Attestkeep

Manuel produit

La sécurité des images de container pour Kubernetes. Attestkeep scanne les images, vérifie leurs signatures, décide à l'admission si un pod peut démarrer, refuse ce que vos équipes ont placé sur la deny list et conserve une trace signée de chaque décision. Ce manuel décrit ce qui est installé, comment l'exploiter et ce que cela coûte.

Attestkeep 1.4.0 · Chart Helm `oci://ghcr.io/attestkeep/charts/attestkeep` version 1.4.0 · manuel daté du 29 octobre 2026.

Les captures d'écran ont été prises le 29 octobre 2026 sur une installation dont la console indique la version 1.4.0, en Enterprise Edition sur Kubernetes v1.35.1, avec Falco connecté. La console était en anglais, et ce manuel cite ses libellés tels qu'elle les affiche. L'annexe en dit plus. Les commandes, réglages et limites viennent de la documentation du produit sur docs.attestkeep.com, du chart et de la console. Rien dans ce manuel n'est fictif.

attestkeep.com · docs.attestkeep.com · support@attestkeep.com

Sommaire

1. Ce qu'est Attestkeep

2. Ce qui tourne dans votre cluster

3. Comment se prend une décision d'admission

4. Installation, mise à niveau et premier démarrage

5. La console

6. Les policies d'admission

7. La deny list

8. Les événements runtime de Falco

9. Prouver que le contrôle s'applique

10. Paquets de preuves et modèle de confiance

11. Exploitation, export SIEM et sauvegarde

12. Licences

13. Éditions et prix

14. Sécurité, support et votre interlocuteur

Annexe. Sources vérifiées pour ce manuel

1. Ce qu'est Attestkeep

Attestkeep est un operator Kubernetes. Il scanne les images de vos workloads, vérifie leurs signatures et attestations, et répond à l'API server au moment de l'admission par une autorisation ou un refus, selon une policy que vous écrivez. Chaque décision est enregistrée. Ces enregistrements deviennent un rapport par période qu'un auditeur peut vérifier avec ses propres outils.

Tout tourne dans votre cluster. Une seule release Helm installe l'operator, son API, la console et la base de données dans un namespace qui vous appartient. Les noms d'images, les résultats de scan et la topologie du cluster ne sortent pas du cluster. Le contrôle de licence envoie trois valeurs une fois par jour.

Ce qu'il fait

- **Scan des vulnérabilités.** Trivy est intégré à l'image. Les vulnérabilités sont listées par gravité, avec la version corrigée quand elle existe.
- **Vérification des signatures.** Vérification cosign par clé ou en keyless, et attestations in-toto avec une ancienneté maximale. À partir de la 1.4.0, une policy peut lister les clés publiques qu'elle accepte, et seule compte alors une signature vérifiée par l'une d'elles.
- **Contrôle à l'admission.** Seuils par gravité, règles sur les tags et les registries, épinglage par digest, exigence de signature et d'attestation, quinze contrôles de durcissement des workloads, un blocage fondé sur le catalogue Known Exploited Vulnerabilities de la CISA, un blocage des systèmes de base hors support, un contrôle de provenance SLSA qui vérifie qui a signé, les déclarations VEX du producteur, et des règles plus strictes pour les pods privilégiés ou exposés.
- **Deny list.** Quand quelqu'un décide qu'une image, une CVE, une version de paquet ou un exécutable ne doit pas tourner, une seule entrée le dit. Elle est vérifiée avant le break-glass et avant toute règle de la policy, dans chaque namespace dont la policy applique la liste. Un second outil lit les exécutables à l'intérieur de chaque image, si bien qu'un runtime compilé depuis les sources, sans trace de paquet, est trouvé lui aussi. Un workload en cours n'est jamais arrêté automatiquement. Un administrateur peut le mettre à l'arrêt à la main, une fois, par un take down, puis le rétablir. Le chapitre 7 en parle.
- **Triage avec échéance.** Vous pouvez accepter un risque avec une justification et une date de fin. Le break-glass laisse passer une image nommée pendant un incident. Les deux sont enregistrés.
- **Paquets de preuves.** Un document signé par période d'audit, rapporté au catalogue de contrôles de votre choix. Il dit ce qu'il ne peut pas prouver autant que ce qu'il prouve.
- **Journal d'audit.** Modifications de policies, exceptions à l'admission, décisions de triage, exports de preuves et connexions dans un même journal.
- **Notifications.** Slack, Teams, e-mail et un webhook générique. Une notification nomme le cluster, jamais sa topologie.
- **Gestionnaires de tickets.** Une nouvelle vulnérabilité critique ouvre un ticket dans votre GitHub, GitLab ou Jira, un par digest d'image et par CVE. Le ticket est fermé avec un commentaire qui en donne la raison : corrigé, traité en triage ou image retirée. Seul *corrigé* compte comme remédiation.
- **Événements runtime de Falco.** Votre propre Falco envoie ses événements à l'operator en TLS mutuel, directement ou via Falcosidekick. Chaque événement est rattaché à l'image et au pod d'où il vient. Les policies runtime que vous écrivez dans la console décident de la suite : garder une trace, envoyer une notification, ouvrir un ticket ou arrêter le pod. Un événement runtime ne change jamais une décision d'admission. Le chapitre 8 en parle.
- **Export SIEM.** Les admissions refusées, les correspondances et take downs de la deny list, les événements runtime, les changements de policy et de triage, les téléchargements de preuves, les connexions échouées et les nouvelles vulnérabilités critiques partent vers votre SIEM au moment où ils se produisent : Splunk, Microsoft Sentinel, Elasticsearch, syslog ou un collecteur HTTP générique. Le chapitre 11 en parle.
- **Authentification unique et rôles.** Google, GitHub, GitLab, Okta, Microsoft Entra ID, Keycloak ou tout fournisseur OIDC, dans les éditions payantes. Les rôles décident qui peut modifier une policy.

Ce qu'il ne fait pas

Attestkeep n'est pas un capteur runtime. Il n'observe ni les appels système, ni les flux réseau, ni les processus d'un container en cours. Falco fait cette détection, et Attestkeep reçoit seulement ce que Falco rapporte. Attestkeep ne fournit pas Falco, ne l'installe pas, ne le configure pas et n'y écrit pas.

Il n'arrête pas de lui-même les workloads en cours. Une entrée de deny list et une policy en mode Enforce refusent les nouveaux pods. Arrêter ce qui tourne déjà reste toujours la décision d'une personne, ou d'une policy runtime que vous avez écrite et que le chart autorise.

Ce n'est pas un outil de posture cloud, et il ne lit ni votre compte cloud ni vos politiques IAM. Il ne scanne ni Terraform, ni les manifestes, ni la configuration du cluster par rapport à un benchmark. Il ne vérifie pas si une fonction vulnérable est réellement appelée, ne lit pas vos dépôts et n'ouvre pas de pull requests de dépendances. docs.attestkeep.com indique quoi faire tourner à côté d'Attestkeep pour chacun de ces besoins.

Pourquoi ne pas construire ce contrôle avec des outils libres ?

C'est possible. Il existe de bons moteurs open source de policies d'admission et des scanners dans le cluster. Si vous voulez seulement bloquer `:latest` et refuser les CVE critiques, vous pouvez le faire sans Attestkeep. Ce qui manque généralement à un tel montage, c'est la trace dont se servent les audits et les analyses d'incident :

- **Un registre des décisions dont l'intégrité est vérifiable.** Chaque décision d'admission est hachée à l'écriture et scellée sous des checkpoints signés et chaînés. Si un verdict est modifié dans la base, le document de preuve suivant désigne l'enregistrement.
- **Des décisions liées à la policy qui les a prises.** Chaque enregistrement porte le hash de la policy telle qu'elle était à ce moment.
- **Des trous signalés.** Les périodes où le webhook ne tournait pas ou n'a pas pu être vérifié sont enregistrées et imprimées dans la preuve. Le rapprochement liste ce qui est entré pendant ces périodes.
- **Un seul produit.** Résultats de scan, vérification des signatures, admission, deny list, exceptions et paquet de preuves sont maintenus ensemble, et votre équipe n'a pas à entretenir la colle entre eux.

2. Ce qui tourne dans votre cluster

Un webhook d'admission a son mot à dire sur chaque pod. Voici l'inventaire de ce que cela implique.

Objet	Par défaut	Rôle
Deployment `attestkeep`	2 réplicas, anti-affinité préférée entre nodes, PodDisruptionBudget minAvailable 1	L'operator : webhook d'admission (8443), console et API (8080), métriques (9090), et l'écouteur des intégrations Falco (8444) quand `integrations.falco.enabled` est activé. Le travail du contrôleur est sérialisé par un advisory lock PostgreSQL, si bien que les réplicas se multiplient sans bail d'élection de leader. À partir de la 1.4.0, un réplica qui s'arrête quitte le Service avant de cesser de répondre : son hook preStop attend `webhook.preStopSleepSeconds` (5 par défaut), puis il termine les revues en cours.
Deployment `attestkeep-scanner`	1 réplica	Exécute les scans Trivy, la vérification cosign et, quand la deny list en a besoin, le contrôle des exécutables `attestkeep-binscan`, toujours hors du chemin d'admission. Il ne reçoit jamais de trafic d'admission et, à partir de la 1.4.0, ne contacte jamais le serveur de licences.
StatefulSet `attestkeep-postgresql`	1 réplica, 20 GiB, postgres:16.15-alpine	La base de données fournie. Si vous pointez `externalDsn` vers votre propre PostgreSQL, cet objet n'est pas créé.

Tous les containers du produit tournent sans root (uid 65532), avec un système de fichiers racine en lecture seule, toutes les capabilities retirées et seccomp RuntimeDefault. Les requests par défaut sont de 100m de CPU et 256Mi par workload du produit.

Au niveau du cluster, on trouve quatre CRD du groupe `attestkeep.com` (ImageSecurityPolicy, ImageSecurityReport, AttestationRenewal, ImageSecurityUsage) et une ValidatingWebhookConfiguration. Le webhook couvre CREATE et UPDATE des pods, y compris la sous-ressource `pods/ephemeralcontainers` dans laquelle écrit `kubectl debug`, avec un timeout de 15 secondes et `failurePolicy: Ignore` par défaut. Il n'y a pas de webhook de mutation. Le produit ne réécrit jamais vos workloads.

Les permissions à connaître

- Le ClusterRole lit les nodes (get, list). C'est ainsi que l'operator compte les nodes Ready face au plafond de nodes de la licence. Il garde seulement le nombre, jamais un nom de node.
- Les Secrets sont lisibles dans tout le cluster avec `get` seulement, pas `list`. L'operator peut lire un pull secret dont il a trouvé le nom dans la spec d'un pod. Il ne peut pas lister le contenu d'un namespace.
- `pods/eviction` n'est accordé qu'avec `rbac.allowEnforce: true` ou `rbac.denyList.takedown: true` (tous deux false par défaut). Une policy runtime qui arrête des pods et `runtimeReconciliation: enforce` exigent `rbac.allowEnforce: true`, quoi que le ClusterRole accorde par ailleurs. Sans ce réglage, ils sont refusés et enregistrés. La permission de take down sert uniquement la deny list.
- La sous-ressource scale des Deployments, StatefulSets et ReplicaSets (get, update) n'est accordée qu'avec `rbac.denyList.takedown: true`. Elle permet à un administrateur de ramener un workload à zéro réplica puis de le rétablir. Elle change combien de réplicas tournent, jamais ce qu'ils exécutent.
- Les Deployments, ReplicaSets, StatefulSets, DaemonSets, Jobs et CronJobs sont lisibles (get, list) dans toute installation à partir de la 1.4.0. L'operator lit la chaîne de propriétaires au-dessus d'un pod, si bien qu'un refus est compté contre le Deployment et non son ReplicaSet. C'est une lecture seule.
- L'operator crée des événements Kubernetes : un Warning sur le workload qu'un refus de la deny list a bloqué, et sur une policy dont une clé de signature n'est pas une clé publique PEM. Il écrit aussi le statut de ses propres policies, où se trouve la condition `SignatureKeysValid`.

- Un Role de namespace autorise l'écriture de trois Secrets nommés dans le namespace de l'operator : le certificat du webhook, `attestkeep-evidence-key`, et `<release>-falco-mtls`, qui n'existe que si l'intégration Falco est active. À partir de la 1.4.0, `attestkeep-evidence-key` est aussi l'identité de licence de l'installation (chapitre 12).
- Un second Role, `<release>-rule-validation`, permet à l'operator de créer, lire et supprimer des pods et des ConfigMaps dans son propre namespace et de lire les logs des pods. L'éditeur de règles Falco s'en sert pour lancer un pod de validation ponctuel. Il ne peut lancer de pods dans aucun autre namespace.

Toutes les connexions sortantes

Ce tableau sert à votre demande d'ouverture de flux. Il n'y a ni SDK d'analytics, ni remontée de plantages, ni télémétrie.

Destination	Obligatoire ?	Réglage	Si elle est bloquée
lic.attestkeep.com:443	Oui, sauf activation hors ligne	`license.serverUrl` ; `license.caBundle` pour une CA privée ou un proxy qui inspecte le TLS sur le trajet	L'activation en ligne est impossible. Une installation activée continue avec son certificat et repasse en Community quand il expire. À partir de la 1.4.0, un seul réplica de l'operator fait le contrôle quotidien.
ghcr.io (base de vulnérabilités Trivy):443	Pour scanner	`trivy.dbRepository`, pour un miroir	La base existante continue de servir et vieillit. `attestkeep_vulnerability_db_age_seconds` indique de combien.
Vos registries d'images:443	Pour scanner	Les pull secrets des specs de pods	Les scans de ces images échouent. L'admission suit le réglage de la policy pour les images non scannées.
ghcr.io/attestkeep/attestkeep-k8s:443	À l'installation	`image.repository`, pour un miroir	Les pods ne démarrent pas.
Sigstore TUF et Rekor public:443	Non	`scan.cosignOffline: true`	Avec une signature par clé et le mode hors ligne, la vérification reste locale.
Flux des versions	Non, désactivé par défaut	`updates.endpoint`	Rien. Une installation sans adresse configurée ne regarde jamais.
Vos destinations SIEM	Non, configurées par vous	Settings > SIEM export	Les événements attendent et sont réessayés. Un événement toujours pas livré au bout de 24 heures est abandonné et compté sur la page.
Votre collecteur OTLP, vos URL Slack, Teams ou webhook, votre serveur SMTP, votre fournisseur SSO	Non, configurés par vous	Selon la fonction	Cette fonction échoue et le journalise. Un échec de notification ne bloque jamais l'admission.

Si notre serveur de licences était compromis

Il ne pourrait donner aucun ordre à votre cluster. Le protocole de licence connaît deux types de réponse : un certificat et un état de renouvellement. Aucun chemin de code ne laisse une réponse du serveur choisir une policy, un namespace, une règle d'image, une URL ou un programme à exécuter. Les certificats sont signés en Ed25519, vérifiés avec une clé publique compilée dans le binaire et liés à l'empreinte de votre cluster.

Un serveur hostile pourrait répondre *revoked*. L'installation passerait alors à une admission qui se contente d'avertir, ce qui se voit dans la console et dans chaque réponse d'admission. Il ne pourrait ni placer un workload, ni modifier une policy. Si le serveur ne répond pas du tout, rien ne se passe : les erreurs réseau comptent comme une panne, et l'installation tourne avec son certificat jusqu'à son expiration.

3. Comment se prend une décision d'admission

Le webhook répond depuis le cache. Le scan ne tourne jamais dans le chemin d'admission, un registry lent ne ralentit donc pas un déploiement.

1. L'API server interroge sur un pod : à sa création ou sa mise à jour, et quand `kubectl debug` ajoute un container à un pod en cours.
2. Le tag de l'image est résolu en digest. Le résultat reste une minute en cache, si bien qu'une rafale de déploiements identiques fait un seul appel au registry.
3. Le résultat du scan et le verdict de signature du digest sont lus en cache.
4. Les règles qui n'exigent aucun appel au registry passent d'abord : la liste des registries autorisés, `:latest` et l'exigence de digest.
5. La deny list est vérifiée, avant le break-glass et avant toute autre règle de la policy. Une image listée est refusée même si son scan est propre, et même quand un break-glass est ouvert (chapitre 7).
6. La policy est évaluée. La réponse est une autorisation ou un refus, avec une raison.
7. La décision est enregistrée, autorisations comprises. Une preuve qui ne liste que des refus ne peut pas montrer qu'un contrôle fonctionnait.

Aucune de ces étapes n'attend un scan. Si un digest n'a jamais été vu, l'étape trois n'a pas de réponse, et la policy décide de ce qui se passe.

Les événements Falco ne font pas partie de ce chemin. Un événement runtime ne change jamais l'admission d'un pod.

Les dry runs. À partir de la 1.4.0, `kubectl apply --dry-run=server` est décidé et reçoit la réponse qu'aurait une vraie requête, sans rien laisser derrière lui : ni ligne de registre, ni correspondance de deny list, ni événement Kubernetes, ni notification, ni métrique, rien de décompté sur le quota d'images et aucun scan mis en file.

Quand la base ne répond pas. Une revue d'admission accorde deux secondes au total à son travail en base. Ensuite, les règles statiques tranchent aussitôt : `:latest`, la liste des registries autorisés et l'exigence de digest continuent de refuser. L'enregistrement et les verdicts qui dépendent du cache se dégradent, et la panne est journalisée une fois. Avant la 1.4.0, un réplica quittait le Service tant que la base était injoignable, et des réponses d'admission se perdaient. Le chapitre 11 donne le détail.

Containers de débogage. À partir de la 1.3.7, un container ajouté avec `kubectl debug` passe aussi par le webhook. Seul le container ajouté est jugé. Son image est scannée et soumise à la policy, et une policy en mode Enforce le refuse comme toute autre image. Les containers déjà présents dans le pod ne sont pas réexaminés. Une image de débogage n'est comptée ni dans le quota mensuel d'images ni dans le plafond de nodes, et la raison de sa ligne de registre se termine par `; subresource ephemeralcontainers`. Avant la 1.3.7, un tel container n'était jamais envoyé au webhook, quelle que soit son image.

Les deux réglages qui fixent votre posture

Réglage	Par défaut	Ce qu'il signifie
`webhook.failurePolicy` (chart)	`Ignore`	Ce que fait l'API server quand l'operator ne répond pas. `Ignore` admet les pods sans contrôle pendant une panne. `Fail` refuse jusqu'à ce que l'operator réponde.
`admissionTiming` (policy)	`AllowAndScan`	Ce qui arrive à un digest vu pour la première fois. `AllowAndScan` l'admet et met un scan en file. `DenyUntilScanned` le refuse tant qu'aucun scan n'existe. C'est le partenaire habituel de `Fail`.

Aucun des deux choix ne convient à tout le monde. `Ignore` garde le cluster déployable pendant une panne de l'operator et laisse un trou dans la preuve. Le paquet de preuves signale ce trou. `Fail` donne un contrôle sans trou, mais une

panne de l'operator bloque alors les déploiements. Les deux réglages existent dans toutes les éditions, Community comprise.

Ce qui est entré pendant un trou tourne encore ensuite. Le rapprochement runtime le trouve. Selon un calendrier (toutes les heures par défaut), il compare l'identifiant d'image de chaque container en cours avec le registre d'admission. Un digest sans admission derrière lui est signalé, notifié ou, avec `runtimeReconciliation: enforce`, arrêté pour repasser par le webhook.

> **Le premier scan d'un nouveau digest se fait en arrière-plan.** Sur un cluster neuf, il peut prendre quelques minutes, parce que le scanner doit tirer l'image et que la base de vulnérabilités n'est pas encore en cache. Jusqu'à la fin, l'image est à l'état *Unscanned* sur la page Images. Les admissions suivantes de ce digest utilisent le rapport terminé. *Rescan* lance un scan à la demande.

4. Installation, mise à niveau et premier démarrage

Avant de commencer

- Kubernetes 1.27 ou plus récent.
- Le droit de créer une ValidatingWebhookConfiguration, donc cluster-admin à la première installation.
- Environ 20 GiB pour le PostgreSQL fourni, ou le DSN d'un PostgreSQL que vous exploitez déjà. L'export SIEM exige PostgreSQL 13 ou plus récent. Celui fourni est en version 16.
- Un compte Attestkeep sur attestkeep.com et une clé de licence : la clé Community gratuite ou celle de l'offre achetée.

Vérifier la release avant d'installer

Chaque release est signée sur son digest. La vérification par clé n'a besoin que de votre registry et de la clé publique publiée sur docs.attestkeep.com/cosign.pub.

```
curl -sO https://docs.attestkeep.com/cosign.pub
cosign verify --key cosign.pub ghcr.io/attestkeep/attestkeep-k8s:1.4.0
```

Installer

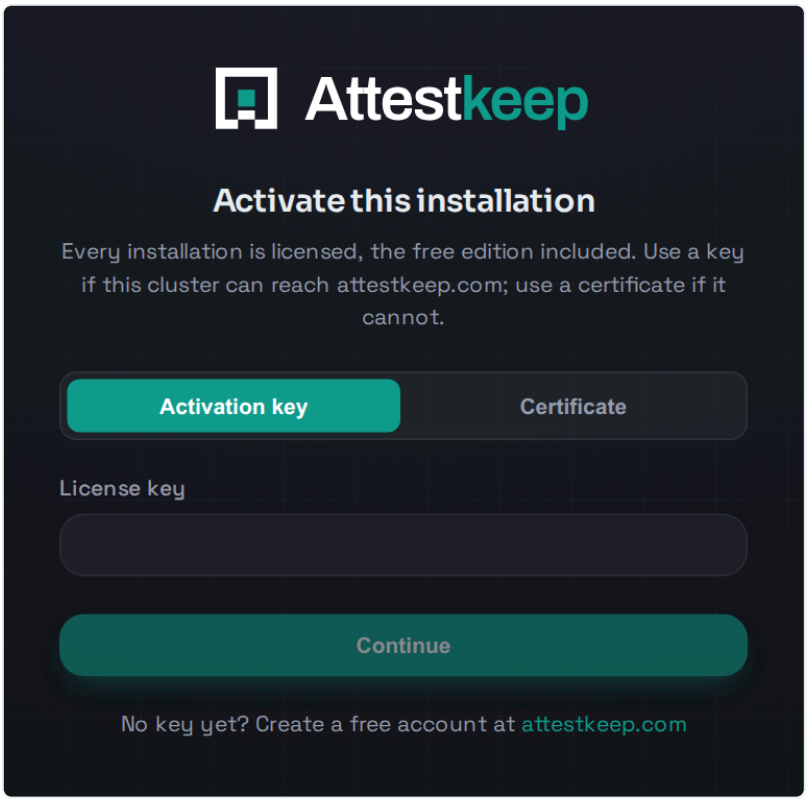
```
# installe depuis le registry ; aucun `helm repo add` nécessaire
helm install attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --version 1.4.0 \
  --namespace attestkeep --create-namespace \
  --set clusterName=production-eu \
  --wait
```

Renseignez `clusterName` dès la première installation, même pour un essai. Les notifications, les documents de preuve, le tableau de bord et Settings > About s'en servent pour nommer l'installation. Sans lui, ils reprennent le nom du premier node. C'est un nom que vous choisissez. Aucun nom d'hôte ni aucune adresse ne quitte le cluster.

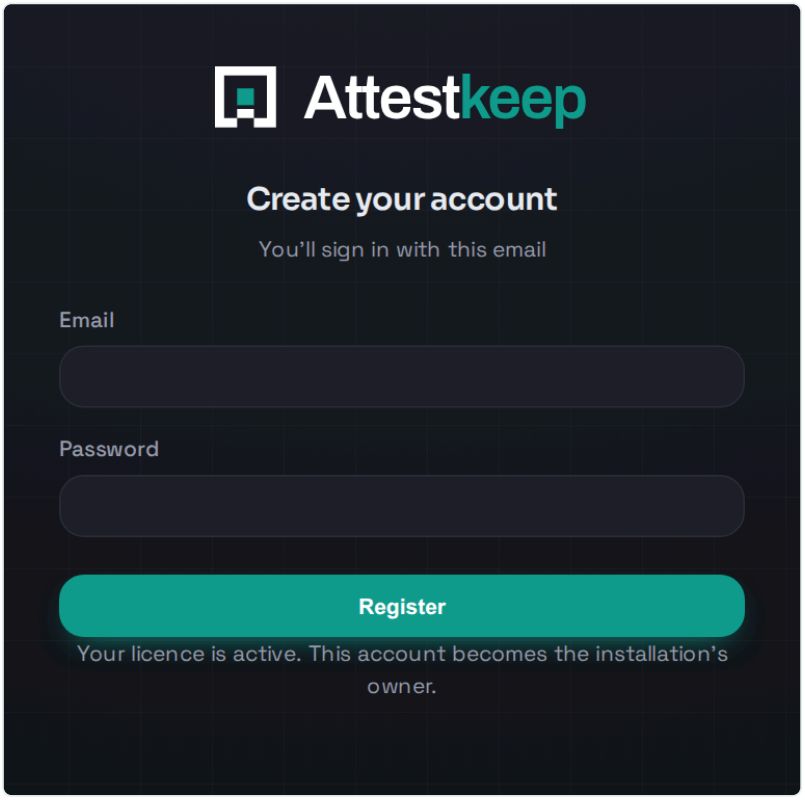
Vérifier que tout tourne et ouvrir la console

```
kubectl -n attestkeep get pods
kubectl -n attestkeep port-forward svc/attestkeep 8080:8080
```

Ouvrez `http://localhost:8080`. Une installation neuve affiche une seule fois deux écrans : l'activation, puis le compte owner. Ensuite, la console s'ouvre sur la vue d'ensemble de sécurité décrite au chapitre 5.



