最受歡迎的CKS考試備考經驗，覆蓋大量的Linux Foundation認證CKS考試知識點

P.S. PDFExamDumps在Google Drive上分享了免費的、最新的CKS考試題庫：https://drive.google.com/open?id=1sDZAqtdUbbx1Nn89GpKwovcunsdS0zBK

為什麼大多數人選擇PDFExamDumps，是因為PDFExamDumps的普及帶來極大的方便和適用。是通過實踐檢驗了的，PDFExamDumps提供 Linux Foundation的CKS考試認證資料是眾所周知的，許多考生沒有信心贏得 Linux Foundation的CKS考試認證，擔心考不過，所以你得執行PDFExamDumps Linux Foundation的CKS的考試培訓資料，有了它，你會信心百倍，真正的作了考試準備。

該考試設計成有挑戰性的，需要高水平的 Kubernetes 安全專業知識和經驗。候選人需要在各種領域展示他們的知識和技能，包括 Kubernetes 網絡安全、身份驗證和授權、容器安全以及 Kubernetes 安全工具。

CKS 認證旨在針對 Kubernetes 和容器化應用程式的 IT 專業人員，包括安全專業人員、DevOps 工程師、系統管理員和開發人員。該認證要求候選人展示他們在各種 Kubernetes 安全主題上的專業知識，例如保護 Kubernetes 元件、保護容器映像、保護網路通訊以及實施安全政策。

>> CKS考試備考經驗 <<

CKS在線題庫，CKS認證考試

在如今競爭激烈的IT行業中，通過了Linux Foundation CKS 認證考試是有很多好處的。因為有了Linux Foundation CKS 認證證書就可以提高收入。拿到了Linux Foundation CKS 認證證書的人往往要比沒有證書的同行工資高很多。可是Linux Foundation CKS 認證考試不是很容易通過的，所以PDFExamDumps是一個可以幫助你增長收入的網站.

Linux Foundation的CKS（Certified Kubernetes Security Specialist）認證考試是IT專業人士極為追求的認證，以展示他們在保障Kubernetes集群方面的專業知識和能力。Kubernetes是一個廣泛用於容器編排和管理的開源平臺。然而，像任何技術一樣，使用它存在安全風險。CKS考試旨在測試個人保障Kubernetes集群和工作負載的能力。

最新的 Kubernetes Security Specialist CKS 免費考試真題 (Q12-Q17):

問題 #12
Create a User named john, create the CSR Request, fetch the certificate of the user after approving it.
Create a Role name john-role to list secrets, pods in namespace john
Finally, Create a RoleBinding named john-role-binding to attach the newly created role john-role to the user john in the namespace john.
To Verify: Use the kubectl auth CLI command to verify the permissions.

答案：

解題說明：
se kubectl to create a CSR and approve it.
Get the list of CSRs:
kubectl get csr

Approve the CSR:
kubectl certificate approve myuser
Get the certificate
Retrieve the certificate from the CSR:
kubectl get csr/myuser -o yaml
here are the role and role-binding to give john permission to create NEW_CRD resource:
kubectl apply -f roleBindingJohn.yaml --as=john
rolebinding.rbac.authorization.k8s.io/john_external-rosource-rb created kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: john_crd
namespace: development-john
subjects:
- kind: User
name: john
apiGroup: rbac.authorization.k8s.io
roleRef:
kind: ClusterRole
name: crd-creation
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: crd-creation
rules:
- apiGroups: ["kubernetes-client.io/v1"]
resources: ["NEW_CRD"]
verbs: ["create, list, get"]

問題 #13
Describe now you would design a security posture for a Kubernetes cluster using the CIS Kubernetes Benchmark as a guideline. Include key areas to focus on, relevant security controls, and how you would monitor and enforce compliance with the benchmark.

