



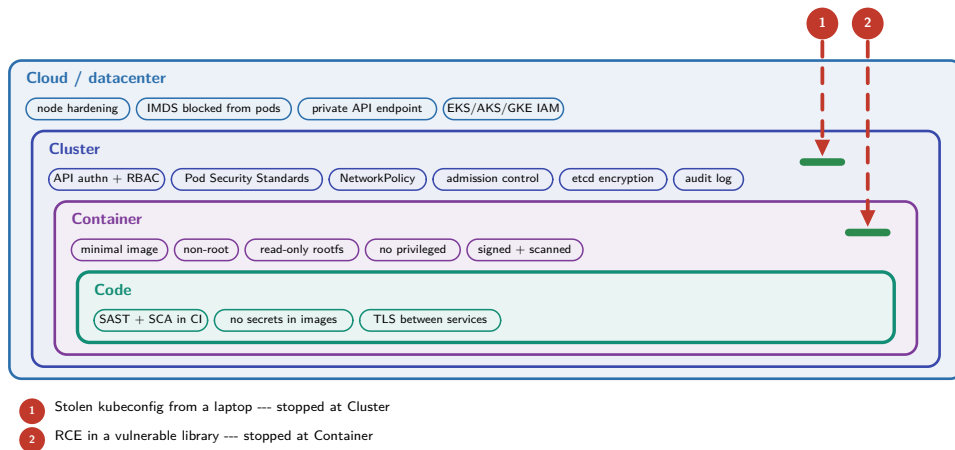
Kubernetes Security

4C model, threat model, Pod Security Standards, RBAC, NetworkPolicy, admission control, secrets, images, Falco, audit, CIS | CKS, CCSP, SY0-701

#Kubernetes #RBAC #PodSecurity #Containers



Model and threat landscape



The 4C model (Cloud, Cluster, Container, Code) as defence in depth: each layer inherits the weaknesses of the one outside it; an attack is stopped by the first layer whose control holds. SAST = static code analysis, SCA = dependency vulnerability scanning, RBAC = role-based access control, etcd = the cluster key-value store

Component

	Initial access	Priv. escalation	Lateral move	Data theft	Persistence
API server	5	4	3	3	4
etcd	2	3	1	5	2
kubelet (10250)	4	5	3	2	2
Pod / workload	4	5	4	3	4
Image registry	3	2	1	2	5
Network / CNI	2	1	5	3	1
Secrets	1	4	4	5	3

Attack class →

Where each component is exposed (5 = primary target for that class)

Port	Component	Exposure risk
6443/tcp	kube-apiserver	public endpoint + anonymous auth = full cluster; restrict source IPs
2379-2380/tcp	etcd client / peer	unauthenticated etcd = every secret in clear; mutual TLS (client certificates) only
10250/tcp	kubelet API	--anonymous-auth=false, --authorization-mode=Webhook; exec into any pod otherwise
10255/tcp	kubelet read-only	legacy pod and env listing; --read-only-port=0
10257, 10259/tcp	controller-manager, scheduler	bind to 127.0.0.1
30000-32767/tcp	NodePort services	every node listens; prefer LoadBalancer/Ingress with allowlists

Pod — Smallest deployable unit: one or more containers sharing a network namespace and volumes, scheduled on one node.

Service account (Kubernetes) — Namespaced identity for pods; its token authenticates the pod to the API server.

Admission controller — API server plug-in that validates or mutates an object after authentication and authorisation, before it is stored.

etcd — Distributed key-value store holding all cluster state, including Secrets; reading it equals owning the cluster.