Écran 1, *Activate this installation*. Toute installation est sous licence, l'édition gratuite comprise. Dans l'onglet *Activation key*, vous collez votre clé puis choisissez *Continue*. La clé est échangée contre un certificat signé que l'opérateur vérifie dans le cluster. Un cluster qui ne joint pas attestkeep.com passe par l'onglet *Certificate* (chapitre 12).



Écran 2, *Create your account*. La ligne sous *Register* confirme que la licence est active et que ce compte devient le propriétaire de l'installation. L'écran se ferme dès que le compte existe. Si vous avez acheté un plan après une installation avec une clé Community, activez la clé payante plus tard sur la page Usage and License.

Si l'activation n'atteint pas le serveur de licences, l'écran le dit et en donne la cause. Un certificat auquel l'operator ne fait pas confiance (une CA privée sans `license.caBundle`), une connexion refusée ou expirée et un nom qui ne se résout

pas affichent tous "The licence server could not be reached: its certificate is not trusted, or the connection failed. If it uses a private CA, set license.caBundle in the chart." Avant la 1.4.0, les mêmes échecs ressemblaient à une clé invalide. Le chapitre 12 liste les messages.

Utiliser votre propre base

Les migrations sont intégrées au binaire et appliquées à la première connexion. Chacune est enregistrée et ne tourne qu'une fois. Pour utiliser une base que vous exploitez déjà, placez le DSN dans un Secret plutôt que sur la ligne de commande :

```
kubectl -n attestkeep create secret generic attestkeep-db \
  --from-literal=dsn='postgres://user:pass@db.internal:5432/attestkeep?sslmode=require'

helm upgrade attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --namespace attestkeep \
  --set postgresql.enabled=false \
  --set externalDsnExistingSecret.name=attestkeep-db
```

Les registries privés ne demandent aucune configuration. L'operator utilise, pour le scan comme pour la vérification de signature, les pull secrets que référence le pod et ceux de son service account.

La désinstallation se fait avec `helm uninstall`. La configuration du webhook est supprimée avec la release, et le contrôle d'admission s'arrête à ce moment. Le volume de la base n'est pas supprimé.

Mise à niveau

Une mise à niveau, c'est `helm upgrade`. Les migrations intégrées appliquent ce qui est nouveau et sautent le reste. `helm upgrade --reuse-values` fonctionne entre ces versions : les valeurs ajoutées depuis votre version installée prennent leur défaut du chart. Les migrations ne vont que vers l'avant. Là où une étape ci-dessous liste des migrations, revenir à la version précédente passe par la restauration d'une sauvegarde de la base, pas par `helm rollback`.

De → vers	Migrations	Que faire
1.3.7 → 1.4.0	0036 à 0046	Faites cette mise à niveau précise avec `--set webhook.failurePolicy=Fail` puis remettez la valeur d'avant, ou faites-la pendant un gel des changements. Un réplica 1.3.7 s'arrête sans se vider, et pendant environ une seconde une requête qu'il reçoit est admise sans contrôle sous `Ignore`. À partir de la 1.4.0, chaque mise à niveau vide les réplicas. Chaque migration attend au plus cinq secondes son verrou, puis le réplica redémarre et réessaie. Revenir à la 1.3.7 passe par une restauration. Les nouvelles valeurs du chart sont toutes facultatives, et le take down reste désactivé tant que vous ne posez pas `rbac.denyList.takedown: true`. Le ClusterRole obtient la lecture des objets de workload au-dessus d'un pod. Les canaux de notification existants ne sont pas abonnés à *Deny list hit* : cochez-le là où les refus doivent être signalés. Une base externe doit être en PostgreSQL 13 ou plus récent pour l'export SIEM. Une installation activée hors ligne avant la 1.4.0 a besoin d'une nouvelle demande d'activation avant que les téléchargements de preuves signées fonctionnent (chapitre 12).
1.3.6 → 1.3.7	0033 à 0035	Passez par `helm upgrade`, pas par un simple changement d'image : la nouvelle règle de webhook pour `pods/ephemeralcontainers` est dans le chart. À partir de cette version, `kubectl debug` avec une image que la policy n'autorise pas est refusé. 0033 ajoute la table qui conserve les exécutions What if, 0034 ajoute deux colonnes à la table des images et 0035 les remplit depuis le registre. Revenir à la 1.3.6 passe par une restauration. Aucune nouvelle valeur de chart.
1.3.2 → 1.3.6	Aucune	Rien à faire. Les versions 1.3.3, 1.3.4, 1.3.5 et 1.3.6 n'apportent ni migration ni nouvelle valeur de chart.
1.3.1 → 1.3.2	Aucune	Rien côté operator. `helm rollback` vers la 1.3.1 fonctionne. Si vous avez recopié les valeurs falcoctl de l'éditeur de règles de la 1.3.1, remettez `falco-rules:5` dans `install.refs` et `follow.refs`. Sans lui, falcoctl n'installe plus le fichier `/etc/falco/falco_rules.yaml` de Falco.
1.3.0 → 1.3.1	0026 à 0032	La ressource de policy n'a plus de champ `spec.runtime`. Retirez-le de vos fichiers de policy si vous l'aviez posé : `kubectl apply` refuse une policy qui le contient encore avec `unknown field "spec.runtime"`. La migration 0031 transforme les règles runtime enregistrées avec l'action supprimée en notification avec ticket. Les nouveaux canaux de notification sont abonnés à Runtime alert. Les canaux existants gardent leurs abonnements.
1.2.x → 1.3.0	0025	Réactivez le cluster. Un operator 1.3.0 ou plus récent exige un certificat qui porte le plafond de nodes et refuse un ancien avec "re-activate with a key issued by licence service 1.3.0 or later".
1.1.x → 1.2.0	0021 à 0024	Les événements Falco envoyés via Falcosidekick avec un bearer token n'arrivent plus tant que Falco n'est pas connecté en TLS mutuel (chapitre 8). Supprimer un utilisateur désactive désormais le compte.

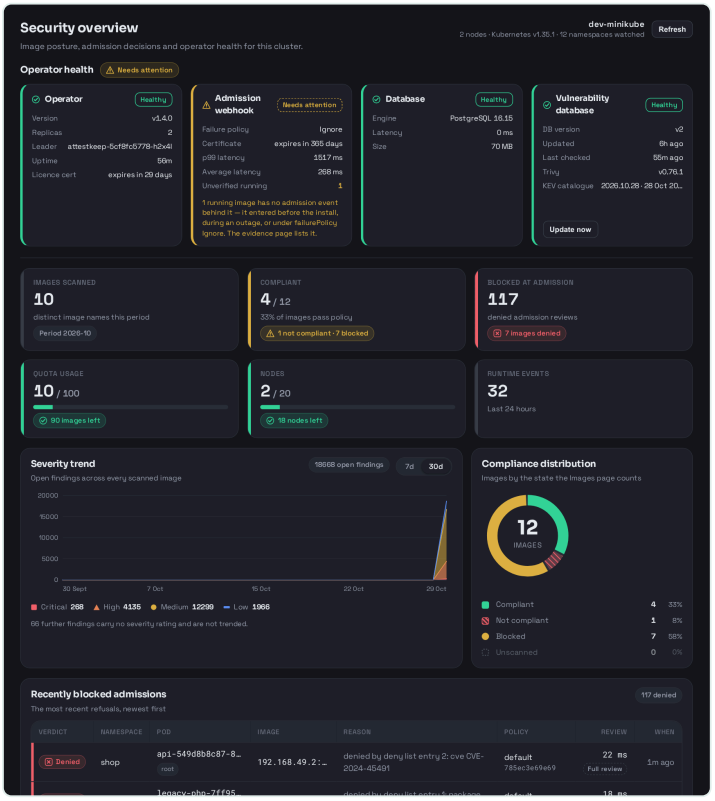
Une installation qui gère son propre RBAC ajoute le Secret `<release>-falco-mtls` au Role avant d'activer l'intégration Falco.

5. La console

La console est livrée avec la release. La même release Helm qui installe l'operator la sert. Elle ne charge ni police, ni script, ni analytics de l'extérieur. Elle existe en anglais et en turc, en thème clair ou sombre, avec un panneau d'aide sur chaque écran. Ce manuel montre l'interface anglaise et nomme menus et boutons tels qu'ils y apparaissent.

La barre latérale a quatre groupes. *Security* contient Images, Vulnerabilities, Policies, Deny list et What-if history. *Falco* contient Policies, Rules, Events et Falco integration, et apparaît quand l'intégration Falco est active. *Compliance* contient Compliance evidence et l'Audit Log. *Settings* contient Organisation, Appearance, My Notifications, Webhooks, Scanning, SMTP, Single Sign-On, Logging & telemetry, SIEM export, Usage and License, Users, Backup et About.

La vue d'ensemble de sécurité



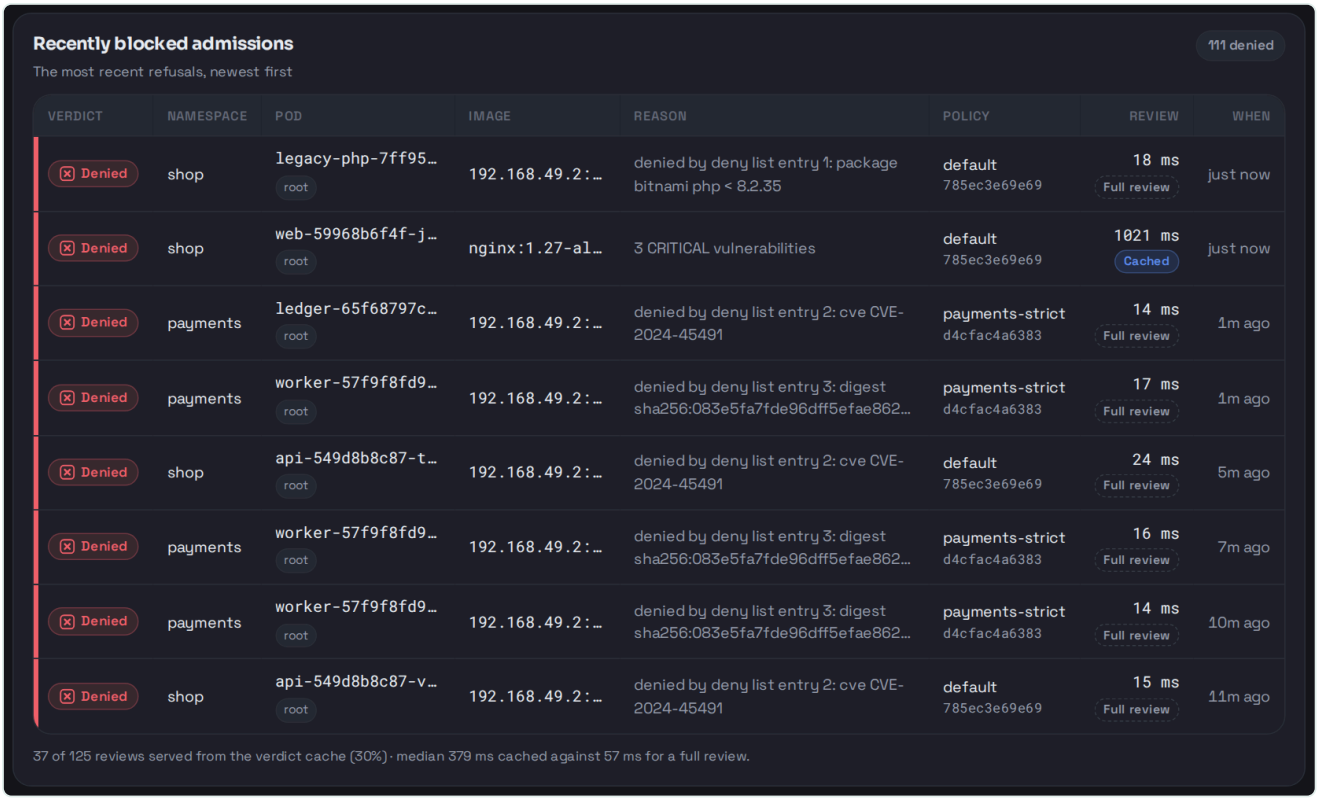
La vue de sécurité de l'installation photographiée pour ce manuel : version 1.4.0 avec deux réplicas, deux nodes, Kubernetes v1.35.1. La tuile *Admission webhook* demande de l'attention, car une image en cours d'exécution n'a aucun enregistrement d'admission derrière elle, et la tuile dit comment cela arrive. En dessous : 10 noms d'image scannés sur la période, 4 images conformes sur 12, 1 non conforme et 7 bloquées, 117 revues d'admission refusées, *Quota usage* à 10 sur 100, *Nodes* à 2 sur 20 et 32 événements runtime sur les dernières 24 heures. La répartition de conformité compte les mêmes quatre états que la page Images.

À partir de la 1.3.5, les deux tuiles ne lèvent plus d'alerte d'elles-mêmes. La tuile du webhook ne compare pas la latence à un budget fixe, parce que la première revue d'une image résout son digest auprès du registry et prend plus de temps que les revues suivantes, servies par le cache. L'operator rafraîchit la base de vulnérabilités et le catalogue KEV toutes les six heures. Un administrateur peut appuyer sur *Update now* pour télécharger tout de suite. La tuile dit alors si la base a changé, était déjà à jour, ou pourquoi le téléchargement a échoué, et la demande entre dans le journal d'audit sous `vulnerability_db_refresh`.

La carte Nodes est arrivée en 1.3.1. Tant que le cluster dépasse son plafond, le bandeau indique depuis quand et la date à partir de laquelle les nouveaux noms d'image seront refusés. Une fois cette date passée, il indique que le contrôle s'applique. Rien de ce qui tourne déjà ne s'arrête, dans un cas comme dans l'autre.

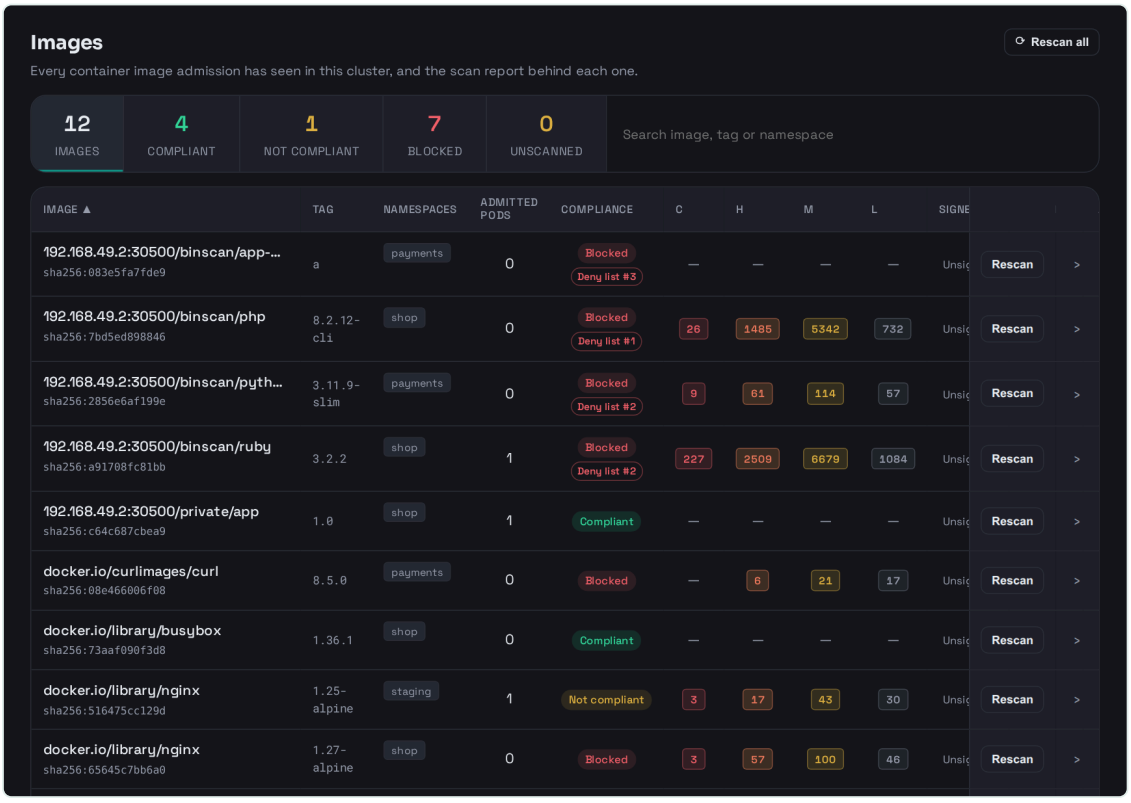
À partir de la 1.4.0, la tuile de conformité, ses pastilles et la répartition comptent les quatre mêmes états que la page Images, selon la même règle, et les deux pages donnent les mêmes chiffres. Une image dont la dernière admission a été

refusée est Blocked même si son scan respecte la policy, et une image admise sous break-glass est Not compliant. Les parts Break-glass et Unknown ont disparu. La *Severity trend* compte les vulnérabilités ouvertes de toutes les images connues, c'est-à-dire les lignes de la page Vulnerabilities, et son titre est la somme des quatre bandes en dessous. *Leader* désigne le réplica qui détient le verrou du contrôleur.



Recently blocked admissions ne liste que les refus, du plus récent au plus ancien. Un refus de la deny list nomme son entrée, par exemple *denied by deny list entry 1: package bitnami php < 8.2.35*. Un refus des règles d'une policy donne la règle, ici 3 CRITICAL vulnerabilities. Chaque ligne montre la policy et son hash, la durée de revue et si le verdict venait du cache. La ligne sous le tableau compare les revues servies par le cache et les revues complètes.

Images



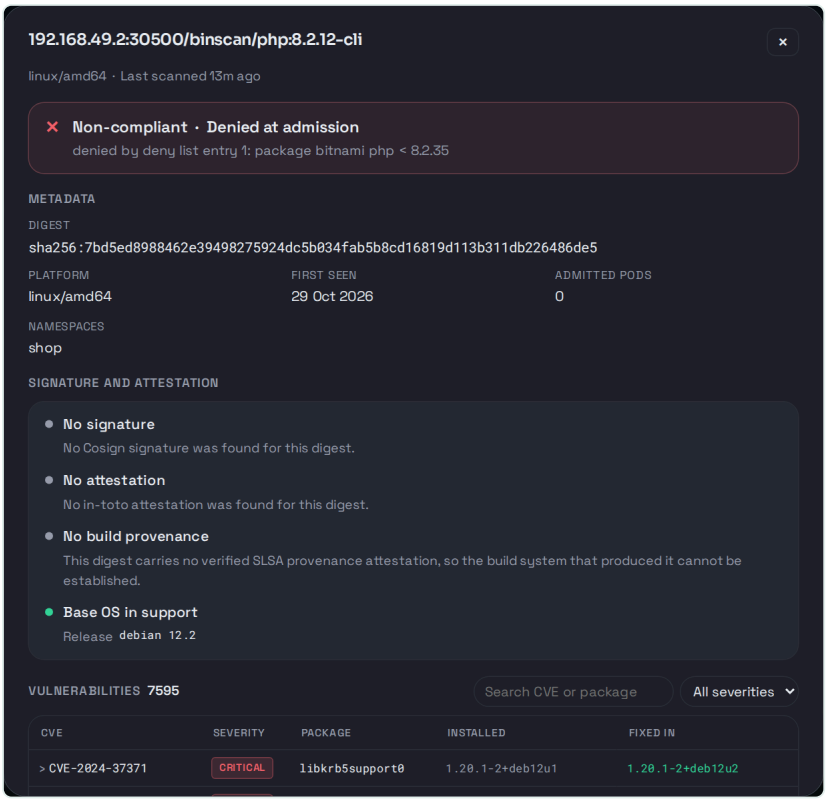
Toutes les images que l'admission a vues dans ce cluster : 12 images, 4 conformes, 1 non conforme, 7 bloquées et aucune non scannée, si bien que les tuiles s'additionnent pour donner la liste. Une ligne refusée par la deny list porte *Deny list #N* sous son état. Les colonnes donnent les namespaces, les pods admis, les findings par sévérité et l'état de signature. *Rescan* met en file le scan d'une image, *Rescan all* celui de toutes.

À partir de la 1.3.7, chaque image est dans un seul des quatre états, et les tuiles s'additionnent donc pour donner la liste. Le premier état qui s'applique l'emporte :

- **Blocked** : sa dernière revue d'admission a été refusée.
- **Unscanned** : admise, sans scan pour l'instant.
- **Compliant** : admise, scannée et dans la policy.
- **Not compliant** : admise et scannée mais hors de la policy, en général sous une policy en mode Audit ou pendant un break-glass.

Blocked suit la dernière revue d'admission, pas le verdict que le scanner rend plus tard. Une image admise sous une policy en Audit avant son premier scan, et chez qui on découvre ensuite des vulnérabilités critiques, est Not compliant, et son détail affiche "Admitted; the next admission would be denied". À partir de la 1.4.0, la pastille Compliance de chaque ligne nomme son état avec les mêmes mots, les filtres sélectionnent sur cet état, et le verdict propre au scan se trouve dans le détail de l'image. Une ligne refusée par la deny list porte le badge *Deny list #N*, avec le refus au survol. La colonne *Admitted pods* compte les pods admis avec l'image depuis sa première apparition. Elle ne baisse pas quand un pod s'arrête, et une admission refusée n'est jamais comptée.

À partir de la 1.3.1, les listes Images et Vulnerabilities se lisent page par page. Filtres, recherche et tri s'exécutent dans l'operator, et les compteurs au-dessus de chaque tableau couvrent toutes les lignes.

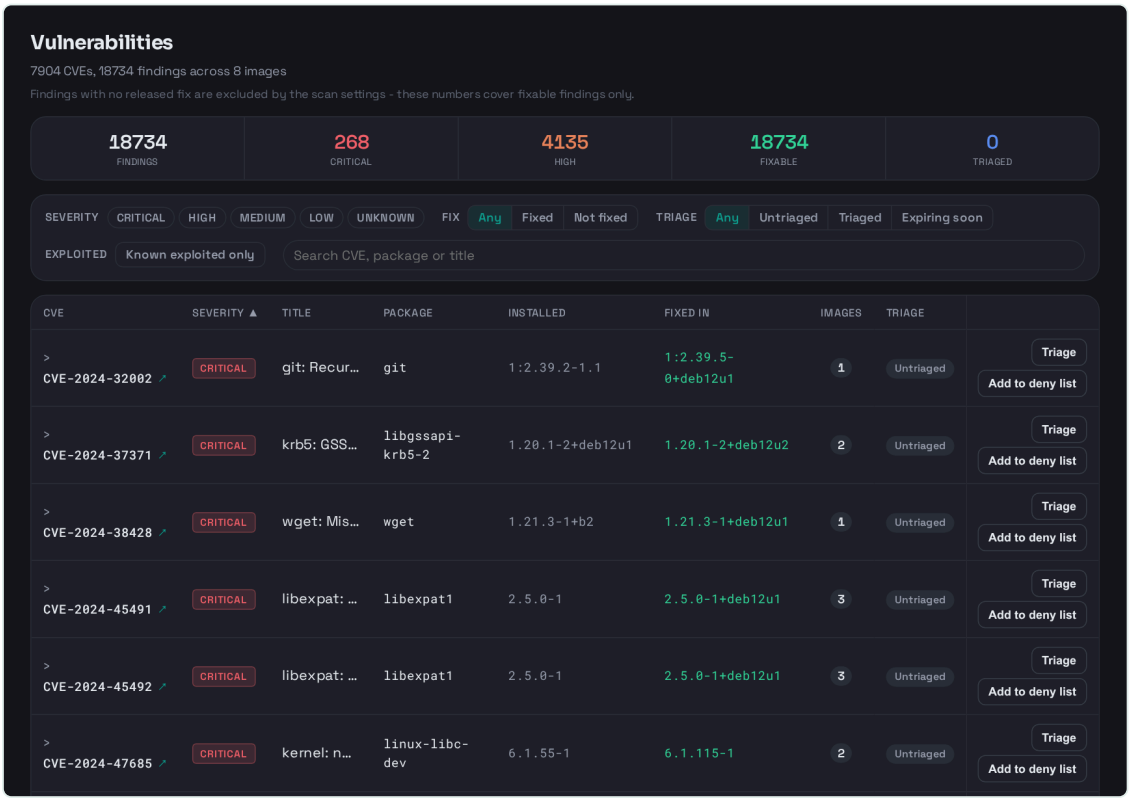


Le panneau de `php:8.2.12-cli`. L'admission l'a refusée à cause de l'entrée 1 de la deny list, et le panneau cite le refus. Au-dessus des findings, il répond à quatre questions : y a-t-il une signature cosign, une attestation in-toto, une build provenance SLSA vérifiée, et l'OS de base est-il encore supporté. Ici Debian 12.2 est supporté et le scan rapporte 7595 vulnérabilités.

Le détail propose *Add to deny list* avec le digest de l'image déjà rempli, et affiche *Listed* pour un digest déjà sur la liste. Quand la deny list contient une entrée à laquelle répond le contrôle des exécutables, le détail montre aussi ses réponses pour ce digest sous *Deny list checks* (chapitre 7).

Le même scan est disponible dans le cluster pour qui le lit avec kubectl. Chaque `ImageSecurityReport` porte le résumé dans `status.summary`, les vulnérabilités dans `status.vulnerabilities` (les plus graves d'abord, 1000 au plus) et l'heure du scan dans `status.lastScanned`. Une décision de triage est recopiée sur le rapport avec `triagedBy` à `user:<id de compte>`, jamais un nom.

Vulnerabilities et triage

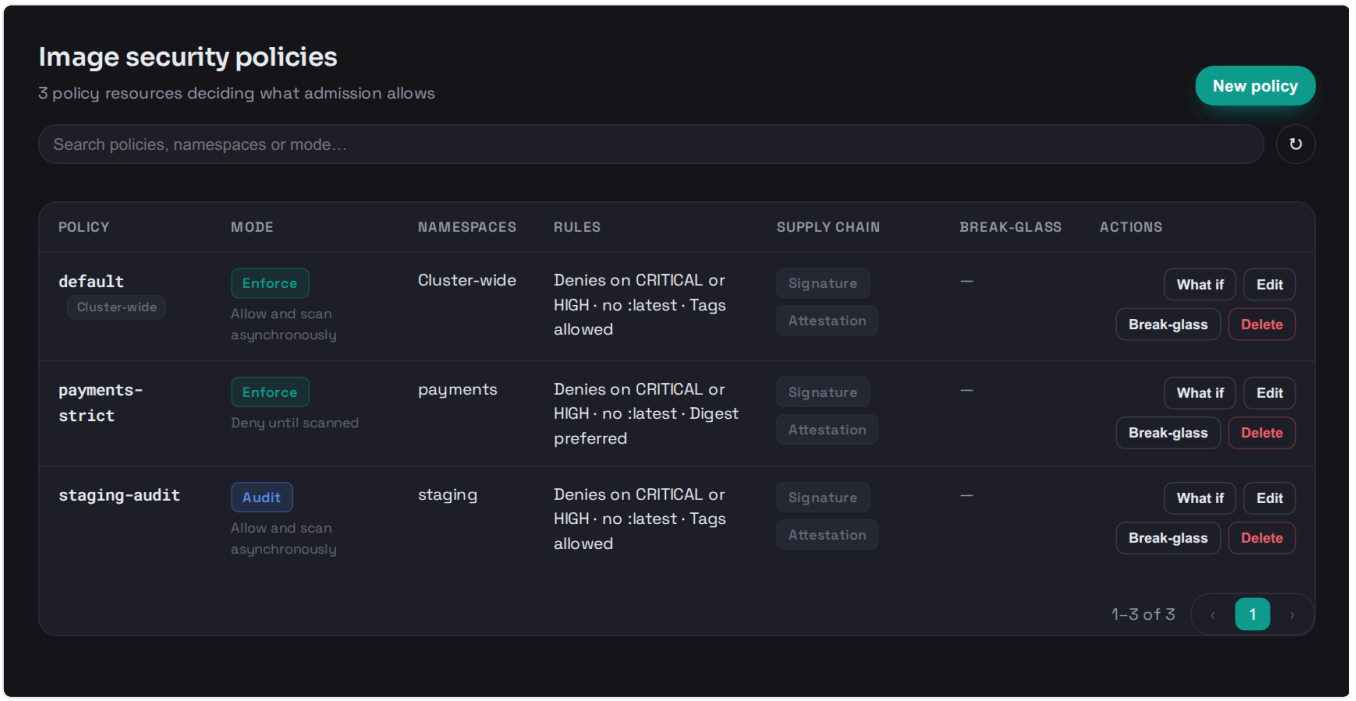


Les findings de toutes les images, groupés par CVE, versions installée et corrigée côte à côte : 7904 CVE et 18734 findings sur 8 images. La ligne sous le titre indique que les réglages de scan de cette installation écartent les findings sans correctif publié. Chaque ligne propose *Triage* et, depuis la 1.4.0, *Add to deny list*.

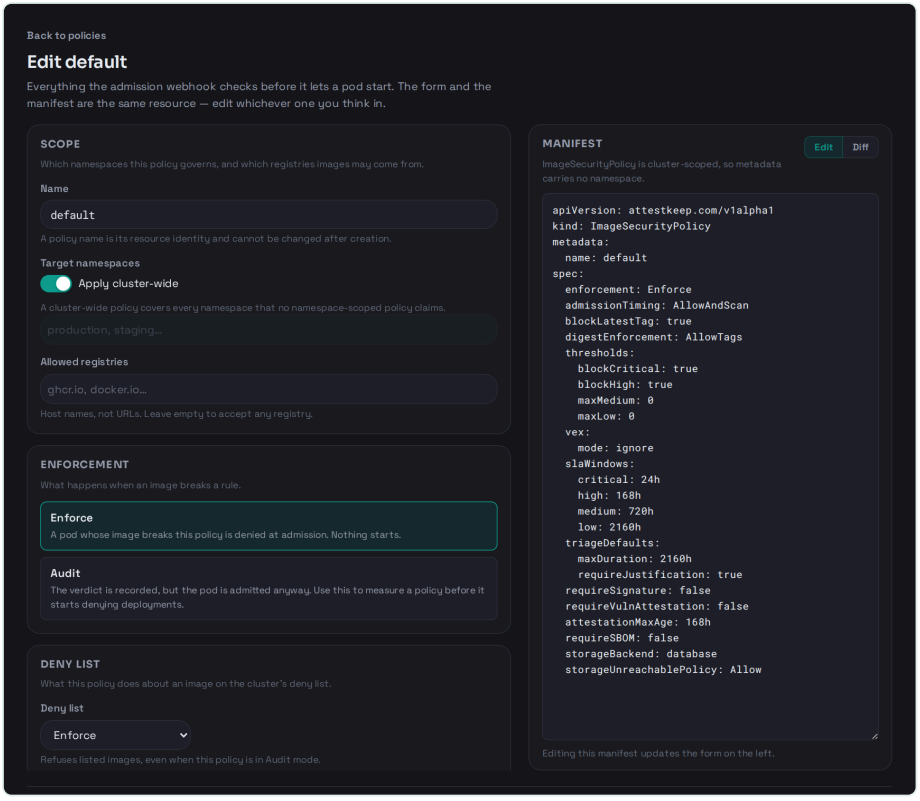
À partir de la 1.4.0, le titre compte toutes les CVE que sélectionnent les filtres, pas seulement les 25 de la page, et les vulnérabilités et images à côté sont comptées sous les mêmes filtres. Un digest tiré par deux références, son tag et son digest, est une seule image : ses vulnérabilités comptent une fois, et elle apparaît une fois avec toutes ses références. *Add to deny list* sur une ligne propose l'identifiant d'avis, ou pour une entrée de paquet, le paquet et sa version installée.

Le triage est un enregistrement que lit un auditeur, il ne peut donc pas être une mise en sourdine silencieuse. Vous choisissez un statut (false positive, accepted risk ou under investigation), écrivez une justification et fixez une échéance. À l'échéance, la vulnérabilité retrouve le verdict de la policy. Chaque décision enregistre qui l'a prise et quand, et apparaît dans le journal d'audit et dans le prochain paquet de preuves.

Les policies d'admission



Trois policies. `default` applique ses règles sur tout le cluster, `payments-strict` les applique dans `payments` et refuse tant que l'image n'est pas scannée, et `staging-audit` tourne en mode Audit dans `staging`. Chaque ligne montre le mode, les namespaces couverts, un résumé des règles et les contrôles de supply chain exigés, avec *What if*, *Edit*, *Break-glass* et *Delete*.



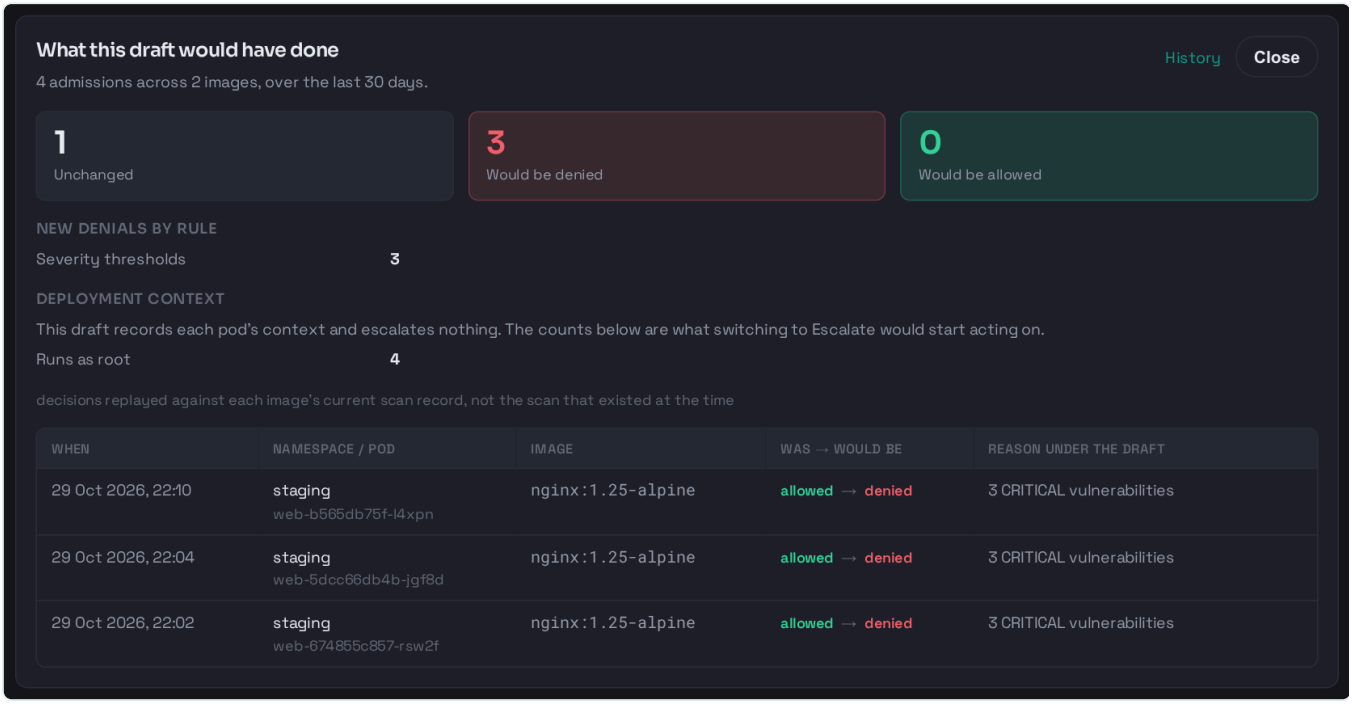
L'éditeur de `default`. Le formulaire et le manifeste sont la même ressource, vous modifiez celui que vous préférez. La nouvelle section *Deny list* est sur *Enforce* : les images listées sont refusées même quand la policy elle-même est en mode Audit.

À partir de la 1.3.7, *Apply to cluster* n'apparaît que pour les comptes autorisés à écrire des policies. Les autres gardent le résultat What if, Download et Copy for git. *Cancel* demande confirmation avant de jeter des modifications non enregistrées. À partir de la 1.4.0, un enregistrement conserve les parties de la ressource que le formulaire ne modifie pas, comme la trace d'une fenêtre de break-glass refermée ou un bloc écrit avec kubectl. Le formulaire a une section *Deny list*

avec *Enforce*, *Audit* et *Off* (chapitre 7). Une nouvelle policy part des mêmes délais de SLA et de triage que l'operator et la documentation (critical 24h, high 168h, medium 720h, low 2160h, triage 2160h). Une policy écrite avec kubectl qui laisse un délai vide s'ouvre avec ces valeurs, et le diff le signale avant l'application.

Ce qu'une policy refuserait dans votre cluster dépend de vos images, de vos scans et de vos décisions de triage, pas seulement de son texte. *What if* répond à cette question à partir des enregistrements du cluster. Le brouillon est rejoué sur les admissions des 7, 30 ou 90 derniers jours, avec la même fonction que le webhook d'admission. Rien n'est appliqué : aucun pod n'est touché, aucun registry n'est contacté et le compteur de licence ne bouge pas. Les décisions sont rejouées avec le scan actuel de chaque image, pas celui de l'époque.

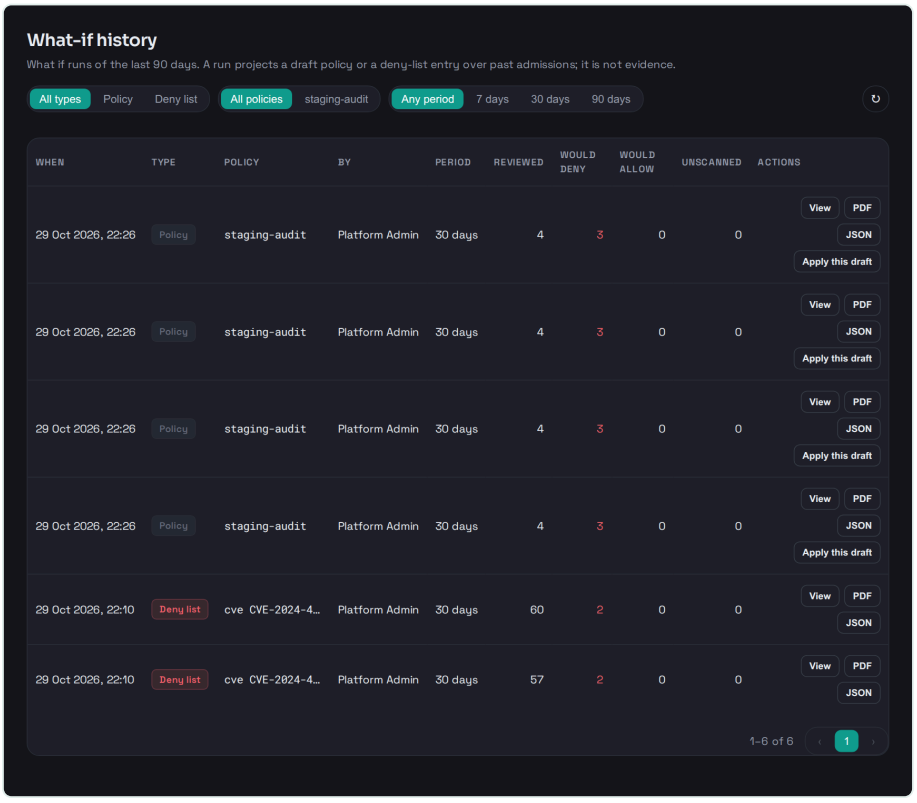
What if sur une ligne de la liste des policies est le chemin court. Une policy en Audit est rejouée comme si elle était en Enforce, et une policy en Enforce telle qu'elle est, sur les trente derniers jours. Le résultat s'ouvre dans l'éditeur, et rien n'est appliqué.



`staging-audit` rejouée comme brouillon appliqué sur les 30 derniers jours : 4 admissions sur 2 images, 1 inchangée, 3 seraient refusées par les seuils de sévérité et 0 serait autorisée. Le panneau liste les règles derrière les nouveaux refus, les signaux de deployment context des pods rejoués (4 tournent en root) et chaque admission modifiée avec la raison du brouillon.

Scan now. Quand des images de la période n'ont pas de scan, le résultat les nomme. Un deployer ou un administrateur peut mettre en file des scans pour exactement ces digests, 500 au plus par requête, et chaque requête entre dans le journal d'audit sous `rescan`. Le rejeu ne se relance pas tout seul. Appuyez de nouveau sur *What if* une fois les scans terminés.

What-if history



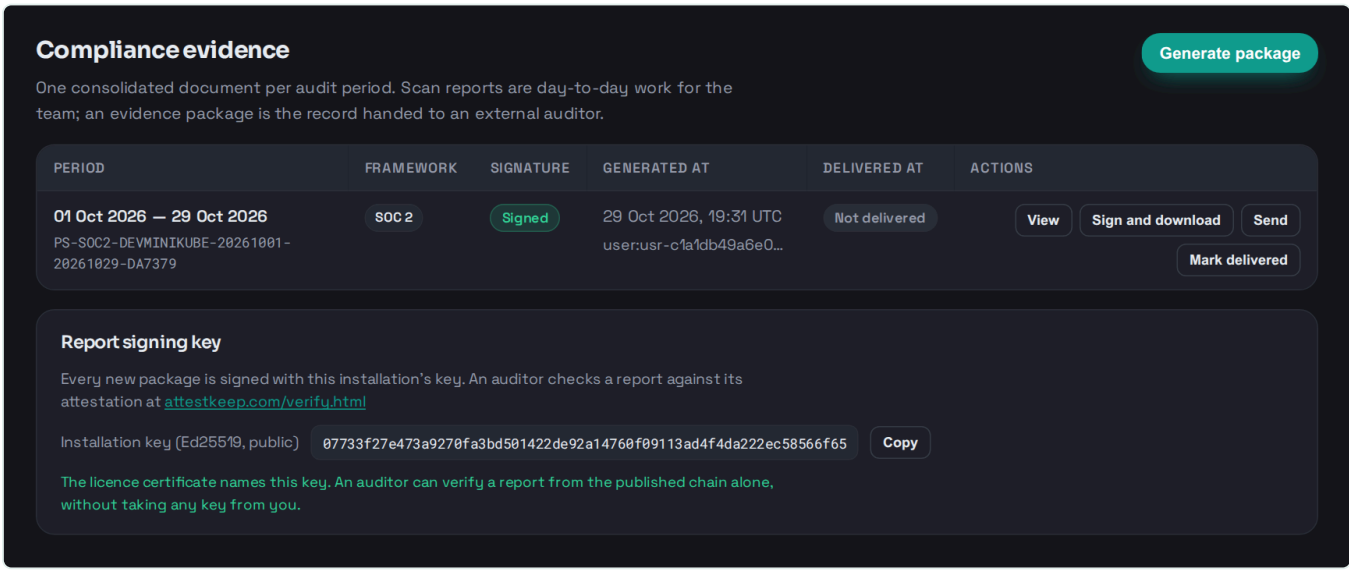
Security > What-if history garde les deux sortes d'exécution. La colonne et le filtre *Type* distinguent un brouillon de policy d'une entrée de deny list : ici quatre exécutions de `staging-audit` et deux de l'entrée pour CVE-2024-45491. Chaque exécution indique qui l'a lancée, la période et les comptes. *Apply this draft* n'est proposé que pour les exécutions de policy.

À partir de la 1.4.0, l'historique garde deux sortes d'exécutions : un brouillon de policy, et une entrée de deny list essayée avant son ajout ou depuis sa ligne (chapitre 7). Une colonne *Type* et un filtre par type les distinguent. Un rejeu lancé par un deployer ou un administrateur est conservé quatre-vingt-dix jours, cinq cents exécutions au plus de chaque sorte. Un viewer peut aussi rejouer, la réponse s'affiche mais n'est pas conservée. Chaque exécution peut être revue, téléchargée en PDF ou en JSON, ou, par un administrateur, ouverte dans l'éditeur avec *Apply this draft*. Cela charge le brouillon enregistré et n'applique rien.

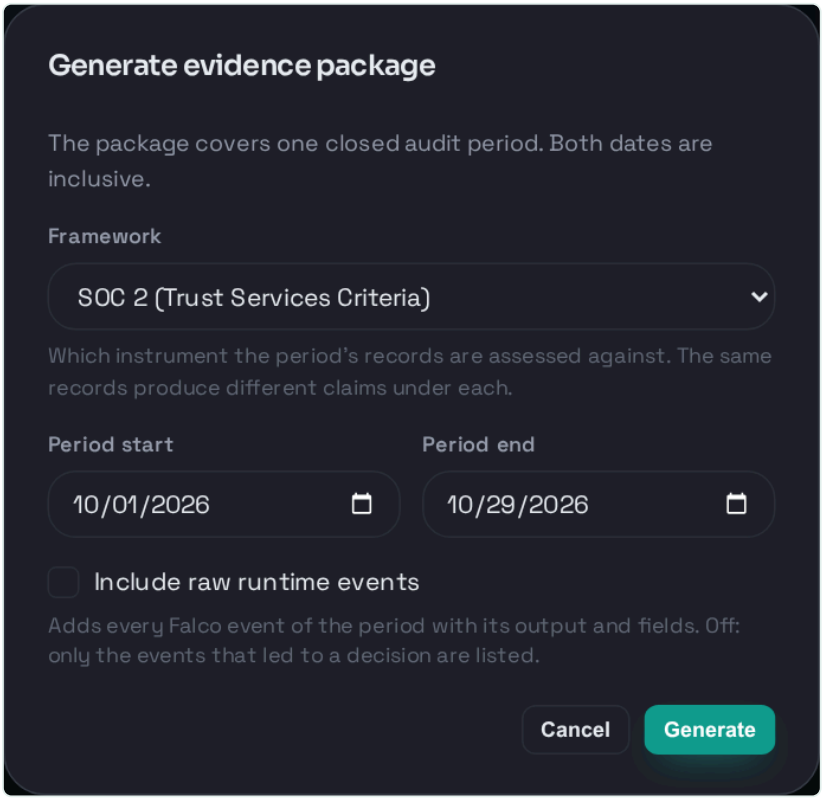
> **Un téléchargement est une projection, pas une preuve.** Le PDF le dit sur chaque page et n'est ni signé ni scellé. Une exécution n'entre jamais dans un paquet de preuves, parce que le paquet consigne ce qui s'est passé et un rejeu ce qui se serait passé. Chaque téléchargement entre dans le journal d'audit sous `policy_whatif_download`.

L'API est `POST /api/v1/policies/whatif`. Elle accepte un `source` facultatif, la policy dont le brouillon est issu (un nom inconnu reçoit `400`), et répond avec `kept`, `run_id`, `unscanned`, `unscanned_total` et `notes` à côté des compteurs. Un compte peut lancer au plus dix rejeux par minute sur chaque réplica de l'operator. Le suivant reçoit `429` et `Retry-After`.

Compliance evidence

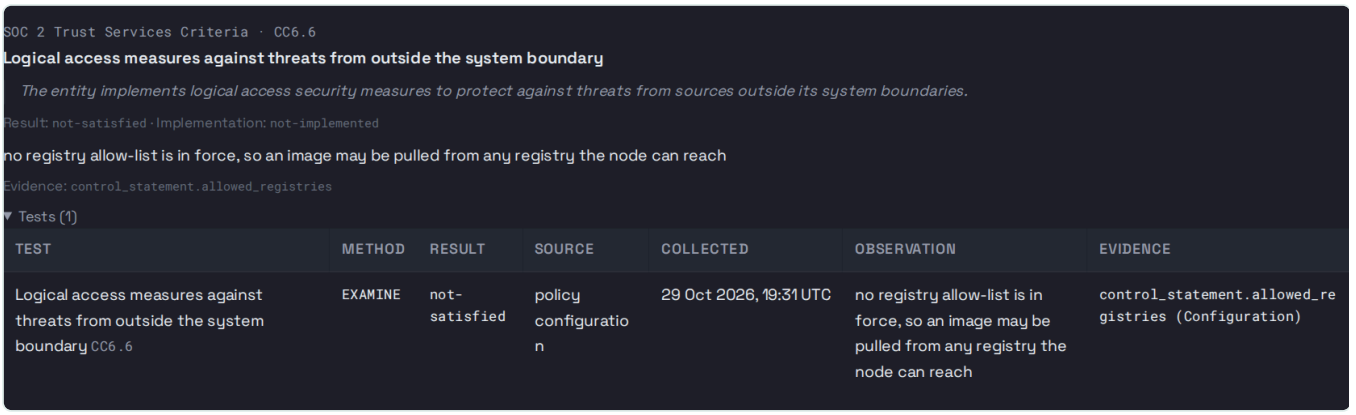


Un document par période d'audit. La ligne donne la période, le framework, l'état de signature, quand et par quel compte le paquet a été généré, écrit `user:<id de compte>`, et s'il a été remis. En dessous, la clé de signature publique de l'installation, contre laquelle un auditeur vérifie un rapport.



Generate evidence package. La liste *Framework* contient les jeux de contrôles cartographiés que la licence autorise. La période est une période d'audit close, les deux dates incluses. *Include raw runtime events* ajoute tous les événements Falco de la période. Décochée, seuls les événements qui ont mené à une décision sont listés.

Quand une licence couvre moins de frameworks que l'installation n'en a déjà utilisés, par exemple à la fin d'une licence payante, le prochain owner ou administrateur qui se connecte doit choisir lesquels garder. Le chapitre 12 donne le détail.



SOC 2 Trust Services Criteria · CC6.6

Logical access measures against threats from outside the system boundary

The entity implements logical access security measures to protect against threats from sources outside its system boundaries.

Result: not-satisfied · Implementation: not-implemented

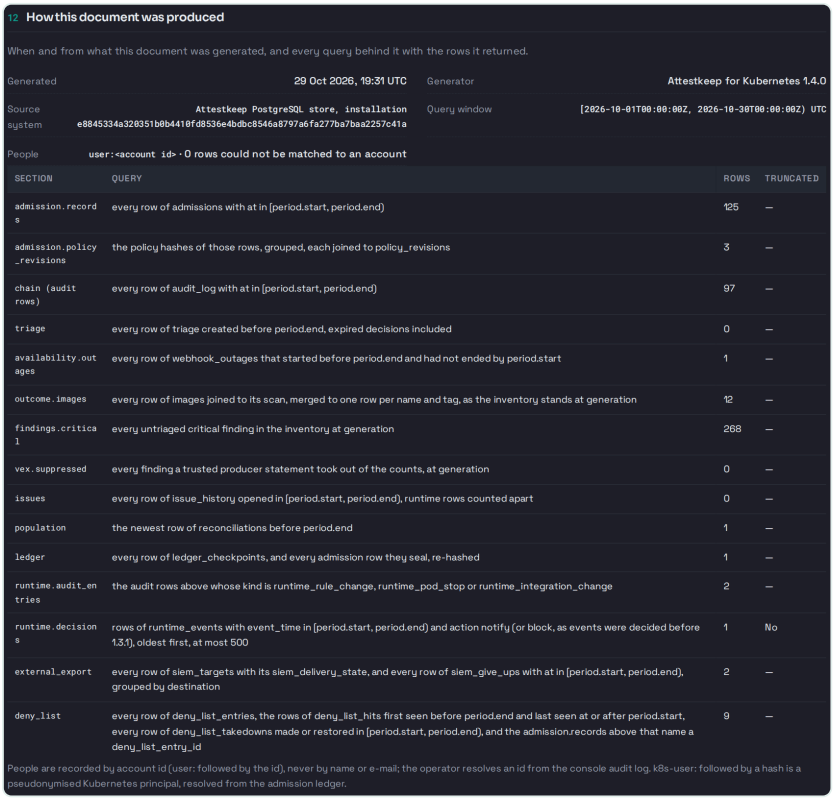
no registry allow-list is in force, so an image may be pulled from any registry the node can reach

Evidence: control_statement.allowed_registries

▼ Tests (1)

TEST	METHOD	RESULT	SOURCE	COLLECTED	OBSERVATION	EVIDENCE
Logical access measures against threats from outside the system boundary CC6.6	EXAMINE	not-satisfied	policy configuration	29 Oct 2026, 19:31 UTC	no registry allow-list is in force, so an image may be pulled from any registry the node can reach	control_statement.allowed_registries (Configuration)

Une exigence dans la vue du rapport, son test ouvert. CC6.6 n'est pas satisfaite parce qu'aucune liste de registries autorisés n'est en vigueur, et le test le dit. La méthode est EXAMINE parce qu'elle lit une configuration. Le tableau donne aussi le résultat, la source, le moment de sa lecture, l'observation et le chemin dans le rapport.



12 How this document was produced

When and from what this document was generated, and every query behind it with the rows it returned.

Generated 29 Oct 2026, 19:31 UTC

Generator Attestkeep for Kubernetes 1.4.0

Source system Attestkeep PostgreSQL store, installation e8845334a320351b0b4410fd8536e4bdbc8546a8797a6fa277ba7baa2257c41a

Query window [2026-10-01T00:00:00Z, 2026-10-30T00:00:00Z) UTC

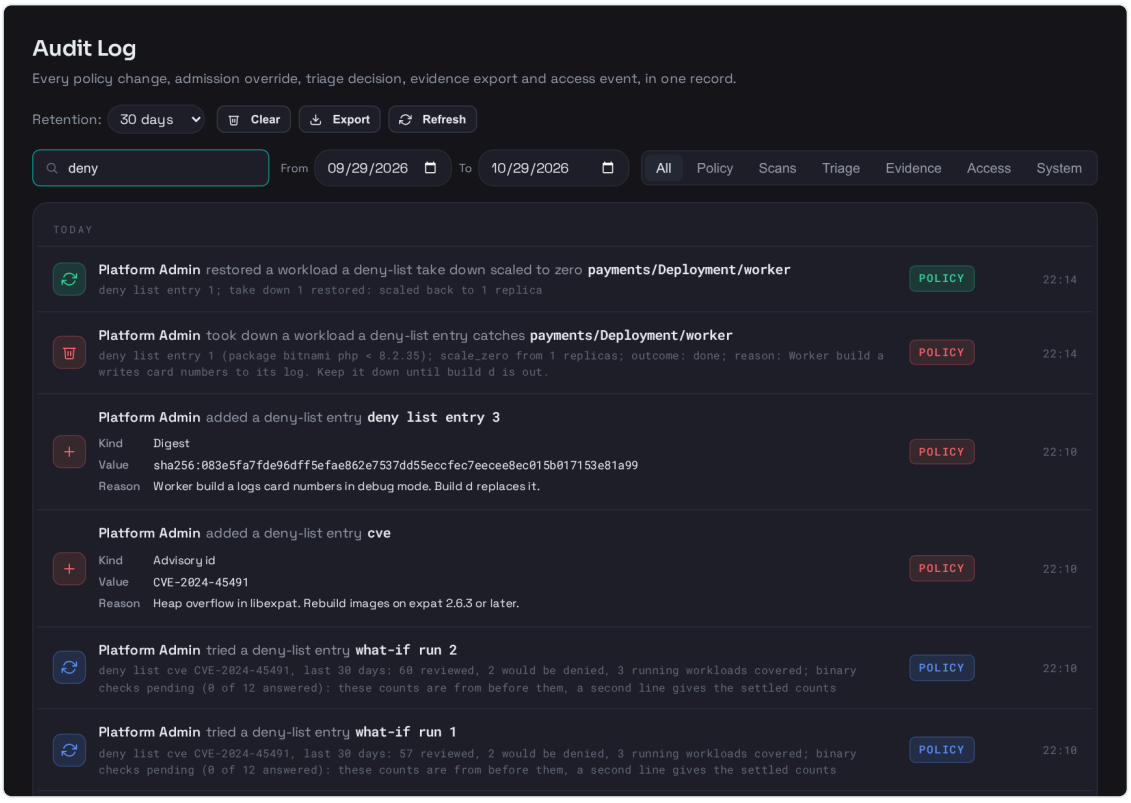
People user:<account id> · 0 rows could not be matched to an account

SECTION	QUERY	ROWS	TRUNCATED
admission.records	every row of admissions with at in [period.start, period.end)	125	—
admission.policy_revisions	the policy hashes of those rows, grouped, each joined to policy_revisions	3	—
chain (audit rows)	every row of audit_log with at in [period.start, period.end)	97	—
triage	every row of triage created before period.end, expired decisions included	0	—
availability.outages	every row of webhook_outages that started before period.end and had not ended by period.start	1	—
outcome.images	every row of images joined to its scan, merged to one row per name and tag, as the inventory stands at generation	12	—
findings.critical	every untriaged critical finding in the inventory at generation	268	—
vex.suppressed	every finding a trusted producer statement took out of the counts, at generation	0	—
issues	every row of issue_history opened in [period.start, period.end), runtime rows counted apart	0	—
population	the newest row of reconciliations before period.end	1	—
ledger	every row of ledger_checkpoints, and every admission row they seal, re-hashed	1	—
runtime.audit_entries	the audit rows above whose kind is runtime_rule_change, runtime_pod_stop or runtime_integration_change	2	—
runtime.decisions	rows of runtime_events with event_time in [period.start, period.end) and action notify (or block, as events were decided before 1.3.1), oldest first, at most 500	1	No
external_export	every row of siem_targets with its siem_delivery_state, and every row of siem_give_ups with at in [period.start, period.end), grouped by destination	2	—
deny_list	every row of deny_list_entries, the rows of deny_list_hits first seen before period.end and last seen at or after period.start, every row of deny_list_takedowns made or restored in [period.start, period.end), and the admission.records above that name a deny_list_entry_id	9	—

People are recorded by account id (user: followed by the id), never by name or e-mail; the operator resolves an id from the console audit log. k8s-user: followed by a hash is a pseudonymised Kubernetes principal, resolved from the admission ledger.

How this document was produced dans le paquet montré dans ce manuel, généré par la 1.4.0. On y trouve l'heure de génération, le générateur, le système source, la fenêtre de requête (fin exclue, UTC), la façon de nommer les personnes et la requête de chaque section avec les lignes renvoyées. Les deux dernières lignes, `external_export` et `deny_list`, sont nouvelles en 1.4.0.

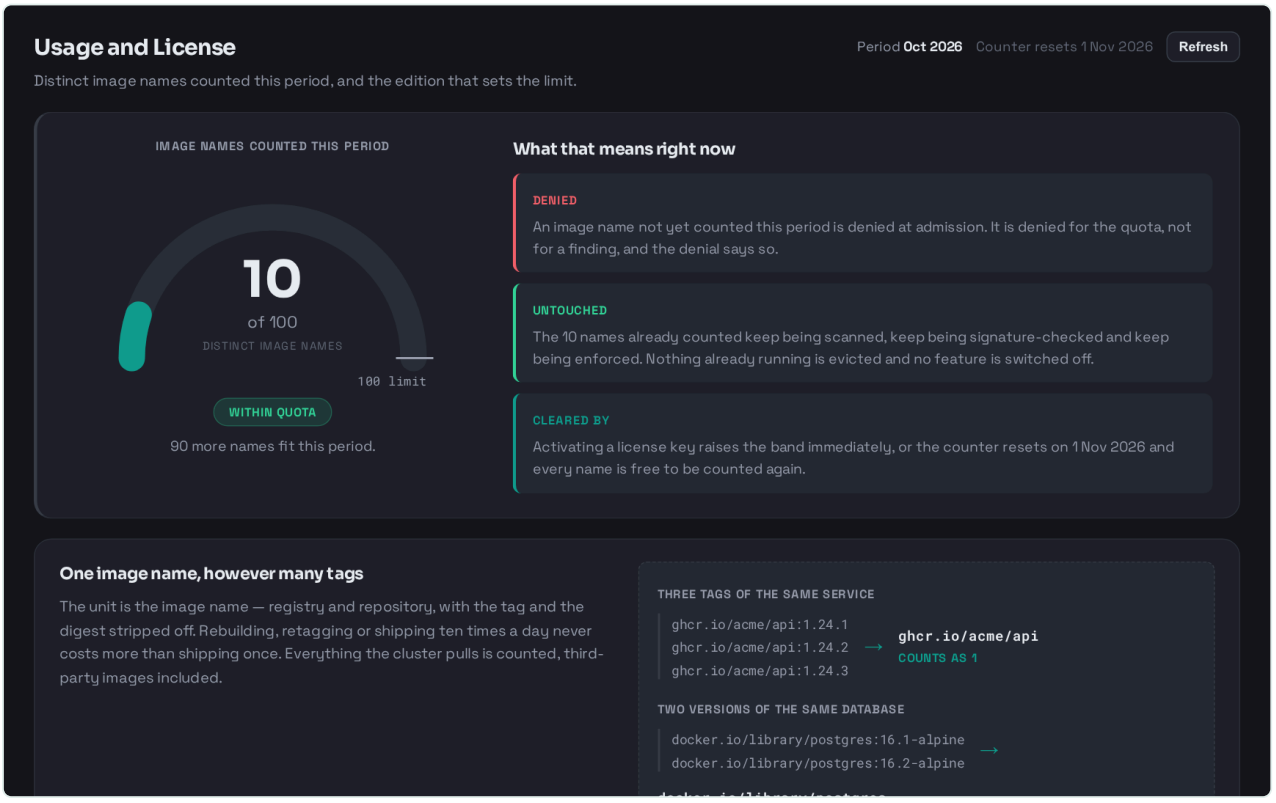
Audit Log



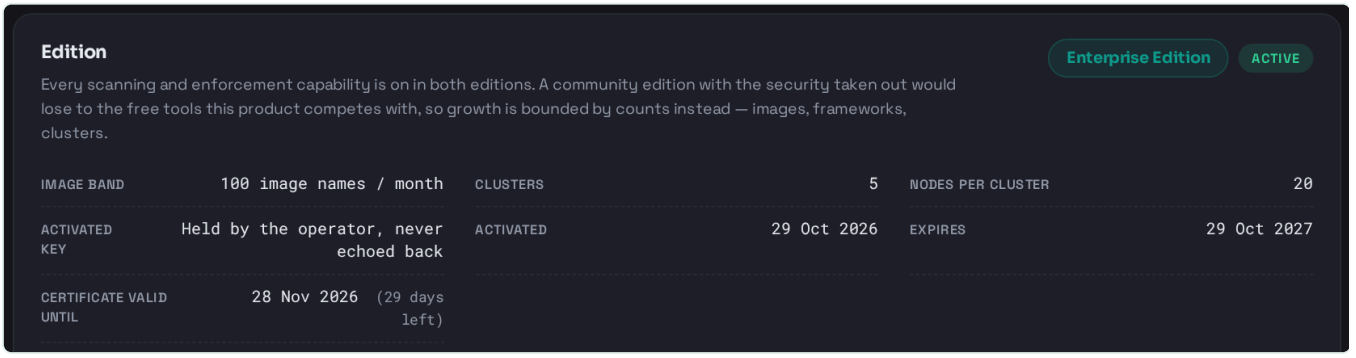
Le journal d'audit recherché sur *deny*. En haut, une restauration et le take down qui la précède, puis deux entrées ajoutées avec leur type, leur valeur et leur raison, et deux exécutions What if d'une entrée. On cherche par acteur, policy, image ou CVE, et on filtre par catégorie : Policy, Scans, Triage, Evidence, Access ou System.

Les heures s'affichent dans votre heure locale, et l'export CSV garde ses horodatages ISO. Les modifications et tests de canaux de notification, les changements de l'intégration Falco et des policies runtime, les pods arrêtés par une policy runtime, les actions de validation, d'enregistrement et de publication de l'éditeur de règles, les exécutions et téléchargements What if, Scan now et Update now y sont tous consignés. À partir de la 1.4.0, s'y ajoutent les entrées de deny list ajoutées et retirées, les take downs, les rétablissements et Check again (`deny_list_add` , `deny_list_remove` , `deny_list_takedown` , `deny_list_restore` , `deny_list_check_again`), ainsi que les destinations SIEM ajoutées, modifiées, testées et retirées (`siem_target_add` , `siem_target_update` , `siem_target_test` , `siem_target_remove`). Les actions sur les canaux n'entrent pas dans les paquets de preuves, parce qu'elles décrivent ce qu'une personne a fait à l'installation, pas ce que le contrôle a décidé à propos d'une image.

Usage and License



Les noms d'image distincts comptés sur la période : 10 sur 100, dans le quota, encore 90 avant que les nouveaux noms soient refusés. La page explique ce qui se passe à la limite. Depuis la 1.4.0, un nom refusé au titre de l'allocation n'est pas compté, et une tranche pleine affiche *Allowance reached*.



La section *Edition* de cette installation : Enterprise Edition, active, avec 100 noms d'image par mois, 5 clusters et 20 nodes par cluster. La licence court du 29 octobre 2026 au 29 octobre 2027. Le certificat que détient l'opérateur est valable jusqu'au 28 novembre 2026. Un certificat en ligne dure au plus trente jours, et la vérification quotidienne de la licence le remplace avant son expiration (chapitre 12).

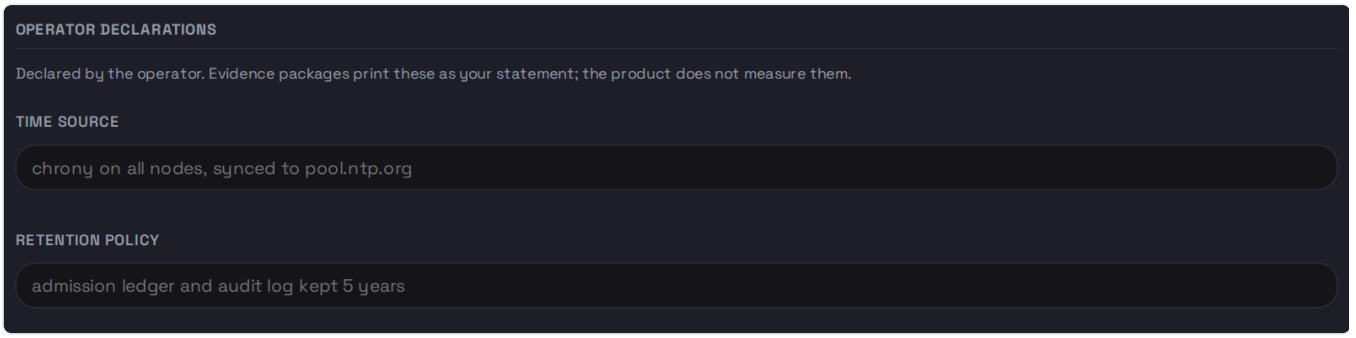
L'unité est le nom d'image : registry et dépôt, sans tag ni digest. Retaguer ou livrer dix fois par jour coûte autant que livrer une fois. À partir de la 1.3.1, les chiffres du mois en cours sur cette page et sur le tableau de bord sont lus par rapport à la licence en vigueur. Après la fin d'une licence, ils montrent le quota Community, les mêmes chiffres que ceux qu'applique l'admission.

À partir de la 1.4.0, un nom d'image refusé à cause du quota mensuel n'est pas compté dedans. Le compteur s'arrête à la limite et affiche par exemple 8 of 8 avec *Allowance reached*, là où la 1.3.7 continuait de compter les noms refusés au-delà. Le refus et son enregistrement d'admission sont inchangés.

Organisation et utilisateurs

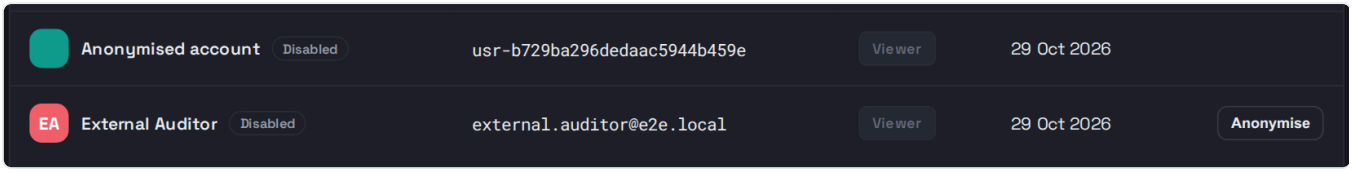
Settings > Organisation contient ce que les paquets de preuves disent de l'organisation derrière l'installation. Sous *Operator declarations*, vous indiquez votre source de temps et votre règle de conservation. Le produit ne mesure ni l'une

ni l'autre, et les paquets les impriment comme déclarées par l'exploitant.



Les deux champs de déclaration. Sur cette installation, ils sont vides. Le texte gris est l'exemple que le formulaire affiche tant que vous n'avez rien saisi.

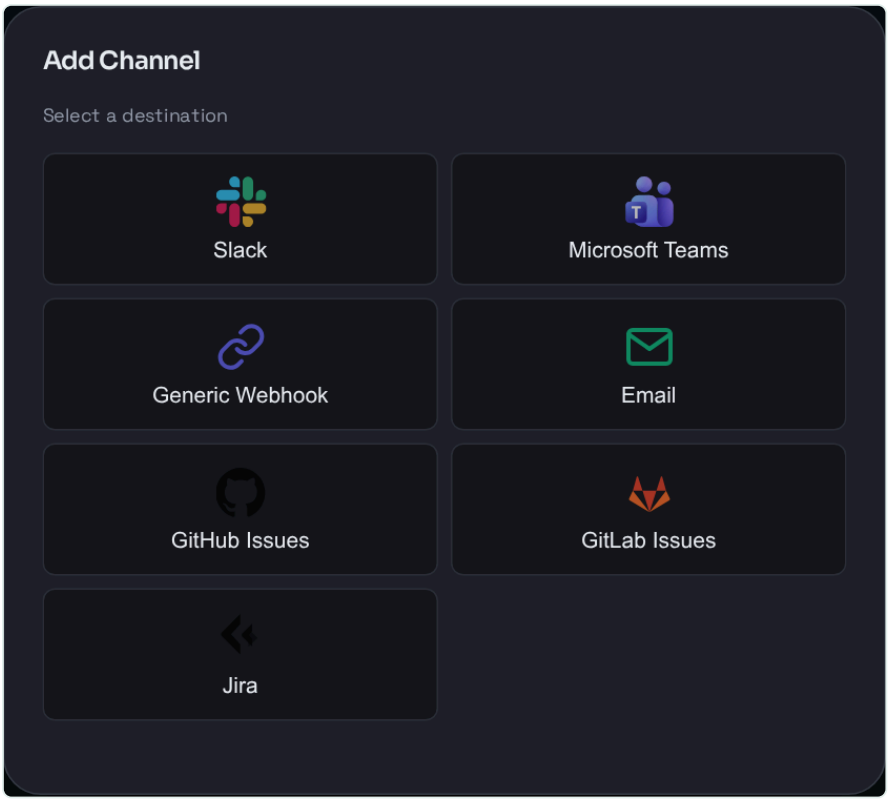
Supprimer un utilisateur désactive le compte au lieu de l'effacer. Un compte désactivé ne peut plus se connecter et ses sessions prennent fin. Son identifiant et ses lignes d'audit restent, si bien que chaque `user:<id>` d'un paquet déjà émis se résout encore. Pour une demande d'effacement, un admin peut ensuite *Anonymise* le compte désactivé. Cela vide son nom, son e-mail et son avatar et réécrit en `user:<id>` chaque copie de son nom dans le journal d'audit, les décisions de triage, les policies runtime et leurs révisions. Les paquets déjà signés gardent ce avec quoi ils ont été signés, et les commentaires déjà publiés dans un gestionnaire de tickets ne changent pas.



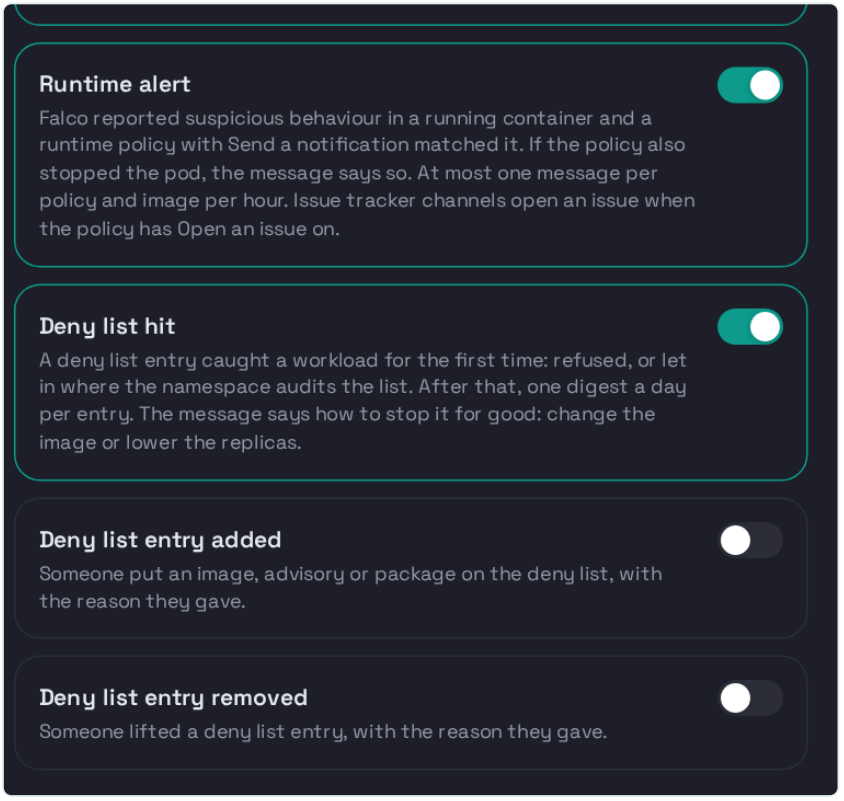
Un compte anonymisé reste dans la liste, désactivé, avec son identifiant de compte à la place de l'adresse e-mail. Le compte désactivé en dessous propose encore *Anonymise*.

Canaux de notification

Les canaux se configurent sous Settings > Webhooks, sur la page *Notification Channels*. Les préférences e-mail de chacun se trouvent sous Settings > My Notifications.



Add Channel. Un outil de suivi d'incidents est un canal comme les autres, avec les mêmes abonnements. *Test* sur un canal de ce type n'écrit rien. Il lit une fois le dépôt ou le projet et indique si l'identifiant fonctionne.



Une partie de la liste *Events* d'un canal. *Deny list hit* est activé, comme sur tout canal créé à partir de la 1.4.0 : un message la première fois qu'une entrée attrape un workload, puis un récapitulatif par jour et par entrée. *Deny list entry added* et *Deny list entry removed* sont désactivés par défaut. *Runtime alert* est décrit au chapitre 8.

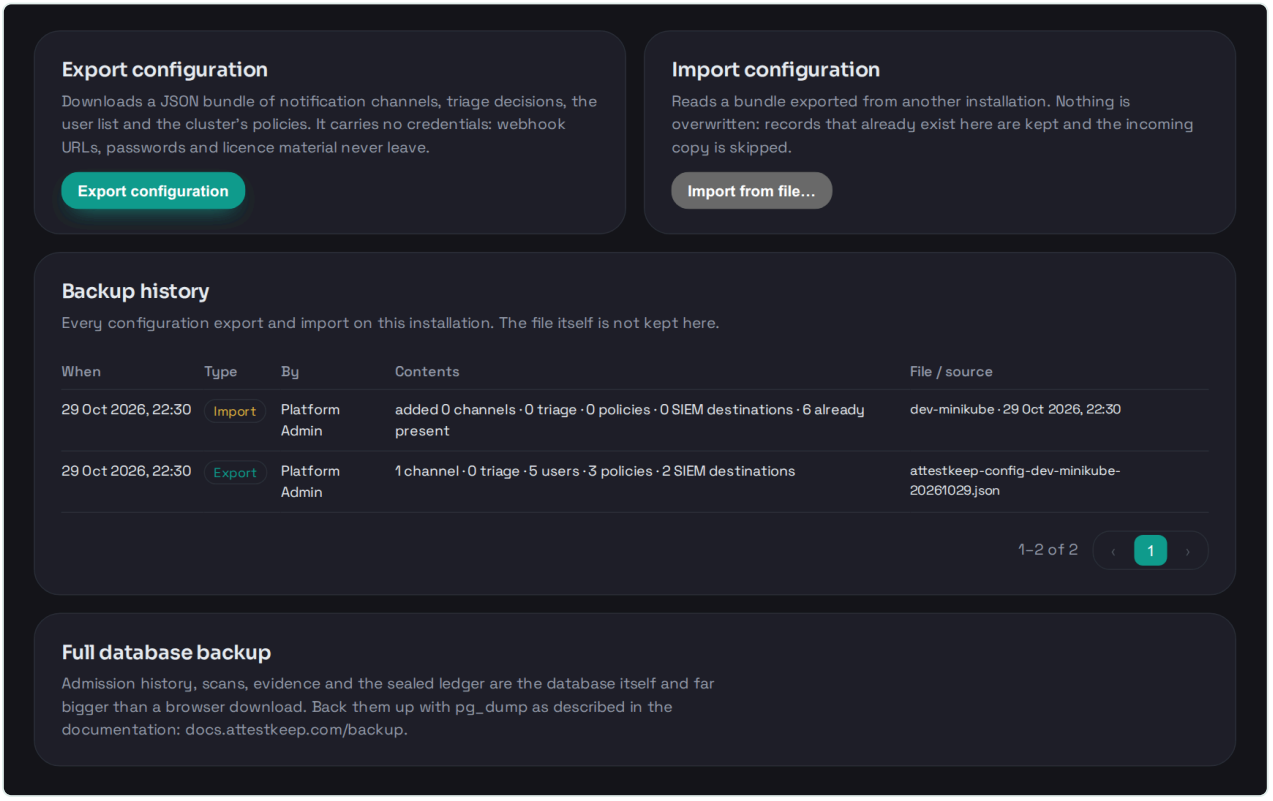
À partir de la 1.4.0, un canal peut être informé de la deny list. *Deny list hit* est envoyé la première fois qu'une entrée attrape un workload, avec l'entrée, sa raison et la marche à suivre, puis une fois par jour et par entrée sous forme de récapitulatif. Un nouveau canal l'a coché. Un canal créé avant la 1.4.0 garde ses abonnements. Un refus de la deny list

n'est pas envoyé en plus comme *Admission blocked*, qui sert aux refus par les règles propres d'une policy. *Deny list entry added* et *Deny list entry removed* sont désactivés par défaut.

Chaque envoi, par mail, webhook, Slack ou Teams, dispose de 30 secondes de la connexion au dernier octet. Un envoi qui dépasse ce délai figure dans le journal des notifications comme non livré avec "no answer within 30s". Au plus 64 envois tournent en même temps et au plus 64 notifications attendent. Au-delà, une notification est abandonnée, notée comme non livrée et comptée dans `attestkeep_notifications_dropped_total`.

Le token d'un gestionnaire de tickets est le seul identifiant conservé par Attestkeep qui peut écrire dans un système hors de votre cluster. L'API ne le renvoie jamais. Il est chiffré en AES-256-GCM avant stockage, avec une clé que le chart garde dans le Secret `<release>-data-key` et jamais dans la base. Utilisez `api.dataKey.existingSecret` pour fournir la clé depuis votre propre gestionnaire de secrets. Sans la clé, un canal de tickets ne peut pas être enregistré. Rien d'autre n'en dépend.

Backup history



Settings > Backup. *Export configuration* télécharge un bundle JSON des canaux de notification, des décisions de triage, de la liste des utilisateurs, des policies et, depuis la 1.4.0, des destinations SIEM, sans identifiants. *Backup history* liste chaque export et import, avec quand, par qui et ce que contenait le bundle. L'opérateur ne garde pas le fichier. La sauvegarde de la base de données vous revient (chapitre 11).

6. Les policies d'admission

Une seule ressource à l'échelle du cluster décide de ce qui peut tourner : `ImageSecurityPolicy` dans le groupe d'API `attestkeep.com`. La console la modifie directement, et `kubectl apply` la modifie depuis un dépôt. Les deux agissent sur le même objet. Une modification s'applique dès la décision d'admission suivante, sans redémarrage.

Comment un namespace obtient sa policy. S'applique la policy dont `spec.targetNamespaces` cite le namespace. Sinon s'applique la policy sans `targetNamespaces`, le défaut global. Un montage courant associe une policy globale et une policy plus stricte pour les namespaces qui en ont besoin.

Le défaut fourni

```
apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: default
spec:
  enforcement: Enforce          # Audit enregistre seulement ce qui aurait été refusé
  admissionTiming: AllowAndScan # DenyUntilScanned refuse tout ce qui n'a pas de scan
  blockLatestTag: true          # un tag mobile peut changer ce qui tourne sans déploiement
  digestEnforcement: AllowTags  # AllowTags | PreferDigest | Required
  denyList: "enforce"           # enforce | audit | "off" (chapitre 7)
  thresholds:
    blockCritical: true
    blockHigh: true
    maxMedium: 0                # 0 signifie ici illimité
    maxLow: 0
  slaWindows:
    critical: 24h
    high: 168h
    medium: 720h
    low: 2160h
  requireSignature: false       # à activer quand votre pipeline produit des signatures
  requireSBOM: false
  requireVulnAttestation: false
  attestationMaxAge: 168h
  triageDefaults:
    maxDuration: 2160h
    requireJustification: true
```

Une policy plus stricte pour les namespaces de production

```
apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: prod-hard-gate
spec:
  targetNamespaces: ["prod", "payments"]
  enforcement: Enforce
  admissionTiming: DenyUntilScanned
  blockLatestTag: true
  digestEnforcement: Required
  allowedRegistries:
    - registry.example.com
    - ghcr.io/your-org
  thresholds:
    blockCritical: true
    blockHigh: true
  requireSignature: true
  requireSBOM: true
  requireVulnAttestation: true
  attestationMaxAge: 168h
  workloadHardening:
    mode: enforce               # off | warn | enforce
    minSeverity: high
    exempt: []
```

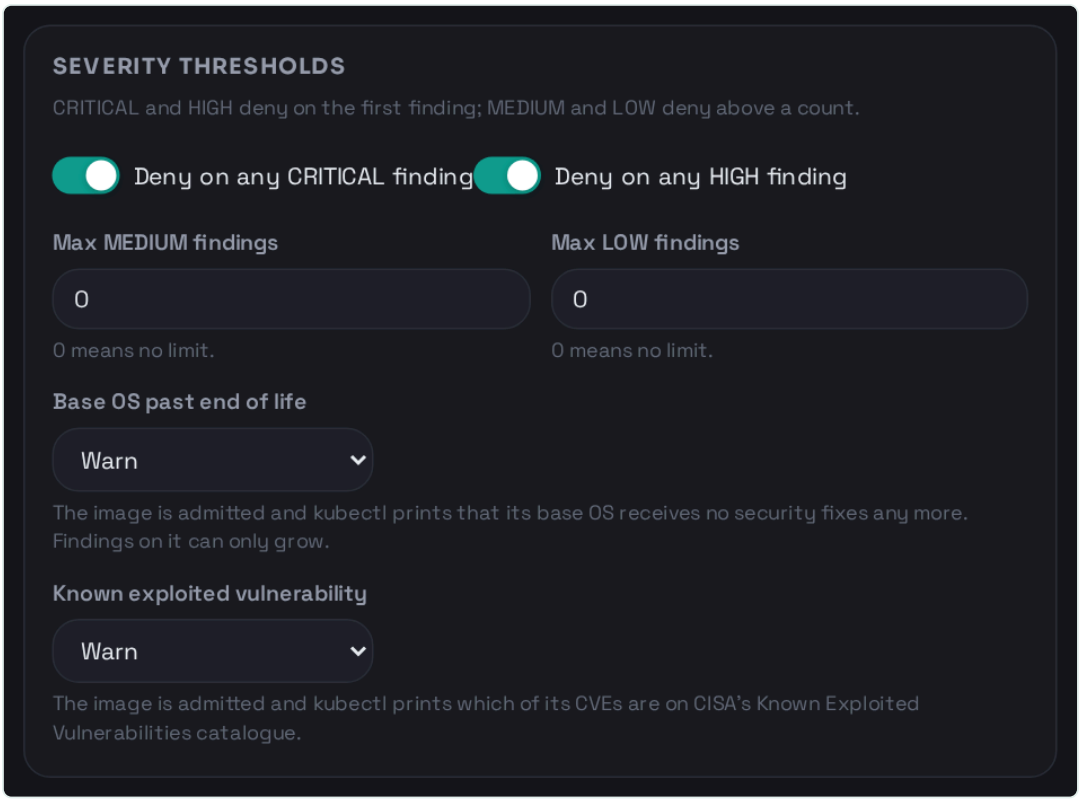
Système de base en fin de vie et CVE exploitées

Deux contrôles regardent autre chose qu'un nombre de vulnérabilités par gravité. Tous deux acceptent `ignore`, `warn` ou `block`, et tous deux valent `warn` par défaut. Une policy écrite avant leur arrivée continue d'admettre ce qu'elle admettait, et dit désormais aussi pourquoi elle ne devrait pas.

```
spec:
  # Le système de base est sorti de la période de support de son éditeur,
  # aucun correctif de sécurité ne sera plus publié. warn (défaut) admet
  # et le signale ; block refuse quels que soient les compteurs ; ignore
  # ne dit rien.
  endOfLifeOS: warn

  # L'image contient une CVE du catalogue Known Exploited Vulnerabilities
  # de la CISA. warn (défaut) admet et nomme les CVE ; block refuse ;
  # ignore ne dit rien.
  knownExploited: warn
```

Le catalogue KEV est conservé en copie locale et rafraîchi toutes les six heures. Un cluster sans accès à internet pointe `kev.source` vers un miroir, ou laisse la valeur vide pour garder la copie qu'il a. Les CVE traitées en triage sont exclues de ce contrôle, comme des seuils de gravité. Un refus nomme les CVE, si bien que la personne qui déploie sait quoi corriger. Le tableau de bord affiche la version du catalogue utilisée, et les paquets de preuves l'inscrivent à côté de la version de la base de vulnérabilités.



La section *Severity thresholds* de l'éditeur. *Base OS past end of life* et *Known exploited vulnerability* sont tous deux sur Warn, et le texte sous chacun dit ce que fait ce réglage.

Clés de signature

À partir de la 1.4.0, une policy peut dire, par clé, de qui elle accepte la signature. `signatureKeys` liste des clés publiques PEM, par exemple votre `cosign.pub`. Avec `requireSignature: true`, seule compte une signature vérifiée par l'une d'elles.

```
spec:
  requireSignature: true
  signatureKeys:
    - |
      -----BEGIN PUBLIC KEY-----
      MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...
      -----END PUBLIC KEY-----
```

Le scan enregistre le signataire sous l'empreinte de la clé, `key sha256:…`, et le paquet de preuves la cite. Avec `scan.cosignOffline`, la vérification n'a pas besoin du journal de transparence. Les attestations SBOM et de vulnérabilités restent en keyless et n'ont pas de champ de clé. L'API server refuse une entrée qui n'a pas la forme d'une clé publique PEM. Une entrée qui ne se lit toujours pas est ignorée et signalée à trois endroits : la condition de statut `SignatureKeysValid` de la policy, un événement Warning et la page des policies.

Provenance du build

Ce contrôle vérifie quel système de build a produit l'image et à partir de quelle source. C'est le seul contrôle lié à une identité. `requireSBOM` et `requireVulnAttestation` vérifient que quelqu'un a signé une attestation. Ce contrôle vérifie qui.

```
spec:
  provenance:
    # 0 (défaut, y compris quand le bloc est absent) désactive le contrôle.
    # Au-dessus de 0, l'image doit porter une provenance SLSA vérifiée qui
    # nomme un builder listé ci-dessous, crédité d'au moins ce niveau.
    minLevel: 2
    trustedBuilders:
      # Un "*" final compare par préfixe ; les autres id comparent à l'identique.
      # Une liste vide avec minLevel au-dessus de 0 n'admet rien.
      - id: https://github.com/acme/build/.github/workflows/release.yml@*
        # Votre déclaration sur ce builder. La provenance SLSA ne porte pas
        # de niveau, car un niveau décrit la façon dont le builder est exploité.
        level: 3
        # Lu dans le certificat de signature. S'il est absent, il doit égaler
        # l'id du builder. Si issuer est absent, tout émetteur est accepté.
        signer: https://github.com/acme/build/.github/workflows/release.yml@*
        issuer: https://token.actions.githubusercontent.com
```

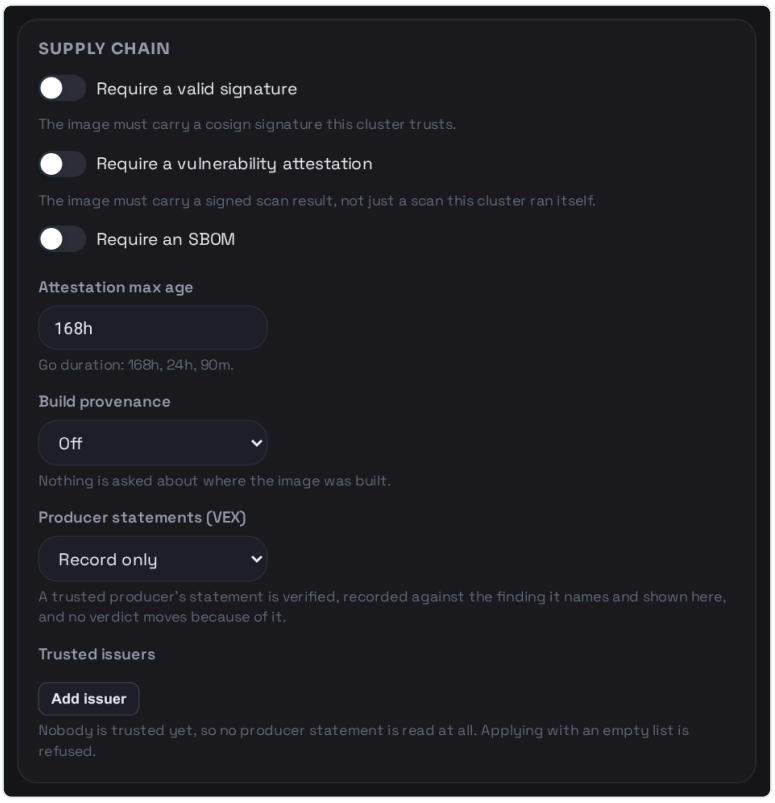
Une image est refusée si sa provenance SLSA vérifiée (v1 ou v0.2) ne nomme pas un builder de confiance, si l'identité ou l'émetteur du certificat de signature n'est pas celui que l'entrée accepte, ou si le niveau du builder est inférieur à `minLevel`. La raison nomme le builder, le signataire ou le niveau. Les deux identités sont lues dans le certificat conservé avec l'attestation. Si ce certificat ne peut pas être récupéré ou lu, l'image est refusée.

Déclarations du producteur (VEX)

Un scanner rapporte ce qui est installé dans une image. Il ne peut pas dire si le code vulnérable est atteignable ni s'il s'exécute un jour. Celui qui a construit l'image le peut, et OpenVEX est le format prévu pour cela. Attestkeep ne lit une déclaration VEX que si elle arrive sous forme d'attestation in-toto signée sur l'image et que la policy nomme son signataire. Les déclarations non signées, celles de signataires que la policy ne nomme pas et celles qui portent sur un autre produit sont enregistrées et ignorées.

```
spec:
  vex:
    # ignore (défaut) : les déclarations de confiance sont vérifiées,
    # notées sur la vulnérabilité et montrées dans la console ; aucun
    # verdict ne change.
    # apply : une déclaration de confiance not_affected ou fixed retire la
    # vulnérabilité des compteurs sur lesquels décide l'admission.
    mode: apply
    trustedIssuers:
      # Un signataire keyless, par identité de certificat. Un "*" final
      # compare par préfixe. issuer est facultatif et comparé à l'identique.
      - identity: https://github.com/acme/images/*
        issuer: https://token.actions.githubusercontent.com
      # Ou un signataire par clé, par clé publique. Une entrée a une
      # identité ou une clé, pas les deux.
      - key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...
          -----END PUBLIC KEY-----
```

`affected` et `under_investigation` ne suppriment jamais une vulnérabilité. Une vulnérabilité supprimée sort des compteurs qu'utilisent les seuils de gravité et la règle sur les CVE exploitées, et la raison de l'admission indique combien de vulnérabilités ont été supprimées et par qui. `mode: apply` avec une liste `trustedIssuers` vide est refusé, parce que cela ressemblerait à un contrôle sans rien faire. Un changement de mode s'applique au pod suivant. Un nouvel émetteur s'applique au scan suivant du digest.



La section *Supply chain*. Signature, attestation de vulnérabilités et SBOM ne sont pas exigés, *Build provenance* est sur Off et *Producer statements (VEX)* sur *Record only*, la valeur par défaut. Aucun émetteur n'est encore de confiance, donc aucune déclaration n'est lue.

Là où le pod va tourner

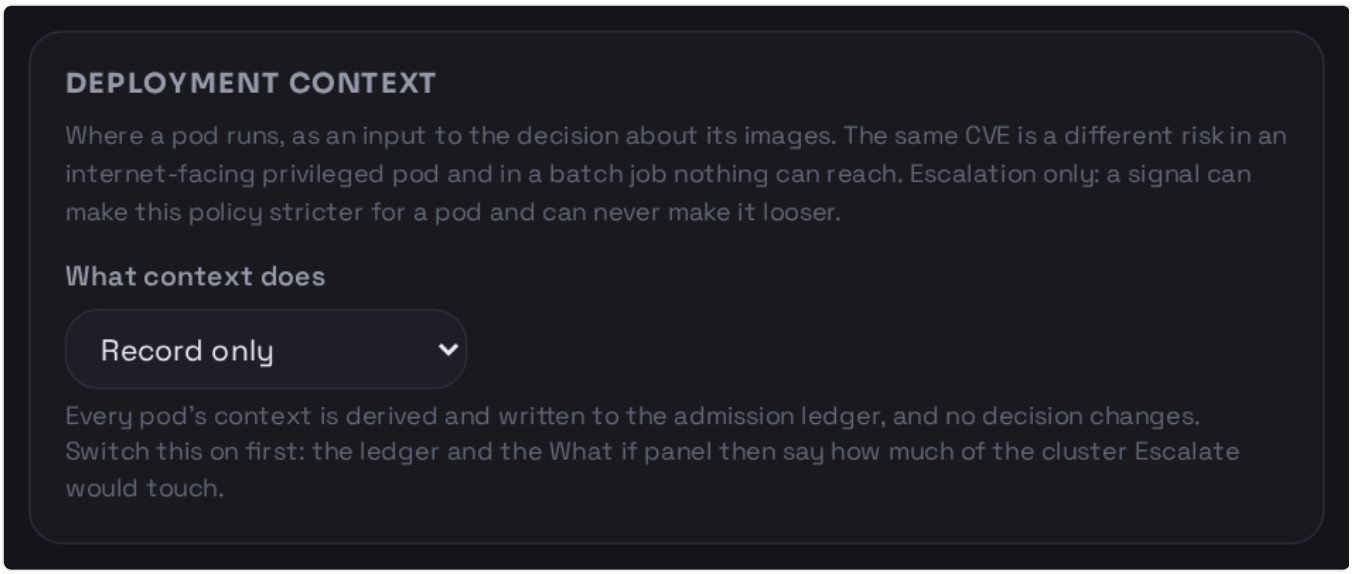
La même vulnérabilité HIGH pèse plus dans un pod privilégié exposé à internet que dans un batch que personne ne peut atteindre. `deploymentContext` permet à une policy de traiter ces pods plus sévèrement. Il ne fait que durcir une décision. Aucun signal, ni aucune combinaison de signaux, n'admet un pod que la policy refuserait autrement.

```
spec:
  deploymentContext:
    mode: Escalate      # Ignore (défaut) | Escalate
    signals:            # vide signifie les six
      - exposed
      - privileged
```

Six signaux sont calculés pour chaque pod et inscrits sur chaque ligne du registre, dans les deux modes : `privileged`, `root`, `hostNamespace`, `hostFilesystem`, `capabilities` et `exposed`. Seul `exposed` regarde au-delà du pod : hostNetwork, un hostPort, un Service NodePort ou LoadBalancer qui sélectionne le pod, ou un Service ClusterIP derrière un Ingress. Un Service ClusterIP sans Ingress ne compte pas comme une exposition. En `Escalate`, un pod qui porte un signal sélectionné est jugé ainsi :

La policy dit	Un pod avec un signal sélectionné est jugé comme
`thresholds.blockCritical: true`	bloquant aussi HIGH
`thresholds.blockHigh: true`	bloquant aussi MEDIUM
`knownExploited: warn`	`block`
`endOfLifeOS: warn`	`block`

`maxMedium` et `maxLow` ne changent pas, et `ignore` reste `ignore`. Le refus nomme le signal et, pour l'exposition, le Service, l'Ingress ou le hostPort qui l'a causée.



La section *Deployment context* sur *Record only*, c'est-à-dire `Ignore`. Le contexte de chaque pod est écrit dans le registre d'admission et aucune décision ne change. Le registre et le panneau What if montrent ensuite combien de pods `Escalate` toucherait.

