKXBbIU_I

If you have a dream to get the Linux Foundation certification? Why don't you begin to act? The first step is to pass CKAD exam. Time will wait for no one. Only if you pass the CKAD exam, can you get a better promotion. And if you want to pass it more efficiently, we must be the best partner for you. Because we are professional CKAD Questions torrent provider, and our CKAD training materials are worth trusting; because we make great efforts on our CKAD learning guide, we do better and better in this field for more than ten years. Our CKAD study guide is your best choice.

The CKAD Certification is recognized as a valuable credential by employers and is becoming increasingly sought after in the industry. By earning the CKAD certification, developers can demonstrate their expertise in Kubernetes application development and increase their job prospects and earning potential. Linux Foundation Certified Kubernetes Application Developer Exam certification also provides a clear path for career advancement, as it demonstrates a commitment to continuous learning and professional development.

>> Exam CKAD Forum <<

Quiz 2026 Professional Linux Foundation Exam CKAD Forum

The CKAD certificate is hard to get. If you really crave for it, our CKAD guide practice is your best choice. We know it is hard for you to make decisions. You will feel sorry if you give up trying. Also, the good chance will slip away if you keep standing still. Our price is reasonable and inexpensive. You totally can afford for our CKAD Preparation engine. And we give some discounts from time to time, so you can buy at a more favorable price.

The CKAD certification exam is a 2-hour online exam that consists of 19-20 performance-based tasks that are designed to test an individual's skills in Kubernetes application development. CKAD exam is conducted on a live Kubernetes cluster, and the tasks are designed to simulate real-world scenarios. CKAD Exam can be taken from anywhere in the world, and the results are provided immediately after the completion of the exam.

Linux Foundation Certified Kubernetes Application Developer Exam Sample Questions (Q46-Q51):

NEW QUESTION # 46
You are building a microservice called 'order-service' that handles order processing. You need to configure a Securitycontext for the 'order-service' container tnat ensures it can access the network to communicate With other services and access specific hostPath volumes, but it should not have root privileges.

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Define the Securitycontext:

- Create a 'securityContext' section within the 'spec.template.spec.containers' block for your 'order-service' container.
- Set 'runAslJser' to a non-root IJID (e.g., 1001) to prevent running as the root user-
- Set 'allowPrivilegeEscalation' to 'false' to prevent the container from escalating its privileges.
- Set 'capabilities' to an empty array (so') to disable any additional capabilities.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      containers:
      - name: order-service
        image: your-image:latest
        securityContext:
          runAsUser: 1001
          allowPrivilegeEscalation: false
          capabilities:
            drop: []
        volumeMounts:
        - name: order-data
          mountPath: /data
          readOnly: false
        - name: config-volume
          mountPath: /config
          readOnly: true
      volumes:
      - name: order-data
        hostPath:
          path: /data
      - name: config-volume
        hostPath:
          path: /config
```


2. Mount HostPath Volumes: - Define 'volumeMountS for the required hostPath volumes. - Specify the mount path within the container C Idata' and 'Iconfig' in this example) and the volume name. - Define corresponding 'volumes with the 'hostPath' type, specifying the source path on the host and the volume name. 3. Create the Deployment: - Apply the Deployment YAML file using 'kubectl apply -f order-service-deployment-yaml' - The 'securitycontext' restricts the container's access to the host system's resources and prevents privilege escalation. - Setting 'runAsUserS to a non-root I-IID ensures that tne container runs as a non-root user - 'allowPrivilegeEscalation' prevents the container from elevating its privileges, even if it has the necessary capabilities. - The 'capabilities' section allows you to explicitly detine WhiCh capabilities the container snould nave. In this case, an empty array disables all additional capabilities, restricting the container's potential actions. - The 'volumeMounts' define how hostPath volumes are mounted within the container, providing access to specific directories on the host system. This configuration ensures that the 'order-service' container can access specific hostPath volumes and the network for communication with other services without running as root and without any additional capabilities, enhancing security.

NEW QUESTION # 47
You are working on a Kubernetes application that requires a scheduled job to run a data processing script every day at midnight. The script takes approximately 30 minutes to complete and requires access to a persistent volume to store its output dat a. How

would you create a Job resource that meets these requirements?

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
I). Create a Persistent Volume Claim:
- Define a Persistent Volume Claim (PVC) to request the necessary storage space.
- Specify the access mode and storage class according to your cluster configuration.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-processing-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: "standard"
```

