This comprehensive study guide is aligned with the **official CKS 2026 curriculum** (Kubernetes v1.34+). It combines content from Benjamin Muschko's *Certified Kubernetes Security Specialist (CKS) Study Guide* (O'Reilly, 2023) with critical **2024-2026 curriculum additions** including SBOM, KubeLinter, and Cilium/Istio pod-to-pod encryption.

The CKS is a performance-based exam lasting **2 hours** with 15-20 hands-on tasks. The passing score requires approximately 67%. This guide provides exam-aligned competencies, fast-path commands, common traps, and verification steps for each topic.

Prerequisites: Valid CKA certification required. This guide assumes CKA-level Kubernetes administration knowledge.

- **Exam-Day Quick Sheet (3-Minute Read)**

Critical kubectl shortcuts:

```
alias k=kubectl
alias kn='kubectl config set-context --current --namespace'
export do='--dry-run=client -o yaml'
```

Fast verification commands:

```
# Check cluster health
kubectl get nodes
kubectl get pods -A | grep -v Running

# RBAC verification
kubectl auth can-i --list --as=system:serviceaccount:default:my-sa

# Network policies in namespace
kubectl get networkpolicies -n <namespace>

# Security context check
kubectl get pod <pod> -o jsonpath='{.spec.securityContext}'

# Audit logs location
/var/log/kubernetes/audit/

# Falco logs
journalctl -fu falco

# AppArmor status
aa-status

# seccomp profiles
/var/lib/kubelet/seccomp/
```

AppArmor (Kubernetes 1.30+):

```
# Container-level (preferred)
spec:
  containers:
  - name: app
    securityContext:
      appArmorProfile:
        type: [Localhost](http://Localhost)
        localhostProfile: <profile-name>

# Pod-level (applies to all containers)
spec:
  securityContext:
    appArmorProfile:
      type: RuntimeDefault  # or [Localhost/Unconfined]
(http://Localhost/Unconfined)
```

⚠ **BREAKING CHANGE (1.34):** Old annotation method is **removed**. Use `securityContext.appArmorProfile` only.

Key file locations:

- API Server config: `/etc/kubernetes/manifests/kube-apiserver.yaml`
- Kubelet config: `/var/lib/kubelet/config.yaml`
- Audit policy: `/etc/kubernetes/audit/policy.yaml`
- etcd data: `/var/lib/etcd`
- PSA namespace labels: `pod-security.kubernetes.io/{enforce|audit|warn}`

Emergency fixes:

```
# Restart kubelet after config changes
systemctl daemon-reload && systemctl restart kubelet

# Force pod deletion
kubectl delete pod <pod> --force --grace-period=0

# Check API server logs
crictl logs <api-server-container-id>
```

Table of Contents

Domain 1 — Cluster Setup (15%)

Exam Checklist

- [ ] Create deny-all and allow-specific NetworkPolicies
- [ ] Run kube-bench and fix CIS benchmark findings
- [ ] Configure Ingress with TLS termination
- [ ] Protect cloud metadata endpoints (169.254.169.254)
- [ ] Verify Kubernetes binaries using SHA256 checksums

Core Concepts

NetworkPolicies — The Foundation of Zero-Trust Networking

In Kubernetes, the default networking model allows unrestricted communication between all pods across all namespaces—a dangerous baseline for production environments. NetworkPolicies implement the principle of least privilege at the network layer by acting as a firewall for pod-to-pod communication.

How NetworkPolicies Work:

- NetworkPolicies are namespace-scoped resources that select pods using label selectors
- They define allowed ingress (incoming) and egress (outgoing) traffic rules
- **Critical concept**: If no NetworkPolicy selects a pod, ALL traffic is allowed (default-allow)
- Once ANY NetworkPolicy selects a pod, only explicitly allowed traffic passes (default-deny for that pod)
- The CNI plugin must support NetworkPolicies (Calico, Cilium, Weave Net do; Flannel does NOT)

Policy Types:

- `Ingress`: Controls incoming traffic to selected pods
- `Egress`: Controls outgoing traffic from selected pods
- Both can be combined in a single policy

Selector Logic:

- `podSelector: {}` (empty) selects ALL pods in the namespace
- Multiple selectors in a single `from`/`to` block = AND logic
- Multiple `from`/`to` blocks = OR logic

CIS Benchmarks and kube-bench

The Center for Internet Security (CIS) publishes security configuration benchmarks—detailed guidelines for hardening systems against cyber attacks. The CIS Kubernetes Benchmark provides specific recommendations for securing Kubernetes clusters, covering the API server, etcd, controller manager, scheduler, and worker nodes.

kube-bench is an open-source Go application that automates CIS benchmark verification. It checks your cluster configuration against CIS recommendations and reports:

- **PASS**: Configuration meets CIS recommendation
- **FAIL**: Configuration violates CIS recommendation (security risk)
- **WARN**: Manual verification needed
- **INFO**: Informational only

Common kube-bench findings to fix:

- Anonymous authentication enabled on API server
- Insecure port not disabled
- Profiling enabled
- Authorization mode not set properly
- Audit logging not configured

Ingress TLS Termination

Ingress resources route external HTTP/HTTPS traffic to internal Services. TLS termination at the Ingress layer provides encryption for external traffic while allowing unencrypted internal communication (reducing computational overhead).

Key TLS concepts:

- TLS certificates are stored in Kubernetes Secrets of type `kubernetes.io/tls`
- The Secret must contain `tls.crt` (certificate) and `tls.key` (private key)
- Ingress references the Secret in `spec.tls[].secretName`
- Multiple TLS hosts can be configured with different certificates
- Self-signed certificates work but browsers will show warnings

Cloud Metadata Protection

Cloud providers expose metadata APIs at link-local addresses (AWS/GCP: 169.254.169.254, Azure: 169.254.169.254) that provide:

- Instance credentials and IAM roles
- Cloud provider API tokens
- Instance metadata (hostname, network config)

Security risk: A compromised pod can access these credentials and potentially escalate privileges to the cloud account level. Protection methods:

1. **NetworkPolicies**: Block egress to 169.254.169.254
2. **Cloud-native controls**: AWS IMDSv2 requires session tokens; GCP Workload Identity
3. **Node-level iptables**: Block metadata access at the host level

Hands-on Recipes

- **1.1 Default Deny-All NetworkPolicy**

What it looks like in the exam: "Restrict all ingress/egress traffic for namespace X except specific allowed connections."

Fast path:

```
apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: secure-ns
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

⚠ **Trap:** Empty `podSelector: {}` selects ALL pods. Missing `policyTypes` defaults to Ingress only.

✅ **Verify:**

```
kubectl exec -n secure-ns <pod> -- wget --timeout=1 <target-ip>:80
# Should timeout
```

- **1.2 Allow Specific Ingress Traffic**

What it looks like in the exam: "Allow frontend pods to access backend pods on port 3000 only."

Fast path:

```
apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 3000
```

⚠ **Trap:** `namespaceSelector` AND `podSelector` in same `-from` block = AND logic. Separate `-from` blocks = OR logic.

✅ **Verify:**

```
kubectl exec frontend-pod -- wget --spider --timeout=1 <backend-ip>:3000
# Should succeed
kubectl exec other-pod -- wget --spider --timeout=1 <backend-ip>:3000
# Should fail
```

- **1.3 Run kube-bench CIS Benchmark**

What it looks like in the exam: "Run CIS benchmark against control plane and fix failed checks."

Fast path:

```
# Run as Job on control plane
kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-
bench/main/job.yaml

# View results
kubectl logs job/kube-bench

# Common fix: Disable anonymous auth in API server
# Edit /etc/kubernetes/manifests/kube-apiserver.yaml
# Add: --anonymous-auth=false
```

⚠ **Trap:** API server restarts automatically after manifest change—wait for pod to stabilize before re-running kube-bench.

✅ **Verify:**

```
kubectl delete job kube-bench
kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-
bench/main/job.yaml
kubectl logs job/kube-bench | grep FAIL
```

- **1.4 Configure Ingress with TLS**

What it looks like in the exam: "Create an Ingress with TLS termination for domain example.com."

Fast path:

```
# Generate self-signed cert
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout tls.key -out tls.crt -subj "/CN=[example.com]
(http://example.com)"