Référence des champs

Champ	Valeurs	Ce qu'il décide
enforcement	Enforce | Audit	Refuser, ou seulement enregistrer ce qui aurait été refusé.
admissionTiming	AllowAndScan | DenyUntilScanned	Le sort d'une image qui n'a pas encore de scan.
blockLatestTag	bool	Refuser le tag `:latest`.
digestEnforcement	AllowTags | PreferDigest | Required	Si les images doivent être épinglées par digest.
allowedRegistries	liste d'hôtes	Vide, tout registry est accepté. Non vide, tous les autres sont refusés.
denyList	enforce | audit | "off"	Ce que fait une correspondance de deny list dans les namespaces de cette policy. Défaut `enforce`, qui refuse même sous `enforcement: Audit` (chapitre 7).
targetNamespaces	liste	Vide, c'est la policy globale. Non vide, elle se limite à ces namespaces.
thresholds	blockCritical, blockHigh, maxMedium, maxLow	Refuser à la moindre vulnérabilité critique ou haute. Fixer des plafonds pour medium et low (0 signifie illimité).
slaWindows	durées	Combien de temps une vulnérabilité de chaque gravité peut rester ouverte avant que les rapports la disent en retard.
requireSignature / requireSBOM / requireVulnAttestation	bool	Refuser les images sans signature cosign vérifiée, sans attestation SBOM ou sans attestation de scan de vulnérabilités.
signatureKeys	liste de clés publiques PEM	Avec `requireSignature`, seule une signature vérifiée par l'une de ces clés est acceptée.
attestationMaxAge	durée	Une attestation plus ancienne ne compte plus.
workloadHardening	mode, minSeverity, exempt	Si les containers privilégiés, les montages hostPath et réglages semblables refusent, avertissent ou passent.
endOfLifeOS	ignore | warn | block	Les images dont le système de base est hors support. Défaut `warn`.
knownExploited	ignore | warn | block	Les images qui portent une CVE du catalogue KEV de la CISA. Défaut `warn`.
provenance	minLevel 0 à 3, trustedBuilders (id, level, signer, issuer)	Refuser les images sans provenance vérifiée d'un builder de confiance au niveau demandé. Défaut `minLevel: 0`, désactivé.
vex	mode ignore | apply, trustedIssuers	Si la déclaration not_affected ou fixed d'un producteur de confiance retire une vulnérabilité des compteurs. Défaut `ignore`.
deploymentContext	mode Ignore | Escalate, signals	Juger plus sévèrement les pods qui portent un signal sélectionné. Défaut `Ignore`.
runtimeReconciliation	audit | notify | enforce	Ce que fait le rapprochement périodique d'une image en cours sans admission derrière elle.

breakGlass	enabled, reason, approvedBy, auditRef, expiresAt, affectedImages	Une exception nommée et approuvée, qui expire. Chaque usage est enregistré.
triageDefaults	maxDuration, requireJustification	Combien de temps un risque accepté peut tenir, et s'il exige une justification.
storageUnreachablePolicy	Allow | Deny	La réponse d'admission tant que la base est injoignable.

Ce que déclenche un événement Falco ne fait pas partie de la policy d'admission. Ce sont les policies runtime de la console qui le fixent (chapitre 8).

Break-glass

Quand la production est à l'arrêt et que la solution est une image que la policy refuse, le break-glass laisse entrer les images nommées, enregistre chaque usage et expire tout seul :

```
breakGlass:
  enabled: true
  reason: "INC-2417 : l'image de retour arrière porte une vulnérabilité high connue"
  approvedBy: "astreinte, responsable d'incident"
  auditRef: "INC-2417"
  expiresAt: "2026-09-01T06:00:00Z"
  affectedImages:
    - registry.example.com/payments/api:1.41.2
```

Chaque admission passée uniquement grâce au break-glass est marquée comme telle dans la piste d'audit et dans les rapports de conformité.

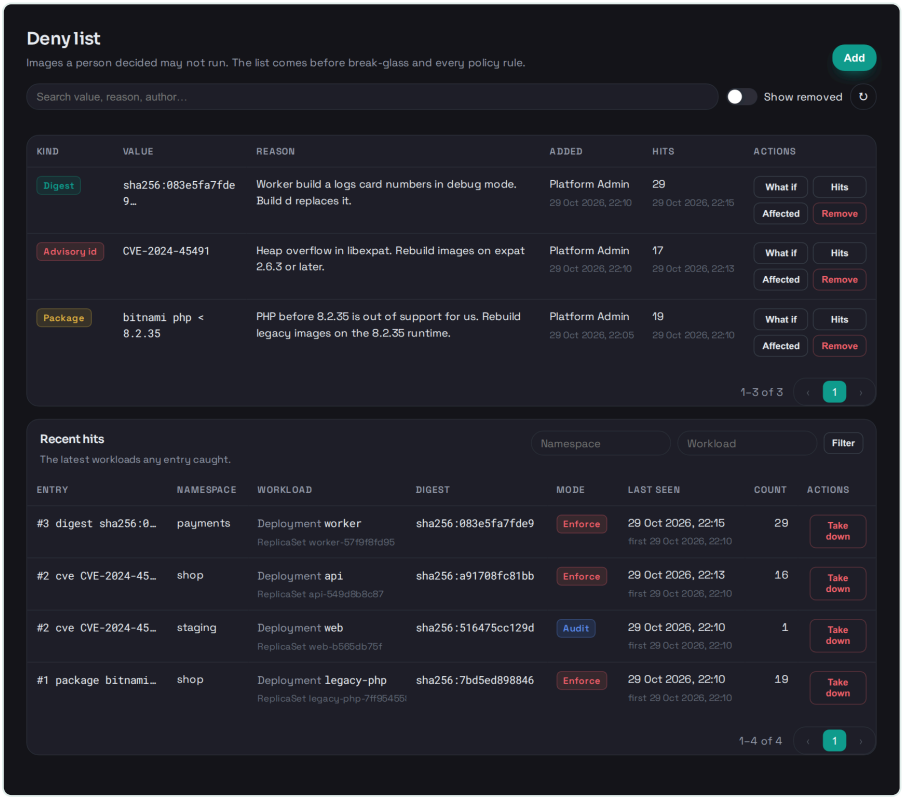
Les images que couvre le break-glass sont admises sans les règles d'image de la policy : la liste des registries autorisés, les règles sur `:latest` et l'épinglage par digest, le résultat du scan et ses règles, les signatures, les attestations et la provenance du build, le quota d'images et le plafond de nodes. La deny list et le durcissement des workloads s'appliquent toujours. `affectedImages` se compare au nom de l'image, si bien que `nginx`, `docker.io/library/nginx` et `nginx:1.27` désignent une même image, et un tag dans la liste ne la restreint pas. Une liste vide couvre toutes les images. Avant la 1.4.0, la liste n'était pas enregistrée à l'ouverture du break-glass, et une fenêtre limitée à des images nommées couvrait donc toutes les images de la policy.

7. La deny list

Une policy répond automatiquement à une question : cette image respecte-t-elle les règles ? Une entrée de deny list répond à une autre : quelqu'un a-t-il décidé que cette image ne doit pas tourner ? La deny list est l'endroit où cette décision est écrite, et l'admission la consulte avant tout le reste.

Il y a une liste par cluster, sous Security > Deny list. Le champ `denyList` de chaque policy décide de ce que ses namespaces font d'une correspondance. Une entrée n'est jamais supprimée. La retirer la lève, et la ligne reste avec qui l'a retirée, quand et pourquoi. Ajouter et retirer sont des actions d'administrateur, et les deux exigent une raison.

> **Une liste vide ne change rien.** Sans entrée active, l'admission ne résout aucun digest de plus et ne refuse rien. Passer en 1.4.0 ne change aucune décision tant que personne n'ajoute d'entrée.



Security > Deny list avec trois entrées : un digest, l'avis CVE-2024-45491 et le paquet `bitnami php < 8.2.35`. Chaque ligne donne la raison, qui l'a ajoutée et quand, et combien de hits elle a eus. *Recent hits* nomme ensuite les workloads attrapés, par namespace et par digest, et indique si la policy du namespace appliquait la liste ou se contentait de l'auditer, comme dans `staging`.

Types d'entrée

Type	Correspond à	Exemple
Digest	Un build d'image précis, comparé au digest que l'admission a résolu pour le pod et à l'index d'image ou aux manifestes de plateforme auxquels on sait qu'il appartient.	`sha256:9f2a…d4e5`
Identifiant d'avis	Toute image dont le dernier scan rapporte l'avis, qu'un correctif existe ou non.	`CVE-2021-44228`, `GHSA-jfh8-c2jp-5v3q`
Paquet	Toute image dont le dernier scan liste le paquet dans la plage de versions.	`maven org.apache.logging.log4j:log4j-core < 2.17.1`
Hash d'exécutable	Toute image qui contient un fichier, par le sha256 de ses octets, quels que soient son nom et son chemin.	un sha256
Chemin d'exécutable	Toute image qui a un exécutable à un chemin, ou un fichier de ce nom n'importe où. Une règle de chemin touche aussi les images de base.	`/bin/sh`, `/usr/local/bin/*`, `nc`
Paquet, écosystème `bitnami`	Toute image avec un runtime dont la version, lue dans l'exécutable lui-même, est dans la plage. Ne correspond jamais à un paquet de distribution.	`bitnami php < 8.2.35`

Une entrée d'avis accepte les schémas sous lesquels les scanners rapportent leurs résultats, car pour certains d'entre eux cet identifiant est le seul nom : CVE, GHSA, GO, PYSEC, RUSTSEC, OSV, MAL, GSD, DLA, DSA, ELA, USN, RHSA, RHBA, ALSA, ALBA, ELSA, CGA, BIT, TEMP, SUSE-SU et openSUSE-SU. Tout le reste est refusé à l'ajout, puisqu'un identifiant qu'aucun scanner ne rapporte ne correspondrait à rien.

Une entrée de paquet nomme un écosystème, un paquet et un opérateur : en dessous, jusqu'à, exactement, à partir de, au-dessus, entre (borne basse incluse, borne haute exclue) ou toute version. Deux plages font deux entrées. Les versions sont comparées avec le comparateur que le scanner lui-même utilise pour l'écosystème : npm, Go, Cargo, Composer, NuGet, Maven, PyPI, RubyGems, deb, apk et rpm. Une version que le comparateur ne sait pas lire est refusée à l'ajout, plutôt que comparée comme du texte, où 10.0 passerait avant 9.0. Une entrée avec *au-dessus* ou *à partir de* couvre aussi chaque version publiée après son ajout, et What if le dit.

Sa place dans la décision

L'ordre est la deny list, puis le break-glass, puis les règles de la policy. Une image listée est refusée même si son scan est propre, et même pendant un break-glass : l'entrée est une décision délibérée, et elle passe avant la porte de secours. Le refus se lit ainsi :

```
denied by deny list entry 7: package maven org.apache.logging.log4j:log4j-core < 2.17.1
```

Les règles qui n'exigent aucun appel au registry, la liste des registries autorisés, `blockLatestTag` et l'exigence de digest, passent d'abord. Sous une policy en Enforce, leur refus suffit. Sous une policy en Audit, la revue continue jusqu'à la deny list, et une image listée y est donc refusée aussi.

`denyList` sur la policy	Effet d'une correspondance dans ses namespaces
`enforce` (défaut)	Refuse le pod, même si la policy est en mode Audit. Un champ vide ou absent vaut `enforce`.
`audit`	Admet le pod avec l'avertissement *would be denied by deny list entry …* et enregistre la correspondance.
`"off"`	La liste n'est pas consultée.

Écrivez `off` entre guillemets en YAML : un `off` nu est un booléen pour un lecteur YAML 1.1. Après la mise à niveau, chaque policy enregistre une nouvelle révision, parce que ce champ fait partie du contenu que le registre hache.

Les requêtes qu'elle examine. Chaque container, init container et ephemeral container d'un nouveau pod. Pour `kubectl debug`, seulement le container que la requête ajoute. Les autres mises à jour d'un pod existant, un label, une annotation ou un finalizer, ne sont ni refusées ni comptées. Un dry run reçoit la réponse d'une vraie requête et n'enregistre aucune correspondance.

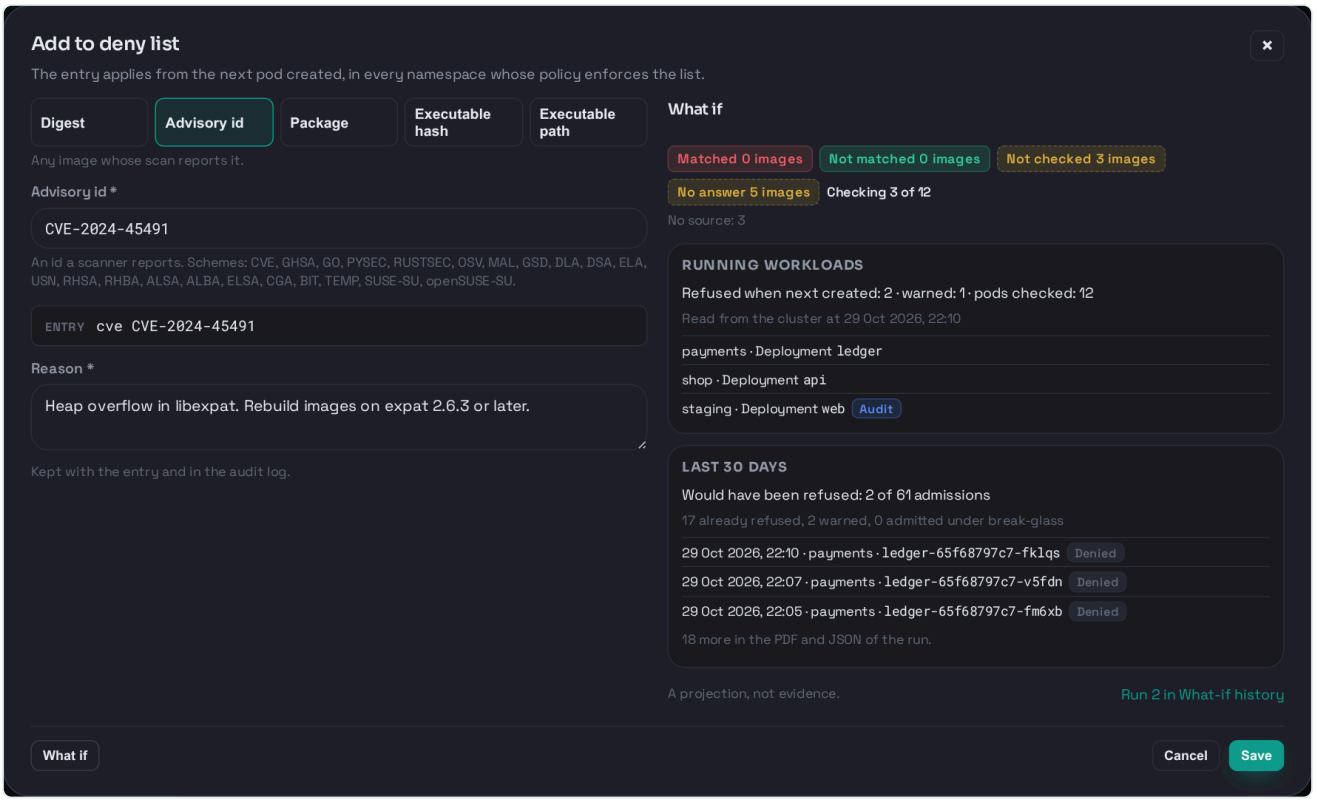
Rien de ce qui tourne n'est arrêté. Une entrée refuse un workload la prochaine fois qu'un de ses pods est créé. Le contrôleur du workload continue d'essayer, à son propre rythme, jusqu'à ce que son image change.

Quand un digest ne se résout pas. Tant que la liste contient une entrée, l'admission résout le digest du pod avant le break-glass. Si le registry ne répond pas, un pod sous une policy en Enforce avec `denyList: enforce` est refusé avec *deny list could not be evaluated: digest unresolved*. Sous une policy en Audit, ou avec `denyList: audit`, il est admis avec cet avertissement. Une image que seuls les nodes peuvent tirer, avec des identifiants de node plutôt qu'un pull secret référencé par le pod ou son service account, ne peut pas être résolue par l'operator et est refusée dans les namespaces en Enforce tant que la liste n'est pas vide. Épinglez-la par digest, ou référencez un pull secret depuis le pod. Si la base ne peut pas être lue, la liste n'est pas appliquée et le pod porte l'avertissement *deny list unavailable*.

What if, avant d'ajouter l'entrée

What if essaie une entrée avant que quiconque le découvre à l'admission, dans la fenêtre Add avant l'enregistrement ou depuis la ligne d'une entrée. Il répond à deux questions côte à côte.

- **Les workloads en cours qu'elle couvre**, lus dans le cluster à ce moment et regroupés par le workload auquel chaque pod appartient : le Deployment au-dessus de son ReplicaSet, le CronJob au-dessus de son Job, un StatefulSet, un DaemonSet ou le pod lui-même. Sur un grand cluster, la lecture s'arrête après 20 secondes, 20 000 pods ou 2 000 images distinctes, et indique alors qu'elle est partielle et combien de pods elle a examinés.
- **Les admissions passées qu'elle aurait refusées**, sur les 30 derniers jours par défaut et jusqu'à 90 : les lignes du registre décidées de nouveau sous la policy actuelle de chaque namespace, avec cette entrée pour toute liste. Les admissions passées sous break-glass sont jugées aussi, puisque le break-glass ne contourne jamais la deny list.



What if dans la boîte d'ajout, avant l'enregistrement de l'entrée. Parmi les workloads en cours, 2 seraient refusés à leur prochaine création et 1 averti, là où `staging` audite la liste. Sur les 30 derniers jours, 2 admissions sur 61 auraient été refusées, en plus de 17 déjà refusées et 2 averties. La vérification des exécutables tournait encore, 3 images sur 12, et les pastilles indiquent combien d'images n'avaient pas encore de réponse. L'exécution est gardée comme exécution 2 dans What-if history.

Rien n'est arrêté et rien n'est enregistré. Une exécution lancée par un deployer ou un administrateur est conservée dans Security > What-if history avec le type *Deny list*, quatre-vingt-dix jours comme toute exécution et au plus les cinq cents dernières exécutions de deny list, comptées à part des exécutions de policy. Elle se télécharge en PDF ou en JSON comme projection, pas comme preuve, et n'entre jamais dans un paquet de preuves.

Correspondances

Chaque refus et chaque avertissement d'audit causé par une entrée est compté contre le workload : une ligne par entrée, namespace, workload et digest, avec la première et la dernière fois où il a été vu et le nombre de fois. Les pods d'un Deployment comptent pour le Deployment et ceux d'un CronJob pour le CronJob, si bien qu'un rollout ne crée pas de nouvelle ligne. La page de la deny list montre sous la liste les dernières correspondances de toutes les entrées, et *Hits* sur une ligne montre celles de cette entrée, filtrables par namespace ou par workload. La ligne de registre de chaque refus nomme l'entrée qui l'a décidé.

Exécutables : ce qu'un scanner ne liste pas

Trivy lit ce que les gestionnaires de paquets ont enregistré. Un runtime compilé depuis les sources et copié dans une image n'a pas de telle trace, et Trivy ne liste donc aucun paquet pour lui. Un PHP 8.2.12 compilé depuis l'archive de php.net, dans une image sans labels, montre 91 paquets Debian et aucun php. Les entrées d'exécutables et l'écosystème `bitnami` reçoivent leur réponse d'un second outil de l'image de l'operator, `attestkeep-binscan`, qui lit les exécutables de chaque image. Le scanner le lance comme il lance Trivy.

- **Seulement quand il le faut.** Il ne tourne que si la liste contient une entrée à laquelle il répond : un hash ou un chemin d'exécutable, un paquet `bitnami` ou un identifiant d'avis. Sans elle, il ne tourne jamais et aucune image n'est tirée une seconde fois.
- **Seulement des octets.** L'image est tirée par digest avec les pull secrets du workload lui-même, ses couches sont lues de haut en bas en retirant les fichiers supprimés, et chaque fichier ELF est haché. Le nom de l'image, ses labels, son environnement et son image de base ne sont jamais lus.

- **La version vient du fichier.** La version d'un runtime est lue dans les données en lecture seule de l'exécutable et confirmée par une seconde chaîne du même fichier. Les produits sont php, node, python, ruby, nginx, apache, postgresql, curl et openssl. Un build passé par `strip` est lu quand même : `strip` retire les symboles, pas les données en lecture seule.
- **Trois réponses.** Correspondance, avec le fichier, son sha256, la version et la signature qui l'a trouvée. Pas de correspondance. Ou non vérifié, avec la raison : identifiants, registry, image disparue, limite, erreur d'outil ou pas de source. Non vérifié ne compte jamais comme propre.

Limites de parcours. Un fichier ELF de plus de 256 MiB est haché mais pas lu pour sa version, au plus 4 000 fichiers ELF sont lus par image, et un parcours s'arrête après 60 secondes. Ces limites sont la valeur de chart `binscanLimits`. Un contrôle complet, téléchargement, parcours et comparaison des versions, a son propre délai, `scan.binscanTimeout` (10 minutes). Avant de télécharger, le contrôle additionne la taille des couches de l'image, et une image plus grosse que son budget disque n'est pas téléchargée. Le budget est la limite de `/tmp` du scanner, `scan.tmpSizeLimit` (4Gi), partagée entre ses workers, ou `scan.binscanDiskBudget` s'il est renseigné. Une réponse arrêtée par une limite ou le délai se lit *not checked (limit)* et la nomme. Un administrateur peut lancer *Check again* sur une image qui n'a pas pu être vérifiée, avec les limites en vigueur.

Quand il tourne. Une nouvelle entrée est vérifiée d'abord contre les images des pods en cours, puis contre les digests scannés sur les 90 derniers jours. Une nouvelle image est vérifiée lors de son scan, et les images en cours qu'une entrée n'a pas encore atteintes sont vérifiées toutes les heures. Un pod dont l'image n'a pas encore de réponse pour une entrée d'exécutable ou `bitnami` est jugé comme une image non scannée : `DenyUntilScanned` refuse, `AllowAndScan` admet avec un avertissement. Une entrée d'avis n'attend pas : les résultats de Trivy décident aussitôt, et une correspondance que le contrôle trouve plus tard ne fait que s'y ajouter.

Les identifiants d'avis atteignent aussi ces runtimes. Avec une entrée d'avis sur la liste, les runtimes trouvés sont comparés aux plages de vulnérabilités Bitnami de la base Trivy que l'operator tient déjà. Un runtime que ces plages ne couvrent pas, comme curl ou openssl, se lit "no vulnerability data in this source", jamais "not matched".



Deny list checks dans le détail de `php:8.2.12-cli`, à partir d'une liste de 905 exécutables. L'entrée 1, `bitnami php < 8.2.35`, correspond à `/usr/local/bin/php` et à trois autres fichiers en version 8.2.12. L'entrée 2, un identifiant d'avis, affiche *Not checked* : la source n'a pas de données de vulnérabilité pour le curl et l'openssl trouvés.

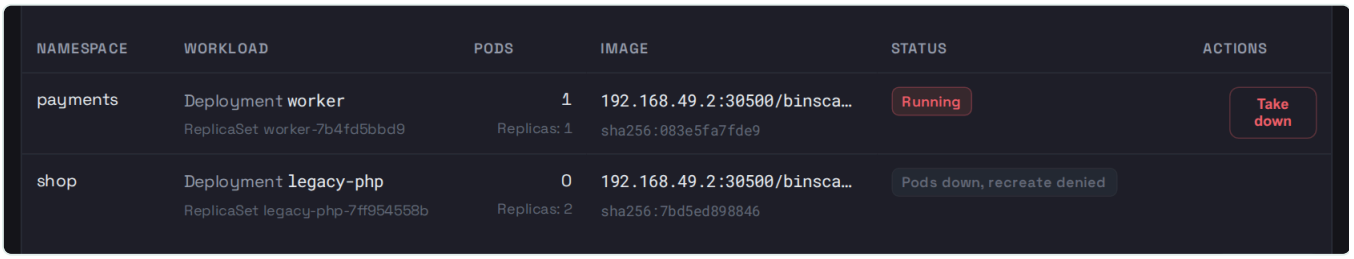
Ce qu'il ne voit pas : un exécutable dont la chaîne de version a été retirée ou modifiée, un exécutable sans en-têtes de section, un exécutable lié statiquement et passé par strip qu'on ne reconnaîtrait qu'à un nom de symbole, les exécutables compressés, les runtimes téléchargés au démarrage du container, les versions écrites hors des exécutables, le code applicatif interprété et les images qui ne peuvent pas être tirées. L'absence d'en-têtes de section, un exécutable compressé et une image impossible à tirer apparaissent comme non vérifiés, avec la cause.

Workloads concernés et take down

Une entrée refuse les nouveaux pods. Elle n'arrête jamais ce qui tourne déjà, et aucun mode de policy ne le fait de lui-même. *Affected* sur la ligne d'une entrée liste, en direct depuis le cluster, les workloads qu'elle attrape maintenant et ceux qu'elle a refusés, chacun avec un statut :

- **Running** : des pods qui exécutent une image listée sont en marche.
- **Pods down, recreate denied** : aucun ne tourne, et les nouveaux pods du contrôleur sont refusés.

