AYI NEDJIMI Consultants

Kubernetes Security : RBAC et Network Policies 2026

Catégorie : Cloud Security Lecture : 8 min Publié le : 05/03/2026 Auteur : Ayi NEDJIMI

Sécurisez Kubernetes avec RBAC et Network Policies : configuration avancée des rôles, segmentation réseau des pods, audit et bonnes pratiques 2026.

Résumé exécutif

RBAC et Network Policies sont les deux piliers fondamentaux de la sécurité Kubernetes. Ce guide détaille leur configuration avancée, les erreurs courantes à éviter et les stratégies de déploiement pour des clusters de production sécurisés.

Quiconque a déjà hérité d'un cluster Kubernetes en production sait que la configuration par défaut est dangereusement permissive. Par défaut, tous les pods peuvent communiquer entre eux sans restriction, les comptes de service disposent de tokens montés automatiquement, et les rôles RBAC initiaux sont souvent créés avec des wildcards par commodité pendant la phase de développement puis jamais restreints. Cette permissivité initiale transforme chaque cluster Kubernetes en terrain de jeu idéal pour un attaquant qui a obtenu un premier point d'entrée. Après avoir audité plus de quarante clusters Kubernetes en production dans des secteurs variés allant de la fintech à l'industrie pharmaceutique, je constate que les mêmes erreurs de configuration RBAC et Network Policy reviennent systématiquement, et que leur correction réduit drastiquement la surface d'attaque exploitable lors des exercices de pentest internes que nous conduisons régulièrement.

Comment fonctionne le RBAC Kubernetes en détail ?

Le *Role-Based Access Control* de Kubernetes repose sur quatre objets API : **Role** (permissions dans un namespace), **ClusterRole** (permissions cluster-wide), **RoleBinding** (association role-sujet dans un namespace) et **ClusterRoleBinding** (association cluster-wide). Chaque Role définit des règles composées de trois éléments : les **apiGroups** (core, apps, rbac.authorization.k8s.io...), les **resources** (pods, deployments, secrets, configmaps...) et les **verbs** (get, list, watch, create, update, patch, delete).

Le piège principal réside dans l'utilisation des wildcards. Un `ClusterRole` avec `resources: ["*"]` et `verbs: ["*"]` confère des permissions équivalentes à cluster-admin. Même un Role namespaced avec des wildcards permet l'accès aux Secrets du namespace, incluant les tokens de service accounts et les credentials applicatifs. La documentation officielle de Kubernetes Security détaille chaque verb et ses implications. Pour les techniques d'exploitation RBAC, notre article sur attaques RBAC Kubernetes couvre les vecteurs d'attaque spécifiques.

Configuration	Risque	Impact	Remédiation
ClusterRoleBinding cluster-admin	Critique	Contrôle total du cluster	Limiter aux break-glass
Wildcard resources/verbs	Haut	Accès non restreint	Lister explicitement
automountServiceAccountToken	Haut	Token SA dans chaque pod	Désactiver par défaut
Escalate/bind/impersonate verbs	Critique	Escalade de privilèges	Restreindre strictement
Secrets access broad	Haut	Exfiltration credentials	Limiter par nom

Mon avis : La majorité des clusters Kubernetes que j'audite ont des ClusterRoleBindings vers cluster-admin pour des comptes de service applicatifs. C'est l'équivalent de donner un accès root à chaque application sur un serveur Linux. Le RBAC Kubernetes est puissant mais sa granularité rend la configuration correcte chronophage — investissez ce temps, il vous sauvera lors du prochain incident.

Quelles permissions RBAC sont dangereuses ?

Certains verbs RBAC sont intrinsèquement dangereux et doivent être strictement contrôlés. Le verb **escalate** sur les resources roles/clusterroles permet de s'attribuer des permissions supérieures à celles qu'on possède. Le verb **bind** sur les rolebindings/clusterrolebindings permet de s'associer à des rôles existants plus privilégiés. Le verb **impersonate** permet d'agir en tant qu'un autre utilisateur ou service account. La combinaison `create` sur les pods avec le montage de service account tokens permet de lancer un pod avec les permissions d'un service account privilégié.

Au-delà des verbs, certaines combinaisons de resources et verbs sont dangereuses : `get secrets` dans un namespace sensible expose tous les credentials, `create pods/exec` permet l'exécution de commandes dans des pods existants (équivalent SSH), et `patch deployments` permet de modifier l'image d'un conteneur pour injecter du code malveillant. Notre guide sur les évasion de conteneur Docker et les techniques d'attaques CI/CD GitOps détaille comment ces permissions sont exploitées en pratique lors des pentests.

