LinuxFoundation.CKS.v2026-08-01.q72

試験コード：	CKS
試験名称：	Certified Kubernetes Security Specialist (CKS)
認証ベンダー：	Linux Foundation
無料問題の数：	72
バージョン：	v2026-08-01
ページの閲覧量：	106
問題集の閲覧量：	794

https://www.jpnshiken.com/shiken/LinuxFoundation.CKS.v2026-08-01.q72.html

質問: **1**
シミュレーション
コンテクスト
テスト目的で、kubeadm がプロビジョニングしたクラスターの API サーバー
認証されていないアクセスや不正アクセスを許可するように設定されていた。
タスク
まず、クラスターのAPIサーバーを以下のように設定してセキュリティを確保します。
匿名認証を禁止する
認証モードとしてNode、RBACを使用する
アドミッションコントローラNodeRestrictionを使用する
このクラスターは、コンテナのランタイムとしてDocker Engineを使用します。必要に応じて、dockerコマンドを使用して実行中のコンテナのトラブルシューティングを行ってください。
kubectl は認証なしおよび不正アクセスを使用するように設定されています。この設定を変更する必要はありませんが、クラスターのセキュリティを強化すると kubectl が動作しなくなることに注意してください。
セキュアなクラスターにアクセスするには、etc/kubernetes/admin.conf にあるクラスターの元の kubectl 設定ファイルを使用できます。
次に、クリーンアップするために ClusterRoleBinding を削除します。
システム:匿名。
正解:
完全な解決策については、以下の説明をご覧ください。
Explanation:
1) コントロールプレーンノードへのSSH接続
ssh cks000002
sudo -i
2) APIサーバーの静的ポッドマニフェストを編集する
kubeadm の API サーバーは静的 Pod として実行されます。
/etc/kubernetes/manifests/kube-apiserver.yaml 内
3) 必要なAPIサーバーのセキュリティ設定を適用する
3.1 匿名認証を禁止する
コマンド: セクションを検索し、この行が存在することを確認してください。
--anonymous-auth=false
3.2 認証モードNode、RBACを使用する
以下の行が必ず存在することを確認してください（AlwaysAllow は無効にしてください）。

--authorization-mode=Node,RBAC
❌ 存在する場合は削除してください:
--authorization-mode=AlwaysAllow
3.3 アドミッションコントローラのNodeRestrictionを有効にする
--enable-admission-plugins を見つけて、NodeRestriction が含まれていることを確認してください。
正しい例：
--enable-admission-plugins=NodeRestriction
他のプラグインが既に存在する場合は、NodeRestriction を追加します。例:
--enable-admission-plugins=NamespaceLifecycle,ServiceAccount,NodeRestriction
4) ファイルを保存し、kubelet が API サーバーを再起動するまで待ちます。
保存して終了するだけです（:wq）
KubeletはAPIサーバーのポッドを自動的に再起動します。
5) kubectlをセキュアな設定に切り替える
APIサーバーのセキュリティ強化後、現在のkubectlは動作しなくなります。
export KUBECONFIG=/etc/kubernetes/admin.conf
アクセスを確認する：
kubectl get nodes
6) 安全でない ClusterRoleBinding を削除する
システムを削除:匿名バインディング:
kubectl delete clusterrolebinding system:anonymous
削除を確認する：
kubectl get clusterrolebinding | grep anonymous
（出力なし＝正解）
7) クイック検証（オプションだが高速）
APIサーバーフラグのチェック:
grep -n "anonymous-auth" /etc/kubernetes/manifests/kube-apiserver.yaml
grep -n "authorization-mode" /etc/kubernetes/manifests/kube-apiserver.yaml grep -n "NodeRestriction" /etc/kubernetes/manifests/kube-apiserver.yaml

質問: **2**
次のコマンドを使用して、クラスター/構成コンテキストを切り替えることができます。[desk@cli] $ kubectl config use-context dev デフォルトの拒否 NetworkPolicy は、他の NetworkPolicy が定義されていない名前空間で Pod を誤って公開することを回避します。
タスク: 名前空間 test に、Ingress + Egress タイプのすべてのトラフィックに対して deny-network という名前の新しいデフォルト拒否 NetworkPolicy を作成します。新しい NetworkPolicy は、名前空間 test 内のすべての Ingress + Egress トラフィックを拒否する必要があります。
新しく作成したデフォルト拒否ネットワークポリシーを、test 名前空間で実行されているすべての Pod に適用します。
スケルトンマニフェストファイルは /home/cert_masters/network-policy.yaml にあります。
正解:
master1 $ k get pods -n test --show-labels
名前 準備完了 ステータス 再起動 年齢 ラベル
test-pod 1/1 実行中 0 34秒 role=test,run=test-pod
テスト中 1/1 実行中 0 17d 実行=テスト中
$ vim netpol.yaml

apiVersion: networking.k8s.io/v1
種類: ネットワークポリシー
メタデータ:
名前: deny-network
名前空間: テスト
仕様:
podSelector: {}
ポリシーの種類:
- イングレス
- 出口
master1 $ k apply -f netpol.yaml
説明
コントロールプレーン $ k get pods -n test --show-labels
名前 準備完了 ステータス 再起動 年齢 ラベル
test-pod 1/1 実行中 0 34秒 role=test,run=test-pod
テスト中 1/1 実行中 0 17d 実行=テスト中
master1 $ vim netpol1.yaml
apiVersion: networking.k8s.io/v1
種類: ネットワークポリシー
メタデータ:
名前: deny-network
名前空間: テスト
仕様:
podSelector: {}
ポリシーの種類:
- イングレス
- 出口
master1 $ k apply -f netpol1.yaml 参照: https://kubernetes.io/docs/concepts/services-networking/network-policies/ 説明 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:
名前: deny-network
名前空間: テスト
仕様:
podSelector: {}
ポリシーの種類:
- イングレス
- 出口
master1 $ k apply -f netpol1.yaml 参考: https://kubernetes.io/docs/concepts/services-networking/network-policies/

質問**: 3**

Kubernetes クラスターでは、複数のマイクロサービスが別々の名前空間にデプロイされています。「web-app」名前空間内の特定のマイクロサービスが、「api-gateway」名前空間内のサービスとのみ通信できるようにしたいと考えています。NetworkPolicies を使用して、これをどのように実現できますか？

正解:

解決策（六ステップバイステップ）：

1. 対象サービスの特定: アクセス制限が必要な「web-app」名前空間内の特定のマイクロサービスを特定します。ここでは、そのマイクロサービスの名前を「web-service」とします。

2 ネットワークポリシーの作成: 許可される通信を定義するために、「web-service-access-yamr」という名前の NetworkPolicy YAML ファイルを作成します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-service-access
  namespace: web-app
spec:
  podSelector:
    matchLabels:
      app: web-service
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          namespace: api-gateway
```


- このポリシーにより、「web-app」名前空間の「web-services」ポッドが「api-gateways」名前空間のサービスと通信できるようになります。3. ネットワーク ポリシーの適用:「kubectr」を使用して NetworkPolicy を適用します。bash kubectl apply -f web-service-access-yaml 4. ネットワーク ポリシーの確認: NetworkPolicy が適用されていることを確認します。bash kubectl get networkpolicies -n web-app 5. アクセスのテスト:「web-apps」名前空間の「web-service」ポッドから「api-gateway」名前空間のサービスへの通信をテストします。この通信は許可されるはずです。「web-service」ポッドから他の名前空間のサービスへの通信を試します。この通信はブロックされるはずです。この NetworkPolicy は、「web-services」ポッドが「api-gateway」名前空間のサービスとのみ通信するように制限します。これにより、異なる名前空間にデプロイされたマイクロ サービス間で特定の通信パターンが効果的に強制されます。

質問: 4

組織向けにカスタムKubernetesディストリビューションを構築する場合、ディストリビューションに含まれるKubernetesバイナリの整合性を構築および検証するための安全なプロセスを確立してください。

正解:

解決策（手順別）：

1. ソースから Kubernetes をビルドする: 公式ソースコードリポジトリ (httpswgithub.com/kubernetes/kubernetesl) から Kubernetes バイナリをビルドします (https://wwwgoogle.com/url?sa=E&source=gmail&q=https://github.com/kubernetes/kubernetes))- クリーンなビルド環境と信頼できるソースコードソースを使用してください。

2. 再現可能なビルドを実装する: Bazel や Buildah など、再現可能なビルドをサポートするビルド システムを使用します。これにより、同じソース コードから常に同じバイナリ出力が生成されることが保証されます。

3. チェックサムの生成と検証 :ビルドされたすべてのバイナリのSHA-256チェックサムを生成し、安全な場所に保存します。配布物に含める前に、バイナリのチェックサムを検証してください。

4. バイナリに署名する: コード署名証明書を使用してバイナリに署名します。これにより、ユーザーはバイナリの真正性と完全性を検証できます。

5. バイナリと署名を公開する: バイナリと対応する署名を安全なリポジトリに公開します。ユーザーに明確な指示を提供します。

バイナリを使用する前に署名を確認してください。

6. 信頼できるCI/CDシステムを使用する：信頼性が高く安全なCI/CDシステムを使用して、ビルドと検証プロセスを自動化します。これにより、ビルドパイプラインの整合性とセキュリティが確保されます。

質問: 5

あなたは、コンテナイメージがプライベートレジストリからプルされるKubernetesクラスタを操作しています。セキュリティのベストプラクティスでは、承認されたレジストリからのイメージプルのみを許可するようにクラスタを構成することが推奨されています。イメージアドミッションコントローラを使用してこのポリシーを適用する方法を説明し、アドミッションコントロール構成の具体的な例を示してください。

正解:

解決策（六ステップバイステップ）：

1. イメージアドミッションコントローラーを有効にする：

- アドミッションコントローラーをインストールします。「ImagePolicyWebhook」アドミッションコントローラーは、コンテナイメージにポリシーを適用します。Kubernetesデプロイメントの一部としてインストールすることも、Helmチャートを使用してインストールすることもできます。
2. ポリシー構成を作成する：
- ポリシーの定義 :YAMLファイルを使用して、アクセス制御のルールを定義します。このポリシーでは、許可されるレジストリを指定します。
- ポリシー設定例：

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: imagepolicy-webhook
webhooks:
- name: imagepolicy.example.com
  rules:
  - apiGroups: ["apps", "extensions", "batch"] # Example: Deployments, DaemonSets, Jobs
    apiVersions: ["v1", "v1beta1", "v1beta2"]
    resources: ["pods", "deployments", "daemonsets", "jobs"]
    operations: ["CREATE", "UPDATE"]
  admissionReviewVersions: ["v1"]
  clientConfig:
    service:
      name: imagepolicy-webhook
      namespace: default
      path: /validate
  failurePolicy: Fail # Fail requests if the policy doesn't match
  sideEffects: None # Don't mutate requests
  timeoutSeconds: 2 # Timeout for requests
  # Additional settings based on the specific admission controller
```

3. サービスの構成: - サービスの作成: アドミッションコントローラのエンドポイントを公開する Kubernetes サービスを作成します。- サービス構成例:

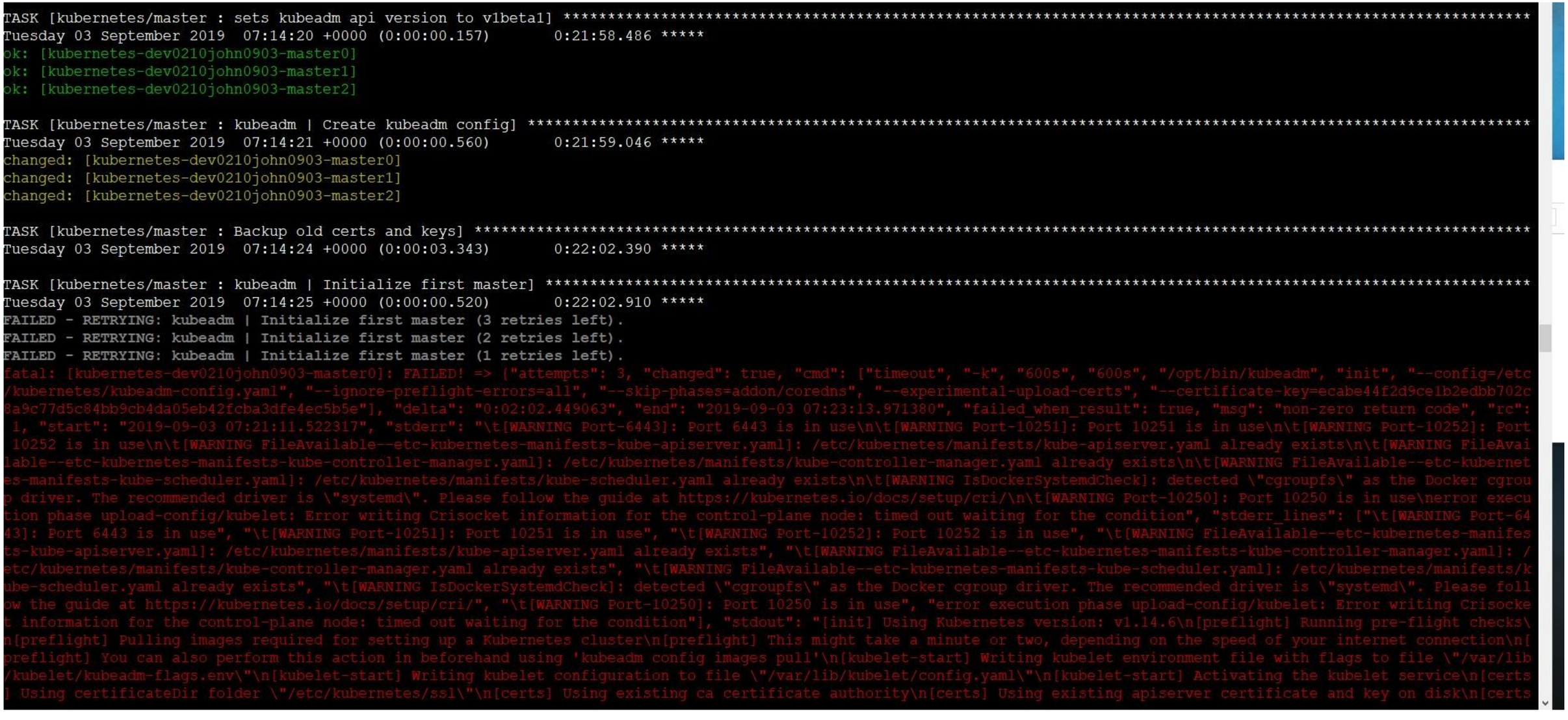
```
apiVersion: v1
kind: Service
metadata:
  name: imagepolicy-webhook
  namespace: default
spec:
  selector:
    app: imagepolicy-webhook
  ports:
  - port: 443
    targetPort: 8443
```

4. ポリシーエンジンをデプロイします。- デプロイメントを作成します。ポリシーエンジンを実行するための Kubernetes デプロイメントを作成します (例: アドミッションコントローラ ロジックを含むコンテナ イメージ)。- デプロイメント構成の例:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: imagepolicy-webhook
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: imagepolicy-webhook
  template:
    metadata:
      labels:
        app: imagepolicy-webhook
    spec:
      containers:
      - name: imagepolicy-webhook
        image:
        ports:
        - containerPort: 8443
```

5. 構成の検証: - イメージのプルをテストします: 承認済みレジストリと未承認レジストリの両方からイメージをプルします。- ポリシーの適用を監視します: アドミッションコントローラのログを監視して、未承認レジストリからのプルが正常にブロックされていることを確認します。- セキュリティを検証します: ポリシーが未承認のコンテナイメージソースの使用を効果的に防止し、クラスタのセキュリティ体制を強化していることを確認します。

質問: **6**
コンテナイメージスキャナがクラスタ上にセットアップされています。
ディレクトリ内の設定が不完全です
/etc/kubernetes/confcontrol と、HTTPS エンドポイント https://test-server.local.8081/image_policy を備えた機能的なコンテナイメージスキャナ
1. 入退室管理プラグインを有効にします。
2. コントロール構成を検証し、暗黙的拒否に変更します。
最後に、イメージタグが「latest」のポッドをデプロイして、設定をテストしてください。
正解:
ssh-add ~/.ssh/tempprivate
eval "$(ssh-agent -s)"
cd contrib/terraform/aws
vi terraform.tfvars
terraform init
terraform apply -var-file=credentials.tfvars
ansible-playbook -i ./inventory/hosts ./cluster.yml -e ansible_ssh_user=core -e bootstrap_os=coreos -b --become-user=root --flush-cache -e ansible_user=core



質問: **7**

シミュレーション

etcdに保存されているシークレットは保存時に安全ではありません。etcdctlコマンドユーティリティを使用してシークレット値を確認できます。例: ETCDCTL_API=3 etcdctl get /registry/secrets/default/cks-secret --cacert="ca.crt" --cert="server.crt" --key="server.key" 出力

暗号化構成を使用して、プロバイダーのAES-CBCとIDを使用してリソースシークレットを保護し、保存されているシークレットデータを暗号化して、すべてのシークレットが新しい構成で暗号化されるようにするマニフェストを作成します。
正解:
それについてのご意見をお聞かせください。

質問: 8
あなたは、機密データを扱うアプリケーションをホストするKubernetesクラスタ上で作業しています。クラスタにデプロイする前に、アプリケーションのコンテナイメージの静的解析を実行して、潜在的なセキュリティ脆弱性を特定する必要があります。
正解:
解決策（ステップバイステップ）：
1. 静的解析ツールを選択する：
- コンテナイメージに適した静的解析ツールを選択してください。よく使われるツールには以下のようなものがあります。
- トリビア: https://aquasecurity.github.io/trivy/
- Snyk: https://snyk.io/
- Anchore Engine: [https://anchore.com/l(https://anchore.com/)
2. ツールをインストールして設定する：
選択したツールをマシンにインストールするか、CI/CDパイプラインに統合してください。
- コンテナイメージの脆弱性をスキャンするようにツールを設定します。
3. コンテナ画像をスキャンする：
- ツールのコマンドラインインターフェースまたはAPIを使用して、コンテナイメージをスキャンします。
ツールへの入力として、画像名またはタグを指定してください。
4. 結果を分析する：
このツールは、特定された脆弱性の詳細を記載したレポートを生成します。
報告書を確認し、脆弱性の深刻度と影響度に基づいて、修復措置の優先順位を決定する。
- ツールの機能を使用して、脆弱性の状況とその修復状況を追跡します。

質問: 9
あなたはkubeadmを使用してオンプレミス環境にKubernetesクラスターをデプロイする責任があります。クラスターをデプロイする前に、kubeadm、kubelet、およびkubectlバイナリの整合性を確認してください。
正解:
解決策（手順別）：
1. バイナリのダウンロード :Kubernetesの公式リリースページから、必要なバージョンのkubeadm、kubelet、kubectlのバイナリをダウンロードしてください。
(httpswgitnub.com/kubernetes/kllbernetes,treleases](httpswwww.google.com/url?
sa=E&source=gmail&q=httpswgithub.com/kubernetes/kubernetes/releases))。
2. チェックサムを検証する :ダウンロードしたバイナリのSHA-256チェックサムを、リリースページに記載されているチェックサムと比較します。

バッシュ
sna256sum kubeadm kubelet kubectl
3. 署名の検証（オプション）より強力な保証が必要な場合は、対応する署名ファイル（.asc）をダウンロードし、以下の方法で署名を検証します。
Kubernetesの公式公開鍵。
バッシュ
gpg --verify kubeadm.sha256.asc kubeadm
4. バイナリのインストール: バイナリの整合性を確認したら、ノード上の適切な場所にインストールします。
バッシュ
sudo install -o root -g root -m 0755 cubeadm cube cubectl /usr/bin/
5. クラスターのデプロイに進みます: バイナリの検証とインストールが完了したら、kubeadm を使用して Kubernetes クラスターのデプロイに進むことができます。

質問: **10**
複数のKubernetesクラスターが存在する状況を想像してみてください。中央集権型イメージレジストリからのイメージのみをすべてのクラスターにデプロイすることで、安全なサプライチェーンを構築したいと考えています。これを実現する方法を説明してください。
正解:
解決策（ステップバイステップ）：
1. 中央集権型画像レジストリ：
- すべてのコンテナイメージの唯一の信頼できる情報源として機能する、集中型イメージレジストリを構築する -
- 人気のある選択肢には以下のようなものがあります。
- Docker Hub：個人プロジェクトやオープンソースプロジェクト向けの無料プランを提供する公開レジストリ。
- Harbor：脆弱性スキャンやアクセス制御などの機能を備えたオープンソースのレジストリ。
- Google Container Registry (GCR): Google Cloud Platformと統合されたレジストリで、イメージ署名やストレージ管理などの機能を提供します。
2. クラスタアクセスの設定：
- すべてのKubernetesクラスターがこの集中型イメージレジストリにアクセスできることを確認してください。
プライベートレジストリの場合は、認証および認可メカニズムを設定して、どのクラスターがどのイメージにアクセスできるかを制御します。
3. イメージプルポリシーを実装する：
- 各クラスターで、中央レジストリからのイメージを使用するデプロイメントに対して「imagePullPolicy」を「Always」に設定します。これにより、すべてのポッドがイメージをプルすることが保証されます。
レジストリから直接イメージを取得し、キャッシュされたイメージへの依存を回避します。
- 例（プライベートレジストリの「nginx:latest」を使用したデプロイメントの場合）：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: my-private-registry.example.com/nginx:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

4. イメージ署名を有効にする (オプション): - セキュリティをさらに強化するためにイメージ署名を実装します - 信頼できるキーを使用して、中央レジストリ内のイメージに署名します - Kubernetes クラスターを構成して、信頼できるキーで署名されたイメージのみがデプロイできるようにします。5. 監視と監査: - イメージのプル、デプロイ、および潜在的な脆弱性を追跡するために、堅牢な監視と監査を実装します。6. ソフトウェアサプライチェーン管理 (SSCM) ツールを検討する: - 専用の SSCM ツールを使用して、脆弱性スキャン、ポリシーの適用、アクセス制御など、イメージのライフサイクル全体を管理します。JFrog Xray や Aqua Security などのツールは、このプロセスを自動化するのに役立ちます。

質問: **11**
重要なアプリケーションを実行しているKubernetesクラスタがあり、そのアプリケーションはボリュームとしてマウントされた機密性の高い設定ファイルを使用しています。この設定ファイルには、承認されたユーザーのみがアクセスできるようにしたいと考えています。Kubernetes R8ACを使用して、必要なロール、バインディング、サービスアカウントを含め、この設定ファイルへのアクセスを制限するにはどうすればよいでしょうか？
正解:
解決策 （ステップバイステップ） :
1. サービスアカウントを作成する
構成ファイルへのアクセスが必要なアプリケーション用に、サービスアカウントを作成します。
- 例 :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sensitive-app-sa
```

