

CKS Exam Quick Steps Reference

Exam: 2 hours | 15-20 Questions | 67% Pass | Hands-on

This is your visual step-by-step cheat sheet

❖ CKS Domain Weights

Domain	Weight	Key Topics
1. Cluster Setup	15%	NetworkPolicy, CIS/kube-bench, Ingress TLS, Metadata Protection
2. Cluster Hardening	15%	RBAC, ServiceAccount, K8s Upgrade
3. System Hardening	10%	AppArmor, Seccomp, OS Hardening
4. Minimize Microservice Vulns	20%	PSA, SecurityContext, Secrets Encryption, RuntimeClass, Gatekeeper
5. Supply Chain Security	20%	Trivy, Kubesec, KubeLinter, SBOM, ImagePolicyWebhook
6. Monitoring & Runtime	20%	Falco, Audit Logs, Container Immutability

❖ 🔐 First Things First - Set Aliases!

```
alias k=kubectl
alias kn='kubectl config set-context --current --namespace'
export do="--dry-run=client -o yaml"
source <(kubectl completion bash)
complete -o default -F __start_kubectl k
```

1 NetworkPolicy - Default Deny

1. Create namespace
kubectl create ns <ns>
2. Create NetworkPolicy YAML:
 - podSelector: {} <- ALL pods
 - policyTypes: [Ingress, Egress]
 - NO rules = DENY ALL
3. kubectl apply -f <file>
4. kubectl get netpol -n <ns>

Key YAML:

```
spec:
  podSelector: {}
  policyTypes: [Ingress, Egress]
  # No rules = deny all
```

2 NetworkPolicy - Allow Specific

1. Identify source/dest pods (labels)
2. Create policy with:
 - podSelector: target pods
 - ingress.from: source pods
 - egress.to: dest pods
 - ALWAYS add DNS (port 53 UDP/TCP)
3. kubectl apply -f <file>
4. Test: kubectl exec <pod> -- wget

DNS Egress (always add):

```
egress:
  - ports:
    - protocol: UDP
      port: 53
    - protocol: TCP
      port: 53
```

3 CIS Benchmark / kube-bench

- ```
1. ssh controlplane
2. kube-bench run --targets=master
3. Fix API server:
 vim /etc/kubernetes/manifests/
 kube-apiserver.yaml
 - --anonymous-auth=false
 - --profiling=false
 - --authorization-mode=Node, RBAC

4. Fix kubelet (on nodes):
 vim /var/lib/kubelet/config.yaml
 authentication:
 anonymous:
 enabled: false
 authorization:
 mode: Webhook
 readOnlyPort: 0

5. sudo systemctl restart kubelet
6. Re-run kube-bench to verify
```

---

### 4 RBAC - Role & RoleBinding

---

1. Create ServiceAccount:  
kubectl create sa <sa> -n <ns>
2. Create Role (namespace-scoped):  
kubectl create role <role>  
--verb=get,list,create  
--resource=pods,deployments  
-n <ns>
3. Create RoleBinding:  
kubectl create rolebinding <rb>  
--role=<role>  
--serviceaccount=<ns>:<sa>  
-n <ns>
4. Test:  
kubectl auth can-i create pods  
--as=system:serviceaccount:  
<ns>:<sa> -n <ns>

#### API Groups:

| Group                                           | Resources |
|-------------------------------------------------|-----------|
| ...<br>pods, services, secrets, configmaps      |           |
| apps<br>deployments, daemonsets, statefulsets   |           |
| networking.k8s.io<br>networkpolicies, ingresses |           |

## 5 RBAC - ClusterRole (cluster-wide)

```
1. kubectl create clusterrole <cr>
 --verb=get,list,watch
 --resource=nodes,pods

2. kubectl create clusterrolebinding
 <crb> --clusterrole=<cr>
 --serviceaccount=<ns>:<sa>

3. Test:
 kubectl auth can-i list nodes
 --as=system:serviceaccount:
 <ns>:<sa>
```

## 6 ServiceAccount Security

```
1. Create SA with no auto-mount:
 automountServiceAccountToken: false

2. Update Pod/Deployment spec:
 - serviceAccountName: <sa>
 - automountServiceAccountToken:false

3. Create minimal Role (least priv)
 - NO secrets unless required

4. Verify no token:
 kubectl exec <pod> -- ls
 /var/run/secrets/kubernetes.io/
 -> Should fail (no token mounted)
```

