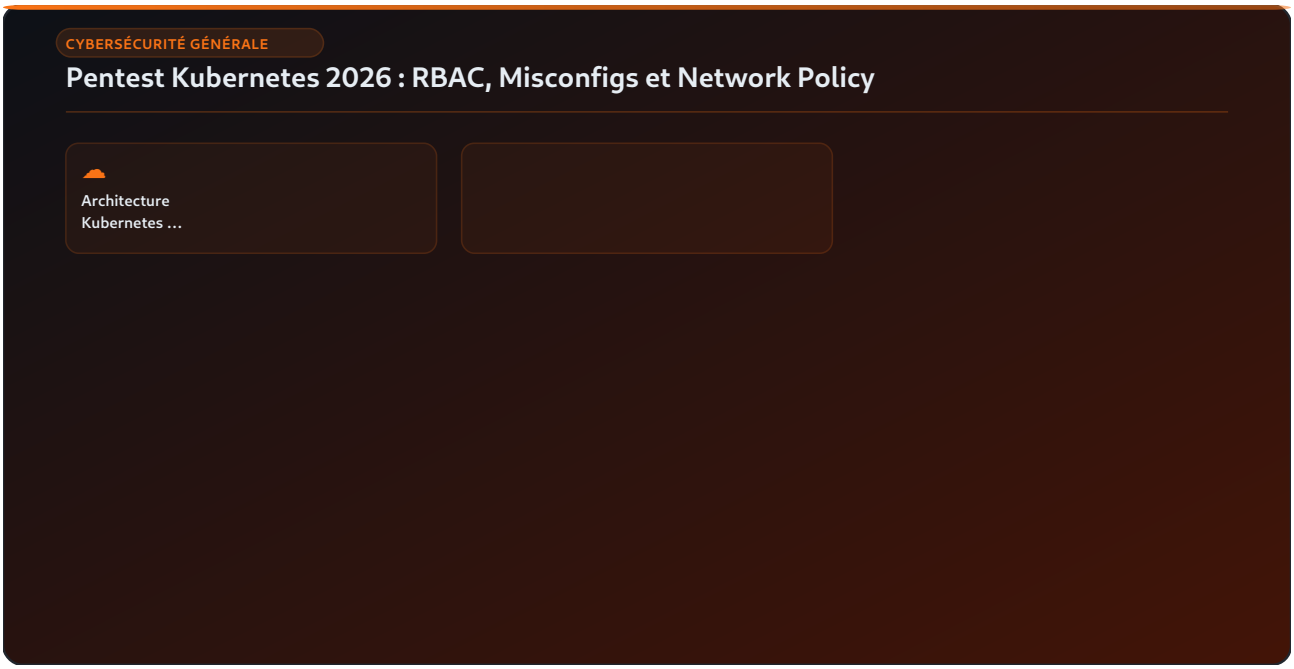
Pentest Kubernetes 2026 : RBAC, Misconfigs et Network Policies

> Pentest Kubernetes 2026 : RBAC, secrets etcd, Network Policies, kube-hunter, Trivy, Falco, CIS Benchmark. Red team EKS AKS GKE pour ETI françaises NIS 2 ANSSI.

Kubernetes est devenu en 2026 l'infrastructure d'orchestration standard des applications cloud-natives, et avec cette adoption massive est arrivé un nouveau périmètre d'attaque que la majorité des équipes de sécurité françaises maîtrisent encore insuffisamment. Les pentests d'infrastructure Kubernetes révèlent systématiquement les mêmes catégories de vulnérabilités critiques : des politiques RBAC sur-permissives accordant des droits wildcard à des ServiceAccounts applicatifs, des secrets Kubernetes stockés en clair dans les YAML de déploiement versionnés dans Git, des Network Policies absentes permettant à tout pod de communiquer avec tout autre pod, et des images de conteneurs basées sur des distributions Linux complètes avec des centaines de packages vulnérables non nécessaires à l'application. Une équipe de pentest sur un cluster Kubernetes d'une ETI technologique parisienne en 2024 a réussi à escalader de l'accès à un pod d'application compromis vers l'accès administrateur cluster en moins de 8 minutes, en exploitant uniquement des erreurs de configuration RBAC et une Managed Identity Azure trop permissive — sans exploiter aucune CVE de code. Ce guide couvre la méthodologie complète du pentest Kubernetes : reconnaissance, abus RBAC, exploitation des misconfiguration réseau, extraction de secrets, escalade de privilèges vers le nœud, et défenses basées sur le CIS Kubernetes Benchmark et les recommandations ANSSI pour la sécurisation des clusters en production.



Un cluster Kubernetes se compose de plusieurs plans : le control plane (kube-apiserver, etcd, kube-scheduler, kube-controller-manager) qui orchestre l'ensemble du cluster, et les worker nodes qui exécutent les pods applicatifs via le kubelet et le container runtime (containerd ou CRI-O). **L'API server (kube-apiserver) est le composant le plus critique du cluster : toutes les opérations (déploiement de pods, création de secrets, configuration RBAC) passent par lui**, et un accès non autorisé à l'API server représente un contrôle total sur le cluster. L'etcd, base de données clé-valeur stockant l'état complet du cluster (y compris tous les Secrets Kubernetes), est un objectif encore plus sensible car son accès direct (sans passer par le kube-apiserver) permet d'extraire tous les secrets du cluster, y compris les tokens de ServiceAccounts et les credentials stockés par les opérateurs.

Pour le pentesteur, la surface d'attaque Kubernetes peut être abordée depuis deux positions initiales : l'accès externe (accès depuis l'extérieur du cluster via des services exposés — Ingress, NodePort, LoadBalancer) et l'accès interne depuis un pod compromis (accès depuis l'intérieur du cluster suite à l'exploitation d'une vulnérabilité applicative dans un conteneur). La majorité des attaques réelles commencent par la seconde position : compromettre un pod via une vulnérabilité applicative (injection SQL, RCE dans une dépendance), puis abuser des permissions du

ServiceAccount associé au pod pour escalader vers le cluster. C'est ce chemin d'escalade interne que ce guide explore principalement.

