# 唯一無二CKS模擬対策 |素晴らしい合格率のCKS Exam |素敵なCKS: Certified Kubernetes Security Specialist (CKS)


P.S. Tech4ExamがGoogle Driveで共有している無料かつ新しいCKSダンプ：https://drive.google.com/open?id=1TVGaHfdLwc53wxa0mxjKf1ZZYHJezlNh

自分の幸せは自分で作るものだと思われます。ただ、社会に入るIT卒業生たちは自分能力の不足で、CKS試験向けの仕事を探すのを悩んでいますか？それでは、弊社のLinux FoundationのCKS練習問題を選んで実用能力を速く高め、自分を充実させます。その結果、自信になる自己は面接のときに、面接官のいろいろな質問を気軽に回答できて、順調にCKS向けの会社に入ります。

CKS試験は、候補者が現実のシナリオでKubernetesセキュリティ原則を適用する能力を評価するパフォーマンスベースの評価です。試験は、Kubernetesクラスタのセットアップ、RBAC認証、ネットワークポリシー、コンテナセキュリティ、その他のセキュリティベストプラクティスを含む、幅広いトピックをカバーしています。これは、実際にKubernetes環境を使用してさまざまなタスクを実行する必要があるハンズオン試験です。

**>> CKS模擬対策 <<**

## CKS を効率よく取得したい人に一番お勧めしたい一冊です。

今後のCKS学習教材について心配がある場合は、学習教材が問題の解決に役立ちます。弊社のCKS学習教材の高品質をお約束するために、当社には優れた技術スタッフがおり、販売後の完璧なサービスシステムがあります。さらに重要なことは、当社のCKSガイド質問と完璧なアフターサービスが、地元および海外のお客様に認められていることです。模擬試験に合格する場合は、学習エンジンが必須の選択肢になると考えています。過去数年間、CKSガイドの質問を購入する人が増えています。

Linux Foundation Certified Kubernetes Security Specialist（CKS）試験は、Kubernetesのセキュリティ専門家の専門知識を検証する認定です。この認定試験は、安全なKubernetesクラスターを設計、展開、管理できる専門家の知識、スキル、能力をテストするように設計されています。 CKS認定試験は、候補者がKubernetesのセキュリティ原則とベストプラクティスの事前知識と経験を持つことを要求する高度なレベルの認定です。

Linux Foundation Certified Kubernetes Security Specialist（CKS）試験は、コンテナベースのアプリケーションとKubernetesプラットフォームを保護する候補者の知識とスキルをテストするように設計されています。 Kubernetesは、コンテナオーケストレーションと管理の事実上の基準となり、この技術を採用する組織が増えるにつれて、これらの環境を確保および管理できる訓練を受けた専門家の必要性が重要になります。 CKS認定は、Kubernetesのセキュリティの概念と実践における候補者の習熟度を示す業界に認識された資格情報です。

## Linux Foundation Certified Kubernetes Security Specialist (CKS) 認定 CKS 試験問題 (Q130-Q135):

**質問 # 130**
You are tasked with implementing a security policy that prohibits the use of privileged containers in your Kubernetes cluster.

Implement a solution that uses KubeLinter to enforce this policy by automatically scanning all deployments and preventing deployments that violate the policy.

**正解：**

解説：
Solution (Step by Step):
1. Install KubeLinter: Download and install the 'kubevar binary from the official GitHub repository.
2. Create a custom KubeLinter check: Define a custom check that prohibits the use of privileged containers. This check can be defined in a separate
YAML file or embedded in your '.kubeval.yaml configuration file.

```
# custom-checks.yaml
privileged-container:
  message: "Privileged containers are not allowed."
  path: spec.template.spec.containers[].securityContext.privileged
  rule:
    type: 'anyOf'
    conditions:
      - type: 'isNull'
      - type: 'isFalse'
```

3. Configure KubeLinter to use the custom check: Add the custom check to your .kuoeval.yaml configuration file.

```
extends:
  - ./custom-checks.yaml
```

4. Integrate KubeLinter into your CI/CD pipeline: Add a step to your pipeline that runs KubeLinter against your deployment YAML manifests. This step should be executed before the manifests are deployed to the cluster.

```
# .gitlab-ci.yml
stages:
  - validate
  - deploy

kubeval:
  stage: validate
  image: ghcr.io/stackrox/kubeval:latest
  script:
    - kubeval --strict --config .kubeval.yaml .yaml
  allow_failure: false
```

5. (Optional) Implement an admission controller: For real-time enforcement, deploy an admission controller that uses KubeLinter to validate deployments as they are created or updated. This will prevent any deployments that violate the policy from being created in the cluster. Tools like Kyverno or Gatekeeper can be used to create and enforce such policies.

