DUMPSARENA

Certified Kubernetes Security Specialist (CKS)

Linux Foundation CKS

Version Demo

Total Demo Questions: 9

Total Premium Questions: 99

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Cluster Setup	7
Topic 2, Cluster Hardening	50
Topic 3, System Hardening	7
Topic 4, Minimize Microservice Vulnerabilities	15
Topic 5, Supply Chain Security	7
Topic 6, Monitoring, Logging and Runtime Security	13
Total	99

QUESTION NO: 1 - (SIMULATION)

Documentation Upgrading kubeadm clusters

You must connect to the correct host . Failure to do so may result in a zero score.

[candidate@base] $ ssh cks000034

Context

The kubeadm provisioned cluster was recently upgraded, leaving one node on a slightly older version due to workload compatibility concerns.

Task

Upgrade the cluster node compute-0 to match the version of the control plane node.

Use a command like the following to connect to the compute node:

[candidate@cks000034] $ ssh compute-0

Do not modify any running workloads in the cluster.

Do not forget to exit from the compute node once you have completed your tasks:

[candidate@icompute-e] $ exit

ANSWER: See the explanation for the answer

Explanation:

Connect to the specified host first and determine the *exact* kubelet version of the control-plane node. Do **not** drain, delete, restart, scale, or otherwise alter workloads; a drain would evict running workload Pods and is not required for this task.

```
ssh cks000034
kubectl get nodes -o custom-
columns=NAME:.metadata.name,VERSION:.status.nodeInfo.kubeletVersion
```

Record the control-plane version, for example `v1.29.8`, then connect to the worker:

```
ssh compute-0
```

On `compute-0`, upgrade `kubeadm` to the package corresponding to that exact version. First inspect the package versions available in the configured repository:

```
sudo apt-get update
apt-cache madison kubeadm
```

Install the package whose version begins with the control-plane version (without the leading `v`). For an older kubeadm repository this commonly has the `-00` suffix; for example, if the control plane is `v1.29.8`:

```
sudo apt-mark unhold kubeadm 2>/dev/null || true
sudo apt-get install -y kubeadm=1.29.8-00
sudo apt-mark hold kubeadm
sudo kubeadm upgrade node
```

If `apt-cache madison` shows a newer package suffix instead (for example `1.29.8-1.1`), use that exact displayed package version rather than inventing `-00`.

Then install the matching `kubelet` version and restart it. Installing matching `kubectl` too is consistent with the kubeadm upgrade procedure when it is installed on the node:

```
sudo apt-mark unhold kubelet kubectl 2>/dev/null || true
sudo apt-get install -y kubelet=1.29.8-00 kubectl=1.29.8-00
sudo apt-mark hold kubelet kubectl
sudo systemctl daemon-reload
```

```
sudo systemctl restart kubelet
```

Replace `1.29.8-00` everywhere with the exact package version selected for the actual control-plane version. Finally, exit the compute node as requested and verify from the cluster host:

```
exit
kubectl get nodes -o custom-columns=NAME:.metadata.name,STATUS:.status.conditions[-1].type,VERSION:.status.nodeInfo.kubeletVersion
```

The `compute-0` kubelet version should now equal the control-plane node version. Do not run `kubectl drain` or `kubectl uncordon`, since this task explicitly requires leaving running workloads untouched.

QUESTION NO: 2 - (SIMULATION)

You can switch the cluster/configuration context using the following command:[desk@cli] $ kubectl config use-context dev Context:A CIS Benchmark tool was run against the kubeadm created cluster and found multiple issues that must be addressed.Task:Fix all issues via configuration and restart the affected components to ensure the new settings take effect.Fix all of the following violations that were found against the API server:1.2.7 authorization-mode argument is not set to AlwaysAllow FAIL1.2.8 authorization-mode argument includes Node FAIL1.2.7 authorization-mode argument includes RBAC FAILFix all of the following violations that were found against the Kubelet:4.2.1 Ensure that the anonymous-auth argument is set to false FAIL4.2.2 authorization-mode argument is not set to AlwaysAllow FAIL (Use Webhook autumn/authz where possible)Fix all of the following violations that were found against etcd:2.2 Ensure that the client-cert-auth argument is set to true

ANSWER: See the explanation for the answer

Explanation:

Use the `dev` context, then update the kubeadm static-Pod manifests on the control-plane node and the kubelet configuration on the affected node(s).

```
kubectl config use-context dev
ssh master1
```

On `master1`, edit `/etc/kubernetes/manifests/kube-apiserver.yaml`. In the API server container `command:` arguments, ensure there is exactly one authorization mode argument and that it is:

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

Remove `AlwaysAllow` if it appears in an existing `--authorization-mode` value. Do not add a second authorization-mode flag.