# Create TLS secret
kubectl create secret tls example-tls --cert=tls.crt --key=tls.key
```

```
apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)
kind: Ingress
metadata:
  name: example-ingress
spec:
  tls:
  - hosts:
    - [example.com](http://example.com)
    secretName: example-tls
  rules:
  - host: [example.com](http://example.com)
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 80
```

⚠ **Trap:** TLS secret must be type `kubernetes.io/tls`with keys `tls.crt` and `tls.key`. Wrong key names cause silent failure.

✅ **Verify:**

```
curl -k https://example.com --resolve [example.com:443]
(http://example.com:443):<ingress-ip>
```

- **1.5 Protect Cloud Metadata Endpoint**

What it looks like in the exam: "Block pod access to cloud metadata API."

Fast path:

```
apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)
kind: NetworkPolicy
metadata:
  name: block-metadata
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  # Allow DNS
  - to:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
  # Allow all other egress EXCEPT metadata
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        - 169.254.169.254/32
```

⚠ **Trap:** Must include DNS egress rule if pods need DNS resolution—metadata block alone may break networking.

✅ **Verify:**

```
kubectl exec <pod> -- curl --connect-timeout 1 http://169.254.169.254/
# Should timeout or fail
```

- **1.6 Verify Platform Binaries**

What it looks like in the exam: "Verify kubectl binary integrity against official checksum."

Fast path:

```
# Download checksum
VERSION=v1.34.0
curl -LO "https://dl.k8s.io/release/${VERSION}/bin/linux/amd64/kubectl"
curl -LO
"https://dl.k8s.io/release/${VERSION}/bin/linux/amd64/kubectl.sha256"

# Verify checksum
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

⚠ **Trap:** Ensure you're using the checksum file matching your binary version and architecture.

✅ **Verify:** Output shows "kubectl: OK" or "kubectl: FAILED".

Timed Labs (Domain 1)

Lab 1.1 (10 min): Create a namespace `secure-app`. Deploy nginx pod labeled `role=web` and alpine pod labeled `role=client`. Create NetworkPolicy allowing only client→web on port 80. Verify client can reach web, but external pods cannot.

Lab 1.2 (15 min): Run kube-bench on control plane node. Identify 3 FAIL results. Fix at least 2 of them by modifying API server configuration. Re-run kube-bench to verify fixes.

Lab 1.3 (12 min): Create TLS secret from provided cert/key. Create Ingress routing `secure.example.com` to backend service with TLS termination. Verify HTTPS works using curl.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
NetworkPolicy not blocking	CNI doesn't support NP	Check CNI (Calico, Cilium support NP)
kube-bench job pending	No node selector match	Remove node selector or fix labels
Ingress TLS not working	Wrong secret type/keys	Recreate with `kubectl create secret tls`
API server not starting	YAML syntax error	`crictl logs <container>` for errors

Mini Review (Domain 1)

Q1: What does `podSelector: {}` mean in a NetworkPolicy?

A1: Selects all pods in the namespace.

Q2: Which admission plugin should be enabled for production clusters?

A2: NodeRestriction (limits kubelet node/pod access).

Q3: What port does the cloud metadata service use?

A3: Port 80 at IP 169.254.169.254.

Q4: Where is the API server manifest located?

A4: `/etc/kubernetes/manifests/kube-apiserver.yaml`

Q5: What tool automates CIS benchmark checks?

A5: kube-bench.

Q6: What secret type is required for Ingress TLS?

A6: `kubernetes.io/tls`

Q7: How do you restart API server after config change?

A7: It auto-restarts when manifest changes (kubelet watches /etc/kubernetes/manifests/).

Q8: What's the difference between Ingress and NetworkPolicy?

A8: Ingress = L7 HTTP routing; NetworkPolicy = L3/L4 pod communication rules.

Q9: What happens if no NetworkPolicy exists in a namespace?

A9: All traffic allowed (default allow).

Q10: How do you verify a binary's integrity?

A10: Compare SHA256 checksum: `sha256sum --check`

Domain 2 — Cluster Hardening (15%)

Exam Checklist

- [ ] Implement RBAC with Role, ClusterRole, RoleBinding, ClusterRoleBinding
- [ ] Minimize service account permissions
- [ ] Disable service account token automounting
- [ ] Restrict API server access
- [ ] Perform Kubernetes version upgrade

Core Concepts

API Server Request Processing

Every request to the Kubernetes API server passes through four critical stages:

1. **Authentication**: Validates the identity of the requester. Methods include client certificates, bearer tokens (service account tokens), and authentication proxies. Anonymous access is disabled by default in production.
2. **Authorization**: After identity is confirmed, RBAC checks if the user has permission for the requested action. Uses Role/ClusterRole definitions bound to subjects via RoleBindings/ClusterRoleBindings.
3. **Admission Control**: Validates and optionally modifies the request. Includes mutating webhooks (modify resources) and validating webhooks (accept/reject). Key plugins: NodeRestriction, PodSecurity, ResourceQuota.
4. **Validation**: Finally validates the resource against the API schema before persisting to etcd.

RBAC Deep Dive

Role-Based Access Control consists of four key resources:

- **Role**: Grants permissions within a specific namespace. Defines verbs (get, list, create, delete, etc.) on resources (pods, secrets, configmaps, etc.).
- **ClusterRole**: Same as Role but cluster-scoped. Required for cluster-wide resources (nodes, persistentvolumes) or when used across namespaces.
- **RoleBinding**: Maps a Role to subjects (users, groups, service accounts) within a namespace. Can also reference a ClusterRole but limits scope to the namespace.
- **ClusterRoleBinding**: Maps a ClusterRole to subjects cluster-wide.

Creating Users (Certificate-Based):

1. Generate private key: `openssl genrsa -out user.key 2048`
2. Create CSR: `openssl req -new -key user.key -out user.csr -subj "/CN=username/O=group"`
3. Submit CSR to Kubernetes: Create CertificateSigningRequest resource
4. Approve CSR: `kubectl certificate approve <csr-name>`
5. Create Role/RoleBinding for the user
6. Generate kubeconfig with the certificate

Service Account Security

By default, every pod mounts a service account token at `/var/run/secrets/[kubernetes.io/serviceaccount/token](http://kubernetes.io/serviceaccount/token)`. This token allows API access with whatever permissions the SA has. Security best practices:

- **Disable automounting**: Set `automountServiceAccountToken: false` on the ServiceAccount or Pod spec
- **Create purpose-specific SAs**: Don't use the default SA; create minimal-privilege SAs per workload
- **Bound tokens**: Use `kubectl create token <sa-name>` for time-limited tokens instead of legacy long-lived tokens
- **RBAC binding format**: When binding to a ServiceAccount, use `system:serviceaccount:<namespace>:<sa-name>`

Kubernetes Version Upgrades

Kubernetes follows semantic versioning (major.minor.patch) with new minor releases every ~3 months. Security fixes are backported to the previous 2 minor versions. Upgrade rules:

- **Never skip minor versions** - upgrade sequentially (1.32 → 1.33 → 1.34)
- **Upgrade order**: kubeadm first, then control plane components, then kubelet/kubectl
- **Process**: drain node → upgrade kubeadm → `kubeadm upgrade apply` → upgrade kubelet/kubectl → restart kubelet → uncordon node

Hands-on Recipes

- **2.1 Create RBAC for Limited User**

What it looks like in the exam: "Create role allowing user to only list and get pods in namespace dev."

Fast path:

```
# Create Role
kubectl create role pod-reader --verb=get,list --resource=pods -n dev

# Create RoleBinding
kubectl create rolebinding dev-pod-reader --role=pod-reader --user=jane -n
dev
```

⚠ **Trap:** Role vs ClusterRole—Role is namespaced, ClusterRole is cluster-wide. Using RoleBinding with ClusterRole limits scope to namespace.

✅ **Verify:**

```
kubectl auth can-i list pods --as=jane -n dev  # yes
kubectl auth can-i delete pods --as=jane -n dev  # no
kubectl auth can-i list pods --as=jane -n default  # no
```

- **2.2 Secure Service Account Configuration**

What it looks like in the exam: "Create service account that doesn't automount API token."