**質問 # 131**
Your Kubernetes cluster iS running a web application that requires access to a database hosted on an external Cloud provider. Describe how you can secure the connection between the application and the database using TLS/SSL encryption and identity-based authentication.

**正解：**

解説：
Solution (Step by Step) :
1. Configure TLS/SSL Encryption:
- Generate Certificate: Obtain a TLS/SSL certificate from a trusted certificate authority (CA) or use a self-signed certificate for development purposes-
- Install Certificate on Database Server: Install the certificate on the database server, making it available to the database service.
- Configure Database Service: Configure the database service to accept connections only over TLS/SSL.
- Configure Application Container:
- Mount Certificate: Mount the TLS/SSL certificate into the application container as a secret.
- Configure Application Code: Update the application code to use the certificate when connecting to the database.
2. Implement Identity-Based Authentication:
- Create Database User: Create a dedicated database user specifically for the web application.
- Grant Permissions: Grant appropriate permissions to the database user, limiting access to the necessary tables and data.
- Use Authentication Plugin: Configure the database service to use an authentication plugin that supports identity-based authentication.
- Generate Database Credentials: Generate database credentials (username and password) for the application.
- Store Credentials Secretly: Store the database credentials securely as a Kubernetes secret.
- Access Credentials from Application: Configure the application to access the database credentials from the secret.
3. Connect Application to Database:

- Configure Connection String: Update the application's connection string to use TLS/SSL and the database user credentials.
- Example Connection String:
jdbc:postgresql://database-host:5432/database-name?ssl=true&sslmode=require&user=app user&password=app-password
4. Security Considerations:
- Certificate Validation: Ensure the certificate is validated by the application to prevent man-in-the-middle attacks.
- Secure Credential Management: Implement strong security measures to protect the database credentials stored as secrets.
- Access Control: Limit access to the database to only authorized users and applications.
- Network Isolatiom Consider using network policies to isolate the web application from other workloads and restrict unnecessary network traffic.

**質問 # 132**
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

**正解：**

解説：
$ k get ns qa --show-labels
NAME STATUS AGE LABELS
qa Active 47m env=stage
$ k get pods -n dev --show-labels
NAME READY STATUS RESTARTS AGE LABELS
product 1/1 Running 0 3s env=dev-team
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: restricted-policy
namespace: dev
spec:
podSelector:
matchLabels:
env: dev-team
policyTypes:
- Ingress
ingress:
- from:
- namespaceSelector:
matchLabels:
env: stage
- podSelector:
matchLabels:
env: stage
[desk@cli] $ k get ns qa --show-labels
NAME STATUS AGE LABELS
qa Active 47m env=stage
[desk@cli] $ k get pods -n dev --show-labels
NAME READY STATUS RESTARTS AGE LABELS
product 1/1 Running 0 3s env=dev-team
[desk@cli] $ vim netpol2.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: restricted-policy

namespace: dev
spec:
podSelector:
matchLabels:
env: dev-team
policyTypes:
- Ingress
ingress:
- from:
- namespaceSelector:
matchLabels:
env: stage
- podSelector:
matchLabels:
env: stage
[desk@cli] $ k apply -f netpol2.yaml Reference: https://kubernetes.io/docs/concepts/services-networking/network-policies/
[desk@cli] $ k apply -f netpol2.yaml Reference: https://kubernetes.io/docs/concepts/services-networking/network-policies/

**質問 # 133**
You have a Kubernetes cluster running a critical application with a Deployment named 'myapp-deployment. You suspect a recent image update has introduced a vulnerability that's causing the application to crash frequently.
You need to investigate this issue and determine the exact phase of the attack and the potential bad actor responsible. You have access to the following resources: Kubernetes audit logs: Enabled at the cluster level.
Container logs: Available for all pods associated with the 'myapp-deployments Network traffic logs: Captured by a network security solution. How would you use these resources to identify the attack phase, the potential bad actor, and the source of the vulnerability?

**正解：**

解説：
Solution (Step by Step) :
1. Analyze Kubernetes Audit Logs:
Focus on events related to the 'myapp-deployment: Search for entries related to pod creation, deletion, image pulls, and resource updates. Look for suspicious activity: Pay attention to any unusual image updates, unauthorized access attempts, or resource changes that occurred around the time of the crashes.
Identify the user or service account responsible for the changes: This could point to a potential bad actor if the user'service account is not expected to modify the Deployment.
2. Examine Container Logs:
Search for crash messages and error codes: This will provide insights into the specific cause of the application crashes.
Identify any unusual or suspicious activity within the container: Look for signs of malicious processes, unauthorized network connections, or data exfiltration attempts.
3. Analyze Network Traffic Logs:
Identify the source of the compromised image: Network logs can reveal the IP address of the registry or repository from which the vulnerable image was pulled.
Examine network connections from the affected pods: Look for unusual or unauthorized outbound connections that could indicate malware or communication with a malicious server.
4. Correlate Findings:
Combine information from the different logs to build a comprehensive picture of the attack.
For example, if you find a suspicious image pull in the audit logs, and the container logs show signs of malware activity, you have strong evidence of malicious image vulnerability.
Example Code Snippets:
Kubernetes Audit Logs (using kubectl):
bash
kubectl logs -f -n kube-system kube-apiserver -c kube-apiserver | grep "myapp-deployment" | grep "Create" | grep "Image"
Container Logs (using kubectl):
bash
kubectl logs -f myapp-deployment-pod-name -c myapp
Network Traffic Logs (using a network security tool like Falco):
falco -f falco.yaml -o json

Note: The specific commands and tools may vary depending on your Kubernetes environment and security tools.

**質問 # 134**
You have a Kubernetes cluster running a web application. You want to enforce secure communication between the web server pods and the database pods in a separate namespace. How would you implement this using TLS certificates and Secrets?

**正解：**

解説：
Solution (Step by Step):
1. Generate TLS Certificates: Generate a certificate authority (CA) certificate and server/client certificates.
- You can use tools like OpenSSL or Let's Encrypt to generate these certificates-
2. Create Secrets: Create Kubernetes Secrets to store the certificates.
- Secret for CA Certificate: Create a Secret with the CA certificate and private key.
- Secret for Server Certificate: Create a Secret With the server certificate and private key.
- Secret for Client Certificate: Create a Secret with the client certificate and private key (optional, if you want to enforce client authentication).
3. Mount Certificates: Mount the Secrets containing the certificates into the pods.
- Web Server Pods: Mount the CA certificate and server certificate Secret
- Database Pods: Mount the CA certificate and client certificate Secret (optional, if you want to enforce client authentication).
4. Configure TLS: Configure your web server and database applications to use the mounted certificates for TLS communication.
- Web Server: Configure it to use the server certificate and private key for HTTPS communication.
- Database: Configure it to accept TLS connections and use the client certificate (if client authentication is enabled).
Example using OpenSSL for generating certificates and Kubernetes Secrets:
Generating Certificates:
bash
# Generate a CA certificate and key
openssl req -x509 -newkey rsa:2048 -keyout ca.key -out ca.crt \
-days 365 -nodes -subj "/C=US/ST=CA/L=Los Angeles/O=Example Inc./CN=Example CA"
# Generate a server certificate and key
openssl req -newkey rsa:2048 -keyout server.key -out server.csr \
-subj Angeles/O=Example Inc./CN=example.com"
openssl x509 -req -in server.csr -CA cmcrt -CAkey cakey -CAcreateserial \
-out server.cn -days 365 -sha256 -extensions v3_req
# Generate a client certificate and key (optional)
openssl req -newkey rsa:2048 -keyout client.key -out client_csr \
-subj Angeles/O=Example Inc./CN=client.example.com"
openssl x509 -req -in client.csr -CA ca.crt -CAkey cakey -CAcreateseriaI
-out client.crt -days 365 -sha256 -extensions v3_req
Creating Secrets:

```
# Secret for CA certificate
apiVersion: v1
kind: Secret
metadata:
  name: ca-cert
  namespace:
type: Opaque
data:
  ca.crt:
  ca.key:

# Secret for server certificate
apiVersion: v1
kind: Secret
metadata:
  name: server-cert
  namespace:
type: Opaque
data:
  server.crt:
  server.key:

# Secret for client certificate (optional)
apiVersion: v1
kind: Secret
metadata:
  name: client-cert
  namespace:
type: Opaque
data:
  client.crt:
  client.key:
```



Mounting Secrets in Pods: - Web Server Pod: Mount the 'ca-cen' and 'server-cert Secrets. - Database Pod: Mount the 'ca-cert' and 'client-cert Secrets (if client authentication is enabled). Important Notes: - This implementation assumes you have the necessary knowledge about TLS certificates and secrets management in Kubernetes. - You need to configure your web server and database applications to use the certificates and enforce TLS communicatiom - Ensure the security of your certificates and private keys, as they are critical for secure communication.

**質問 # 135**

.....

問題集をダウンロードせよCKSリンクグローバル

- CKS日本語関連対策 □ CKS予想試験 □ CKSリンクグローバル □ ➤ www.jpshiken.com □サイトにて（CKS）問題集を無料で使おうCKSオンライン試験
- www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, www.stes.tyc.edu.tw, lms.ait.edu.za, www.stes.tyc.edu.tw, ncon.edu.sa, Disposable vapes

さらに、Tech4ExamCKSダンプの一部が現在無料で提供されています：https://drive.google.com/open?id=1TVGaHfdLwc53wxa0mxjKf1ZZYHJezlNh