Also edit `/etc/kubernetes/manifests/etcd.yaml` and ensure the etcd command arguments contain:

```
- --client-cert-auth=true
```

Both are static Pod manifests. Saving these files causes the kubelet to automatically recreate the API server and etcd with the new arguments; verify after a short wait:

```
kubectl get pods -n kube-system
kubectl get --raw='/readyz?verbose'
```

On `worker1` (and on every node whose kubelet is in scope), edit `/var/lib/kubelet/config.yaml`. Set anonymous authentication to disabled and use webhook authorization rather than AlwaysAllow:

```
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true

authorization:
  mode: Webhook
```

```
  webhook:
    cacheAuthorizedTTL: 0s
    cacheUnauthorizedTTL: 0s
```

In particular, replace/remove any existing `authentication.anonymous.enabled: true` and `authorization.mode: AlwaysAllow` values; do not leave duplicate YAML keys. Retain the existing webhook authentication configuration when present, so the kubelet uses webhook authn/authz where available.

Restart the kubelet after saving its configuration:

```
sudo systemctl restart kubelet
sudo systemctl is-active kubelet
```

The resulting required settings are: API server `--authorization-mode=Node,RBAC`, etcd `--client-cert-auth=true`, and kubelet `anonymous.enabled: false` with `authorization.mode: Webhook`.

QUESTION NO: 3 - (SIMULATION)

Given an existing Pod named nginx-pod running in the namespace test-system, fetch the service-account-name used and put the content in /candidate/KSC00124.txt

Create a new Role named dev-test-role in the namespace test-system, which can perform update operations, on resources of type namespaces.

Create a new RoleBinding named dev-test-role-binding, which binds the newly created Role to the Pod ' s ServiceAccount (found in the Nginx pod running in namespace test-system).

ANSWER: See the explanation for the answer

Explanation:

Retrieve the ServiceAccount used by `nginx-pod`, save its name in the required file, then create the Role and RoleBinding:

```
SA=$(kubectl -n test-system get pod nginx-pod -o jsonpath='{.spec.serviceAccountName}')
printf '%s' "$SA" > /candidate/KSC00124.txt

kubectl -n test-system create role dev-test-role \
  --verb=update \
  --resource=namespaces

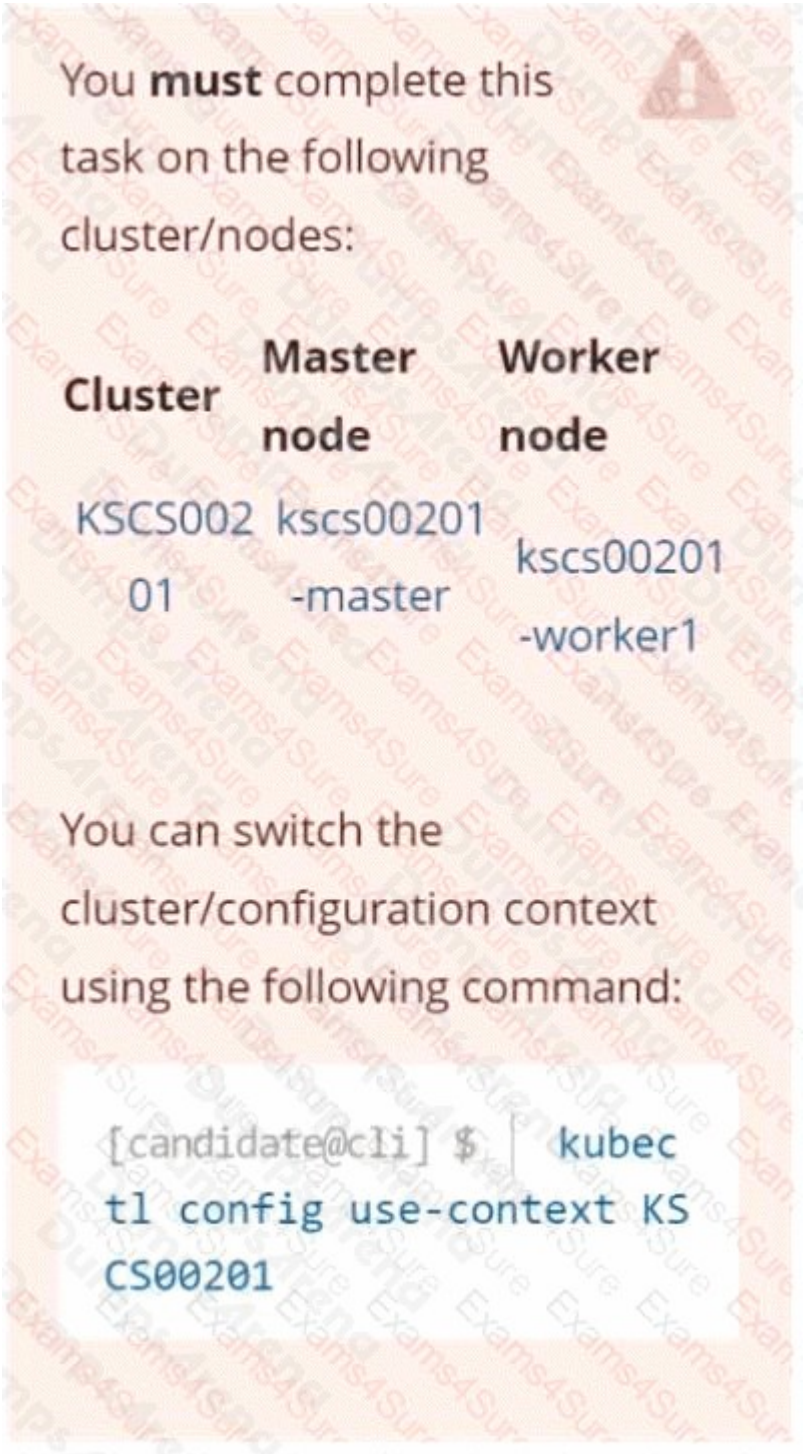
kubectl -n test-system create rolebinding dev-test-role-binding \
  --role=dev-test-role \
  --serviceaccount=test-system:"$SA"
```