Fast path:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: secure-sa
  namespace: default
automountServiceAccountToken: false
```

Or at Pod level:

```
spec:
  serviceAccountName: secure-sa
  automountServiceAccountToken: false
```

⚠ **Trap:** Pod-level setting overrides ServiceAccount-level setting.

✅ **Verify:**

```
kubectl exec <pod> -- ls /var/run/secrets/[kubernetes.io/serviceaccount/]
(http://kubernetes.io/serviceaccount/)
# Should show "No such file or directory"
```

- **2.3 Create Service Account with Minimal Permissions**

What it looks like in the exam: "Create SA that can only list pods in its namespace."

Fast path:

```
# Create SA
kubectl create sa pod-lister -n app-ns

# Create Role
kubectl create role list-pods --verb=list --resource=pods -n app-ns

# Bind Role to SA
kubectl create rolebinding pod-lister-binding \
  --role=list-pods \
  --serviceaccount=app-ns:pod-lister -n app-ns
```

⚠ **Trap:** ServiceAccount format in rolebinding is `namespace:name`.

✅ **Verify:**

```
kubectl auth can-i list pods --as=system:serviceaccount:app-ns:pod-lister
-n app-ns
```

- **2.4 Upgrade Kubernetes Cluster**

What it looks like in the exam: "Upgrade control plane from 1.33 to 1.34."

Fast path:

```
# Drain control plane
kubectl drain <control-plane-node> --ignore-daemonsets

# Upgrade kubeadm
apt-get update && apt-get install -y kubeadm=1.34.0-*

# Plan upgrade
kubeadm upgrade plan

# Apply upgrade
kubeadm upgrade apply v1.34.0

# Upgrade kubelet and kubectl
apt-get install -y kubelet=1.34.0-* kubectl=1.34.0-*
systemctl daemon-reload && systemctl restart kubelet

# Uncordon
kubectl uncordon <control-plane-node>
```

⚠ **Trap:** Always upgrade kubeadm first, then kubelet/kubectl. Never skip minor versions.

✅ **Verify:**

```
kubectl get nodes
kubectl version
```

Timed Labs (Domain 2)

Lab 2.1 (8 min): Create service account `deploy-sa` in namespace `staging`. Grant it permission to create, list, delete deployments only. Verify permissions.

Lab 2.2 (10 min): Create a user certificate signing request (CSR), approve it, create kubeconfig entry. Grant user read-only access to pods cluster-wide.

Lab 2.3 (15 min): Perform kubeadm upgrade on a single-node cluster from current version to next minor version.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
"Forbidden" errors	Missing RBAC binding	Check `kubectl auth can-i --list --as=<user>`
SA token still mounted	Pod-level override needed	Set `automountServiceAccountToken: false` in Pod spec
Upgrade failed	Version skipped	Upgrade one minor version at a time
API server inaccessible	Wrong certificates	Check `/etc/kubernetes/pki/` certs

Mini Review (Domain 2)

Q1: What's the difference between Role and ClusterRole?

A1: Role is namespaced; ClusterRole is cluster-scoped.

Q2: How do you disable SA token automounting?

A2: Set `automountServiceAccountToken: false` on SA or Pod.

Q3: Where is the SA token mounted in pods?

A3: `/var/run/secrets/[kubernetes.io/serviceaccount/](http://kubernetes.io/serviceaccount/)`

Q4: What command tests if a user can perform an action?

A4: `kubectl auth can-i <verb> <resource> --as=<user>`

Q5: What's the upgrade order for Kubernetes components?

A5: kubeadm → control plane → kubelet/kubectl

Q6: How do you create a token for a service account?

A6: `kubectl create token <sa-name>`

Q7: What does RoleBinding with ClusterRole do?

A7: Grants ClusterRole permissions only in the RoleBinding's namespace.

Q8: What verbs are needed for full pod management?

A8: create, get, list, watch, update, patch, delete

Q9: What happens to existing workloads during upgrade?

A9: They continue running; drain node first to reschedule.

Q10: How do you verify RBAC is working?

A10: `kubectl auth can-i --list --as=system:serviceaccount:<ns>:<sa>`

Domain 3 — System Hardening (10%)

Exam Checklist

- [ ] Minimize host OS attack surface
- [ ] Apply least-privilege IAM principles
- [ ] Restrict network access with firewall rules
- [ ] Implement AppArmor profiles for containers
- [ ] Configure seccomp profiles for pods

Core Concepts

OS Footprint Minimization

Every service running on a node is a potential attack vector. Minimize the host OS footprint by:

- **Disable unnecessary services**: Use `systemctl stop <service>` and `systemctl disable <service>` to stop and prevent auto-start
- **Remove unneeded packages**: `apt purge --auto-remove <package>` removes packages and dependencies
- **Reference**: CIS Ubuntu Benchmark provides specific hardening recommendations
- **Identify listening services**: `ss -ltpn` or `netstat -tulpn` shows all open ports and associated processes

IAM and Access Management

- **User operations**: `adduser` (create), `userdel` (delete), `su` (switch user), `sudo` (execute as root)
- **Group management**: `groupadd`, `groupdel`, `usermod -g <group> <user>`
- **File permissions**: `chmod` (change mode), `chown` (change owner)
- **User entry format in /etc/passwd**: `username:password:UID:GID:comment:home:shell`

Network Access Control

Limit network exposure using firewalls:

```
# Using ufw (Uncomplicated Firewall)
ufw default deny incoming
ufw default deny outgoing
ufw allow 6443/tcp   # API server
ufw allow 10250/tcp  # kubelet
ufw allow 2379:2380/tcp  # etcd
ufw enable
```

AppArmor

AppArmor is a Linux kernel security module that restricts program capabilities based on profiles:

- **Profile location**: `/etc/apparmor.d/`
- **Load profile**: `apparmor_parser -r /etc/apparmor.d/<profile>`
- **Enforcement modes**:
  - `enforce`: Blocks and logs violations
  - `complain`: Logs violations but doesn't block
- **Check loaded profiles**: `aa-status`
- **Apply to Pod**: Use securityContext/ appArmorProfile
- **Critical**: Profile must be loaded on every node where the pod might run

seccomp (Secure Computing Mode)

Seccomp filters system calls from userspace to kernel, reducing attack surface:

- **Profile types**:
  - `RuntimeDefault`: Uses container runtime's default profile (recommended baseline)
  - `Localhost`: Custom profile from `/var/lib/kubelet/seccomp/`
  - `Unconfined`: No restrictions (avoid in production)
- **Profile location**: Custom profiles go in `/var/lib/kubelet/seccomp/` on nodes
- **Profile actions**:
  - `SCMP_ACT_ALLOW`: Permit the syscall
  - `SCMP_ACT_ERRNO`: Deny and return error
  - `SCMP_ACT_LOG`: Log but allow
- **Apply via securityContext**:

```
spec:
  securityContext:
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: busybox
```

Or for custom profiles:

```
spec:
  securityContext:
    seccompProfile:
      type: [Localhost](http://Localhost)
      localhostProfile: profiles/custom-profile.json
  containers:
  - name: app
    image: busybox
```

Hands-on Recipes

- **3.1 Apply AppArmor Profile to Pod**

What it looks like in the exam: "Configure pod to use AppArmor profile that denies file writes."

Fast path:

First, create profile on node:

```
# /etc/apparmor.d/deny-write
#include <tunables/global>
profile deny-write flags=(attach_disconnected) {
  #include <abstractions/base>
  file,
  deny /** w,
}
```

Load profile:

```
apparmor_parser -r /etc/apparmor.d/deny-write
```

Reference in Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: apparmor-pod
spec:
  containers:
  - name: app
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    securityContext:
      appArmorProfile:
        type: Localhost
        localhostProfile: deny-write
```

⚠ **Trap:** Profile must be loaded on the node where pod runs. Use `securityContext.appArmorProfile`, NOT annotations (deprecated in 1.30, removed in 1.34).

✅ **Verify:**

```
aa-status | grep deny-write
kubectl exec apparmor-pod -- touch /test
# Should fail: Permission denied
```

- **3.2 Configure seccomp Profile**

What it looks like in the exam: "Apply RuntimeDefault seccomp profile to pod."

Fast path (RuntimeDefault):

```
apiVersion: v1
kind: Pod
metadata:
  name: seccomp-pod
spec:
  securityContext:
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
```

Custom profile (at /var/lib/kubelet/seccomp/profiles/audit.json):

```
spec:
  securityContext:
    seccompProfile:
      type: [Localhost](http://Localhost)
      localhostProfile: profiles/audit.json
  containers:
  - name: test
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
```

⚠ **Trap:** Custom profiles must exist at `/var/lib/kubelet/seccomp/` on the node. Path is relative to this directory.