## 7 AppArmor Profiles

1. ssh <node>
2. Check profile loaded:  
sudo aa-status | grep <profile>
3. Load if needed:  
sudo apparmor\_parser -r  
/etc/apparmor.d/<profile>
4. Add to Pod spec:  
containers:  
- securityContext:  
  appArmorProfile:  
    type: Localhost  
    localhostProfile: <profile>
5. kubectl apply & verify

**Profile Types:** [RuntimeDefault](#) | [Localhost](#) | [Unconfined](#)

---

## 8 Seccomp Profiles

---

1. RuntimeDefault (easiest):  
spec:  
  securityContext:  
    seccompProfile:  
      type: RuntimeDefault
2. Custom Localhost profile:  
- Profile at:  
  /var/lib/kubelet/seccomp/<file>  
- Pod spec:  
  seccompProfile:  
    type: Localhost  
    localhostProfile: <file>.json
3. kubectl apply & verify running

## 9 Pod Security Admission (PSA)

1. Label namespace:  
kubectl label ns <ns>  
pod-security.kubernetes.io/  
enforce=restricted
2. Restricted Pod MUST have:  
[x] runAsNonRoot: true  
[x] seccompProfile: RuntimeDefault  
[x] allowPrivilegeEscalation: false  
[x] capabilities.drop: ["ALL"]  
[x] No hostPath, hostNetwork, hostPID  
[x] No privileged containers
3. Best practices (add for nginx etc):  
– readOnlyRootFilesystem: true  
– emptyDir for /tmp, /var/cache,  
/var/run (writable paths)
4. Test: run non-compliant pod  
→ Should be rejected

**Levels:** privileged | baseline | restricted

**Modes:** enforce | warn | audit

## 10 Secrets Encryption at Rest

1. Generate key:  
`head -c 32 /dev/urandom | base64`
2. Create `/etc/kubernetes/encryption-config.yaml`:  
`!! aescbc FIRST, identity LAST !!`
3. Edit `kube-apiserver.yaml`:  
`--encryption-provider-config=/etc/kubernetes/encryption-config.yaml`  
+ volumeMounts + volumes
4. Wait for API restart:  
`watch "crictl ps | grep apiserver"`
5. Re-encrypt existing secrets:  
`kubectl get secrets -A -o json | kubectl replace -f -`
6. Verify in etcd (encrypted):  
`etcdctl get /registry/secrets/...`  
-> Should start with k8s:enc:aescbc

## 1 1 SecurityContext Hardening

Add to Pod/Container spec:

```
spec:
 securityContext: # Pod-level
 runAsNonRoot: true
 runAsUser: 1000
 fsGroup: 1000
 seccompProfile:
 type: RuntimeDefault
 containers:
 - securityContext: # Container
 allowPrivilegeEscalation: false
 readOnlyRootFilesystem: true
 capabilities:
 drop: ["ALL"]
```

Add emptyDir for writable paths

## 1 2 Trivy Image Scanning

1. Scan for HIGH/CRITICAL:  
`trivy image --severity HIGH,CRITICAL  
<image>:<tag>`
2. Compare images:  
`trivy image nginx:1.19 > old.txt  
trivy image nginx:alpine > new.txt`
3. Choose image with fewer vulns
4. Update deployment:  
`kubectl set image deploy/<name>  
<container>=<safer-image>`

**Quick flags:** `--severity` | `-q` (quiet) | `--ignore-unfixed`

## 1 3 Kubesec Analysis

1. Scan manifest:  
kubesec scan <file>.yaml
2. Check score (target: 8+)
3. Add security features:  
+1 runAsNonRoot: true  
+1 readOnlyRootFilesystem: true  
+1 capabilities.drop: ALL  
+1 resources.limits  
+1 automountServiceAccountToken: false
4. Rescan and verify score >= 8

## 1 4 Falco Runtime Security

1. ssh <node> (where Falco runs)
2. Create rule file:  
/etc/falco/rules.d/<name>.yaml
3. Rule structure:
  - rule: <name>
  - desc: <description>
  - condition: <expression>
  - output: <message with %fields>
  - priority: WARNING|ALERT|etc
4. Restart Falco:  
sudo systemctl restart  
  falco-modern-bpf
5. Trigger & check logs:  
kubectl exec <pod> -- /bin/sh  
journalctl -u falco-modern-bpf -f

**Common macros:**

```
- macro: spawned_process
 condition: evt.type in (execve, execveat)
- macro: container
 condition: container.id != host
```