The file `/candidate/KSC00124.txt` will contain only the ServiceAccount name assigned to the existing Pod. The RoleBinding uses that same ServiceAccount as its subject and references `dev-test-role`.

Optional verification:

```
cat /candidate/KSC00124.txt
kubectl -n test-system get role dev-test-role
kubectl -n test-system get rolebinding dev-test-role-binding -o yaml
```

QUESTION NO: 4 - (SIMULATION)



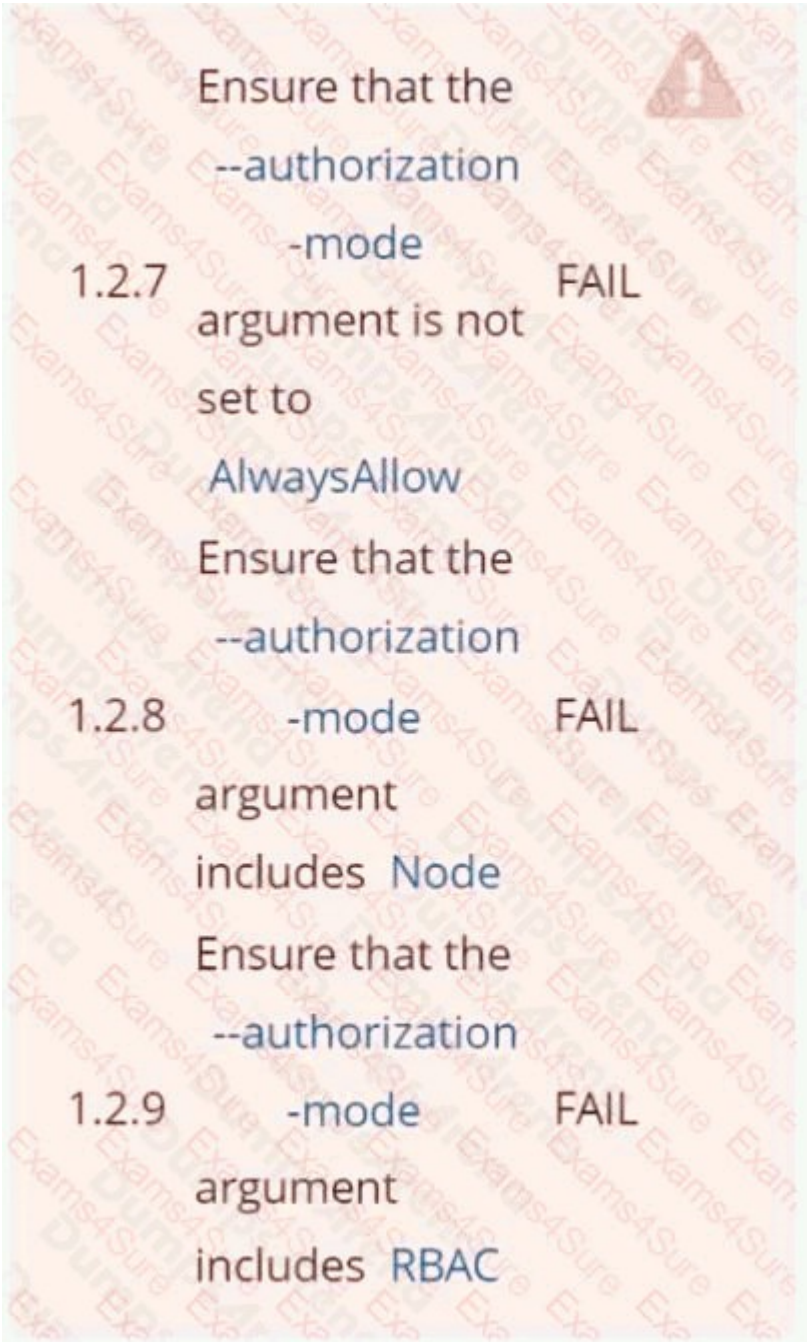
Context

A CIS Benchmark tool was run against the kubeadm-created cluster and found multiple issues that must be addressed immediately.

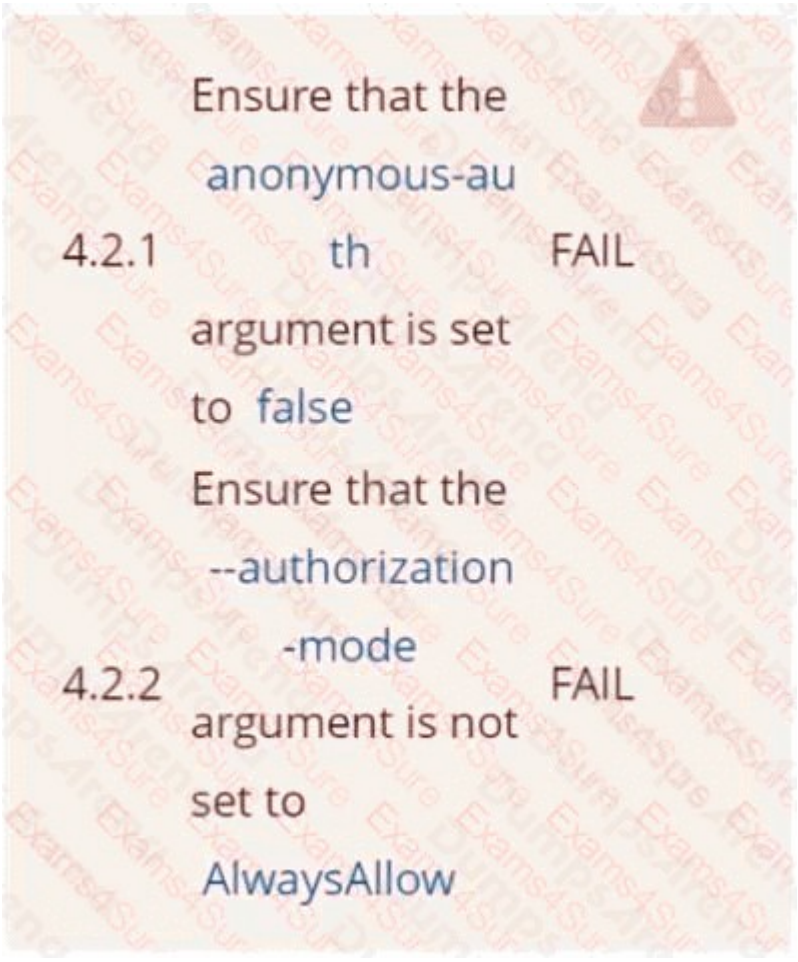
Task

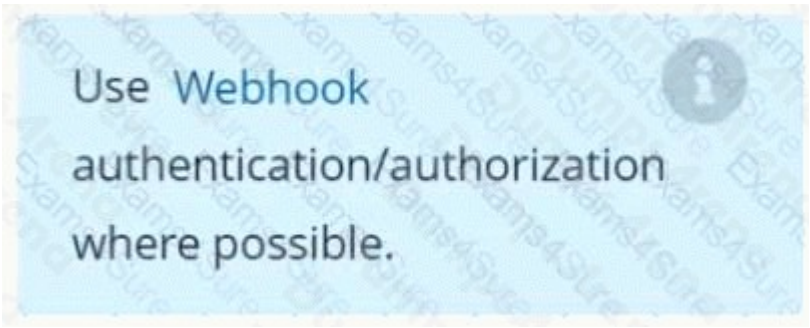
Fix all issues via configuration and restart the affected components to ensure the new settings take effect.

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

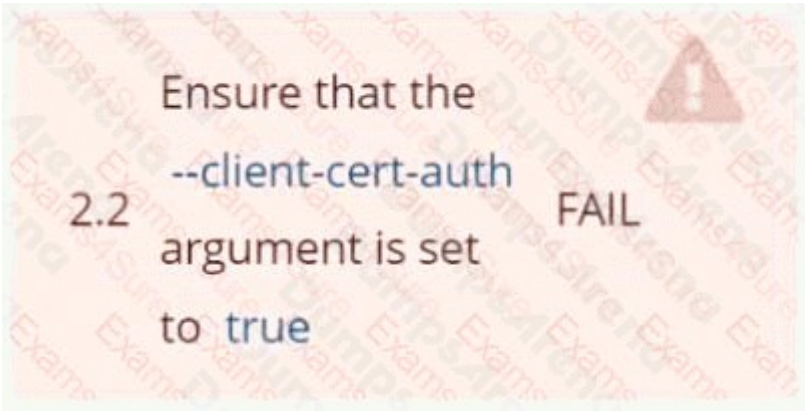


Fix all of the following violations that were found against the Kubelet:





Fix all of the following violations that were found against etcd:



ANSWER: See the explanation for the answer

Explanation:

Apply the fixes on the control-plane node `kscs00201-master`, and apply the kubelet configuration on **both** the master and worker node (`kscs00201-worker1`).