2. Define the Job Resource: - Create a Job resource With a 'cronJob' schedule to trigger the execution at midnight every day. - Specify the 'backoffLimit' to control the number of retries in case of failures. - Define the 'spec-template.spec.containers' section for the container running the data processing script. - Mount the PVC to the container using 'volumeMounts'.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: data-processing-job
spec:
  template:
    spec:
      restartPolicy: Never
      containers:
      - name: data-processing
        image: your-image-name:latest
        command: ["/bin/sh", "-c", "your-data-processing-script.sh"]
        volumeMounts:
        - name: data-volume
          mountPath: /data
      volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: data-processing-pvc
  schedule: "0 0   " # Run at midnight daily
  backoffLimit: 3
```

3. Create the Job: - Apply the Job YAML file using 'kubectl apply -f data-processing-job.yamr 4. Verify Job Execution: - Use 'kubectl get jobS to monitor the status of the Job. - Check the 'status. completionTime' to verify that the Job completed successfully. - Verity that the output data is stored in the mounted persistent volume. 5. Update the Script - Update the 'your-data-processing-script.sh'' with the necessary commands to process the data and store the output in the ''ldata' directory. 6. Monitor the Job: - Continuously monitor the Job's status and logs using 'kubectl logs' to ensure it runs correctly. Note: Replace 'your-image-namelatest and 'your-data-processing-scriptsh' with the actual image name and script file respectively,

NEW QUESTION # 48
You must connect to the correct host . Failure to do so may result in a zero score.
[candidate@base] $ ssh ckad00034c
Task
A Deployment named content-marlin-deployment , running in namespace content-marlin is exposed via Ingress content-marlin-ingress .
The manifest files for the Deployment, Service and Ingress can be found at /home/candidate/content-marlin/.
The Deployment is supposed to be reachable at
http://content-marlin.local/content-marlin , but requesting this URL is currently returning an error.

Answer:

Explanation:
See the Explanation below for complete solution.
Explanation:
ssh ckad00034c
You're debugging an Ingress # Service # Deployment chain. The fastest way is:
* reproduce the error with the right Host header
* check Ingress rules (host/path/backend)
* check Service (selector/port/targetPort)
* check Endpoints (do we have ready pods?)
* fix the manifest(s) under /home/candidate/content-marlin/
* apply + re-test
Below are the exact commands + the most common fixes for this exact symptom.
1) Reproduce the failing request correctly
Even if DNS isn't set up, you can test with a Host header:
curl -i -H "Host: content-marlin.local"
http://127.0.0.1/content-marlin
If your ingress controller is not on localhost, find the NodePort/LoadBalancer IP. In these labs it's often localhost via a local proxy, but if needed:
kubectl get svc -A | egrep -i 'ingress|nginx'
kubectl get nodes -o wide
(But start with the localhost curl above.)
2) Inspect the provided manifests (this is what you must edit)
cd /home/candidate/content-marlin/
ls -l
sed -n '1,200p' *.yaml
Also view what's currently live in the cluster:
kubectl -n content-marlin get deploy,svc,ingress
kubectl -n content-marlin describe ingress content-marlin-ingress
kubectl -n content-marlin get ingress content-marlin-ingress -o yaml
What to look for in the Ingress:
* spec.rules.host should be content-marlin.local
* spec.rules.http.paths[].path should match /content-marlin
* Backend service name must be your service
* Backend service port must match the service port (name or number)
* pathType should be Prefix (usually safest)
3) Validate Service # Pod wiring (most common real cause)
3.1 Check service selector and ports
kubectl -n content-marlin get svc -o wide
kubectl -n content-marlin describe svc content-marlin-deployment 2>/dev/null || true kubectl -n content-marlin describe svc Identify the service that the Ingress points to (from describe ingress).
Check if the Service selector matches pod labels:
kubectl -n content-marlin get pods --show-labels
kubectl -n content-marlin get svc <SERVICE_NAME> -o jsonpath='{.spec.selector}{"\n"}'
3.2 Check endpoints (this tells you instantly if traffic can reach pods) kubectl -n content-marlin get endpoints kubectl -n content-marlin get endpoints <SERVICE_NAME> -o wide
* If ENDPOINTS is empty # Service selector doesn't match Pods OR Pods aren't Ready.
3.3 If endpoints empty, check pod readiness and labels
kubectl -n content-marlin get pods -o wide
kubectl -n content-marlin describe pod <pod-name>
4) Apply the most likely fix patterns
Fix pattern A: Ingress path needs rewrite
If your app serves / but you route /content-marlin, you often need rewrite.
Edit content-marlin-ingress manifest (in /home/candidate/content-marlin/) to include:
* path: /content-marlin
* pathType: Prefix
* annotation: rewrite to / (common for nginx ingress)
Example (typical nginx-ingress):
metadata:
annotations:
nginx.ingress.kubernetes.io/rewrite-target: /
spec:

rules:
- host: content-marlin.local
http:
paths:
- path: /content-marlin
pathType: Prefix
backend:
service:
name: <SERVICE_NAME>
port:
number: 80
If your ingress controller is not nginx, rewrite annotation may differ. But in CKAD labs, it's very often nginx.
Fix pattern B: Ingress points to wrong Service port
If the Ingress backend says port 80 but your Service exposes 8080 (or uses a named port), align them:
* Either change Ingress backend port.number
* Or change Service spec.ports[].port / targetPort
Fix pattern C: Service selector mismatch (endpoints empty)
If pods have label app=content-marlin but service selector is app=content-marlin-deployment (or vice versa), fix the Service selector to match pod labels.
Service should have:
spec:
selector:
app: <label-that-actually-exists-on-pods>
Fix pattern D: Service targetPort wrong
If container listens on 8080 but service targetPort is 80, fix it:
spec:
ports:
- port: 80
targetPort: 8080
5) Apply the corrected manifests
After editing the YAMLs under /home/candidate/content-marlin/:
kubectl apply -f /home/candidate/content-marlin/
Wait for readiness:
kubectl -n content-marlin rollout status deploy content-marlin-deployment kubectl -n content-marlin get endpoints kubectl -n content-marlin describe ingress content-marlin-ingress
6) Re-test the URL
curl -i -H "Host: content-marlin.local"
http://127.0.0.1/content-marlin
If you still get errors, also check ingress controller logs/events quickly:
kubectl -n content-marlin get events --sort-by=.lastTimestamp | tail -n 30 kubectl get pods -A | egrep -i 'ingress|nginx' The fastest way for you to finish in 1 shot Run these and paste the output (I'll tell you exactly which line to change and what to change it to):
kubectl -n content-marlin describe ingress content-marlin-ingress
kubectl -n content-marlin get svc -o wide
kubectl -n content-marlin get endpoints -o wide
kubectl -n content-marlin get pods --show-labels
sed -n '1,200p' /home/candidate/content-marlin/*.yaml
But even without pasting, if you follow steps 2-4 above, you'll find the broken link (Ingress rule, Service port, selector, or rewrite) and fix it cleanly.

NEW QUESTION # 49
You are developing a microservices application and want to deploy it to Kubernetes using Helm. You have two services: 'user-service and 'order-service. The 'order-service depends on the "user-service'. How would you use Helm to manage these deployments, ensuring that the 'order- service' only starts after the 'user-service' is successfully deployed and running?

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :

1. Create a Helm Chart for Each Service:
- 'user-service' chart:
- Create a 'values.yamr file for the 'user-service' chart.
- Define the container image, resources, and any other necessary configurations for the 'user-service'.
- 'order-service' chart:
- Create a 'values-yamr file for the 'order-service' chart
- Define the container image, resources, and any other necessary configurations for the 'order-service'
- In tne 'values.yamr, add a dependency on the 'user-service' chart.