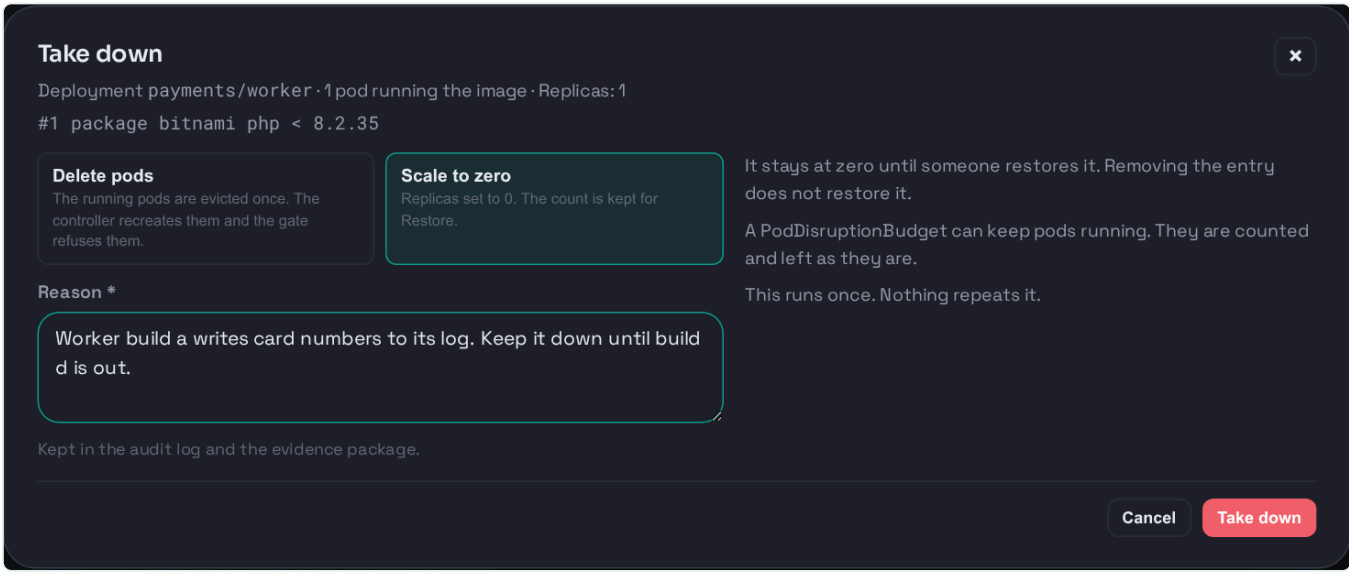
- **Scaled to zero** : un take down l'a ramené à zéro réplica et il existe toujours.
- **Restored externally** : il a été ramené à zéro ici et redemande des réplicas. Quelqu'un, ou un outil GitOps, les a remis. Il est affiché et laissé tel quel.



Affected, lu en direct dans le cluster. Le Deployment `worker` de `payments` exécute encore une image listée, d'où le bouton *Take down*. `legacy-php` dans `shop` n'a plus de pod actif et ses nouveaux pods sont refusés.

Avec `rbac.denyList.takedown: true` dans le chart, un administrateur peut faire un *Take down* d'un workload en cours, avec une raison. Cela agit une fois, et rien ne le répète. Le take down est enregistré avant que le cluster soit touché, et son issue après : fait, bloqué par un budget, ou échoué avec l'erreur.

- **Delete pods** (le défaut) évince une fois les pods qui exécutent une image listée. L'état souhaité n'est pas touché. Le contrôleur crée des remplaçants, l'admission les refuse tant que l'entrée tient, et le workload apparaît dégradé jusqu'à ce que son image change ou que l'entrée soit retirée. Il se rétablit alors tout seul.
- **Scale to zero**, pour un Deployment, un StatefulSet ou un ReplicaSet, met les réplicas à 0 par la sous-ressource scale et garde le nombre précédent. Il refuse si ce nombre n'est plus celui qu'affichait la fenêtre. **Restore** remet le nombre. Retirer l'entrée ne le fait pas : rien ne change l'échelle d'un workload sans un clic. Un DaemonSet, un Job, un CronJob ou un pod isolé est mis à l'arrêt en supprimant ses pods.

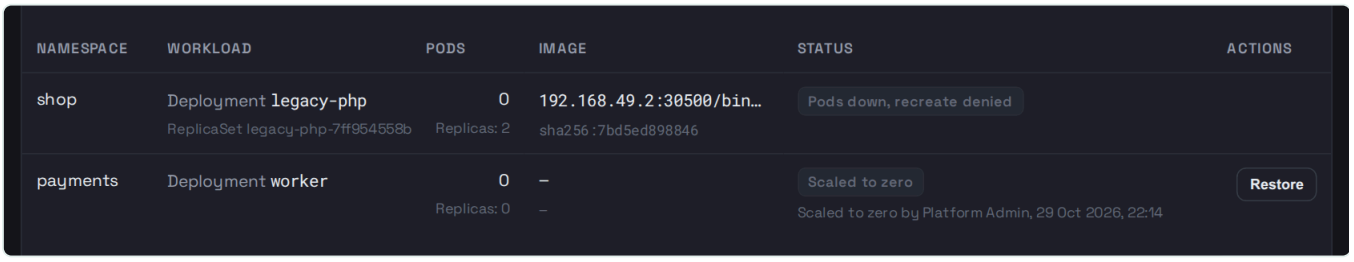


Take down du Deployment `payments/worker`, un pod en cours. *Scale to zero* est choisi, le nombre de réplicas est donc gardé pour Restore. Les notes disent que retirer l'entrée ne restaure rien, qu'un PodDisruptionBudget peut garder des pods actifs, et que l'action s'exécute une fois sans jamais se répéter. La raison est gardée dans le journal d'audit et le paquet de preuves.

Les PodDisruptionBudgets sont respectés. Les pods sont évincés par l'API d'éviction, et un pod qu'un budget protège continue de tourner. La ligne affiche alors "Blocked by PodDisruptionBudget" avec le nombre de pods conservés, et rien ne force ni ne réessaie.

GitOps. Un workload suivi par Argo CD ou Flux est signalé dans le panneau, et la fenêtre prévient que l'outil remettra un scale to zero. Quand il le fait, la ligne affiche *Restored externally* et Attestkeep ne le réduit plus jamais. La vraie correction se fait à la source : changer l'image, ou baisser les réplicas.

Sans `rbac.denyList.takedown: true`, le bouton apparaît désactivé et l'API répond `409`. Un namespace que l'admission n'examine pas, celui de l'operator ou un namespace que le webhook contourne, n'est jamais mis à l'arrêt : ses pods recréés ne seraient pas contrôlés, et un take down ne ferait que relancer la même image.



Après le take down, la ligne affiche *Scaled to zero*, avec qui l'a fait et quand, et propose *Restore*. Restore remet le nombre de réplicas gardé. Rien d'autre ne fait remonter le workload.

Notifications, événements et preuve

Un refus laisse un événement Kubernetes Warning, de raison `DenyListRefused`, qui nomme l'entrée, sur le workload, si bien que `kubectl describe` dit pourquoi ses pods ne démarrent pas. Pour un pod sans propriétaire, il va sur le namespace. Il y en a au plus un par workload et par entrée toutes les cinq minutes. L'ImageSecurityReport d'une image refusée porte `status.denyList` avec l'entrée et le refus. Les canaux abonnés à *Deny list hit* sont prévenus comme le décrit le chapitre 5.

Le paquet de preuves a une section *Deny list* : les entrées en vigueur à la fin de la période et celles ajoutées ou retirées pendant, avec qui, quand et pourquoi ; les admissions que les entrées ont refusées et laissées passer ; les exceptions, c'est-à-dire les correspondances admises parce qu'un namespace audite la liste ; chaque take down et chaque rétablissement avec sa raison ; et pour une entrée d'exécutable, le fichier, son hash et la version. Le hash de la policy ne couvre pas la liste. L'entrée qui a décidé une admission est nommée dans l'enregistrement de cette admission.

Qui peut faire quoi

Lire la liste, ses correspondances et ses workloads concernés, et lancer What if, demandent la même autorité que lire une policy. Ajouter et retirer une entrée, mettre un workload à l'arrêt et le rétablir sont des actions d'administrateur, et toutes sauf le rétablissement exigent une raison. Check again est aussi une action d'administrateur. Le journal d'audit les classe sous `deny_list_add`, `deny_list_remove`, `deny_list_takedown`, `deny_list_restore` et `deny_list_check_again`. L'API se trouve sous `/api/v1/deny-list`. Un champ que le serveur refuse revient en `400` avec le nom du champ, et une seconde entrée active de même valeur donne `409`.

8. Les événements runtime de Falco

Falco observe ce que font les containers après leur démarrage. Attestkeep peut recevoir les événements de Falco, rattacher chacun à l'image et au pod qui l'ont produit, et y réagir par des policies runtime que vous écrivez dans la console.

Attestkeep ne fournit pas Falco, ne l'installe pas, ne le configure pas et n'y écrit pas. Votre Falco et ses règles font la détection. Un événement runtime ne change jamais une décision d'admission. Une policy runtime peut notifier, ouvrir un ticket et arrêter le pod d'où vient l'événement.

Comment cela fonctionne

1. Une règle Falco se déclenche sur un node, par exemple quand quelqu'un ouvre un shell interactif dans un container.
2. L'événement part en JSON vers le port d'intégration de l'operator, en TLS mutuel. Falco l'envoie depuis son propre `http_output`, ou Falcosidekick le relaie depuis sa sortie webhook. Dans les deux cas, l'émetteur présente un certificat client émis par l'operator.
3. L'operator enregistre l'événement et le rattache à l'image et au digest qu'exécute le container.
4. Les policies runtime sont évaluées dans l'ordre. La première policy active qui correspond décide. Si aucune ne correspond, l'événement est gardé comme trace et rien n'est envoyé.
5. L'événement et la décision apparaissent sous Falco > Events, à côté de l'image sur la page Images, et dans le paquet de preuves.

Prérequis

- Falco installé avec le chart Helm amont `falcosecurity/falco`. Les valeurs documentées visent le chart 9.2.0 (Falco 0.45.0, Falcosidekick 2.32.0). Les versions plus anciennes n'ont pas été vérifiées.
- `falco.json_output: true`.
- Un accès réseau des pods Falco ou Falcosidekick vers le Service de l'operator sur le port d'intégration, 8444 par défaut.
- Un compte admin. Seuls les admins voient la page d'intégration, téléchargent les certificats et modifient les policies.
- Pour une policy qui arrête des pods : `rbac.allowEnforce: true` dans le chart Attestkeep. `rbac.denyList.takedown` ne compte pas pour cela.
- Pour les messages et les tickets : un canal de notification, ou un canal GitHub, GitLab ou Jira, abonné à Runtime alert.

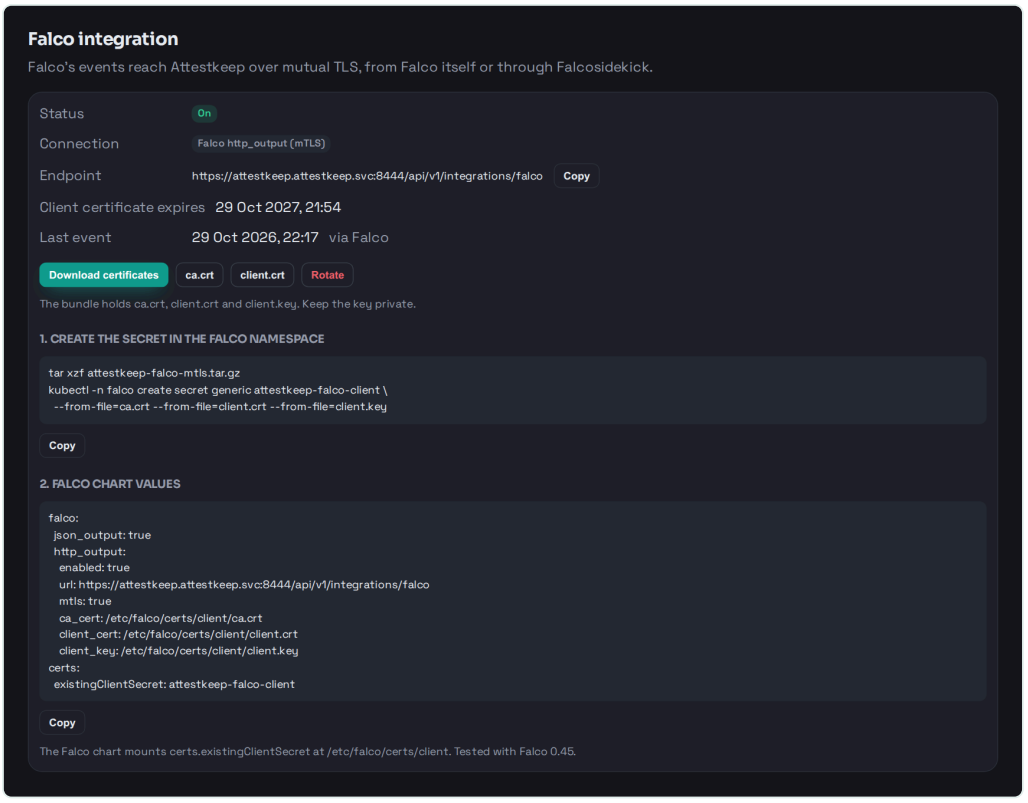
Connecter Falco

Activez l'intégration dans le chart Attestkeep et choisissez comment Falco se connecte :

```
integrations:
  port: 8444
  falco:
    enabled: true
    connection: mtls   # ou sidekick
```

Avec `mtls`, le défaut, Falco poste chaque événement depuis son propre `http_output` directement vers l'operator. Choisissez `sidekick` si votre Falco envoie déjà à Falcosidekick. La sortie webhook de Sidekick relaie alors vers l'operator avec le même certificat client. Toute autre valeur fait échouer le chart.

Quand l'intégration est active, l'operator crée sa propre petite CA et un certificat client (`CN=falco`, valable un an) et les garde dans le Secret `<release>-falco-mtls`. Le port d'intégration sert une seule route, `POST /api/v1/integrations/falco`, et n'accepte qu'un certificat client signé par cette CA. Il n'y a pas de token.

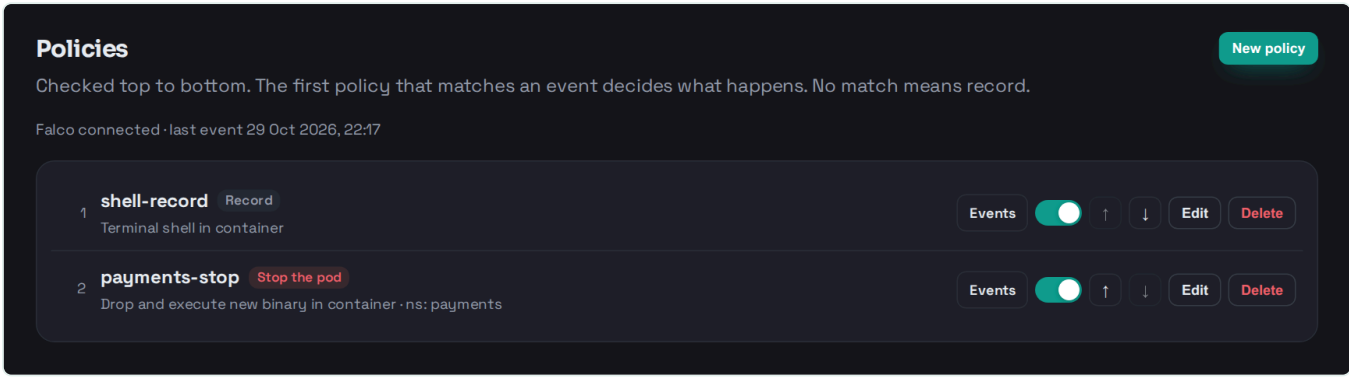


Falco > Falco integration sur l'installation de ce manuel. Le statut est On, la connexion passe par le `http_output` de Falco lui-même en mTLS, et le dernier événement est arrivé via Falco. *Download certificates* enregistre un bundle avec `ca.crt`, `client.crt` et `client.key`. En dessous, la commande qui crée le Secret dans le namespace de Falco et les valeurs du chart Falco pour la connexion choisie.

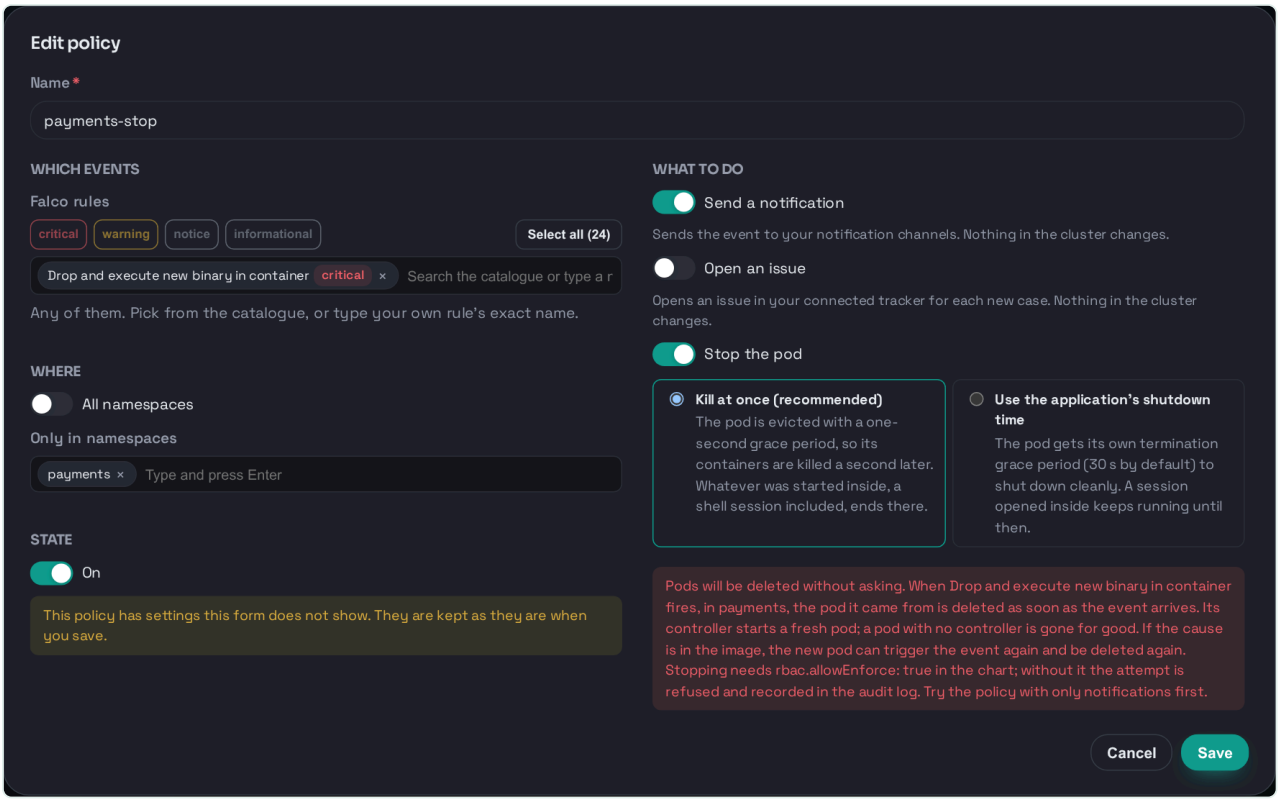
Chaque téléchargement du lot de certificats entre dans le journal d'audit. `client.key` doit rester privé, car quiconque le détient peut poster des événements auxquels vos policies réagiront. *Rotate* émet un nouveau certificat client depuis la même CA. L'ancien reste valable jusqu'à son expiration, et les événements continuent d'arriver pendant que vous remplacez le Secret. Quand le certificat serveur de l'operator est renouvelé, la page affiche "Certificates changed on <date>" jusqu'à ce que vous téléchargiez de nouveau le lot.

Les policies runtime

Falco > Policies liste vos policies runtime dans l'ordre. Chaque événement enregistré est comparé à elles en partant du haut, et la première policy active qui correspond décide. Les flèches changent l'ordre. Un changement s'applique tout de suite sur le réplica qui l'a enregistré, et en moins de 15 secondes sur les autres.



Deux policies runtime, vérifiées de haut en bas. `shell-record` enregistre *Terminal shell in container*. `payments-stop` arrête le pod sur *Drop and execute new binary in container* dans `payments`. Une policy est étiquetée par son effet ici, dans Events et dans le paquet de preuves.



Le formulaire de `payments-stop` : une règle Falco, un namespace, *Send a notification* et *Stop the pod* activés, *Kill at once (recommended)* choisi. L'encadré rouge prévient que les pods concernés seront supprimés sans confirmation et, depuis la 1.4.0, qu'arrêter un pod demande `rbac.allowEnforce: true` dans le chart.

Champ	Effet
Name	Obligatoire, 128 caractères au plus, unique.
Falco rules	La policy correspond aux événements de n'importe laquelle de ces règles, par nom exact. Choisissez dans la liste Rules ou tapez le nom de votre propre règle. Au moins une est obligatoire.
All namespaces / Only in namespaces	Limite la policy aux événements des pods des namespaces listés.
Send a notification	Envoie l'événement à chaque canal actif abonné à Runtime alert.
Open an issue	Ouvre un ticket via les canaux de tickets abonnés à Runtime alert.
Stop the pod	Arrête le pod d'où vient l'événement, avec *Kill at once* ou *Use the application's shutdown time*.
State	On ou Off. Une nouvelle policy démarre sur Off. Une policy sur Off est conservée et ne fait rien.

Si l'une des trois actions est active, la policy est enregistrée avec l'action `notify`. Si les trois sont inactives, elle est enregistrée en `record` : les événements qui correspondent sont consignés contre la policy, et rien d'autre ne se passe. L'API (`/api/v1/runtime/rules`) accepte plus de conditions que le formulaire n'en montre : tags, priorité, source, champs de sortie, dernier verdict et signature de l'image, et une fenêtre de `count` événements en `duration`. Elle accepte aussi l'action `allow`, qui garde l'événement et ne fait rien d'autre. Si une policy a des réglages que le formulaire ne montre pas, le formulaire le dit et les conserve à l'enregistrement.

Arrêter le pod

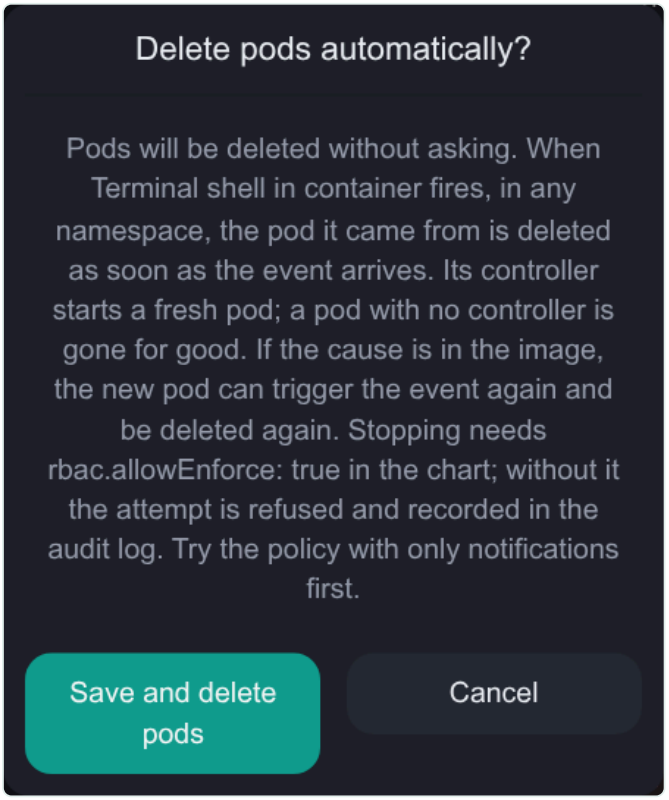
Une policy avec *Stop the pod* arrête le pod nommé dans l'événement dès que celui-ci arrive, ou dès que la fenêtre est atteinte si la policy en a une. L'operator demande à l'API server d'évincer le pod, et les PodDisruptionBudgets s'appliquent. Si un budget n'autorise pas l'éviction, le pod continue de tourner et le journal d'audit en note la raison.

- *Kill at once* (`stop_mode: immediate` , le défaut) : l'éviction utilise un délai de grâce d'une seconde. Tout ce qui a été lancé dans les containers, shell ouvert compris, s'arrête là.
- *Use the application's shutdown time* (`stop_mode: graceful`) : le pod reçoit son propre délai de terminaison, 30 secondes sauf si le pod fixe une autre valeur. Un shell ouvert à l'intérieur tourne jusque-là.