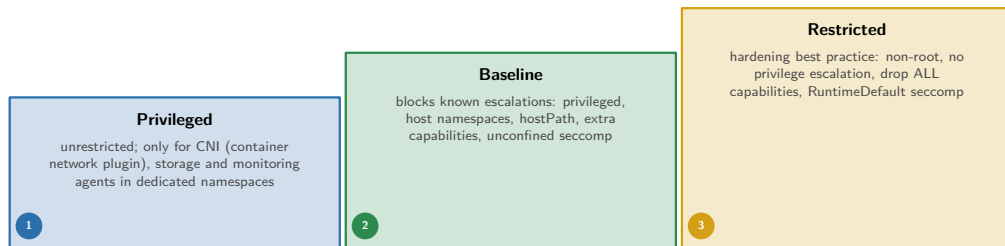
Container escape — Breaking out of container isolation to the host, typically via privileged mode, host mounts, kernel or runtime bugs.

EXAM TIP

MITRE **ATT&CK Containers** and the Microsoft Kubernetes threat matrix: expect questions mapping *exec into pod*, *privileged container*, *hostPath mount*, *cluster-admin binding* and *stolen SA token* to escalation or persistence.



Pod Security Standards and RBAC



Pod Security Standards (PSS), enforced per namespace by the built-in Pod Security Admission controller (PSA); the older PodSecurityPolicy (PSP) was removed in 1.25

- **Service account tokens** bound, expiring, audience-scoped tokens (TokenRequest API) are the default since 1.22; set `automountServiceAccountToken: false` on pods that never call the API; legacy long-lived secret tokens are flagged by `kubectl get secrets --field-selector type=kubernetes.io/service-account-token`
- **Authentication** OIDC with the corporate identity provider (or the managed service's IAM: EKS access entries, GKE Groups, AKS Entra integration); client certificates cannot be revoked, so avoid them for humans; `--anonymous-auth=false`

Field	Baseline	Restricted
<code>securityContext.privileged</code>	false	false
<code>hostNetwork, hostPID, hostIPC</code>	false	false
<code>volumes[].hostPath, hostPorts</code>	forbidden	forbidden
<code>capabilities.add</code>	only the default runtime set (CHOWN, NET_BIND_SERVICE, SETUID...)	drop ALL, add only NET_BIND_SERVICE
<code>seccompProfile.type</code>	must not be Unconfined	must be RuntimeDefault or Localhost
<code>runAsNonRoot / runAsUser</code>	-	true / non-zero
<code>allowPrivilegeEscalation</code>	-	false
Volume types	any except hostPath	configMap, secret, emptyDir, projected, downwardAPI, csi, ephemeral, PVC

Enable PSA on a namespace (enforce, plus warn/audit at the stricter level)

```
kubectl label ns prod \
  pod-security.kubernetes.io/enforce=baseline \
  pod-security.kubernetes.io/enforce-version=v1.33 \
  pod-security.kubernetes.io/warn=restricted \
  pod-security.kubernetes.io/audit=restricted
```

■ Role / ClusterRole rules: apiGroups, resources, verbs

- **Role**
 - namespaced permissions
- **ClusterRole**
 - cluster-wide or reused in namespaces via RoleBinding; non-resource URLs; aggregation labels

■ RoleBinding / ClusterRoleBinding roleRef + subjects

- **Subjects**
 - User (from cert CN or OIDC claim, no object)
 - Group (`system:masters` bypasses RBAC, `system:authenticated`, `system:serviceaccounts:NS`)
 - ServiceAccount `NS:name`

■ Risky permissions each is a path to cluster-admin

- **Verbs**
 - `escalate, bind, impersonate, *`
- **Resources**
 - `secrets get/list, pods/exec, pods create` (mount any SA token, hostPath), `nodes/proxy, serviceaccounts/token, clusterrolebindings create, webhook configurations`