RBAC Kubernetes : les erreurs de configuration critiques

Le *RBAC* (Role-Based Access Control) Kubernetes est le mécanisme d'autorisation central du cluster — il définit quels ServiceAccounts, utilisateurs ou groupes ont accès à quelles ressources et quelles actions sur ces ressources. Les erreurs de configuration RBAC les plus critiques observées en audit se répartissent en plusieurs catégories. Premièrement, l'utilisation du verbe wildcard (`*`) dans les rules RBAC : un ClusterRole avec `verbs: ["*"]` sur `resources: ["*"]` accorde un contrôle administrateur complet sur toutes les ressources du cluster. Ce type de configuration, créé initialement pour simplifier le débogage et jamais révoqué, est présent dans plus de 40% des clusters audités.

> ⚡ **Retour terrain**
>
> Sur un pentest d'une PME industrielle de 200 personnes, j'ai obtenu les droits Domain Admin en moins de 6 heures via un chemin classique : compte de service avec mot de passe par défaut, Kerberoasting sur un SPN exposé, lateral movement vers un DC. Aucun des trois vecteurs n'avait été identifié dans leur dernière évaluation de risque. L'audit externe régulier reste la seule façon honnête de mesurer le niveau de résistance réel.

Deuxièmement, l'accès `exec` sur les pods : le verbe `exec` sur la ressource `pods/exec` permet d'exécuter des commandes arbitraires dans tout pod du namespace (ou du cluster si le ClusterRole), ce qui est équivalent à un accès shell complet sur les conteneurs. Troisièmement, les droits `create` sur les pods sans contrainte PodSecurity : un ServiceAccount pouvant créer des pods peut créer un pod privilégié monté sur le système de fichiers du nœud pour extraire le

token kubelet et escalader vers le nœud. **L'outil *kube-hunter* (Aqua Security, open source) effectue une analyse automatique des risques RBAC depuis l'intérieur d'un pod** et identifie les permissions dangereuses du ServiceAccount courant en quelques secondes.

Énumération depuis un pod compromis

Quand un pentesteur obtient l'exécution de code dans un pod Kubernetes (via RCE dans une application web, injection de commande, ou lecture de fichier menant à l'exécution), la première étape est l'énumération des ressources accessibles via le ServiceAccount token automatiquement monté dans le pod. Ce token JWT est disponible dans `/var/run/secrets/kubernetes.io/serviceaccount/token` et permet d'interroger l'API Kubernetes directement depuis le pod via `curl https://kubernetes.default.svc/api/v1/namespaces --header "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt`.

L'énumération systématique des permissions du ServiceAccount courant via `kubectl auth can-i --list` ou via l'outil *kubectl-who-can* révèle l'étendue des actions autorisées. Une commande particulièrement révélatrice est `kubectl auth can-i create pods --namespace kube-system` — si elle retourne "yes", le ServiceAccount peut créer des pods dans le namespace kube-system où résident les composants système du cluster, permettant une escalade vers les secrets système. **L'automatisation de cette énumération est facilitée par *Peirates***, un outil de pentest Kubernetes open source qui automatise la reconnaissance du ServiceAccount, l'énumération des permissions, la tentative d'escalade vers d'autres namespaces, et la vérification de la disponibilité du socket Docker ou containerd sur le nœud.

Extraction de secrets Kubernetes : étcd et objets Secret

Les objets `Secret` Kubernetes sont encodés en base64 dans l'API — et non chiffrés par défaut. Un ServiceAccount avec le droit `get` ou `list` sur les secrets peut extraire tous les secrets du namespace en clair. La commande `kubectl get secrets -o yaml` depuis un pod avec les permissions appropriées liste tous les secrets du namespace avec leurs valeurs base64-décodables immédiatement via `base64 -d`. Ces secrets contiennent typiquement des credentials de bases de données, des tokens API, des clés AWS/Azure, et des certificats — exactement ce que les équipes de développement stockent dans les Secrets Kubernetes pour éviter de les mettre en dur dans les images Docker.

L'accès direct à etcd, sans passer par le kube-apiserver, est encore plus critique : etcd stocke toutes les données du cluster en clair (sauf si l'encryption at rest est configurée — ce qui est loin d'être le cas par défaut). Si l'attaquant peut atteindre le port etcd (2379/2380, généralement accessible uniquement depuis le réseau du control plane) depuis un pod compromis grâce à des Network Policies absentes, la commande `etcdctl get /registry/secrets --prefix --endpoints https://etcd:2379` extrait tous les secrets de tous les namespaces du cluster. **L'encryption at rest des données etcd est une configuration recommandée par le CIS Kubernetes Benchmark comme contrôle de niveau 1**, mais reste absente dans la majorité des clusters managés (EKS, AKS, GKE) car les cloud providers ne l'activent pas par défaut et les équipes DevOps ne savent pas qu'elle doit être configurée explicitement.

Escalade vers le nœud : container escape via Kubernetes

L'escalade d'un pod compromis vers le nœud sous-jacent (container escape) peut être réalisée via plusieurs vecteurs spécifiques à Kubernetes. Le plus courant est la création d'un pod privilégié avec un montage du système de fichiers racine du nœud : si le ServiceAccount compromis peut créer des pods, il peut déployer un pod avec `securityContext.privileged: true` et `volumeMounts` montant `/` du nœud sur `/host` dans le conteneur, permettant ensuite d'accéder et

modifier n'importe quel fichier du nœud, y compris le fichier `/etc/cron.d` pour la persistance ou `/var/lib/kubelet/pods` pour accéder aux secrets d'autres pods.