✅ **Verify:**

```
kubectl get pod seccomp-pod -o
jsonpath='{.spec.securityContext.seccompProfile}'
```

- **3.3 Minimize Network Access**

What it looks like in the exam: "Block unnecessary outbound connections from node."

Fast path (using ufw):

```
# Allow SSH
ufw allow 22/tcp

# Allow Kubernetes API
ufw allow 6443/tcp

# Allow kubelet
ufw allow 10250/tcp

# Enable firewall
ufw enable
```

⚠ **Trap:** Ensure kubelet (10250), API server (6443), and etcd (2379-2380) ports are allowed.

✅ **Verify:**

```
ufw status
```

Timed Labs (Domain 3)

Lab 3.1 (12 min): Create AppArmor profile that blocks network access. Load it on worker node. Create pod using this profile and verify network calls fail.

Lab 3.2 (8 min): Create pod with RuntimeDefault seccomp profile. Verify it's applied. Compare behavior with Unconfined profile.

Lab 3.3 (10 min): Identify and disable 3 unnecessary systemd services on a cluster node. Document services stopped.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
Pod stuck in `Blocked`	AppArmor profile not loaded	`apparmor_parser -r <profile>`
seccomp profile not found	Wrong path	Check `/var/lib/kubelet/seccomp/`
Container crashes with seccomp	Profile too restrictive	Use `RuntimeDefault` first
Node unreachable after firewall	Blocked k8s ports	Allow 6443, 10250, 2379-2380

Mini Review (Domain 3)

Q1: Where are AppArmor profiles stored?

A1: `/etc/apparmor.d/`

Q2: How do you check loaded AppArmor profiles?

A2: `aa-status`

Q3: Where are seccomp profiles stored for Kubernetes?

A3: `/var/lib/kubelet/seccomp/`

Q4: What's the default seccomp profile type?

A4: Unconfined (no restrictions)

Q5: How do you load an AppArmor profile?

A5: `apparmor_parser -r /etc/apparmor.d/<profile>`

Q6: What seccomp type uses the container runtime's default?

A6: RuntimeDefault

Q7: How is AppArmor specified for containers? *(Updated for Kubernetes 1.30+)*

A7: Use `securityContext.appArmorProfile` field with `type: [Localhost](http://Localhost)` and `localhostProfile: <profile-name>`. Can be set at pod or container level. ⚠ The old annotation method (`container.apparmor.security.beta.kubernetes.io/*`) is **deprecated since 1.30 and removed in 1.34**.

Q8: What are common Kubernetes ports to allow in firewall?

A8: 6443 (API), 10250 (kubelet), 2379-2380 (etcd)

Q9: What command shows listening ports?

A9: `ss -tulpn` or `netstat -tulpn`

Q10: How do you disable a systemd service?

A10: `systemctl disable --now <service>`

Domain 4 — Minimize Microservice Vulnerabilities (20%)

Exam Checklist

- [ ] Configure Pod security contexts (runAsNonRoot, readOnlyRootFilesystem)
- [ ] Implement Pod Security Standards (PSS) with Pod Security Admission (PSA)
- [ ] Use OPA Gatekeeper for policy enforcement
- [ ] Manage Kubernetes Secrets securely
- [ ] Configure container runtime sandboxes (gVisor, Kata)
- [ ] Implement pod-to-pod encryption (Cilium, Istio mTLS)

Core Concepts

Security Contexts

Security contexts define privilege and access control settings at Pod or container level:

- **Pod-level securityContext**: Applies to all containers in the pod
- **Container-level securityContext**: Overrides pod-level settings for that container
- **Key fields**:
  - `runAsNonRoot: true` - Prevents running as root user (UID 0)
  - `runAsUser: \<uid\>` - Specifies exact UID to run as
  - `runAsGroup: \<gid\>` - Specifies primary group
  - `fsGroup: \<gid\>` - Supplemental group for volume access
  - `privileged: true` - Grants host-level access (AVOID in production)
  - `allowPrivilegeEscalation: false` - Prevents gaining more privileges than parent
  - `readOnlyRootFilesystem: true` - Makes container filesystem read-only
  - `capabilities.drop: ["ALL"]` - Removes all Linux capabilities

Pod Security Admission (PSA)

PSA enforces Pod Security Standards at namespace level using labels:

- **Label format**: [`pod-security.kubernetes.io/<mode>](http://pod-security.kubernetes.io/<mode>): <level>`
- **Modes**:
  - `enforce`: Reject pods that violate policy
  - `audit`: Log violations but allow pod
  - `warn`: Show warning but allow pod
- **Levels** (cumulative restrictions):
  - `privileged`: Unrestricted (default if no label)
  - `baseline`: Minimal restrictions (no privileged, no hostNetwork, etc.)
  - `restricted`: Hardened best practices (non-root, read-only filesystem, drop ALL capabilities)
- PSA is enabled by default in Kubernetes 1.23+

OPA Gatekeeper

Open Policy Agent with Gatekeeper provides custom policy enforcement:

- Uses **Rego language** for policy definition
- **ConstraintTemplate**: Defines the policy logic and parameter schema
- **Constraint**: Implements the template with specific parameters and scope
- More flexible than PSA - can enforce any policy on any API resource
- Install: `kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml`

Secrets Encryption at Rest

By default, etcd stores Secrets in base64 encoding (NOT encrypted). Enable encryption:

1. Create EncryptionConfiguration with `aescbc` or `secretbox` provider
2. Generate encryption key: `head -c 32 /dev/urandom | base64`

3. Add API server flag: `--encryption-provider-config=/path/to/config`
4. Re-encrypt existing secrets: `kubectl get secrets -A -o json | kubectl replace -f -`
5. Verify: Use `etcdctl get /registry/secrets/default/<secret>` - should show encrypted data (`k8s:enc:aescbc:v1:key1`)

Runtime Sandboxes

Provide stronger isolation than standard containers:

- **gVisor (runsc)**: User-space kernel that intercepts syscalls, reducing kernel attack surface
- **Kata Containers**: Runs containers in lightweight VMs with dedicated kernels
- Use **RuntimeClass** to specify handler: `handler: runsc` or `handler: kata`
- Pod references via `spec.runtimeClassName: gvisor`
- Verify gVisor: `kubectl exec pod -- dmesg` shows "Starting gVisor..."

mTLS for Pod-to-Pod Encryption

Mutual TLS encrypts and authenticates both sides of pod communication:

- **Certificate complexity**: Requires CA, CSR management, rotation
- **Service meshes** (Istio, Linkerd): Automate mTLS with sidecar proxies
- **CNI-level encryption** (Cilium, Calico): Transparent WireGuard/IPsec encryption
- Cilium WireGuard is lower overhead than service mesh mTLS

Hands-on Recipes

- **4.1 Secure Pod Security Context**

What it looks like in the exam: "Configure pod to run as non-root with read-only filesystem."

Fast path:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  containers:
  - name: app
    image: nginx
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
    volumeMounts:
    - name: tmp
      mountPath: /tmp
    - name: cache
      mountPath: /var/cache/nginx
    - name: run
      mountPath: /var/run
  volumes:
  - name: tmp
    emptyDir: {}
  - name: cache
    emptyDir: {}
  - name: run
    emptyDir: {}
```

⚠ **Trap:** nginx needs writable directories—use emptyDir volumes. Some images require root—use alternatives like bitnami/nginx.

✅ **Verify:**

```
kubectl exec secure-pod -- id
kubectl exec secure-pod -- touch /test  # Should fail
```

- **4.2 Enable Pod Security Admission**

What it looks like in the exam: "Enforce restricted PSS on namespace."

Fast path:

```
apiVersion: v1
kind: Namespace
metadata:
  name: restricted-ns
  labels:
    [pod-security.kubernetes.io/enforce](http://pod-
security.kubernetes.io/enforce): restricted
    [pod-security.kubernetes.io/enforce-version](http://pod-
security.kubernetes.io/enforce-version): latest
    [pod-security.kubernetes.io/audit](http://pod-
security.kubernetes.io/audit): restricted
    [pod-security.kubernetes.io/warn](http://pod-
security.kubernetes.io/warn): restricted
```

⚠ **Trap:** PSA modes: enforce (reject), audit (log), warn (warning). Use all three for defense in depth.

✅ **Verify:**

