Kubernetes Security Scenarios for CKS

Practical, Exam-Style Labs

Author: DevOpsDynamo

2026 Edition

Disclaimer

This book is an independent resource created to help candidates prepare for the Certified Kubernetes Security Specialist (CKS®) exam. It is not affiliated with, endorsed by, or sponsored by The Linux Foundation or the Cloud Native Computing Foundation (CNCF®). Kubernetes®, Certified Kubernetes Administrator (CKA®), and Certified Kubernetes Security Specialist (CKS®) are registered trademarks of The Linux Foundation. All references to these terms are for informational and educational purposes only.

Contents

Chapter 1

Practical Security Scenarios

Each scenario in this chapter is written to match real CKS style tasks. You get:

- The exam style objective.
- A clear lab simulation so you can build the broken state yourself.
- A step by step solution flow that you can reuse in the exam.
- Verification and homework to lock in the pattern.

1.0.1 Scenario 1: Fixing Pod Failure Due to Incorrect ServiceAccount

Objective:

Identify and fix a pod that never gets created because it references a non existent ServiceAccount. Confirm that the pod starts successfully after you correct the configuration.

Exam Style Task (What The Question Looks Like)

You are connected to a cluster with context `cks-cluster`. In namespace `ops`, a Deployment named `analytics` is not creating any pods. The pods should use a ServiceAccount called `sa-metrics`, with ServiceAccount tokens not mounted by default.

Fix the issue so that one replica of `analytics` runs successfully in namespace `ops`.

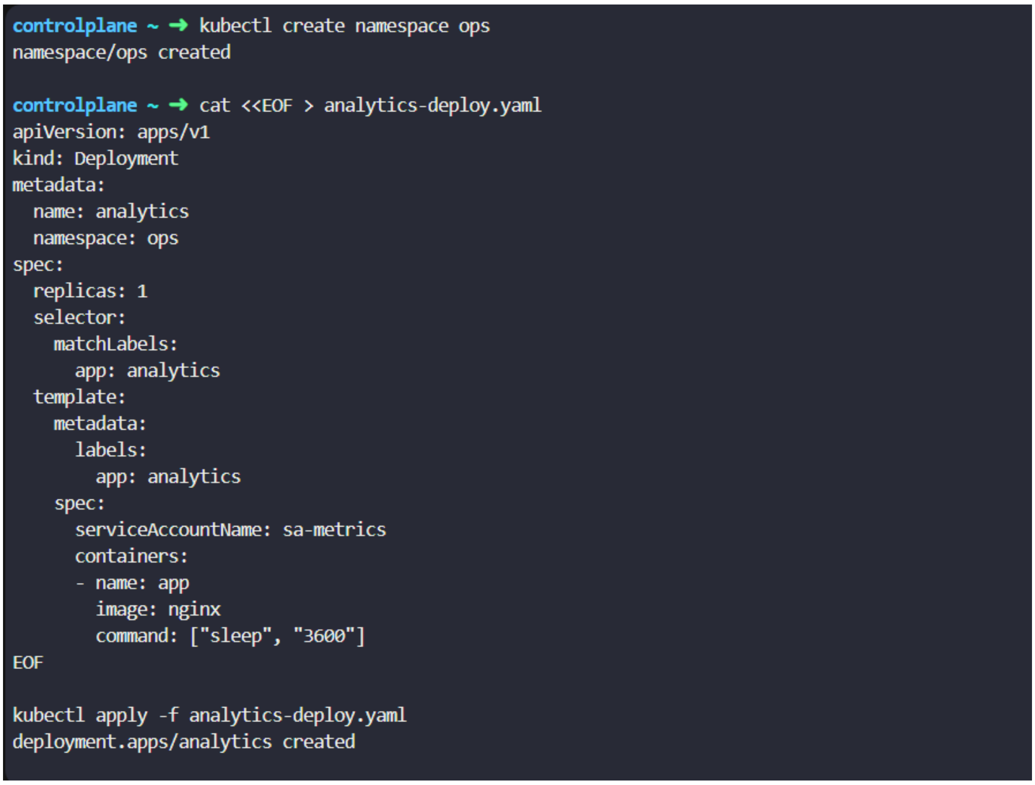
You can assume the namespace already exists in the exam.

Lab Preparation (Simulate The Broken State)

In your own practice lab, you may not have anything prepared. Use these steps to recreate the broken situation before you fix it.

Step 0: Create Namespace And Broken Deployment

```
kubectl create namespace ops

cat <<EOF > analytics-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: analytics
  namespace: ops
spec:
  replicas: 1
  selector:
    matchLabels:
      app: analytics
  template:
    metadata:
      labels:
        app: analytics
    spec:
      serviceAccountName: sa-metrics
      containers:
      - name: app
        image: nginx
        command: ["sleep", "3600"]
EOF

kubectl apply -f analytics-deploy.yaml
```

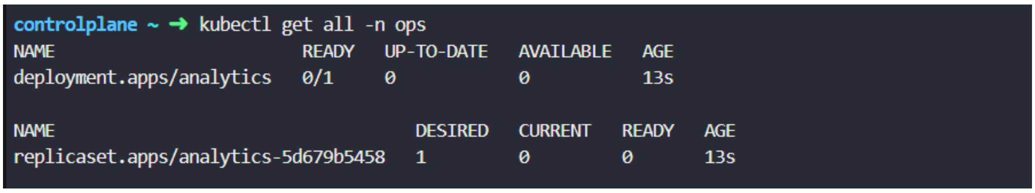


```
controlplane ~ ➜ kubectl create namespace ops
namespace/ops created

controlplane ~ ➜ cat <<EOF > analytics-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: analytics
  namespace: ops
spec:
  replicas: 1
  selector:
    matchLabels:
      app: analytics
  template:
    metadata:
      labels:
        app: analytics
    spec:
      serviceAccountName: sa-metrics
      containers:
      - name: app
        image: nginx
        command: ["sleep", "3600"]
EOF

kubectl apply -f analytics-deploy.yaml
deployment.apps/analytics created
```

Check the state:

```
kubectl get all -n ops
```

Expected:



```
controlplane ~ ➜ kubectl get all -n ops
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytics      0/1     0            0           13s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/analytics-5d679b5458   1         0         0       13s
```

```
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytics   0/1     0            0           1m

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/analytics-xxxxx      1         0         0       1m
```

No pods exist. The ReplicaSet wants 1 pod but `CURRENT` is 0 because the API server is rejecting pod creation.

