1 KCSA

Generated on: 2026-01-13 15:04:20 **Version:** 1.0

1.1 Table of Contents

1.2 Overview

The **Kubernetes and Cloud Native Security Associate (KCSA)** exam demonstrates a user's foundational knowledge and skills in security technologies in the cloud native ecosystem.

1.3 Exam Overview

Detail	Information
Exam Format	Multiple Choice
Number of Questions	60
Duration	90 minutes
Passing Score	75%
Certification Validity	3 years
Cost	$250 USD
Retake Policy	1 free retake

1.4 Exam Domains & Weights

Domain	Weight
Overview of Cloud Native Security	14%
Kubernetes Cluster Component Security	22%
Kubernetes Security Fundamentals	22%
Kubernetes Threat Model	16%
Platform Security	16%
Compliance and Security Frameworks	10%

1.5 Prerequisites

- Basic understanding of Kubernetes concepts (KCNA recommended)
- Familiarity with Linux command line
- General security concepts knowledge

1.6 Study Resources

1.6.1 Official Resources

- KCSA Exam Curriculum
- Kubernetes Security Documentation
- CNCF Security Whitepaper

1.6.2 Recommended Courses

- Kubernetes Security Essentials (LFS260)
- Securing Kubernetes (LFS482)

1.6.3 Practice Resources

- Kubernetes Security Best Practices
- OWASP Kubernetes Security Cheat Sheet

1.7 Quick Navigation

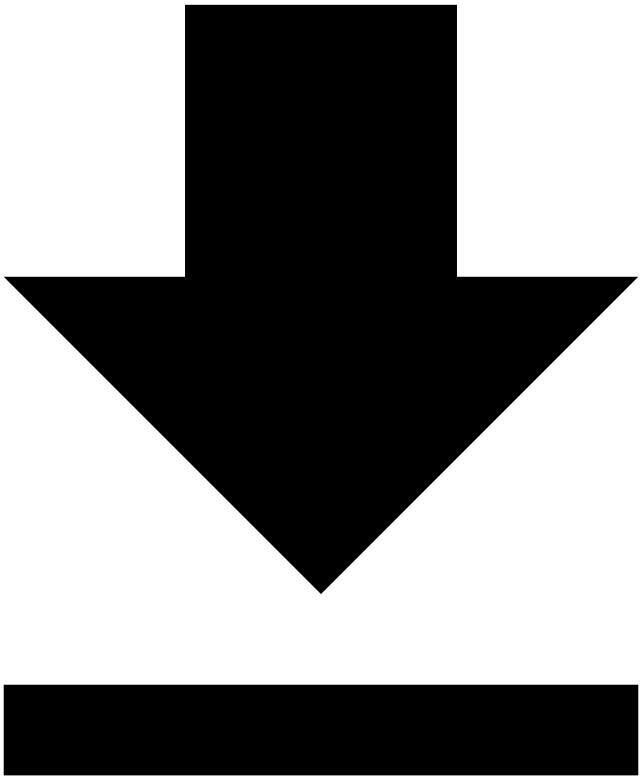
1.8 Exam Tips

1. **Understand the 4Cs of Cloud Native Security** - Cloud, Cluster, Container, Code
2. **Know RBAC thoroughly** - Roles, ClusterRoles, RoleBindings, ClusterRoleBindings
3. **Study Network Policies** - Ingress and egress rules
4. **Understand Pod Security** - Security contexts, Pod Security Standards
5. **Review common attack vectors** - STRIDE, MITRE ATT&CK for Kubernetes
6. **Time management** - 90 seconds per question on average

1.9 Registration

Register for KCSA Exam

1.10 Cloud Native Security Overview



Download PDF Version

This domain covers the foundational concepts of security in cloud native environments.

1.11 The 4Cs of Cloud Native Security

Cloud native security is built on four layers, each building upon the previous:

```
┌─────────────────────────────────────┐
│                Code                 │  ← Application security
├─────────────────────────────────────┤
│              Container              │  ← Container security
```

1.11.1 Cloud Layer

The foundation of your security posture:

- **Infrastructure security** - Physical and virtual infrastructure
- **Network security** - Firewalls, VPCs, security groups
- **Identity and Access Management (IAM)** - Cloud provider access controls
- **Data encryption** - At rest and in transit

1.11.2 Cluster Layer

Kubernetes-specific security:

- **API Server security** - Authentication, authorization
- **etcd security** - Encryption, access controls
- **Node security** - OS hardening, kubelet configuration
- **Network policies** - Pod-to-pod communication rules

1.11.3 Container Layer

Container runtime security:

- **Image security** - Vulnerability scanning, trusted registries
- **Runtime security** - Seccomp, AppArmor, SELinux
- **Resource limits** - CPU, memory constraints
- **Read-only filesystems** - Immutable containers

1.11.4 Code Layer

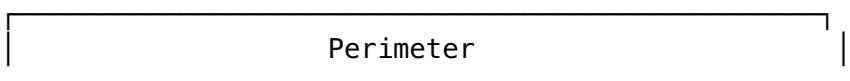
Application-level security:

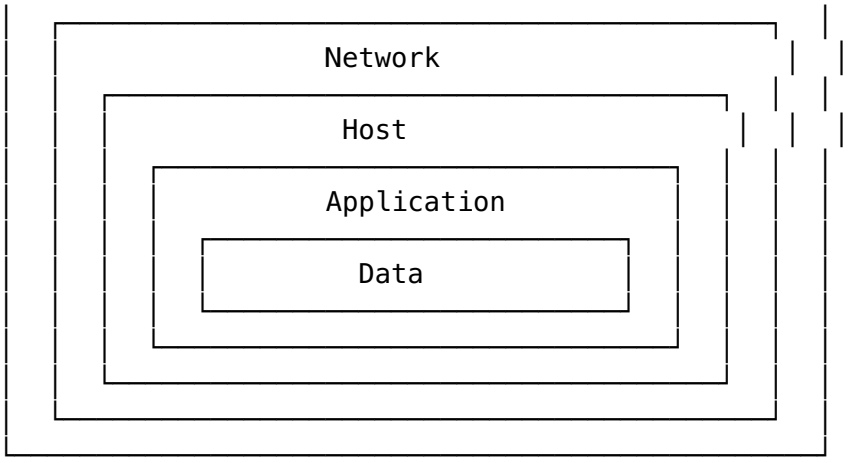
- **Secure coding practices** - Input validation, output encoding
- **Dependency management** - Vulnerability scanning
- **Secrets management** - No hardcoded credentials
- **TLS/mTLS** - Encrypted communications

1.12 Cloud Native Security Principles

1.12.1 Defense in Depth

Multiple layers of security controls:





1.12.2 Least Privilege

Grant only the minimum permissions necessary:

- Use specific RBAC roles instead of cluster-admin
- Limit service account permissions
- Restrict network access with Network Policies
- Use read-only file systems where possible

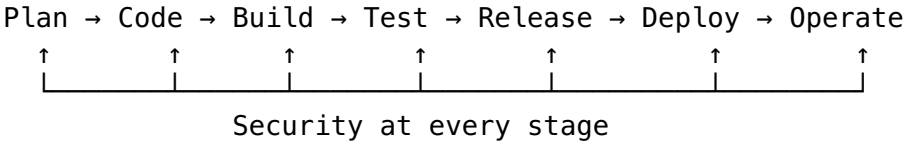
1.12.3 Zero Trust

Never trust, always verify:

- Authenticate all requests
- Authorize based on identity and context
- Encrypt all communications
- Continuously validate security posture

1.12.4 Shift Left Security

Integrate security early in the development lifecycle:



1.13 Cloud Provider Security

1.13.1 Shared Responsibility Model

Layer	Cloud Provider	Customer
Physical Security	✓	
Network Infrastructure	✓	
Hypervisor		

Layer	Cloud Provider	Customer
	✓	
Operating System	Managed K8s: ✓	Self-managed: ✓
Kubernetes Control Plane	Managed K8s: ✓	Self-managed: ✓
Kubernetes Worker Nodes		✓
Applications		✓
Data		✓

1.13.2 Cloud Security Services

Service Type	AWS	GCP	Azure
IAM	IAM	Cloud IAM	Azure AD
Secrets	Secrets Manager	Secret Manager	Key Vault
KMS	KMS	Cloud KMS	Key Vault
Security Scanning	Inspector	Security Command Center	Defender