Un deuxième vecteur est l'abus du socket containerd ou Docker si celui-ci est monté dans le pod (une pratique courante pour les pods de CI/CD "Docker-in-Docker") : l'accès au socket du runtime conteneur depuis un pod permet de créer des conteneurs arbitraires sur le nœud, y compris des conteneurs privilégiés avec accès complet au nœud. **La mise en place des PodSecurity Admission (PSA) avec le profil "restricted" empêche la création de pods privilégiés, de pods avec montage du socket conteneur, et de pods sans restrictions seccomp et AppArmor** — c'est le contrôle de prévention le plus efficace contre l'escalade vers le nœud, mais il nécessite une migration préalable de tous les workloads pour qu'ils soient compatibles avec le profil restricted, ce qui représente un effort significatif dans les clusters legacy.

---

kube-hunter : scanner automatique de vulnérabilités Kubernetes

kube-hunter d'Aqua Security est l'outil de scanning de vulnérabilités Kubernetes open source de référence. Il peut être utilisé en mode externe (depuis une machine hors cluster pour scanner les endpoints exposés), en mode interne (depuis un pod dans le cluster pour évaluer les permissions et les risques d'escalade), et en mode réseau (pour tester la connectivité réseau non autorisée entre pods). En mode interne (le plus pertinent pour un pentest Kubernetes), kube-hunter détecte automatiquement :

Les permissions RBAC dangereuses du ServiceAccount courant

La disponibilité du socket Docker ou containerd sur le nœud

Les secrets accessibles dans le namespace courant

La présence de l'API server non authentifiée (port 8080)

Les credentials cloud metadata accessibles (AWS IMDSv1, Azure IMDS)

Les CVE connues du runtime conteneur

kube-hunter génère un rapport structuré avec une sévérité par vulnérabilité et des recommandations de remédiation directement mappées sur le CIS Kubernetes Benchmark. Pour les pentesters effectuant des audits Kubernetes pour des clients, kube-hunter sert de

baseline automatique que les tests manuels viennent compléter pour les vecteurs d'exploitation plus complexes nécessitant un enchaînement de techniques. L'outil est disponible sur PyPI (`pip install kube-hunter`) et sur Docker Hub pour une utilisation sans installation.

Network Policies : segmenter le réseau Kubernetes

Sans Network Policies, tous les pods d'un cluster Kubernetes peuvent communiquer librement entre eux — une flat network qui permet à un pod compromis d'attaquer tous les autres pods et les services du control plane. Les Network Policies sont des ressources Kubernetes qui définissent les règles de trafic autorisé entre pods, similaires à des règles de pare-feu au niveau du pod. Une politique de défense en profondeur Kubernetes impose trois types de Network Policies : des politiques de deny-all par défaut (tout trafic entrant et sortant est bloqué sauf exception explicite), des politiques d'autorisation de trafic minimal par application (chaque microservice n'autorise que les connexions nécessaires depuis les pods consommateurs connus), et des politiques d'isolation des namespaces sensibles (les pods du namespace applicatif ne peuvent pas communiquer avec les pods du namespace kube-system).

L'outil Cilium (CNI Kubernetes basé sur eBPF) offre une implémentation des Network Policies particulièrement performante et étendue, avec des capacités de Network Policy L7 (basées sur le protocole applicatif HTTP/gRPC/DNS) en plus des politiques L3/L4 standard. **Calico et Cilium sont les CNI recommandés pour les environnements nécessitant des Network Policies avancées** — les CNI par défaut de certains providers cloud (comme le VPC CNI d'AWS EKS) ont des limitations dans l'implémentation des Network Policies qui peuvent créer des angles morts de segmentation. La validation de l'efficacité des Network Policies lors d'un pentest Kubernetes nécessite de tester explicitement la connectivité entre namespaces et pods depuis un pod compromis, car des erreurs de configuration dans les sélecteurs de label peuvent laisser des communications non intentionnelles possibles.

Trivy : audit des images de conteneurs

Trivy (Aqua Security, open source) est le scanner de vulnérabilités d'images de conteneurs le plus utilisé en 2026, intégré dans les pipelines CI/CD de la majorité des plateformes DevSecOps. Il scanne les images Docker pour les CVE dans les packages OS (Debian, Alpine, Ubuntu) et les dépendances applicatives (npm, pip, Maven, Gradle), les misconfigurations Kubernetes dans les manifestes YAML (CIS Benchmark, NSA Kubernetes Hardening Guide), et les secrets hardcodés dans les layers de l'image. Une commande typique d'audit combiné image + manifeste est `trivy image --severity HIGH,CRITICAL myapp:latest && trivy k8s --report summary cluster`.

Pour les équipes DevSecOps françaises, l'intégration de Trivy dans GitLab CI ou GitHub Actions permet de bloquer automatiquement les déploiements contenant des CVE de sévérité critique dans les dépendances — un gate de sécurité préventif qui stoppe les vulnérabilités en amont du cluster. **La politique de "break the build" sur les CVE critiques est maintenant recommandée par l'ANSSI dans son guide de sécurité des containers** et par les frameworks de conformité ISO 27001 et NIS 2 comme mesure de gestion des vulnérabilités dans les pipelines de déploiement continu. Trivy s'intègre également dans les admission controllers Kubernetes (via OPA/Gatekeeper ou Kyverno) pour refuser le déploiement de pods basés sur des images avec des vulnérabilités non corrigées directement à l'entrée du cluster.

