New Linux Foundation CKS Test Prep & CKS Updated Dumps

P.S. Free & New CKS dumps are available on Google Drive shared by Test4Cram: https://drive.google.com/open?id=15wAQ9Vy6gXQtnvrxpQamgC2Av-SDU6Dn

For consolidation of your learning, our PDF， Software and APP online versions of the CKS exam questions also provide you with different sets of practice questions and answers. Doing all these sets of the CKS study materials again and again, you enrich your knowledge and maximize chances of an outstanding exam success. And the content of the three version is the same, but the displays are totally differnt. If you want to know them before the payment, you can free download the demos of our CKS leaning braindumps.

The CKS certification exam is a rigorous test of an IT professional's knowledge and skills in Kubernetes security. CKS exam consists of 17 tasks that must be completed within two hours. The tasks are designed to test the candidate's ability to identify and mitigate security risks in Kubernetes clusters and workloads. CKS Exam is a hands-on test, which means that the candidate must demonstrate their ability to perform tasks in a live Kubernetes environment.

>> New Linux Foundation CKS Test Prep <<

CKS Updated Dumps - Valid CKS Test Practice

The aim of Linux Foundation CKS test torrent is to help you optimize your IT technology and get the CKS certification by offerring the high quality and best accuracy CKS study material. If you want to pass your CKS Actual Exam with high score, Test4Cram CKS latest exam cram is the best choice for you. The high hit rate of CKS test practice will help you pass and give you surprise.

The CKS Exam is designed for professionals who have experience in Kubernetes administration and are familiar with container security concepts. CKS exam covers a wide range of topics related to Kubernetes security, including securing cluster components, securing container images, securing network communication, and securing Kubernetes API.

Linux Foundation Certified Kubernetes Security Specialist (CKS) Sample Questions (Q29-Q34):

NEW QUESTION # 29
SIMULATION
You can switch the cluster/configuration context using the following command:
[desk@cli] $ kubectl config use-context test-account
Task: Enable audit logs in the cluster.
To do so, enable the log backend, and ensure that:
1. logs are stored at /var/log/Kubernetes/logs.txt
2. log files are retained for 5 days
3. at maximum, a number of 10 old audit log files are retained
A basic policy is provided at /etc/Kubernetes/logpolicy/audit-policy.yaml. It only specifies what not to log.
Note: The base policy is located on the cluster's master node.

Edit and extend the basic policy to log:
1. Nodes changes at RequestResponse level
2. The request body of persistentvolumes changes in the namespace frontend
3. ConfigMap and Secret changes in all namespaces at the Metadata level Also, add a catch-all rule to log all other requests at the Metadata level Note: Don't forget to apply the modified policy.

Answer:

Explanation:
See the Explanation belowExplanation:
$ vim /etc/kubernetes/log-policy/audit-policy.yaml
- level: RequestResponse
userGroups: ["system:nodes"]
- level: Request
resources:
- group: "" # core API group
resources: ["persistentvolumes"]
namespaces: ["frontend"]
- level: Metadata
resources:
- group: ""
resources: ["configmaps", "secrets"]
- level: Metadata
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
Add these
- --audit-policy-file=/etc/kubernetes/log-policy/audit-policy.yaml
- --audit-log-path=/var/log/kubernetes/logs.txt
- --audit-log-maxage=5
- --audit-log-maxbackup=10
Explanation:
[desk@cli] $ ssh master1
[master1@cli] $ vim /etc/kubernetes/log-policy/audit-policy.yaml
apiVersion: audit.k8s.io/v1 # This is required.
kind: Policy
Don't generate audit events for all requests in RequestReceived stage.
omitStages:
- "RequestReceived"
rules:
Don't log watch requests by the "system:kube-proxy" on endpoints or services
- level: None
users: ["system:kube-proxy"]
verbs: ["watch"]
resources:
- group: "" # core API group
resources: ["endpoints", "services"]
Don't log authenticated requests to certain non-resource URL paths.
- level: None
userGroups: ["system:authenticated"]
nonResourceURLs:
- "/api*" # Wildcard matching.
- "/version"
Add your changes below
- level: RequestResponse
userGroups: ["system:nodes"] # Block for nodes
- level: Request
resources:
- group: "" # core API group
resources: ["persistentvolumes"] # Block for persistentvolumes
namespaces: ["frontend"] # Block for persistentvolumes of frontend ns
- level: Metadata
resources:

- group: "" # core API group
resources: ["configmaps", "secrets"] # Block for configmaps & secrets
- level: Metadata # Block for everything else
[master1@cli] $ vim /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
annotations:
kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 10.0.0.5:6443 labels:
component: kube-apiserver
tier: control-plane
name: kube-apiserver
namespace: kube-system
spec:
containers:
- command:
- kube-apiserver
- --advertise-address=10.0.0.5
- --allow-privileged=true
- --authorization-mode=Node,RBAC
- --audit-policy-file=/etc/kubernetes/log-policy/audit-policy.yaml #Add this
- --audit-log-path=/var/log/kubernetes/logs.txt #Add this
- --audit-log-maxage=5 #Add this
- --audit-log-maxbackup=10 #Add this
...
output truncated
Note: log volume & policy volume is already mounted in vim /etc/kubernetes/manifests/kube-apiserver.yaml so no need to mount it.

NEW QUESTION # 30
You suspect that the Kubernetes binaries on your cluster nodes may have been tampered with. Implement a process to verify the integrity of the binaries and identify any potential compromises.

Answer:

Explanation:
Solution (Step by Step):
1. Establish a known-good baseline: Obtain known-good copies of the Kubernetes binaries from a trusted source, such as the official Kubernetes release page or your distribution's package repository.
2. Calculate checksums: Calculate the SHA-256 checksums of the known-good binaries and the binaries on your nodes.
bash
sha256sum /usr/bin/kubeadm lusr/bin/kubelet 'usr/bin/kubectl
3. Compare checksums: Compare the checksums of the binaries on your nodes with the checksums of the known-good binaries. Any discrepancies indicate potential tampering.
4. Inspect binaries for modifications: If checksum mismatches are found, use tools like 'diff or 'cmp' to compare the suspect binaries with the known- good binaries to identify specific modifications.
5. Analyze system logs: Review system logs, such as audit logs and syslog, for any suspicious activity related to the Kubernetes binaries or processes.
6. Reinstall binaries from a trusted source: If tampering is confirmed, reinstall the Kubernetes binaries from a trusted source.
7. Investigate the root cause: Conduct a thorough investigation to determine the root cause of the tampering and take steps to prevent future compromises. This may involve reviewing access controls, network security, and security monitoring practices.

NEW QUESTION # 31
SIMULATION
Create a new ServiceAccount named backend-sa in the existing namespace default, which has the capability to list the pods inside the namespace default.
Create a new Pod named backend-pod in the namespace default, mount the newly created sa backend-sa to the pod, and Verify that the pod is able to list pods.
Ensure that the Pod is running.

Answer:

Explanation:
A service account provides an identity for processes that run in a Pod.
When you (a human) access the cluster (for example, using kubectl), you are authenticated by the apiserver as a particular User Account (currently this is usually admin, unless your cluster administrator has customized your cluster). Processes in containers inside pods can also contact the apiserver. When they do, they are authenticated as a particular Service Account (for example, default).
When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace. If you get the raw json or yaml for a pod you have created (for example, kubectl get pods/<podname> -o yaml), you can see the spec.serviceAccountName field has been automatically set.
You can access the API from inside a pod using automatically mounted service account credentials, as described in Accessing the Cluster. The API permissions of the service account depend on the authorization plugin and policy in use.
In version 1.6+, you can opt out of automounting API credentials for a service account by setting automountServiceAccountToken: false on the service account:
apiVersion: v1
kind: ServiceAccount
metadata:
name: build-robot
automountServiceAccountToken: false
...
In version 1.6+, you can also opt out of automounting API credentials for a particular pod:
apiVersion: v1
kind: Pod
metadata:
name: my-pod
spec:
serviceAccountName: build-robot
automountServiceAccountToken: false
...
The pod spec takes precedence over the service account if both specify a automountServiceAccountToken value.

NEW QUESTION # 32
You have a Kubernetes cluster that hosts a web application using a Deployment. The Deployment's service exposes the application on port 80. You want to restrict access to the web application to only authorized IP addresses, while allowing access to the Kubernetes API server from any IP address.

Answer:

Explanation:
Solution (Step by Step) :
1. Create a Network Policy:
- Create a Network Policy that allows access to the web application only from the authorized IP addresses.
- Here's an example network policy:
- Replace '10.0.0.0/24' With the authorized IP addresses you want to allow. - This policy allows outbound traffic to any IP address.
- Create the policy using 'kubectl apply -f web-app-allow-list.yaml 2. Create a Network Policy for the Kubernetes API Server: - Create a Network Policy that allows access to the Kubernetes API server from any IP address. - Here's an example network policy:
- Create the policy using 'kubectl apply -f api-server-allow-all.yaml 3. Verify the Network Policies: - Use 'kubectl get networkpolicy -n default' to verify that the 'web-app-allow-list' Network Policy is created and 'kubectl get networkpolicy -n kube- system' to verify that the 'api-server-allow-all' Network Policy is created. 4. Test the Access: - Attempt to access the web application from a machine within the authorized IP address range. You should be able to access the application. - Attempt to access the web application from a machine outside the authorized IP address range. You should be unable to access tne application. 5. Verify API Server Access: - Try to connect to the Kubernetes API server from any machine using 'kubectl'. You should be able to connect successfully. Note: This approach assumes that the web application is running in the 'default' namespace. If it's running in a different namespace, adjust the 'namespaces field in the 'web-app-allow-list' Network Policy accordingly.

NEW QUESTION # 33
You must complete this task on the following cluster/nodes: Cluster: immutable-cluster Master node: master1 Worker node:

worker1 You can switch the cluster/configuration context using the following command: [desk@cli] $ kubectl config use-context immutable-cluster Context: It is best practice to design containers to be stateless and immutable. Task: Inspect Pods running in namespace prod and delete any Pod that is either not stateless or not immutable. Use the following strict interpretation of stateless and immutable: 1. Pods being able to store data inside containers must be treated as not stateless. Note: You don't have to worry whether data is actually stored inside containers or not already. 2. Pods being configured to be privileged in any way must be treated as potentially not stateless or not immutable.

Answer:

Explanation:

Reference: https://kubernetes.io/docs/concepts/policy/pod-security-policy/ https://cloud.google.com/architecture/best-practices-for-operating-containers

NEW QUESTION # 34

......