1.14 Security Observability

1.14.1 Logging

Essential logs to collect:

- API Server audit logs
- Container runtime logs
- Application logs
- Network flow logs

1.14.2 Monitoring

Key security metrics:

- Failed authentication attempts
- RBAC denials
- Network policy violations
- Resource quota breaches

1.14.3 Alerting

Critical security alerts:

- Privileged container creation

- Secrets access patterns
- Unusual API activity
- Pod security violations

1.15 Key Concepts to Remember

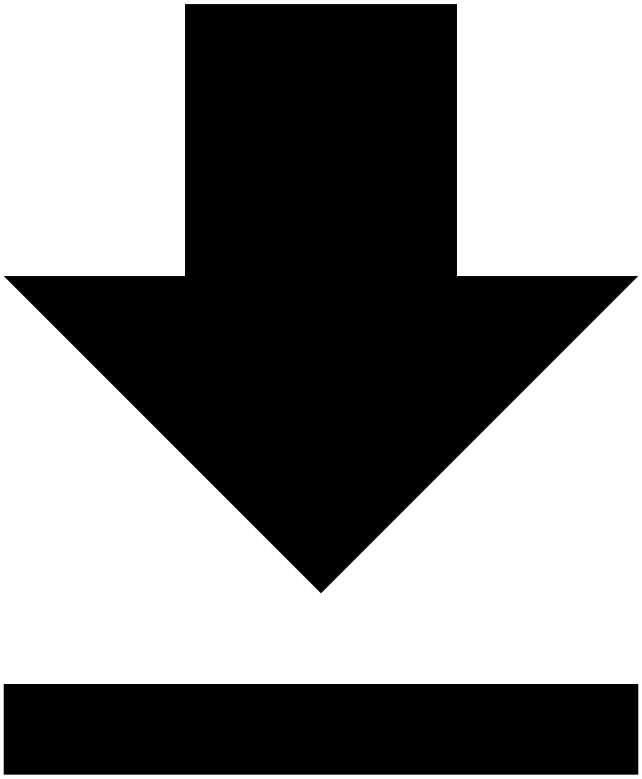
1. **4Cs model** - Cloud, Cluster, Container, Code
2. **Defense in depth** - Multiple security layers
3. **Least privilege** - Minimum necessary permissions
4. **Zero trust** - Never trust, always verify
5. **Shared responsibility** - Know what you're responsible for

1.16 Practice Questions

1. What are the 4Cs of Cloud Native Security?
2. In the shared responsibility model, who is responsible for application security?
3. What does "shift left" mean in the context of security?
4. Name three principles of zero trust security.
5. What is defense in depth?

Back to KCSA Overview | Next: Kubernetes Cluster Component Security →

1.17 Cluster Component Security



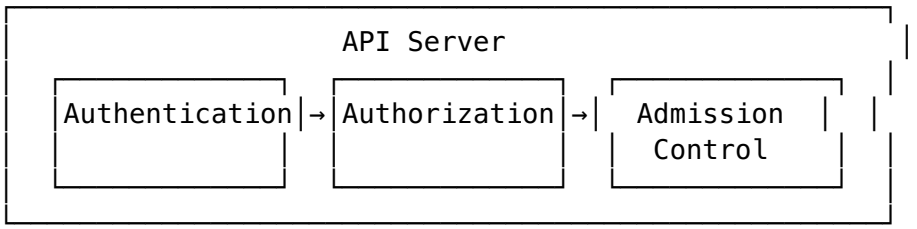
Download PDF Version

This domain covers securing the core components of a Kubernetes cluster.

1.18 Control Plane Security

1.18.1 API Server Security

The API Server is the front door to your cluster:



Security configurations:

- Enable TLS for all communications
- Use strong authentication methods
- Configure appropriate authorization modes
- Enable audit logging
- Restrict anonymous access

```
# API Server security flags
--anonymous-auth=false
--authorization-mode=Node,RBAC
--enable-admission-plugins=NodeRestriction,PodSecurity
--audit-log-path=/var/log/kubernetes/audit.log
--tls-cert-file=/etc/kubernetes/pki/apiserver.crt
--tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

1.18.2 etcd Security

etcd stores all cluster state and secrets:

Best practices:

- Enable TLS for client and peer communication
- Encrypt data at rest
- Restrict access to etcd
- Regular backups with encryption
- Run etcd on dedicated nodes

```
# etcd security configuration
--cert-file=/etc/kubernetes/pki/etcd/server.crt
--key-file=/etc/kubernetes/pki/etcd/server.key
--client-cert-auth=true
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
--peer-key-file=/etc/kubernetes/pki/etcd/peer.key
```

1.18.3 Controller Manager Security

Security configurations:

- Use service account credentials
- Enable TLS
- Rotate service account tokens

1.18.4 Scheduler Security

Security configurations:

- Enable TLS
- Use secure binding address
- Configure authentication

1.19 Node Security

1.19.1 Kubelet Security

The kubelet runs on every node:

```
# Kubelet security configuration
--anonymous-auth=false
--authorization-mode=Webhook
--client-ca-file=/etc/kubernetes/pki/ca.crt
--tls-cert-file=/etc/kubernetes/pki/kubelet.crt
--tls-private-key-file=/etc/kubernetes/pki/kubelet.key
--rotate-certificates=true
--protect-kernel-defaults=true
--read-only-port=0
```

Key security settings:

Setting	Recommended Value	Purpose
`--anonymous-auth`	false	Disable anonymous access
`--authorization-mode`	Webhook	Use API server for authorization
`--read-only-port`	0	Disable read-only port
`--protect-kernel-defaults`	true	Protect kernel settings

1.19.2 Container Runtime Security

containerd security:

- Use seccomp profiles
- Enable AppArmor/SELinux
- Configure runtime class

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: secure-runtime
handler: runsc  # gVisor
```

1.19.3 Node Hardening

OS-level security:

- Minimize installed packages
- Apply security patches regularly
- Configure firewall rules
- Enable audit logging

- Use immutable infrastructure

1.20 Authentication Methods

1.20.1 X.509 Client Certificates

```
# Generate client certificate
openssl genrsa -out user.key 2048
openssl req -new -key user.key -out user.csr -subj "/CN=user/
        O=group"
# Sign with cluster CA
```

1.20.2 Service Account Tokens

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
automountServiceAccountToken: false  # Disable auto-mount
```

1.20.3 OpenID Connect (OIDC)

```
# API Server OIDC configuration
--oidc-issuer-url=https://accounts.google.com
--oidc-client-id=my-client-id
--oidc-username-claim=email
--oidc-groups-claim=groups
```

1.20.4 Webhook Token Authentication

External authentication service integration.

1.21 Authorization Modes

1.21.1 Node Authorization

Authorizes API requests made by kubelets.

1.21.2 RBAC (Role-Based Access Control)

Most commonly used authorization mode:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
```

```yaml
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

1.21.3 ABAC (Attribute-Based Access Control)

Policy file-based authorization (less common).

1.21.4 Webhook Authorization

External authorization service.

1.22 Admission Controllers

1.22.1 Built-in Admission Controllers

Controller	Purpose
PodSecurity	Enforce Pod Security Standards
NodeRestriction	Limit kubelet permissions
ResourceQuota	Enforce resource limits
LimitRanger	Set default resource limits
ServiceAccount	Automate service account management

1.22.2 Validating vs Mutating

```
Request → Mutating Webhooks → Validating Webhooks → etcd
              (modify)            (accept/reject)
```

1.22.3 Pod Security Admission

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: secure-namespace
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

1.23 Secrets Management

1.23.1 Kubernetes Secrets

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  password: cGFzc3dvcmQ=  # base64 encoded
```

Security considerations:

- Secrets are base64 encoded, NOT encrypted by default
- Enable encryption at rest
- Use RBAC to restrict access
- Consider external secret managers

1.23.2 Encryption at Rest

```yaml
# EncryptionConfiguration
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: <base64-encoded-key>
      - identity: {}
```