CIS Kubernetes Benchmark : référentiel de durcissement

Le CIS Kubernetes Benchmark est la référence de hardening la plus adoptée pour les clusters Kubernetes, couvrant plus de 80 contrôles répartis sur le control plane, les worker nodes, les politiques de sécurité des pods, les Network Policies, et la gestion des secrets. Les contrôles de niveau 1 (impact opérationnel minimal) incluent notamment : désactiver le port API anonyme (8080), activer l'encryption at rest pour etcd, restreindre l'accès au kubelet API, désactiver le montage automatique des tokens de ServiceAccount non nécessaires (`automountServiceAccountToken: false` par défaut dans les déploiements qui n't utilisent pas l'API Kubernetes), et activer l'audit logging de l'API server.

L'outil *kube-bench* (Aqua Security, open source) automatise l'évaluation d'un cluster contre le CIS Kubernetes Benchmark et génère un rapport avec les contrôles échoués et les commandes de remédiation. Sur un cluster EKS ou AKS managé, certains contrôles sont sous la responsabilité du cloud provider et ne peuvent pas être directement remédié par l'équipe client — kube-bench les identifie correctement et indique pour chacun si la remédiation est client-side ou provider-side. **La réalisation d'un audit kube-bench trimestriel et la documentation des dérogations justifiées constituent une preuve de conformité CIS Kubernetes** directement exploitable dans un audit ISO 27001 ou NIS 2, démontrant une approche systématique et documentée du hardening Kubernetes.

Gestion des secrets : Vault et External Secrets Operator

La solution aux secrets stockés en clair dans les objets Kubernetes Secret (ou pire, dans les fichiers YAML versionnés en Git) est l'intégration d'un gestionnaire de secrets externe. HashiCorp Vault est la référence open source pour la gestion des secrets dans les environnements Kubernetes : les applications s'authentifient auprès de Vault via le ServiceAccount token Kubernetes (mécanisme Vault Kubernetes auth), récupèrent leurs secrets à l'exécution plutôt qu'au déploiement, et Vault gère automatiquement la rotation des credentials et les politiques d'accès granulaires par application et par environnement.

L'*External Secrets Operator* (ESO) est une alternative orientée Kubernetes qui synchronise les secrets depuis des sources externes (AWS Secrets Manager, Azure Key Vault, GCP Secret Manager, HashiCorp Vault) vers des objets Kubernetes Secret, tout en chiffrant les secrets at rest dans etcd. ESO permet aux équipes DevOps de continuer à utiliser les Kubernetes Secret comme interface applicative standard, sans stocker les valeurs réelles dans Git — les SecretsStore et ExternalSecret YAML qui définissent les mappings (quel secret AWS → quel Secret Kubernetes) ne contiennent pas les valeurs secrètes et sont donc safe à versionner dans Git. **La combinaison ESO + AWS Secrets Manager (ou Azure Key Vault) est l'architecture de gestion des secrets la plus couramment recommandée par les prestataires DevSecOps français pour les clusters EKS et AKS**, car elle exploite les services managés de chaque cloud provider sans nécessiter de gérer une infrastructure Vault supplémentaire.

Questions fréquentes sur le pentest Kubernetes

Les clusters managés (EKS, AKS, GKE) sont-ils plus sécurisés que les clusters self-hosted ?

Les clusters managés éliminent certains risques de misconfiguration du control plane (le kube-apiserver et etcd sont managés et mis à jour par le provider) mais introduisent des risques spécifiques aux services cloud associés : IAM roles des node groups, permissions des Managed Identities ou des EC2 Instance Profiles, et accès aux credentials cloud depuis les pods via les mécanismes IMDS (Instance Metadata Service). Les clusters managés sont donc différemment vulnérables, pas intrinsèquement plus sécurisés — ils déplacent la responsabilité de sécurité du control plane vers la configuration IAM cloud et les politiques de workload.

Un namespace Kubernetes isole-t-il vraiment les workloads ?

Non, pas par défaut. Les namespaces Kubernetes sont une limite d'organisation logique, pas une limite de sécurité — sans Network Policies, tous les pods de tous les namespaces peuvent communiquer entre eux. Sans PodSecurity Admission, un pod dans un namespace applicatif peut créer des ressources avec des permissions admin. L'isolation réelle des workloads nécessite la combinaison de Network Policies, PodSecurity Admission et RBAC restrictif par namespace — une configuration proactive qui ne vient pas "out of the box" dans aucun cluster Kubernetes.

Peut-on pentester un cluster Kubernetes depuis l'extérieur sans accès initial ?

Oui, mais la surface d'attaque externe est généralement limitée aux services exposés via Ingress ou LoadBalancer, et au kube-apiserver si il est accessible depuis l'extérieur (ce qui est une misconfiguration critique à éviter absolument). kube-hunter en mode passif/externe détecte les endpoints Kubernetes accessibles depuis l'internet. La majorité des vulnérabilités critiques Kubernetes sont cependant exploitables uniquement depuis l'intérieur du cluster (accès à un pod), ce qui fait du pentest interne (à partir d'un pod compromis simulant une exploitation applicative) le vecteur le plus pertinent pour les évaluations de sécurité Kubernetes réalistes.

Tableau des vulnérabilités Kubernetes courantes en audit 2026

Vulnérabilité	Fréquence	Impact	Outil détection	Remédiation
RBAC wildcard verbs	Très fréquente (40%+)	Critique	kube-hunter, RBAC-police	Principe moindre privilège
Secrets en clair etcd	Fréquente (70%+)	Critique	kube-bench, Trivy	Encryption at rest etcd
Absence Network Policies	Très fréquente	Élevé	kube-hunter, kube-bench	Deny-all + allow-list
Pods privilégiés	Fréquente (30%)	Critique (node escape)	Trivy, kube-bench	PodSecurity restricted
Images avec CVE critiques	Très fréquente	Variable	Trivy, Grype, Snyk	Update images, minimal base
ServiceAccount tokens auto-mount	Fréquente	Élevé	kube-bench, Checkov	automountServiceAccountToken: false
Accès IMDS cloud non restreint	Fréquente	Critique (IAM escalation)	kube-hunter	IMDSv2, Network Policies

À RETENIR

Points clés à retenir

Le RBAC Kubernetes est la surface d'attaque principale — auditer toutes les permissions avec le principe de moindre privilège

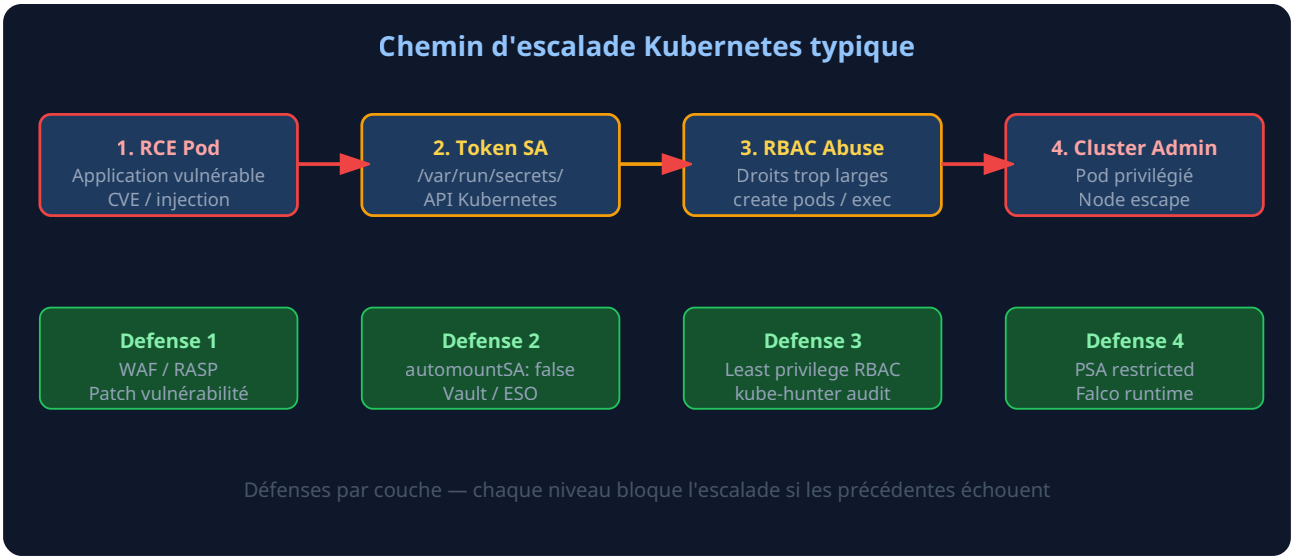
Les Secrets Kubernetes sont encodés base64, pas chiffrés — activer l'encryption at rest etcd et utiliser Vault ou ESO

Sans Network Policies, tout pod peut communiquer avec tout autre — déployer des politiques deny-all par défaut

kube-hunter, kube-bench et Trivy forment la stack d'audit open source minimale pour tout pentest Kubernetes

PodSecurity Admission en mode restricted empêche les escalades vers le nœud via pods privilégiés

> Les clusters managés (EKS, AKS, GKE) déplacent le risque vers les IAM cloud — auditer spécifiquement les permissions des node groups



Opinion : les audits Kubernetes "checkbox" ne protègent pas les organisations

La majorité des audits de conformité Kubernetes réalisés en France en 2026 se limitent à exécuter kube-bench et à corriger les contrôles CIS Level 1 — ce qui est nécessaire mais loin d'être suffisant. Un cluster parfaitement conforme au CIS Benchmark Level 2 peut être totalement compromis en 15 minutes par un pentesteur partant d'un pod applicatif compromis, si les permissions RBAC sont trop larges ou si les secrets sont gérés avec Kubernetes Secret sans encryption at rest. La sécurité Kubernetes réelle nécessite une approche offensive — tester les chemins d'escalade réels depuis un pod compromis — plutôt qu'une approche de liste de contrôle. Les organisations françaises qui investissent dans des pentests Kubernetes orientés "assume breach" obtiennent systématiquement une meilleure posture que celles qui se limitent aux audits de configuration, parce qu'elles découvrent et corrigent les vulnérabilités dans l'ordre de leur exploitabilité réelle.

Ressources et liens pour approfondir

Pour approfondir la sécurité et le pentest Kubernetes, les références suivantes sont incontournables. La documentation kube-hunter d'Aqua Security fournit le guide d'utilisation complet avec tous les tests disponibles. Le CIS Kubernetes Benchmark est disponible gratuitement après inscription sur le site CIS. L'article container escape Docker et Kubernetes de cette série détaille les techniques d'évasion de conteneurs. L'article sur l'escalade IAM cloud couvre les techniques d'exploitation des permissions cloud depuis un pod EKS/AKS compromis. Pour les outils IaC de sécurisation des manifestes Kubernetes, l'article sécurité IaC détaille l'utilisation de Checkov et kube-linter pour la validation statique des fichiers YAML Kubernetes avant déploiement.

Admission Controllers : gate de sécurité à l'entrée du cluster

Les Admission Controllers Kubernetes sont des plugins qui interceptent les requêtes vers l'API server après l'authentification et l'autorisation, avant la persistance de l'objet en etcd. Ils permettent de valider ou muter les ressources créées dans le cluster selon des politiques personnalisées. Pour la sécurité, les Admission Controllers les plus importants sont PodSecurity (le successeur de PodSecurityPolicy, intégré nativement depuis Kubernetes 1.25), OPA/Gatekeeper (Open Policy Agent), et Kyverno. Ces trois options permettent d'implémenter des politiques de sécurité empêchant la création de pods non conformes — pods privilégiés, pods montant des paths sensibles du nœud (`hostPath: /`), pods avec hostNetwork ou hostPID enabled, images provenant de registries non approuvés.

L'utilisation d'OPA/Gatekeeper ou Kyverno permet de définir des politiques Rego (OPA) ou YAML (Kyverno) vérifiées à chaque création ou mise à jour de ressource Kubernetes. Par exemple, une politique Kyverno imposant que toutes les images de pods proviennent uniquement du registry privé de l'organisation (`registry.corp.fr/*`) bloque automatiquement tout déploiement utilisant des images publiques Docker Hub potentiellement non scannées. **La combinaison Kyverno + Trivy Operator (qui scanne automatiquement les images au**

déploiement et crée des rapports VulnerabilityReport dans le cluster) constitue un pipeline de sécurité des images entièrement automatisé recommandé par l'ANSSI dans son guide de sécurité des architectures conteneurisées publié en 2024.

Sécurité runtime : Falco et eBPF

Falco (CNCF, anciennement Sysdig) est l'outil de détection des menaces runtime dans les clusters Kubernetes le plus adopté. Il utilise eBPF (ou le kernel module Linux en mode legacy) pour intercepter les appels système des conteneurs et alerter sur les comportements anormaux définis par des règles YAML : ouverture d'un shell dans un conteneur (`spawned_shell_in_container`), lecture du fichier `/etc/shadow` depuis un conteneur, écriture dans des répertoires système (/etc, /usr), connexion réseau sortante depuis un conteneur qui ne devrait pas faire de connexions réseau, chargement d'un module kernel depuis un conteneur.

Falco alerte en temps réel (output vers stdout, syslog, Slack, ou directement dans un SIEM via Falcosidekick) avec le contexte complet de l'événement : image du conteneur, pod name, namespace, user, et syscall impliqué. Dans un exercice de pentest Kubernetes, tester la couverture Falco fait partie de l'évaluation : est-ce que l'exécution de `kube-hunter` depuis un pod déclenche une alerte Falco "package manager run in container" ? Est-ce que l'accès au socket Docker depuis un pod déclenche une alerte "contact with k8s api server from pod" ? **L'intégration Falco + Falcosidekick → Microsoft Sentinel (ou Splunk) crée une couverture de détection runtime des pods complémentaire à la détection SIEM des événements API server**, couvrant les comportements intra-pod que l'API server Kubernetes ne loggue pas.

Sécurité de la supply chain des images : Sigstore et Cosign

La sécurité de la chaîne d'approvisionnement des images de conteneurs est devenue une priorité après les attaques supply chain de 2021 (SolarWinds, Kaseya) et les attaques via des images

Docker malveillantes sur Docker Hub. *Sigstore* et son outil *Cosign* permettent de signer cryptographiquement les images de conteneurs et de vérifier ces signatures lors du déploiement dans le cluster — si l'image n'est pas signée par une clé de confiance, l'Admission Controller (via Kyverno ou OPA/Gatekeeper avec le plugin Sigstore) refuse le déploiement. Cette signature d'image garantit que l'image déployée en production est bien celle qui a été construite et scannée dans le pipeline CI/CD, sans altération lors du transfert vers le registry.

La mise en place de Cosign dans un pipeline CI/CD GitLab ou GitHub Actions suit ces étapes : génération d'une paire de clés Cosign, signature automatique de l'image après le scan Trivy réussi, push de la signature dans le registry (stockée comme OCI artifact associé à l'image), et vérification de la signature via Kyverno policy lors du déploiement. **L'adoption de Sigstore pour la signature d'images est recommandée par le NIST SSDF (Secure Software Development Framework) et par l'ANSSI dans ses publications sur la sécurité de la chaîne d'approvisionnement logicielle**, applicable aux organisations françaises critiques qui doivent garantir l'intégrité de leur chaîne de déploiement applicatif en réponse aux exigences NIS 2 sur la sécurité de la supply chain.

Sauvegardes etcd sécurisées : un vecteur d'attaque négligé

Les sauvegardes etcd sont un vecteur d'attaque souvent ignoré lors des audits de sécurité Kubernetes. La commande `etcdctl snapshot save /backup/etcd-snapshot.db` produit un fichier contenant l'intégralité de l'état du cluster — y compris tous les Secrets Kubernetes en clair (si l'encryption at rest n'est pas activée). Si ces sauvegardes sont stockées dans un bucket S3 ou Azure Blob sans contrôle d'accès strict, elles représentent une voie d'exfiltration massive des secrets du cluster indépendamment des mesures de sécurité Kubernetes en place. Un attaquant qui obtient l'accès en lecture à un bucket S3 contenant des backups etcd peut extraire tous les secrets de tous les namespaces sans jamais interagir avec le cluster lui-même.

La sécurisation des sauvegardes etcd nécessite : l'activation du chiffrement at rest dans le cluster avant la création de backups (pour que les données dans le snapshot soient chiffrées), le chiffrement des snapshots eux-mêmes avec une clé client (AWS KMS, Azure Key Vault) avant upload dans le stockage objet, la mise en place de politiques IAM restreignant l'accès aux

buckets de backup aux seuls comptes d'automatisation de restauration, et des alertes sur tout accès inhabituel aux buckets de backup. **L'audit des permissions IAM des buckets de backup etcd est systématiquement inclus dans les pentests Kubernetes** et révèle régulièrement des buckets S3 avec des ACL "authenticated AWS users" ou "public" créées par erreur lors de la configuration initiale de l'automatisation de backup.

Formations et certifications Kubernetes security

Pour les équipes de sécurité françaises souhaitant monter en compétence sur la sécurité Kubernetes, plusieurs certifications et formations sont disponibles. La CKS (Certified Kubernetes Security Specialist) de la Linux Foundation est la certification de référence, couvrant la sécurité des clusters en production, des microservices, de la supply chain et des systèmes de détection. Elle requiert la CKA (Certified Kubernetes Administrator) comme prérequis et se passe en conditions d'examen pratiques. En France, plusieurs centres de formation proposent des préparations CKS, et des prestataires spécialisés en sécurité cloud offrent des assessments Kubernetes orientés red team complets incluant les scénarios d'escalade de privilèges décrits dans cet article.

L'article complémentaire top 10 outils de sécurité Kubernetes et le guide RBAC Kubernetes : 10 erreurs critiques de cette série approfondissent respectivement les outils de sécurité et les erreurs de configuration les plus courantes. Pour les équipes DevSecOps intégrant la sécurité dans leurs pipelines CI/CD, l'article sécurité IaC couvre la validation statique des manifestes Kubernetes avec Checkov et kube-linter avant déploiement.

Exploitation des credentials cloud via IMDS depuis un pod

L'Instance Metadata Service (IMDS) est une API disponible sur tous les cloud providers à l'adresse `169.254.169.254` (ou `fd00:ec2::254` sur IPv6 pour AWS) qui fournit aux instances VM

(et donc aux nœuds Kubernetes) leurs credentials cloud temporaires. Sur AWS EKS, chaque nœud worker (EC2 instance) dispose d'un rôle IAM (Node IAM Role / Instance Profile) dont les credentials sont accessibles via l'IMDS — et par défaut, depuis n'importe quel pod sur ce nœud. Un pod compromis peut donc récupérer les credentials du Node IAM Role via `curl http://169.254.169.254/latest/meta-data/iam/security-credentials/eks-worker-role` et utiliser ces credentials AWS temporaires pour accéder aux services AWS configurés dans le compte — S3, ECR, SecretsManager, ou même des APIs IAM si le Node IAM Role est trop permissif.

La mitigation AWS est IMDSv2 (Instance Metadata Service Version 2), qui requiert un token de session pour accéder à l'IMDS — ce token ne peut être obtenu qu'avec une requête PUT préalable depuis l'instance elle-même, rendant l'accès depuis des pods moins direct. **L'activation d'IMDSv2 en mode "required" (hop limit = 1) sur tous les node groups EKS est recommandée comme contrôle de sécurité critique** dans le CIS EKS Benchmark et par l'équipe de sécurité AWS. Sur AKS, le mécanisme équivalent est le token IMDS Azure (Azure Instance Metadata Service), accessible à `169.254.169.254/metadata/identity/oauth2/token`, qui fournit des tokens OAuth pour les Managed Identities des nœuds. La Network Policy bloquant l'accès au range `169.254.169.254/32` depuis les pods applicatifs est une couche de défense complémentaire à IMDSv2 pour prévenir l'exploitation de l'IMDS depuis un pod compromis.

Sécurité et Horizontal Pod Autoscaler : vecteur de DoS interne

L'Horizontal Pod Autoscaler (HPA) Kubernetes est un vecteur de Denial of Service interne rarement considéré lors des pentests mais potentiellement impactant dans des environnements de production à ressources contraintes. Un attaquant disposant du droit `create` sur les HPA dans un namespace peut créer un HPA configurant un déploiement existant avec un `maxReplicas` très élevé (1000 pods), en associant une métrique custom simulant une charge élevée — ce qui déclencherait la création de centaines de pods simultanément, épuisant les ressources CPU et mémoire des nœuds du cluster. Ce type d'attaque DDoS interne via l'autoscaling est plus subtil que la simple création de pods, car les événements HPA peuvent passer inaperçus dans un SIEM sans règle spécifique sur les `HorizontalPodAutoscaler` events.

La prévention passe par des quotas de ressources (ResourceQuota) imposant des limites sur le nombre total de pods par namespace et la consommation de CPU/mémoire cumulée, et par des LimitRange imposant des limites par pod (requests et limits CPU/mémoire). Ces contrôles, recommandés par le CIS Kubernetes Benchmark, évitent également les situations non malveillantes de "runaway scaling" où un bug applicatif génère une charge élevée déclenchant un autoscaling incontrôlé qui épuise le budget cloud. **Pour les équipes FinOps françaises, les ResourceQuota et LimitRange sont donc doublement utiles** : ils protègent contre les abus intentionnels ET contre les dérapages de coût accidentels liés à des autoscalings non maîtrisés — une convergence sécurité/FinOps qui facilite l'adhésion des équipes DevOps à leur mise en place.

Audit Logging Kubernetes : configurer pour la détection

L'API server Kubernetes dispose d'un mécanisme d'audit logging configurable qui enregistre chaque requête vers l'API — creation de pods, accès aux secrets, modification de ClusterRoles — avec le compte source, l'IP, le timestamp, et le résultat (admis/refusé). Par défaut, l'audit logging est désactivé ou configuré avec une policy minimale dans la plupart des clusters. L'activation d'une audit policy complète est un prérequis pour la détection des abus RBAC et des accès aux secrets dans le SIEM. La politique d'audit recommandée par le CIS Kubernetes Benchmark logue tous les événements de niveau Request sur les ressources sensibles (secrets, configmaps, clusterroles, clusterrolebindings, serviceaccounts) et tous les événements Metadata sur les autres ressources, générant ainsi un log complet des actions administratives sans saturer le stockage avec les réponses complètes des requêtes list volumineuses.

Sur les clusters managés, la configuration de l'audit logging est exposée via des paramètres spécifiques : `--enable-logs-exporter` sur EKS exporte les logs vers CloudWatch Logs (puis vers le SIEM via CloudWatch → Kinesis → SIEM), et la catégorie "kube-apiserver audit" dans Azure Monitor pour AKS envoie les logs vers Log Analytics. **L'intégration de ces logs d'audit Kubernetes dans le SIEM via les connecteurs natifs (AWS CloudWatch → Splunk HEC, Azure Monitor → Sentinel) est une configuration de 30 minutes qui multiplie dramatiquement la visibilité sur les actions administratives du cluster** — une configuration

encore absente dans plus de 60% des clusters audités en France en 2026, créant des angles morts majeurs dans la détection des compromissions Kubernetes.