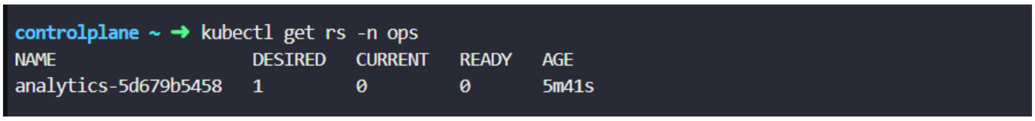
Step By Step Fix (Lab And Exam)

Step 1: Describe The ReplicaSet And Read Events

First, list the ReplicaSets in the namespace:

```
kubectl get rs -n ops
```

You should see something like:

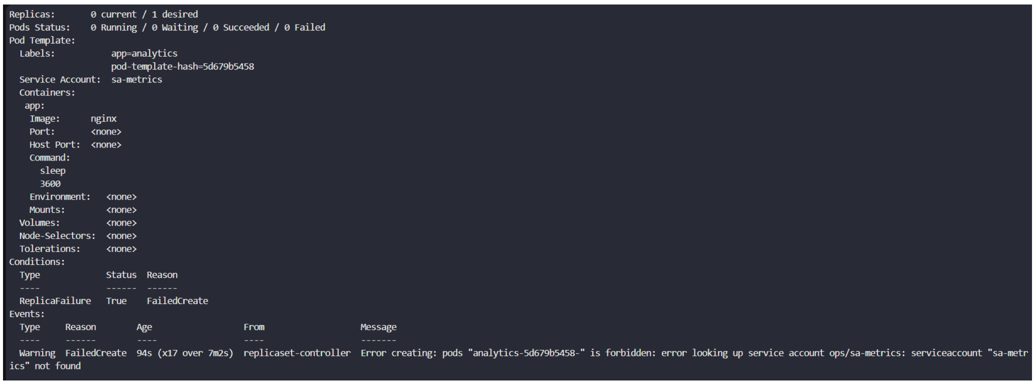


```
controlplane ~ ➜ kubectl get rs -n ops
NAME                     DESIRED   CURRENT   READY   AGE
analytics-5d679b5458     1         0         0       5m41s
```

```
NAME                     DESIRED   CURRENT   READY   AGE
analytics-5d679b5458     1         0         0       1m
```

Now describe that ReplicaSet and check the Events section:

```
kubectl describe rs analytics-5d679b5458 -n ops
```



```
Replicas:       0 current / 1 desired
Pods Status:    0 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:           app=analytics
                    pod-template-hash=5d679b5458
  Service Account:  sa-metrics
  Containers:
   app:
    Image:      nginx
    Port:       <none>
    Host Port:  <none>
    Command:
      sleep
      3600
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type             Status  Reason
  ----             ------  ------
  ReplicaFailure   True    FailedCreate
Events:
  Type     Reason        Age                 From                   Message
  ----     ------        ----                ----                   -------
  Warning  FailedCreate  94s (x17 over 7m2s)  replicaset-controller  Error creating: pods "analytics-5d679b5458-" is forbidden: error looking up service account ops/sa-metrics: serviceaccount "sa-metr
ics" not found
```

Expected event output:

```
Warning  FailedCreate  39s (x14 over 80s)  replicaset-controller  Error creating:
pods "analytics-5d679b5458-" is forbidden:
error looking up service account ops/sa-metrics:
serviceaccount "sa-metrics" not found
```

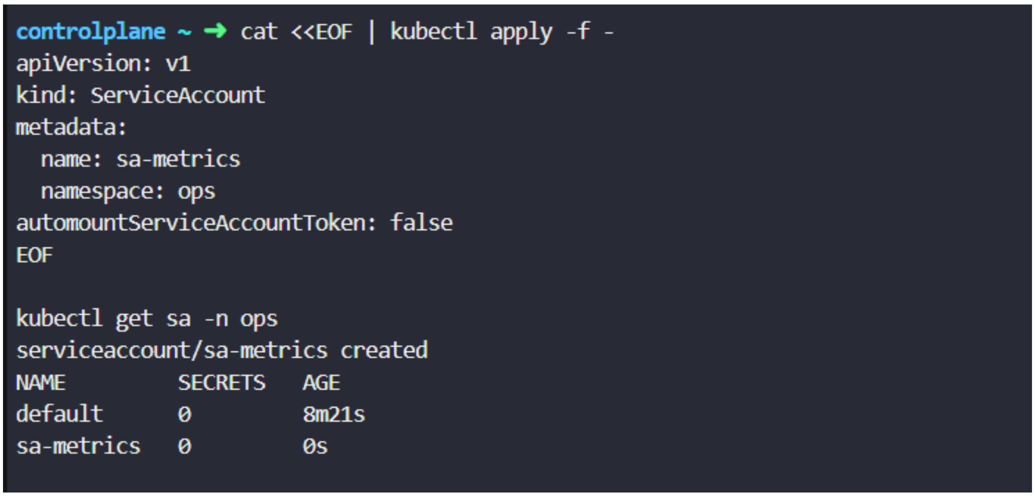
This tells you exactly what is wrong:

- Namespace `ops` exists.
- Deployment `analytics` exists and created a ReplicaSet.
- ServiceAccount `sa-metrics` does not exist in namespace `ops`.

Step 2: Create The ServiceAccount With Token Disabled

Create the ServiceAccount in namespace `ops`:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-metrics
  namespace: ops
automountServiceAccountToken: false
EOF

kubectl get sa -n ops
```



```
controlplane ~ ➜ cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-metrics
  namespace: ops
automountServiceAccountToken: false
EOF

kubectl get sa -n ops
serviceaccount/sa-metrics created
NAME         SECRETS   AGE
default      0         8m21s
sa-metrics   0         0s
```

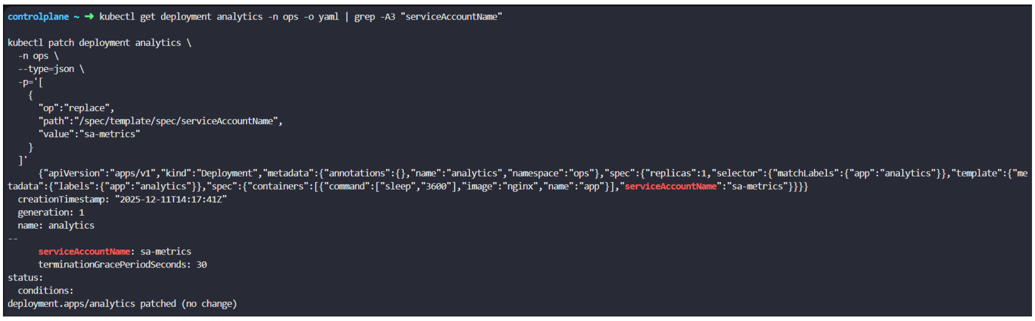
You should now see:

```
NAME         SECRETS   AGE
default      0         2m
sa-metrics   0         10s
```

Step 3: Confirm The Deployment Uses That ServiceAccount

In the exam, the spec usually already references the right ServiceAccount name. Still, verify and patch if needed:

```
kubectl get deployment analytics -n ops -o yaml | grep -A3 "serviceAccountName"

kubectl patch deployment analytics \
  -n ops \
  --type=json \
  -p='[
    {
      "op":"replace",
      "path":"/spec/template/spec/serviceAccountName",
      "value":"sa-metrics"
    }
  ]'
```

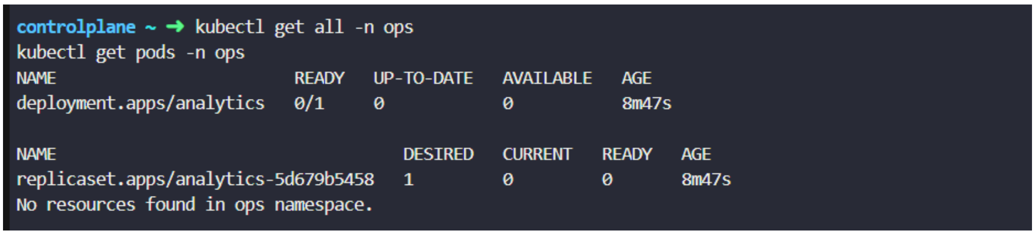


```
controlplane ~ ➜ kubectl get deployment analytics -n ops -o yaml | grep -A3 "serviceAccountName"

kubectl patch deployment analytics \
  -n ops \
  --type=json \
  -p='[
    {
      "op":"replace",
      "path":"/spec/template/spec/serviceAccountName",
      "value":"sa-metrics"
    }
  ]'
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"name":"analytics","namespace":"ops"},"spec":{"replicas":1,"selector":{"matchLabels":{"app":"analytics"}},"template":{"metadata":{"labels":{"app":"analytics"}},"spec":{"containers":[{"command":["sleep","3600"],"image":"nginx","name":"app"}],"serviceAccountName":"sa-metrics"}}}}
  creationTimestamp: "2025-12-11T14:17:41Z"
  generation: 1
  name: analytics
--
      serviceAccountName: sa-metrics
      terminationGracePeriodSeconds: 30
status:
  conditions:
deployment.apps/analytics patched (no change)
```

If the patch says `patched (no change)`, that is fine. It means the field already had the correct value.

Step 4: Check ReplicaSet And Pods

```
kubectl get all -n ops
kubectl get pods -n ops
```



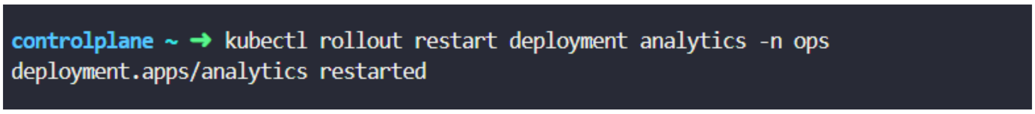
```
controlplane ~ ➜ kubectl get all -n ops
kubectl get pods -n ops
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytics  0/1     0            0           8m47s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/analytics-5d679b5458  1         0         0       8m47s
No resources found in ops namespace.
```

If the original ReplicaSet was created before the ServiceAccount existed, it may still show `CURRENT 0` and there may still be no pods. The Events will still show the old errors.

Step 5: Restart The Rollout To Get A Fresh ReplicaSet

To force a new ReplicaSet that uses the now valid configuration:

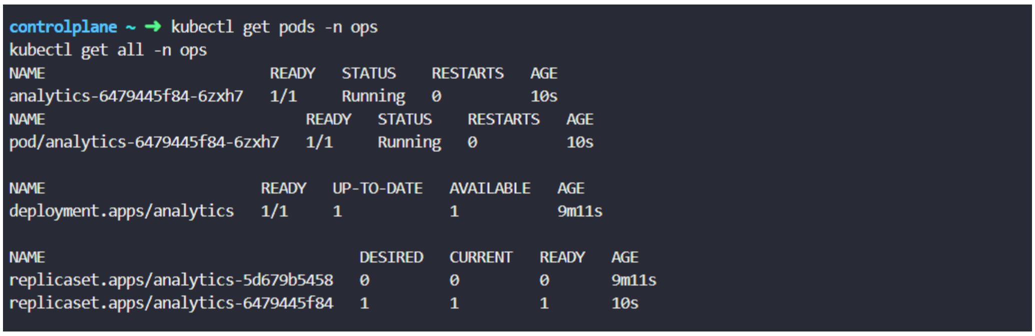
```
kubectl rollout restart deployment analytics -n ops
```



```
controlplane ~ ➜ kubectl rollout restart deployment analytics -n ops
deployment.apps/analytics restarted
```

Then verify:

```
kubectl get pods -n ops
kubectl get all -n ops
```



```
controlplane ~ ➜ kubectl get pods -n ops
kubectl get all -n ops
NAME                        READY   STATUS    RESTARTS   AGE
analytics-6479445f84-6zxh7  1/1     Running   0          10s
NAME                            READY   STATUS    RESTARTS   AGE
pod/analytics-6479445f84-6zxh7  1/1     Running   0          10s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytics  1/1     1            1           9m11s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/analytics-5d679b5458  0         0         0       9m11s
replicaset.apps/analytics-6479445f84  1         1         1       10s
```

Expected:

```
NAME                           READY   STATUS    RESTARTS   AGE
pod/analytics-b878fdbcf-clqcf  1/1     Running   0          30s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytics   1/1     1            1           33m

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/analytics-5d679b5458   0         0         0       33m
replicaset.apps/analytics-b878fdbcf    1         1         1       30s
```

The old ReplicaSet remains for history, but the new one has one running pod.

Step 6: Final Sanity Check

```
kubectl describe pod -n ops \
  $(kubectl get pod -n ops -l app=analytics -o jsonpath='{.items[0].metadata.name}')
```

Check that:

- `Service Account:  sa-metrics`
- Pod status is `Running`

Exam Tip

Exam Tip

- When a Deployment shows `DESIRED 1`, `CURRENT 0`, it usually means the API server is rejecting pod creation.
- Always run `kubectl describe deployment` or `kubectl describe rs` and read the Events. Do not guess.
- If the only error is a missing ServiceAccount, create it with the right namespace and name.
- If the spec is now correct but still no pods are created, use `kubectl rollout restart deployment <name>` to get a clean ReplicaSet.

Homework

Rebuild this scenario on your own without looking at the steps and answer these on a blank terminal:

1. A Deployment in namespace `team-a` uses ServiceAccount `sa-logs` which does not exist. Fix it so that its pods run and do not auto mount tokens.
2. Repeat the same task, but this time change the Deployment so it uses the default ServiceAccount again.
3. Time yourself. Aim to get from first `kubectl get` to a running pod in under 90 seconds.

1.0.2 Scenario 2: Fixing CIS Benchmark Violations On The API Server

Objective:
Interpret a CIS Kubernetes Benchmark report, identify why the API server is marked as non compliant, and correct the configuration in the static pod manifest. Confirm that the API server restarts successfully and that the secure configuration is active.

Understanding CIS Benchmarks

The Center for Internet Security (CIS) publishes a widely used Kubernetes hardening guide. It provides specific tests that evaluate:

- API server flags and permissions
- etcd configuration
- kubelet security
- network policy enforcement
- RBAC usage

A typical Kubernetes environment is scanned using tools such as:

- kube-bench
- kubescape
- Aqua Trivy Kubernetes Benchmark
- OpenSCAP profiles

These tools compare cluster configurations against the CIS profile and produce a pass or fail result.

Sample CIS Benchmark Failure Report

Below is a realistic report excerpt from a CIS audit detecting an insecure API server configuration:

CIS Benchmark Finding

