100% Pass Quiz Efficient CKS - Test Certified Kubernetes Security Specialist (CKS) Collection

2025 Latest ExamcollectionPass CKS PDF Dumps and CKS Exam Engine Free Share: https://drive.google.com/open?id=1rLkK_bs2mRHQmQzAehNUb5KxGpfKBugW

This is a Linux Foundation CKS practice exam software for Windows computers. This CKS practice test will be similar to the actual CKS exam. If user wish to test the Certified Kubernetes Security Specialist (CKS) (CKS) study material before joining ExamcollectionPass, they may do so with a free sample trial. This CKS Exam simulation software can be readily installed on Windows-based computers and laptops. Since it is desktop-based CKS practice exam software, it is not necessary to connect to the internet to use it.

The CKS certification exam is ideal for IT professionals, system administrators, security analysts, and DevOps engineers who are interested in developing expertise in Kubernetes security. Certified Kubernetes Security Specialist (CKS) certification exam is designed to validate the candidate's skills in identifying and mitigating security risks, securing containerized applications, and implementing security best practices in Kubernetes environments. CKS exam tests the candidate's knowledge in various areas, including Kubernetes API authentication and authorization, network policies, secrets management, and container runtime security.

The CKS certification exam consists of multiple-choice questions and performance-based tasks that require candidates to demonstrate their ability to secure Kubernetes clusters. CKS Exam covers a wide range of topics, including Kubernetes security concepts, securing Kubernetes components, securing container images, securing network communication, and monitoring Kubernetes security. Candidates who pass the CKS certification exam will receive a digital badge and a certificate that can be used to showcase their Kubernetes security expertise to potential employers. The CKS certification is a valuable credential for Kubernetes security professionals looking to advance their careers and enhance their credibility in the industry.

>> Test CKS Collection <<

100% Pass CKS - Certified Kubernetes Security Specialist (CKS) Newest Test Collection

ExamcollectionPass has a strong IT elite team. They use their professional eyes searching the latest CKS braindumps and CKS certification training materials. With them, you can save more time to study and pass the CKS Exam. After you purchase our CKS exam dumps, we will offer free update service in one year.

The CKS certification exam is designed to test an individual's expertise in various areas related to Kubernetes security, including cluster setup, securing Kubernetes components, securing container images, securing network and storage configurations, and securing Kubernetes API and authentication. CKS Exam is conducted online, and the duration of the exam is two hours. CKS exam consists of 17-20 performance-based tasks that test an individual's ability to apply their knowledge to real-world scenarios.

Linux Foundation Certified Kubernetes Security Specialist (CKS) Sample Questions (Q19-Q24):

NEW QUESTION # 19
You have a Kubernetes cluster that runs a sensitive application called "banking-app" in a Deployment The application needs access

to a private registry to pull container images. You want to ensure that the "banking-app" container only communicates with the private registry and no other external networks. How can you use NetworkPolicy to enforce this network security restriction?

Answer:

Explanation:
Solution (Step by Step) :
1. Create a NetworkPolicy tor the Private Registry: You'll create a NetworkPolicy that allows the "banking-app" container to communicate with the private registry but blocks access to all other external networks.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: banking-app-registry-access
  namespace: default  # Your application's namespace
spec:
  podSelector:
    matchLabels:
      app: banking-app
  policyTypes:
    - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: banking-app
    - ipBlock:
        cidr: "172.17.0.0/16"  # Replace with your private registry's CIDR
        except: []
```


'podSelectors: This defines which pods are affected by the policy. - 'policyTypeS: This specifies the type of traffic that the policy governs (Ingress in this case). - 'ingress': Defines the allowed incoming traffic. - 'trom': Specifies the source of allowed traffic. - spodSeIector': Allows traffic from other pods with the "banking-app" label. - 'ipBlock': Allows traffic from a specific CIDR range. - 'cidr': Replace '172.17.0.0/16' with the actual CIDR of your private registry. - 'except': Optional for excluding specific IP addresses or ranges. 2 Apply the NetworkPolicy: Apply the YAML file to your cluster: bash kubectl apply -f banking-app-registry-access.yaml 3. Verify NetworkPolicy: After applying the policy, run: bash kubectl get networkpolicy -n default # Replace 'default' with your namespace You should see your new "banking-app-registry-access" NetworkPolicy listed. 4. Test the Policy: - Try to access external networks from within the "banking-app" container. - You should observe that the container is unable to connect to any external services except the private registry. - Make sure your application can still pull images from the private registry. 5. Additional Considerations: - Egress Traffic: You might need to define a separate NetworkPolicy for 'Egress' traffic if you want to allow the "banking-app" to communicate with specific internal services. - Detailed Controls: You can add more specific rules to the 'ingress' section to allow specific ports or protocols from the private registry.

NEW QUESTION # 20
You are working on a Kubernetes cluster that hosts an application that interacts With sensitive data. You need to perform a static analysis of the application's container image to identify potential security vulnerabilities before deploying it to the cluster.

Answer:

Explanation:
Solution (Step by Step) :
1. cnoose a Static Analysis Tool:
- Select a suitable static analysis tool for container images. Some popular options include:
- Trivy: https://aquasecurity.github.io/trivy/
- Snyk: https://snyk.io/
- Anchore Engine: [https://anchore.com/l(https://anchore.com/)
2 Install and Configure the Tool:
- Install the chosen tool on your machine or integrate it into your CI/CD pipeline.
- Configure the tool to scan the container image for vulnerabilities.
3. Scan the Container Image:
- Use the tool's command-line interface or API to scan the container image.
- Provide the image name or tag as input to the tool.

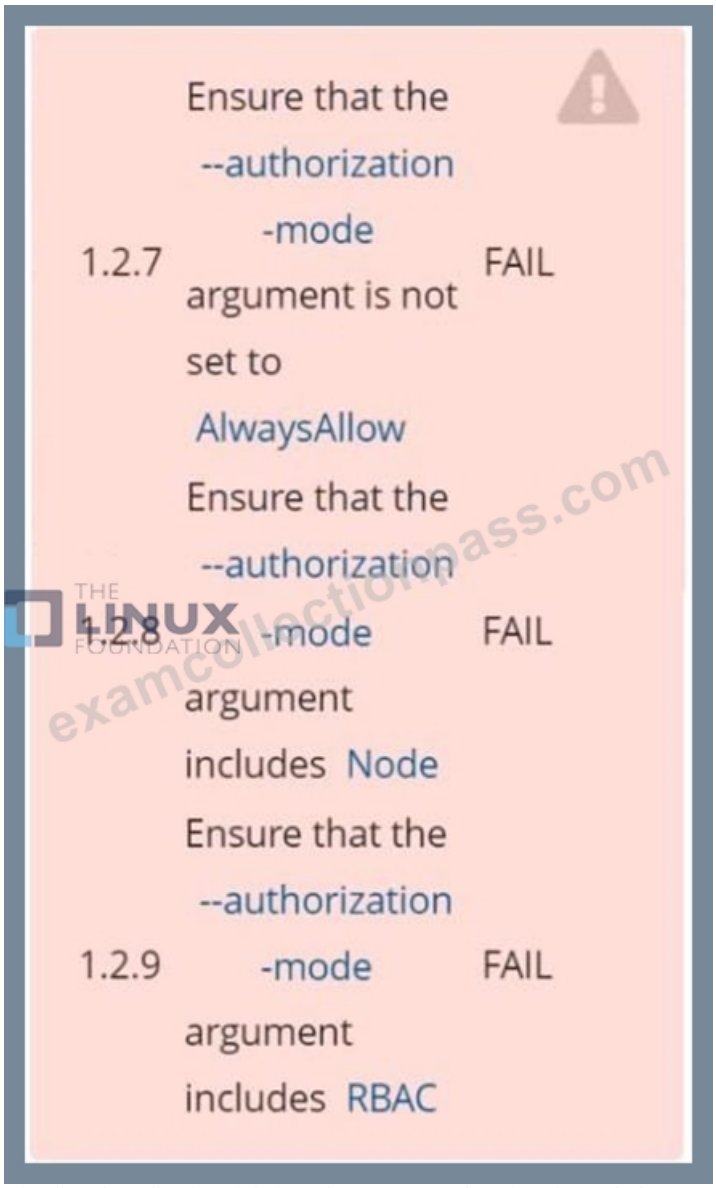
4. Analyze the Results:
- The tool will generate a report detailing the identified vulnerabilities.
- Review the report and prioritize remediation actions based on the severity and impact of the vulnerabilities.
- Use the tool's features to track the status of vulnerabilities and their remediation.

NEW QUESTION # 21
SIMULATION
Using the runtime detection tool Falco, Analyse the container behavior for at least 20 seconds, using filters that detect newly spawning and executing processes in a single container of Nginx.
store the incident file art /opt/falco-incident.txt, containing the detected incidents. one per line, in the format [timestamp],[uid],[processName]

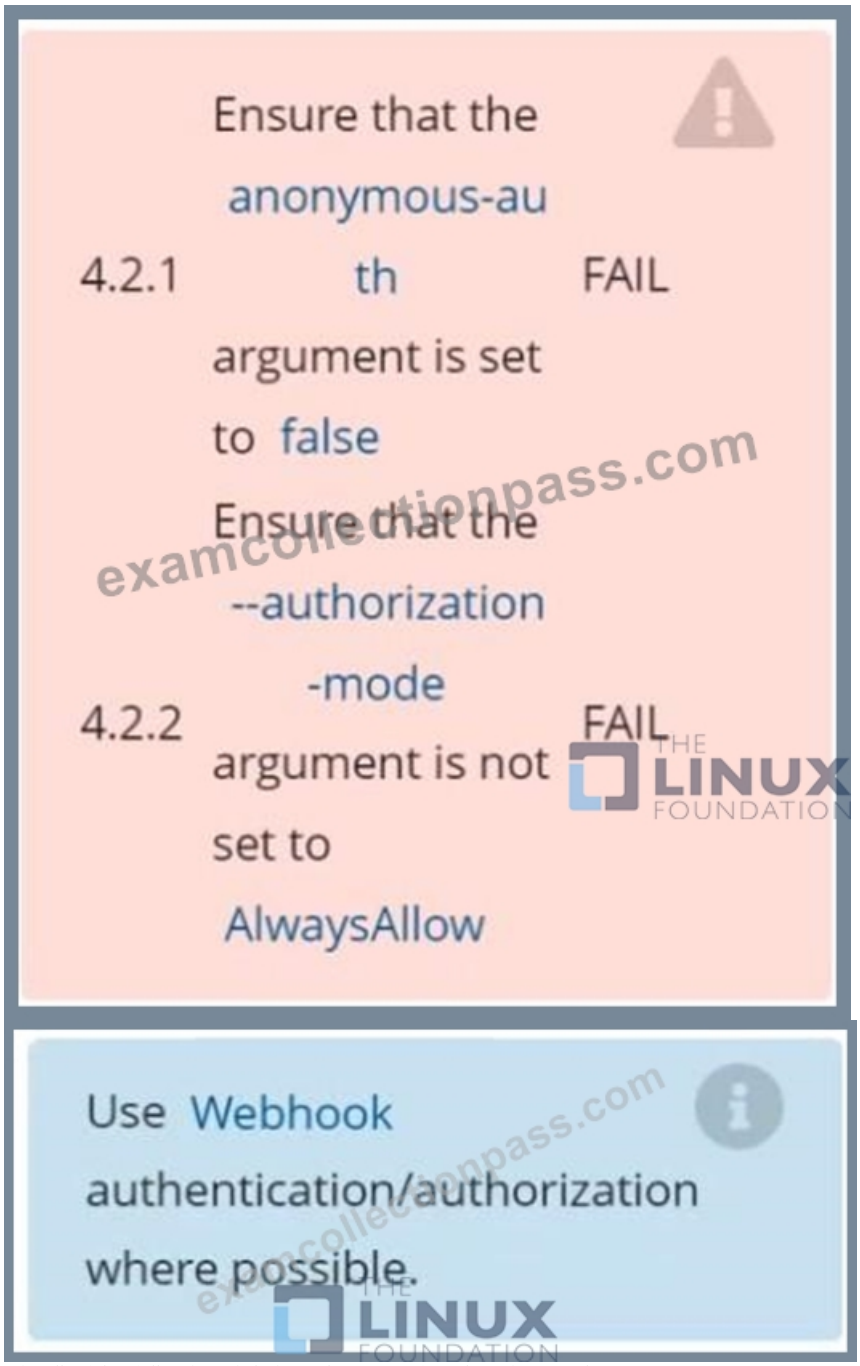
- A. Send us the Feedback on it.

Answer: A

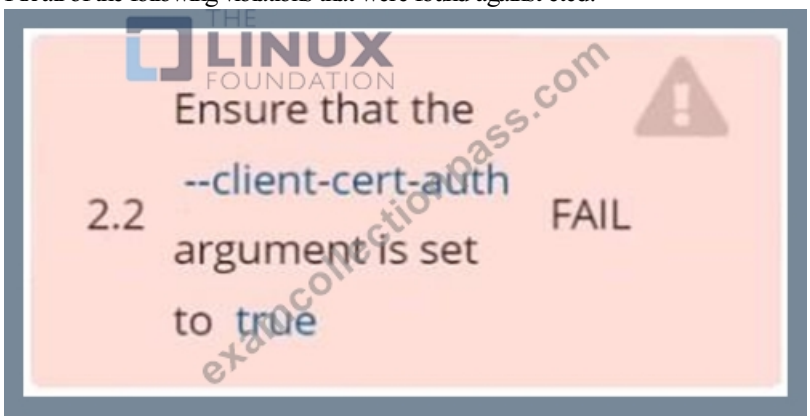
NEW QUESTION # 22
Context
A CIS Benchmark tool was run against the kubeadm-created cluster and found multiple issues that must be addressed immediately.
Task
Fix all issues via configuration and restart the affected components to ensure the new settings take effect.
Fix all of the following violations that were found against the API server:



Fix all of the following violations that were found against the Kubelet:



Fix all of the following violations that were found against etcd:



Answer:

Explanation:

```
candidate@cli:~$ kubectl delete sa/podrunner -n qa
serviceaccount "podrunner" deleted
candidate@cli:~$ kubectl config use-context KSCS00201
Switched to context "KSCS00201".
candidate@cli:~$ ssh kscs00201-master
Warning: Permanently added '10.240.86.194' (ECDSA) to the list of known hosts.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@kscs00201-master:~# vim /etc/kubernetes/manifests/kube-apiserver.yaml
root@kscs00201-master:~# systemctl daemon-reload
root@kscs00201-master:~# systemctl restart kubelet.service
root@kscs00201-master:~# systemctl enable kubelet.service
root@kscs00201-master:~# systemctl status kubelet.service
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Fri 2022-05-20 14:19:31 UTC; 29s ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 134205 (kubelet)
      Tasks: 16 (limit: 76200)
     Memory: 39.5M
     CGroup: /system.slice/kubelet.service
             └─134205 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kub>

May 20 14:19:35 kscs00201-master kubelet[134205]: I0520 14:19:35.420825  134205 reconciler.>
May 20 14:19:35 kscs00201-master kubelet[134205]: I0520 14:19:35.420863  134205 reconciler.>
May 20 14:19:35 kscs00201-master kubelet[134205]: I0520 14:19:35.420907  134205 reconciler.>
May 20 14:19:35 kscs00201-master kubelet[134205]: I0520 14:19:35.420928  134205 reconciler.>
May 20 14:19:36 kscs00201-master kubelet[134205]: I0520 14:19:36.572353  134205 request.go:>
May 20 14:19:37 kscs00201-master kubelet[134205]: I0520 14:19:37.112347  134205 prober_mana>
May 20 14:19:37 kscs00201-master kubelet[134205]: E0520 14:19:37.185076  134205 kubelet.go:>
May 20 14:19:37 kscs00201-master kubelet[134205]: I0520 14:19:37.645798  134205 kubelet.go:>
May 20 14:19:38 kscs00201-master kubelet[134205]: I0520 14:19:38.184062  134205 kubelet.go:>
May 20 14:19:40 kscs00201-master kubelet[134205]: I0520 14:19:40.036042  134205 prober_mana>
lines 1-22/22 (END)
```



```
de Agent
et.service; enabled; vendor preset: enabled)
ce.d

5-20 14:19:31 UTC; 29s ago



trap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet>

5]: I0520 14:19:35.420825  134205 reconciler.go:221] "operationExecutor.VerifyControllerAtt>
5]: I0520 14:19:35.420863  134205 reconciler.go:221] "operationExecutor.VerifyControllerAtt>
5]: I0520 14:19:35.420907  134205 reconciler.go:221] "operationExecutor.VerifyControllerAtt>
5]: I0520 14:19:35.420928  134205 reconciler.go:157] "Reconciler: start to sync state"
5]: I0520 14:19:36.572353  134205 request.go:665] Waited for 1.049946364s due to client-sid>
5]: I0520 14:19:37.112347  134205 prober_manager.go:255] "Failed to trigger a manual run" p>
5]: E0520 14:19:37.185076  134205 kubelet.go:1711] "Failed creating a mirror pod for" err="
5]: I0520 14:19:37.645798  134205 kubelet.go:1693] "Trying to delete pod" pod="kube-system/>
5]: I0520 14:19:38.184062  134205 kubelet.go:1698] "Deleted mirror pod because it is outdat>
5]: I0520 14:19:40.036042  134205 prober_manager.go:255] "Failed to trigger a manual run" p>
~
~
lines 1-22/22 (END)
```