Network Policies : micro-segmentation des pods

Les *Network Policies* sont le mécanisme natif Kubernetes pour contrôler le trafic réseau entre les pods. Par défaut, sans Network Policy, tous les pods peuvent communiquer entre eux — c'est le modèle "flat network" qui facilite le mouvement latéral. Une Network Policy définit des règles d'ingress et/ou d'egress pour les pods sélectionnés par des labels. Dès qu'une Network Policy cible un pod, celui-ci passe en mode "deny by default" pour le type de trafic concerné (ingress, egress ou les deux).

Stratégie recommandée : commencez par déployer une **Default Deny All** Network Policy dans chaque namespace, puis ajoutez progressivement des règles d'autorisation spécifiques. Cette approche whitelist est l'inverse de l'approche blacklist mais offre une sécurité nettement supérieure. Attention : les Network Policies nécessitent un CNI compatible (Calico, Cilium, WeaveNet) — le CNI par défaut de certaines distributions ne les supporte pas.

Les principes de segmentation réseau détaillés dans notre article sur segmentation réseau VLAN firewall sont directement transposables à l'environnement Kubernetes via les Network Policies.

Chez un éditeur SaaS avec un cluster EKS de 200 pods, nous avons implémenté des Network Policies Calico en mode audit pendant deux semaines pour identifier les flux légitimes, puis basculé en mode enforcement. Le nombre de connexions inter-pods autorisées est passé de 40 000 (tout communique avec tout) à 847 flux explicitement autorisés. Lors du pentest suivant, l'attaquant qui avait compromis un pod frontend n'a pas pu pivoter vers les pods backend ni accéder au pod de base de données, contrairement au pentest précédent où il avait atteint la base de données en moins de dix minutes.

Comment auditer les configurations RBAC existantes ?

L'audit RBAC commence par l'inventaire des ClusterRoleBindings et RoleBindings. Utilisez `kubectl auth can-i --list` pour vérifier les permissions effectives d'un sujet donné. L'outil open-source **kubectl-who-can** inverse la question : "qui peut effectuer cette action sur cette ressource ?" L'outil **rakkess** affiche une matrice de permissions par ressource. Pour un audit complet, **KubeAudit** et **kube-bench** évaluent la configuration du cluster contre les CIS Kubernetes Benchmark.

En production, activez les **Audit Logs** Kubernetes au niveau RequestResponse pour les opérations sensibles (secrets, RBAC, exec) et Request pour le reste. Envoyez ces logs vers votre SIEM pour détecter les tentatives d'escalade de privilèges en temps réel. Les techniques d'audit Terraform décrites dans audit Terraform compliance complètent cette approche en vérifiant la conformité des manifestes Kubernetes avant le déploiement. L'ANSSI publie des recommandations complémentaires applicables aux clusters Kubernetes hébergés en France.

Faut-il utiliser des OPA Gatekeeper ou Kyverno ?

Les **Admission Controllers** comme OPA Gatekeeper et Kyverno ajoutent une couche de contrôle qui bloque les déploiements non conformes avant qu'ils n'atteignent le cluster. **OPA Gatekeeper** utilise le langage Rego pour définir des contraintes, tandis que **Kyverno** utilise des policies déclaratives en YAML natif Kubernetes. Les deux peuvent enforcer des règles telles que : interdire les conteneurs root, exiger des resource limits, bloquer les images non signées, imposer des labels obligatoires, et empêcher le montage de volumes hostPath.

Pour la sécurité RBAC spécifiquement, Kyverno peut valider que les RoleBindings créés par les équipes de développement ne référencent que des ClusterRoles approuvés, et que les ServiceAccounts créés n'ont pas `automountServiceAccountToken: true`. Cette approche préventive est complémentaire au RBAC natif et aux Network Policies — elle forme le troisième pilier de la sécurité Kubernetes.

À retenir : La sécurité Kubernetes repose sur trois piliers complémentaires : RBAC pour le contrôle d'accès, Network Policies pour la segmentation réseau, et Admission Controllers pour la prévention des déploiements non conformes. Aucun de ces mécanismes seul ne suffit — c'est leur combinaison qui crée une posture de défense en profondeur efficace contre les mouvements latéraux et les escalades de privilèges.

Peut-on sécuriser les Service Accounts Kubernetes ?

Les Service Accounts sont le principal vecteur d'escalade de privilèges dans Kubernetes. Depuis la version 1.24, Kubernetes ne crée plus automatiquement de tokens persistants pour les Service Accounts, mais les tokens éphémères projetés via TokenRequest API sont toujours montés par défaut. Désactivez `automountServiceAccountToken` au niveau du ServiceAccount et du Pod spec, puis montez explicitement les tokens uniquement dans les pods qui en ont besoin. Utilisez des **Bound Service Account Tokens** avec audience et expiration restreintes pour limiter la portée et la durée de validité des tokens.