```
# Try creating non-compliant pod
kubectl run test --image=nginx -n restricted-ns
# Should be rejected with PSS violations listed
```

- **4.3 Create OPA Gatekeeper Constraint**

What it looks like in the exam: "Enforce that all pods must have resource limits."

Fast path:

ConstraintTemplate:

```
apiVersion: [templates.gatekeeper.sh/v1]
(http://templates.gatekeeper.sh/v1)
kind: ConstraintTemplate
metadata:
  name: k8srequiredlimits
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLimits
  targets:
  - target: [admission.k8s.gatekeeper.sh]
(http://admission.k8s.gatekeeper.sh)
    rego: |
      package k8srequiredlimits
      violation[{"msg": msg}] {
        container := [input.review]
(http://input.review).object.spec.containers[_]
        not container.resources.limits
        msg := sprintf("Container %v has no resource limits",
[[container.name](http://container.name)])
      }
```

Constraint:

```
apiVersion: [constraints.gatekeeper.sh/v1beta1]
(http://constraints.gatekeeper.sh/v1beta1)
kind: K8sRequiredLimits
metadata:
  name: require-limits
spec:
  match:
    kinds:
    - apiGroups: [""]
      kinds: ["Pod"]
```

⚠ **Trap:** Apply ConstraintTemplate first, wait for it to be ready, then apply Constraint.

✅ **Verify:**

```
kubectl run test --image=nginx  # Should be rejected
```

- **4.4 Encrypt Secrets at Rest in etcd**

What it looks like in the exam: "Configure etcd encryption for secrets."

Fast path:

Create encryption config:

```
# /etc/kubernetes/enc/encryption-config.yaml
apiVersion: [apiserver.config.k8s.io/v1]
(http://apiserver.config.k8s.io/v1)
kind: EncryptionConfiguration
resources:
- resources:
  - secrets
  providers:
  - aescbc:
      keys:
      - name: key1
        secret: <base64-encoded-32-byte-key>
  - identity: {}
```

Generate key:

```
head -c 32 /dev/urandom | base64
```

Update API server:

```
# In /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --encryption-provider-config=/etc/kubernetes/enc/encryption-
config.yaml
    volumeMounts:
    - name: enc
      mountPath: /etc/kubernetes/enc
      readOnly: true
  volumes:
  - name: enc
    hostPath:
      path: /etc/kubernetes/enc
      type: DirectoryOrCreate
```

⚠ **Trap:** Existing secrets aren't re-encrypted automatically. Run `kubectl get secrets -A -o json | kubectl replace -f -` to re-encrypt.

✅ **Verify:**

```
ETCDCTL_API=3 etcdctl get /registry/secrets/default/my-secret \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key | hexdump -C
# Should show encrypted data, not plaintext
```

- **4.5 Use RuntimeClass for gVisor**

What it looks like in the exam: "Run pod with gVisor sandbox."

Fast path:

Create RuntimeClass:

```
apiVersion: [node.k8s.io/v1](http://node.k8s.io/v1)
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
```

Use in Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: sandboxed-pod
spec:
  runtimeClassName: gvisor
  containers:
  - name: app
    image: nginx
```

⚠ **Trap:** gVisor must be installed and configured on nodes. Not all workloads are compatible.

✅ **Verify:**

```
kubectl exec sandboxed-pod -- dmesg | head
# Should show gVisor kernel messages
```

Timed Labs (Domain 4)

Lab 4.1 (10 min): Create namespace with PSA restricted enforcement. Deploy pod that violates PSS, observe rejection. Fix pod spec to comply and deploy successfully.

Lab 4.2 (12 min): Configure etcd encryption for secrets. Create new secret, verify it's encrypted in etcd. Re-encrypt existing secrets.

Lab 4.3 (15 min): Install Gatekeeper. Create ConstraintTemplate and Constraint requiring all pods to have label `owner`. Verify enforcement.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
Pod rejected by PSA	Security context non-compliant	Add required securityContext fields
Gatekeeper not blocking	Constraint not matched	Check `spec.match.kinds` in constraint
etcd encryption not working	API server not restarted	Check API server logs, wait for restart
RuntimeClass not found	handler not configured	Verify containerd config on nodes

Mini Review (Domain 4)

Q1: What are the three PSS levels?

A1: Privileged, Baseline, Restricted

Q2: How do you enforce PSA on a namespace?

A2: Label: [`pod-security.kubernetes.io/enforce](http://pod-security.kubernetes.io/enforce): restricted`

Q3: What capability should always be dropped?

A3: ALL (then add back only what's needed)

Q4: How do you prevent privilege escalation?

A4: `securityContext.allowPrivilegeEscalation: false`

Q5: Where is the encryption config specified for API server?

A5: `--encryption-provider-config` flag

Q6: What's the difference between PSA and Gatekeeper?

A6: PSA is built-in with fixed policies; Gatekeeper allows custom Rego policies.

Q7: How do you make a container filesystem read-only?

A7: `securityContext.readOnlyRootFilesystem: true`

Q8: What RuntimeClass handler is used for gVisor?

A8: `runsc`

Q9: How do you run as non-root?

A9: `runAsNonRoot: true` and `runAsUser: <non-zero-uid>`

Q10: What's the default behavior if no PSA labels exist?

A10: Privileged (no restrictions)

Domain 5 — Supply Chain Security (20%)

Exam Checklist

- [ ] Build minimal container images (multi-stage, distroless)
- [ ] Understand and generate SBOMs (Trivy, bom)
- [ ] Scan images for vulnerabilities (Trivy)
- [ ] Restrict image registries (Gatekeeper, ImagePolicyWebhook)
- [ ] Perform static analysis (Kubesec, KubeLinter)
- [ ] Sign and verify container images

Core Concepts

Minimal Base Images

Smaller images mean fewer vulnerabilities and smaller attack surface:

- **Alpine Linux**: ~7MB, uses musl libc and BusyBox
- **Distroless images**: ~2MB, contain ONLY the application and runtime dependencies - no shell, package manager, or OS utilities
- Distroless prevents many attacks that rely on shell access

Multi-stage Builds

Separate build environment from runtime environment:

```
# Build stage - includes compilers, tools
FROM golang:1.19.4-alpine AS build
WORKDIR /app
COPY . .
RUN go build -o myapp

# Runtime stage - minimal image
FROM alpine:3.17.0
COPY --from=build /app/myapp /myapp
ENTRYPOINT ["/myapp"]
```

This drastically reduces final image size and removes build tools from production.

Layer Optimization

- Each `FROM`, `COPY`, `RUN`, `CMD` creates a layer
- Combine RUN commands with `&&` to reduce layers
- Fewer layers = faster builds, smaller images

Image Signing and Verification

- `docker trust sign` creates image digest (SHA256 hash)
- Use digest notation for immutable references: `image@sha256:abc123...`
- Tags can be overwritten; digests cannot
- Kubernetes validates digests when specified

Registry Security

- **Public registries**: docker.io, gcr.io, quay.io
- **Private registries**: Enterprise/company-hosted
- Whitelist allowed registries via:
  - OPA Gatekeeper K8sAllowedRepos constraint
  - ImagePolicyWebhook admission controller (calls external HTTPS service)

Static Analysis

- **Hadolint**: Lints Dockerfiles for best practices (tagging, WORKDIR usage)
- **Kubesec**: Scores Kubernetes manifests for security (higher score = better)
- **KubeLinter**: Checks YAML and Helm charts for misconfigurations, CI/CD friendly with non-zero exit codes

Vulnerability Scanning (Trivy)

- Scans images for CVEs from vulnerability databases
- Filter by severity: `trivy image --severity HIGH,CRITICAL <image>`
- Output formats: table (default), JSON, SARIF
- Can scan running pods, filesystems, and SBOMs
- Generates SBOMs in SPDX or CycloneDX format

SBOM (Software Bill of Materials)

Complete inventory of all software components:

- **SPDX**: ISO standard, focus on licensing and compliance
- **CycloneDX**: OWASP format, focus on security
- Tools: `bom generate`, `trivy image --format cyclonedx`
- Required by US Executive Order 14028 for federal software

Hands-on Recipes

- **5.1 Scan Image with Trivy**

What it looks like in the exam: "Identify and report HIGH/CRITICAL vulnerabilities in image."

Fast path:

```
# Scan image
trivy image nginx:1.21.6

# Filter by severity
trivy image --severity HIGH,CRITICAL nginx:1.21.6

# Output to JSON
trivy image --format json --output results.json nginx:1.21.6

# Scan running pods
for pod in $(kubectl get pods -o
jsonpath='{.items[*].spec.containers[*].image}'); do
  trivy image --severity CRITICAL $pod
done
```

⚠ **Trap:** Trivy needs network access to download vulnerability DB. Use `--skip-db-update` if pre-downloaded.

✅ **Verify:** Check output shows vulnerability table with CVE IDs and severity.

- **5.2 Generate SBOM**

What it looks like in the exam: "Generate SBOM in SPDX format for container image."

Fast path (using bom):

```
# Generate SPDX SBOM
bom generate --image nginx:1.23.1 --format spdx --output nginx-sbom.spdx

# Generate JSON format
bom generate --image nginx:1.23.1 --format json --output nginx-sbom.json
```

Using Trivy:

```
# Generate CycloneDX SBOM
trivy image --format cyclonedx --output sbom.cdx nginx:1.23.1

# Generate SPDX-JSON
trivy image --format spdx-json --output sbom.spdx.json nginx:1.23.1
```

Scan existing SBOM for vulnerabilities:

```
trivy sbom --format json --output vuln-report.json sbom.spdx.json
```

⚠ **Trap:** Different tools support different SBOM formats. Know which format each tool produces.

✅ **Verify:** Output file contains component list with names, versions, and licenses.

- **5.3 Analyze Manifest with Kubesec**

What it looks like in the exam: "Scan pod manifest and fix security issues."

Fast path:

```
# Scan manifest file
kubesec scan pod.yaml

# Scan from stdin
kubectl get pod mypod -o yaml | kubesec scan -

# Using online API
curl -sSX POST --data-binary @pod.yaml https://v2.kubesec.io/scan
```

Common fixes for Kubesec warnings:

```
spec:
  containers:
  - name: app
    securityContext:
      runAsNonRoot: true
      runAsUser: 10000
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
    resources:
      limits:
        cpu: "500m"
        memory: "128Mi"
```

⚠ **Trap:** Higher Kubesec score = better security. Score of 0 or negative indicates issues.

✅ **Verify:** Re-run kubesec to see improved score and fewer warnings.

- **5.4 Analyze with KubeLinter**

What it looks like in the exam: "Run static analysis on Kubernetes manifests."

Fast path:

```
# Scan single file
kube-linter lint pod.yaml

# Scan directory
kube-linter lint ./manifests/

# Scan Helm chart
kube-linter lint ./my-chart/

# List available checks
kube-linter checks list

# Run specific checks only
kube-linter lint --include "run-as-non-root,no-read-only-root-fs" pod.yaml
```

⚠ **Trap:** KubeLinter returns non-zero exit code on findings—useful for CI/CD.

✅ **Verify:** Fix reported issues and re-run until no errors.

- **5.5 Restrict Image Registries with Gatekeeper**

What it looks like in the exam: "Only allow images from approved registry."

Fast path:

ConstraintTemplate:

```
apiVersion: [templates.gatekeeper.sh/v1]
(http://templates.gatekeeper.sh/v1)
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
  - target: [admission.k8s.gatekeeper.sh]
(http://admission.k8s.gatekeeper.sh)
    rego: |
      package k8sallowedrepos
      violation[{"msg": msg}] {
        container := [input.review]
(http://input.review).object.spec.containers[_]
        satisfied := [good | repo = input.parameters.repos[_]; good =
startswith(container.image, repo)]
        not any(satisfied)
        msg := sprintf("Container %v uses image %v not from allowed
repos", [[container.name](http://container.name), container.image])
      }
```

Constraint:

```
apiVersion: [constraints.gatekeeper.sh/v1beta1]
(http://constraints.gatekeeper.sh/v1beta1)
kind: K8sAllowedRepos
metadata:
  name: allowed-repos
spec:
  match:
    kinds:
    - apiGroups: [""]
      kinds: ["Pod"]
  parameters:
    repos:
    - "[gcr.io/my-company/](http://gcr.io/my-company/)"
    - "[docker.io/library/](http://docker.io/library/)"
```

⚠ **Trap:** Include trailing slash in registry prefix. Check init containers and ephemeral containers too.

✅ **Verify:**

```
kubectl run test --image=unauthorized-registry/app:v1
# Should be rejected
```

- **5.6 Verify Image with SHA256 Digest**

What it looks like in the exam: "Use image digest instead of tag."

Fast path:

```
# Get image digest
docker pull nginx:1.23.1
docker inspect --format='{{index .RepoDigests 0}}' nginx:1.23.1
# Output: nginx@sha256:abc123...
```

Use in Pod:

```
spec:
  containers:
  - name: nginx
    image: nginx@sha256:abc123def456...
```

⚠ **Trap:** Digest is immutable—guaranteed same image content. Tags can be overwritten.

✅ **Verify:** `kubectl describe pod` shows digest in image field.

Timed Labs (Domain 5)

Lab 5.1 (10 min): Scan 3 images with Trivy. Identify which has CRITICAL vulnerabilities. Delete pods using vulnerable images.

Lab 5.2 (8 min): Generate SBOM for kube-apiserver image using bom. Save as SPDX JSON format.

Lab 5.3 (12 min): Create Gatekeeper policy restricting images to `gcr.io/google-containers/` only. Test with allowed and disallowed images.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
Trivy DB download failed	Network/proxy issue	Use `--skip-db-update` with cached DB
KubeLinter returns errors	Misconfigurations found	Fix issues reported in output
Image pull fails with digest	Digest doesn't exist	Verify digest with `docker manifest inspect`
Gatekeeper not enforcing	Constraint scope wrong	Check `spec.match` in constraint

Mini Review (Domain 5)

Q1: What tool scans images for CVEs?

A1: Trivy

Q2: What are the two main SBOM formats?

A2: SPDX and CycloneDX

Q3: What does Kubesec analyze?

A3: Kubernetes resource manifests for security best practices

Q4: What does KubeLinter check?

A4: YAML files and Helm charts for misconfigurations

Q5: How do you generate SBOM with Trivy?

A5: `trivy image --format cyclonedx --output sbom.json <image>`

Q6: What's better: image tag or digest?

A6: Digest—immutable and guarantees exact image content

Q7: How do you filter Trivy results by severity?

A7: `trivy image --severity HIGH,CRITICAL <image>`

Q8: What's a distroless image?

A8: Image with no shell, package manager, or OS utilities

Q9: How do you scan an SBOM for vulnerabilities?

A9: `trivy sbom <sbom-file>`

Q10: What command generates SBOM using bom tool?

A10: `bom generate --image <image> --format spdx --output <file>`

Domain 6 — Monitoring, Logging, and Runtime Security (20%)

Exam Checklist

- [ ] Install and configure Falco for behavioral analytics
- [ ] Create and modify Falco rules
- [ ] Ensure container immutability (readOnlyRootFilesystem, distroless)
- [ ] Configure Kubernetes audit logging
- [ ] Investigate security incidents using audit logs

Core Concepts

Falco

Falco is a behavioral analytics engine that monitors syscalls and Kubernetes audit events:

- **Rules location**:
  - `/etc/falco/falco_rules.yaml` - Default rules (don't modify, may be overwritten on updates)
  - `/etc/falco/falco_rules.local.yaml` - Custom rules (use this for modifications)
- **View logs**: `journalctl -fu falco`
- **Restart after changes**: `systemctl restart falco`
- **Common detections**: Shell spawned in container, package management process, sensitive file access

Falco Rule Structure:

```
- rule: <name>           # Rule name
  desc: <description>    # Human-readable description
  condition: <filter>    # When to trigger (boolean expression)
  output: <message>      # Alert message format
  priority: <level>      # Severity (EMERGENCY, ALERT, CRITICAL, ERROR,
WARNING, NOTICE, INFO, DEBUG)
```

- **Macros**: Reusable condition snippets (e.g., `spawned_process`, `container`)
- **Lists**: Reusable value lists (e.g., `shell_binaries: [bash, sh, zsh]`)

Container Immutability

Prevents runtime modifications to containers:

- `readOnlyRootFilesystem: true` - Blocks all filesystem writes
- Use **emptyDir volumes** for necessary writable paths (/tmp, /var/cache)
- **Distroless images** - No shell means no interactive modifications
- Inject configuration via **ConfigMaps** and **Secrets** as volumes
- Benefits: Prevents malware installation, config tampering, and unauthorized changes

Audit Logging

Records all API requests for security analysis and compliance:

- **Policy stages**:
  - `RequestReceived`: Event generated immediately
  - `ResponseStarted`: Response headers sent (long-running requests)
  - `ResponseComplete`: Response body completed
  - `Panic`: Panic occurred
- **Audit levels** (what to log):
  - `None`: Don't log
  - `Metadata`: Log request metadata (user, timestamp, resource, verb)
  - `Request`: Log metadata + request body
  - `RequestResponse`: Log metadata + request + response body

Audit Configuration:

```
apiVersion: [audit.k8s.io/v1](http://audit.k8s.io/v1)
kind: Policy
rules:
# Don't log read requests to certain resources
- level: None
  resources:
  - group: ""
    resources: ["events", "pods/log"]
# Log secrets at RequestResponse level
- level: RequestResponse
  resources:
  - group: ""
    resources: ["secrets", "configmaps"]
# Log everything else at Metadata level
- level: Metadata
```

API Server Flags:

- `--audit-policy-file=/etc/kubernetes/audit/policy.yaml`
- `--audit-log-path=/var/log/kubernetes/audit/audit.log`
- `--audit-log-maxage=30` (days to retain)
- `--audit-log-maxbackup=10` (files to keep)
- `--audit-log-maxsize=100` (MB per file)

Log Investigation:

Audit logs are JSON format. Use `jq` for parsing:

```
# Find secret access
cat audit.log | jq 'select(.objectRef.resource=="secrets")'

# Filter by user
cat audit.log | jq 'select(.user.username=="suspicious-user")'

# Find pod exec events
cat audit.log | jq 'select(.objectRef.subresource=="exec")'
```

Hands-on Recipes

- **6.1 Check Falco Logs for Suspicious Activity**

What it looks like in the exam: "Identify security events using Falco."

Fast path:

```
# Check Falco service status
systemctl status falco

# View Falco logs
journalctl -fu falco

# Search for specific events
journalctl -u falco | grep "shell was spawned"
journalctl -u falco | grep "Package management"
journalctl -u falco | grep "sensitive file"
```

Trigger test events:

```
# Shell into container (triggers rule)
kubectl exec -it <pod> -- /bin/bash

# Modify files (triggers rule)
kubectl exec <pod> -- apt update
```

⚠ **Trap:** Falco must be running on the node where the pod runs. Check which node with `kubectl get pod -o wide`.

✅ **Verify:** See corresponding log entries in Falco journal.

- **6.2 Modify Falco Rules**

What it looks like in the exam: "Change Falco rule output format or priority."

Fast path:

Edit `/etc/falco/falco_rules.local.yaml` (not the default rules file):

```
- rule: Terminal shell in container
  desc: A shell was spawned in a container
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
  output: >
    ALERT: Shell opened in container
    (user=%[user.name](http://user.name) container=%[container.name]
(http://container.name) image=%container.image.repository)
  priority: WARNING
```

Restart Falco:

```
systemctl restart falco
```

⚠ **Trap:** Changes to `/etc/falco/falco_rules.yaml` may be overwritten on update. Use `falco_rules.local.yaml`.

✅ **Verify:**

```
journalctl -fu falco | grep "ALERT"
```

- **6.3 Configure Kubernetes Audit Logging**

What it looks like in the exam: "Enable audit logging for secrets access."

Fast path:

Create audit policy `/etc/kubernetes/audit/policy.yaml`:

```
apiVersion: [audit.k8s.io/v1](http://audit.k8s.io/v1)
kind: Policy
rules:
# Log secrets at RequestResponse level
- level: RequestResponse
  resources:
  - group: ""
    resources: ["secrets"]
# Log pod exec/attach at Metadata level
- level: Metadata
  resources:
  - group: ""
    resources: ["pods/exec", "pods/attach"]
# Log all other requests at Metadata level
- level: Metadata
  omitStages:
  - RequestReceived
```

Update API server (`/etc/kubernetes/manifests/kube-apiserver.yaml`):

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --audit-policy-file=/etc/kubernetes/audit/policy.yaml
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
    - --audit-log-maxsize=100
    volumeMounts:
    - mountPath: /etc/kubernetes/audit
      name: audit-policy
      readOnly: true
    - mountPath: /var/log/kubernetes/audit
      name: audit-log
  volumes:
  - name: audit-policy
    hostPath:
      path: /etc/kubernetes/audit
      type: DirectoryOrCreate
  - name: audit-log
    hostPath:
      path: /var/log/kubernetes/audit
      type: DirectoryOrCreate
```

⚠ **Trap:** Create directories before API server restart. Wait for API server pod to restart.

✅ **Verify:**

```
ls /var/log/kubernetes/audit/
cat /var/log/kubernetes/audit/audit.log | jq . | head -50
```

- **6.4 Investigate Security Incident Using Audit Logs**

What it looks like in the exam: "Find who accessed secrets in last hour."

Fast path:

```
# Find secret access events
cat /var/log/kubernetes/audit/audit.log | jq
'select(.objectRef.resource=="secrets")'

# Filter by user
cat /var/log/kubernetes/audit/audit.log | jq
'select(.user.username=="suspicious-user")'

# Filter by time
cat /var/log/kubernetes/audit/audit.log | jq
'select(.requestReceivedTimestamp > "2026-01-13T00:00:00Z")'

# Find pod exec events
cat /var/log/kubernetes/audit/audit.log | jq
'select(.objectRef.subresource=="exec")'
```

⚠ **Trap:** Audit logs are JSON—use `jq` for parsing. Large log files may need `grep` first.

✅ **Verify:** Output shows relevant events with user, resource, and timestamp.

- **6.5 Ensure Container Immutability**

What it looks like in the exam: "Configure pod so container filesystem cannot be modified."

Fast path:

```
apiVersion: v1
kind: Pod
metadata:
  name: immutable-pod
spec:
  containers:
  - name: app
    image: [gcr.io/distroless/static:nonroot]
(http://gcr.io/distroless/static:nonroot)
    securityContext:
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      allowPrivilegeEscalation: false
    command: ["/bin/sleep", "3600"]
    volumeMounts:
    - name: tmp
      mountPath: /tmp
  volumes:
  - name: tmp
    emptyDir: {}
```

⚠ **Trap:** Some apps need writable temp dirs—use emptyDir volumes for those paths only.

✅ **Verify:**

```
kubectl exec immutable-pod -- touch /test
# Should fail: Read-only file system
```

Timed Labs (Domain 6)

Lab 6.1 (10 min): Shell into a container. Find the corresponding Falco alert in logs. Identify user, container, and timestamp.

Lab 6.2 (15 min): Configure audit logging to capture secrets access at RequestResponse level. Access a secret. Find the event in audit logs.

Lab 6.3 (8 min): Create immutable pod using distroless image with read-only filesystem. Verify no write operations are possible.

Troubleshooting Patterns

Symptom	Likely Cause	Quick Fix
Falco not logging	Service not running	`systemctl start falco`
Audit logs empty	Policy not loaded	Check API server args and paths
API server crash after audit config	Invalid YAML	Validate policy with `kubectl --dry-run`
No Falco events for pod	Wrong node	Check `kubectl get pod -o wide`

Mini Review (Domain 6)

Q1: Where are Falco rules stored?

A1: `/etc/falco/falco_rules.yaml` and `/etc/falco/falco_rules.local.yaml`

Q2: How do you view Falco logs?

A2: `journalctl -fu falco`

Q3: What are the audit log levels?

A3: None, Metadata, Request, RequestResponse

Q4: Where is the audit policy file specified?

A4: API server flag `--audit-policy-file`

Q5: How do you restart Falco after rule changes?

A5: `systemctl restart falco`

Q6: What makes a container immutable?

A6: readOnlyRootFilesystem, distroless image, config via volumes

Q7: What tool parses JSON audit logs?

A7: `jq`

Q8: Where are audit logs stored?

A8: Path specified by `--audit-log-path` (e.g., `/var/log/kubernetes/audit/audit.log`)

Q9: What Falco file should you edit for custom rules?

A9: `/etc/falco/falco_rules.local.yaml`

Q10: What securityContext field prevents filesystem writes?

A10: `readOnlyRootFilesystem: true`

Gaps vs 2026 Curriculum

The following topics are in the **2024-2026 CKS curriculum** but **not covered (or minimally covered)** in Muschko's 2023 book:

Gap 1: Software Bill of Materials (SBOM)

Why it matters: SBOMs provide transparency into software components, enabling vulnerability tracking and regulatory compliance (required by US Executive Order 14028).

Mini-lesson:

What is SBOM? A complete inventory of all components, libraries, and dependencies in software. Two main formats:

- **SPDX** (ISO standard): Focus on licensing and compliance
- **CycloneDX** (OWASP): Focus on security and vulnerability tracking

Tools:

- **bom**: Kubernetes SIG Release tool for SPDX generation
- **Trivy**: Generates both CycloneDX and SPDX formats
- **Syft**: Anchore's SBOM generator

Practice Task:

```
# Install bom
go install [sigs.k8s.io/bom/cmd/bom@latest]
(http://sigs.k8s.io/bom/cmd/bom@latest)

# Generate SBOM for an image
bom generate --image nginx:1.23.1 --format spdx --output nginx-sbom.spdx

# Using Trivy for CycloneDX format
trivy image --format cyclonedx --output sbom.cdx nginx:1.23.1
```

✅ **Verify:** SBOM contains package names, versions, and licenses.

📚 **LOOK UP IN OFFICIAL DOCS:**

- Trivy SBOM documentation
- Kubernetes SIG Release bom tool
- SPDX specification
- CycloneDX specification

Gap 2: KubeLinter Static Analysis

Why it matters: KubeLinter catches security misconfigurations before deployment, complementing Kubesec with broader Kubernetes-specific checks.

Mini-lesson:

What is KubeLinter? Open-source static analysis tool from StackRox (Red Hat) that checks:

- Security best practices
- Production readiness
- Kubernetes anti-patterns

Key differences from Kubesec:

- KubeLinter checks Helm charts natively
- More checks (19+ built-in)
- Configurable check inclusion/exclusion
- CI/CD friendly (non-zero exit on findings)

Practice Task:

```
# Install
curl -Lo kube-linter https://github.com/stackrox/kube-
linter/releases/latest/download/kube-linter-linux
chmod +x kube-linter
sudo mv kube-linter /usr/local/bin/

# Scan manifests
kube-linter lint pod.yaml
kube-linter lint ./manifests/

# List available checks
kube-linter checks list
```

✅ **Verify:** No errors returned from kube-linter.

📚 **LOOK UP IN OFFICIAL DOCS:**

- KubeLinter GitHub: https://github.com/stackrox/kube-linter
- KubeLinter checks list
- Configuration file format

Gap 3: Cilium Pod-to-Pod Encryption

Why it matters: Cilium provides transparent encryption using WireGuard or IPsec at the CNI level, without requiring application changes or service mesh sidecars.

Mini-lesson:

What is Cilium Encryption? Node-to-node encryption using:

- **WireGuard**: Modern, fast, simple (recommended)
- **IPsec**: Traditional, widely supported

Key concepts:

- Encryption happens at L3/L4 (network layer)
- No application changes needed
- Works with any workload
- Lower overhead than service mesh mTLS

Practice Task:

```
# Check Cilium status
kubectl -n kube-system exec ds/cilium -- cilium status

# Enable WireGuard encryption (during install)
helm install cilium cilium/cilium \
  --set encryption.enabled=true \
  --set encryption.type=wireguard

# Verify encryption
kubectl -n kube-system exec ds/cilium -- cilium status --verbose | grep
Encryption
kubectl -n kube-system exec ds/cilium -- cilium encrypt status

# For IPsec
helm install cilium cilium/cilium \
  --set encryption.enabled=true \
  --set encryption.type=ipsec \
  --set encryption.ipsec.key="$(echo -n 'random-key-here' | base64)"
```

✅ **Verify:** `cilium encrypt status` shows encryption enabled.

📚 **LOOK UP IN OFFICIAL DOCS:**

- Cilium Transparent Encryption
- WireGuard configuration
- IPsec configuration

Gap 4: Istio mTLS for Pod-to-Pod Encryption

Why it matters: Istio provides application-layer (L7) mTLS with automatic certificate rotation, authentication, and authorization policies.

Mini-lesson:

What is Istio mTLS? Mutual TLS between services where:

- Both client and server present certificates
- Certificates auto-rotated by Istio
- Traffic encrypted end-to-end
- Works with PeerAuthentication and AuthorizationPolicy

Key concepts:

- **PERMISSIVE mode**: Accept both plaintext and mTLS (migration)
- **STRICT mode**: Only accept mTLS (production)
- Sidecar injection required for pods

Practice Task:

```
# Label namespace for sidecar injection
kubectl label namespace default istio-injection=enabled

# Deploy workloads (sidecars auto-injected)
kubectl apply -f deployment.yaml

# Enable STRICT mTLS for namespace
kubectl apply -f - <<EOF
apiVersion: [security.istio.io/v1beta1](http://security.istio.io/v1beta1)
kind: PeerAuthentication
metadata:
  name: default
  namespace: default
spec:
  mtls:
    mode: STRICT
EOF

# Verify mTLS
istioctl x describe pod <pod-name>
kubectl exec <pod> -c istio-proxy -- curl [localhost:15000/config_dump]
(http://localhost:15000/config_dump) | grep -i tls
```

✅ **Verify:** Traffic between injected pods shows mTLS enabled.

📚 **LOOK UP IN OFFICIAL DOCS:**

- Istio PeerAuthentication
- Istio Security (mTLS)
- Sidecar injection

7-Day Crash Plan

Day 1: Cluster Setup (Domain 1)

- Morning: NetworkPolicies (deny-all, allow-specific)
- Afternoon: kube-bench, CIS benchmarks
- Evening: Ingress TLS, metadata protection
- Practice: Labs 1.1, 1.2, 1.3

Day 2: Cluster Hardening (Domain 2)

- Morning: RBAC deep dive (Role, ClusterRole, bindings)
- Afternoon: Service account security, token management
- Evening: Kubernetes upgrade process
- Practice: Labs 2.1, 2.2, 2.3

Day 3: System Hardening (Domain 3)

- Morning: AppArmor profiles and enforcement
- Afternoon: seccomp profiles (RuntimeDefault, Localhost)
- Evening: Network hardening, firewall rules
- Practice: Labs 3.1, 3.2, 3.3

Day 4: Microservice Vulnerabilities (Domain 4)

- Morning: Security contexts, PSA/PSS
- Afternoon: OPA Gatekeeper policies
- Evening: Secrets encryption, RuntimeClass
- Practice: Labs 4.1, 4.2, 4.3

Day 5: Supply Chain Security (Domain 5)

- Morning: Trivy scanning, SBOM generation
- Afternoon: Kubesec, KubeLinter static analysis
- Evening: Image signing, registry restriction
- Practice: Labs 5.1, 5.2, 5.3

Day 6: Monitoring/Runtime (Domain 6)

- Morning: Falco installation, rules, log analysis
- Afternoon: Audit logging configuration
- Evening: Container immutability, incident investigation
- Practice: Labs 6.1, 6.2, 6.3

Day 7: Full Review + Mock Exam

- Morning: Review all Mini Reviews (60 questions)
- Afternoon: Time yourself on all Timed Labs
- Evening: killer.sh practice exam
- Focus: Commands speed, file locations, troubleshooting

Key Commands Cheat Sheet (Memorize)

```
# RBAC
kubectl auth can-i --list --as=<user>
kubectl create role/rolebinding/clusterrole/clusterrolebinding

# Security
aa-status  # AppArmor
journalctl -fu falco  # Falco logs
trivy image <image>  # Vulnerability scan
kube-linter lint <manifest>  # Static analysis
kubesec scan <manifest>  # Security score

# Audit
cat /var/log/kubernetes/audit/audit.log | jq '.'

# Quick edits
kubectl edit <resource>
vim /etc/kubernetes/manifests/kube-apiserver.yaml

# Debugging
crictl ps  # Container runtime
crictl logs <id>  # Container logs
kubectl logs -n kube-system kube-apiserver-<node>
```

Good luck on your CKS exam! 🎯

Book Reference: Muschko, Benjamin. Certified Kubernetes Security Specialist (CKS) Study Guide. O'Reilly Media, 2023.