**Output fields:** %proc.name | %container.name | %k8s.pod.name | %user.name

---

## 1 5 Audit Logs

1. Create audit policy:  
`/etc/kubernetes/audit-policy.yaml`
  - level: None|Metadata|Request|  
RequestResponse
  - resources: [secrets, pods, etc]
  - verbs: [create, delete, etc]
2. Edit kube-apiserver.yaml:  
`--audit-policy-file=<path>`  
`--audit-log-path=<log-path>`  
`--audit-log-maxage=30`  
+ volumeMounts + volumes
3. `mkdir -p /var/log/kubernetes/audit`
4. Wait for API restart
5. Test & find entry:  
`kubectl create secret ...`  
`grep <secret> <audit-log>`

**Audit Levels:** None → Metadata → Request → RequestResponse

---

## 1 6 RuntimeClass / gVisor

1. Verify RuntimeClass exists:  
kubectl get runtimeclass gvisor
2. Add to Pod spec:  
spec:  
  runtimeClassName: gvisor
3. kubectl apply & verify running
4. Verify gVisor:  
kubectl exec <pod> -- dmesg | head  
-> Should show gVisor kernel

## 1 7 ImagePolicyWebhook

1. Create webhook kubeconfig:  
/etc/kubernetes/admission/  
  image-policy-kubeconfig.yaml
2. Create admission config:  
/etc/kubernetes/admission/  
  admission-config.yaml  
  - defaultAllow: false (DENY if down)
3. Edit kube-apiserver.yaml:  
  --enable-admission-plugins=  
    NodeRestriction,ImagePolicyWebhook  
  --admission-control-config-file=  
    <admission-config-path>  
  + volumeMounts + volumes
4. Wait & test allowed/denied images

## 1 8 Binary Verification

1. Get cluster version:  
kubectl version
2. Download official checksum:  
curl -LO https://dl.k8s.io/release/  
<version>/bin/linux/amd64/  
kubectl.sha512
3. Calculate local checksum:  
sha512sum \$(which kubectl)
4. Compare:  
MATCH -> GENUINE  
NO MATCH -> TAMPERED
5. Save conclusion to file

## 1 9 Node Metadata Protection

1. Create NetworkPolicy to block  
169.254.169.254/32
2. Policy structure:  
spec:  
  podSelector: {}  
  policyTypes: [Egress]  
  egress:  
    - to:  
      - ipBlock:  
        cidr: 0.0.0.0/0  
        except:  
          - 169.254.169.254/32  
    - ports: [UDP/TCP 53] # DNS
3. Test metadata access -> should fail  
wget http://169.254.169.254/...

## 2 0 Ingress TLS

1. Generate cert:  

```
openssl req -x509 -nodes -days 365
-newkey rsa:2048
-keyout tls.key -out tls.crt
-subj "/CN=<domain>"
```
2. Create TLS secret:  

```
kubectl create secret tls <name>
--cert=tls.crt --key=tls.key
-n <ns>
```
3. Create Ingress with TLS:  

```
spec:
 tls:
 - hosts: [<domain>]
 secretName: <tls-secret>
 rules: ...
```
4. kubectl apply & verify

## 2 1 OPA Gatekeeper (Policy Enforcement)

1. Verify Gatekeeper installed:  

```
kubectl get pods -n gatekeeper-system
```
2. Create ConstraintTemplate (policy):  

```
apiVersion: templates.gatekeeper.sh
kind: ConstraintTemplate
spec.targets[].rego: <policy-logic>
```
3. Create Constraint (apply policy):  

```
apiVersion: constraints.gatekeeper.sh
kind: <TemplateName>
spec.match.kinds: [Pod, Deployment]
spec.parameters: <values>
```
4. Apply Template FIRST, then Constraint
5. Test: create violating resource  
-> Should be rejected

**Common Use Cases:**

- Restrict allowed image registries
  - Require resource limits on pods
  - Enforce required labels
- 

## 2 2 SBOM (Software Bill of Materials)

---

1. Generate SBOM with Trivy:  
`trivy image --format cyclonedx  
-o sbom.json <image>`
2. Or generate SPDX format:  
`trivy image --format spdx-json  
-o sbom.spdx.json <image>`
3. Generate with bom tool:  
`bom generate --image <image>  
--format spdx -o sbom.spdx`
4. Scan existing SBOM for vulns:  
`trivy sbom sbom.json`