2. ロールの作成: - 設定ファイルへの読み取り専用アクセスを許可するロールを作成します。- 例:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: sensitive-app-role
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get"]
```

3. 役割をサービス アカウントにバインドします。- 役割をサービス アカウントにバインドしてアクセスを許可します。- 例:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sensitive-app-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: sensitive-app-role
subjects:
- kind: ServiceAccount
  name: sensitive-app-sa
  namespace: default
```

4. デプロイメントの更新: - デプロイメント YAML を更新して、サービス アカウントを使用し、ボリューム マウントを指定します。- 例:

```
ind: Deployment
etadata:
 name: sensitive-app
pec:
 replicas:
 selector:
   matchLabels:
     app: sensitive-app
 template:
   metadata:
     labels:
       app: sensitive-app
   spec:
     serviceAccountName: sensitive-app-sa
     containers:
     - name: sensitive-app
       image: your-image:latest
       volumeMounts:
       - name: sensitive-config
         mountPath: /etc/config
     volumes:
     - name: sensitive-config
       configMap:
```

5. 変更を適用する: - 'kubectl apply -f' コマンドを使用して、サービス アカウント、ロール、ロール バインディング、および更新されたデプロイメントを適用します。

質問**: 12**
シミュレーション
このタスクは、以下のクラスター/ノードで完了する必要があります。
クラスター: トレース
マスターノード: master
ワーカーノード: worker1
以下のコマンドを使用して、クラスタ/構成コンテキストを切り替えることができます。
[desk@cli] $ kubectl config use-context trace
前提条件 :SysdigまたはFalcoのドキュメントを使用できます。
タスク：
Pod tomcatに属する単一のコンテナ内で、プロセスが頻繁に異常な動作を起こしたり、奇妙な処理を実行したりするなどの異常を検出するために、検出ツールを使用してください。
利用できるツールは2つあります。

1. ハヤブサ
2. シスディック
ツールはworker1ノードにのみプリインストールされています。
新たに生成および実行されるプロセスを検出するフィルターを使用して、コンテナの動作を少なくとも40秒間分析します。
インシデントファイルを/home/cert_masters/reportに以下の形式で保存してください。
[タイムスタンプ]、[UID]、[プロセス名]
注 :インシデントファイルは必ずクラスターのワーカーノードに保存し、マスターノードに移動しないでください。
正解:
下記の解説をご覧ください
Explanation:
$vim /etc/falco/falco_rules.local.yaml
- ルール: コンテナのドリフトが検出されました (open+create)
desc: open+create によりコンテナ内に新しい実行ファイルが作成されました
条件: >
evt.type は (open,openat,creat) で、
evt.is_open_exec=true および
コンテナと
runc_writing_exec_fifo ではなく、
runc_writing_var_lib_docker ではなく、
user_known_container_drift_activities ではなく、
evt.rawres>=0
出力: >
%evt.time,%user.uid,%proc.name # これを追加する/Falcoのドキュメントを参照
優先度: エラー
$kill -1 <ファルコのPID>
Explanation:
[desk@cli] $ ssh node01
[node01@cli] $ vim /etc/falco/falco_rules.yaml
「Container Drift Detected」を検索し、falco_rules.local.yamlに貼り付けてください。
[node01@cli] $ vim /etc/falco/falco_rules.local.yaml
- ルール: コンテナのドリフトが検出されました (open+create)
desc: open+create によりコンテナ内に新しい実行ファイルが作成されました
条件: >
evt.type は (open,openat,creat) で、
evt.is_open_exec=true および
コンテナと
runc_writing_exec_fifo ではなく、
runc_writing_var_lib_docker ではなく、
user_known_container_drift_activities ではなく、
evt.rawres>=0

出力: >
%evt.time,%user.uid,%proc.name # これを追加する/Falcoのドキュメントを参照
優先度: エラー
[node01@cli] $ vim /etc/falco/falco.yaml

質問: **13**
準備済みのランタイムハンドラ runsc を使用して、untrusted という名前の RuntimeClass を作成します。
gVisorランタイムクラスで実行するために、名前空間defaultにalpine:3.13.2イメージのPodを作成します。
正解:
確認 :ポッドを実行してdmesgを実行すると、次のような出力が表示されます。

質問: **14**
名前空間の制限内で、ボリュームタイプとしてpersistentvolumeclaimのみを許可するPSPを作成します。
persistentvolumeclaim 以外のボリュームを持つ Pod がマウントされるのを防止する、prevent-volume-policy という名前の新しい PodSecurityPolicy を作成します。
制限付き名前空間に、psp-sa という名前の新しいサービスアカウントを作成します。
新しく作成したPodセキュリティポリシーprevent-volume-policyを使用する、psp-roleという名前の新しいClusterRoleを作成します。
作成した ClusterRole psp-role を作成した SA psp-sa にバインドする、psp-role-binding という名前の新しい ClusterRoleBinding を作成します。
ヒント：
また、Pod マニフェストでシークレットをマウントしようとして、設定が正しく機能しているかどうかを確認してください。マウントは失敗するはずです。
PODマニフェスト：
APIバージョン: v1
種類: ポッド
メタデータ:
名前：

仕様:
コンテナ:
- 名前 :
画像 :
ボリュームマウント:
- 名前 :
マウントパス:
ボリューム:
- 名前 :
秘密 :
シークレットネーム:
正解:
APIバージョン: policy/v1beta1
種類: サブセキュリティポリシー
メタデータ:
名前: 制限付き
注釈:
seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime/default' apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default' seccomp.security.alpha.kubernetes.io/defaultProfileName: 'runtime/default'
apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default' spec:
特権: false
root権限への昇格を防ぐために必要です。
allowPrivilegeEscalation: false
これは非ルート + 権限昇格の禁止と重複しています。
しかし、我々はそれを多層防御のために提供することができる。
必要なドロップ機能:
- 全て
コアボリュームタイプを許可します。
ボリューム:
- 'configMap'
- 'emptyDir'
- 予測された」
- '秘密'
- 'downwardAPI'
クラスター管理者が設定した persistentVolumes は安全に使用できるものと仮定します。
- 'persistentVolumeClaim'
hostNetwork: false
hostIPC: false
hostPID: false
runAsUser:
コンテナをroot権限なしで実行するように要求します。

ルール: 'MustRunAsNonRoot'
seLinux:
このポリシーは、ノードが SELinux ではなく AppArmor を使用していることを前提としています。
ルール: 'RunAsAny'
補足グループ:
ルール: 'MustRunAs'
範囲:
ルートグループの追加を禁止します。
- 最小値: 1
最大: 65535
fsGroup:
ルール: 'MustRunAs'
範囲:
ルートグループの追加を禁止します。
- 最小値: 1
最大: 65535
readOnlyRootFilesystem: false

質問**: 15**
Kubernetes クラスター上で重要なアプリケーションが稼働しています。このアプリケーションは、アプリケーションを実行しているポッドのみがアクセスできる永続ボリュームに保存された機密データを使用しています。いずれかのポッドが侵害された場合でも、攻撃者がこの機密データにアクセスできないようにしたいと考えています。どのようなセキュリティのベストプラクティスを実装しますか？
正解**:**
解決策（ステップバイステップ）：
1. ボリューム暗号化:
- BitLockerやLUKSなどのツールを使用して、保存されている永続ボリュームを暗号化する。
暗号化キーは安全に保管し、できればKubernetesクラスターの外部に保管してください。
・鍵管理システムを使用して、暗号化キーを安全に管理する。
各ボリュームごとに異なる暗号化キーを使用してください。
2. アクセス制御：
- 永続ボリュームへのアクセスを、重要なアプリケーションを実行しているポッドのみに制限します。
- Kubernetes RBAC を利用して、アプリケーション Pod の実行を担当するサービス アカウントに最小限の権限を付与します。
サービスアカウントに広範な権限を付与することは避け、必要なリソースのみにアクセスを制限してください。
3. ポッドセキュリティポリシー（PSP）：
- PSPを実装して、ポッドが利用できる機能とリソースを制限する。
- ポッドが機密性の高いボリュームにアクセスしたり、特権的な権限を持つことを制限します。
- 明示的に承認されていないボリュームをポッドがマウントすることを防止するポリシーを適用する。
- 侵害されたポッドによる潜在的な影響を制限するために、厳格なPSPルールを定義する。
4. ネットワークセグメンテーション：
- Kubernetesクラスタを他のネットワークから隔離し、送受信トラフィックを許可された送信元と宛先のみに制限します。
・ファイアウォールルールを実装し、クラスターへの不正アクセスを防止する。

ネットワークセグメンテーションを利用して、攻撃者がネットワーク接続を介して永続ボリュームにアクセスすることを防止します。
5. ランタイムセキュリティ：
- FalcoやKubernetes Admission Controllerなどのランタイムセキュリティツールを使用して、Pod内の悪意のあるアクティビティを監視および防止します。
- ランタイムセキュリティツールを設定し、永続ボリューム内の機密データへのアクセス試行を検出してブロックする。
- Kubernetes環境内に侵入検知・防御システム（IDS/IPS）を実装する
6. 定期的なセキュリティ監査：
セキュリティ対策が効果的であることを確認するため、定期的にセキュリティ監査を実施する。
暗号化、アクセス制御、および実行時セキュリティ対策の有効性を評価する。
セキュリティ上の脆弱性を速やかに特定し、修復する。
7. 不変インフラストラクチャ：
-不変インフラストラクチャの原則を用いて、攻撃対象領域を最小限に抑え、攻撃者が永続ボリュームを変更することを防ぎます。
- アプリケーションコードと構成を不変コンテナとしてデプロイする -
永続ボリュームに直接変更を加えることは避けてください。

```
# Example Kubernetes RBAC configuration for restricted access to the persistent volume
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: my-app-volume-reader
  namespace: default
rules:
- apiGroups: ["persistentvolume"]
  resources: ["persistentvolumes"]
  verbs: ["get", "list", "watch"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-app-volume-reader-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: my-app-volume-reader
subjects:
- kind: ServiceAccount
  name: my-app-sa
  namespace: default
```


質問: **16**
名前空間 testing で実行されている nginx-test ポッドに制限するための restrict-np という名前のネットワーク ポリシーを作成します。
Pod nginx-test への接続を許可する Pod は以下のとおりです。
1. 名前空間 default 内のポッド
2. ラベル version:v1 を持つ、任意の名前空間内のポッド。
ネットワークポリシーを必ず適用してください。
正解:
これについてのご意見をお聞かせください。

質問: **17**
johnという名前のユーザーを作成し、CSRリクエストを作成し、承認後にユーザーの証明書を取得します。
名前空間john内のシークレットやポッドを一覧表示するロール名john-roleを作成します。
最後に、john-role-binding という名前の RoleBinding を作成し、新しく作成したロール john-role を名前空間 john 内のユーザー john に割り当てます。確認するには、kubectl auth CLI コマンドを使用して権限を確認します。
正解:
kubectlを使用してCSRを作成し、承認します。
CSRのリストを取得する：
kubectl get csr
CSRを承認する：
kubectl certificate approve myuser
証明書を取得する
CSRから証明書を取得します。
kubectl get csr/myuser -o yaml
以下は、JohnにNEW_CRDリソースを作成する権限を与えるためのロールとロールバインディングです。
kubectl apply -f roleBindingJohn.yaml --as=john
rolebinding.rbac.authorization.k8s.io/john_external-rosource-rb が作成されました。種類: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 メタデータ:
名前: john_crd
名前空間: development-john
主題:
- 種類: ユーザー
名前 :ジョン
apiグループ: rbac.authorization.k8s.io
役割参照:
種類: ClusterRole
名前: crd-creation
種類: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
メタデータ:
名前: crd-creation
ルール:
- apiGroups: ["kubernetes-client.io/v1"]
リソース: ["NEW_CRD"]
動詞: ["作成する、一覧表示する、取得する"]

質問: **18**
クラスターで監査ログを有効にするには、ログバックエンドを有効にし、

1. ログは /var/log/kubernetes-logs.txt に保存されます。
2. ログファイルは12日間保持されます。
3. 最大で8個の古い監査ログファイルが保持されます。
4. 回転前の最大サイズを200MBに設定する
基本ポリシーを編集して拡張し、以下の内容をログに記録するようにします。
1. RequestResponse でのネームスペースの変更
2. 名前空間 kube-system 内のシークレットの変更に関するリクエストボディをログに記録します。
3. コアおよび拡張機能内のその他のすべてのリソースをリクエストレベルでログに記録します。
4. メタデータレベルで「pods/portforward」、「services/proxy」をログに記録します。
5. ステージ RequestReceived を省略する
メタデータレベルでのその他のすべてのリクエスト
正解:
Kubernetesの監査機能は、クラスタに関するセキュリティ関連の時系列レコードを提供します。Kube-apiserverが監査を実行します。各リクエストは実行の各段階でイベントを生成し、そのイベントは特定のポリシーに従って前処理され、バックエンドに書き込まれます。ポリシーによって記録される内容が決定され、バックエンドはレコードを永続化します。
CIS（Center for Internet Security）のKubernetesベンチマークコントロールへの準拠の一環として、監査ログを設定することをお勧めします。
監査ログは、cluster.yml ファイルに以下の設定を追加することで、デフォルトで有効にできます。
サービス:
kube-api:
監査ログ:
有効: true
監査ログが有効になっている場合、デフォルト値は /etc/kubernetes/audit-policy.yaml で確認できます。ログバックエンドは、監査イベントを JSONlines 形式でファイルに書き込みます。ログ監査バックエンドは、次の kube-apiserver フラグを使用して設定できます。
--audit-log-path は、ログバックエンドが監査イベントを書き込むために使用するログファイルパスを指定します。このフラグを指定しない場合、ログバックエンドは無効になります。- は標準出力を意味します。
--audit-log-maxage は、古い監査ログファイルを保持する最大日数を定義します。
--audit-log-maxbackup は、保持する監査ログファイルの最大数を定義します。
--audit-log-maxsize は、監査ログファイルがローテーションされる前の最大サイズをメガバイト単位で定義します。クラスターのコントロールプレーンが kube-apiserver を Pod として実行している場合は、監査レコードが永続化されるように、hostPath をポリシーファイルとログファイルの場所にマウントすることを忘れないでください。例:
--audit-policy-file=/etc/kubernetes/audit-policy.yaml \
--audit-log-path=/var/log/audit.log

質問: 19
特権コンテナと特定の機能の使用を制限するポッドセキュリティポリシー（PSP）を使用するようにKubernetesクラスタを構成する必要があります。「production」名前空間内の特定のポッドのみが「NET_ADMIN」機能で実行できるようにしたいと考えています。
正解:
解決策（ステップバイステップ）：
1. PSPIを作成する
- 'production' 名前空間のポッドの機能を除き、特権コンテナと機能の使用を制限する PSP を定義します。

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted-psp
spec:
  privileged: false
  fsGroup:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  capabilities:
    drop: ["ALL"]
    add: ["NET_ADMIN"]
  volumes:
    - 'configMap'
    - 'secret'
    - 'persistentVolumeClaim'
    - 'hostPath'
    - 'emptyDir'
  hostNetwork: false
  hostPID: false
  hostIPC: false
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
  seLinux:
    rule: 'RunAsAny'
  runAsUser:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  # Allowed HostPorts (optional)
  # hostPorts:
  #   - min: 1024
  #     max: 65535
  # Allow Specific Namespaces (optional)
  seLinux:
    rule: 'RunAsAny'
  runAsUser:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  # Allowed HostPorts (optional)
  # hostPorts:
  #   - min: 1024
  #     max: 65535
  # Allow Specific Namespaces (optional)
  allowedNamespaces:
    - production
```

2. PSPバインディングを作成する :- PSPを「production」名前空間にバインドする -

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted-psp
spec:
  privileged: false
  fsGroup:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  capabilities:
    drop: ["ALL"]
    add: ["NET_ADMIN"]
  volumes:
    - 'configMap'
    - 'secret'
    - 'persistentVolumeClaim'
    - 'hostPath'
    - 'emptyDir'
  hostNetwork: false
  hostPID: false
  hostIPC: false
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
  seLinux:
    rule: 'RunAsAny'
  runAsUser:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  # Allowed HostPorts (optional)
  # hostPorts:
  #   - min: 1024
  #     max: 65535
  # Allow Specific Namespaces (optional)
  allowedNamespaces:
    - production
```


3. Pod の作成: - 「production」名前空間に Pod を作成し、「NET_ADMIN」権限を持つ「securitycontext」を指定します。

4. YAML ファイルを適用します: - 'kubectl apply -f' を使用して作成した YAML ファイルを適用します。5. 権限を確認します: - 'NET_ADMIN' 権限を持つ他の名前空間に Pod を作成してみてください。拒否されるはずです。

質問: **20**
シミュレーション
testing ネームスペース内に Nginx-pod という名前の Pod を作成し、Nginx-pod 用の nginx-svc という名前のサービスを作成します。任意のイングレスを使用し、tls でイングレスを実行し、ポートをセキュアにします。
正解:
ご意見をお聞かせください

質問: **21**
シミュレーション

コンテクスト
PodSecurityPolicyは、特定の名前空間における特権Podの作成を防止するものとする。
タスク
特権Podの作成を防止する、prevent-psp-policyという名前の新しいPodSecurityPolicyを作成します。
新しく作成した PodSecurityPolicy prevent-psp-policy を使用する、restrict-access-role という名前の新しい ClusterRole を作成します。
既存のネームスペース 「staging」内に、psp-restrict-sa という名前の新しいサービスアカウントを作成します。
最後に、新しく作成した ClusterRole restrict-access-role を新しく作成した ServiceAccount psp-restrict-sa にバインドする、restrict-access-bind という名前の新しい ClusterRoleBinding を作成します。

正解:

下記の解説をご覧ください

Explanation:


apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
name: "prevent-psp-policy"
spec:
privileged: false
seLinux:
rule: RunAsAny
runAsUser:
rule: RunAsAny
supplementalGroups:
rule: RunAsAny
fsGroup:
rule: RunAsAny

```
candidate@cli:~$ vim /home/candidate/KSMV00102/pod-security-policy.yaml
candidate@cli:~$ cat /home/candidate/KSMV00102/pod-security-policy.yaml
---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: "prevent-psp-policy"
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
candidate@cli:~$ kubectl create -f /home/candidate/KSMV00102/pod-security-policy.yaml
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy/prevent-psp-policy created
candidate@cli:~$ cat /home/candidate/KSMV00102/cluster-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ""
rules:
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role.yaml
```


```
candidate@cli:~$ kubectl create clusterrole restrict-access-role --verb=use --resource=psp --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: restrict-access-role
rules:
- apiGroups:
  - policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: "restrict-access-role"
rules:
- apiGroups:
  - policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

```
candidate@cli:~$ kubectl create clusterrole restrict-access-role --verb=use --resource=psp -
-dry-run=client --resource-name=prevent-psp-policy -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: restrict-access-role
rules:
- apiGroups:
  - policy
  resourceNames:
  - prevent-psp-policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: "restrict-access-role"
rules:
- apiGroups:
  - policy
  resourceNames:
  - prevent-psp-policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

```
candidate@cli:~$ kubectl create -f /home/candidate/KSMV00102/cluster-role.yaml
clusterrole.rbac.authorization.k8s.io/restrict-access-role created
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ cat /home/candidate/KSMV00102/service-account.yaml
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: "psp-restrict-sa"
  namespace: "staging"
```

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ""
  namespace: ""
candidate@cli:~$ vim /home/candidate/KSMV00102/service-account.yaml
candidate@cli:~$ cat /home/candidate/KSMV00102/service-account.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: "psp-restrict-sa"
  namespace: "staging"
candidate@cli:~$ kubectl get sa -n staging
NAME      SECRETS   AGE
default   1         6h6m
candidate@cli:~$ kubectl create -f /home/candidate/KSMV00102/service-account.yaml
serviceaccount/psp-restrict-sa created
candidate@cli:~$ kubectl get sa -n staging
NAME              SECRETS   AGE
default           1         6h6m
psp-restrict-sa   1         2s
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ kubectl create clusterrolebinding restrict-access-bind --clusterrole=restri
ct-access-role --serviceaccount=staging:psp-restrict-sa --dry-run -o yaml
W0520 14:41:23.502004   47627 helpers.go:598] --dry-run is deprecated and can be replaced wi
th --dry-run=client.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: restrict-access-bind
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: restrict-access-role
subjects:
- kind: ServiceAccount
  name: psp-restrict-sa
  namespace: staging
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role
cluster-role-binding.yaml  cluster-role.yaml
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role
cluster-role-binding.yaml  cluster-role.yaml
candidate@cli:~$ vim /home/candidate/KSMV00102/cluster-role-binding.yaml
```

質問: 22

あなたは、パブリックレジストリのコンテナイメージを使用する「my-app」という名前のデプロイメントを含むKubernetesクラスタを実行しています。最近のデプロイメントの更新によって、コンテナの1つに脆弱性が生じた可能性があると疑っています。Trivyなどのコンテナイメージスキャンツールを使用して、脆弱性を特定し、対処する方法を説明してください。

正解:

解決策（ステップバイステップ）：

1. Trivyをインストールして設定する：

- TrivyをシステムまたはKubernetesクラスター内にインストールしてください。Trivyは、コンテナイメージ、ファイルシステム、アプリケーションをスキャンできる多機能な脆弱性スキャナーです。

2. コンテナ画像をスキャンする：

- 「my-app」デプロイメントで使用されるコンテナイメージに対してTrivyを実行します。
バッシュ
トリビーイメージ例/nginx:latest
3. スキャン結果を分析する：
Trivyスキャンレポートを確認してください。このレポートには、コンテナイメージで検出された脆弱性がすべて記載されています。脆弱性の深刻度、説明、潜在的な影響などの情報も提供されます。
4. 脆弱性に対処する：
脆弱性が発見された場合は、リスクを軽減するために適切な措置を講じてください。これには以下が含まれます。
- コンテナイメージの更新: 脆弱性が修正された新しいバージョンのコンテナイメージが利用可能な場合は、更新されたイメージを使用するようにデプロイメントを更新します。
- セキュリティ対策の実施 :ネットワークアクセスの制限、コンテナ権限の制限、セキュリティ強化ツールの使用など、コンテナ内で追加のセキュリティ制御を実装することを検討してください。
- リスクの受容：脆弱性のリスクが低いと判断され、更新や軽減が実行不可能な場合は、リスクを受容して脆弱性を綿密に監視することを選択できます。
5. CI/CDパイプラインとの統合：
TrivyをCI/CDパイプラインに統合することで、コンテナイメージをKubernetesクラスターにデプロイする前に自動的にスキャンできます。これにより、脆弱性を早期に発見し、本番環境への侵入を防ぐことができます。

質問: **23**
シミュレーション
コンテクスト
kubeadmでプロビジョニングされたクラスターに対して、監査機能を実装する必要があります。
タスク
まず、クラスターのAPIサーバーを以下のように再構成します。
基本的な監査方針は、
/etc/kubernetes/logpolicy/audit-policy.yaml が使用されます。
. ログは /var/log/kubernetes/audit-logs.txt に保存されます。
最大2つのログが10日間保持されます。
このクラスターは、コンテナのランタイムとしてDocker Engineを使用します。必要に応じて、dockerコマンドを使用して実行中のコンテナのトラブルシューティングを行ってください。
基本ポリシーでは、ログに記録しない内容のみが指定されています。
次に、基本ポリシーを編集して拡張し、以下の内容をログに記録します。
リクエストレスポンスレベルでのネームスペース間の相互作用
. 名前空間 webapps 内のデプロイメントのインタラクションのリクエストボディ
メタデータレベルでのすべてのネームスペースにおけるConfigMapとSecretの相互作用
メタデータレベルでのその他のすべてのリクエスト
APIサーバーが拡張ポリシーを使用していることを確認してください。
そうしない場合、減点される可能性があります。
正解:
完全な解決策については、以下の説明をご覧ください。
Explanation:
1) 正しいホストに接続します
ssh cks000028
sudo -i
（試験でホスト名が異なる場合は、問題バナーに表示されているホスト名を使用してください。）
2) APIサーバーの静的ポッドマニフェストを編集する

APIサーバーはkubeadm内の静的Podです。
/etc/kubernetes/manifests/kube-apiserver.yaml 内
3) APIサーバーを設定して監査を有効にする
コマンドセクション内に、以下のフラグがすべて存在することを確認してください。
（不足している場合は追加し、存在する場合は修正してください。）
3.1 指定された監査ポリシーファイルを使用する
- --audit-policy-file=/etc/kubernetes/logpolicy/audit-policy.yaml
3.2 監査ログを所定の場所に保存する
- --audit-log-path=/var/log/kubernetes/audit-logs.txt
3.3 最大2つのログファイルを保持する
--audit-log-maxbackup=2
3.4 ログを10日間保持する
--audit-log-maxage=10
✅ 例（ファイルにはもっと多くのフラグが含まれている可能性がありますが、問題ありません）：
- 指示：
- be-apiserver
- --audit-policy-file=/etc/kubernetes/logpolicy/audit-policy.yaml
- --audit-log-path=/var/log/kubernetes/audit-logs.txt
--audit-log-maxbackup=2
--audit-log-maxage=10
保存して終了：
:wq
APIサーバーは自動的に再起動します（静的ポッド）。
オプションのクイックチェック：
ドッカー ps | grep kube-apiserver
4) 監査ポリシーを編集および拡張する
指定された基本ポリシーを開きます。
/etc/kubernetes/logpolicy/audit-policy.yaml 内
ファイルには既に、ログに記録してはいけない内容に関するルールが含まれています。
既存のルールは削除せず、その下にルールを追加してください。
5) 必要な監査ルールを追加する（正確な順序で）
以下の規則をこの順序で追加してください（監査方針においては順序が重要です）。
5.1 RequestResponse でのネームスペースのやり取りをログに記録する
- レベル: リクエストレスポンス
リソース：
- グループ："
リソース: ["名前空間"]
5.2 名前空間webappsにデプロイメント要求本文をログ記録する
- レベル: リクエストレスポンス
名前空間: ["webapps"]

リソース：
- グループ: "アプリ"
リソース: ["デプロイメント"]
5.3 メタデータにおけるConfigMapとSecretのやり取りのログ記録（すべての名前空間）
- レベル: メタデータ
リソース：
- グループ："
リソース: ["configmaps", "secrets"]
5.4 その他のすべてのリクエストをメタデータにログ記録する
これは最後でなければならない
- レベル: メタデータ
5.5 最終的な audit-policy.yaml は次のように終了する必要があります。
#（上記に既存の「ログを記録しない」ルールがあります）
- レベル: リクエストレスポンス
リソース：
- グループ："
リソース: ["名前空間"]
- レベル: リクエストレスポンス
名前空間: ["webapps"]
リソース：
- グループ: "アプリ"
リソース: ["デプロイメント"]
- レベル: メタデータ
リソース：
- グループ："
リソース: ["configmaps", "secrets"]
- レベル: メタデータ
保存して終了：
:wq
6) APIサーバーがEXTENDEDポリシーを使用していることを確認してください
マニフェストをタップして再読み込みを保証してください。
touch /etc/kubernetes/manifests/kube-apiserver.yaml
数秒お待ちください。
7) 監査が正常に機能していることを確認する
7.1 監査ログファイルが存在するか確認する
ls -l /var/log/kubernetes/audit-logs.txt
7.2 テストアクティビティを生成する
kubectl get namespaces
kubectl get configmaps -A
7.3 ログが書き込まれていることを確認する

tail -n 20 /var/log/kubernetes/audit-logs.txt
監査記録が表示されるはずです。

質問: **24**
あなたはKubernetesクラスタにデプロイする新しいマイクロサービスを開発しています。マイクロサービスのKubernetes YAMLマニフェストがセキュリティのベストプラクティスに準拠し、クラスタ構成と互換性があることを確認する必要があります。デプロイ前にKubeLinterを使用してYAMLマニフェストを検証するソリューションを実装してください。
正解:
解決策（手順別）：
1. KubeLinter のインストール: 公式 GitHub リポジトリから 'kubevar バイナリ' をダウンロードしてインストールします。
2. KubeLinter 設定ファイルを作成します (オプション): プロジェクトのルート ディレクトリに .kubeval.yaml ファイルを定義して、カスタム ルールやチェックを指定します。
3. KubeLinter を使用して YAML マニフェストを検証します。「kubeval」コマンドを使用して、YAML マニフェストを Kubernetes スキーマおよびカスタムルールに対して検証します。
バッシュ
kubeval deployment.yaml service.yaml
4. KubeLinterをCI/CDパイプラインに統合する :パイプラインに、YAMLマニフェストに対してKubeLinterを実行するステップを追加します。このステップは、マニフェストがクラスターにデプロイされる前に実行する必要があります。

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

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


5. KubeLinterによって報告された問題に対処します。KubeLinterの出力を分析し、特定された問題に対処するためにYAMLマニフェストに必要な変更を加えます。

質問: **25**
シミュレーション
サービスはシステム内のポート 389 で実行されています。プロセスのプロセス ID を見つけて、開いているすべてのファイルの名前を /candidate/KH77539/files.txt に保存し、バイナリも削除します。
正解:
以下の説明をご覧ください。
root# netstat -ltnup
アクティブなインターネット接続 サーバーのみ）
プロトコル 受信キュー 送信キュー ローカルアドレス 外部アドレス 状態 PID/プログラム名 tcp 0 0 127.0.0.1:17600 0.0.0.0:* LISTEN 1293/dropbox tcp 0 0 127.0.0.1:17603 0.0.0.0:* LISTEN 1293/dropbox tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 575/sshd tcp 0 0 127.0.0.1:9393 0.0.0.0:* LISTEN 900/perl tcp 0 0 :::80 :::* LISTEN 9583/docker-proxy tcp 0 0 :::443 :::* LISTEN 9571/docker-proxy udp 0 0 0.0.0.0:68 0.0.0.0:* 8822/dhcpcd
...
root# netstat -ltnup | grep ':22'
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 575/sshd
ssコマンドはnetstatコマンドの後継コマンドです。
それでは、ssコマンドを使って、どのプロセスがポート22でリッスンしているかを確認する方法を見ていきましょう。
root# ss -ltnup 'sport = :22'
Netid 状態 受信キュー 送信キュー ローカルアドレス:ポート ピアアドレス:ポート
tcp LISTEN 0 128 0.0.0.0:22 0.0.0.0:* users:("sshd",pid=575,fd=3))

質問: 26
クラスターで監査ログを有効にするには、ログバックエンドを有効にし、
1. ログは /var/log/kubernetes/kubernetes-logs.txt に保存されます。
2. ログファイルは5日間保持されます。
3. 最大で10個の古い監査ログファイルが保持されます。
基本ポリシーを編集して拡張し、以下の内容をログに記録するようにします。
1. RequestResponseにおけるCronjobsの変更
2. kube-system 名前空間におけるデプロイメントの変更に関するリクエスト本文をログに記録します。
3. コアおよび拡張機能内のその他のすべてのリソースをリクエストレベルでログに記録します。
4. エンドポイントで「system:kube-proxy」による監視リクエストをログに記録しない、または
正解:

```
candidate@cli:~$ kubectl config use-context KSRS00602
Switched to context "KSRS00602".
candidate@cli:~$ ssh ksrs00602-master
Warning: Permanently added '10.240.86.243' (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@ksrs00602-master:~# cat /etc/kubernetes/logpolicy/sample-policy.yaml
---
apiVersion: audit.k8s.io/v1
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"
  # Edit form here below
root@ksrs00602-master:~# vim /etc/kubernetes/logpolicy/sample-policy.yaml
```

質問: **27**
タスク
名前空間 dev-team で実行されている Pod users-service へのアクセスを制限するために、pod-access という名前の NetworkPolicy を作成します。
Pod users-serviceへの接続を許可するPodは以下のとおりです。

Make sure to apply the NetworkPolicy.

You can find a skeleton manifest file at /home/candidate/KSSH00301/network-policy.yaml

正解:

```
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
```

```
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 15:25:33 +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
```

質問: **28**
EKSを使用してAWS上にKubernetesクラスターをデプロイしています。クラスターを操作する前に、AWS CLIとEKSプラットフォームバイナリの信頼性と整合性を確認してください。
正解:
解決策（手順別）：
I). AWS CLI をインストールします: 公式 Web サイト ([https://aws.amazon.com/clif](https://www.google.com/url?sa=E&source=gmail&q=https://aws.amazon.com/cli/)).
2. AWS CLI のインストールを確認します。`aws -version` コマンドを使用してバージョンを確認し、正しくインストールされていることを確認します。
3. AWS認証情報の設定: `saws configure`コマンドを使用してAWS認証情報を設定します。
4. EKS APIサーバーエンドポイントを確認します。「aws eks describe-cluster」コマンドを使用して、EKSクラスターのAPIサーバーエンドポイントを取得します。
エンドポイントが、お住まいの地域で想定される形式およびドメイン名と一致していることを確認してください。
バッシュ
aws ekS describe-cluster -name my-cluster -query "cluster.endpoint" -output text
5. EKS APIサーバー証明書の真正性を確認します。「openssl s_client」コマンドを使用してEKS APIサーバー証明書を取得し、証明書チェーンと発行者を確認します。
バッシュ
openssl s_client -connect :443 /dev/null | openssl x509 -in - -text -noout
6. (オプション) AWS CLI を使用して EKS コンポーネントをさらに検証します。AWS CLI を使用して、コントロール プレーン、ワーカー ノード、ネットワークなどの他の EKS コンポーネントの状態と構成を確認できます。

質問: **29**

あなたは、機密性の高いアプリケーションを実行しているKubernetesクラスタのセキュリティを確保する任務を負っています。このクラスタ内のユーザーアカウントとサービスアカウントの両方に対して、「最小権限」の原則をどのように実装するかを説明してください。
正解:
解決策（ステップバイステップ）：
1. ユーザーの役割と権限:
- 異なるユーザーグループに対し、それぞれの責任に基づいて最小限の権限を持つ特定の役割を定義します。
例えば、開発者はアプリケーションのデプロイ権限を持ち、運用チームのメンバーはリソースの管理権限を持つといった具合です。
- Kubemetes の RBAC (ロールベースアクセス制御) を使用してロールを定義し、ユーザーに割り当てます。
2. サービスアカウントの権限:
・クラスター内の各アプリケーションまたはサービスごとに、個別のサービスアカウントを作成します。
サービスアカウントには、それぞれの特定のタスクを実行するために必要な権限のみを付与してください。
・広範な権限を持つデフォルトのサービスアカウントの使用は避けてください。
サービスアカウントに対して最小限の権限を定義することで、「最小権限の原則」を適用する。
3. ポッドセキュリティポリシー（PSP）：
- PSPを実装してPodにセキュリティ制約を適用し、Podがアクセスできるリソースを制限する。
- PSPを定義して、特定のコンテナイメージのみを許可したり、特権コンテナを無効にしたり、リソース要求を制限したり、その他のセキュリティ制御を適用したりします。
- Kubernetes 1.25以降では、PSPの代替としてPod Security Admission（PSA）の使用を検討してください。
4. ネットワークポリシー：
- ポッドとサービス間のネットワーク通信を制御するためのネットワークポリシーを実装する。
- ポッド間で必要なトラフィックのみを許可するルールを定義し、不要な接続や不正な接続を制限します。
5. 秘密管理
パスワードやAPIキーなどの機密情報を保存するために、Kubernetes Secretsを利用します。
最小権限の原則に基づいて、機密情報へのアクセスを制限する。
機密情報をデプロイメントYAMLファイルに直接保存することは避けてください。

質問: 30

サードパーティ製の Helm チャートを使用して、アプリケーションを Kubernetes クラスターにデプロイしています。Helm チャートがセキュリティのベストプラクティスに準拠し、クラスターに脆弱性をもたらさないことを確認する必要があります。デプロイ前に KubeLinter を使用して Helm チャートを静的に分析するソリューションを実装してください。

正解:

解決策（手順別）：

1. KubeLinter をインストールします: 公式 GitHub リポジトリから 'kubevar バイナリをダウンロードしてインストールします。
2. Helm チャートをレンダリングします。「helm template」コマンドを使用して、Helm チャートを Kubernetes YAML マニフェストにレンダリングします。

バッシュ

```
helm template my-chart -f values.yaml > rendered-templates.yaml
```

3. KubeLinter を使用してレンダリングされた YAML マニフェストを検証します。「kubevar コマンド」を使用して、レンダリングされた YAML マニフェストを検証します。
Kubernetesスキーマとカスタムルール。
バッシュ
kubeval rendered-templates.yaml
4. KubeLinterをCI/CDパイプラインに統合する :パイプラインに、Helmチャートをレンダリングし、レンダリングされたチャートに対してKubeLinterを実行するステップを追加します。
YAMLマニフェスト。この手順は、チャートをデプロイする前に実行する必要があります。

質問: **31**
機密データを含むKubernetesクラスターを保護する責任があります
a. クラスター内のポッド間のすべての通信が暗号化されていることを確認する必要があります。ポッド間で相互TLS認証を強制するソリューションを実装してください。
正解:
解決策（手順別）：
1. 各ポッドの証明書とキーを生成します。openssr などのツールを使用して自己署名証明書を生成するか、認証局 (CA) を使用して証明書を発行できます。各ポッドには、独自の秘密鍵と CA によって署名された証明書が必要です。
2. 証明書とキーを保存するためのKubernetes Secretを作成します。このSecretは、各Podにボリュームとしてマウントする必要があります。

```
apiVersion: v1
kind: Secret
metadata:
  name: pod-certs
type: kubernetes.io/tls
data:
  tls.crt:
  tls.key:
```

3. 相互TLSに証明書を使用するようにPodを設定します。通常、これには環境変数またはコマンドライン引数を設定して、証明書ファイルとキーファイルの場所を指定します。
4. Istio や Linkerd などのサービスメッシュをデプロイします。これらのツールは、証明書管理と mTLS の強制のプロセスを自動化できます。自動証明書ローテーション、mTLS 構成を管理するための集中制御プレーン、トラフィック暗号化などの機能を提供します。重要な考慮事項: 証明書管理: 証明書の発行と更新のための安全で自動化されたプロセスを実装します。リソースオーバーヘッド: mTLS はパフォーマンスオーバーヘッドを引き起こす可能性があるため、実装後にアプリケーションのパフォーマンスを監視します。トラブルシューティング: mTLS に関連する接続の問題のトラブルシューティング計画を用意します。

質問: **32**
以下のコマンドを使用して、クラスタ/構成コンテキストを切り替えることができます。
[desk@cli] $ kubectl config use-context stage
コンテクスト：
PodSecurityPolicyは、特定の名前空間における特権Podの作成を防止するものとする。
タスク：
1. 特権Podの作成を防止する、deny-policyという名前の新しいPodSecurityPolcyを作成します。
2. 新しく作成したPodSecurityPolicy deny-policyを使用する、deny-access-roleという名前の新しいClusterRoleを作成します。
3. 既存の名前空間 development に psd-denial-sa という名前の新しいサービスアカウントを作成します。

最後に、restrict-access-bind という名前の新しい ClusterRoleBindind を作成し、新しく作成した ClusterRole deny-access-role を、新しく作成した ServiceAccount psp-denial-sa にバインドします。
正解:
特権コンテナを許可しないための PSP を作成します
apiVersion: rbac.authorization.k8s.io/v1
種類: ClusterRole
メタデータ:
名前: アクセス拒否ロール
ルール:
- apiGroups: ['policy']
リソース: ['podsecuritypolicies']
動詞: ['使う']
リソース名:
- 拒否ポリシー」
k create sa psp-denial-sa -n 開発
apiVersion: rbac.authorization.k8s.io/v1
種類: ClusterRoleBinding
メタデータ:
名前: アクセス制限
役割参照:
種類: ClusterRole
名前: アクセス拒否ロール
apiグループ: rbac.authorization.k8s.io
主題:
- 種類: サービスアカウント
名前: psp-denial-sa
名前空間: 開発
説明
master1 $ vim psp.yaml
APIバージョン: policy/v1beta1
種類: サブセキュリティポリシー
メタデータ:
名前: 拒否ポリシー
仕様:
privileged: false # 特権ポッドを許可しない!
seLinux:
ルール: RunAsAny
補足グループ:
ルール: RunAsAny
runAsUser:
ルール: RunAsAny

fsGroup:
ルール: RunAsAny
ボリューム:
- '*'
master1 $ vim cr1.yaml
apiVersion: rbac.authorization.k8s.io/v1
種類: ClusterRole
メタデータ:
名前: アクセス拒否ロール
ルール:
- apiGroups: ['policy']
リソース: ['podsecuritypolicies']
動詞: ['使う']
リソース名:
- 拒否ポリシー」
master1 $ k create sa psp-denial-sa -ndevelopment
master1 $ vim cb1.yaml
apiVersion: rbac.authorization.k8s.io/v1
種類: ClusterRoleBinding
メタデータ:
名前: アクセス制限
役割参照:
種類: ClusterRole
名前: アクセス拒否ロール
apiグループ: rbac.authorization.k8s.io
主題:
特定のサービスアカウントを承認する:
- 種類: サービスアカウント
名前: psp-denial-sa
名前空間: 開発
master1 $ k apply -f psp.yaml master1 $ k apply -f cr1.yaml master1 $ k apply -f cb1.yaml 参照: https://kubernetes.io/docs/concepts/policy/pod-security-policy/ master1 $ k apply -f cr1.yaml master1 $ k apply -f cb1.yaml master1 $ k apply -f psp.yaml master1 $ k apply -f cr1.yaml master1 $ k apply -f cb1.yaml 参照: https://kubernetes.io/docs/concepts/policy/pod-security-policy/

質問**: 33**
以下のコマンドを使用して、クラスター/構成コンテキストを切り替えることができます。[desk@cli] $ kubectl config use-context test-account タスク: クラスターで監査ログを有効にします。
そのためには、ログバックエンドを有効にし、以下の点を確認してください。
1. ログは /var/log/Kubernetes/logs.txt に保存されます。
2. ログファイルは5日間保持されます
3. 最大で10個の古い監査ログファイルが保持されます。
基本的なポリシーは /etc/Kubernetes/logpolicy/audit-policy.yaml に記述されています。これはログに記録しない内容のみを指定します。注：基本ポリシーはクラスタのマスターノードに配置されます。

基本ポリシーを編集して拡張し、以下の項目をログに記録するようにします。1. RequestResponse レベルでのノードの変更 2. 名前空間フロントエンドでの persistentvolumes のリクエスト本文の変更 3. メタデータ レベルですべての名前空間での ConfigMap および Secret の変更 また、メタデータ レベルで他のすべてのリクエストをログに記録するための包括的なルールを追加します。注: 変更したポリシーを適用することを忘れないでください。
正解:
$ vim /etc/kubernetes/log-policy/audit-policy.yaml
- レベル: リクエストレスポンス
ユーザーグループ: ["system:nodes"]
- レベル: リクエスト
リソース：
- グループ: "" # コア API グループ
リソース: ["persistentvolumes"]
名前空間: ["frontend"]
- レベル: メタデータ
リソース：
- グループ："
リソース: ["configmaps", "secrets"]
- レベル: メタデータ
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml 以下を追加
- --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
説明
[desk@cli] $ ssh master1 [master1@cli] $ vim /etc/kubernetes/log-policy/audit-policy.yaml apiVersion: audit.k8s.io/v1 # これは必須です。
種類: ポリシー
RequestReceived ステージのすべてのリクエストに対して監査イベントを生成しない。
段階を省略:
- 『リクエスト受信」
ルール:
エンドポイントまたはサービスで 『system:kube-proxy」による監視リクエストをログに記録しない
- レベル: なし
ユーザー: ["system:kube-proxy"]
動詞: ["見る"]
リソース：
- グループ: "" # コア API グループ
リソース: ["エンドポイント", "サービス"]
特定の非リソース URL パスへの認証済みリクエストをログに記録しない。
- レベル: なし
ユーザーグループ: ["system:authenticated"]
リソース以外のURL:
- "/api*" # ワイルドカードマッチング。

- "/version"
下記に変更を加えてください
- レベル: リクエストレスポンス
userGroups: ["system:nodes"] # ノード用のブロック
- レベル: リクエスト
リソース：
- グループ: "" # コア API グループ
リソース: ["persistentvolumes"] # 永続ボリューム用のブロック
namespaces: ["frontend"] # フロントエンド ns の永続ボリュームをブロックします
- レベル: メタデータ
リソース：
- グループ: "" # コア API グループ
リソース: ["configmaps", "secrets"] # configmaps と secrets 用のブロック
- レベル: メタデータ # それ以外のすべてをブロックします
[master1@cli] $ vim /etc/kubernetes/manifests/kube-apiserver.yaml
APIバージョン: v1
種類: ポッド
メタデータ:
注釈:
kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 10.0.0.5:6443 ラベル:
コンポーネント: kube-apiserver
階層: 制御プレーン
名前: kube-apiserver
名前空間: kube-system
仕様:
コンテナ:
- 指示：
- be-apiserver
--advertise-address=10.0.0.5
--allow-privileged=true
--authorization-mode=Node,RBAC
- --audit-policy-file=/etc/kubernetes/log-policy/audit-policy.yaml #これを追加
- --audit-log-path=/var/log/kubernetes/logs.txt #これを追加
- --audit-log-maxage=5 #これを追加
- --audit-log-maxbackup=10 #これを追加
...
出力が切り捨てられています
注: ログボリュームとポリシーボリュームは、vim /etc/kubernetes/manifests/kube-apiserver.yaml に既にマウントされているため、マウントする必要はありません。参照: https://kubernetes.io/docs/tasks/debug-application-cluster/audit/

質問: 34

コンテナ化されたアプリケーションをKubernetesクラスターに構築およびデプロイするための、安全なCI/CDパイプラインを実装する必要があります。このパイプラインには、脆弱性の混入リスクを最小限に抑えるため、各段階でセキュリティチェックと検証ステップを含める必要があります。どのようなセキュリティのベストプラクティスに従いますか？
正解:
解決策（ステップバイステップ）：
1. ソースコードのセキュリティ：
- 静的アプリケーションセキュリティテスト（SAST）SASTツールをCIICDパイプラインに統合して、ソースコードの脆弱性を特定します。
- 依存関係のスキャン：依存関係スキャンツールを使用して、アプリケーションの依存関係における既知の脆弱性を特定します。
- コードレビュー：潜在的な脆弱性を発見するために、本番ブランチへのすべての変更に対して強制的なコードレビューを実施する。
2. コンテナイメージのセキュリティ
- コンテナイメージのスキャン：コンテナイメージをスキャンして、脆弱性やマルウェアを検出します。
- マルチステージビルド：マルチステージDockerビルドを使用して、より小さく安全なコンテナイメージを作成します。
- 署名入り画像：容器に添付する画像に署名することで、画像の真正性を保証し、改ざんを防止します。
3. インフラストラクチャのセキュリティ：
- Infrastructure as Code (IaC): IaCツールを使用してKubernetesのインフラストラクチャと構成を定義し、一貫性とセキュリティを確保します。
- ポリシーの適用：デプロイ中にセキュリティのベストプラクティスを適用するために、Kubernetes のアドミッション コントローラーとポリシーを実装します。
4. 展開時のセキュリティ：
- ロールベースアクセス制御（RBAC）RBACを使用して、機密性の高いKubernetesリソースへのアクセスを制限します。
- ネットワークポリシー：ポッド間の通信を制御するためのネットワークポリシーを実装します。
- 展開戦略：セキュリティインシデントの影響を最小限に抑えるため、ローリングアップデートやカナリア展開などの展開戦略を選択してください。
5. モニタリングと監査：
- Kubernetesのログ記録と監視：イベントを追跡し、潜在的なセキュリティインシデントを特定するために、ログ記録と監視を設定します。
- セキュリティ監査：CI/CDパイプラインとKubernetesクラスターのセキュリティコンプライアンスを定期的に監査します。
6. 継続的なセキュリティ評価：
- セキュリティスキャン：ソースコード、コンテナイメージ、インフラストラクチャを定期的にスキャンして脆弱性を検出します。
- 脆弱性管理：発見された脆弱性を追跡し、修復する。
7. 安全な開発手法：
- 安全なコーディング基準：安全なコーディング基準とベストプラクティスを徹底する。
- セキュリティトレーニング：開発者に対し、一般的な脆弱性に対する意識を高めるためのセキュリティトレーニングを提供する。
- セキュリティバグ報奨金：倫理的なハッカーが脆弱性を発見して報告するよう促すために、セキュリティバグ報奨金の提供を検討してください。

```yaml
# Example GitLab CI/CD YAML file for secure containerized deployment
stages:
  - build
  - test
  - scan
  - deploy

variables:
  # Configure your security tools
  # Example: Snyk for dependency scanning and container image scanning
  SNYK_TOKEN: "$SNYK_TOKEN"

build:
  stage: build
  image: docker:latest
  script:
    - docker build -t my-app .
    - docker push my-app

test:
  stage: test
  image: node:latest
  script:
    - npm install
    - npm test

scan:
  stage: scan
  image: snyk/snyk
  script:
    - snyk test
    - snyk container test my-app:latest
  # Configure snyk to fail the pipeline if vulnerabilities are found
  environment:
    SNYK_TOKEN: $SNYK_TOKEN

deploy:
  stage: deploy
  image: docker:latest
  script:
    - docker login -u $DOCKER_USER -p $DOCKER_PASSWORD $DOCKER_REGISTRY
    - docker push my-app:latest
    - kubectl apply -f deployment.yaml
```


質問: 35

Kubernetes クラスター上で、「web-app」という名前の Web アプリケーションがデプロイされており、このデプロイでは 「web-app-sa」というサービス アカウントが使用されています。「web-app-sa」には、クラスター内の特定のリソースにアクセスするために必要な RBAC ロールと権限が付与されています。デプロイメント仕様に誤って作成または追加された可能性のある、許可されていないサービス アカウントが 「web-app」デプロイで使用されないようにするための戦略を実装したいと考えています。

正解:

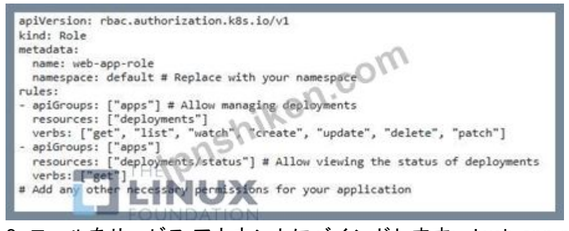
解決策 （ステップバイステップ） :

1. Webアプリケーション用のサービスアカウントを作成する

- 「web-app-sa.yaml」という名前のサービスアカウントYAMLファイルを作成します。

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

2. サービス アカウントのロールを作成します: - 'web-app-sa' に必要な権限を付与するために、'web-app-role.yaml' という名前のロール YAML ファイルを作成します。



```
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. ロールをサービス アカウントにバインドします: - 'web-app-rolebinding.yamr' という名前の ROIeBinding YAML ファイルを作成し、'web-app-roles' を 'web-app-sa' にバインドします。

```
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. Webアプリケーションのデプロイメントを作成します。- 「web-app-sa」とその他の必要な構成を指定する「web-app-deployment.yaml」という名前のデプロイメントYAMLファイルを作成します。

```
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. サービス アカウント、ロール、ロール バインディング、およびデプロイメントを適用します。- kubectl apply -f web-app-sa.yaml web-app-role.yaml web-app-rolebinding.yaml web-app-deployment.yaml を使用して YAML ファイルを適用します。6. 承認されていないサービス アカウントでテストします。- 新しいサービス アカウント (例: 'unauthorized-sa') を作成し、それを 'web-app-deployment' YAML ファイルに追加します。- デプロイメントを更新します。承認されていないサービス アカウントには必要な権限がないため、これは失敗するはずです。- また、承認されていないサービス アカウントを使用してポッドを作成し、権限のないリソースにアクセスできないことを確認することもできます。これらの手順に従うことで、'web-app' デプロイメントが承認された 'web-app-sa' のみを使用してリソースにアクセスすることを保証し、承認されていないサービス アカウントの使用に関連するリスクを軽減するポリシーを効果的に適用できます。

質問: **36**

あなたは、機密性の高いワークロードを実行するKubernetesクラスターのセキュリティを確保する任務を負っています。クラスター内のすべてのPodに対して、最小権限アクセスを強制するメカニズムを実装する必要があります。

正解:

解決策（ステップバイステップ）：

1. 権限を制限したサービスアカウントを作成する：

- 最小限の権限を持つ新しいサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: limited-service-account
```

2. 権限を制限したロールを作成する: - ポッドに必要な権限のみを付与するロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: limited-role
  namespace:
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["core"]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

3. ロールをサービスアカウントにバインドします: - ロールをサービスアカウントにバインドします:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: limited-role-binding
  namespace:
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: limited-role
subjects:
- kind: ServiceAccount
  name: limited-service-account
  namespace:
```

4. PodS をサービス アカウントを使用するように設定する - デプロイメント YAML を更新して、サービス アカウントを使用するようにします。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: limited-service-account
      containers:
      - name: my-app
        image: my-image:latest
      # ... other pod configurations
```

質問: 37

シミュレーション

johnという名前のユーザーを作成し、CSRリクエストを作成し、承認後にユーザーの証明書を取得します。

名前空間john内のシークレットやポッドを一覧表示するロール名john-roleを作成します。

最後に、john-role-binding という名前の RoleBinding を作成し、新しく作成したロール john-role を名前空間 john 内のユーザー john に割り当てます。

確認方法 :kubectl auth CLIコマンドを使用して権限を確認します。
正解:
下記の説明をご覧ください
Explanation:
kubectlを使用してCSRを作成し、承認します。
CSRのリストを取得する：
kubectl get csr
CSRを承認する：
kubectl certificate approve myuser
証明書を取得する
CSRから証明書を取得します。
kubectl get csr/myuser -o yaml
以下は、JohnにNEW_CRDリソースを作成する権限を与えるためのロールとロールバインディングです。
kubectl apply -f roleBindingJohn.yaml --as=john
rolebinding.rbac.authorization.k8s.io/john_external-rosource-rb が作成されました。種類: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 メタデータ:
名前: john_crd
名前空間: development-john
主題:
- 種類: ユーザー
名前 :ジョン
apiグループ: rbac.authorization.k8s.io
役割参照:
種類: ClusterRole
名前: crd-creation
種類: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
メタデータ:
名前: crd-creation
ルール:
- apiGroups: ["kubernetes-client.io/v1"]
リソース: ["NEW_CRD"]
動詞: ["作成する、一覧表示する、取得する"]

質問: **38**
攻撃対象領域を最小限に抑え、セキュリティ上の脆弱性の可能性を低減する、安全なKubernetesクラスタ構成を実装する必要があります。以下の領域に焦点を当て、実装するセキュリティ強化策について説明してください。
- ネットワークセキュリティ :クラスターネットワークを不正アクセスや攻撃から保護するための対策を実施する。
- アドミッションコントロール :ポッド作成時にセキュリティのベストプラクティスを適用するようにアドミッションコントローラーを設定します。
- セキュリティコンテキスト :リソース制限と権限制限を適用するために、ポッドのセキュリティコンテキストを設定します。
- シークレット管理 クラスター内で使用される機密データに対して、安全なシークレット管理を実装します。
正解:

解決策（ステップバイステップ）：
1. ネットワークセキュリティ：
- ネットワークポリシー：ポッド、サービス、および外部エンティティ間の通信を制御するためのネットワークポリシーを実装します。
- ファイアウォールルール：クラスタレベルでファイアウォールルールを設定し、不正な受信および送信トラフィックをブロックします。
- Podの分離：Podセキュリティポリシーとネットワーク名前空間を利用して、Pod同士およびホストシステムからPodを分離します。
- TLS暗号化：APIサーバー、ノード、およびポッド間の通信にTLS暗号化を有効にします。
2. 入学管理：
- PodSecurityPolicy: PodSecurityPolicy を使用して、リソース制限、権限制限、
およびホストリソースへのアクセス権。
- 名前空間認証：名前空間へのアクセスを、承認されたユーザーおよびサービスアカウントに制限します。
- ResourceQuota 名前空間内のリソース消費を制限するために、リソースクォータを設定します。
- NetworkPolicy: NetworkPolicyを使用して、Podと他のエンティティ間のネットワークトラフィックを制御します。
3. セキュリティコンテキスト:
- 特権コンテナ：どうしても必要な場合を除き、特権コンテナの実行は避けてください。
- 機能: コンテナから不要な機能を削除して、攻撃対象領域を縮小します。
- ユーザーおよびグループID：非rootユーザーおよびグループIDを使用してコンテナを実行し、アクセスを制限します。
- 読み取り専用ルートファイルシステム：コンテナのルートファイルシステムを読み取り専用としてマウントし、意図しない変更を防ぎます。
4. 機密情報の管理：
- シークレットの保管：Vault、HashiCorp Vault、AWS Secrets Managerなどの専用のシークレット管理ソリューションを使用して、シークレットを安全に保存します。
- アクセス制御：役割やIDに基づいて機密情報へのアクセスを制限する、堅牢なアクセス制御ポリシーを実装します。
- ローテーション：機密情報が漏洩した場合のリスクを最小限に抑えるため、機密情報を定期的にローテーションしてください。
- シークレットの注入：環境変数、ボリュームマウント、APIなどの安全な方法を使用して、シークレットをポッドに注入します。
5. その他の強化策：
- 定期的な脆弱性スキャン：クラスタコンポーネントとコンテナイメージを定期的にスキャンして脆弱性を検出します。
- ログ記録と監視：不審な活動やセキュリティインシデントを検出するために、包括的なログ記録と監視を実施します。
- セキュリティ監査：定期的にセキュリティ監査を実施し、潜在的なセキュリティ上の弱点を特定して対処します。
- セキュリティのベストプラクティス：Kubernetesのセキュリティに関するベストプラクティスおよび業界標準を遵守してください。

質問: **39**
デフォルトのRBAC構成でKubernetesクラスタが稼働しています。ユーザーが特定のネームスペースにのみアクセスし、そのネームスペース内で特定のアクションを実行できるようにするロールを作成する必要があります。たとえば、ユーザーが「development」ネームスペース内のポッド、デプロイメント、サービスを表示できるようにしつつ、「productions」ネームスペース内のポッドの作成と削除のみを許可したい場合などです。
正解:
解決策（ステップバイステップ）：
1. 「development」名前空間のロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: development-viewer
  namespace: development
rules:
- apiGroups: ["apps", "core", "extensions"]
  resources: ["pods", "deployments", "services"]
  verbs: ["get", "list", "watch"]
```

2. 「production」ネームスペースのロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: production-pod-manager
  namespace: production
rules:
- apiGroups: ["core"]
  resources: ["pods"]
  verbs: ["create", "delete"]
```

3. 「development」名前空間用のROIeBindingを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: development-viewer-binding
  namespace: development
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: development-viewer
subjects:
- kind: User
  name:
```

