CKS Pass4sure Training - CKS Latest Vce & CKS Free Demo

DOWNLOAD the newest Prep4cram CKS PDF dumps from Cloud Storage for free: https://drive.google.com/open?id=1x1Yo-Hdl-lnmAzwBljMnk3f-MNud1b51

There are Certified Kubernetes Security Specialist (CKS) (CKS) exam questions provided in Certified Kubernetes Security Specialist (CKS) (CKS) PDF questions format which can be viewed on smartphones, laptops, and tablets. So, you can easily study and prepare for your Certified Kubernetes Security Specialist (CKS) (CKS) exam anywhere and anytime. You can also take a printout of these Linux Foundation PDF Questions for off-screen study.

The CKS Exam is a vendor-neutral certification, meaning that it is not tied to any specific vendor or technology. This makes it an

ideal certification for security professionals who work with Kubernetes platforms and containerized workloads regardless of the vendor or technology used. Certified Kubernetes Security Specialist (CKS) certification is also recognized globally, making it a valuable asset for security professionals seeking to advance their careers in the Kubernetes and containerization space.

>> Test CKS Discount Voucher <<

Test CKS Discount Voucher Exam Instant Download | Updated Linux Foundation CKS: Certified Kubernetes Security Specialist (CKS)

Prep4cram provides a clear and superior solutions for each Linux Foundation CKS Exam candidates. We provide you with the Linux Foundation CKS exam questions and answers. Our team of IT experts is the most experienced and qualified. Our test questions and the answer is almost like the real exam. This is really amazing. More importantly, the examination pass rate of Prep4cram is highest in the worldwide.

Linux Foundation Certified Kubernetes Security Specialist (CKS) Sample Questions (Q34-Q39):

NEW QUESTION # 34
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:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-app-allow-list
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web-app
  ingress:
  - from:
    - ipBlock:
        cidr: 10.0.0.0/24 # Replace with your authorized IP addresses
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
```


- 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:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-server-allow-all
  namespace: kube-system
spec:
  podSelector:
    matchLabels:
      k8s-app: kube-apiserver
  ingress:
  - from:
    - ipBlock:
        cidr: 0.0.0.0/0
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
```

- 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 # 35
SIMULATION
Cluster: qa-cluster
Master node: master Worker node: worker1
You can switch the cluster/configuration context using the following command:
[desk@cli] $ kubectl config use-context qa-cluster
Task:
Create a NetworkPolicy named restricted-policy to restrict access to Pod product running in namespace dev.
Only allow the following Pods to connect to Pod products-service:
1. Pods in the namespace qa
2. Pods with label environment: stage, in any namespace

Answer:

Explanation:
See the Explanation belowExplanation:

```
candidate@cli:~$ kubectl config use-context KSSH00301
Switched to context "KSSH00301".
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ kubectl get ns dev-team --show-labels
NAME       STATUS   AGE     LABELS
dev-team   Active   6h39m   environment=dev,kubernetes.io/metadata.name=dev-team
candidate@cli:~$ kubectl get pods -n dev-team --show-labels
NAME            READY   STATUS    RESTARTS   AGE     LABELS
users-service   1/1     Running   0          6h40m   environment=dev
candidate@cli:~$ ls
KSCH00301  KSMV00102  KSSC00301  KSSH00401      test-secret-pod.yaml
KSCS00101  KSMV00301  KSSH00301  password.txt   username.txt
candidate@cli:~$ vim np.yaml
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: pod-access
  namespace: dev-team
spec:
  podSelector:
    matchLabels:
      environment: dev
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              environment: dev
          - podSelector:
              matchLabels:
                environment: testing
```

```
candidate@cli:~$ vim np.yaml
candidate@cli:~$ cat np.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: pod-access
  namespace: dev-team
spec:
  podSelector:
    matchLabels:
      environment: dev
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              environment: dev
        - podSelector:
            matchLabels:
              environment: testing
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ kubectl create -f np.yaml -n dev-team
networkpolicy.networking.k8s.io/pod-access created
candidate@cli:~$ kubectl describe netpol -n dev-team
Name:         pod-access
Namespace:    dev-team
Created on:   2022-05-20 15:15:38 +0000 UTC
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     environment=dev
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: environment=dev
    From:
      PodSelector: environment=testing
  Not affecting egress traffic
  Policy Types: Ingress
candidate@cli:~$ cat KSSH00301/network-policy.yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ""
  namespace: ""
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from: []
    - from: []
candidate@cli:~$ cp np.yaml KSSH00301/network-policy.yaml
candidate@cli:~$ cat KSSH00301/network-policy.yaml
```

NEW QUESTION # 36
You are running a web application in a Kubernetes cluster using a Deployment. You want to implement a security measure to ensure that the application container only has access to the necessary system calls and files. You're worried about potential exploits that could give the container excessive privileges. Explain how you would use Seccomp profiles to achieve this, and provide an example Seccomp profile using a JSON format.

Answer:

Explanation:
Solution (Step by Step) :
1. Understand Seccomp: Seccomp (Secure Computing Mode) is a Linux kernel feature that allows you to restrict the system calls that a process can make. You can define a profile that lists the allowed system calls, effectively creating a "sandbox" for the container.
2. Create a Seccomp Profile: You can create a Seccomp profile in a JSON format. Here's an example:

```
json

defaultAction": "KILL",
syscalls": [

name": "open",
action": "ALLOW",
args": [

index": 0,
value": "/var/run/secrets/kubernetes.io/serviceaccount",
op": "EQ"

name": "read",
action": "ALLOW"

name": "write",
action": "ALLOW"

name": "stat",
action": "ALLOW"

name": "lstat",
action": "ALLOW"

name": "fstat",
action": "ALLOW"

name": "fstatfs",
action": "ALLOW"

name": "getdents64",
action": "ALLOW"

name": "getuid",
action": "ALLOW"

name": "geteuid",
action": "ALLOW"

name": "getgid",
action": "ALLOW"

name": "getegid",
action": "ALLOW"

name": "getppid",
action": "ALLOW"

name": "clock_gettime",
action": "ALLOW"

name": "gettimeofday",
action": "ALLOW"

name": "exit",
action": "ALLOW"

name": "exit_group",
action": "ALLOW"

name": "kill",
action": "ALLOW"
```


3. Apply the Seccomp Profile: You can apply the Seccomp profile to your container using the 'securitycontext' field in your Deployment YAML.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: my-web-app
        image: my-web-app:latest
        securityContext:
          seccompProfile:
            localhost:
              type: SeccompProfileLocalhost
              localSocket: /var/run/seccomp
        ports:
        - containerPort: 80
      volumes:
      - name: secret-volume
        secret:
          secretName: my-app-secret
```

4. Test and Verify: After deploying your Deployment, test your application and make sure it functions as expected. You can verity that the Seccomp profile is working by attempting to run commands within the container that are not allowed by your profile.

NEW QUESTION # 37
You have a Kubernetes cluster running a web application deployment named 'web-app' that uses a service account called 'web-app-sa' The 'web-app-sa' has been granted the necessary RBAC roles and permissions to access specific resources in the cluster. You want to implement a strategy to prevent the 'web-app' deployment from using unauthorized service accounts that might be accidentally created or added to the deployment spec.

Answer:

Explanation:
Solution (Step by Step) :
1. Create a Service Account for the Web Applicatiom
- Create a Service Account YAML file named 'web-app-sa.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: web-app-sa
  namespace: default # Replace with your namespace
```

2. Create a Role for the Service Account: - Create a Role YAML file named 'web-app-role.yaml to grant the necessary permissions to the 'web-app-sa':

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: web-app-role
  namespace: default # Replace with your namespace
rules:
- apiGroups: ["apps"] # Allow managing deployments
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "delete", "patch"]
- apiGroups: ["apps"]
  resources: ["deployments/status"] # Allow viewing the status of deployments
  verbs: ["get"]
# Add any other necessary permissions for your application
```

3. Bind the Role to the Service Account: - Create a ROIeBinding YAML file named 'web-app-rolebinding.yamr to bind the 'web-app-roles to the 'web-app-sa':

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: web-app-rolebinding
  namespace: default # Replace with your namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: web-app-role
subjects:
- kind: ServiceAccount
  name: web-app-sa
  namespace: default # Replace with your namespace
```

4. Create tne Web Application Deployment: - Create a Deployment YAML file named 'web-app-deployment.yaml that specifies the 'web-app-sa' and any other necessary configuration:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      serviceAccountName: web-app-sa
      containers:
      - name: web-app-container
        image: nginx:latest
```

5. Apply the Service Account, Role, RoleBinding, and Deployment: - Apply the YAML files using kubectl apply -f web-app-sa.yaml web-app-role.yaml web-app-rolebinding.yaml web-app-deployment.yaml 6. Test With unauthorized Service Accounts: - Try creating a new Service Account (e.g., 'unauthorized-sa') and adding it to the 'web-app-deployment YAML file. - Try updating the deployment. This should fail because the unauthorized service account does not have the necessary permissions. - You can also try creating a pod with the unauthorized service account to see that it cannot access resources it doesn't have permission for. By following these steps, you effectively enforce a policy that ensures the 'web-app' deployment only uses the authorized 'web-app-sa' for resource access, mitigating the risks associated with unauthorized service account usage.

NEW QUESTION # 38
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:
$ 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. Reference: https://kubernetes.io/docs/tasks/debug-application-cluster/audit/ Note: log volume & policy volume is already mounted in vim /etc/kubernetes/manifests/kube-apiserver.yaml so no need to mount it. Reference: https://kubernetes.io/docs/tasks/debug-application-cluster/audit/

NEW QUESTION # 39
.....

The language of our CKS study materials is simple. The learners may come from many social positions and their abilities to master our CKS study materials are varied. Based on this consideration we apply the most simple and easy-to-be-understood language to help the learners no matter he or she is the students or the in-service staff, the novice or the experienced employee which have worked for many years. CKS Study Material use the simple language to explain the answers and detailed knowledge points and the concise words to show the complicated information about the CKS study material.