**Formats:** [CycloneDX](#) (OWASP) | [SPDX](#) (ISO standard)

---

## 2 3 KubeLinter (Static Analysis)

---

1. Scan manifest:  
`kube-linter lint <file>.yaml`
2. Scan directory:  
`kube-linter lint ./manifests/`
3. Scan Helm chart:  
`kube-linter lint ./my-chart/`
4. List available checks:  
`kube-linter checks list`
5. Run specific checks only:  
`kube-linter lint --include "run-as-non-root,no-read-only-root-fs" <file>.yaml`
6. Fix issues and rescan

**Note:** Non-zero exit code on findings (CI/CD friendly)

---

## 2 4 Kubernetes Version Upgrade

---

1. Drain control plane node:  
kubectl drain <node>  
--ignore-daemonsets
2. Upgrade kubeadm FIRST:  
apt-get update  
apt-get install -y kubeadm=1.XX.0-\*
3. Plan and apply upgrade:  
kubeadm upgrade plan  
kubeadm upgrade apply v1.XX.0
4. Upgrade kubelet & kubectl:  
apt-get install -y  
kubelet=1.XX.0-\* kubectl=1.XX.0-\*
5. Restart kubelet:  
systemctl daemon-reload  
systemctl restart kubelet
6. Uncordon node:  
kubectl uncordon <node>

**Rule:** NEVER skip minor versions (1.32->1.33->1.34)

---

## 2 5 mTLS / Pod-to-Pod Encryption

---

**ISTIO mTLS:**

1. Label ns for sidecar injection:  
kubectl label ns <ns>  
istio-injection=enabled
2. Create PeerAuthentication:  
apiVersion: security.istio.io/v1beta1  
kind: PeerAuthentication  
spec.mtls.mode: STRICT
3. Verify:  
istioctl x describe pod <pod>

**CILIUM WireGuard:**

1. Enable during install:  
helm install cilium --set  
encryption.enabled=true  
encryption.type=wireguard
2. Verify:  
cilium encrypt status

**Modes:** **STRICT** (mTLS only) | **PERMISSIVE** (both)

---

## Critical File Paths

---

| Path                         | Purpose                               |
|------------------------------|---------------------------------------|
| /etc/kubernetes/manifests/   | Static pod manifests (API, etcd, etc) |
| /var/lib/kubelet/config.yaml | Kubelet configuration                 |
| /var/lib/kubelet/seccomp/    | Seccomp profiles                      |
| /etc/apparmor.d/             | AppArmor profiles                     |
| /etc/falco/rules.d/          | Custom Falco rules                    |
| /etc/kubernetes/pki/         | Cluster certificates                  |
| /etc/kubernetes/audit/       | Audit policy location                 |
| /var/log/kubernetes/audit/   | Audit log files                       |

---

# ⚡ Quick Commands Cheat Sheet

```
RBAC testing
kubectl auth can-i <verb> <resource> --as=system:serviceaccount:<ns>:<sa>
-n <ns>
kubectl auth can-i --list --as=<user> -n <ns>

Debug
kubectl describe pod <pod> -n <ns>
kubectl logs <pod> -n <ns>
kubectl exec -it <pod> -n <ns> -- /bin/sh

Watch API server restart
watch "crictl ps | grep kube-apiserver"

etcd access
ETCDCTL_API=3 etcdctl --cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key get <key>

Falco logs
journalctl -u falco-modern-bpf -f

AppArmor
aa-status
apparmor_parser -r /etc/apparmor.d/<profile>

Container inspection
crictl ps
crictl inspect <container-id>
```

## ✖ Common Mistakes to AVOID

| Mistake                                           | Fix                                                                 |
|---------------------------------------------------|---------------------------------------------------------------------|
| Forgot <code>-n &lt;namespace&gt;</code>          | ALWAYS specify namespace                                            |
| Didn't wait for API restart                       | <code>watch crictl ps \  grep api</code>                            |
| Wrong output file path                            | Double-check question paths                                         |
| Missing DNS in NetworkPolicy                      | Add port 53 UDP/TCP egress                                          |
| Missing seccomp for PSA                           | Add <code>seccompProfile.type: RuntimeDefault</code>                |
| Missing <code>capabilities.drop: ALL</code>       | Required for PSA restricted                                         |
| Put <code>identity: {}</code> first in encryption | Encryption provider MUST be first                                   |
| Forgot to re-encrypt secrets                      | <code>kubectl get secrets -A -o json \  kubectl replace -f -</code> |

## 🎯 Exam Day Flow

1. Set aliases FIRST
2. Read question FULLY (note ns, paths, names)
3. Use imperative commands when possible
4. VERIFY after each step
5. Flag hard questions → skip → return later
6. Check output paths match exactly
7. Watch for restart requirements

Good luck on your CKS exam! 🚀