4. 「production」名前空間のロールバインディングを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: production-pod-manager-binding
  namespace: production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: production-pod-manager
subjects:
- kind: User
  name:
```

5. 'kubectl apply -f' を使用して YAML ファイルを適用します。6. 権限を確認します。それぞれのネームスペースで許可されているアクションを実行してみてください。ロールで定義されているアクションを正常に実行できるはずです。

質問: 40

あなたは、複数のポッドを実行する「web-app」という名前のデプロイメントを持つKubernetesクラスタを管理しています。これらのポッドは、「database-service」という名前の別のKubernetesサービスでホストされているデータベースにアクセスします。ポッドがデータベースサービスにのみ接続でき、クラスタ内の他のサービスにはアクセスできないようにする必要があります。

正解:

解決策（ステップバイステップ）：

1. ネットワークポリシーを作成する：

- 次の内容で「web-app-policy.yaml」という名前のネットワークポリシーYAMLファイルを作成します。

2. ネットワーク ポリシーの適用: - Kubectl apply -f web-app-policy.yamr を使用してポリシーを適用します。3. ネットワーク ポリシーの確認: - 'kubectl get networkpolicieS' を使用してネットワーク ポリシーのステータスを確認し、適用されていることを確認します。4. 接続性のテスト: - 'web-apps デプロイメント内のポッドからデータベース サービスにアクセスしてみます。- 同じポッドから他のサービスにアクセスしてみます。- ネットワーク ポリシーの制限により、'database-service' にのみアクセスできるはずです。

質問: **41**
あなたは、パブリックレジストリのコンテナイメージを使用する「my-app」という名前のデプロイメントを持つKubernetesクラスタを実行しています。最近のデプロイメントの更新によって、コンテナの1つに脆弱性が生じた可能性があると疑っています。コンテナイメージを再構築せずにセキュリティパッチを適用したいと考えています。ここで、「kpatch」のようなコンテナパッチツールを使用してこれを実装し、デプロイメントを更新する方法を説明してください。
正解:
解決策（ステップバイステップ）：
1. 「kpatch」をインストールします。
- システムまたは Kubernetes クラスター内に「kpatch」ツールをインストールしてください。「kpatch」は、Linux カーネルやユーザー空間プログラムを再構築することなくパッチを適用するためのユーティリティです。
2. 脆弱なライブラリを特定する：
Trivyのような脆弱性スキャナーを使用して、コンテナイメージ内の特定の脆弱性のあるライブラリを特定します。
3. 脆弱なライブラリにパッチを適用する：
- 'kpatch' を使用して、実行中のコンテナ内の脆弱なライブラリにセキュリティパッチを適用します。
パッチを適用するには、パッチファイルとコンテナのプロセスIDを指定して「kpatch apply」コマンドを使用できます。
4. デプロイメントを更新する
- 「kpatch」を使用すると実行中のコンテナにパッチを適用できますが、コンテナが再起動するとパッチが失われることに注意してください。永続性を確保するには、パッチが適用されたコンテナイメージを使用するようにデプロイメントを更新する必要があります。
信頼できるソースからパッチ適用済みのコンテナイメージを入手するか、独自のパッチ適用済みイメージを作成してください。
- 「my-app」のデプロイメント構成を更新して、レジストリからパッチ適用済みのイメージを取得するようにします。
5. パッチの検証：
デプロイメントを更新した後、実行中のコンテナに対して脆弱性スキャンを実行し、パッチが正常に適用されたことを確認してください。

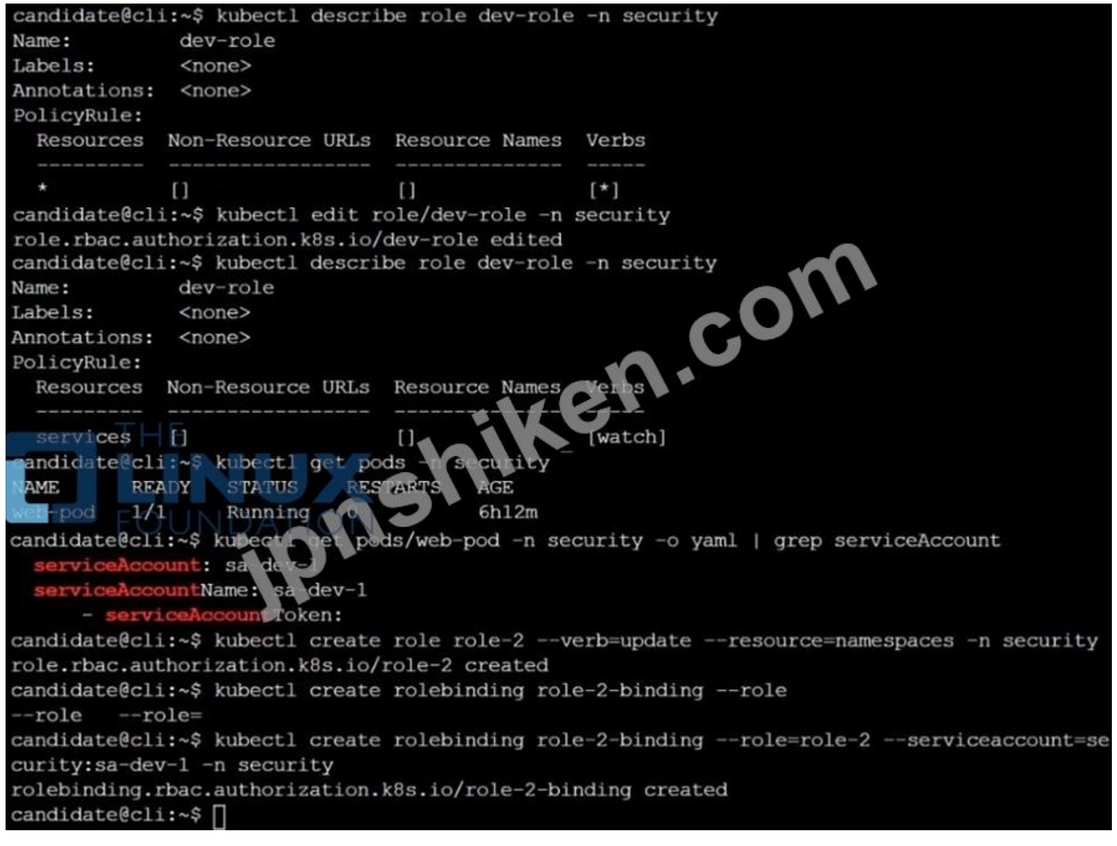
質問: **42**
Kubernetes クラスターでは、機密性の高い API にアクセスするユーザーの認証と認可に、スネアされたシークレットを使用しています。しかし、クラスター内のシークレットが漏洩し、不正アクセスにつながる可能性を懸念しています。HashiCorp Vault のようなシークレット管理ソリューションを使用して、クラスター内のシークレットを安全に管理し、不正アクセスのリスクを最小限に抑えるにはどうすればよいでしょうか？
正解:
解決策（ステップバイステップ）：
1. HashiCorp Vault をデプロイする:
- クラスター内またはクラスター外の安全なインフラストラクチャにVaultをインストールします。これは専用の仮想マシンまたはKubernetesポッドでも構いません。

- Vaultに適切なセキュリティ設定を行います。これには、TLSの有効化、認証方法の設定、アクセス制御の設定が含まれます。
2. シークレット管理の設定：
- Vault 内に共有シークレットを管理するためのシークレット エンジンを作成します。これは「KV」エンジンでも、シークレットを保存するための専用エンジンでも構いません。
共有シークレットはVaultに安全に保管してください。
- Vault を設定して、承認されたアプリケーションまたはサービスのみがシークレットにアクセスできるようにします。これは、Vault のロールとポリシーを使用して実現できます。
3. VaultとKubernetesを統合する
Kubernetes Vaultプロバイダーを使用して、クラスターをVaultに接続します。これにより、KubernetesはVaultに保存されているシークレットにアクセスできるようになります。
- Kubernetes を構成して、シークレット管理に Vault を使用するようにします。これは、Vault Secret Manager を作成することで実現できます。Vault Secret Manager を使用すると、シークレットを Pod やその他の Kubernetes リソースに注入できます。
4. アプリケーションをアップデートする：
アプリケーションを修正し、Kubernetes シークレットから直接アクセスするのではなく、Vault からシークレットを取得するようにします。これにより、アプリケーションが Vault 内の安全なシークレットとやり取りすることが保証されます。
5. シークレットを監視し、ローテーションする：
- Vault内のシークレットへのアクセスと使用状況を定期的に監視する。
共有シークレットを定期的にローテーションするプロセスを実装してください。これにより、シークレットが漏洩した場合の不正アクセスリスクを最小限に抑えることができます。
このアプローチでは、Kubernetesからシークレットを削除し、専用のシークレット管理システムを使用することで、クラスターのシークレットのセキュリティを大幅に向上させます。
この複数段階のプロセスにより、機密データが安全に保管され、承認されたサービスまたはアプリケーションのみがアクセスできることが保証されます。

質問: **43**
シミュレーション
ネームスペース test-system で実行されている既存の Pod nginx-pod が与えられた場合、使用されているサービス アカウント名を取得し、その内容を /candidate/KSC00124.txt に配置します。ネームスペース test-system に、ネームスペースタイプのリソースに対して更新操作を実行できる新しいロール dev-test-role を作成します。
dev-test-role-binding という名前の新しい RoleBinding を作成し、新しく作成した Role を Pod の ServiceAccount (名前空間 test-system で実行されている Nginx Pod 内にあります) にバインドします。
正解:
以下の説明をご覧ください。

```
candidate@cli:~$ kubectl config use-context KSCH00201
Switched to context "KSCH00201".
candidate@cli:~$ kubectl get pods -n security
NAME      READY   STATUS    RESTARTS   AGE
web-pod   1/1     Running   0          6h9m
candidate@cli:~$ kubectl get deployments.apps -n security
No resources found in security namespace.
candidate@cli:~$ kubectl describe rolebindings.rbac.authorization.k8s.io -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  dev-role
Subjects:
  Kind            Name      Namespace
  ----            ----      ---------
  ServiceAccount  sa-dev-1
candidate@cli:~$ kubectl describe role dev-role -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *          []                 []              [*]
candidate@cli:~$ kubectl edit role/dev-role -n security
```

```
  uid: b4c9ddd6-2729-43bd-81bd-b2d22f14c4cd
rules:
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - watch
```



質問: **44**
シミュレーション
準備済みのrunscという名前のランタイムハンドラを使用して、gvisor-rcという名前のRuntimeClassを作成します。
名前空間サーバーにNginxイメージのPodを作成し、gVisorランタイムクラスで実行します。
正解:
以下の説明をご覧ください。
gVisor のランタイム クラスをインストールします
{ # ステップ 1: RuntimeClass をインストールします
cat <<EOF | kubectl apply -f -
APIバージョン: node.k8s.io/v1beta1
種類: ランタイムクラス
メタデータ:

名前: gvisor
ハンドラー: runsc
EOF
}
gVisorランタイムクラスを使用してPodを作成します。
{ # ステップ 2: ポッドを作成する
cat <<EOF | kubectl apply -f -
APIバージョン: v1
種類: ポッド
メタデータ:
名前: nginx-gvisor
仕様:
ランタイムクラス名: gvisor
コンテナ:
- 名前: nginx
画像: nginx
EOF
}
Podが実行されていることを確認してください。
{ # ステップ 3: ポッドを取得する
kubectl get pod nginx-gvisor -o wide
}

質問**: 45**
クラスター: dev
マスターノード: master1 ワーカーノード: worker1
以下のコマンドを使用して、クラスター/構成コンテキストを切り替えることができます。[desk@cli] $ kubectl config use-context dev タスク: 安全な名前空間にある、adam という名前の既存のシークレットの内容を取得します。
ユーザー名フィールドは /home/cert-masters/username.txt というファイルに、パスワードフィールドは /home/cert-masters/password.txt というファイルに保存してください。
1. 両方のファイルを新規作成する必要があります。現状では存在しません。2. 以降の手順では、作成したファイルを使用/変更しないでください。必要に応じて、新しい一時ファイルを作成してください。
安全な名前空間に、newsecret という名前の新しいシークレットを作成し、以下の内容を記述します。ユーザー名: dbadmin パスワード: moresecurepas 最後に、ボリュームを介してシークレット newsecret にアクセスできる新しい Pod を作成します。
名前空間: safe
ポッド名: mysecret-pod
コンテナ名: db-container
画像: redis
ボリューム名: secret-vol
マウントパス: /etc/mysecret
正解**:**

```
witched to context "KSMV00201".
candidate@cli:~$ kubectl get secret -n monitoring
AME                  TYPE                                  DATA   AGE
db1-test              Opaque                                2      6h23m
default-token-cqqf6   kubernetes.io/service-account-token   3      6h23m
candidate@cli:~$ kubectl get secret/db1-test -n monitoring
AME       TYPE     DATA   AGE
db1-test   Opaque   2      6h23m
candidate@cli:~$ kubectl get secret/db1-test -n monitoring -o yaml
piVersion: v1
ata:
  password: QVU3dHh1bXFOTHZt
  username: cHJvZHVjdGlvbi0x
ind: Secret
etadata:
  creationTimestamp: "2022-05-20T08:37:33Z"
  name: db1-test
  namespace: monitoring
  resourceVersion: "2588"
  uid: 659bd4ac-e0ba-4d9f-b411-816f2aedf7e6
ype: Opaque
candidate@cli:~$ echo "cHJvZHVjdGlvbi0x" | base64 -d
roduction-1candidate@cli:~$
andidate@cli:~$
andidate@cli:~$ echo "cHJvZHVjdGlvbi0x" | base64 -d  > /home/candidate/username.txt
andidate@cli:~$ cat /home/candidate/username.txt
roduction-1candidate@cli:~$
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ echo "QVU3dHh1bXFOTHZt" | base64 -d
U7txumqNLvmcandidate@cli:~$ echo "QVU3dHh1bXFOTHZt" | base64 -d > /home/candidate/password.
xt
candidate@cli:~$ cat /home/candidate/password.txt
U7txumqNLvmcandidate@cli:~$
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ kubectl create secret generic test-workflow --from-literal=username=dev-dat
base --from-literal=password=aV7HR7nU3JLx -n monitoring
ecret/test-workflow created
candidate@cli:~$
candidate@cli:~$
candidate@cli:~$ kubectl  -n monitoring run test-secret-pod --image=httpd --dry-run=client -
 yaml > test-secret-pod.yaml
```

```
ind: Pod
etadata:
 labels:
   run: test-secret-pod
 name: test-secret-pod
 namespace: monitoring
pec:
 volumes:
   - name: dev-volume
     secret:
       secretName: test-workflow
 containers:
 - image: httpd
   name: dev-container
   resources: {}
   volumeMounts:
     - name: dev-volume
       mountPath: /etc/credentials
 dnsPolicy: ClusterFirst
 restartPolicy: Always
```



```
candidate@cli:~$ kubectl  -n monitoring run test-secret-pod --image=httpd --dry-run=client -
o yaml > test-secret-pod.yaml
candidate@cli:~$ vim test-secret-pod.yaml
candidate@cli:~$ cat test-secret-pod.yaml
```

質問: **46**
このタスクは、以下のクラスター/ノードで完了する必要があります。
クラスター: トレース
マスターノード: master
ワーカーノード: worker1
以下のコマンドを使用して、クラスタ/構成コンテキストを切り替えることができます。
[desk@cli] $ kubectl config use-context trace
前提条件 :SysdigまたはFalcoのドキュメントを使用できます。
タスク：
Pod tomcatに属する単一のコンテナ内で、プロセスが頻繁に異常な動作を起こしたり、奇妙な処理を実行したりするなどの異常を検出するために、検出ツールを使用してください。
利用できるツールは2つあります。

1. ハヤブサ
2. シスディック
ツールはworker1ノードにのみプリインストールされています。
新たに生成および実行されるプロセスを検出するフィルターを使用して、コンテナの動作を少なくとも40秒間分析します。
インシデントファイルを/home/cert_masters/reportに以下の形式で保存してください。
[タイムスタンプ]、[UID]、[プロセス名]
注 :インシデントファイルは必ずクラスターのワーカーノードに保存し、マスターノードに移動しないでください。
正解:
$vim /etc/falco/falco_rules.local.yaml
- ルール: コンテナのドリフトが検出されました (open+create)
desc: open+create によりコンテナ内に新しい実行ファイルが作成されました
条件: >
evt.type は (open,openat,creat) で、
evt.is_open_exec=true および
コンテナと
runc_writing_exec_fifo ではなく、
runc_writing_var_lib_docker ではなく、
user_known_container_drift_activities ではなく、
evt.rawres>=0
出力: >
%evt.time,%user.uid,%proc.name # これを追加する/Falcoのドキュメントを参照
優先度: エラー
$kill -1 <ファルコのPID>
説明
[desk@cli] $ ssh node01
[node01@cli] $ vim /etc/falco/falco_rules.yaml
「Container Drift Detected」を検索し、falco_rules.local.yamlに貼り付けてください。
[node01@cli] $ vim /etc/falco/falco_rules.local.yaml
- ルール: コンテナのドリフトが検出されました (open+create)
desc: open+create によりコンテナ内に新しい実行ファイルが作成されました
条件: >
evt.type は (open,openat,creat) で、
evt.is_open_exec=true および
コンテナと
runc_writing_exec_fifo ではなく、
runc_writing_var_lib_docker ではなく、
user_known_container_drift_activities ではなく、
evt.rawres>=0
出力: >
%evt.time,%user.uid,%proc.name # これを追加する/Falcoのドキュメントを参照

優先度: エラー
[node01@cli] $ vim /etc/falco/falco.yaml

質問: **47**
staging ネームスペース内の Pod が、同じネームスペース内の他の Pod のポート 80 に接続できるようにする、allow-np という名前のネットワーク ポリシーを作成します。
ネットワークポリシーを確認してください。
1. ポート80でリッスンしていないポッドへのアクセスを許可しません。
2. Pod からのアクセスは許可されていません。ネームスペース staging にもありません。
正解:
apiVersion: networking.k8s.io/v1
種類: ネットワークポリシー
メタデータ:
名前: ネットワークポリシー
仕様:
podSelector: {} #名前空間にデプロイされたすべてのポッドを選択します
ポリシーの種類:
- イングレス
侵入:
- ポート: #入力トラフィックはポート80のみで許可されます
- プロトコル: TCP
ポート: 80

質問: **48**
あなたは、機密データがConfigMapに保存されているKubernetesクラスターのセキュリティを確保する任務を負っています。ConfigMapは、Pod内で実行されている様々なアプリケーションからアクセスされ、その一部は「default」名前空間に配置されています。これらのConfigMapへの不正アクセスを防止する必要があります。
ロールベースアクセス制御（RBAC）を活用してConfigMapへのアクセスを制限し、「default」名前空間内の承認されたPodのみが特定のConfigMapにアクセスできるようにするにはどうすればよいでしょうか？
正解:
解決策（ステップバイステップ）：

1. ロールの定義: 'default' 名前空間に、特定の ConfigMap のみへのアクセスを許可する 'configmap-readerw' という名前のロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: configmap-reader
  namespace: default
rules:
- apiGroups: ["v1"]
  resources: ["configmaps"]
  verbs: ["get", "list", "watch"]
  resourceNames: ["sensitive-data", "app-config"]
```

2. ロールをサービスアカウントにバインドします: 「default」名前空間に 「configmap-app-sa」という名前のサービスアカウントを作成し、「configmap-reader」ロールをそれにバインドします。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: configmap-app-sa
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: configmap-app-sa-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: configmap-reader
subjects:
- kind: ServiceAccount
  name: configmap-app-sa
  namespace: default
```

3. Pods が ServiceAccount を使用するように構成します: これらの ConfigMap へのアクセスが必要な 'defaults' 名前空間で実行されている Pods を更新して、'configmap-app-sa' ServiceAccount を使用するようにします。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: configmap-app-sa
      containers:
      - name: my-app-container
        image: my-app-image
        envFrom:
        - configMapRef:
            name: sensitive-data
        # ... other container configuration
```

「configmap-reader」ロールは、関連付けられたサービスアカウントを持つポッドに特定のConfigMap（「sensitive-data」、「app-config」）へのアクセスを許可します。「configmap-app-sa-binding」はこのロールを「configmap-app-sa」サービスアカウントにバインドします。このサービスアカウントを使用するポッドは、許可されたConfigMapにのみアクセスでき、他のConfigMapへのアクセスは制限されます。注 :このアプローチでは、「default」名前空間の承認されたポッドのみが特定のConfigMapにアクセスできることが保証されます。アクセスを制限する特定のConfigMapを含めるように、ロール定義の「resourceName」を調整する必要があります。ポッドが使用するサービスアカウントに、ConfigMapにアクセスするための適切な権限が付与されていることを確認してください。この例では、RBACを使用してConfigMapへのアクセスを制限し、承認されたポッドのみが機密データにアクセスできるようにする方法を示します。特定のセキュリティ要件を満たすために、必要に応じて権限とサービスアカウントを調整する必要があります。

質問: 49
Kubernetesクラスタに新しいアプリケーションをデプロイしようとしています。このアプリケーションは機密データベースにアクセスする必要があります。アプリケーションコンテナが、権限が制限された専用のサービスアカウントを使用してのみデータベースにアクセスできるようにしたいと考えています。データベースへのアクセスを安全にするために、このアプリケーション用に権限が制限されたサービスアカウントを作成および使用する方法を説明してください。
正解:
解決策（ステップバイステップ）：
1. サービスアカウントの作成: アプリケーション用に、「db-access-sa」のような分かりやすい名前の新しいサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: db-access-sa
  namespace: default  # Your application's namespace
```

- YAML ファイルをクラスターに適用します: bash kubectl apply -f db-access-sa.yaml 2. ロールとロール定義を作成します: サービス アカウントの特定の権限を定義するロールを作成します。このロールは、読み取りや書き込みなど、データベースにアクセスするために必要な権限のみを付与する必要があります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: db-access-role
  namespace: default
rules:
- apiGroups: ["database.example.com"]  # Replace with your database API group
  resources: ["databases"]
  verbs: ["get", "list", "watch", "create", "update", "delete"]
```

- ロールをサービスアカウントに関連付けるRoleBindingを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: db-access-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: db-access-role
subjects:
- kind: ServiceAccount
  name: db-access-sa
  namespace: default
```

- これらの YAML ファイルをクラスターに適用します。3. アプリケーション コンテナの設定: アプリケーションの Deployment または Pod 定義を変更します。新しく作成したサービス アカウントを使用するように 'serviceAccountName' フィールドを指定してください。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: my-app:latest
        # ... other container configurations ...
      serviceAccountName: db-access-sa
      # ... other pod configurations ...
```