```
ID: 1.2.7
Title: Ensure that the --authorization-mode argument is not set to AlwaysAllow
Severity: High

Description:
The API server must enforce Node and RBAC authorization.
Using AlwaysAllow disables all authorization checks and permits unrestricted
access.

Remediation:
Edit the API server manifest and set:
    --authorization-mode=Node,RBAC

Detected Configuration:
    --authorization-mode=AlwaysAllow
```

What This Means:

- Anyone who can reach the API server can perform any action.
- RBAC permissions, roles, and bindings are completely bypassed.
- kubelets are not restricted to their own nodes.
- Cluster compromise is trivial; this is a critical severity issue.

In short: **The cluster is effectively running without access control.**

This is why CIS marks this as a High severity violation.

How To Respond (Exam Mindset)

In the exam:

- The scan already tells you exactly what is wrong.
- The fix always lives inside `/etc/kubernetes/manifests/kube-apiserver.yaml`.
- You do not troubleshoot connectivity or logs unless the API server fails to come back.
- The only correct fix is to edit the flag in place.
- The kubelet restarts the pod automatically after you save the file.

Do not delete the pod. Do not restart the kubelet. Do not patch anything through kubectl. The static pod manifest is the single source of truth.

How To Respond (Real World Security Operations)

A real cloud security engineer would:

1. Review the finding and confirm if it is accurate.
2. Check when the configuration was changed and by whom.
3. Escalate immediately due to severity (AlwaysAllow is catastrophic).
4. Fix the API server manifest.

5. Monitor the restart and ensure API availability.
6. Re-run the CIS scan to confirm remediation.
7. Document the incident and apply change controls to prevent recurrence.

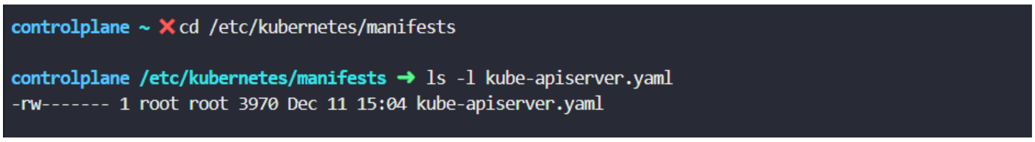
This mindset prepares candidates for both production realities and exam scenarios.

Lab Preparation (Simulate The Misconfiguration)

To practice this scenario, intentionally break your lab cluster.

Step 0: Connect To The Control Plane Node

```
ssh controlplane
cd /etc/kubernetes/manifests
ls -l kube-apiserver.yaml
```



```
controlplane ~ ✗ cd /etc/kubernetes/manifests

controlplane /etc/kubernetes/manifests ➜ ls -l kube-apiserver.yaml
-rw------- 1 root root 3970 Dec 11 15:04 kube-apiserver.yaml
```

Step 1: Insert An Insecure Authorization Mode

```
sudo vi kube-apiserver.yaml
```

Modify the command section so that it contains:

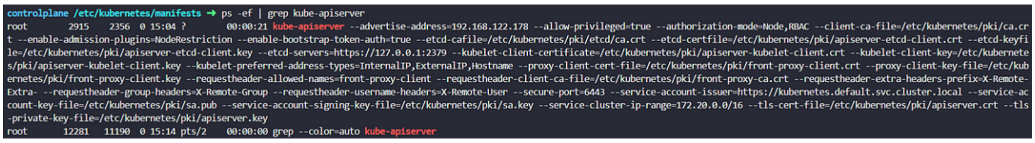
```
- --authorization-mode=AlwaysAllow
```

Save and exit. The kubelet restarts the API server using this insecure mode. You now have the same broken condition the CIS report described.

Step By Step Fix (Lab And Exam)

Step 1: Confirm The Insecure Configuration

```
ps -ef | grep kube-apiserver
```



```
controlplane /etc/kubernetes/manifests ➜ ps -ef | grep kube-apiserver
root        2915    2356  0 15:04 ?        00:00:21 kube-apiserver --advertise-address=192.168.122.178 --allow-privileged=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.cr
t --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-keyfi
le=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubelet-client-key=/etc/kubernete
s/pki/apiserver-kubelet-client.key --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-file=/etc/kub
ernetes/pki/front-proxy-client.key --requestheader-allowed-names=front-proxy-client --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-prefix=X-Remote-
Extra- --requestheader-group-headers=X-Remote-Group --requestheader-username-headers=X-Remote-User --secure-port=6443 --service-account-issuer=https://kubernetes.default.svc.cluster.local --service-ac
count-key-file=/etc/kubernetes/pki/sa.pub --service-account-signing-key-file=/etc/kubernetes/pki/sa.key --service-cluster-ip-range=172.20.0.0/16 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls
-private-key-file=/etc/kubernetes/pki/apiserver.key
root       12281   11190  0 15:14 pts/2    00:00:00 grep --color=auto kube-apiserver
```

Expected:

```
--authorization-mode=AlwaysAllow
```

Step 2: Edit The Static Pod Manifest

```
cd /etc/kubernetes/manifests
sudo vi kube-apiserver.yaml
```

Replace the insecure flag:

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

Save and exit.

Step 3: Watch The API Server Restart

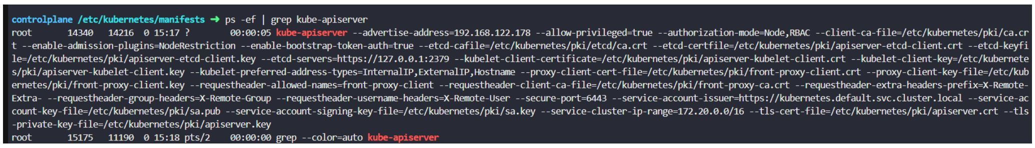
```
watch "kubectl get pods -n kube-system | grep apiserver"
```

Wait for:

- old apiserver pod terminating
- new apiserver pod creating
- new pod Running

Step 4: Confirm The Secure Configuration

```
ps -ef | grep kube-apiserver
```



```
controlplane /etc/kubernetes/manifests ➜ ps -ef | grep kube-apiserver
root       14340   14216  0 15:17 ?        00:00:05 kube-apiserver --advertise-address=192.168.122.178 --allow-privileged=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.cr
t --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-keyfi
le=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubelet-client-key=/etc/kubernete
s/pki/apiserver-kubelet-client.key --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-file=/etc/kub
ernetes/pki/front-proxy-client.key --requestheader-allowed-names=front-proxy-client --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-prefix=X-Remote-
Extra- --requestheader-group-headers=X-Remote-Group --requestheader-username-headers=X-Remote-User --secure-port=6443 --service-account-issuer=https://kubernetes.default.svc.cluster.local --service-ac
count-key-file=/etc/kubernetes/pki/sa.pub --service-account-signing-key-file=/etc/kubernetes/pki/sa.key --service-cluster-ip-range=172.20.0.0/16 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls
-private-key-file=/etc/kubernetes/pki/apiserver.key
root       15175   11190  0 15:18 pts/2    00:00:00 grep --color=auto kube-apiserver
```

Look for:

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

Expected Outcome

- API server restarts cleanly.
- CIS scan no longer reports violation 1.2.7.
- RBAC and Node authorization are now enforced cluster wide.

Exam Tip

Exam Tip

- Every CIS API server fix happens in the static pod manifest, not with kubectl edits.
- API server failing to restart usually means indentation or YAML formatting is wrong.
- Always verify with `ps -ef`. It is the clearest way to confirm active flags.
- If the apiserver pod never appears, recheck the manifest for syntax mistakes.

Homework

1. Change the API server to use an invalid flag. Observe how the component fails and fix it.
2. Run a tool such as `kube-bench` and identify two additional API server findings. Apply fixes and rerun the scan.
3. Practice correcting flags such as:
   - `-anonymous-auth=false`
   - `-profiling=false`
   - `-audit-log-path=/var/log/apiserver/audit.log`

1.0.3 Scenario 3: Enforcing Secure Authorization Modes For API Server And Kubelet

Note: This scenario builds on Scenario 2 by extending CIS compliance to the Kubelet. You will secure both the API server and the Kubelet to ensure proper authentication and authorization controls.

Understanding The CIS Findings

The CIS Kubernetes Benchmark includes several controls focused on authorization. A failure in any of these controls exposes the cluster to unauthorized access, privilege escalation, or node level compromise.

A real CIS scan may produce a report similar to this.

CIS Benchmark Summary

```
ID: 1.2.7
Title: Ensure that the --authorization-mode argument is not set to AlwaysAllow
Severity: High
Finding: FAIL
Detected: --authorization-mode=AlwaysAllow

ID: 4.2.1
Title: Ensure that the Kubelet anonymous-auth argument is set to false
Severity: High
Finding: FAIL
Detected: anonymous-auth: true

ID: 4.2.2
Title: Ensure that the Kubelet authorization-mode is set to Webhook
Severity: High
Finding: FAIL
Detected: authorization-mode: AlwaysAllow

Recommended Remediation:
  API Server: --authorization-mode=Node,RBAC
  Kubelet: anonymous-auth: false
  Kubelet: authorization-mode: Webhook
```

Interpretation:

- The API server is running without access control.
- The Kubelet is accepting unauthenticated requests.
- The Kubelet is not enforcing authorization decisions from the API server.

This is a critical security risk because it bypasses Kubernetes RBAC entirely and exposes node level APIs.

How To Respond (Exam Mindset)

- API server fixes always happen in static pod manifests.
- Kubelet fixes always happen in `/var/lib/kubelet/config.yaml`.
- You do not apply kubectl patches for these components.
- You must verify changes using `ps -ef`.

How To Respond (Real World Security Operations)

A production remediation workflow would include:

1. Reviewing the CIS report and confirming the flags.
2. Applying configuration changes in version-controlled manifests.
3. Restarting components during an approved maintenance window.
4. Re-running CIS scans to validate compliance.
5. Documenting the incident and updating compliance baselines.

This scenario helps build both exam agility and operational thinking.

Lab Preparation (Simulate Misconfiguration)

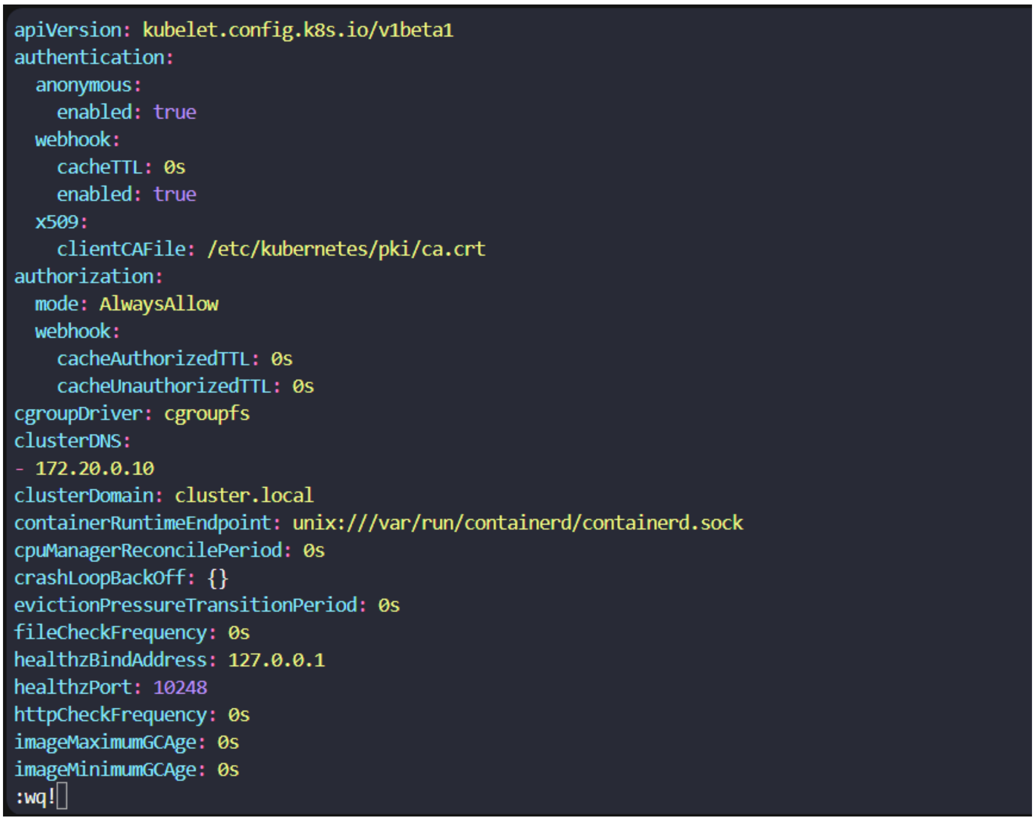
To recreate the CIS violations, intentionally weaken both the API server and Kubelet.

Step 0: Introduce Insecure API Server Flags

```
ssh controlplane
cd /etc/kubernetes/manifests
sudo vi kube-apiserver.yaml
```

Modify the command section to include the insecure line:

```
- --authorization-mode=AlwaysAllow
```



```
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: true
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: AlwaysAllow
  webhook:
    cacheAuthorizedTTL: 0s
    cacheUnauthorizedTTL: 0s
cgroupDriver: cgroupfs
clusterDNS:
- 172.20.0.10
clusterDomain: cluster.local
containerRuntimeEndpoint: unix:///var/run/containerd/containerd.sock
cpuManagerReconcilePeriod: 0s
crashLoopBackOff: {}
evictionPressureTransitionPeriod: 0s
fileCheckFrequency: 0s
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0s
imageMaximumGCAge: 0s
imageMinimumGCAge: 0s
:wq!
```

Save and exit. The kubelet will restart the API server using this insecure mode.

Step 1: Introduce Insecure Kubelet Settings

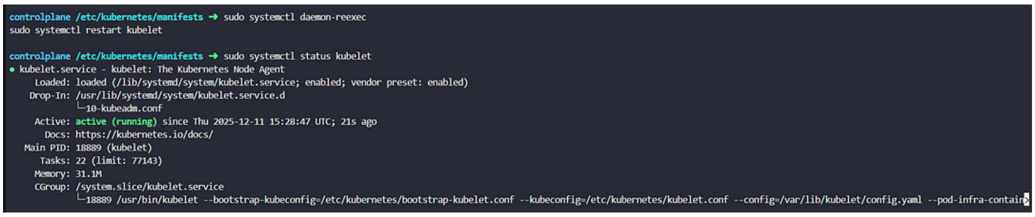
```
sudo vi /var/lib/kubelet/config.yaml
```

Insert or modify these insecure values:

```
authentication:
  anonymous:
    enabled: true

authorization:
  mode: AlwaysAllow
```

Restart the Kubelet:

```
sudo systemctl daemon-reexec
sudo systemctl restart kubelet
```



```
controlplane /etc/kubernetes/manifests ➜ sudo systemctl daemon-reexec
sudo systemctl restart kubelet

controlplane /etc/kubernetes/manifests ➜ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Thu 2025-12-11 15:28:47 UTC; 21s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 18889 (kubelet)
      Tasks: 22 (limit: 77143)
     Memory: 31.1M
     CGroup: /system.slice/kubelet.service
             └─18889 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --pod-infra-contain>
```

Your lab now matches the misconfiguration detected in the CIS report.

Step By Step Fix (Lab And Exam)

Step 1: Correct The API Server Configuration

Navigate to the static pod manifest directory:

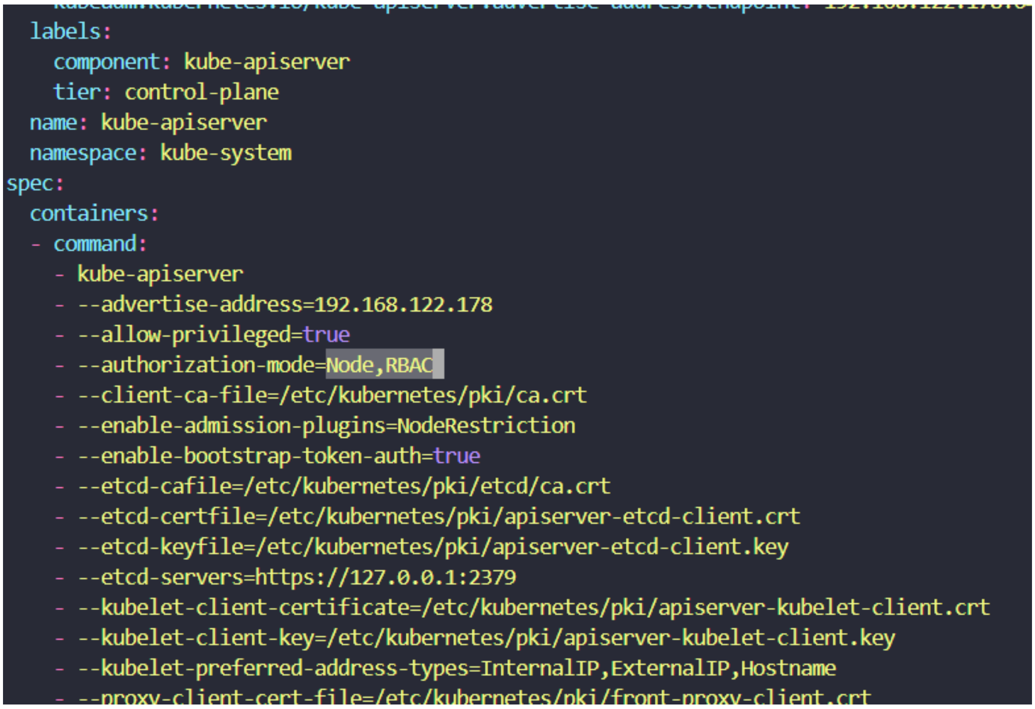
```
cd /etc/kubernetes/manifests
sudo vi kube-apiserver.yaml
```

Locate the insecure flag:

```
- --authorization-mode=AlwaysAllow
```

Replace it with the secure CIS compliant value:

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

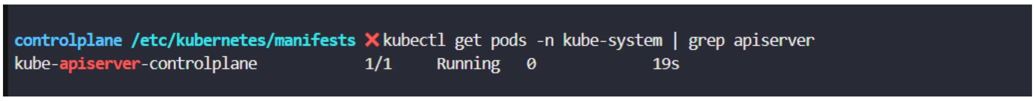


```
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.122.178:6
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.122.178
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --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
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
```

Save and exit. The kubelet will restart the API server automatically.

Verify the restart:

```
kubectl get pods -n kube-system | grep apiserver
```



```
controlplane /etc/kubernetes/manifests ✗kubectl get pods -n kube-system | grep apiserver
kube-apiserver-controlplane           1/1     Running   0             19s
```

Step 2: Correct The Kubelet Configuration

Open the Kubelet configuration file:

```
sudo vi /var/lib/kubelet/config.yaml
```

Apply the secure CIS recommended configuration:

```
authentication:
  anonymous:
    enabled: false

authorization:
  mode: Webhook
```

Save and exit.

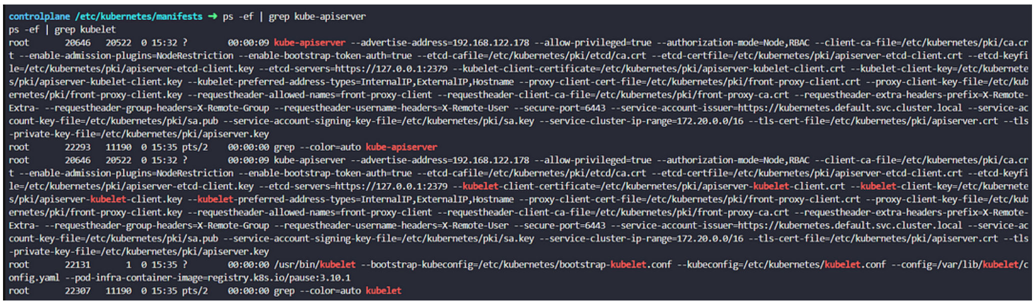
Restart the Kubelet so it loads the updated config:

```
sudo systemctl daemon-reexec
sudo systemctl restart kubelet
```

Validation

Verify both binaries are running with secure flags:

```
ps -ef | grep kube-apiserver
ps -ef | grep kubelet
```



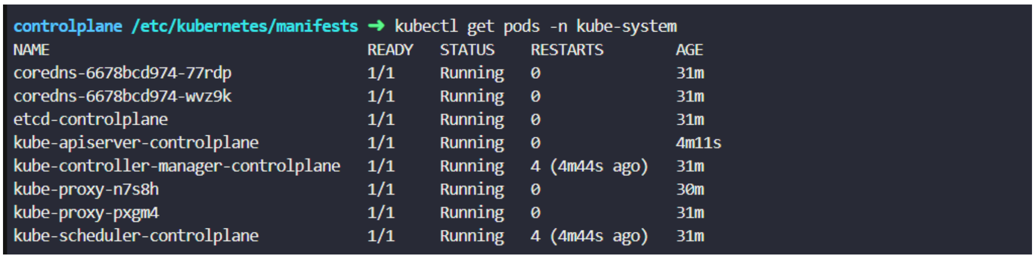
```
controlplane /etc/kubernetes/manifests ➜ ps -ef | grep kube-apiserver
ps -ef | grep kubelet
root       20646   20522  0 15:32 ?        00:00:09 kube-apiserver --advertise-address=192.168.122.178 --allow-privileged=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.cr
t --enable-admission-plugins=NodeRestriction --enable-bootstrap-token-auth=true --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt --etcd-keyfi
le=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubelet-client-key=/etc/kubernete
s/pki/apiserver-kubelet-client.key --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-file=/etc/kub
ernetes/pki/front-proxy-client.key --requestheader-allowed-names=front-proxy-client --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-prefix=X-Remote-
Extra- --requestheader-group-headers=X-Remote-Group --requestheader-username-headers=X-Remote-User --secure-port=6443 --service-account-issuer=https://kubernetes.default.svc.cluster.local --service-ac
count-key-file=/etc/kubernetes/pki/sa.pub --service-account-signing-key-file=/etc/kubernetes/pki/sa.key --service-cluster-ip-range=172.20.0.0/16 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls
-private-key-file=/etc/kubernetes/pki/apiserver.key
root       22293   11190  0 15:35 pts/2    00:00:00 grep --color=auto kube-apiserver
root       20646   20522  0 15:32 ?        00:00:09 kube-apiserver --advertise-address=192.168.122.178 --allow-privileged=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.cr
t --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-keyfi
le=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubelet-client-key=/etc/kubernete
s/pki/apiserver-kubelet-client.key --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-file=/etc/kub
ernetes/pki/front-proxy-client.key --requestheader-allowed-names=front-proxy-client --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-prefix=X-Remote-
Extra- --requestheader-group-headers=X-Remote-Group --requestheader-username-headers=X-Remote-User --secure-port=6443 --service-account-issuer=https://kubernetes.default.svc.cluster.local --service-ac
count-key-file=/etc/kubernetes/pki/sa.pub --service-account-signing-key-file=/etc/kubernetes/pki/sa.key --service-cluster-ip-range=172.20.0.0/16 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls
-private-key-file=/etc/kubernetes/pki/apiserver.key
root       22131       1  0 15:35 ?        00:00:00 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/c
onfig.yaml --pod-infra-container-image=registry.k8s.io/pause:3.10.1
root       22307   11190  0 15:35 pts/2    00:00:00 grep --color=auto kubelet
```

You should see:

- `-authorization-mode=Node,RBAC` for the API server
- `anonymous-auth=false` for the Kubelet
- `authorization-mode=Webhook` for the Kubelet

Check control plane health:

```
kubectl get pods -n kube-system
```



```
controlplane /etc/kubernetes/manifests ➜ kubectl get pods -n kube-system
NAME                                   READY   STATUS    RESTARTS        AGE
coredns-6678bcd974-77rdp               1/1     Running   0               31m
coredns-6678bcd974-wvz9k               1/1     Running   0               31m
etcd-controlplane                      1/1     Running   0               31m
kube-apiserver-controlplane            1/1     Running   0               4m11s
kube-controller-manager-controlplane   1/1     Running   4 (4m44s ago)   31m
kube-proxy-n7s8h                       1/1     Running   0               30m
kube-proxy-pxgm4                       1/1     Running   0               31m
kube-scheduler-controlplane            1/1     Running   4 (4m44s ago)   31m
```

The `kube-apiserver` pod must show `Running`.

Expected Outcome

- API server and Kubelet now meet CIS authorization requirements.
- Anonymous Kubelet access is disabled.
- Insecure modes such as AlwaysAllow are eliminated.
- All components restarted cleanly without errors.

Exam Tip

Exam Tip

- The API server is always fixed through static pod manifests in the manifests directory.
- The Kubelet is always fixed through its configuration file in `/var/lib/kubelet/config.yaml`.
- Use `ps -ef` to confirm the actual running process flags.
- YAML indentation errors in the Kubelet config will prevent it from starting.

Homework

1. Set `anonymous-auth=true` again and practice fixing it until you can correct the issue in under one minute.
2. Intentionally break the Kubelet by misspelling the authorization mode and observe how it behaves. Repair it.
3. Run a CIS scan using `kube-bench` and confirm all authorization checks now pass.

1.0.4 Scenario 4: Hardening Control Plane Components For CIS Compliance

High Level Goal

A CIS Kubernetes Benchmark scan reported multiple findings on your control plane. Your job is to review and harden the **API server**, the **Kubelet**, and **etcd** so that:

- The API server does not run with insecure or missing authorization settings.
- The Kubelet does not allow anonymous requests and uses webhook authorization.
- etcd requires client certificate authentication.

Cluster Topology (Lab):

- Control plane node: `controlplane`
- Worker node: `node01`
- OS: Ubuntu 22.04
- Runtime: `containerd`
- Kubernetes version: `v1.34.0`

> Important
>
> Real exam clusters may already be close to CIS compliant. In the lab you may need to **deliberately introduce** a bad flag first so you can practice spotting and fixing it. Always restore the component to a **valid** and **supported** configuration. Adding flags that do not exist will crash the control plane.

CIS Scan Report And What It Means

Assume a CIS scan produced a report similar to:

```
[FAIL] 1.2.7 Ensure that the --authorization-mode argument
           is not set to AlwaysAllow
[FAIL] 1.1.12 Ensure that the --client-cert-auth argument
             is set to true for etcd
[FAIL] 4.2.1 Ensure that the --anonymous-auth argument
           is set to false on the Kubelet
[WARN] 1.2.x Confirm that no insecure API server ports
             are exposed to unauthenticated clients
```

Interpretation:

- API server is either using `AlwaysAllow` or not using `Node,RBAC`.
- etcd is not enforcing client certificate authentication.
- Kubelet accepts anonymous requests.
- The scan wants you to make sure there is **no insecure port**. On modern clusters this usually means: there is **no** insecure port flag at all, not an extra `-insecure-port` line.

In the exam the question will usually say something like:

Example exam style wording *A CIS scan reported that the API server is configured with an insecure authorization mode. Update the API server configuration on the control plane node* *so that it uses Node and RBAC authorization. Restart the component and verify that the node remains Ready.*

You respond by editing static pod manifests or config files, not by running the scanner.

Step 1: Harden The API Server

Lab Task On the `controlplane` node:

```
cd /etc/kubernetes/manifests
sudo vi kube-apiserver.yaml
```

Inside the container `spec`, locate the `command:` section. You might see something like:

```
spec:
  containers:
  - name: kube-apiserver
    image: k8s.gcr.io/kube-apiserver:v1.34.0
    command:
    - kube-apiserver
    - --advertise-address=192.168.251.249
    - --etcd-servers=https://127.0.0.1:2379
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --authorization-mode=AlwaysAllow
```

Replace the insecure mode with the CIS recommended one:

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

On modern clusters you **do not** add any explicit insecure port flag. You just make sure there is no line enabling an insecure port.

Save and exit:

```
:wq
```

The kubelet automatically restarts the static pod when the manifest changes.

Watch for the new pod:

```
kubectl get pods -n kube-system | grep apiserver
```

Wait until the pod is `Running` and not restarting.

Exam viewpoint In the exam you would not install any extra tools. You edit `/etc/kubernetes/manifests/kube-apiserver.yaml`, fix the flag, save, and check:

```
kubectl get nodes
```

If the node is still `Ready`, your change is accepted.

Step 2: Harden The Kubelet

Lab Task Still on `controlplane`:

```
sudo vi /var/lib/kubelet/config.yaml
```

Ensure these sections exist:

```
authentication:
  anonymous:
    enabled: false

authorization:
  mode: Webhook
```

Save the file then restart the kubelet:

```
sudo systemctl daemon-reexec
sudo systemctl restart kubelet
```

Validate:

```
ps -ef | grep kubelet | grep -v grep
```

You should not see any old flags like `-anonymous-auth=true`.

Exam viewpoint In the exam the config may already live in `/var/lib/kubelet/config.yaml`. You edit it, set `anonymous.enabled: false` and `authorization.mode: Webhook`, then run the documented restart commands that are usually part of the question or the node notes.

Step 3: Enforce etcd Client Certificate Authentication

Lab Task Open the etcd static pod manifest on `controlplane`:

```
cd /etc/kubernetes/manifests
sudo vi etcd.yaml
```

Locate the `command:` list. Make sure it includes:

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

In a kubeadm based lab you will usually already see flags like:

```
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
```

Do not change those paths. They match the certificates generated by kubeadm.

Save and exit. etcd is also a static pod, so the kubelet restarts it for you.

Check etcd:

```
sudo crictl ps -a | grep etcd
sudo crictl logs $(sudo crictl ps -a --name etcd -q | head -n1) | tail
```

You should not see certificate errors. If etcd is healthy, the apiserver can connect.

Exam viewpoint The exam will not ask you to generate new etcd certificates from scratch. You usually only toggle `-client-cert-auth=true` or similar flags and keep the existing certificate files.

Step 4: Final Validation

Run a quick health check:

```
kubectl get nodes
kubectl get pods -n kube-system
```

Then review running processes:

```
ps -ef | grep kube-apiserver | grep -v grep
ps -ef | grep kubelet | grep -v grep
ps -ef | grep etcd | grep -v grep
```

You should be able to spot:

- `-authorization-mode=Node,RBAC` on `kube-apiserver`.
- Kubelet using config file fields that disable anonymous auth and use webhook auth.
- `-client-cert-auth=true` on `etcd`.

If all control plane pods in `kube-system` are `Running`, the cluster is stable.

Success Criteria

- API server pod is running with `Node,RBAC` authorization and no insecure authorization mode.
- Kubelet does not allow anonymous requests and is configured with webhook authorization.
- etcd enforces client certificate authentication and is healthy.
- `kubectl get nodes` returns both nodes in `Ready` state.

Always Remember

- Use static pod manifests for API server and etcd, use config files and systemd for the Kubelet.
- Do not invent flags. If you are unsure whether a flag still exists, search the official docs before adding it.
- After every change, validate with both `ps -ef` and `kubectl get pods -n kube-system`.
- If `kubectl` breaks, debug from the node using `crictl logs` and by checking the manifests directly.

Homework: Your Own CIS Hardening Drill

1. Intentionally break the API server in the lab by putting `-authorization-mode=AlwaysAllow` back, then fix it again.
2. Change the Kubelet config to allow anonymous auth, restart, and confirm that your CIS style checks would fail, then restore it.
3. On a fresh kubeadm cluster, inspect all control plane manifests and list every flag that relates to encryption, certificates, or authorization. For each one, explain in your own words why it matters.

Premium Version Scenarios

In the premium version, you will find the following scenarios:

Additional Scenarios

Get More Books and the CKA + CKS Bundle

If you want more hands-on DevOps and Kubernetes books, labs, and bundles, visit my store here:

- Payhip Store:
  `https://payhip.com/DevOpsDynamo`
- Gumroad Store:
  `https://devopsdynamo.gumroad.com/`

Recommended: If you are preparing for both certifications, grab the **CKA + CKS Bundle** so you can build strong Kubernetes fundamentals first, then move into security hardening and advanced CKS scenarios.