```
# order-service/values.yaml
dependencies:
  - name: user-service
    repository: ./user-service
    version: 1.0.0
```

2. Configure Helm for Dependency Management: - Use the '-dependency-update' flag to ensure that Helm automatically updates the 'user-service chart before deploying the 'order-service' bash helm dependency update order-service 3. Deploy the Services Using Helm: - Deploy the 'user-service chart: bash helm install user-service Juser-service - Deploy the 'order-service' chart: bash helm install order-service ./order-service - Helm will automatically handle the dependency between the services, ensuring that the 'user-services is deployed before the 'order-service' 4. Verify Deployment and Dependency: - Use ' kubectl get pods -l app=user-service' and 'kubectl get pods -l app=order-service' to verify that the pods are running. - You Should observe that the 'user-service' pods are up and running before the 'order-services pods start. - You can also use 'kubectl describe pod' to see the pod events and confirm that the 'order-service' pod is waiting for the 'user-service' to be ready before starting.,

NEW QUESTION # 50
You are running a Deployment for a database service with 3 replicas. You want to ensure that only one pod is updated at a time, but you need to guarantee that the database service remains available throughout tne update process. How would you configure the Deployment to achieve this?

Answer:

Explanation:
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Update the Deployment YAMLI
- Update the 'replicas' to 3.
- Define 'maxiJnavailable: 1 ' and 'maxSurge: O' in the 'strategy-rollingupdate' section to control the rolling update process.
- Use a 'readiness probe' within your container definition to ensure that the pod is considered ready only when tne database is successfully started and connected.
- Configure a 'strategy-type' to 'RollingUpdate' to trigger a rolling update when the deployment is updated.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      containers:
        - name: database
          image: database-image:latest
          imagePullPolicy: Always
          readinessProbe:
            tcpSocket:
              port: 5432
            initialDelaySeconds: 15
            periodSeconds: 5
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
```

2. Create the Deployment: - Apply the updated YAML file using 'kubectl apply -f database-deployment-yamp 3. Verify the Deployment - Check the status of the deployment using 'kubectl get deployments database-deployment to confirm the rollout and updated replica count. 4. Trigger the Automatic Update: - Pusn a new image to the Docker Hub repository. 5. Monitor the

Deployment - Use 'kubectl get pods -l to monitor the pod updates during the rolling update process. You will observe that only one pod is terminated at a time. The readiness probe will ensure that a new pod is only considered ready when it's successfully connected to the database. 6. Check for Successful Update: - Once the deployment is complete, use 'kubectl describe deployment database-deployment to see that the 'updatedReplicas' field matches the 'replicas' field, indicating a successful update.,

NEW QUESTION # 51

.....

P.S. Free 2026 Linux Foundation CKAD dumps are available on Google Drive shared by iPassleader: https://drive.google.com/open?id=1MQZkRYU2fPMD-YAahr6iQ4WXKXBbIU_I