4. データベースアクセスをテストする: アプリケーションをデプロイします。アプリケーションコンテナがデータベースに正常に接続し、必要な操作を実行できることを確認します。5. 権限を確認する: アプリケーションコンテナが、明示的に権限が付与されていない他のリソースにアクセスできないことを確認します。

質問: **50**
ランタイム検出ツールであるFalcoを使用して、Nginxの単一コンテナ内で新たに生成および実行されるプロセスを検出するフィルターを使用し、コンテナの動作を少なくとも20秒間分析します。
正解:
検出されたインシデントを含むインシデントファイル /opt/falco-incident.txt を保存します。1 行に 1 つのインシデントを次の形式で記述します。
[タイムスタンプ]、[UID]、[プロセス名]

質問: **51**
コンテクスト
Pod の ServiceAccount にバインドされたロールに、過度に寛容な権限が付与されています。以下の手順を実行して、権限セットを削減してください。
タスク
security 名前空間で実行されている web-pod という名前の既存の Pod があるとします。
Pod の ServiceAccount sa-dev-1 にバインドされている既存のロールを編集して、監視操作のみを、サービスタイプのリソースに対してのみ実行できるようにします。
名前空間セキュリティに、role-2 という名前の新しいロールを作成します。このロールは、名前空間タイプのリソースに対してのみ、更新操作を実行できるようにします。
新しく作成したロールをPodのServiceAccountにバインドする、role-2-bindingという名前の新しいRoleBindingを作成します。

正解:

```
candidate@cli:~$ kubectl config use-context KSCH00201
Switched to context "KSCH00201".
candidate@cli:~$ kubectl get pods -n security
NAME      READY   STATUS    RESTARTS   AGE
web-pod   1/1     Running   0          6h9m
candidate@cli:~$ kubectl get deployments.apps -n security
No resources found in security namespace.
candidate@cli:~$ kubectl describe rolebindings.rbac.authorization.k8s.io -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  dev-role
Subjects:
  Kind            Name      Namespace
  ----            ----      ---------
  ServiceAccount  sa-dev-1
candidate@cli:~$ kubectl describe role dev-role -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *          []                 []              [*]
candidate@cli:~$ kubectl edit role/dev-role -n security
```

質問: **52**
特定のネームスペースで機密性の高いワークロードを実行しているKubernetesクラスターがあります。このネームスペースへのアクセスを許可されたユーザーのみに制限する必要があります。ロールベースアクセス制御（RBAC）を使用してこれを実現するにはどうすればよいでしょうか？
正解:
解決策（手順別）：
1. ロールの作成: 機密性の高い名前空間にアクセスするために必要な権限のみを付与するロールを定義します。
- 名前: 'namespace-access-role' (任意の名前を選択できます)
- 名前空間: アクセスを制限したい名前空間。

- ルール:
- リソース: ロールがアクセスを許可するリソースを指定します。例: 'pods'、'deployments'、'services' など。
- 動詞: リソースに対して許可されるアクションを定義します。たとえば、「get」、「list」、「watch」、「create」、「update」、「delete」などです。
- ApiGroups: リソースが属する API グループを指定します。例: 'apps'、'extensions' など。
ワイルドカードを使用すると、すべてのリソースまたはすべての動詞へのアクセスを許可できます。
2. ROIeBinding を作成します: ロールを特定のユーザーまたはグループに関連付けます。
- 名前: 'namespace-access-binding' (任意の名前を選択できます)
- 名前空間: アクセスを制限したい名前空間。
_ 役割参照。
- 種類: 'ロール' (ロールを使用しているため)
- 名前: 作成した役割の名前。
- APIグループ: 'rbac.authorization.k8s.i0'
- 対象: このロールへのアクセス権限を持つユーザーまたはグループを定義します。
- 種類: ユーザーかグループかを指定します。
- 名前 :ユーザー名またはグループ名。
- ApiGroup: 「bac.authorization.k8s.io」
3. 役割と役割定義を適用する:
- ロールとロールバインディングを作成するには、それぞれ「kubectl apply -f role.yaml」と「kubectl apply -f rolebinding.yaml」を使用します。
ロールとロールインデックスのYAML例：
役割 (role-yaml)

ロールバインディング（olebinding.yaml）

- ロール「namespace-access-role」は、名前空間内の「deployments」、「pods」、「services」、「secrets」へのアクセス権限を付与します。- ロールバインディング「namespace-access-binding」は、このロールをユーザーに関連付けます。- この設定により、名前空間へのアクセスはユーザーのみに制限されます。重要な注意事項: - R8AC は、Kubernetes のリソースへのアクセスを制御する強力なメカニズムです。- さまざまな RBAC コンポーネント (ロール、ロールバインディング、クラスタロール、クラスタロールバインディング) とその使用方法を理解することが重要です。- 最小限の権限アクセスを確保し、セキュリティを強化するために、きめ細かな権限を定義します。

質問: **53**
Kubernetesにおけるセキュリティコンテキストの役割と、コンテナイメージに関連する潜在的なセキュリティリスクを軽減するためにセキュリティコンテキストをどのように活用するかを説明してください。
正解:
解決策（ステップバイステップ）：
1. セキュリティコンテキストの理解：
- Kubernetes のセキュリティ コンテキストは、コンテナのセキュリティ属性を定義し、システム リソースと機能へのアクセスを制御します。
コンテナイメージに関連するセキュリティポリシーを適用し、リスクを軽減します。
2. 主要なセキュリティコンテキスト設定:
- runASUser: コンテナを実行するユーザーIDを指定します。これにより、コンテナユーザーが必要としない可能性のあるファイルやリソースへのアクセスを制限できます。
- runAsGroup: 'runAsUser' と同様ですが、グループ ID 用です。
- fsGroup: ファイルシステムのアクセス許可を制御します。これを設定することで、特定のファイルやディレクトリへのアクセス権を付与できます。
- readOnlyRootFilesystem: コンテナがルートファイルシステムを変更することを防止します
- privileged: コンテナに完全なルート権限を付与します。可能な限り使用を避けるべきです。
- allowPfivilegeEscalatiom コンテナが権限を昇格できるかどうかを制御します。
- capabilities: コンテナが使用できるLinuxの機能を定義します。これにより、特定のシステムリソースや操作へのアクセスを制限できます。
- seLinuxOptions: コンテナの SELinux コンテキストの動作を制御します。これを使用して、SELinux に基づく追加のセキュリティ ポリシーを適用できます。
3. イメージセキュリティのためのセキュリティコンテキストの使用：
- 権限の制限: コンテナの権限を制限するには、「runAsUser」、「runAsGroup」、「privileged」、「allowPrivilegeEscalation」を設定します。
- ファイルシステムへのアクセス制御: 'tsGroup' と 'readOnlyRootFilesystem' を利用して、コンテナがファイルやディレクトリを変更する機能を制限し、潜在的な脆弱性の影響を最小限に抑えます。
- 機能の制限: 「capabilities」フィールドを使用して、コンテナの実行に必要な機能のみを選択的に有効にします。これにより、悪意のある攻撃を防ぐことができます。
機密性の高いシステムリソースへのアクセスを防止するコード。
- SELinux ポリシーの適用: 'seLinuxOptions' を設定して、全体的なセキュリティ要件に合わせたより厳格なセキュリティ ポリシーを適用します。
4. デプロイメントYAMLにおけるセキュリティコンテキストの例：

5. ベストプラクティス: - 最小権限の原則: セキュリティコンテキストに最小権限の原則を適用します。コンテナには、必要なリソースと機能のみを付与します。- セキュリティコンテキスト制約: クラスタのセキュリティコンテキスト制約 (SCC) を定義します。SCCS は、すべてのポッドにわたってセキュリティポリシーを適用します。- 定期的な監査: セキュリティコンテキスト設定を定期的に確認し、セキュリティ要件の変化に合わせて調整します。- セキュリティツールの検討: Kubernetes Security Posture Management (KSPM) やセキュリティスキャンソリューションなどのツールを使用して、セキュリティコンテキスト構成の適用と監視を支援します。

質問: **54**
シミュレーション
コンテクスト
コンテナイメージスキャナをkubeadmでプロビジョニングされたクラスタに完全に統合する必要があります。
タスク
/etc/kubernetes/bouncer にある不完全な設定と、https://smooth-yak.local/review に HTTPS エンドポイントを持つ機能的なコンテナイメージスキャナが与えられた場合、検証アドミッションコントローラを実装するために、以下のタスクを実行します。
まず、APIサーバーを再構成して、提供されたAdmissionConfigurationをサポートするようにすべてのアドミッションプラグインを有効にします。
次に、バックエンド障害発生時に画像を拒否するように ImagePolicyWebhook の設定を再構成します。
次に、バックエンドの設定を完了して、コンテナイメージスキャナのエンドポイントである https://smooth-yak.local/review を指定します。
最後に、設定をテストするために、/home/candidate/vulnerable.yaml で定義されているテストリソースをデプロイします。このリソースは、アクセスを拒否されるべきイメージを使用しています。
必要に応じて、リソースを削除して再作成することができます。
コンテナイメージスキャナのログファイルは、/var/log/nginx/access_log にあります。
正解:
完全な解決策については、以下の説明をご覧ください。
Explanation:
以下は、CKS試験のQ3における「指示通りに実行」形式のランブックです。ファイル名を推測する必要がないように、最小限の検出コマンドと、設定すべき正確な行/ブロックが含まれています。
質問3 - ImagePolicyWebhook（入学の検証）- 試験手順
0) SSH + root
ssh cks000002
sudo -i
1) 提供された設定ファイルを特定する（推測はしない）
ls -la /etc/kubernetes/bouncer
探しているファイルは通常、次のような名前です。
admission_configuration.yaml (AdmissionConfiguration)
imagepolicywebhook.yaml (ImagePolicyWebhookConfiguration) または AdmissionConfiguration kubeconfig (webhook kubeconfig) 内に埋め込まれた ImagePolicyWebhook 設定。どちらがどちらか分からない場合は、以下をご覧ください。
grep -R "ImagePolicyWebhook" -n /etc/kubernetes/bouncer
grep -R "AdmissionConfiguration" -n /etc/kubernetes/bouncer
grep -R "kubeconfig" -n /etc/kubernetes/bouncer
パートA - 必要なアドミッションプラグインを有効にするためにAPIサーバーを再構成する
2) APIサーバーの静的ポッドマニフェストを編集する
/etc/kubernetes/manifests/kube-apiserver.yaml 内
2.1 アドミッションプラグイン ImagePolicyWebhook を有効にする
次の行から始まる行を探してください。
- --enable-admission-plugins
ImagePolicyWebhook がそのカンマリストに含まれていることを確認してください。
例（リストは異なる場合があります。ImagePolicyWebhook を追加してください）：

- --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
フラグが存在しない場合は、コマンド:: の下に 1 行追加します。
- --enable-admission-plugins=ImagePolicyWebhook
2.2 APIサーバーを、提供されたAdmissionConfigurationにポイントする
同じファイル内に、このフラグが存在することを確認してください（AdmissionConfiguration が含まれている /etc/kubernetes/bouncer 内のファイルを使用します）。
- --admission-control-config-file=/etc/kubernetes/bouncer/admission_configuration.yaml ファイル名が異なる場合は、手順 1 で見つけた実際のファイル名を使用しますが、フラグ名は正確に --admission-control-config-file のままにします。
保存/終了:
:wq
静的ポッドは自動的に再起動します（kubeletはマニフェストを監視します）。
オプションのクイックウォッチ：
ドッカー ps | grep kube-apiserver
＃ または：
crictl ps | grep kube-apiserver
パートB - バックエンド障害時に画像を拒否するようにImagePolicyWebhookを設定する
3) ImagePolicyWebhookの設定を編集する
あなたのクラスターでは、以下のいずれかが当てはまります。
オプション 1 (これらのタスクで最も一般的): ImagePolicyWebhook 設定はスタンドアロン ファイルです。/etc/kubernetes/bouncer 内の kind: ImagePolicyWebhookConfiguration を含むファイルを編集します。
grep -R "kind: ImagePolicyWebhookConfiguration" -n /etc/kubernetes/bouncer vi /etc/kubernetes/bouncer/<THE_FILE_YOU_FOUND>.yaml 正確に設定（または確認）してください。
デフォルト許可: false
オプション2 ：ImagePolicyWebhookの設定はAdmissionConfiguration内に埋め込まれています。AdmissionConfigurationファイルを編集してください。
/etc/kubernetes/bouncer/admission_configuration.yaml 内
ImagePolicyWebhook のプラグインセクションを見つけて、設定に以下が含まれていることを確認してください。
デフォルト許可: false
✓ 保存/終了:
:wq
パートC - バックエンド設定をhttps://smooth-yak.local/reviewに指定する
4) ウェブフックのkubeconfigを編集して、スキャナーエンドポイントを使用するように設定します。
ImagePolicyWebhookの設定で参照されているkubeconfigファイルを見つけてください。
kubeConfigFile を検索します:
grep -R "kubeConfigFile" -n /etc/kubernetes/bouncer
そのkubeconfigパスを開きます（以下の例はパス名です。お使いの環境によって異なる場合があります）。
vi /etc/kubernetes/bouncer/kubeconfig
kubeconfigで、クラスタサーバーを正確に設定します。
クラスター:
- クラスター:
サーバー: https://smooth-yak.local/review
✓ 保存/終了:
:wq
パートD - 再起動の影響（APIサーバーが設定を正しく認識していることを確認する）
既に /etc/kubernetes/manifests/kube-apiserver.yaml を編集したため、API サーバーが再起動しました。

安全かつ迅速に行うには、マニフェストを「タッチ」して強制的に再起動してください（コンテンツの変更は不要です）。
touch /etc/kubernetes/manifests/kube-apiserver.yaml
パートE - テスト：脆弱なワークロードを適用し、拒否されることを確認する
5) admin kubeconfig を使用してください（古い kubectl config では動作しない可能性があるため）。
export KUBECONFIG=/etc/kubernetes/admin.conf
kubectl get nodes
6) テストリソースをデプロイする（拒否されるべきである）
kubectl apply -f /home/candidate/vulnerable.yaml
想定される結果：入場エラー／拒否メッセージ。
既に存在する場合：
kubectl delete -f /home/candidate/vulnerable.yaml
kubectl apply -f /home/candidate/vulnerable.yaml
パートF - スキャナが呼び出されたことを確認する（ログチェック）
7) スキャナーのアクセスログを確認する
tail -n 50 /var/log/nginx/access_log
/review にアクセスするリクエストが表示されるはずです。
拒否されない場合に確認すべき簡単な項目
以下の手順を順番に実行してください。
APIサーバーフラグを確認してください：
grep -n "enable-admission-plugins" /etc/kubernetes/manifests/kube-apiserver.yaml grep -n "admission-control-config-file" /etc/kubernetes/manifests/kube-apiserver.yaml deny-on-failure を確認します。
grep -R "defaultAllow" -n /etc/kubernetes/bouncer
必ず提示してください:
デフォルト許可: false
エンドポイントを確認してください:
grep -R "server: https://smooth-yak.local/review" -n /etc/kubernetes/bouncer APIサーバーログ（Dockerランタイム）：
ドッカー ps | grep kube-apiserver
docker logs $(docker ps -q --filter name=kube-apiserver) --tail 80
以下の出力を貼り付けてください。
ls -/etc/kubernetes/bouncer
grep -R "kind: AdmissionConfiguration" -n /etc/kubernetes/bouncer
grep -R "ImagePolicyWebhook" -n /etc/kubernetes/bouncer

質問: **55**
シミュレーション
指定されたDockerfileを分析して編集します。
FROM ubuntu:latest
apt-get update -y を実行してください
apt-install nginx -y を実行します
entrypoint.sh / をコピーします
エントリポイント ["/entrypoint.sh"]

ユーザー ROOT
ファイル内に存在する 2 つの指示を修正します。これはセキュリティのベストプラクティス上の問題として顕著です。デプロイメント マニフェスト ファイルを分析して編集します。apiVersion: v1 kind: Pod metadata:
名前: security-context-demo-2
仕様:
セキュリティコンテキスト:
runAsUser: 1000
コンテナ:
- 名前: sec-ctx-demo-2
画像: gcr.io/google-samples/node-hello:1.0
セキュリティコンテキスト:
runAsUser: 0
特権: 真
allowPrivilegeEscalation: false
ファイル内に存在する2つのフィールドを修正します。これはセキュリティのベストプラクティス上の重要な問題です。構成設定を追加または削除しないでください。既存の構成設定のみを変更してください。いずれかのタスクで権限のないユーザーが必要な場合は、ユーザーID 5487のユーザーtest-userを使用してください。
正解:
debian:latest から
管理者 k@bogotobogo.com
#1 - 走る
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop RUN apt-get clean
#2 - CMD
#CMD ["htop"]
#CMD ["ls", "-l"]
3 - 作業ディレクトリと環境
作業ディレクトリ /root
ENV DZ バージョン1
$ docker image build -t bogodevops/demo .
ビルドコンテキストをDockerデーモンに送信中 3.072kB
ステップ 1/7 : FROM debian:latest
---> be2868bebaba
ステップ2/7 :MAINTAINER k@bogotobogo.com
---> キャッシュを使用しています
---> e2eef476b3fd
ステップ 3/7 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils
---> キャッシュを使用しています
---> 32fd044c1356
ステップ4/7 :DEBIAN_FRONTEND=noninteractive apt-get install -yq htop を実行します
---> キャッシュを使用しています
---> 0a5b514a209e
ステップ5/7 :apt-get cleanを実行する

---> キャッシュを使用しています
---> 5d1578a47c17
ステップ6/7 :WORKDIR /root
---> キャッシュを使用しています
---> 6b1c70e87675
ステップ 7/7 : ENV DZ バージョン1
---> キャッシュを使用しています
---> cd195168c5c7
cd195168c5c7 のビルドに成功しました
bogodevops/demo:latest のタグ付けに成功しました

質問: **56**
Kubernetesクラスタ上でDeploymentを使用してWebアプリケーションを実行しています。アプリケーションコンテナが必要なシステムコールとファイルのみにアクセスできるように、セキュリティ対策を実装したいと考えています。コンテナに過剰な権限を与えてしまう可能性のある脆弱性を懸念しています。Seccompプロファイルを使用してこれを実現する方法を説明し、JSON形式のSeccompプロファイルの例を示してください。
正解:
解決策（ステップバイステップ）：
1. Seccompを理解する：Seccomp（セキュアコンピューティングモード）は、プロセスが実行できるシステムコールを制限できるLinuxカーネルの機能です。許可するシステムコールをリストしたプロファイルを定義することで、コンテナの「サンドボックス」を効果的に作成できます。
2. Seccompプロファイルの作成：SeccompプロファイルはJSON形式で作成できます。以下に例を示します。

```
son

'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. Seccomp プロファイルの適用: Deployment YAML の「securitycontext」フィールドを使用して、Seccomp プロファイルをコンテナに適用できます。

```
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. テストと検証 :デプロイメントをデプロイした後、アプリケーションをテストして、期待どおりに動作することを確認します。Seccompプロファイルで許可されていないコマンドをコンテナ内で実行してみることで、Seccompプロファイルが正しく機能していることを検証できます。

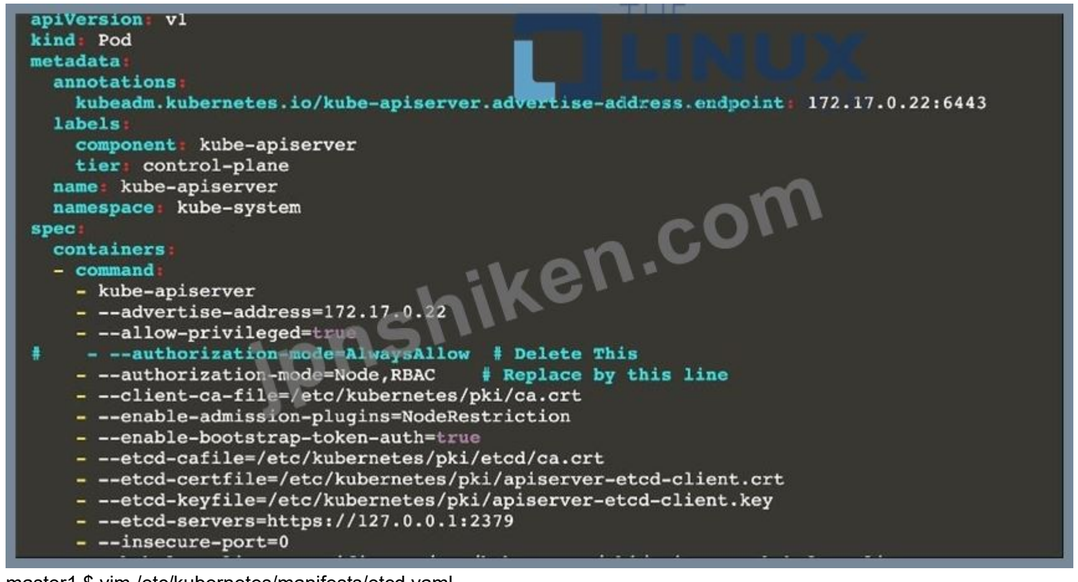
質問: **57**
シミュレーション
ドキュメントのデプロイ、Pod、名前空間
正しいホストに接続する必要があります。接続に失敗すると、スコアが0点になる可能性があります。
[candidate@base] $ ssh cks000028
コンテクスト

コンテナの不変性を確保するには、既存のPodを更新する必要があります。
タスク
既存のデプロイメント（名前空間amp で実行中の lamp-deployment という名前）を以下のように変更し、そのコンテナを以下のようにします。
ユーザーID 20000で実行します
読み取り専用のルートファイルシステムを使用する
権限昇格を禁止する
デプロイメントのマニフェストファイルは、/home/candidate/finer-sunbeam/lamp-deployment.yaml にあります。
正解:
完全な解決策については、以下の説明をご覧ください。
Explanation:
1) 正しいホストに接続します
ssh cks000028
sudo -i
2) 正しいkubeconfigを使用する（試験では安全）
export KUBECONFIG=/etc/kubernetes/admin.conf
3) 提供されたデプロイメントマニフェストを開きます
/home/candidate/finer-sunbeam/lamp-deployment.yaml
4) Podテンプレートのセキュリティ設定のみを編集します（これらのフィールドを追加/変更します）。内部：
仕様: -> テンプレート: -> 仕様:
4.1 コンテナをユーザー20000として実行するように設定する
コンテナ securityContext: の下に追加（または変更）します。
セキュリティコンテキスト:
runAsUser: 20000
4.2 ルートファイルシステムを読み取り専用にする
同じコンテナのsecurityContext内で、以下を保証します。
readOnlyRootFilesystem: true
4.3 権限昇格の禁止
同じコンテナのsecurityContext内で、以下を保証します。
allowPrivilegeEscalation: false
✅ コンテナセクションは次のようになります（例 既存のイメージ/ポートなどを維持してください）。
仕様:
テンプレート：
仕様:
コンテナ:
- 名前: <コンテナ名>
画像：＜変更なし＞
セキュリティコンテキスト:
runAsUser: 20000
readOnlyRootFilesystem: true
allowPrivilegeEscalation: false

コンテナが複数ある場合は、各コンテナに同じsecurityContextを適用してください。
保存して終了：
:wq
5) マニフェストを適用する デプロイメントを更新し、Podを再作成する）
kubectl -n lamp apply -f /home/candidate/finer-sunbeam/lamp-deployment.yaml
6) 展開を待つ
kubectl -n lamp rollout status deployment/lamp-deployment
7) セキュリティ設定が有効になっていることを確認する
7.1 Podが実行されていることを確認する
kubectl -n lamp get pods -l app=lamp -o wide
ラベルが異なる場合は、kubectl -n lamp get pods を実行してください）
7.2 実行中のPodで3つのフィールドを確認する
Pod名を選択して、以下を実行します。
POD=$(kubectl -n lamp get pods -o jsonpath='{.items[0].metadata.name}') kubectl -n lamp get pod $POD -o jsonpath='{.spec.containers[0].securityContext.runAsUser}{"\n"}{.spec.containers[0].securityContext.readOnlyRootFilesystem}{"\n"}
{.spec.containers[0].securityContext.allowPrivilegeEscalation}{"\n"}' 期待される出力:
20000
真実
間違い
readOnlyRootFilesystem=true の後にポッドが失敗した場合
要件を変更しないでください タスクで要求されています）。通常、アプリはボリューム経由で書き込み可能なディレクトリを必要としますが、タスクではそれを要求していません。したがって、マニフェストに既にボリュームがあり、これらのsecurityContextフィールドのみが必要な場合のみ調整してください。

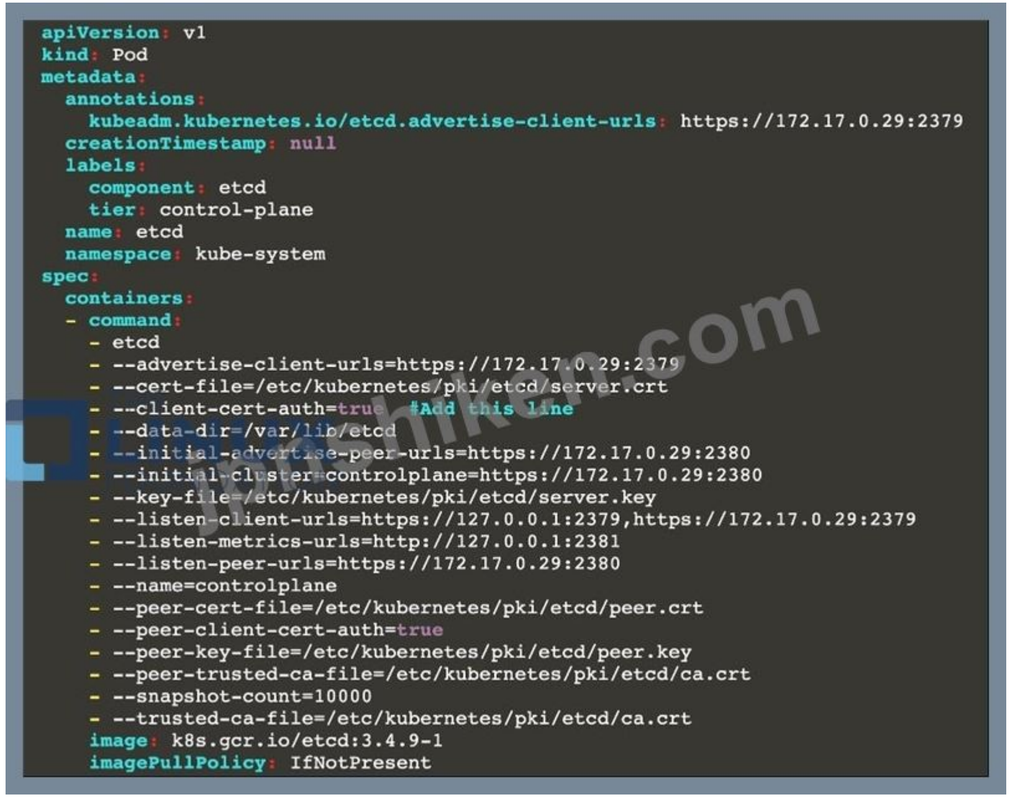
質問: 58
次のコマンドを使用して、クラスタ/構成コンテキストを切り替えることができます。[desk@cli] $ kubectl config use-context dev コンテキスト: kubeadm で作成されたクラスタに対して CIS ベンチマーク ツールが実行され、対処する必要のある複数の問題が見つかりました。タスク: 構成によってすべての問題を修正し、影響を受けるコンポーネントを再起動して、新しい設定が有効になるようにします。APIサーバーに対して検出された以下の違反をすべて修正します。1.2.7 authorization-mode引数がAlwaysAllowに設定されていません FAIL 1.2.8 authorization-mode引数にNodeが含まれています FAIL 1.2.7 authorization-mode引数にRBACが含まれています FAIL Kubeletに対して検出された以下の違反をすべて修正します。4.2.1 anonymous-auth引数がfalseに設定されていることを確認します FAIL 4.2.2 authorization-mode引数がAlwaysAllowに設定されていません FAIL (可能な場合はWebhook autumn/authzを使用してください) etcdに対して検出された以下の違反をすべて修正します。2.2 client-cert-auth引数がtrueに設定されていることを確認します
正解:
worker1 $ vim /var/lib/kubelet/config.yaml
匿名：
enabled: true #これを削除
enabled: false #これに置き換えてください
認証:
モード: 常に許可 #これを削除
モード: Webhook #これに置き換えてください
worker1 $ systemctl restart kubelet. # kubelet 設定をリロードするには、master1 に ssh します。master1 $ vim /etc/kubernetes/manifests/kube-apiserver.yaml - -- authorization-mode=Node,RBAC master1 $ vim /etc/kubernetes/manifests/etcd.yaml - --client-cert-auth=true 説明 worker1 に ssh します。worker1 $ vim /var/lib/kubelet/config.yaml apiVersion: kubelet.config.k8s.io/v1beta1 authentication:
匿名：
enabled: true #これを削除
enabled: false #これに置き換えてください

ウェブフック:
cacheTTL: 0秒
有効: true
x509:
clientCAファイル: /etc/kubernetes/pki/ca.crt
認証:
モード: 常に許可 #これを削除
モード: Webhook #これに置き換えてください
ウェブフック:
cacheAuthorizedTTL: 0秒
キャッシュの認証解除TTL: 0秒
cgroupDriver: systemd
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0秒
退去圧力遷移期間: 0秒
ファイルチェック頻度: 0秒
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0秒
画像最小GC年齢: 0秒
種類: KubeletConfiguration
ログ記録: {}
nodeStatusReportFrequency: 0秒
nodeStatusUpdateFrequency: 0秒
resolvConf: /run/systemd/resolve/resolv.conf
rotateCertificates: true
実行時リクエストタイムアウト: 0秒
staticPodPath: /etc/kubernetes/manifests
ストリーミング接続アイドルタイムアウト: 0秒
同期周波数: 0秒
volumeStatsAggPeriod: 0秒
worker1 $ systemctl restart kubelet. # kubelet の設定をリロードするには、master1 に ssh で接続します。master1 $ vim /etc/kubernetes/manifests/kube-apiserver.yaml



```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 172.17.0.22:6443
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.17.0.22
    - --allow-privileged=true
#    - --authorization-mode=AlwaysAllow  # Delete This
    - --authorization-mode=Node,RBAC     # Replace by this line
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
```

master1 $ vim /etc/kubernetes/manifests/etcd.yaml



```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/etcd.advertise-client-urls: https://172.17.0.29:2379
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://172.17.0.29:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true  #Add this line
    - --data-dir=/var/lib/etcd
    - --initial-advertise-peer-urls=https://172.17.0.29:2380
    - --initial-cluster=controlplane=https://172.17.0.29:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://172.17.0.29:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://172.17.0.29:2380
    - --name=controlplane
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: k8s.gcr.io/etcd:3.4.9-1
    imagePullPolicy: IfNotPresent
```

質問: **59**
ネームスペース test-system で実行されている既存の Pod nginx-pod が与えられた場合、使用されているサービス アカウント名を取得し、その内容を /candidate/KSC00124.txt に配置します。ネームスペース test-system に、ネームスペースタイプのリソースに対して更新操作を実行できる新しいロール dev-test-role を作成します。
dev-test-role-binding という名前の新しい RoleBinding を作成し、新しく作成した Role を Pod の ServiceAccount (名前空間 test-system で実行されている Nginx Pod 内にあります) にバインドします。
正解:

```
candidate@cli:~$ kubectl config use-context KSCH00201
Switched to context "KSCH00201".
candidate@cli:~$ kubectl get pods -n security
NAME      READY   STATUS    RESTARTS   AGE
web-pod   1/1     Running   0          6h9m
candidate@cli:~$ kubectl get deployments.apps -n security
No resources found in security namespace.
candidate@cli:~$ kubectl describe rolebindings.rbac.authorization.k8s.io -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  dev-role
Subjects:
  Kind            Name      Namespace
  ----            ----      ---------
  ServiceAccount  sa-dev-1
candidate@cli:~$ kubectl describe role dev-role -n security
Name:         dev-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *          []                 []              [*]
candidate@cli:~$ kubectl edit role/dev-role -n security
```

```
  uid: b4c9ddd6-2729-43bd-8fbd-b2d227f4c4cd
rules:
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - watch
```

質問: **60**

Kubernetes クラスター上で「my-app」という名前のデプロイメントを実行していますが、予期しないクラッシュが発生しています。クラッシュログによると、コンテナのメモリ使用量がデプロイメント YAML で定義されたリソース制限を超えています。Kubernetes のリソースクォータとアドミッションコントローラーを活用して、このような事態の再発を防ぐ方法を説明してください。

正解:

解決策（ステップバイステップ）：

1. リソースクォータを作成する:

- 特定のネームスペース内のポッドが消費できるリソースを制限するリソースクォータを定義します。

CPU、メモリ、ストレージ、その他のリソースの制限値を指定します。

例えば、「my-app」名前空間内のポッドごとにメモリ使用量を2Giに制限するには、次のようにします。

2. ResourceQuota アドミッション コントローラーを有効にする: - Kubernetes クラスターで 「ResourceQuota」アドミッション コントローラーが有効になっていることを確認します。これは通常、「kube-apiserver」構成で 「admissioncontrolr」フラグを設定することで実行できます。3. ResourceQuota を適用する: - 「kubectl apply -f resource-quota_yaml」を使用して、「my-app」名前空間に ResourceQuota を適用します。4. デプロイメントを更新する - デプロイメントの YAML ファイルを変更して、コンテナのリソース要求と制限を指定し、それらが定義された ResourceQuota 制限の範囲内であることを確認します。例:

5. 更新されたデプロイメントを適用します - 'kubectl apply -f deployment.yaml' を使用して更新されたデプロイメントを適用します。6. 監視と評価: - "my-app" 名前空間のポッドのリソース消費を監視し、必要に応じて ResourceQuota 制限を調整して、クラスターが安定していることを確認します。

質問: **61**
シミュレーション
名前空間 testing で実行されている nginx-test ポッドに制限するための restrict-np という名前のネットワーク ポリシーを作成します。
Pod nginx-test への接続を許可する Pod は以下のとおりです。
1. 名前空間 default 内のポッド
2. ラベル version:v1 を持つ、任意の名前空間内のポッド。
ネットワークポリシーを必ず適用してください。
正解:
これについてのご意見をお聞かせください。

質問: **62**
あなたは、複数のデプロイメントで異なるマイクロサービスを実行しているKubernetesクラスタを管理しています。権限昇格やその他のセキュリティ脆弱性のリスクを最小限に抑えるため、すべてのPodが適切なセキュリティコンテキスト制約（SCC）で実行されていることを確認する必要があります。SCCを使用してPodのセキュリティ標準を実装および適用する方法を説明し、一般的なセキュリティ制約の具体的な例と、さまざまなデプロイメントシナリオに合わせてそれらを構成する方法を示してください。

正解:

解決策（ステップバイステップ）：

1. セキュリティコンテキスト制約（SCC）を定義する：

- 新しいSCCリソースを作成します。以下は、「restricted-scc」という名前の制限付きSCCの例です。

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: restricted-scc
spec:
  allowPrivilegeEscalation: false  # Disable privilege escalation
  runAsUser:
    type: RunAsAny
    # Add specific user IDs if you need more control
  seLinuxContext:
    type: RunAsAny
    # Define specific SELinux labels if needed
  supplementalGroups:
    type: RunAsAny
    # Define specific supplemental groups if needed
  readOnlyRootFilesystem: true  # Mount root filesystem as read-only
  privileged: false             # Disallow running as privileged container
  allowHostNetwork: false       # Disallow using host network
  allowHostPorts: false         # Disallow binding to host ports
  allowHostDir: false           # Disallow mounting host directories
  allowHostIPC: false           # Disallow sharing host IPC namespace
  allowHostPID: false           # Disallow sharing host PID namespace
  allowHostDevices: false       # Disallow access to host devices
  allowVolumeExpansion: false  # Disallow expanding volumes
  volumes:
    - 'configMap'
    - 'secret'
    - 'downwardAPI'
    - 'emptyDir'
    - 'persistentVolumeClaim'
    - 'projected'
    - 'serviceAccount'
```

2. SCC をデプロイメントに適用する: - SCC を適用するには、デプロイメント リソースに「securitycontext」セクションを追加します。- 例を以下に示します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: nginx:latest
        securityContext:
          # Use the previously defined SCC
          securityContextConstraints: restricted-scc
```

3. テストと評価: - SCC を使用してデプロイした後、デプロイをテストし、ポッドが期待どおりのセキュリティ制限で作成されていることを確認します。- 'kubectl get pods -l app=my-apps' を使用して、ポッドの状態とセキュリティ コンテキストを確認します。例の主要セキュリティ制約: - 'allowPrivilegeEscalation: false': コンテナが権限を昇格することを防止します。- 'readOnlyRootFilesystem: true': ルート ファイルシステムの変更を防止し、悪意のあるコードによる改ざんのリスクを軽減します。- 'privileged: false: コンテナをルート権限で実行することを禁止し、セキュリティ リスクを軽減します。- 'volumes': 使用できるボリュームの種類を制限し、機密データまたはリソースへのアクセスを制限します。デプロイ シナリオ: - 機密データを扱う重要なサービスには、提供されているような非常に制限の厳しい SCC を使用します。- 重要度の低いサービスには、より寛容な SCC が必要になる場合があります。- セキュリティ要件のレベルに応じて、異なる SCC を作成し、それぞれを適用できます。重要な注意事項: - 本番環境に実装する前に、必ず SCC を徹底的にテストしてください。- SCC が効果的であり、セキュリティのベストプラクティスに準拠していることを確認するために、定期的に SCC を見直し、更新してください。- Kubernetes セキュリティ スキャン ツールを使用して、デプロイメントと SCC 構成における潜在的な脆弱性を特定することを検討してください。

質問: **63**
シミュレーション
クラスターで監査ログを有効にするには、ログバックエンドを有効にし、
1. ログは /var/log/kubernetes-logs.txt に保存されます。
2. ログファイルは12日間保持されます。
3. 最大で8個の古い監査ログファイルが保持されます。
4. 回転前の最大サイズを200MBに設定する
基本ポリシーを編集して拡張し、以下の内容をログに記録するようにします。
1. RequestResponse でのネームスペースの変更
2. 名前空間 kube-system 内のシークレットの変更に関するリクエストボディをログに記録します。
3. コアおよび拡張機能内のその他のすべてのリソースをリクエストレベルでログに記録します。
4. メタデータレベルで「pods/portforward」、「services/proxy」をログに記録します。
5. ステージ RequestReceived を省略する
メタデータレベルでのその他のすべてのリクエスト
正解:
Kubernetesの監査機能は、クラスタに関するセキュリティ関連の時系列レコードを提供します。Kube-apiserverが監査を実行します。各リクエストは実行の各段階でイベントを生成し、そのイベントは特定のポリシーに従って前処理され、バックエンドに書き込まれます。ポリシーによって記録される内容が決定され、バックエンドはレコードを永続化します。
CIS（Center for Internet Security）のKubernetesベンチマークコントロールへの準拠の一環として、監査ログを設定することをお勧めします。
監査ログは、cluster.yml ファイルに以下の設定を追加することで、デフォルトで有効にできます。
サービス:
kube-api:
監査ログ:
有効: true
監査ログが有効になっている場合、デフォルト値は /etc/kubernetes/audit-policy.yaml で確認できます。ログバックエンドは、監査イベントを JSONlines 形式でファイルに書き込みます。ログ監査バックエンドは、次の kube-apiserver フラグを使用して設定できます。
--audit-log-path は、ログバックエンドが監査イベントを書き込むために使用するログファイルパスを指定します。このフラグを指定しない場合、ログバックエンドは無効になります。- は標準出力を意味します。
--audit-log-maxage は、古い監査ログファイルを保持する最大日数を定義します。
--audit-log-maxbackup は、保持する監査ログファイルの最大数を定義します。
--audit-log-maxsize は、監査ログファイルがローテーションされる前の最大サイズをメガバイト単位で定義します。クラスターのコントロールプレーンが kube-apiserver を Pod として実行している場合は、監査レコードが永続化されるように、hostPath をポリシーファイルとログファイルの場所にマウントすることを忘れないでください。例:
--audit-policy-file=/etc/kubernetes/audit-policy.yaml \
--audit-log-path=/var/log/audit.log

質問: **64**
ランタイム検出ツールであるFalcoを使用して、Nginxの単一コンテナ内で新たに生成および実行されるプロセスを検出するフィルターを使用し、コンテナの動作を少なくとも20秒間分析します。
検出されたインシデントを1行に1つずつ、以下の形式でインシデントファイル /opt/falco-incident.txt に保存します。
[タイムスタンプ]、[UID]、[プロセス名]
A. 私たちにあなたの
B. それについてのご意見をお聞かせください。
正解: **B** (コメントを発表する)