1.23.3 External Secret Managers

- HashiCorp Vault
- AWS Secrets Manager
- Azure Key Vault

- Google Secret Manager

1.24 Key Concepts to Remember

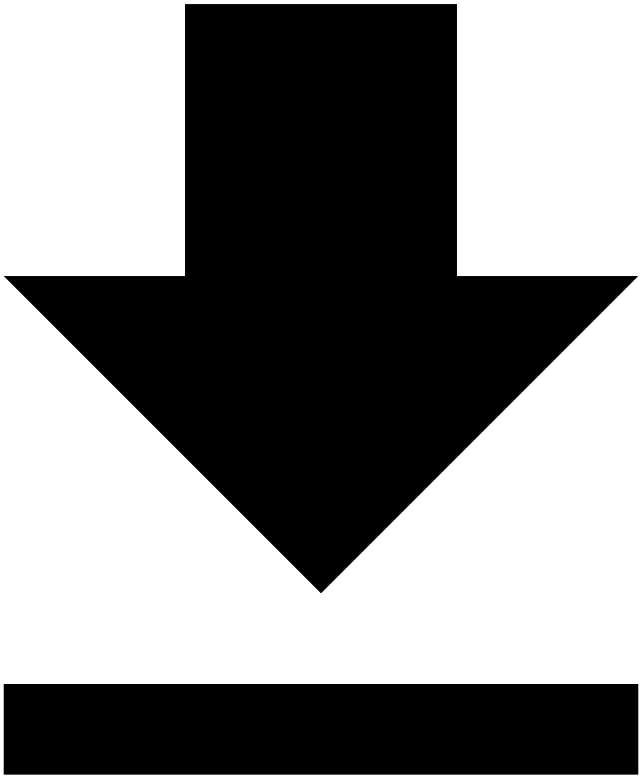
1. **API Server is the gateway** - Secure it thoroughly
2. **etcd contains all secrets** - Encrypt and restrict access
3. **Kubelet security is critical** - Disable anonymous auth
4. **RBAC is the standard** - Use least privilege
5. **Admission controllers** - Last line of defense

1.25 Practice Questions

1. What are the three stages of API request processing?
2. Why should anonymous authentication be disabled on the kubelet?
3. What is the difference between a Role and a ClusterRole?
4. How are Kubernetes Secrets stored by default?
5. What does the NodeRestriction admission controller do?

← Previous: Cloud Native Security Overview | Back to KCSA Overview | Next: Kubernetes Security Fundamentals →

1.26 Kubernetes Security Fundamentals



Download PDF Version

This domain covers core Kubernetes security features and configurations.

1.27 Pod Security

1.27.1 Security Context

Define security settings at the Pod or container level:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 3000
```

```
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: nginx:1.21
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
          - ALL
```

1.27.2 Security Context Fields

Field	Level	Description
`runAsUser`	Pod/Container	UID to run as
`runAsGroup`	Pod/Container	GID to run as
`runAsNonRoot`	Pod/Container	Must run as non-root
`fsGroup`	Pod	Group for volumes
`allowPrivilegeEscalation`	Container	Prevent privilege escalation
`readOnlyRootFilesystem`	Container	Read-only root filesystem
`capabilities`	Container	Linux capabilities
`seccompProfile`	Pod/Container	Seccomp profile
`seLinuxOptions`	Pod/Container	SELinux context

1.27.3 Pod Security Standards

Three policy levels:

Level	Description
Privileged	Unrestricted, allows known privilege escalations
Baseline	Minimally restrictive, prevents known privilege escalations
Restricted	Heavily restricted, follows security best practices

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/enforce-version: latest
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

1.28 Network Policies

1.28.1 Default Behavior

By default, all pods can communicate with all other pods.

1.28.2 Deny All Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

1.28.3 Allow Specific Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080
```

1.28.4 Egress Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-only
  namespace: default
spec:
  podSelector:
    matchLabels:
```

```
      app: restricted
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

1.28.5 Network Policy Selectors

Selector	Description
`podSelector`	Select pods by labels
`namespaceSelector`	Select namespaces by labels
`ipBlock`	Select by IP CIDR range

1.29 RBAC Deep Dive

1.29.1 Role vs ClusterRole

Type	Scope	Use Case
Role	Namespace	Namespace-specific permissions
ClusterRole	Cluster-wide	Cluster resources or aggregation

1.29.2 RoleBinding vs ClusterRoleBinding

Type	Scope	Can Reference
RoleBinding	Namespace	Role or ClusterRole
ClusterRoleBinding	Cluster-wide	ClusterRole only

1.29.3 RBAC Best Practices

```
# Least privilege Role example
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app-namespace
  name: app-deployer
rules:
```

```yaml
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
  resourceNames: ["my-app"]  # Specific resource
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get", "list"]
```

1.29.4 Checking Permissions

```
# Check if you can perform an action
kubectl auth can-i create pods
kubectl auth can-i create pods --
        as=system:serviceaccount:default:my-sa
kubectl auth can-i --list --namespace=default
```

1.30 Service Accounts

1.30.1 Default Service Account

Every namespace has a default service account.

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
automountServiceAccountToken: false
```

1.30.2 Using Service Accounts

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: my-service-account
  automountServiceAccountToken: false  # Disable if not needed
  containers:
  - name: app
    image: my-app:v1
```

1.30.3 Token Projection

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-projected-token
spec:
```

```yaml
  containers:
  - name: app
    image: my-app:v1
    volumeMounts:
    - name: token
      mountPath: /var/run/secrets/tokens
  volumes:
  - name: token
    projected:
      sources:
      - serviceAccountToken:
          path: token
          expirationSeconds: 3600
          audience: my-audience
```

1.31 Resource Quotas and Limits

1.31.1 LimitRange

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: resource-limits
  namespace: default
spec:
  limits:
  - type: Container
    default:
      cpu: "500m"
      memory: "256Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    max:
      cpu: "2"
      memory: "1Gi"
    min:
      cpu: "50m"
      memory: "64Mi"
```

1.31.2 ResourceQuota

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
  namespace: default
spec:
  hard:
    requests.cpu: "4"
```

```
    requests.memory: "8Gi"
    limits.cpu: "8"
    limits.memory: "16Gi"
    pods: "10"
    secrets: "10"
    configmaps: "10"
```

1.32 Image Security

1.32.1 Image Pull Policies

Policy	Behavior
`Always`	Always pull image
`IfNotPresent`	Pull only if not cached
`Never`	Never pull, use cached only

1.32.2 Private Registry Authentication

```
apiVersion: v1
kind: Secret
metadata:
  name: regcred
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: <base64-encoded-docker-config>
---
apiVersion: v1
kind: Pod
metadata:
  name: private-pod
spec:
  containers:
  - name: app
    image: private-registry.io/my-app:v1
  imagePullSecrets:
  - name: regcred
```

1.32.3 Image Digest

Use image digests for immutability:

```
spec:
  containers:
  - name: app
    image: nginx@sha256:abc123...
```

1.33 Key Concepts to Remember

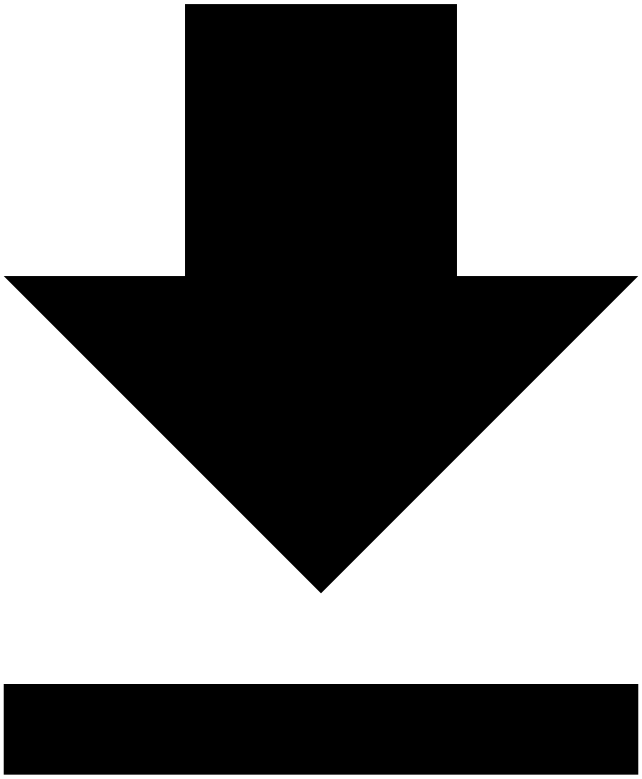
1. **Security contexts** - Define at Pod and container level
2. **Pod Security Standards** - Privileged, Baseline, Restricted
3. **Network Policies** - Default allow, explicit deny
4. **RBAC** - Roles, ClusterRoles, Bindings
5. **Service Accounts** - Disable auto-mount when not needed

1.34 Practice Questions

1. What is the difference between `runAsUser` and `runAsNonRoot`?
2. What happens if no Network Policy exists in a namespace?
3. Can a RoleBinding reference a ClusterRole?
4. How do you prevent a container from escalating privileges?
5. What are the three Pod Security Standards levels?

← Previous: Cluster Component Security | Back to KCSA Overview | Next: Kubernetes Threat Model →

1.35 Kubernetes Threat Model



Download PDF Version

This domain covers threat modeling concepts and common attack vectors in Kubernetes environments.

1.36 Threat Modeling Frameworks

1.36.1 STRIDE Model

Threat	Description	Kubernetes Example
Spoofing	Impersonating something or someone	Stolen service account token
Tampering	Modifying data or code	Modifying ConfigMaps or Secrets
Repudiation	Denying actions	Disabled audit logging
Information Disclosure	Exposing information	Secrets in environment variables
	Disrupting service	Resource exhaustion attacks

Threat	Description	Kubernetes Example
Denial of Service		
Elevation of Privilege	Gaining higher privileges	Container escape

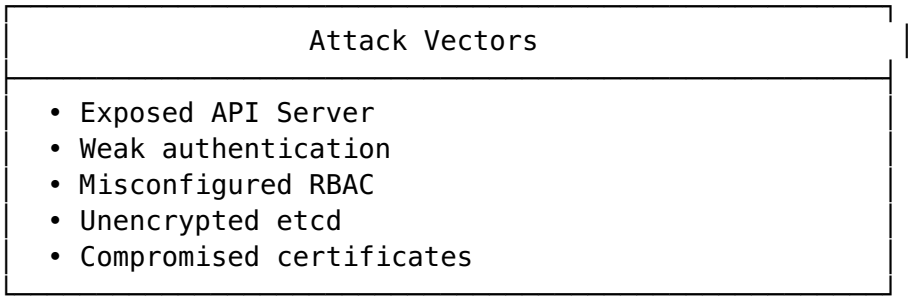
1.36.2 MITRE ATT&CK for Containers

Key tactics relevant to Kubernetes:

Tactic	Description
Initial Access	Gaining entry to the cluster
Execution	Running malicious code
Persistence	Maintaining access
Privilege Escalation	Gaining higher privileges
Defense Evasion	Avoiding detection
Credential Access	Stealing credentials
Discovery	Learning about the environment
Lateral Movement	Moving through the cluster
Impact	Disrupting operations

1.37 Common Attack Vectors

1.37.1 Control Plane Attacks



1.37.2 Node-Level Attacks

- **Kubelet exploitation** - Anonymous access, read-only port
- **Container escape** - Privileged containers, host mounts
- **Node compromise** - SSH access, unpatched vulnerabilities

1.37.3 Pod-Level Attacks

- **Malicious images** - Backdoored or vulnerable images
- **Sidecar injection** - Unauthorized container injection

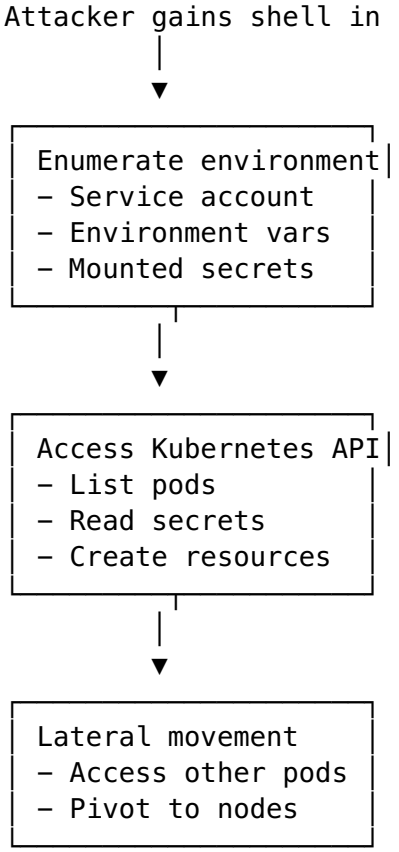
- **Resource abuse** - Cryptomining, DoS

1.37.4 Network-Based Attacks

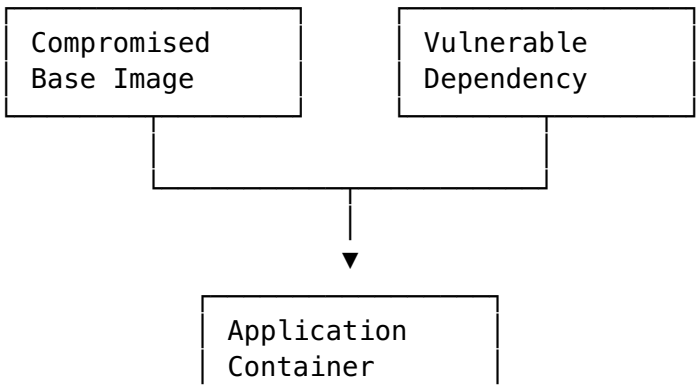
- **Man-in-the-middle** - Unencrypted traffic
- **Pod-to-pod attacks** - No network policies
- **Service impersonation** - DNS spoofing

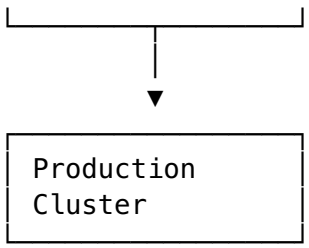
1.38 Kubernetes Attack Scenarios

1.38.1 Scenario 1: Compromised Container

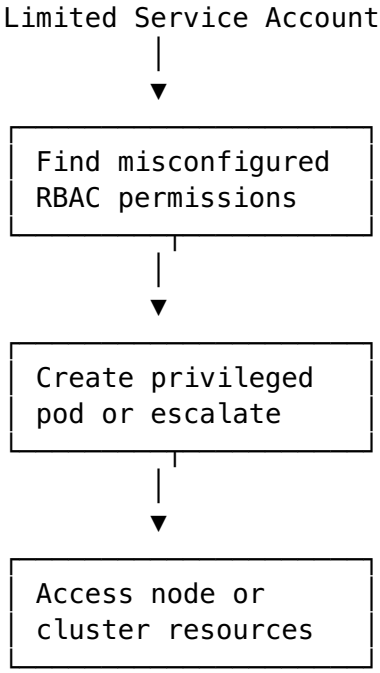


1.38.2 Scenario 2: Supply Chain Attack





1.38.3 Scenario 3: Privilege Escalation



1.39 Threat Mitigation Strategies

1.39.1 API Server Protection

```
# Restrict API access
--anonymous-auth=false
--authorization-mode=Node,RBAC
--enable-admission-plugins=NodeRestriction,PodSecurity
--audit-log-path=/var/log/audit.log
```

1.39.2 Network Segmentation

```
# Default deny all traffic
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
```

```
  - Ingress
  - Egress
```

1.39.3 Pod Security

```yaml
# Restricted pod configuration
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: app:v1
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop: ["ALL"]
```

1.39.4 Runtime Security

Tools for runtime threat detection:

Tool	Purpose
Falco	Runtime security monitoring
Sysdig	Container visibility
Aqua	Full lifecycle security
Prisma Cloud	Cloud native security

1.40 Security Scanning

1.40.1 Image Scanning

```
# Example: Trivy scanner
trivy image nginx:1.21
trivy image --severity HIGH,CRITICAL nginx:1.21
```

1.40.2 Configuration Scanning

```
# Example: kubesec
kubesec scan pod.yaml
```

```
# Example: kube-bench (CIS benchmarks)
kube-bench run --targets master
```

1.40.3 Vulnerability Categories

Category	Examples
CVE	Known vulnerabilities
Misconfigurations	Privileged containers, no limits
Secrets exposure	Hardcoded credentials
Compliance	CIS benchmark failures

1.41 Incident Response

1.41.1 Detection

- Monitor audit logs
- Runtime security alerts
- Anomaly detection
- Network traffic analysis

1.41.2 Containment

- Isolate compromised pods
- Revoke credentials
- Apply network policies
- Cordon affected nodes

1.41.3 Eradication

- Remove malicious workloads
- Patch vulnerabilities
- Rotate secrets
- Update images

1.41.4 Recovery

- Restore from backups
- Redeploy clean workloads
- Verify security posture
- Document lessons learned

1.42 Key Concepts to Remember

1. **STRIDE** - Six categories of threats
2. **MITRE ATT&CK** - Comprehensive attack framework
3. **Defense in depth** - Multiple security layers

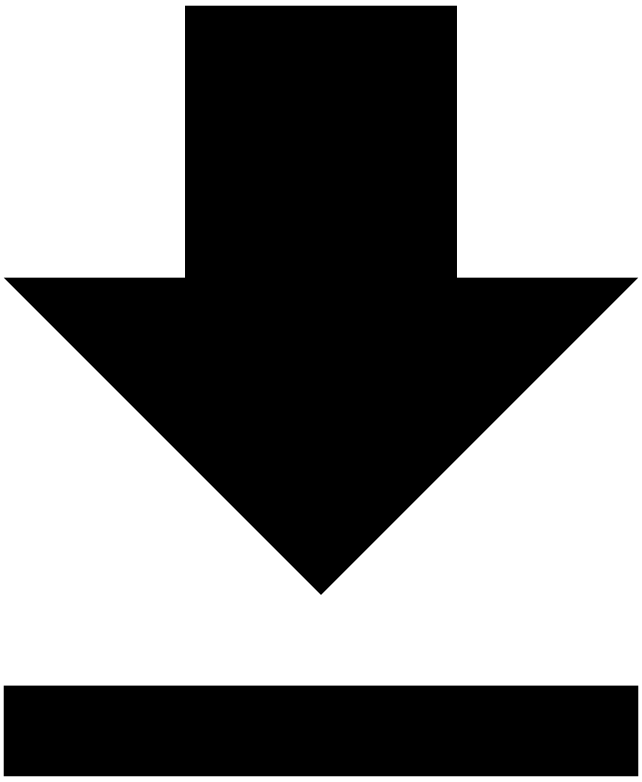
4. **Least privilege** - Minimize permissions
5. **Runtime security** - Detect threats in real-time

1.43 Practice Questions

1. What does STRIDE stand for?
2. Name three common Kubernetes attack vectors.
3. How can you detect a compromised container at runtime?
4. What is the first step in incident response?
5. Why is supply chain security important in Kubernetes?

← Previous: Kubernetes Security Fundamentals | Back to KCSA Overview | Next: Platform Security →

1.44 Platform Security



Download PDF Version

This domain covers securing the underlying platform and infrastructure for Kubernetes.

1.45 Supply Chain Security

1.45.1 Image Security

1.45.1.1 Trusted Registries

```yaml
# OPA/Gatekeeper policy to enforce trusted registries
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: allowed-repos
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    repos:
      - "gcr.io/my-project/"
      - "docker.io/library/"
```

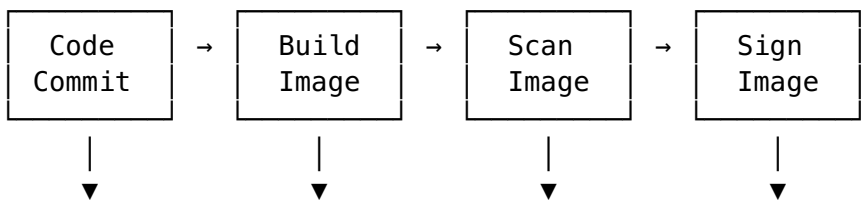
1.45.1.2 Image Signing and Verification

```
# Sign image with cosign
cosign sign --key cosign.key gcr.io/my-project/my-app:v1

# Verify image signature
cosign verify --key cosign.pub gcr.io/my-project/my-app:v1
```

1.45.1.3 Software Bill of Materials (SBOM)

```
# Generate SBOM with syft
syft nginx:1.21 -o spdx-json > nginx-sbom.json

# Scan SBOM for vulnerabilities
grype sbom:nginx-sbom.json
```

1.45.2 Build Pipeline Security



```
SAST/SCA    Dockerfile     Trivy/        Cosign/
scanning    best practices  Grype        Notary
```

1.45.3 Dependency Management

- Pin dependency versions
- Use lock files
- Scan dependencies for vulnerabilities
- Monitor for new CVEs

1.46 GitOps Security

1.46.1 Repository Security

- Enable branch protection
- Require code reviews
- Sign commits with GPG
- Use CODEOWNERS files

1.46.2 Secrets in GitOps

Never store secrets in Git!

Solutions:

Tool	Description
Sealed Secrets	Encrypt secrets for Git storage
SOPS	Encrypt files with various KMS
External Secrets	Sync from external secret stores
Vault	Dynamic secrets management

```yaml
# Sealed Secret example
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  name: my-secret
spec:
  encryptedData:
    password: AgBy3i4OJSWK+PiTySYZZA9rO43...
```

1.46.3 GitOps Deployment Security

```yaml
# Argo CD Application with sync policy
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
```

```
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - Validate=true
      - CreateNamespace=false
```

1.47 Observability and Security

1.47.1 Audit Logging

```
# Audit policy
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: RequestResponse
  resources:
  - group: ""
    resources: ["secrets", "configmaps"]
- level: Metadata
  resources:
  - group: ""
    resources: ["pods", "services"]
- level: None
  users: ["system:kube-proxy"]
  verbs: ["watch"]
```

1.47.2 Audit Log Levels

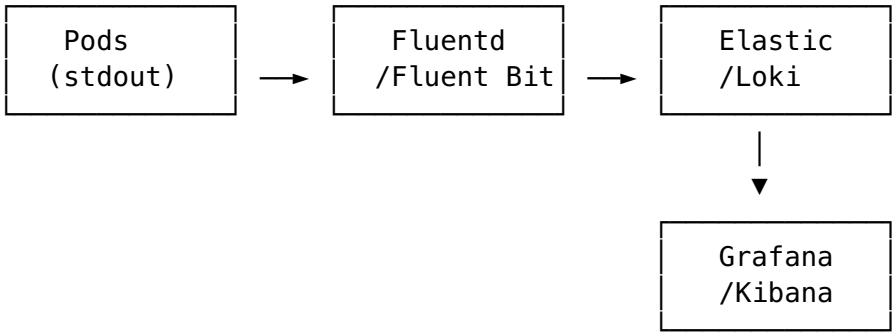
Level	Description
None	Don’t log
Metadata	Log request metadata only
Request	Log metadata and request body
RequestResponse	Log metadata, request, and response

1.47.3 Security Monitoring

```
# Falco rule example
- rule: Terminal shell in container
  desc: Detect shell opened in container
  condition: >
    spawned_process and container and
    shell_procs and proc.tty != 0
  output: >
    Shell opened in container
    (user=%user.name container=%container.name
```

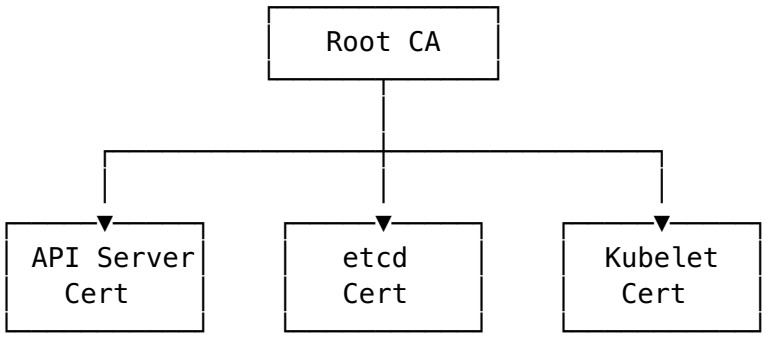
```
    shell=%proc.name parent=%proc.pname)
  priority: WARNING
```

1.47.4 Log Aggregation



1.48 Certificate Management

1.48.1 PKI in Kubernetes



1.48.2 cert-manager

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-app-tls
spec:
  secretName: my-app-tls-secret
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
  dnsNames:
    - my-app.example.com
  duration: 2160h  # 90 days
  renewBefore: 360h  # 15 days
```

1.48.3 Certificate Rotation

- Automate certificate rotation
- Monitor certificate expiration
- Use short-lived certificates
- Implement proper revocation

1.49 Service Mesh Security

1.49.1 mTLS with Istio

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: istio-system
spec:
  mtls:
    mode: STRICT
```

1.49.2 Authorization Policies

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: frontend-policy
  namespace: default
spec:
  selector:
    matchLabels:
      app: frontend
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/default/sa/backend"]
    to:
    - operation:
        methods: ["GET", "POST"]
        paths: ["/api/*"]
```

1.50 Infrastructure as Code Security

1.50.1 Terraform Security

```
# Secure Kubernetes cluster configuration
resource "google_container_cluster" "primary" {
  name     = "my-cluster"
```

```
  location = "us-central1"

  # Enable security features
  enable_shielded_nodes = true

  master_authorized_networks_config {
    cidr_blocks {
      cidr_block   = "10.0.0.0/8"
      display_name = "internal"
    }
  }

  private_cluster_config {
    enable_private_nodes    = true
    enable_private_endpoint = false
    master_ipv4_cidr_block  = "172.16.0.0/28"
  }
}
```

1.50.2 Policy as Code

Tools for enforcing security policies:

Tool	Description
OPA/Gatekeeper	General-purpose policy engine
Kyverno	Kubernetes-native policy engine
Checkov	IaC security scanner
tfsec	Terraform security scanner

1.51 Key Concepts to Remember

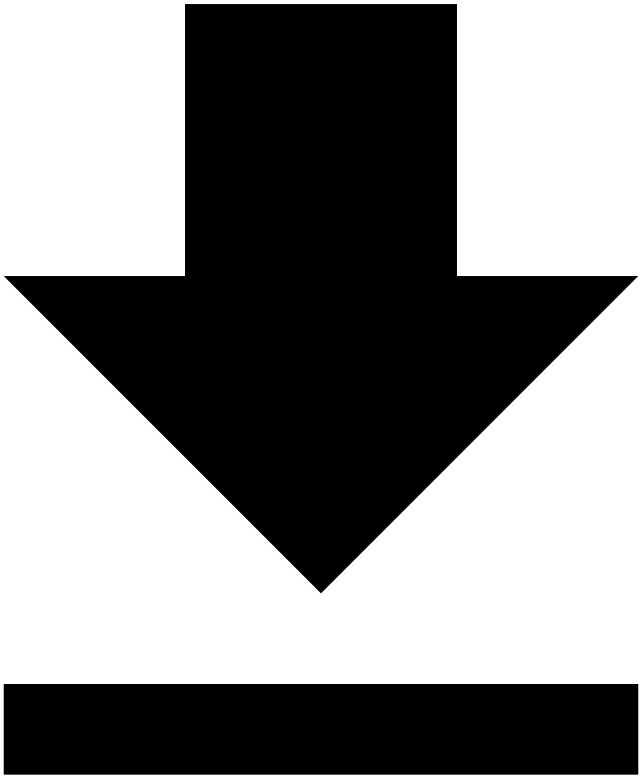
1. **Supply chain security** - Verify and sign images
2. **GitOps** - Never store secrets in Git
3. **Audit logging** - Enable and monitor
4. **Certificate management** - Automate rotation
5. **Service mesh** - mTLS for service-to-service

1.52 Practice Questions

1. What is an SBOM and why is it important?
2. How should secrets be managed in a GitOps workflow?
3. What are the four audit log levels in Kubernetes?
4. What does mTLS provide in a service mesh?
5. Name two policy-as-code tools for Kubernetes.

← Previous: Kubernetes Threat Model | Back to KCSA Overview | Next: Compliance and Security Frameworks →

1.53 Compliance and Security Frameworks



Download PDF Version

This domain covers compliance requirements and security frameworks relevant to Kubernetes environments.

1.54 Compliance Frameworks

1.54.1 CIS Kubernetes Benchmark

The Center for Internet Security (CIS) provides security benchmarks for Kubernetes:

Section	Focus Area
1	Control Plane Components
2	etcd
3	Control Plane Configuration
4	Worker Nodes

Section	Focus Area
5	Policies

1.54.1.1 Running CIS Benchmarks

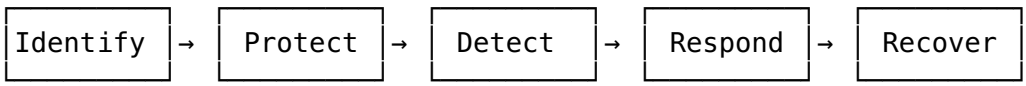
```
# Using kube-bench
kube-bench run --targets master
kube-bench run --targets node
kube-bench run --targets etcd

# Output specific checks
kube-bench run --targets master --check 1.1.1,1.1.2
```

1.54.1.2 Example CIS Controls

Control	Description	Remediation
1.1.1	API server pod spec permissions	chmod 644 /etc/kubernetes/manifests/kube-apiserver.yaml
1.2.1	Anonymous auth disabled	–anonymous-auth=false
4.2.1	Kubelet anonymous auth	–anonymous-auth=false
5.1.1	RBAC enabled	–authorization-mode=RBAC

1.54.2 NIST Cybersecurity Framework

Five core functions:



Function	Kubernetes Application
Identify	Asset inventory, risk assessment
Protect	RBAC, Network Policies, Pod Security
Detect	Audit logging, runtime security
Respond	Incident response procedures
Recover	Backup/restore, disaster recovery

1.54.3 SOC 2

Service Organization Control 2 - Trust Service Criteria:

Criteria	Kubernetes Relevance
Security	Access controls, encryption
Availability	High availability, disaster recovery

Criteria	Kubernetes Relevance
Processing Integrity	Data validation, error handling
Confidentiality	Secrets management, encryption
Privacy	Data handling, retention

1.54.4 PCI DSS

Payment Card Industry Data Security Standard:

Requirement	Kubernetes Implementation
Req 1	Network segmentation (Network Policies)
Req 2	Secure configurations (CIS benchmarks)
Req 3	Protect stored data (Encryption at rest)
Req 4	Encrypt transmission (TLS/mTLS)
Req 7	Restrict access (RBAC)
Req 10	Track and monitor (Audit logging)

1.54.5 HIPAA

Health Insurance Portability and Accountability Act:

- **Access controls** - RBAC, authentication
- **Audit controls** - Logging, monitoring
- **Integrity controls** - Image signing, admission control
- **Transmission security** - TLS encryption

1.55 Security Standards

1.55.1 Pod Security Standards

Kubernetes native security standards:

```yaml
# Namespace labels for Pod Security Standards
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    # Enforce restricted standard
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/enforce-version: v1.28
    # Audit baseline violations
    pod-security.kubernetes.io/audit: baseline
    # Warn on privileged violations
    pod-security.kubernetes.io/warn: privileged
```

1.55.2 NSA/CISA Kubernetes Hardening Guide

Key recommendations:

1. **Scan containers and pods** for vulnerabilities
2. **Run containers as non-root** users
3. **Use immutable container filesystems**
4. **Scan container images** for misconfigurations
5. **Use Pod Security Standards**
6. **Harden all authentication and authorization**
7. **Use network policies** to isolate resources
8. **Encrypt data** in transit and at rest
9. **Enable audit logging**
10. **Restrict access to control plane**

1.56 Audit and Compliance Tools

1.56.1 Policy Enforcement

1.56.1.1 OPA Gatekeeper

```
# Constraint Template
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema:
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels
        violation[{"msg": msg}] {
          provided := {label |
        input.review.object.metadata.labels[label]}
          required := {label | label :=
        input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
```

```
        msg := sprintf("Missing required labels: %v", [missing])
      }
```

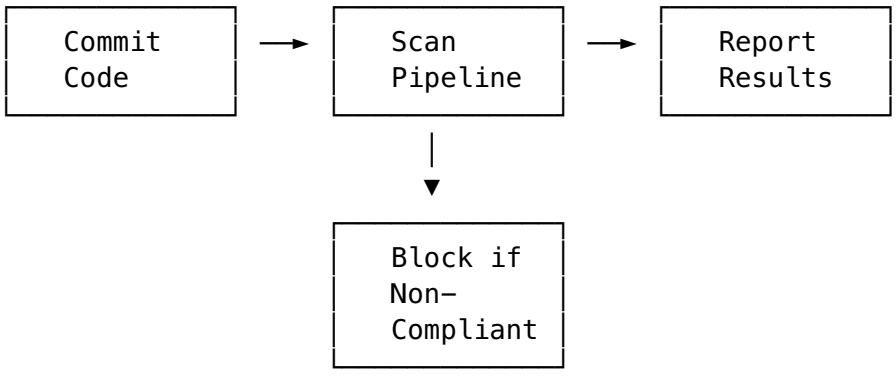
1.56.1.2 Kyverno

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-labels
spec:
  validationFailureAction: Enforce
  rules:
  - name: check-for-labels
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "The label 'app' is required."
      pattern:
        metadata:
          labels:
            app: "?*"
```

1.56.2 Compliance Scanning

Tool	Purpose
kube-bench	CIS benchmark scanning
Trivy	Vulnerability and misconfiguration scanning
Polaris	Best practices validation
Kubescape	NSA/CISA compliance scanning
Checkov	Infrastructure as code scanning

1.56.3 Continuous Compliance



1.57 Documentation and Evidence

1.57.1 Security Documentation

Required documentation for compliance:

- Security policies and procedures
- Network diagrams
- Access control matrices
- Incident response plans
- Change management procedures
- Audit logs and reports

1.57.2 Evidence Collection

```
# Collect cluster configuration
kubectl get nodes -o yaml > nodes.yaml
kubectl get namespaces -o yaml > namespaces.yaml
kubectl get networkpolicies -A -o yaml > networkpolicies.yaml
kubectl get roles,rolebindings -A -o yaml > rbac.yaml

# Export audit logs
kubectl logs -n kube-system kube-apiserver-* > audit.log
```

1.57.3 Compliance Reporting

Report Type	Frequency	Content
Vulnerability	Daily/Weekly	CVE findings
Configuration	Weekly	CIS benchmark results
Access Review	Monthly	RBAC audit
Incident	As needed	Security incidents
Compliance	Quarterly	Framework compliance status

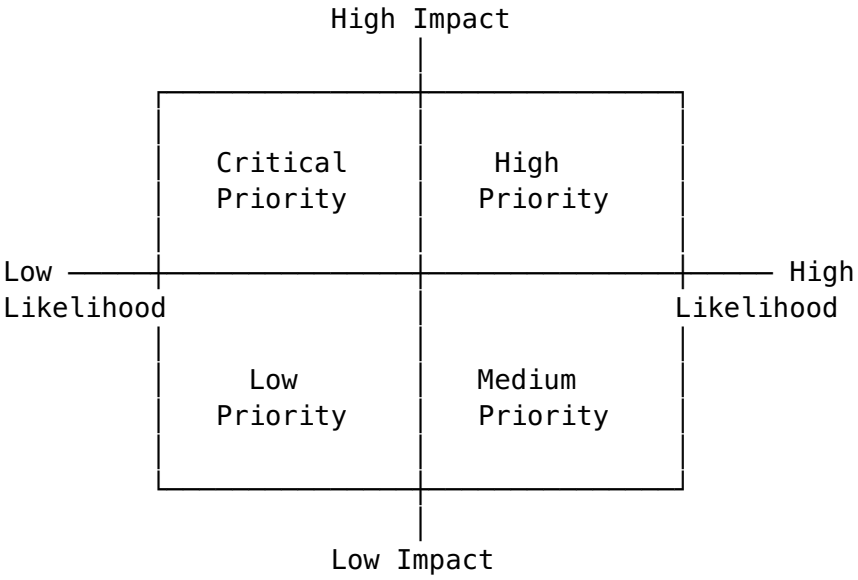
1.58 Risk Management

1.58.1 Risk Assessment

Risk Category	Examples	Mitigation
Technical	Vulnerabilities, misconfigurations	Scanning, hardening
Operational	Human error, process failures	Training, automation
Compliance	Regulatory violations	Policies, audits

Risk Category	Examples	Mitigation
Business	Service disruption	HA, DR planning

1.58.2 Risk Prioritization



1.59 Key Concepts to Remember

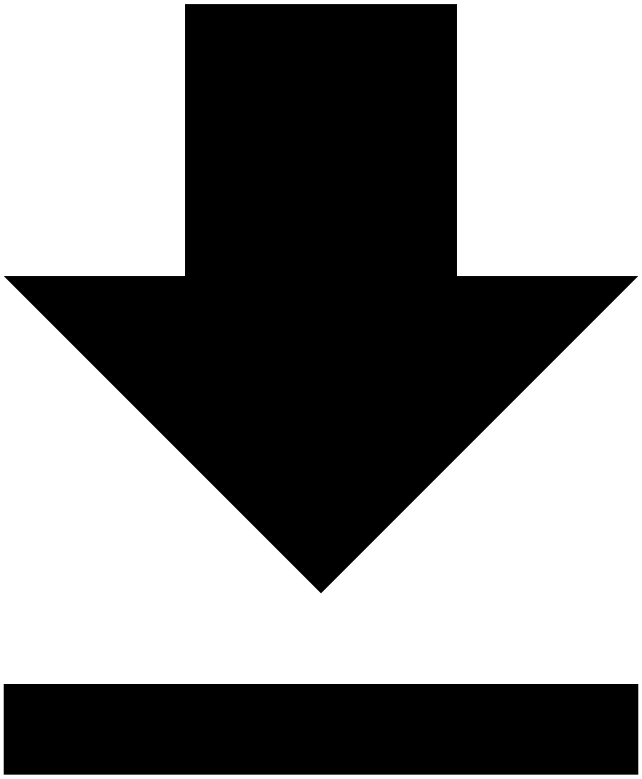
1. **CIS Benchmarks** - Industry standard for Kubernetes security
2. **NIST Framework** - Identify, Protect, Detect, Respond, Recover
3. **Pod Security Standards** - Kubernetes native security levels
4. **Policy engines** - OPA Gatekeeper, Kyverno
5. **Continuous compliance** - Automate scanning and reporting

1.60 Practice Questions

1. What are the five core functions of the NIST Cybersecurity Framework?
2. What tool is commonly used to check CIS Kubernetes benchmarks?
3. What are the three Pod Security Standards levels?
4. Name two policy engines for Kubernetes.
5. What is the purpose of the NSA/CISA Kubernetes Hardening Guide?

← Previous: Platform Security | Back to KCSA Overview | Next: Sample Practice Questions →

1.61 Sample Practice Questions



Download PDF Version

> **Disclaimer**: These are sample practice questions created for study purposes only. They are NOT actual exam questions and are designed to help you test your understanding of KCSA concepts. Real exam questions may differ in format and content.

1.62 Instructions

- Each question has one correct answer unless otherwise specified
- Try to answer without looking at the solutions first
- Review the explanations to understand the concepts better

1.63 Section 1: Cloud Native Security Overview

1.63.1 Question 1.1

What are the 4Cs of Cloud Native Security in order from the outermost to innermost layer?

1. Code, Container, Cluster, Cloud
2. Cloud, Cluster, Container, Code
3. Cluster, Cloud, Code, Container
4. Container, Code, Cloud, Cluster

Show Answer

Answer: B) Cloud, Cluster, Container, Code

The 4Cs represent security layers from infrastructure (Cloud) to application (Code). Each layer builds upon the security of the layer below it.

1.63.2 Question 1.2

In the shared responsibility model for managed Kubernetes services, who is typically responsible for securing the worker node operating system?

1. Cloud provider only
2. Customer only
3. Both cloud provider and customer
4. Neither - it’s automatically secured

Show Answer

Answer: B) Customer only

In managed Kubernetes services (like EKS, GKE, AKS), the cloud provider manages the control plane, but customers are responsible for worker node security, including OS patching and configuration.

1.63.3 Question 1.3

Which security principle states that users and processes should only have the minimum permissions necessary to perform their tasks?

1. Defense in depth
2. Zero trust
3. Least privilege
4. Separation of duties

Show Answer

Answer: C) Least privilege

The principle of least privilege ensures that access rights are limited to only what is required, reducing the potential impact of a security breach.

1.64 Section 2: Kubernetes Cluster Component Security

1.64.1 Question 2.1

Which Kubernetes component stores all cluster state and should be encrypted at rest?

1. kube-apiserver
2. kube-scheduler
3. etcd
4. kube-controller-manager

Show Answer

Answer: C) etcd

etcd is the key-value store that holds all cluster data, including Secrets. Encrypting etcd at rest is critical for protecting sensitive information.

1.64.2 Question 2.2

What is the recommended value for the `--anonymous-auth` flag on the kubelet?

1. true
2. false
3. webhook
4. certificate

Show Answer

Answer: B) false

Anonymous authentication should be disabled on the kubelet to prevent unauthorized access to the kubelet API.

1.64.3 Question 2.3

Which authentication method uses tokens issued by an external identity provider?

1. X.509 client certificates
2. Service account tokens
3. OpenID Connect (OIDC)
4. Static token file

Show Answer

Answer: C) OpenID Connect (OIDC)

OIDC allows Kubernetes to integrate with external identity providers like Google, Azure AD, or Okta for user authentication.

1.64.4 Question 2.4

What is the correct order of request processing in the Kubernetes API server?

1. Authorization → Authentication → Admission Control
2. Authentication → Admission Control → Authorization
3. Authentication → Authorization → Admission Control
4. Admission Control → Authentication → Authorization

Show Answer

Answer: C) Authentication → Authorization → Admission Control

Requests first authenticate (who are you?), then authorize (what can you do?), and finally pass through admission controllers (should this be allowed/modified?).

1.65 Section 3: Kubernetes Security Fundamentals

1.65.1 Question 3.1

Which Pod Security Standard level provides the most restrictive security policies?

1. Privileged
2. Baseline
3. Restricted
4. Default

Show Answer

Answer: C) Restricted

The Restricted level enforces the most stringent security requirements, following current best practices for pod hardening.

1.65.2 Question 3.2

What happens to pod-to-pod traffic in a namespace with no Network Policies defined?

1. All traffic is denied by default
2. All traffic is allowed by default
3. Only traffic within the namespace is allowed
4. Only traffic from the same node is allowed

Show Answer

Answer: B) All traffic is allowed by default

Without Network Policies, Kubernetes allows all pod-to-pod communication. Network Policies must be explicitly created to restrict traffic.

1.65.3 Question 3.3

Which security context field prevents a container process from gaining more privileges than its parent?

1. runAsNonRoot
2. readOnlyRootFilesystem
3. allowPrivilegeEscalation
4. privileged

Show Answer

Answer: C) allowPrivilegeEscalation

Setting `allowPrivilegeEscalation: false` prevents a process from gaining additional privileges through setuid binaries or other mechanisms.

1.65.4 Question 3.4

Can a RoleBinding in namespace “app” grant permissions defined in a ClusterRole?

1. No, RoleBindings can only reference Roles
2. Yes, but only for cluster-scoped resources
3. Yes, the ClusterRole permissions will be scoped to namespace “app”
4. Yes, and the permissions will apply cluster-wide

Show Answer

Answer: C) Yes, the ClusterRole permissions will be scoped to namespace “app”

A RoleBinding can reference a ClusterRole, but the permissions are limited to the namespace where the RoleBinding exists.

1.66 Section 4: Kubernetes Threat Model

1.66.1 Question 4.1

In the STRIDE threat model, what does the “E” stand for?

1. Encryption
2. Exploitation
3. Elevation of Privilege
4. Enumeration

Show Answer

Answer: C) Elevation of Privilege

STRIDE stands for: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

1.66.2 Question 4.2

Which of the following is a common container escape technique?

1. Using a ClusterIP service
2. Mounting the host filesystem
3. Creating a ConfigMap
4. Using a Deployment

Show Answer

Answer: B) Mounting the host filesystem

Mounting sensitive host paths (like `/`, `/etc`, or the Docker socket) can allow attackers to escape the container and access the host system.

1.66.3 Question 4.3

Which tool is commonly used for runtime security monitoring in Kubernetes?

1. Trivy
2. kube-bench
3. Falco
4. kubesec

Show Answer

Answer: C) Falco

Falco is a CNCF project that provides runtime security monitoring by detecting anomalous activity in containers and Kubernetes.

1.67 Section 5: Platform Security

1.67.1 Question 5.1

What is the purpose of signing container images with tools like cosign?

1. To compress the image
2. To verify the image hasn't been tampered with
3. To encrypt the image contents
4. To speed up image pulls

Show Answer

Answer: B) To verify the image hasn't been tampered with

Image signing creates a cryptographic signature that can be verified to ensure the image hasn't been modified since it was signed.

1.67.2 Question 5.2

In a GitOps workflow, how should Kubernetes Secrets be handled?

1. Store them directly in Git
2. Use base64 encoding in Git
3. Use Sealed Secrets or external secret managers
4. Include them in ConfigMaps instead

Show Answer

Answer: C) Use Sealed Secrets or external secret managers

Secrets should never be stored in plain text in Git. Tools like Sealed Secrets, SOPS, or external secret managers (Vault, AWS Secrets Manager) should be used.

1.67.3 Question 5.3

What does mTLS provide in a service mesh?

1. Load balancing only
2. Mutual authentication between services
3. Service discovery
4. Rate limiting

Show Answer

Answer: B) Mutual authentication between services

mTLS (mutual TLS) ensures both the client and server authenticate each other, providing encrypted and authenticated service-to-service communication.

1.68 Section 6: Compliance and Security Frameworks

1.68.1 Question 6.1

Which tool is commonly used to check Kubernetes clusters against CIS benchmarks?

1. Trivy
2. kube-bench
3. Falco
4. OPA

Show Answer

Answer: B) kube-bench

kube-bench is specifically designed to check Kubernetes clusters against CIS Kubernetes Benchmark recommendations.

1.68.2 Question 6.2

What are the five core functions of the NIST Cybersecurity Framework?

1. Plan, Do, Check, Act, Review
2. Identify, Protect, Detect, Respond, Recover
3. Assess, Implement, Monitor, Report, Improve
4. Prevent, Detect, Contain, Eradicate, Recover

Show Answer

Answer: B) Identify, Protect, Detect, Respond, Recover

The NIST CSF provides a framework for managing cybersecurity risk through these five core functions.

1.68.3 Question 6.3

Which Kubernetes-native policy engine uses YAML-based policies?

1. OPA Gatekeeper
2. Kyverno
3. Falco
4. Trivy

Show Answer

Answer: B) Kyverno

Kyverno uses Kubernetes-native YAML policies, while OPA Gatekeeper uses the Rego policy language.

1.69 Scenario-Based Questions

1.69.1 Scenario 1

Your security team has identified that some pods in your cluster are running as root. You need to enforce that all pods in the “production” namespace run as non-root users.

Question: Which is the most appropriate solution?

1. Add a LimitRange to the namespace
2. Apply Pod Security Standards with “restricted” level
3. Create a ResourceQuota

4. Configure a HorizontalPodAutoscaler

Show Answer

Answer: B) Apply Pod Security Standards with "restricted" level

Pod Security Standards at the "restricted" level enforce that containers must run as non-root, among other security requirements.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: restricted
```

1.69.2 Scenario 2

You need to ensure that pods in the "backend" namespace can only communicate with pods in the "database" namespace on port 5432.

Question: What type of resource should you create?

1. Service
2. Ingress
3. NetworkPolicy
4. PodSecurityPolicy

Show Answer

Answer: C) NetworkPolicy

NetworkPolicy resources control pod-to-pod communication. You would create an egress policy in the "backend" namespace allowing traffic to the "database" namespace on port 5432.

1.69.3 Scenario 3

An audit reveals that your cluster's API server allows anonymous requests. What is the recommended remediation?

1. Enable RBAC authorization mode
2. Set `--anonymous-auth=false` on the API server
3. Create a ClusterRole for anonymous users
4. Enable audit logging

Show Answer

Answer: B) Set `--anonymous-auth=false` on the API server

Disabling anonymous authentication prevents unauthenticated requests from accessing the API server.

1.70 Study Tips

1. **Understand the concepts** - Don’t just memorize; understand why security controls exist
2. **Practice with real clusters** - Set up a test cluster and implement security controls
3. **Review official documentation** - Kubernetes security docs are comprehensive
4. **Know the tools** - Familiarize yourself with kube-bench, Trivy, Falco, OPA
5. **Understand compliance frameworks** - Know the basics of CIS, NIST, and Pod Security Standards

---

← Back to KCSA Overview