```
let.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --
o:221] "operationExecutor.VerifyControllerAttachedVolume started for volume \"kube-proxy\"
o:221] "operationExecutor.VerifyControllerAttachedVolume started for volume \"lib-modules\"
o:221] "operationExecutor.VerifyControllerAttachedVolume started for volume \"flannel-cfg\"
o:157] "Reconciler: start to sync state"
65] Waited for 1.049946364s due to client-side throttling, not priority and fairness, reque
er.go:255] "Failed to trigger a manual run" probe="Readiness"
711] "Failed creating a mirror pod for" err="pods \"kube-apiserver-kscs00201-master\" alrea
693] "Trying to delete pod" pod="kube-system/kube-apiserver-kscs00201-master" podUID=bb91e1
698] "Deleted mirror pod because it is outdated" pod="kube-system/kube-apiserver-kscs00201-
er.go:255] "Failed to trigger a manual run" probe="Readiness"
~
~
root@kscs00201-master:~# vim /var/lib/kubelet/config.yaml
```

```
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 0s
    cacheUnauthorizedTTL: 0s
cgroupDriver: systemd
```

```
~
~
root@kscs00201-master:~# vim /var/lib/kubelet/config.yaml
root@kscs00201-master:~# vim /var/lib/kubelet/config.yaml
root@kscs00201-master:~# vim /etc/kubernetes/manifests/etcd.yaml
root@kscs00201-master:~# systemctl daemon-reload
root@kscs00201-master:~# systemctl restart kubelet.service
root@kscs00201-master:~# systemctl status kubelet.service
```

```
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Fri 2022-05-20 14:22:29 UTC; 4s ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 135849 (kubelet)
      Tasks: 17 (limit: 76200)
     Memory: 38.0M
     CGroup: /system.slice/kubelet.service
             └─135849 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kub

May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330232  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330259  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330304  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330354  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330378  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330397  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330415  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330433  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330452  135849 reconciler.
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520 14:22:30.330463  135849 reconciler.
lines 1-22/22 (END)
```

```
May 20 14:22:30 kscs00201-master kubelet[135849]: I0520_14:22:30.330463  135849 reconciler.
root@kscs00201-master:~#
root@kscs00201-master:~#
root@kscs00201-master:~#
root@kscs00201-master:~# exit
logout
Connection to 10.240.86.194 closed.
candidate@cli:~$
```

NEW QUESTION # 23
You are building a microservice architecture on Kubernetes- You are using Docker images from a public registry for your applications. One of the microservices is responsible for managing sensitive user data. To minimize the base image footprint and enhance security, you need to create a custom base image that iS as minimal as possible while still containing the required dependencies for your service.
What are the steps you would take to create a custom base image for this microservice? How would you ensure the custom base image is secure, and how would you incorporate it into your deployment process? Provide a step-by-step guide with code examples.

Answer:

Explanation:
Solution (Step by Step) :
1. Choose a Minimal Base Image:
- Select a base image like Alpine Linux, which is known for its small size and security features.
- Use a multi-stage build to minimize the size of the final image.
- Example:
docket-file
FROM alpine:3.16 as builder
Install required dependencies
RUN apk update && apk add --no-cache python3 python3-dev build-base

```
# Build your application
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt

# Final image with minimal dependencies
FROM alpine:3.16
COPY --from=builder /app /app
COPY --from=builder /usr/local/lib/python3.10/site-packages /usr/local/lib/python3.10/site-packages
WORKDIR /app
CMD ["python", "app.py"]
```

2. Security Best Practices: - Use a non-root user inside the container. - Enable security options in your Dockerfile like '-no-cache' to minimize potential vulnerabilities. - Harden the base image: - Remove unnecessary packages and services. - Disable unnecessary pons and protocols. - Set appropriate permissions for files and directories. - Example: dockeflle FROM alpine:3.16 as builder USER nonrootuser RUN apk update && apk add 0--no-cache python3 python3-dev build-base # ... rest of the Dockerfile 3. Deployment Process: - Build the custom base image. - Push the base image to a private registry. - Update the deployment YAML file to use the new base image. - Example:

4. Testing and Monitoring: - Regularly scan the base image for vulnerabilities. - Monitor the container for suspicious activity - Employ security tools like Falco and Clair-

NEW QUESTION # 24

.....