質問: **65**
名前空間の制限内で、ボリュームタイプとしてpersistentvolumeclaimのみを許可するPSPを作成します。
persistentvolumeclaim 以外のボリュームを持つ Pod がマウントされるのを防止する、prevent-volume-policy という名前の新しい PodSecurityPolicy を作成します。
制限付き名前空間に、psp-sa という名前の新しいサービスアカウントを作成します。
新しく作成したポッド セキュリティ ポリシー prevent-volume-policy を使用する psp-role という名前の新しい ClusterRole を作成します。作成した ClusterRole psp-role を作成した SA psp-sa にバインドする psp-role-binding という名前の新しい ClusterRoleBinding を作成します。
ヒント：
また、Pod マニフェストでシークレットをマウントしようとして、設定が正しく機能しているかどうかを確認してください。マウントは失敗するはずです。
PODマニフェスト：
APIバージョン: v1
種類: ポッド
メタデータ:
名前：
仕様:
コンテナ:
- 名前：
画像：
ボリュームマウント:
- 名前：
マウントパス:
ボリューム:
- 名前：
秘密：
シークレットネーム:
正解:
APIバージョン: policy/v1beta1
種類: サブセキュリティポリシー
メタデータ:
名前: 制限付き
注釈:
seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime/default' apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default' seccomp.security.alpha.kubernetes.io/defaultProfileName: 'runtime/default'
apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default' spec:
特権: false
root権限への昇格を防ぐために必要です。
allowPrivilegeEscalation: false
これは非ルート + 権限昇格の禁止と重複しています。
しかし、我々はそれを多層防御のために提供することができる。
必要なドロップ機能:
- 全て
コアボリュームタイプを許可します。

ボリューム:
- 'configMap'
- 'emptyDir'
- 予測された」
- '秘密'
- 'downwardAPI'
クラスター管理者が設定した persistentVolumes は安全に使用できるものと仮定します。
- 'persistentVolumeClaim'
hostNetwork: false
hostIPC: false
hostPID: false
runAsUser:
コンテナをroot権限なしで実行するように要求します。
ルール: 'MustRunAsNonRoot'
seLinux:
このポリシーは、ノードが SELinux ではなく AppArmor を使用していることを前提としています。
ルール: 'RunAsAny'
補足グループ:
ルール: 'MustRunAs'
範囲:
ルートグループの追加を禁止します。
- 最小値: 1
最大: 65535
fsGroup:
ルール: 'MustRunAs'
範囲:
ルートグループの追加を禁止します。
- 最小値: 1
最大: 65535
readOnlyRootFilesystem: false

質問: 66
あなたは、CIS Kubernetesベンチマークに従ってKubernetesクラスタのセキュリティを強化する任務を負っています。etcdクラスタを保護するために、保存データの強力な暗号化構成を使用するソリューションを実装する必要があります。etcdクラスタで必要な構成変更と、Kubernetesコントロールプレーンにおける関連する構成変更を含め、関連する手順を説明してください。
正解:
解決策（ステップバイステップ）：
1. 暗号化キーを生成する：
- 'openssr' や 'gpg' などのツールを使用して強力な暗号化キーを作成します。
例えば、OpenSSLを使用する場合：
バッシュ
openssl genrsa -out etcd-encryption-key.pem 4096

2. etcdの設定：
- etcdサーバーの設定ファイルを編集します。通常は「/etc/etcd/etcd.conf」にありますが、設定によっては同様のパスになる場合があります。
- '--data-dirs' に、暗号化された etcd データを保存するディレクトリを設定します。
- 保存時の暗号化を有効にし、暗号化キーを指定するには、以下の行を追加してください。

```
ETCD_CIPHERTEXT_KEY_FILE="path/to/etcd-encryption-key.pem"
```

- 'path/to:etcd-encryption-key.pem' を暗号化キー ファイルへの実際のパスに置き換えてください。3. Kubernetes コントロール プレーンの構成: - kubeadm を使用している場合は、'kubeadm init' または 'kubeadm joins' コマンドを変更して、tne =etcd.encryption-key' オプションに暗号化キー ファイルへのパスを含めます。例: bash kubeadm init -etcd.encryption-key=path/to/etcd-encryption-key.pem 4. etcd とコントロール プレーン コンポーネントの再起動: - etcd サーバーと 'kube-apiserver' や 'kube-controller-manager' などの Kubernetes コントロール プレーン コンポーネントを再起動して、新しい構成がロードされていることを確認します。5. 暗号化の確認: - コンポーネントを再起動した後、etcd 構成で '-data-dir' として構成されているディレクトリを確認して、etcd データが暗号化されていることを確認します。暗号化されたファイルが表示されるはずです。6. 暗号化キーのローテーション（推奨）:セキュリティを強化するため、暗号化キーを定期的にローテーションしてください。これには、新しいキーの生成、etcd 設定への新しいキーの更新、etcd およびコントロール プレーンコンポーネントの再起動が含まれます。

質問: 67
KubernetesクラスタにNginxポッドがデプロイされています。Nginxポッドが特定のネットワークポートのみにアクセスできるようにし、他のポートへのアクセスを禁止するPodSecurityPolicy（PSP）を設定する必要があります。このPSPを実装するために実行する手順を説明してください。
正解:
解決策（ステップバイステップ）：
1. PodSecurityPolicy (PSP) を作成します。
- PodSecurityPolicy (PSP) YAML ファイルを定義します。このファイルは、Nginx ポッドのネットワークアクセスを制限します。
例えば、Nginxポッドがポート80と443（HTTPとHTTPS）にアクセスできるようにするが、他のすべてのポートへのアクセスを制限するPSPを定義することができます。

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: nginx-psp
spec:
  # ... other PSP settings ...
  hostNetwork: false
  runAsUser:
    rule: "RunAsAny"
  # ... other settings ...
  seLinux:
    rule: "RunAsAny"
  # ... other settings ...
  fsGroup:
    rule: "RunAsAny"
  # ... other settings ...
  supplementalGroups:
    rule: "RunAsAny"
  # ... other settings ...
  volumes:
    # Allow emptyDir
    - name: nginx-data
      new: true
      # ... other settings ...
  hostPorts:
    - port: 80
      protocol: TCP
    - port: 443
      protocol: TCP
```

2. PSPI を適用する - 'kubectl apply -f nginx-psp.yaml' を使用して PSP YAML ファイルを適用します。3. PSP を Nginx Pod にバインドします。- Nginx デプロイメントまたは Pod 定義を更新して、作成した PSP への参照を含む 'securityContext' フィールドを追加します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        securityContext:
          # ... other securityContext settings ...
          # Add the PSP name to the securityContext
          securityContext:
            podSecurityPolicy: nginx-psp
      # ... other container settings ...
```

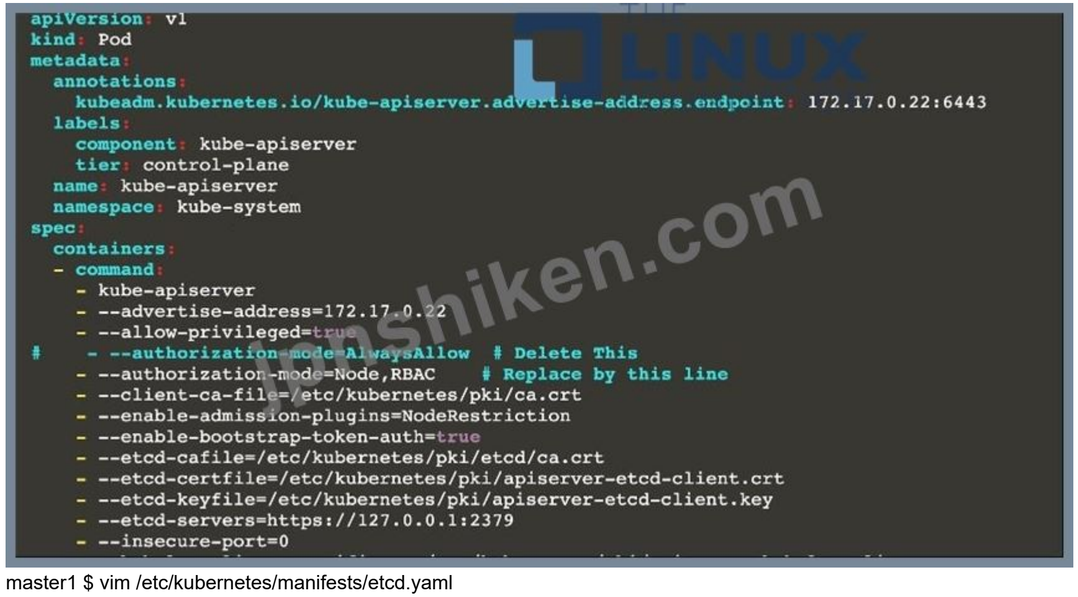
4. PSP を確認する: - 'kubectl describe pod' を実行して、Nginx pod が PSP を使用していることを確認します。 Security Context」セクションに PSP 名がリストされているはずです。5. アクセスをテストする: - Nginx pod がポート 80 と 443 にはアクセスできるが、他のポートにはアクセスできないことを確認します。接続をテストするには、'telnet' や 'nc' などのツールを使用できます。

質問: **68**
シミュレーション
ランタイム検出ツールであるFalcoを使用して、Nginxの単一コンテナ内で新たに生成および実行されるプロセスを検出するフィルターを使用し、コンテナの動作を少なくとも20秒間分析します。
検出されたインシデントを1行に1つずつ、以下の形式でインシデントファイル /opt/falco-incident.txt に保存します。
[タイムスタンプ]、[UID]、[プロセス名]
正解:
それについてのご意見をお聞かせください。

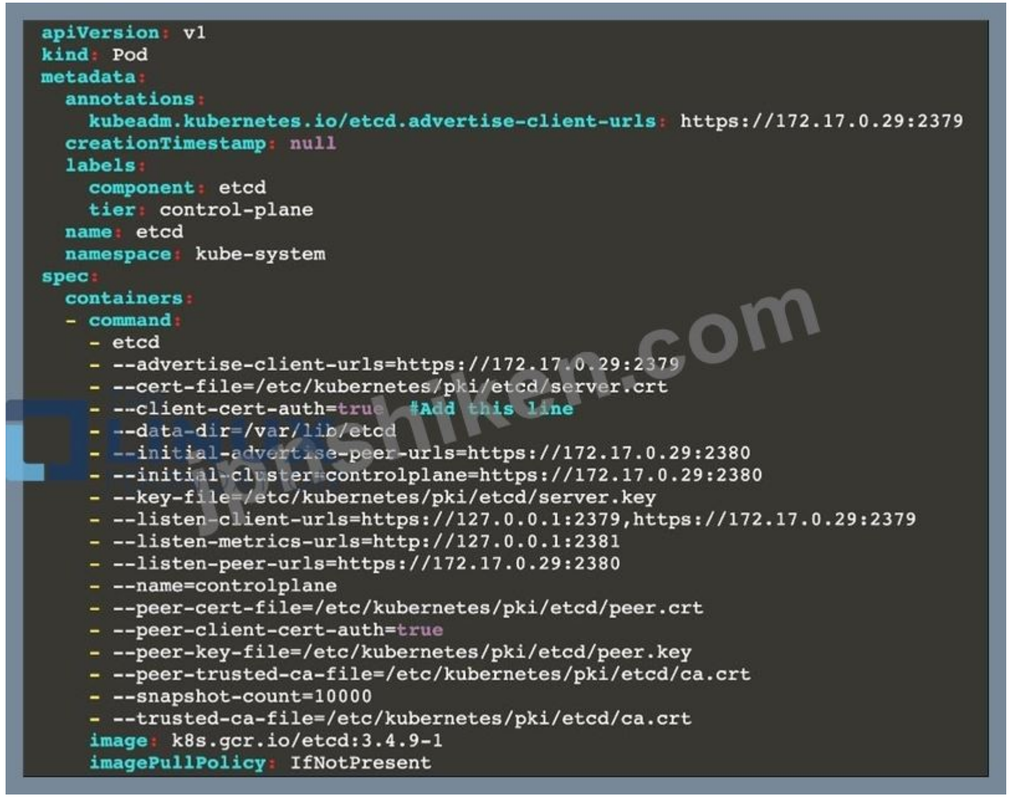
質問: **69**
シミュレーション
以下のコマンドを使用して、クラスタ/構成コンテキストを切り替えることができます。
[desk@cli] $ kubectl config use-context dev
コンテクスト：
kubeadmで作成されたクラスターに対してCISベンチマークツールを実行したところ、対処が必要な複数の問題が見つかりました。
タスク：
設定変更によってすべての問題を修正し、影響を受けるコンポーネントを再起動して、新しい設定が有効になるようにしてください。
APIサーバーに対して検出された以下の違反事項をすべて修正してください。
1.2.7 authorization-mode引数がAlwaysAllowに設定されていません。失敗
1.2.8 authorization-mode引数にNode FAILが含まれる
1.2.7 authorization-mode 引数に RBAC FAIL が含まれます
Kubeletに対して検出された以下の違反をすべて修正してください。
4.2.1 anonymous-auth引数がfalseに設定されていることを確認してください。失敗
4.2.2 authorization-mode 引数が AlwaysAllow に設定されていません FAIL (可能な場合は Webhook autumn/authz を使用してください) etcd に対して見つかった以下の違反をすべて修正してください。
2.2 client-cert-auth引数がtrueに設定されていることを確認してください。

正解:
下記の解説をご覧ください
Explanation:
worker1 $ vim /var/lib/kubelet/config.yaml
匿名：
enabled: true #これを削除
enabled: false #これに置き換えてください
認証:
モード: 常に許可 #これを削除
モード: Webhook #これに置き換えてください
worker1 $ systemctl restart kubelet. # kubelet の設定を再読み込みします
master1にssh接続
master1 $ vim /etc/kubernetes/manifests/kube-apiserver.yaml
-- authorization-mode=Node,RBAC
master1 $ vim /etc/kubernetes/manifests/etcd.yaml
--client-cert-auth=true
Explanation:
worker1にssh接続
worker1 $ vim /var/lib/kubelet/config.yaml
apiVersion: kubelet.config.k8s.io/v1beta1
認証:
匿名：
enabled: true #これを削除
enabled: false #これに置き換えてください
ウェブフック:
cacheTTL: 0秒
有効: true
x509:
clientCAファイル: /etc/kubernetes/pki/ca.crt
認証:
モード: 常に許可 #これを削除
モード: Webhook #これに置き換えてください
ウェブフック:
cacheAuthorizedTTL: 0秒
キャッシュの認証解除TTL: 0秒
cgroupDriver: systemd
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0秒
退去圧力遷移期間: 0秒

ファイルチェック頻度: 0秒
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0秒
画像最小GC年齢: 0秒
種類: KubeletConfiguration
ログ記録: {}
nodeStatusReportFrequency: 0秒
nodeStatusUpdateFrequency: 0秒
resolvConf: /run/systemd/resolve/resolv.conf
rotateCertificates: true
実行時リクエストタイムアウト: 0秒
staticPodPath: /etc/kubernetes/manifests
ストリーミング接続アイドルタイムアウト: 0秒
同期周波数: 0秒
volumeStatsAggPeriod: 0秒
worker1 $ systemctl restart kubelet. # kubelet の設定を再読み込みします
master1にssh接続
master1 $ vim /etc/kubernetes/manifests/kube-apiserver.yaml



master1 $ vim /etc/kubernetes/manifests/etcd.yaml



```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/etcd.advertise-client-urls: https://172.17.0.29:2379
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://172.17.0.29:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true  #Add this line
    - --data-dir=/var/lib/etcd
    - --initial-advertise-peer-urls=https://172.17.0.29:2380
    - --initial-cluster=controlplane=https://172.17.0.29:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://172.17.0.29:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://172.17.0.29:2380
    - --name=controlplane
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: k8s.gcr.io/etcd:3.4.9-1
    imagePullPolicy: IfNotPresent
```

質問: **70**
AWS上でKubernetesクラスターを運用しており、機密データ処理を含むワークロードを実行しています。一部のPodが侵害され、外部サーバーにデータが漏洩している可能性があると疑っています。侵害されたPodを特定し、ネットワークから隔離する必要があります。ネットワークトラフィックの監視、不審なアクティビティの特定、侵害されたPodの隔離に使用するツールと手法を含め、これを実現するための手順を説明してください。
正解:
解決策（手順別）：
1. ネットワークポリシーを有効にする :まず、Kubernetesクラスターでネットワークポリシーを有効にします。これにより、定義済みのルールに基づいてポッド間のネットワークトラフィックが制限されます。
実装：

2. 次のようなツールを使用してネットワーク トラフィックを監視します。Kubernetes ネットワーク ポリシー: クラスターに構成されているネットワーク ポリシーを分析して、疑わしいトラフィック パターンを特定します。Kube-Proxy: 'kubectl proxy' を使用して、クラスター内のネットワーク トラフィックを監視します。受信トラフィックと送信トラフィックを監視して、異常なパターンを特定します。ネットワーク セキュリティ監視ツール: より包括的なネットワーク分析のために、Suricata、Zeek、または tcpdump などの専用のネットワーク セキュリティ監視ツールの使用を検討します。実装: bash kubectl proxy --port=8001 # kubectl proxy を起動します # 別のターミナルで、次のコマンドを実行して、特定の pod へのトラフィックを表示します。curl -v http://localhost'8001/api/v1/namespaces/default/pods//proxy/ # 出力を分析して、疑わしいトラフィックを特定します。3. ログを分析して疑わしいアクティビティを検出します。Kubernetes ログ: 'kubectl logs' などのツールを使用して、pod のログ、特にデータ処理に関連するログを調べます。不正アクセス、データ漏洩の試み、または異常なアクティビティ パターンの兆候を探します。セキュリティ ログ: クラスターを構成して、セキュリティ関連のイベントとログを Elasticsearch、Fluentd、Kibana (EFK) スタックなどの集中ログ システムで収集します。セキュリティ監視ツール: Falco や Auditd などのツールを使用して、Kubernetes クラスター内のセキュリティ関連のイベントを積極的に監視および分析します。実装: bash kubectl logs -f # pod のログを表示します 4. 侵害された Pod を隔離します: ネットワーク セグメンテーション: ネットワーク ポリシーを使用して、疑わしい Pod のネットワーク アクセスを制限します。Pod 中断予算 (PDB): 隔離プロセス中にワークロードが利用できなくなることがないようにします。サービスの中断: 侵害された Pod がサービスに属している場合は、一時的にサービスのエンドポイント リストから削除して、侵害されたサービス インスタンスを隔離することを検討します。実装:

5. 調査と修復: 根本原因分析: 侵害されたポッドが隔離されたら、徹底的な分析を実行して侵害の原因を特定します。これには、システムログ、ネットワークトラフィックの調査、および侵害されたポッドに対するフォレンジック分析の実行が含まれる場合があります。セキュリティ修復: 脆弱性のパッチ適用、セキュリティ構成の更新、およびシステムのナーデン化によって、侵害の根本原因に対処します。復旧と復元: 必要に応じて、漏洩した可能性のあるデータを復元し、システムを安全な状態に復元します。実装: bash # 侵害の原因を調査: kubectl logs -f # kubectl proxy およびネットワーク監視ツールを使用して、ポッドに関連するネットワークトラフィックを分析します。# 侵害を修復: kubectl delete pod # 侵害されたポッドの名前に置き換えてください # セキュリティ構成を更新します # 脆弱性にパッチを適用します # 更新されたセキュリティ対策を備えた新しいコンテナイメージの使用を検討します # 必要に応じてデータを復元します

質問: 71
Kubernetes クラスターには、異なる名前空間で実行されている複数のアプリケーションがあります。「monitoring」名前空間内の Pod のみが「api-server」名前空間内の Pod と通信できるようにするポリシーを適用したいと考えています。NetworkPolicies を使用してこれを実現するにはどうすればよいでしょうか？
正解:
解決策（ステップバイステップ）：
1. ネットワークポリシーの作成: 許可する通信を定義するために、「monitoring-access.yaml」という名前の NetworkPolicy YAML ファイルを作成します。

- このポリシーは、「monitoring」名前空間内のポッドからのみ「api-server」名前空間へのイングレス トラフィックを許可します。2. ネットワーク ポリシーの適用: 'kubectr' を使用して NetworkPolicy を適用します: bash kubectl apply -f monitoring-access-yaml 3. ネットワーク ポリシーの確認: NetworkPolicy が適用されていることを確認します: bash kubectl get networkpolicies -n api-server 4. アクセスのテスト: 「monitoring」名前空間のポッドから「api-server」名前空間のポッドへの通信を試みます。この通信は許可されるはずです。別の名前空間のポッドから「api-server」名前空間のポッドへの通信を試します。この通信はブロックされるはずです。この NetworkPolicy は、「api-server」名前空間へのイングレス トラフィックを制限します。「monitoring」名前空間内のポッドからの接続のみを許可し、これらの名前空間間で制御されたアクセス ポリシーを効果的に適用します。

質問: **72**
コンテナイメージスキャナがクラスタ上にセットアップされています。
ディレクトリ内の設定が不完全です
/etc/kubernetes/confcontrol と、HTTPS エンドポイント https://test-server.local.8081/image_policy を備えた機能的なコンテナイメージスキャナ
正解:
1. 入退室管理プラグインを有効にします。
2. コントロール構成を検証し、暗黙的拒否に変更します。
最後に、イメージタグが「latest」のポッドをデプロイして、設定をテストしてください。