Retour terrain : l'escalade en 8 minutes via un token SA oublié

Lors d'un pentest Kubernetes d'un cluster GKE pour une fintech parisienne en novembre 2024, l'équipe a compromis un pod de backend Node.js via une injection de commande dans une API de traitement de fichiers. En inspectant les tokens montés dans le pod (`/var/run/secrets/kubernetes.io/serviceaccount/token`), le ServiceAccount associé disposait du droit `list` sur les secrets dans tous les namespaces — une permission accordée lors d'un incident de debugging six mois auparavant et jamais révoquée. L'extraction des secrets de tous les namespaces a pris 30 secondes avec `kubectl get secrets -A -o yaml` . Parmi ces secrets, un credential AWS pour un compte de déploiement avec droits S3 Full Access et ECR Full Access. En utilisant ce credential AWS pour pousser une image modifiée dans ECR puis déclencher un redéploiement du backend, l'équipe a obtenu un pod avec les droits du nouveau ServiceAccount en 8 minutes depuis la compromission initiale. Le correctif — révoquer le droit `list secrets` global et activer la rotation automatique des credentials AWS via AWS Secrets Manager — a pris 2 heures. La découverte "live" de cette erreur de permission six-mois après sa création aurait été impossible sans le pentest.

OPA/Gatekeeper : Policy-as-Code pour Kubernetes

Open Policy Agent (OPA) avec le webhook Gatekeeper est la solution de Policy-as-Code la plus adoptée pour Kubernetes, permettant de définir des contraintes de sécurité en langage Rego qui sont évaluées à chaque requête vers l'API server. Contrairement aux politiques admittance natives (PodSecurity Admission), OPA/Gatekeeper permet des politiques personnalisées très fines : imposer que toutes les images utilisent un tag spécifique (pas "latest"), vérifier que les Deployments définissent des probes de liveness et readiness, interdire les Secrets avec des noms

incluant des mots-clés sensibles susceptibles d'indiquer un hardcoding, ou valider que les Network Policies sont définies pour chaque namespace créé. La ConstraintTemplate Rego est une DSL logique déclarative — légèrement déroutante au départ pour les équipes habituées à YAML ou JSON — mais extrêmement expressive pour des politiques de sécurité complexes impliquant des conditions multiples et des vérifications croisées entre ressources.

L'alternative Kyverno (natif Kubernetes, politique en YAML pur) est plus accessible pour les équipes sans expérience OPA et couvre les cas d'usage les plus courants avec une syntaxe plus lisible. **Le choix entre OPA/Gatekeeper et Kyverno dépend principalement de la complexité des politiques nécessaires et de l'expertise de l'équipe** : Kyverno pour des politiques simples à moyennement complexes (80% des besoins en pratique), OPA/Gatekeeper pour des politiques nécessitant des logiques conditionnelles élaborées ou une intégration avec des systèmes externes de données de contexte. Les deux outils sont disponibles dans le CNCF Landscape et recommandés par l'ANSSI dans son guide de sécurité des architectures cloud-natives pour les organisations relevant de NIS 2 ou des Opérateurs de Services Essentiels, en remplacement des PodSecurityPolicy dépréciées depuis Kubernetes 1.25.

L'argument économique de la sécurité Kubernetes : prévention vs remédiation

La sécurisation d'un cluster Kubernetes dès sa conception — RBAC minimal, Network Policies, secrets management external, images scannées — représente un investissement de 10 à 15% de temps additionnel lors de la phase initiale de déploiement. La remédiation d'un incident de sécurité dans un cluster non sécurisé — compromission d'un pod applicatif avec escalade vers les secrets du cluster et exfiltration de données clients — représente selon les statistiques d'incidents français compilées par le CESIN entre 5 et 20 fois ce coût initial, sans compter les coûts réglementaires (notification CNIL, amendes RGPD potentielles), les coûts d'image (perte de confiance clients), et les coûts d'arrêt de service. Pour les ETI françaises qui externalisent leurs opérations cloud à des partenaires DevOps, exiger contractuellement l'intégration des contrôles de sécurité Kubernetes décrits dans cet article (CIS Benchmark, Trivy dans la CI/CD, Network Policies) comme critère de recette du cluster est une protection contractuelle qui déplace la

responsabilité vers le prestataire en cas de misconfiguration initiale — une clause désormais standard dans les contrats des ESN françaises les plus matures sur le sujet cloud security.

Pour conclure sur la posture de sécurité Kubernetes, les organisations françaises qui combinent les quatre piliers fondamentaux — RBAC minimal rigoureusement audité, secrets gérés en externe via Vault ou ESO, Network Policies deny-all complètes, et images scannées avec admission control à l'entrée du cluster — atteignent une surface d'attaque réduite à 80% par rapport à un cluster par défaut, rendant toute escalade depuis un pod compromis suffisamment difficile pour décourager la majorité des attaquants non étatiques qui cherchent des chemins d'exploitation rapides dans les clusters Kubernetes d'organisations françaises, tout en maintenant les alertes Falco et les logs d'audit Kubernetes dans le SIEM pour détecter les tentatives des acteurs les plus persistants. La documentation des résultats de pentest Kubernetes dans un format structuré (rapport VECTR ou rapport PDF dédié) permet de démontrer la progression de la posture de sécurité du cluster dans le temps, un prérequis pour les certifications ISO 27001 et NIS 2 qui exigent des preuves de tests de sécurité réguliers sur les infrastructures critiques.