Pour les workloads qui accèdent à des services cloud externes (AWS S3, Azure Blob, GCP GCS), utilisez les mécanismes de fédération d'identité natifs : **IRSA** (IAM Roles for Service Accounts) sur EKS, **Workload Identity** sur GKE, et **Azure AD Workload Identity** sur AKS. Ces mécanismes éliminent le besoin de stocker des credentials cloud dans des Secrets Kubernetes, réduisant considérablement le risque en cas de compromission d'un pod ou d'un namespace.

La **runtime security** complète les mécanismes préventifs (RBAC, Network Policies, PSS) avec une détection en temps réel des comportements malveillants dans les pods. **Falco**, projet CNCF incubating, monitore les appels système des conteneurs via eBPF et alerte sur les activités suspectes : exécution de shell dans un conteneur, accès au metadata service, modification de fichiers système, téléchargement de binaires suspects, et communications réseau non attendues. Combiné avec des **Seccomp profiles** stricts qui restreignent les appels système autorisés et des **AppArmor profiles** qui limitent l'accès aux fichiers et capabilities, la runtime security crée une dernière ligne de défense qui détecte et peut bloquer les attaques qui auraient réussi à contourner tous les mécanismes préventifs. Déployez Falco comme DaemonSet avec des règles custom adaptées à vos applications pour minimiser les faux positifs et maximiser la valeur de détection de cette couche de sécurité runtime essentielle en environnement Kubernetes de production.

Combien de ServiceAccounts dans vos clusters Kubernetes disposent encore de permissions cluster-admin ou de tokens montés automatiquement sans justification opérationnelle ?

Comment gérer les Pod Security Standards ?

Les **Pod Security Standards** (PSS) remplacent les anciennes PodSecurityPolicies (PSP) dépréciées depuis Kubernetes 1.25. PSS définit trois niveaux de sécurité : **Privileged** (aucune restriction, réservé aux workloads système), **Baseline** (prévient les escalades de privilèges connues, bloque hostNetwork, hostPID, hostIPC, les conteneurs privilégiés et les capabilities

dangereuses), et **Restricted** (le niveau le plus strict, impose non-root, drop ALL capabilities, seccomp, read-only rootfs). Le Pod Security Admission (PSA) controller enforce ces standards au niveau du namespace via des labels.

En production, appliquez le niveau **Restricted** sur les namespaces applicatifs et Baseline sur les namespaces d'infrastructure (monitoring, logging, mesh). Le mode `enforce` bloque les pods non conformes, `audit` les loggue sans bloquer, et `warn` affiche un avertissement au kubectl apply. Commencez par audit et warn pour évaluer l'impact, puis basculez en enforce après avoir ajusté les manifestes. Les Pod Security Standards complètent le RBAC et les Network Policies en ajoutant un contrôle sur les capacités runtime des conteneurs, fermant le triangle sécurité Kubernetes qui empêche les escalades de privilèges depuis l'intérieur des pods en limitant les appels système autorisés et les montages de volumes sensibles.

Pour les cas où PSS est insuffisant, les **RuntimeClasses** permettent de sélectionner un runtime alternatif sécurisé comme gVisor (sandbox applicative) ou Kata Containers (micro-VMs) pour les workloads nécessitant un isolement renforcé au-delà de ce que les namespaces et cgroups Linux standards fournissent.

L'écosystème de sécurité Kubernetes évolue rapidement avec les solutions eBPF-native comme Cilium Tetragon pour la runtime security et les améliorations continues des Pod Security Standards. Maintenez une veille active sur les releases Kubernetes et les CVE affectant kubelet, API server, etcd et les CNI, car les vulnérabilités de ces composants permettent des escalades de privilèges significatives qui contournent les contrôles RBAC et Network Policies soigneusement configurés. L'adoption du format SLSA pour la supply chain des images conteneur renforce également la sécurité globale du cluster dans votre environnement de production.

Sources et références : CISA · Cloud Security Alliance

Conclusion : checklist sécurité Kubernetes

Sécuriser Kubernetes requiert une approche systématique en quatre phases. Phase 1 — RBAC : auditez et restreignez tous les ClusterRoleBindings, éliminez les wildcards, désactivez l'automount des tokens SA. Phase 2 — Network Policies : déployez des default deny dans chaque namespace, puis whitelistez les flux légitimes avec Calico ou Cilium. Phase 3 — Admission Control : déployez Kyverno ou OPA Gatekeeper avec des policies de sécurité de base (no root, resource limits, image registry whitelist). Phase 4 — Monitoring : activez les Audit Logs, déployez Falco pour la détection runtime, et intégrez le tout dans votre SIEM. Cette progression garantit une sécurité Kubernetes robuste et maintenable sur le long terme.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr