AYI NEDJIMI Consultants

Container Registry : Guide Sécurité Images Docker 2026

Catégorie : Cloud Security Lecture : 8 min Publié le : 08/03/2026 Auteur : Ayi NEDJIMI

Sécurisez vos images Docker et votre container registry : scanning de vulnérabilités, signature d'images, policies d'admission et supply chain.

Résumé exécutif

La sécurité des images Docker est un maillon critique de la supply chain conteneur. Ce guide couvre le scanning de vulnérabilités, la signature d'images avec Cosign, les policies d'admission Kubernetes et les bonnes pratiques de build sécurisé.

Chaque image Docker que vous déployez en production est une boîte noire potentielle contenant des vulnérabilités connues, des malwares, des credentials en dur et des configurations système dangereuses. La majorité des organisations scannent leurs images une fois au build mais ne surveillent pas les nouvelles vulnérabilités découvertes après le déploiement, ne vérifient pas l'intégrité des images au moment du pull, et ne contrôlent pas quelles images sont autorisées à s'exécuter sur leurs clusters. Cette chaîne de confiance brisée fait des container registries un vecteur d'attaque privilégié pour les adversaires qui ciblent la supply chain logicielle. Après avoir accompagné de nombreuses équipes dans la sécurisation de leur pipeline de conteneurs, depuis le Dockerfile jusqu'au runtime Kubernetes en passant par les registries intermédiaires, je partage les pratiques éprouvées et les outils qui transforment votre supply chain conteneur en un pipeline vérifié et auditable de bout en bout.

Pourquoi les images Docker sont un vecteur d'attaque majeur ?

Les attaques via les images Docker exploitent plusieurs vecteurs. Le **typosquatting** sur les registries publics (images avec des noms proches de projets populaires mais contenant du malware), les **images de base obsolètes** (un FROM ubuntu:20.04 tire des centaines de paquets dont certains avec des CVE critiques), les **credentials en dur** dans les layers (même supprimés dans une layer ultérieure, ils restent dans l'historique), les **binaires malveillants** ajoutés dans des images populaires compromises (supply chain attack type SolarWinds appliqué au conteneur), et les **configurations non sécurisées** (processus root, capabilities excessives, volumes sensibles).

Notre article sur les techniques d'évasion de conteneur Docker illustre comment un attaquant exploite une image vulnérable pour s'évader du conteneur et compromettre le node hôte. Les conséquences en cascade d'une image compromise sont détaillées dans attaques CI/CD GitOps avec des scénarios de compromission de pipelines CI/CD. La documentation officielle de Kubernetes Security fournit les bonnes pratiques de base pour la sécurité des images conteneur.

Vecteur	Exemple	Fréquence	Mitigation
Image de base vulnérable	CVE dans libssl	Très fréquent	Distroless/Alpine + scan continu
Credentials dans layers	API key dans RUN	Fréquent	Multi-stage builds + secrets
Typosquatting	ngimx au lieu de nginx	Occasionnel	Registry privé + whitelist
Supply chain compromise	Image populaire backdoorée	Rare mais dévastateur	Signature + vérification
Config non sécurisée	USER root, CAP_SYS_ADMIN	Très fréquent	Linting Dockerfile + OPA

Mon avis : Les images distroless de Google ou les images scratch custom sont la meilleure défense : moins de paquets signifie moins de CVE. Une image Alpine avec votre binaire Go compilé statiquement contient quelques Mo contre plusieurs centaines pour une image Ubuntu, avec une surface d'attaque réduite de 95%. L'effort initial de migration vers des images minimales est largement compensé par la réduction de maintenance sécurité.

Comment scanner les images de manière continue ?

Le scanning d'images doit opérer à trois moments : au **build** (dans le pipeline CI/CD), au **push** (dans le registry), et en **continu** (rescan périodique des images déployées). Les outils majeurs incluent : **Trivy** (open source, rapide, couvre OS et dépendances applicatives), **Grype** (par Anchore, focus sur la précision des résultats), **Snyk Container** (intégration CI/CD native avec des recommandations de fix), et les scanners natifs des cloud providers (ECR scanning, ACR Defender for Containers, GCR Artifact Analysis).

Configurez **Trivy** comme gate bloquante dans votre pipeline CI/CD avec un seuil de sévérité : bloquer sur les CRITICAL et HIGH, alerter sur les MEDIUM. Pour le scanning continu, utilisez les fonctionnalités natives des registries managés : **ECR Enhanced Scanning** (basé sur Inspector) rescanne automatiquement les images lors de la publication de nouvelles CVE. Intégrez les résultats dans votre SIEM pour corréler les vulnérabilités d'images avec les workloads déployés en production. Les bonnes pratiques d'AWS Security complètent cette approche avec les services AWS dédiés.

Quelles sont les bonnes pratiques de build sécurisé ?

Le *Dockerfile sécurisé* suit des principes stricts. Utilisez des **images de base spécifiques** avec un tag de version (jamais :latest). Employez des **multi-stage builds** pour séparer l'environnement de compilation de l'image finale. Exécutez les processus en tant qu'**utilisateur non-root** avec l'instruction USER. Minimisez les layers pour réduire l'historique exploitable. Ne copiez que les fichiers nécessaires avec des .dockerignore stricts. Utilisez **COPY au lieu de ADD** (ADD décompresse automatiquement et peut télécharger des URLs, vecteurs d'attaque potentiels).

Pour la gestion des secrets au build time, utilisez les **Docker BuildKit secrets** (`--mount=type=secret`) qui montent le secret temporairement sans le persister dans une layer. Jamais de `ENV API_KEY=xxx` ou de `COPY .env .` dans un Dockerfile. Les techniques de détection de secrets dans les images sont détaillées dans secrets sprawl et collecte. L'audit des Dockerfiles via Terraform et l'IaC est couvert dans audit Terraform compliance.

Un audit de container registry pour un opérateur télécom a révélé que 73% des images en production contenaient des CVE critiques, dont 12% avec des exploits publics disponibles. Plus alarmant, nous avons trouvé des API keys AWS et des mots de passe de bases de données dans l'historique des layers de 8 images, accessibles via un simple `docker history`. La mise en place d'un pipeline de scanning Trivy avec gate bloquante et d'un rescan hebdomadaire a réduit les CVE critiques à 0 en production en trois mois.

Les **policies de rétention** dans le container registry sont essentielles pour maintenir un registry propre et sécurisé. Sans politique de rétention, les registries accumulent des centaines de versions d'images dont la majorité ne sont plus déployées, contiennent des vulnérabilités connues et consomment un stockage coûteux. Configurez des règles de rétention qui conservent les N dernières versions taggées de chaque image, suppriment les images non taggées après 7 jours, et maintiennent indéfiniment les images actuellement déployées en production identifiées par des tags spécifiques comme production ou stable. Harbor offre les politiques de rétention les plus flexibles avec des critères combinables par tag pattern, âge et nombre de versions. ECR propose des Lifecycle Policies basiques mais suffisantes pour la majorité des cas. Complétez avec un scan de conformité qui vérifie quotidiennement que les images déployées en production sont toujours dans le catalogue d'images approuvées et qu'elles ne dépassent pas un âge maximal défini par votre politique de sécurité, typiquement 90 jours pour les images de base et 30 jours pour les images applicatives.

Comment implémenter la signature et la vérification d'images ?

La **signature d'images** garantit que seules les images approuvées et non modifiées s'exécutent en production. **Cosign** (projet Sigstore) est l'outil de référence : il signe les images avec des clés cryptographiques ou des identités OIDC (keyless signing). Le workflow est : build l'image, scan pour les vulnérabilités, signe avec Cosign si le scan passe, push vers le registry. Au deploy time, un **admission controller** Kubernetes (Kyverno ou OPA Gatekeeper) vérifie la signature avant d'autoriser le déploiement du pod.

La vérification de signature dans attaques RBAC Kubernetes montre comment les Network Policies Kubernetes complètent la vérification d'images en contrôlant le trafic réseau des conteneurs autorisés. Les recommandations de sécurité réseau de audit Terraform compliance doivent être appliquées au conteneur réseau du registry lui-même pour prévenir les accès non autorisés au registry interne.

À retenir : La sécurité des images Docker repose sur quatre piliers : des images de base minimales et à jour, un scanning continu à trois niveaux (build, push, runtime), une signature cryptographique avec vérification à l'admission, et des policies de déploiement qui interdisent les images non scannées ou non signées. Chaque pilier manquant est une brèche dans votre supply chain conteneur.

Faut-il un registry privé ou les registries cloud suffisent ?

Les registries managés des cloud providers (ECR, ACR, GCR/Artifact Registry) offrent un excellent rapport fonctionnalités/effort de maintenance : scanning intégré, réplication cross-région, chiffrement, IAM natif. Cependant, pour les environnements multi-cloud ou les exigences de souveraineté, un registry privé auto-hébergé comme **Harbor** (CNCF graduated project) offre des fonctionnalités supplémentaires : scanning intégré Trivy, signature Notary/Cosign, réplication inter-registries, quotas et policies de rétention avancées, et contrôle total sur la localisation des données. Harbor supporte également le stockage d'artefacts OCI non-Docker comme les charts Helm et les modules WASM.

L'adoption des **SBOM (Software Bill of Materials)** devient une exigence réglementaire et une bonne pratique de sécurité supply chain. Les SBOM documentent chaque composant logiciel présent dans une image Docker avec sa version, sa licence et sa provenance. Générez les SBOM au build time avec **Syft** ou **docker sbom** et stockez-les à côté de l'image dans le registry. Les SBOM facilitent l'identification rapide des images affectées lors de la publication d'une nouvelle CVE sans avoir à rescanner chaque image individuellement, et permettent de répondre aux exigences de transparence de la supply chain imposées par des réglementations comme le Cyber Resilience Act européen et le Executive Order américain sur la cybersécurité qui imposent aux éditeurs de logiciels de fournir des SBOM pour leurs produits et composants distribués.

Savez-vous exactement quelles images tournent actuellement en production dans vos clusters, et quand elles ont été scannées pour la dernière fois contre les CVE publiées cette semaine ?

Comment gérer les vulnérabilités dans les images de base ?

La gestion des vulnérabilités dans les images de base est un processus continu qui nécessite une stratégie structurée. Établissez un **catalogue d'images de base approuvées** maintenu par l'équipe sécurité. Ce catalogue contient les images officielles des langages et runtimes utilisés dans votre organisation (Node.js Alpine, Python slim, Go distroless, Java Eclipse Temurin) avec des versions spécifiques épinglées et régulièrement mises à jour. Chaque image du catalogue est scannée quotidiennement, et une mise à jour est déclenchée automatiquement lorsqu'une CVE critique est publiée. Les équipes de développement ne peuvent utiliser que les images du catalogue comme base pour leurs Dockerfiles, garantissant un niveau de sécurité minimal uniforme.

Le processus de **patching d'urgence** pour une CVE critique (CVSS 9+) dans une image de base doit être automatisé autant que possible. Un pipeline dédié reconstruit toutes les images dérivées de l'image de base affectée, les rescanne pour vérifier la correction, les re-signe avec Cosign, et ouvre automatiquement des pull requests dans les repositories applicatifs pour mettre à jour la référence de l'image. Les équipes disposent de 48 heures pour merger et déployer la mise à jour en production. Ce processus réduit le délai de remédiation d'une CVE critique de plusieurs semaines dans les processus manuels à quelques jours, voire quelques heures dans les organisations les plus matures en DevSecOps avec des pipelines de déploiement continu bien rodés.

Complétez cette approche avec des **SBOM** (Software Bill of Materials) générés automatiquement au build time avec des outils comme Syft. Le SBOM documente chaque composant logiciel présent dans l'image avec sa version exacte, facilitant l'identification rapide des images affectées lors de la publication d'une nouvelle vulnérabilité et la conformité avec les exigences réglementaires croissantes sur la transparence de la supply chain logicielle.

Le coût de la sécurité des conteneurs est dérisoire comparé au coût d'une compromission supply chain. Un pipeline complet avec Trivy, Cosign et Kyverno utilise exclusivement des outils open source sans licence. L'investissement principal est le temps d'ingénierie pour la mise en place initiale et la maintenance continue des policies et des mises à jour d'images de base qui garantissent une supply chain sécurisée et auditable.

Sources et références : CISA · Cloud Security Alliance

Conclusion : pipeline de sécurité conteneur de bout en bout

Construisez votre pipeline de sécurité conteneur en quatre phases. Phase 1 : standardisez les images de base (distroless ou Alpine minimal) et intégrez Trivy dans le CI/CD comme gate bloquante. Phase 2 : implémentez la signature d'images avec Cosign et déployez un admission controller Kyverno qui vérifie les signatures. Phase 3 : activez le scanning continu dans votre registry et configurez des alertes sur les nouvelles CVE affectant les images déployées. Phase 4 : implémentez des policies de rétention automatique qui suppriment les images non utilisées et non conformes du registry. Ce pipeline garantit que seules des images scannées, signées et à jour s'exécutent en production.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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