🔗 Network policy and admission control

Default deny + DNS egress (L3/L4, additive allows; needs an enforcing CNI)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: {name: default-deny, namespace: prod}
spec:
  podSelector: {}
  policyTypes: [Ingress, Egress]
  egress:
    - to:
      - namespaceSelector:
          matchLabels: {kubernetes.io/metadata.name: kube-system}
        podSelector: {matchLabels: {k8s-app: kube-dns}}
      ports: [{protocol: UDP, port: 53}, {protocol: TCP, port: 53}]
```

ValidatingAdmissionPolicy	Kyverno	OPA Gatekeeper
Built into the API server (GA 1.30), CEL (Common Expression Language) expressions, no web-hook to operate	Operator + webhook, policies in plain YAML	Operator + webhook, policies in Rego (ConstraintTemplate + Constraint)
Validate only; MutatingAdmissionPolicy beta in 1.34; params from any resource	validate, mutate, generate, cleanup, verifyImages (cosign/notation)	validate, mutate, external data, audit of existing resources
No reports, no dry-run on existing objects (audit via Validating AdmissionPolicyBinding audit action)	PolicyReport CRDs, kyverno apply in CI, background scans	audit results in Constraint status, gator test in CI
Use for simple invariants: labels, registries, no latest	Default choice for teams without Rego skills; image signature checks	Rego (OPA policy language) reuse with Conftest checks on Terraform and other IaC

🔑 Secrets, images and runtime

Secret at rest — base64 only. Enable EncryptionConfiguration on the API server (kms v2 provider with the cloud KMS, else aescbc/secretbox) then rewrite: `kubectl get secrets -A -o json | kubectl replace -f -`. Managed: EKS envelope encryption, GKE application-layer secrets encryption, AKS KMS.

External Secrets Operator — syncs Vault / AWS Secrets Manager / Azure Key Vault / GCP Secret Manager into K8s Secrets via ExternalSecret + SecretStore; rotation follows the backend.

Secrets Store CSI driver — mounts secrets from the backend as files at pod start, optional sync to a Secret; Vault Agent injector does the same with a sidecar.

GitOps encryption — Sealed Secrets (cluster public key, controller decrypts, cluster-bound) or SOPS (age/KMS-encrypted values, decrypted by Flux or Argo CD plugin). Never plain Secrets in git.

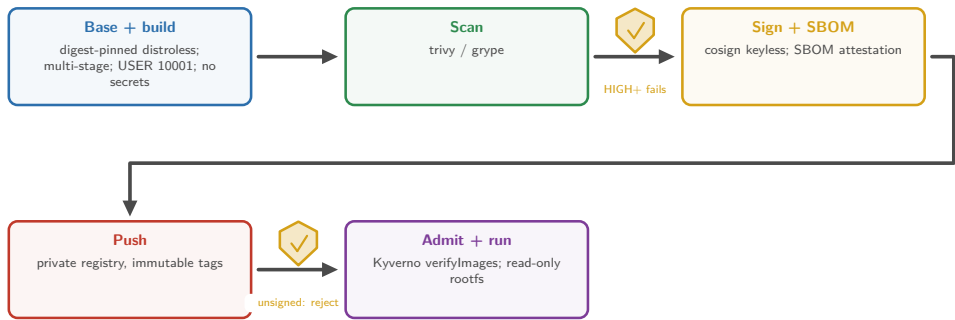


Image supply chain: prove provenance (SLSA levels), list components (SBOM, software bill of materials), enforce at admission. Base images: distroless, Alpine or Chainguard. Admission: Kyverno verifyImages, Sigstore policy-controller or GKE Binary Authorization. Run: drop capabilities, imagePullPolicy: Always, pull secrets per namespace

```
Falco rule: interactive shell in a container
- rule: Shell spawned in container
  desc: Interactive shell started inside a container
  condition: >
    spawned_process and container
    and proc.name in (bash, sh, zsh, dash) and proc.tty != 0
  output: >
    Shell in container (user=%user.name container=%container.name
    image=%container.image.repository cmd=%proc.cmdline)
  priority: WARNING
  tags: [container, shell, mitre_execution]
```

Falco eBPF syscall sensor + k8saudit plugin; default rules *Terminal shell in container, Launch Privileged Container, Contact K8S API Server From Container*; output via Falcosidekick to the SIEM. Alternatives: Tetragon, KubeArmor, GuardDuty Runtime Monitoring, Defender for Containers, GKE Container Threat Detection

Audit logs --audit-policy-file (levels None / Metadata / Request / RequestResponse), --audit-log-path or webhook; secrets at Metadata only. Managed: EKS audit log type, AKS kube-audit-admin, GKE Cloud Audit Logs. Hunts: pods/exec create, system:anonymous, clusterrolebindings writes, 403 bursts from one SA

⚠ Checks, CIS, misconfigurations, self-test

- ⚠ **Frequent findings**
- default SA bound to cluster-admin (or system:serviceaccounts group) by a Helm chart; wildcard * verbs/resources in roles
 - Privileged pod with hostPath: / and hostPID: node escape in one nsenter -t 1 -m -u -i -n sh
 - Node instance role reachable from pods through IMDSv1 (EKS): cluster to cloud account in one curl; public API endpoint or Dashboard with skip login (Tesla 2018 crypto-mining)
 - No NetworkPolicy: flat network, pods reach etcd, kubelet, metadata and databases; secrets in ConfigMaps or pod env; latest tags, no admission control

CIS Kubernetes Benchmark pointers (sections 1-5)

- ✓ 1 Control plane (API server flags): no anonymous requests (--anonymous-auth=false), authorise with --authorization-mode=Node,RBAC, audit logging on (--audit-log-*), secrets encrypted (--encryption-provider-config), debug endpoint off (--profiling=false), admission NodeRestriction (a kubelet may only modify its own node and pods)
- ✓ 2 etcd: require client and peer certificates (--client-cert-auth=true, --peer-client-cert-auth=true), no self-signed auto TLS (--auto-tls=false), a CA separate from the API server's so API client certs cannot open etcd
- ✓ 4 Worker nodes: kubelet refuses anonymous calls and asks the API server to authorise each one (--anonymous-auth=false, --authorization-mode=Webhook), legacy read-only port closed (--read-only-port=0), certificates rotated, kernel sysctls not overridden (--protect-kernel-defaults), kubeconfig and PKI files mode 600 owned by root
- ✓ 5 Policies: no cluster-admin beyond break-glass, no wildcard rules, no secrets in env vars, default SA unused, PSA per namespace, NetworkPolicy per namespace, seccomp RuntimeDefault

Command	Check
kubectl auth can-i --list --as=system:serviceaccount:prod:app	effective permissions of a service account
kubectl auth can-i create clusterrolebindings --as=dev@corp.com	can this user escalate?
kubectl get clusterrolebindings -o json jq -r '.items[] select(.roleRef.name=="cluster-admin") .subjects[]?'	who is cluster-admin
kubectl get pods -A -o json jq -r '.items[] select(.spec.containers[].securityContext.privileged==true) .metadata.name'	privileged pods (jsonpath alternative: -o jsonpath='{..securityContext.privileged}')
kubectl get pods -A -o json jq -r '.items[] select(.spec.hostNetwork==true or .spec.hostPID==true or (.spec.volumes[]?.hostPath)) .metadata.name'	host namespace or hostPath pods
kubectl get ns -o custom-columns='NS:.metadata.name,ENFORCE:.metadata.labels.pod-security\.kubernetes\.io/enforce'	namespaces without PSA enforcement
kubectl get networkpolicy -A; kubectl get sa -A -o json jq '.items[] select(.automountServiceAccountToken!=false) .metadata.name' wc -l	namespaces without policies; SAs auto-mounting tokens
kube-bench run --targets master,node,policies	CIS Kubernetes Benchmark checks (use --benchmark eks-1.x / gke-1.x / aks-1.x)
trivy k8s --report summary cluster	CVEs, misconfigurations, RBAC and secrets across the cluster
kubescape scan framework nsa,mitre; kubeaudit all	NSA/CISA hardening guide and MITRE checks; workload audit

Q1

Which PSS level requires runAsNonRoot: true and dropping ALL capabilities?

A

Restricted.

Q2

Which RBAC verbs allow privilege escalation by design?

A

escalate, bind, impersonate (plus create on pods/exec, secrets read).

Q3

Which admission mechanism needs no webhook or operator?

A

ValidatingAdmissionPolicy (CEL, built into the API server since 1.30).

Q4

How does a pod on EKS reach the AWS account without IRSA?

A

Through the node's instance role on IMDS (169.254.169.254); block with hop limit 1 or a NetworkPolicy.

✳️ Hardening configs and CVEs that shaped the baseline

```
EncryptionConfiguration (KMS v2 first, identity last for reads)
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
- resources: [secrets]
  providers:
  - kms:
      apiVersion: v2
      name: cloud-kms
      endpoint: unix:///var/run/kms/socket.sock
  - identity: {}
# kube-apiserver --encryption-provider-config=<file>
```

```
Restricted pod securityContext (passes PSS restricted)
securityContext:
  runAsNonRoot: true
  runAsUser: 10001
  seccompProfile: {type: RuntimeDefault}
  appArmorProfile: {type: RuntimeDefault} # field GA 1.30
containers:
- name: app
  securityContext:
    allowPrivilegeEscalation: false
    readOnlyRootFilesystem: true
    capabilities: {drop: [ALL]}
```

Dec 2018

CVE-2018-1002105 — API server upgraded-connection handling let a request escalate to kubelet/aggregated API backends with the API server's credentials; first critical Kubernetes CVE

Feb 2019

CVE-2019-5736 (runc) — container process overwrites the host runc binary: container escape when running as root; pushed non-root containers and patched runtimes

Jan 2022

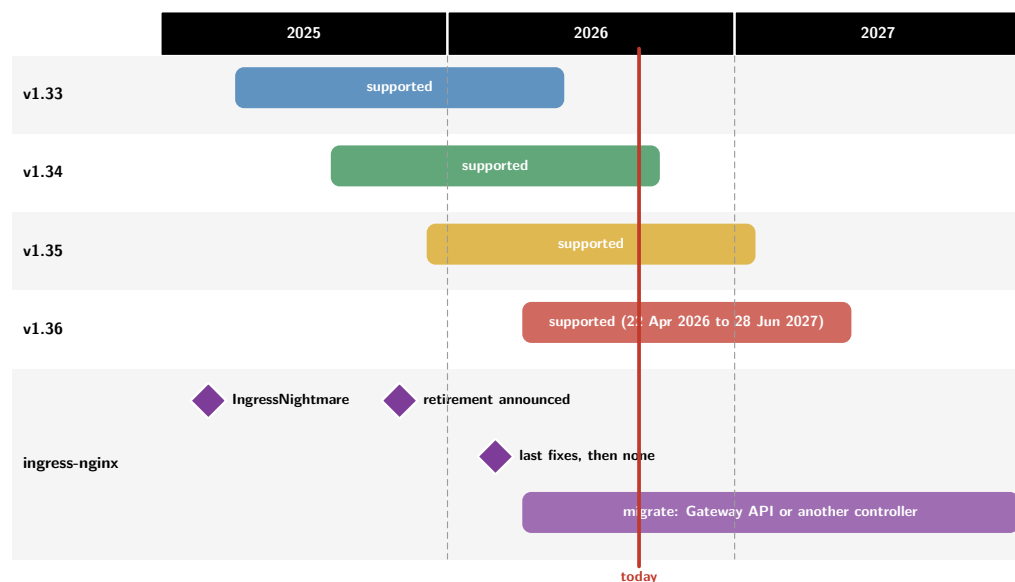
CVE-2022-0185 (kernel) — heap overflow in fs_context reachable with CAP_SYS_ADMIN in a user namespace: escape from unprivileged containers without seccomp; RuntimeDefault seccomp blocks the path

Jan 2024

CVE-2024-21626 (runc, Leaky Vessels) — leaked file descriptor to the host filesystem via WORKDIR: malicious image or exec escapes; fixed in runc 1.1.12

Mar 2025

CVE-2025-1974 (IngressNightmare) — ingress-nginx admission webhook reachable from any pod allowed configuration injection and code execution with the controller's cluster-wide secret access (CVSS 9.8); fix 1.11.5 / 1.12.1, never expose admission webhooks



Upstream support windows (about 14 months per minor version; dates approximate, verify on kubernetes.io/releases) and the ingress-nginx retirement. Managed services add paid extended support (EKS: 12 more months; AKS LTS and GKE Extended channel: up to 2 years), but an unsupported version gets no CVE fixes upstream.

Mnemonic: **RANSE**

- R** **RBAC least privilege** no wildcards, no cluster-admin, review `escalate/bind/impersonate`
- A** **Admission control** PSA restricted + ValidatingAdmissionPolicy/Kyverno for registries, signatures, labels
- N** **NetworkPolicy default deny** per namespace, egress too, block 169.254.169.254
- S** **Secrets encrypted and external** KMS v2 at rest, External Secrets Operator / CSI driver, never in git or env vars
- E** **Eyes on runtime and audit** Falco/Tetragon, audit policy, alert on exec and RBAC writes

Q1 Which seccomp setting would have blocked the CVE-2022-0185 escape path from unprivileged pods?

A `seccompProfile: RuntimeDefault` (blocks unshare of user namespaces in the default profile).

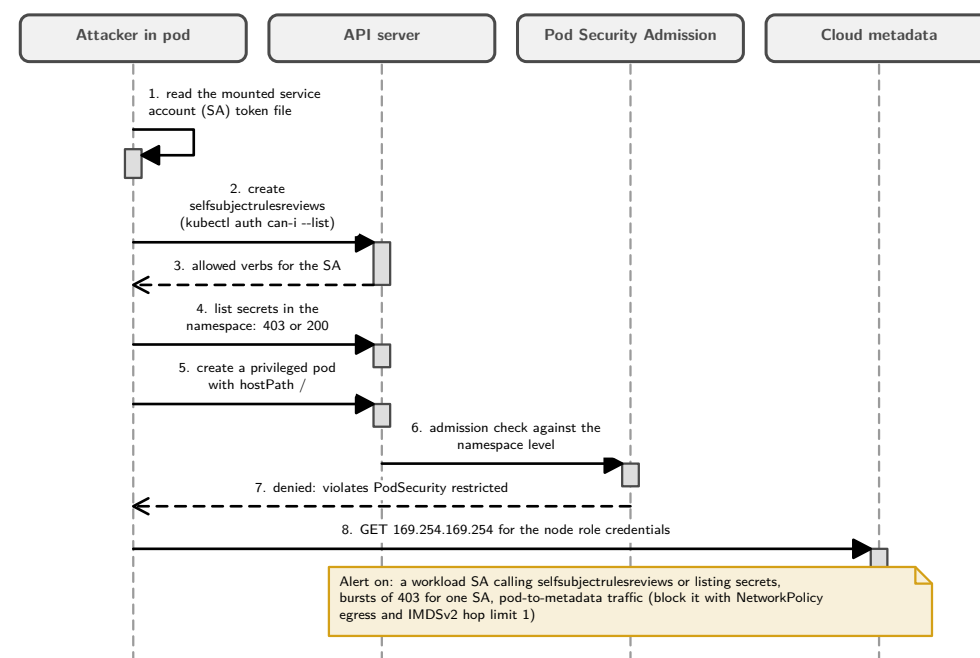
Q2 After enabling encryption at rest, why rewrite all secrets?

A Existing objects stay in plaintext in etcd until rewritten (`kubectl get secrets -A -o json | kubectl replace -f -`).

⚠ Responding to a compromised pod

- 1 Preserve before killing** deleting the pod destroys memory and the writable layer; first record `kubectl get pod -o yaml`, logs, events and the node name

- 2 Isolate the pod** add a label matched by a deny-all NetworkPolicy (ingress and egress); remove it from Service endpoints by changing its app label
- 3 Isolate the node** `kubectl cordon` (no new pods), snapshot the node disk through the cloud API, capture memory if the tooling exists
- 4 Cut identities** rotate the SA token (delete legacy token secrets, restart workloads), revoke the cloud role session (IRSA / Workload Identity) and any secret the pod could read
- 5 Scope** audit logs for the SA (pods/exec, secret reads, RBAC writes), Falco alerts, cloud API calls from the node role
- 6 Rebuild** redeploy from a clean, signed image; drain and replace the node rather than cleaning it; fix the root cause (admission rule, PSS level, NetworkPolicy)



A compromised pod probing the cluster, as the API server audit log and node telemetry show it (technique class and detection points)

```
ir@bastion: prod cluster

$ kubectl auth can-i --list --as=system:serviceaccount:shop:web | grep secrets
secrets [] [] [get list watch]

$ jq -r 'select(.user.username|test("shop:web")) | [.verb,.objectRef.resource,.responseStatus.code] | @tsv' audit.log | sort | uniq -c
14 get      configmaps 200
3 list     secrets   200
1 create   pods      403

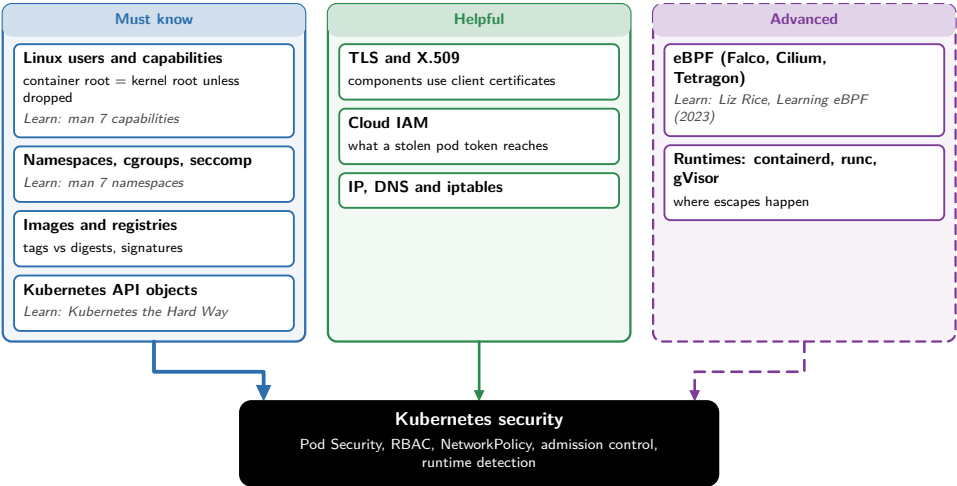
$ kubectl get ns -L pod-security.kubernetes.io/enforce
NAME      STATUS AGE  ENFORCE
shop      Active 412d restricted
batch     Active 97d   
```

Scoping step on a real cluster (fictitious names): the app's service account can read every secret in its namespace (the RBAC gap to fix), the API server audit log shows it tried a secrets list (allowed, 200) and a privileged pod create (denied by admission, 403), and one namespace has no Pod Security enforce label, so the same pod would have been admitted there

TIP

Container forensics tools such as `kubectl debug` ephemeral containers or CRIU checkpoints (kubelet checkpoint API, beta since 1.30) let you inspect or freeze a running container without restarting it; check your managed service supports them before an incident.

Learning path: what to know first



What to learn before Kubernetes security; the CKS (Certified Kubernetes Security Specialist) exam requires an active CKA (Certified Kubernetes Administrator). Every control on this sheet builds on the must-know column