1. API server (master)

Edit `/etc/kubernetes/manifests/kube-apiserver.yaml`. In the API server container `command:` list, ensure there is exactly one authorization mode setting and that it is:

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

This removes/disallows `AlwaysAllow` and enables both the `Node` and `RBAC` authorization modes.

2. etcd (master)

Edit `/etc/kubernetes/manifests/etcd.yaml`. In the etcd container `command:` list, add or correct:

```
- --client-cert-auth=true
```

3. Kubelet (master and worker)

On `kscs00201-master` and `kscs00201-worker1`, edit `/var/lib/kubelet/config.yaml` and ensure the following kubelet configuration is present:

```
authentication:
  anonymous:
    enabled: false
authorization:
  mode: Webhook
```

If those sections already exist, modify their existing values rather than creating duplicate YAML keys. The resulting kubelet settings must be equivalent to `--anonymous-auth=false` and `--authorization-mode=Webhook`.

4. Restart/reload components

Restart kubelet on each node after saving its configuration:

```
sudo systemctl restart kubelet
sudo systemctl status kubelet --no-pager
```

The API server and etcd are static Pods. Saving changes to their manifests under `/etc/kubernetes/manifests` causes kubelet to automatically recreate them. If necessary, restarting kubelet on the master also forces immediate reconciliation:

```
sudo systemctl restart kubelet
```

Verify from the master after the control-plane Pods become ready:

```
kubectl -n kube-system get pods
ps -ef | grep -E 'kube-apiserver|etcd' | grep -E 'authorization-mode=Node,RBAC|client-cert-auth=true'
```

The required final values are: API server `--authorization-mode=Node,RBAC`, kubelet anonymous authentication disabled and authorization mode `Webhook`, and etcd `--client-cert-auth=true`.

QUESTION NO: 5 - (SIMULATION)

A container image scanner is set up on the cluster.

Given an incomplete configuration in the directory

/etc/kubernetes/confcontrol and a functional container image scanner with HTTPS endpoint https://test-server.local.8081/image_policy

1. Enable the admission plugin.

2. Validate the control configuration and change it to implicit deny.

Finally, test the configuration by deploying the pod having the image tag as latest.

ANSWER: See the explanation for the answer

Explanation:

Configure the API server to use `ImagePolicyWebhook` and set the webhook's failure/default behavior to deny. First inspect the supplied files, since the incomplete configuration should contain (or indicate) the required webhook kubeconfig path:

```
sudo ls -la /etc/kubernetes/confcontrol
sudo grep -R "defaultAllow\|kubeConfigFile\|test-server" /etc/kubernetes/confcontrol
```

In the ImagePolicyWebhook admission configuration file under `/etc/kubernetes/confcontrol`, ensure the ImagePolicyWebhook plugin references the supplied scanner kubeconfig and has `defaultAllow: false`. For example:

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    configuration:
      imagePolicy:
        kubeConfigFile: /etc/kubernetes/confcontrol/image-policy-webhook.kubeconfig
        allowTTL: 50
        denyTTL: 50
        retryBackoff: 500
        defaultAllow: false
```

The important required change is: `defaultAllow: false`. This is implicit deny: if the scanner cannot approve an image, or cannot be contacted, the image is rejected.

The referenced kubeconfig must point at the scanner endpoint (using the files/certificates supplied in the directory), for example:

```
clusters:
- cluster:
    server: https://test-server.local:8081/image_policy
  name: image-policy-scanner
```

Then edit the static API server manifest, normally `/etc/kubernetes/manifests/kube-apiserver.yaml`. Preserve all currently enabled plugins and append `ImagePolicyWebhook` to the existing admission plugin list. Also supply the admission configuration file:

```
- --enable-admission-plugins=NodeRestriction,...,ImagePolicyWebhook
- --admission-control-config-file=/etc/kubernetes/confcontrol/admission-configuration.yaml
```

Use the actual admission configuration filename present in `/etc/kubernetes/confcontrol`. Do not replace the existing plugin list; only add `ImagePolicyWebhook`. The kubelet will automatically restart the static API server after the manifest is saved.

Confirm that the API server has restarted and then deploy a pod using a `latest` image:

```
kubectl get --raw='/readyz?verbose'
kubectl run latest-image-test --image=nginx:latest
```

The pod creation should be rejected with a `Forbidden` / image-policy-webhook denial message because the scanner does not approve the `latest` tag. This denial confirms that the admission plugin is active and that the configuration is implicit deny.

QUESTION NO: 6 - (SIMULATION)

Use the kubesec docker images to scan the given YAML manifest, edit and apply the advised changes, and passed with a score of 4 points.

kubesec-test.yaml

apiVersion: v1

kind: Pod

metadata:

name: kubesec-demo

spec:

containers:

- name: kubesec-demo

image: gcr.io/google-samples/node-hello:1.0

securityContext:

readOnlyRootFilesystem: true

Hint: docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < kubesec-test.yaml

ANSWER: See the explanation for the answer

Explanation:

Scan the manifest first:

```
docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < kubesec-test.yaml
```

Keep the existing `readOnlyRootFilesystem: true` and add the remaining three positive security settings under the container `securityContext`. The resulting manifest scores **4** points:

```
apiVersion: v1
kind: Pod
metadata:
  name: kubesec-demo
spec:
  containers:
  - name: kubesec-demo
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
```

Rescan to confirm the score, then apply it:

```
docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < kubesec-test.yaml
kubectl apply -f kubesec-test.yaml
```

`readOnlyRootFilesystem: true`: +1
`runAsNonRoot: true`: +1
`allowPrivilegeEscalation: false`: +1
`capabilities.drop: [ALL]`: +1

QUESTION NO: 7 - (SIMULATION)

Documentation Deployment, Pod, Namespace

You must connect to the correct host . Failure to do so may result in a zero score.

[candidate@base] $ ssh cks000028

Context

You must update an existing Pod to ensure the immutability of its containers.

Task

Modify the existing Deployment named lamp-deployment, running in namespace lamp, so that its containers:

. run with user ID 20000

. use a read-only root filesystem

. forbid privilege escalation

The Deployment ' s manifest file con be found at /home/candidate/finer-sunbeam/lamp-deployment.yaml.

ANSWER: See the explanation for the answer

Explanation:

Connect to the required host and update the Deployment manifest:

```
ssh cks000028
vi /home/candidate/finer-sunbeam/lamp-deployment.yaml
```

Under `spec.template.spec.containers`, add the required **container-level** security context to every container in `lamp-deployment`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: lamp-deployment
  namespace: lamp
spec:
  template:
    spec:
      containers:
      - name: <existing-container-name>
        image: <existing-image>
        securityContext:
          runAsUser: 20000
          readOnlyRootFilesystem: true
          allowPrivilegeEscalation: false
```

Keep all existing fields (image, ports, environment, volumes, etc.) unchanged. If the Pod has more than one container, put the same three settings in each container's `securityContext`.

Apply the updated manifest and confirm rollout:

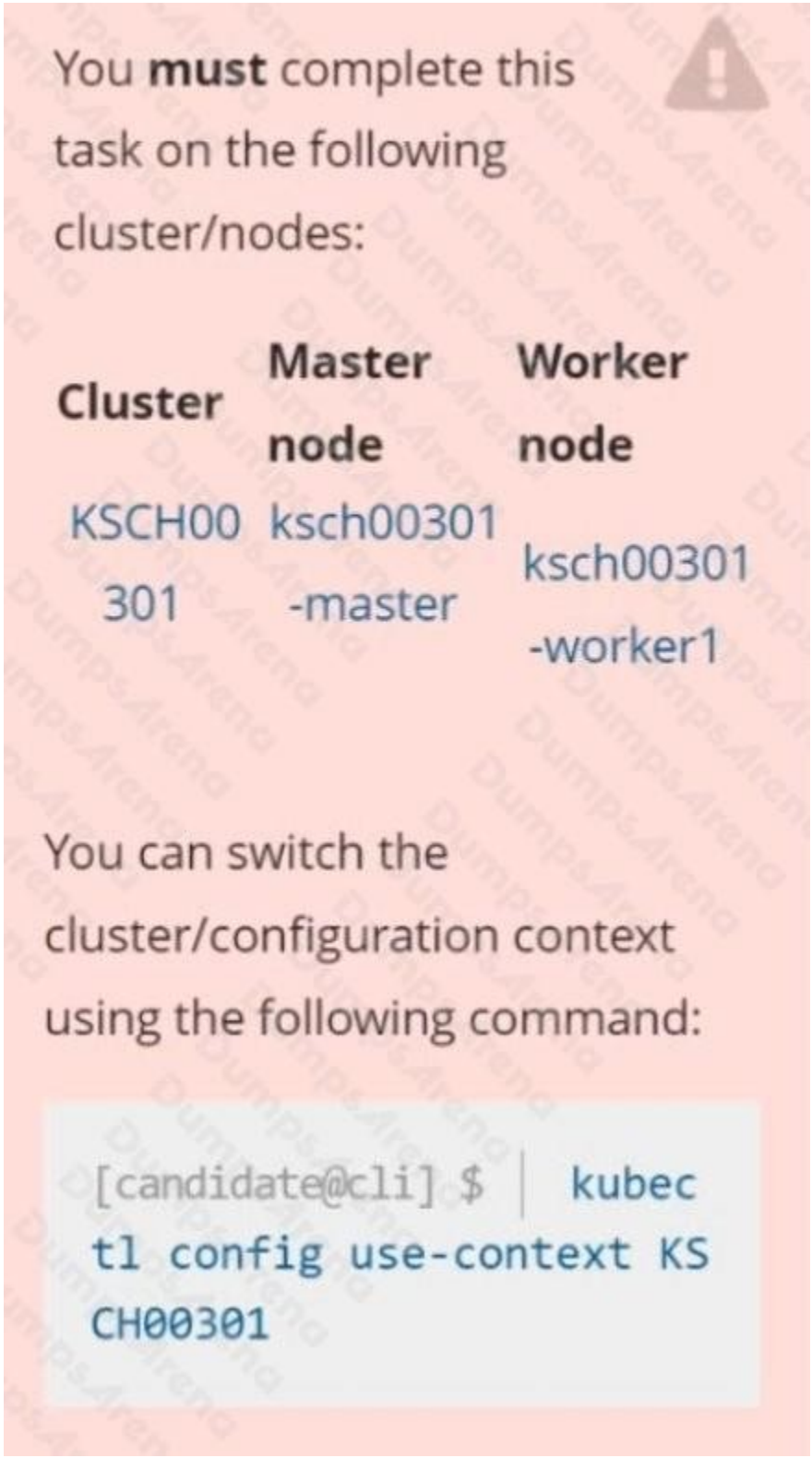
```
kubectl apply -f /home/candidate/finer-sunbeam/lamp-deployment.yaml
kubectl -n lamp rollout status deployment/lamp-deployment
```

Verify the effective template configuration:

```
kubectl -n lamp get deployment lamp-deployment \
  -o jsonpath='{range .spec.template.spec.containers[*]}{.name}{"
runAsUser="}{.securityContext.runAsUser}{"
readOnlyRootFilesystem="}{.securityContext.readOnlyRootFilesystem}{"
allowPrivilegeEscalation="}{.securityContext.allowPrivilegeEscalation}{"\n"}{end}'
```

The expected values for every container are `runAsUser=20000`, `readOnlyRootFilesystem=true`, and `allowPrivilegeEscalation=false`.

QUESTION NO: 8 - (SIMULATION)



Context

Your organization’s security policy includes:

The Pod specified in the manifest file /home/candidate/KSCH00301 /pod-manifest.yaml fails to schedule because of an incorrectly specified ServiceAccount.

Complete the following tasks:

Task

1. Create a new ServiceAccount named frontend-sa in the existing namespace qa. Ensure the ServiceAccount does not automount API credentials.

2. Using the manifest file at /home/candidate/KSCH00301 /pod-manifest.yaml, create the Pod.

3. Finally, clean up any unused ServiceAccounts in namespace qa.

ANSWER: See the explanation for the answer

Explanation:

Use the required cluster context, create the ServiceAccount with token automounting disabled, and then create the Pod from the supplied manifest:

```
kubectl config use-context KSCH00301

kubectl create serviceaccount frontend-sa --namespace=qa
kubectl patch serviceaccount frontend-sa --namespace=qa \
  --type=merge -p '{"automountServiceAccountToken":false}'

kubectl apply -f /home/candidate/KSCH00301/pod-manifest.yaml
```

Confirm that the ServiceAccount is configured correctly and that the Pod was created:

```
kubectl get serviceaccount frontend-sa -n qa -o yaml
kubectl get pods -n qa
```

The ServiceAccount must contain:

```
automountServiceAccountToken: false
```

Finally, inspect ServiceAccounts and remove every *non-default* account in `qa` which is not referenced by a Pod/workload. Do not remove the newly required `frontend-sa` or the Kubernetes `default` ServiceAccount.