答案：

解題說明：
Solution (Step by Step) :
1. Review CIS Kubernetes Benchmark:
- Thoroughly familiarize yourself With the CIS Kubernetes Benchmark, which outlines security best practices and controls.
2. Assess Current Security Posture:
- Audit the current security configuration of your Kubernetes cluster against the CIS benchmark. This includes:
- Cluster Access Control: Verity that access iS restricted to authorized users and accounts.
- Authentication and Authorization: Ensure that strong authentication mechanisms are in place and that roles are properly assigned.
- Image Security: Review the security of images used in your deployments, ensuring they are from trusted sources and have appropriate security measures.
- Network Security: Implement network policies to restrict communication between pods and enforce least-privilege access.
- Pod Security: Define PodSecurityPoIicies to control resources and capabilities available to pods.
- Logging and Monitoring: Configure robust logging and monitoring systems to detect and respond to security incidents.
3. Develop Security Controls:
- Implement security controls based on the CIS benchmark findings. This may include:
- RBAC (Role-Based Access Control): Use RBAC to define granular permissions for users and service accounts.
- Network Policies: Implement network policies to restrict inter-pod communication and external access.
- Admission Controllers: Use admission controllers like PodSecurityPolicy and NetworkPolicy to enforce security policies before deployments are allowed.
- Image Scanning: Regularly scan container images for vulnerabilities.
- Secret Management: Securely manage and store sensitive information using Kubernetes Secrets.
- Logging and Monitoring: Configure centralized logging and monitoring systems to track activity and identity security events.
4. Monitor and Enforce Compliance:
- Continuously monitor the cluster's security posture against the CIS benchmark using tools like:
- Kube-bench: A tool for assessing Kubernetes security posture.
- CIS Kubernetes Benchmark Scanner A dedicated scanner for compliance checks.
- Custom Monitoring Tools: Develop custom tools to monitor specific aspects of the cluster.

- Implement mechanisms to automate security cnecks and enforce compliance. This could involve:
- Automated Security Scanning: Schedule regular security scans.
- Alerting: Configure alerts for security events and non-compliant configurations.
- Remediation: Implement automated remediation actions for security vulnerabilities.
5. Continuous Improvement:
- Regularly review and update the security posture to stay ahead of evolving threats.
- Keep up with the latest security recommendations and updates to the CIS Kubernetes Benchmark.
- Conduct security training for team members to promote awareness and best practices.

問題 #14
Your Kubernetes cluster hosts a sensitive application that uses secrets for storing critical data. You need to implement a robust security measure to ensure that these secrets are protected from unauthorized access.

答案：

解題說明：
Solution (Step by Step):
1. Use Kubernetes Secret Manager Leverage Kubernetes' built-in secret management capabilities to store and manage sensitive data.
- Create a Secret:
2. Restrict Access to Secrets: use R8AC (Role-Based Access Control) to limit access to secrets to authorized users or applications. Create custom roles or cluster roles that allow specific access to secrets based on your security needs. - Create a YAML file for the Custom Role:
- Create a RoleBinding:
3. Mount Secret to Pods: Mount the secret to the pods that require access to the sensitive data. You can use volume mounts in your pod definitions. - Example Pod YAML:
4. Limit Access within Pods: use environment variables or other security mechanisms within your pods to limit access to the secrets to only the necessary code components.

問題 #15
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.

答案：

解題說明：
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.

問題 #16
SIMULATION
You can switch the cluster/configuration context using the following command:
[desk@cli] $ kubectl config use-context dev
A default-deny NetworkPolicy avoid to accidentally expose a Pod in a namespace that doesn't have any other NetworkPolicy defined.
Task: Create a new default-deny NetworkPolicy named deny-network in the namespace test for all traffic of type Ingress + Egress
The new NetworkPolicy must deny all Ingress + Egress traffic in the namespace test.
Apply the newly created default-deny NetworkPolicy to all Pods running in namespace test.
You can find a skeleton manifests file at /home/cert_masters/network-policy.yaml

答案：

解題說明：
See the Explanation below
Explanation:
master1 $ k get pods -n test --show-labels
NAME READY STATUS RESTARTS AGE LABELS
test-pod 1/1 Running 0 34s role=test,run=test-pod
testing 1/1 Running 0 17d run=testing
$ vim netpol.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: deny-network
namespace: test
spec:
podSelector: {}
policyTypes:
- Ingress
- Egress
master1 $ k apply -f netpol.yaml
Explanation:
controlplane $ k get pods -n test --show-labels
NAME READY STATUS RESTARTS AGE LABELS
test-pod 1/1 Running 0 34s role=test,run=test-pod
testing 1/1 Running 0 17d run=testing
master1 $ vim netpol1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: deny-network
namespace: test
spec:
podSelector: {}
policyTypes:
- Ingress
- Egress
master1 $ k apply -f netpol1.yaml

Reference:
https://kubernetes.io/docs/concepts/services-networking/network-policies/ Explanation:
controlplane $ k get pods -n test --show-labels
NAME READY STATUS RESTARTS AGE LABELS
test-pod 1/1 Running 0 34s role=test,run=test-pod
testing 1/1 Running 0 17d run=testing
master1 $ vim netpol1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: deny-network
namespace: test
spec:
podSelector: {}
policyTypes:
- Ingress
- Egress
master1 $ k apply -f netpol1.yaml
https://kubernetes.io/docs/concepts/services-networking/network-policies/

問題 #17
.....