



**Exam** : **CKS**

**Title** : Certified Kubernetes  
Security Specialist (CKS)

**Version** : DEMO

## 1.CORRECT TEXT

Create a PSP that will only allow the persistentvolumeclaim as the volume type in the namespace restricted.

Create a new PodSecurityPolicy named prevent-volume-policy which prevents the pods which is having different volumes mount apart from persistentvolumeclaim.

Create a new ServiceAccount named psp-sa in the namespace restricted.

Create a new ClusterRole named psp-role, which uses the newly created Pod Security Policy prevent-volume-policy

Create a new ClusterRoleBinding named psp-role-binding, which binds the created ClusterRole psp-role to the created SA psp-sa.

Hint:

Also, Check the Configuration is working or not by trying to Mount a Secret in the pod manifest, it should get failed.

POD Manifest:

⌘ apiVersion: v1

⌘ kind: Pod

⌘ metadata:

⌘ name:

⌘ spec:

⌘ containers:

⌘ - name:

⌘ image:

⌘ volumeMounts:

⌘ - name:

⌘ mountPath:

⌘ volumes:

⌘ - name:

⌘ secret:

⌘ secretName:

**Answer:**

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: restricted

annotations:

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:

privileged: false

# Required to prevent escalations to root. allowPrivilegeEscalation: false

# This is redundant with non-root + disallow privilege escalation,

```
# but we can provide it for defense in depth. requiredDropCapabilities:
- ALL
# Allow core volume types.
volumes:
- 'configMap'
- 'emptyDir'
- 'projected'
- 'secret'
- 'downwardAPI'
# Assume that persistentVolumes set up by the cluster admin are safe to use. - 'persistentVolumeClaim'
hostNetwork: false
hostIPC: false
hostPID: false runAsUser:
# Require the container to run without root privileges.
rule: 'MustRunAsNonRoot'
seLinux:
# This policy assumes the nodes are using AppArmor rather than SELinux. rule: 'RunAsAny'
supplementalGroups: rule: 'MustRunAs' ranges:
# Forbid adding the root group.
- min: 1 max: 65535 fsGroup:
rule: 'MustRunAs' ranges:
# Forbid adding the root group. - min: 1
max: 65535
readOnlyRootFilesystem: false
```

## 2.CORRECT TEXT

a. Retrieve the content of the existing secret named default-token-xxxxx in the testing namespace.

Store the value of the token in the token.txt

b. Create a new secret named test-db-secret in the DB namespace with the following content:

username: mysql

password: password@123

Create the Pod name test-db-pod of image nginx in the namespace db that can access test-db-secret via a volume at path /etc/mysql-credentials

### Answer:

To add a Kubernetes cluster to your project, group, or instance:

☞ Navigate to your:

☞ Click Add Kubernetes cluster.

☞ Click the Add existing cluster tab and fill in the details: Get the API URL by running this command:

```
kubectl cluster-info | grep-E'Kubernetes master|Kubernetes control plane'| awk'/http/ {print $NF}'
```

☞ uk.co.certification.simulator.questionpool.PList@dc67810

```
kubectl get secret <secret name>-ojsonpath="{['data']['ca.crt']}"
```

## 3.CORRECT TEXT

Fix all issues via configuration and restart the affected components to ensure the new setting takes effect.

Fix all of the following violations that were found against the API server:-

- ☞ a. Ensure the --authorization-mode argument includes RBAC
- ☞ b. Ensure the --authorization-mode argument includes Node
- ☞ c. Ensure that the --profiling argument is set to false

Fix all of the following violations that were found against the Kubelet:-

- ☞ a. Ensure the --anonymous-auth argument is set to false.
- ☞ b. Ensure that the --authorization-mode argument is set to Webhook.

Fix all of the following violations that were found against the ETCD:-

- a. Ensure that the --auto-tls argument is not set to true

Hint: Take the use of Tool Kube-Bench

**Answer:**

API server:

- ☞ Ensure the --authorization-mode argument includes RBAC

Turn on Role Based Access Control. Role Based Access Control (RBAC) allows fine-grained control over the operations that different entities can perform on different objects in the cluster. It is recommended to use the RBAC authorization mode. Fix - BuildtimeKubernetesapiVersion: v1

kind: Pod

metadata:

creationTimestamp: null

labels:

component: kube-apiserver

tier: control-plane

name: kube-apiserver

namespace: kube-system

spec:

containers:

-command:

+ - kube-apiserver

+ - --authorization-mode=RBAC,Node

image: gcr.io/google\_containers/kube-apiserver-amd64:v1.6.0

livenessProbe:

failureThreshold:8

httpGet:

host:127.0.0.1

path: /healthz

port:6443

scheme: HTTPS

initialDelaySeconds:15

timeoutSeconds:15

name: kube-apiserver-should-pass

resources:

requests:

cpu: 250m

volumeMounts:

```
-mountPath: /etc/kubernetes/
```

```
name: k8s
```

```
readOnly:true
```

```
-mountPath: /etc/ssl/certs
```

```
name: certs
```

```
-mountPath: /etc/pki
```

```
name: pki
```

```
hostNetwork:true
```

```
volumes:
```

```
-hostPath:
```

```
path: /etc/kubernetes
```

```
name: k8s
```

```
-hostPath:
```

```
path: /etc/ssl/certs
```

```
name: certs
```

```
-hostPath:
```

```
path: /etc/pki
```

```
name: pki
```

☞ Ensure the `--authorization-mode` argument includes Node

Remediation: Edit the API server pod specification file `/etc/kubernetes/manifests/kube-apiserver.yaml` on the master node and set the `--authorization-mode` parameter to a value that includes Node.

```
--authorization-mode=Node,RBAC
```

Audit:

```
/bin/ps -ef | grep kube-apiserver | grep -v grep
```

Expected result:

```
'Node,RBAC' has 'Node'
```

☞ Ensure that the `--profiling` argument is set to false

Remediation: Edit the API server pod specification file `/etc/kubernetes/manifests/kube-apiserver.yaml` on the master node and set the below parameter.

```
--profiling=false
```

Audit:

```
/bin/ps -ef | grep kube-apiserver | grep -v grep
```

Expected result:

```
'false' is equal to 'false'
```

Fix all of the following violations that were found against the Kubelet:-

☞ `uk.co.certification.simulator.questionpool.PList@d9af1b0`

Remediation: If using a Kubelet config file, edit the file to set `authentication: anonymous: enabled` to false.

If using executable arguments, edit the kubelet service file

`/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` on each worker node and set the below parameter in `KUBELET_SYSTEM_PODS_ARGS` variable.

```
--anonymous-auth=false
```

Based on your system, restart the kubelet service. For example:

```
systemctl daemon-reload
```

```
systemctl restart kubelet.service
```

Audit:

```
/bin/ps -fC kubelet
```

Audit Config:

```
/bin/cat /var/lib/kubelet/config.yaml
```

Expected result:

⇒ 'false' is equal to 'false'

2) Ensure that the --authorization-mode argument is set to Webhook.

Audit

```
docker inspect kubelet | jq -e '.[0].Args[] | match("--authorization-mode=Webhook").string'
```

Returned Value: --authorization-mode=Webhook

Fix all of the following violations that were found against the ETCD:-a. Ensure that the --auto-tls argument is not set to true

Do not use self-signed certificates for TLS. etcd is a highly-available key value store used by Kubernetes deployments for persistent storage of all of its REST API objects. These objects are sensitive in nature and should not be available to unauthenticated clients. You should enable the client authentication via valid certificates to secure the access to the etcd service.

Fix - BuildtimeKubernetesapiVersion: v1

kind: Pod

metadata:

annotations:

scheduler.alpha.kubernetes.io/critical-pod: ""

creationTimestamp: null

labels:

component: etcd

tier: control-plane

name: etcd

namespace: kube-system

spec:

containers:

-command:

+ - etcd

+ - --auto-tls=true

image: k8s.gcr.io/etcd-amd64:3.2.18

imagePullPolicy: IfNotPresent

livenessProbe:

exec:

command:

- /bin/sh

- -ec

- ETCDCTL\_API=3 etcdctl --endpoints=https://[192.168.22.9]:2379 --

cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt --

key=/etc/kubernetes/pki/etcd/healthcheck-client.key

get foo failureThreshold:8 initialDelaySeconds:15 timeoutSeconds:15 name: etcd-should-fail resources: {}

```
volumeMounts: -mountPath: /var/lib/etcd name: etcd-data
               -mountPath: /etc/kubernetes/pki/etcd
name: etcd-certs hostNetwork:true priorityClassName: system-cluster-critical volumes:
- hostPath:
  path: /var/lib/etcd
  type: DirectoryOrCreate
  name: etcd-data -hostPath:
  path: /etc/kubernetes/pki/etcd
  type: DirectoryOrCreate
  name: etcd-certs
status: {}
```

#### 4.CORRECT TEXT

On the Cluster worker node, enforce the prepared AppArmor profile

```
⌘ #include<tunables/global>
⌘ profile nginx-deny flags=(attach_disconnected) {
⌘ #include<abstractions/base>
⌘ file,
⌘ # Deny all file writes.
⌘ deny/** w,
⌘ }
⌘ EOF'
```

Edit the prepared manifest file to include the AppArmor profile.

```
⌘ apiVersion: v1
⌘ kind: Pod
⌘ metadata:
⌘ name: apparmor-pod
⌘ spec:
⌘ containers:
⌘ - name: apparmor-pod
⌘ image: nginx
```

Finally, apply the manifests files and create the Pod specified on it.

Verify: Try to make a file inside the directory which is restricted.

**Answer:** Send us your Feedback on this.

#### 5.CORRECT TEXT

Create a RuntimeClass named gvisor-rc using the prepared runtime handler named runsc.

Create a Pods of image Nginx in the Namespace server to run on the gVisor runtime class

**Answer:**

```
⌘ Install the Runtime Class for gVisor { # Step 1: Install a RuntimeClass
cat <<EOF | kubectl apply -f -
apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
```



```
name: gvisor  
handler: runsc  
EOF  
}
```

⇒ Create a Pod with the gVisor Runtime Class { # Step 2: Create a pod

```
cat <<EOF | kubectl apply -f -
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
name: nginx-gvisor
```

```
spec:
```

```
runtimeClassName: gvisor
```

```
containers:
```

```
- name: nginx image: nginx
```

```
EOF
```

```
}
```

⇒ Verify that the Pod is running { # Step 3: Get the pod

```
kubectl get pod nginx-gvisor -o wide
```

```
}
```