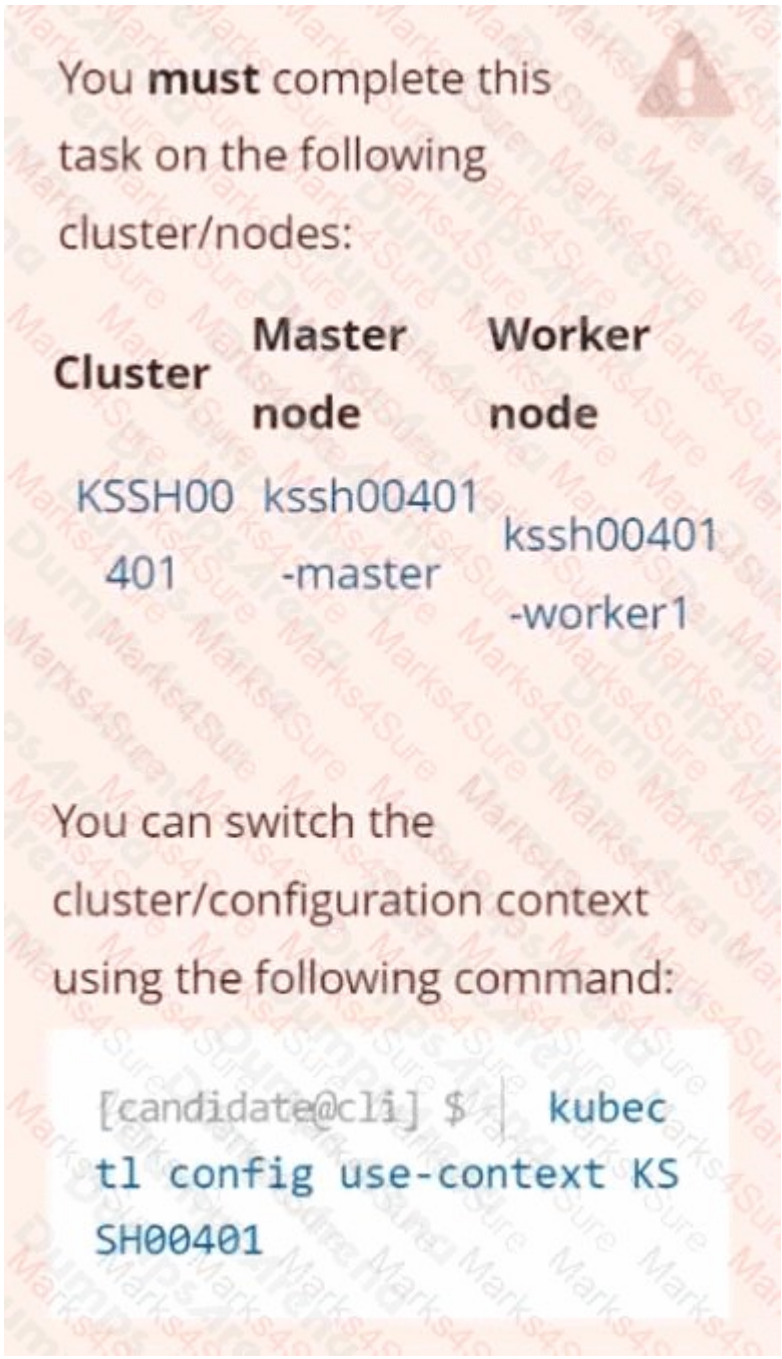
```
kubectl get serviceaccounts -n qa
kubectl get pods -n qa -o custom-
columns=POD:.metadata.name,SERVICEACCOUNT:.spec.serviceAccountName

# For each unused non-default ServiceAccount found:
kubectl delete serviceaccount <unused-serviceaccount-name> -n qa
```

For example, if the only obsolete account displayed is `old-frontend-sa`, remove it with:

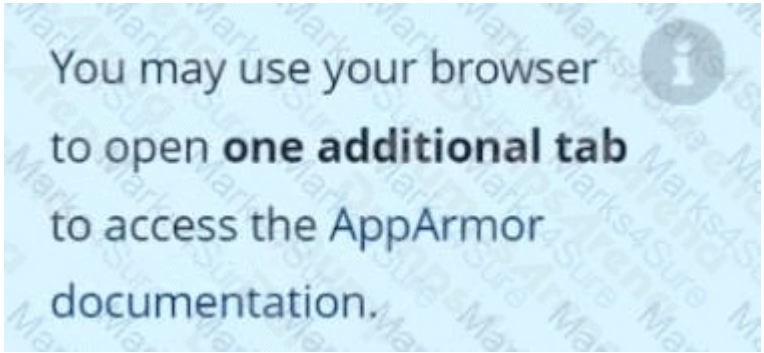
```
kubectl delete serviceaccount old-frontend-sa -n qa
```

QUESTION NO: 9 - (SIMULATION)



Context

AppArmor is enabled on the cluster ' s worker node. An AppArmor profile is prepared, but not enforced yet.



Task

On the cluster ' s worker node, enforce the prepared AppArmor profile located at /etc/apparmor.d/nginx_apparmor.

Edit the prepared manifest file located at /home/candidate/KSSH00401/nginx-pod.yaml to apply the AppArmor profile.

Finally, apply the manifest file and create the Pod specified in it.

ANSWER: See the explanation for the answer

Explanation:

On `kssh00401-worker1`, load/replace the local profile so that it is enforced:

```
sudo apparmor_parser --replace /etc/apparmor.d/nginx_apparmor
sudo aa-status | grep nginx_apparmor
```

Edit `/home/candidate/KSSH00401/nginx-pod.yaml`. Add the AppArmor annotation under the Pod `metadata.annotations`. The container-name part of the annotation must match the container name in the manifest (typically `nginx`):

```
metadata:
  annotations:
    container.apparmor.security.beta.kubernetes.io/nginx: localhost/nginx_apparmor
```

For example, this annotation directs the container named `nginx` to use the profile named `nginx_apparmor` that was loaded on the worker node. Do not use the filesystem path in the annotation; use `localhost/nginx_apparmor`.

Apply the prepared manifest:

```
kubectl apply -f /home/candidate/KSSH00401/nginx-pod.yaml
```

If the cluster version uses the current AppArmor API rather than the legacy annotation, the equivalent container security context is:

```
securityContext:
  appArmorProfile:
    type: Localhost
    localhostProfile: nginx_apparmor
```

Use the legacy annotation above when the supplied manifest/task environment follows the AppArmor tutorial's annotation-based format.