Le contrôleur du pod démarre un nouveau pod. Un pod sans contrôleur ne revient pas. Si la cause est dans l'image, le nouveau pod peut déclencher le même événement et être arrêté à son tour.

- Arrêter exige `rbac.allowEnforce: true` , quoi que le ClusterRole accorde par ailleurs. À partir de la 1.4.0, c'est vrai même quand `rbac.denyList.takedown: true` a donné à l'operator l'éviction de pods pour le take down. Sans ce réglage, l'arrêt est refusé avant que l'API server soit sollicité, le journal d'audit note "not stopped: the chart has not granted pod eviction (set rbac.allowEnforce)", et l'alerte indique que le pod tourne encore. Le rapprochement en `runtimeReconciliation: enforce` suit la même règle.
- Les pods du namespace de l'operator ne sont jamais arrêtés. Aucun autre namespace n'est exclu, alors utilisez *Only in namespaces* pour tenir une policy d'arrêt à l'écart des namespaces système.
- L'événement doit nommer le pod (`k8s.ns.name` et `k8s.pod.name`). Avec les défauts du chart Falco, c'est le cas.
- Un pod déjà en cours d'arrêt n'est pas touché. Chaque réplica de l'operator demande l'arrêt d'un même pod au plus une fois toutes les 10 minutes.

L'enregistrement d'une policy avec *Stop the pod* activé demande d'abord confirmation. La fenêtre dit que les pods seront supprimés sans rien demander, ce qui arrive à un pod avec ou sans contrôleur, et que l'arrêt exige `rbac.allowEnforce: true` dans le chart.



La question posée quand on enregistre une policy où *Stop the pod* vient d'être activé. Elle nomme la règle et les namespaces, dit ce qui arrive à un pod avec et sans contrôleur, et rappelle qu'arrêter un pod demande `rbac.allowEnforce: true` . Sans ce réglage, la tentative est refusée et inscrite au journal d'audit.

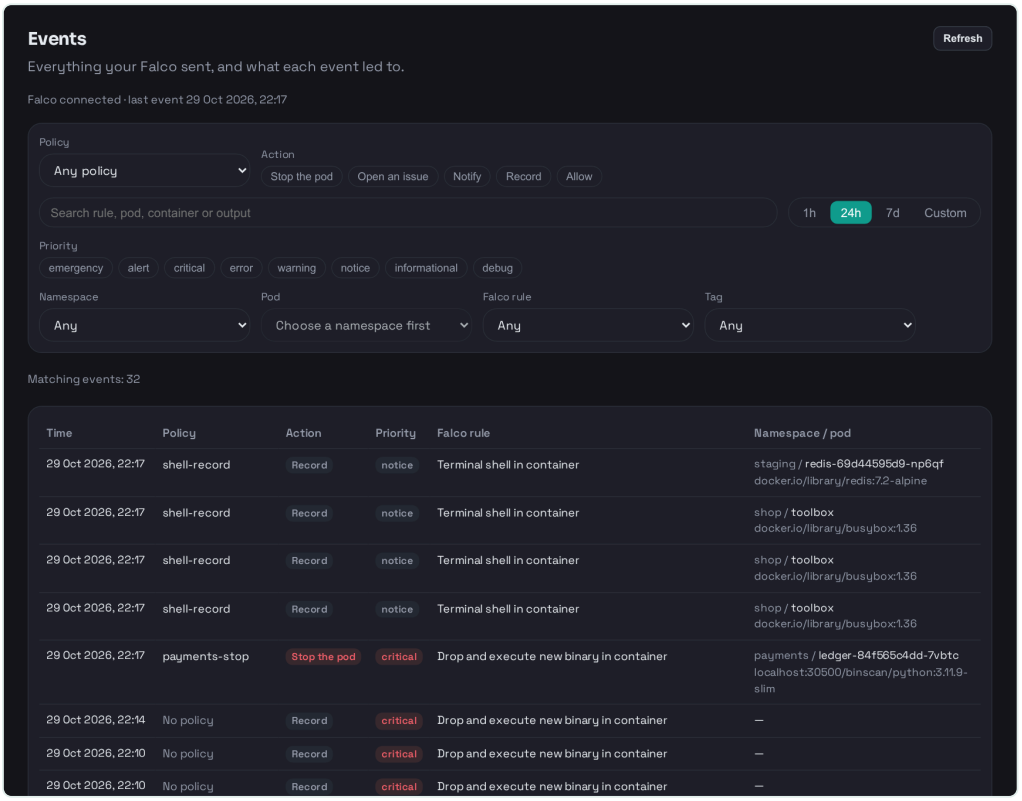
Chaque tentative entre dans le journal d'audit sous `runtime_pod_stop` , avec `policy:<nom>` comme acteur, le pod comme cible, et dans le détail la règle Falco, l'UID du pod, le container, l'image, le digest, le node, l'identifiant d'événement et le résultat. Arrêter un pod ne change pas l'admission. Les nouveaux pods de la même image sont admis comme avant.

Notifications et tickets

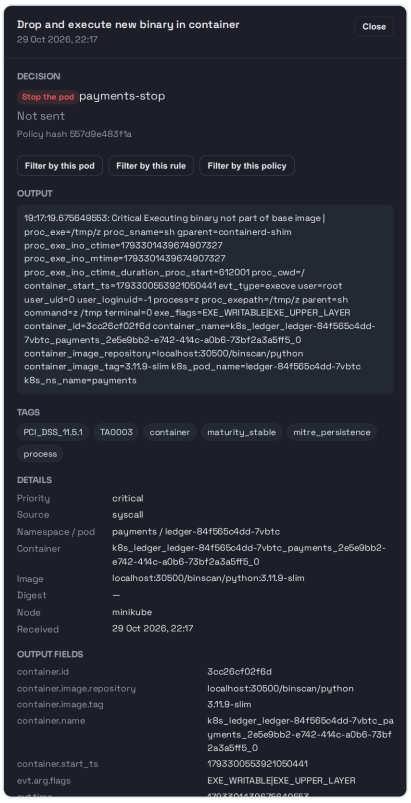
Un message s'intitule `Runtime alert: <règle Falco>` et porte la ligne de sortie de Falco, la règle, la priorité, la source, le namespace, le pod, le container, l'image, le digest, le node, l'heure, la policy et son action. Si la policy a arrêté le pod, le message le dit. Une policy envoie au plus un message par digest d'image et par heure. Cette limite vaut pour les messages et les tickets. Elle ne retarde pas l'arrêt d'un pod.

Un ticket n'est ouvert que si l'événement est rattaché à un digest d'image : un ticket par digest et par règle Falco, intitulé `RUNTIME: <rule> in <image>` et étiqueté `runtime`. Il se ferme tout seul en `quiet` après 7 jours sans que la règle se déclenche sur cette image, ou en `image_retired` quand aucun pod en cours n'a utilisé le digest depuis 24 heures. Aucun des deux ne signifie que le problème a été corrigé. Un événement ne compte comme notifié que si au moins un canal actif est abonné pour le recevoir.

Events



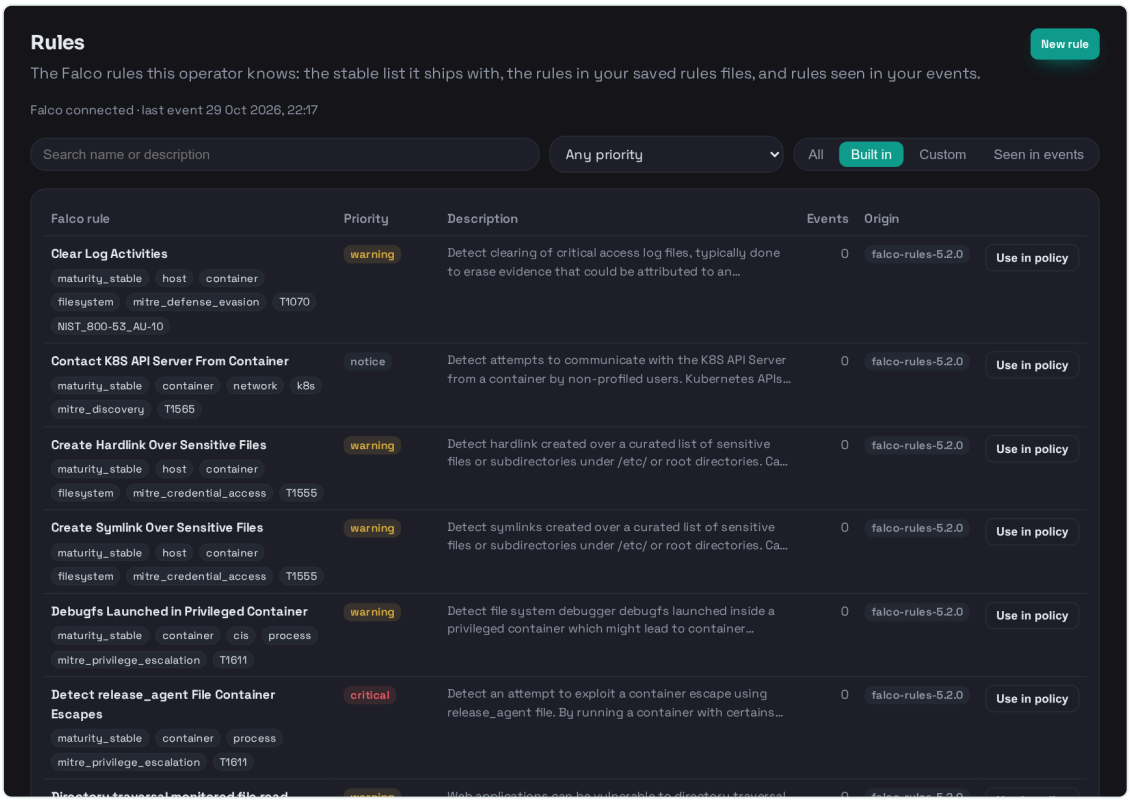
Falco > Events sur les dernières 24 heures : 32 événements. Chaque ligne donne la policy qui correspond et son action, la priorité, la règle Falco, le namespace et le pod. Les événements sans policy affichent *No policy* et restent enregistrés. Les filtres portent sur la policy, l'action, le texte libre, la priorité, le namespace, le pod, la règle Falco et le tag, sur une plage de temps.



Un événement ouvert : *Drop and execute new binary in container* dans `payments`, décision *Stop the pod* par `payments-stop`. Le panneau montre la décision, si un message est parti, le hash de la policy, la ligne de sortie de Falco, les tags, et le namespace, le pod, le container, l'image et le node auxquels l'événement est rattaché. Sur cette installation, le chart ne définit pas `rbac.allowEnforce` : l'arrêt a été refusé et le pod a continué de tourner.

Les règles Falco et l'éditeur de règles (bêta)

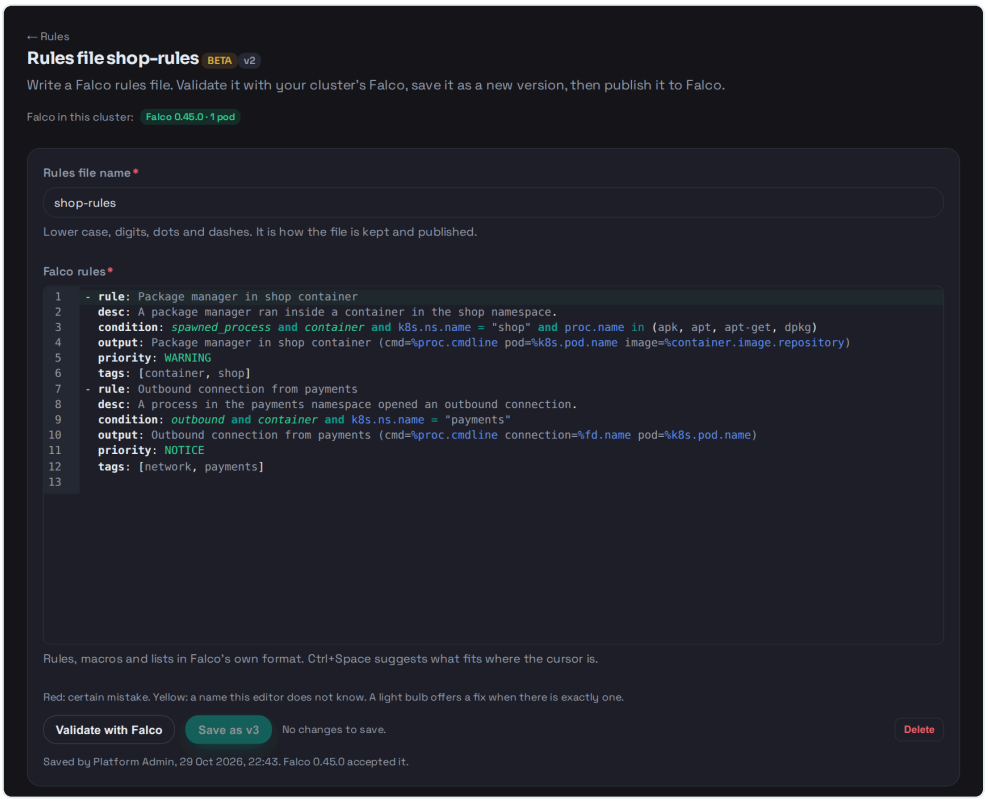
Falco > Rules liste les règles Falco que connaît l'operator : les 25 règles stables de `falcosecurity/rules` livrées avec lui, les règles de vos fichiers de règles enregistrés, et les noms de règles vus dans les événements. *Use in policy* crée une policy à partir d'une règle.



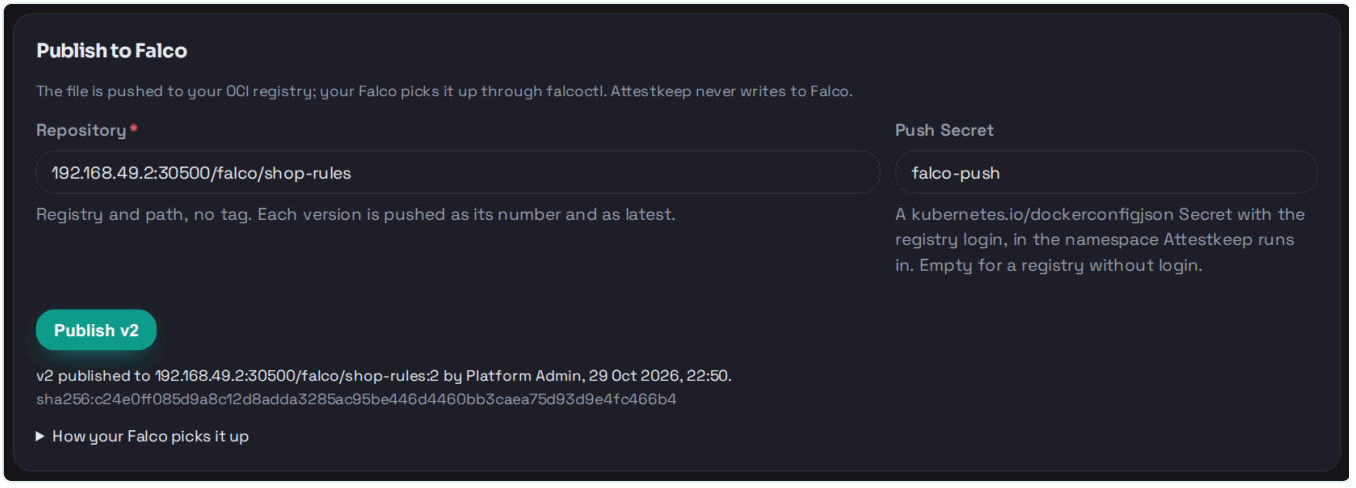
Falco > Rules avec le filtre *Built in*. Chaque règle montre sa priorité, ses tags, sa description, le nombre d'événements produits ici et son origine, `falco-rules-5.2.0`. *Use in policy* démarre une policy runtime à partir de la règle.

New rule (admins) ouvre un éditeur pour un fichier de règles Falco : règles, macros, listes et overrides dans le YAML propre à Falco. L'éditeur est en bêta. La complétion et les contrôles suivent la version de Falco qui tourne dans votre cluster, trouvée par le label `app.kubernetes.io/name=falco`.

- *Validate with Falco* lance dans le namespace de l'operator un pod ponctuel avec la même image que votre Falco. Il charge les règles par défaut de Falco et votre fichier, exécute Falco en mode validation et est supprimé dès que le résultat est lu. Votre Falco en cours n'est pas touché.
- L'enregistrement n'est possible qu'une fois que Falco a accepté exactement le texte de l'éditeur. Le serveur le valide de nouveau avant de le stocker. Chaque enregistrement est une nouvelle version, consignée avec qui l'a enregistrée et quelle version de Falco l'a acceptée. Un nom de règle déjà défini par un autre fichier enregistré est refusé.
- Publish pousse une version enregistrée dans votre propre registry OCI, comme artefact falcoctl `rulesfile`, tagué avec le numéro de version et `latest`. Si le registry exige une connexion, vous donnez le nom d'un Secret `kubernetes.io/dockerconfigjson` du namespace de l'operator. Attestkeep le lit au moment de pousser et ne conserve pas l'identifiant.



Le fichier de règles enregistré `shop-rules` en version 2. La ligne sous le titre montre le Falco trouvé dans le cluster : Falco 0.45.0 sur 1 pod. *Save as v3* reste désactivé tant que le texte n'a pas changé et que Falco ne l'a pas accepté. La dernière ligne dit qui a enregistré la version 2 et quelle version de Falco l'a acceptée.



Publish to Falco : le dépôt dans votre propre registry, le nom du Secret de push et la trace de la publication de la version 2 avec son digest. *How your Falco picks it up* ouvre les valeurs falcoctl pour votre chart Falco.

Votre Falco charge le fichier par falcoctl. Dans les valeurs du chart Falco, ajoutez votre référence à `install.refs` et à `follow.refs`, et gardez `falco-rules:5`, la référence par défaut du chart. Helm remplace les listes au lieu de les fusionner, et sans elle falcoctl n'installe plus `/etc/falco/falco_rules.yaml`. Ajoutez ensuite votre fichier à `rules_files`, après les règles propres de Falco. Supprimer un fichier dans la console retire ses règles de la liste Rules. Ce qui a déjà été publié reste dans le registry et dans chaque Falco qui l'a installé, tant que vous ne retirez pas la référence de falcoctl.

Conservation et métriques

Chaque événement est conservé. Attestkeep n'a pas de seuil de priorité à lui. Les événements plus anciens que `runtime.retentionDays` (90 par défaut) sont supprimés toutes les heures, et `0` les garde sans limite. L'historique des

tickets, les entrées d'audit et les révisions de policies ne sont pas supprimés. Pour que Falco n'envoie pas d'événements que vous ne voulez jamais garder, réglez son propre `falco.priority`.

```
attestkeep_runtime_events_total{source, priority}
attestkeep_runtime_events_dropped_total{reason="duplicate|unresolved_image|throttled"}
attestkeep_runtime_alerts_notified_total
attestkeep_runtime_rule_matches_total{rule, action}
```

Dans `attestkeep_runtime_rule_matches_total`, `rule` est le nom de la policy runtime. `unresolved_image` ne veut pas dire que l'événement est perdu. Falco observe aussi les processus de l'hôte, et un tel événement est conservé sans image.

9. Prouver que le contrôle s'applique

Un webhook d'admission qui ne tourne pas ressemble à un cluster conforme. Cette vérification prend environ cinq minutes, n'utilise que kubectl et se termine par un refus que vous avez vu vous-même. Faites-la après l'installation, après les mises à niveau et chaque fois que vous devez confirmer que le contrôle s'applique.

1. Essayer un pod que la policy doit refuser

```
kubectl run enforcement-check --image=nginx:latest
```

L'API server refuse et cite le webhook. Voici la sortie d'une installation avec la policy par défaut :

```
Error from server: admission webhook "image-security.attestkeep.com" denied the request:
image security policy: nginx:latest: the :latest tag is blocked by policy
```

Le message commence par *Error from server* : le refus vient donc de la chaîne d'admission de l'API server. La raison nomme la règle de la policy, et un développeur qui la lit dans un log de CI sait quoi changer.

2. Vérifier que rien n'a été créé

```
kubectl get pod enforcement-check
Error from server (NotFound): pods "enforcement-check" not found
```

3. Vérifier que la décision a été enregistrée

Le tableau *Recently blocked admissions* du tableau de bord commence maintenant par ce refus, avec la même raison. La métrique `attestkeep_admission_decisions_total{decision="denied"}` a augmenté.

4. Si le pod a été admis

- Le namespace est contourné : kube-system, cert-manager, le namespace de l'operator, ou un namespace étiqueté `attestkeep.com/webhook=bypass`.
- La policy ne couvre pas ce que vous avez essayé. Vérifiez `enforcement: Enforce` et `blockLatestTag: true`.
- Le webhook laisse passer en cas d'échec. Vérifiez que `attestkeep_webhook_distrusted` vaut 0 et que les pods sont prêts.
- L'installation n'est pas activée. Tant qu'aucune clé n'est saisie, rien n'est appliqué, et chaque réponse d'admission le dit dans un avertissement.

Avec DenyUntilScanned

Avec `admissionTiming: DenyUntilScanned`, une nouvelle image est refusée tant que son scan n'existe pas. Sortie d'une installation Community :

```
kubectl run cold-check --image=busybox:1.36.1 --restart=Never -- sleep 3600
Error from server: admission webhook "image-security.attestkeep.com" denied the request:
image security policy: busybox:1.36.1: image has not been scanned yet; a scan has been queued — retry
shortly

# peu après, la même commande :
pod/cold-check created
```

10. Paquets de preuves et modèle de confiance

Un paquet de preuves est un document signé par période d'audit. Il est produit dans le cluster à partir de vos propres enregistrements, et un auditeur peut le vérifier avec ses propres outils. C'est une preuve pour un audit, pas une certification, et le document le dit.

Les cadres auxquels il se rapporte

Texte	Ce qui est couvert
SOC 2, AICPA Trust Services Criteria	13 critères de CC6 à CC9 et Availability A1.
CRA et NIS2 de l'UE : (UE) 2024/2847 et article 21(2)	34 exigences tirées des deux textes.
DORA, règlement (UE) 2022/2554	22 exigences des articles 8 à 11 et de l'article 28.
NIST SP 800-190	15 contre-mesures pour les images, les registries, les orchestrateurs et les containers.
SSDF, NIST SP 800-218 v1.1	13 pratiques de PO, PS, PW et RV.
NIST SP 800-53 Rev. 5	15 contrôles choisis parmi CM, RA, SI, SR, AU et CP.
ISO/IEC 27001:2022 annexe A	12 clauses : 10 contrôles de l'annexe A et des clauses marquées hors périmètre.
RGPD article 32	6 clauses sur la sécurité du traitement.
PCI DSS v4.0.1 (PCI SSC, juin 2024)	13 clauses des exigences 6.3, 6.5, 10.2 et 11.3.
MAS Technology Risk Management Guidelines (18 janvier 2021)	30 clauses des sections 3.3, 6.1, 6.3, 7.2 à 7.6, 11.3, 11.4, 12.1, 12.2, 13.1, 13.6 et 15.
ASD Essential Eight Maturity Model (novembre 2023)	46 clauses : correctifs des applications, correctifs des systèmes d'exploitation et application control, Maturity Levels un à trois.
ISMAP Control Criteria (édition japonaise du 1er août 2024)	42 clauses des critères de contrôle du tableau annexé 3.
ISMS-P Certification Criteria (avis PIPC et MSIT, annexe 7, guide KISA de novembre 2023)	32 clauses des domaines 1 et 2.

Treize textes et 293 clauses en tout, chacune avec sa propre référence. La correspondance est livrée dans l'operator, si bien qu'un cluster isolé rend compte de la même façon qu'un cluster connecté. Chaque contrôle aboutit à l'un de quatre états : *met*, *partial*, *gap* ou *outside_scope*. Une clause n'est satisfaite que si les enregistrements de la période montrent le contrôle à l'œuvre, jamais simplement parce que le produit a la fonction. Une clause à laquelle un cluster ne peut pas répondre est déclarée hors périmètre avec la raison, et elle est comprise dans le total ci-dessus.

Le nombre de frameworks pour lesquels vous pouvez produire des paquets dépend de la licence (chapitre 13).

Sources traduites. Deux correspondances reposent sur des textes non publiés en anglais, et chaque clause le dit. Pour **ISMAP**, il existe une traduction anglaise de référence, mais là où son entrée renvoie seulement à ISO/IEC 27002, l'obligation est traduite de l'édition japonaise du 1er août 2024 et la référence indique laquelle. Pour **ISMS-P**, il n'existe pas de texte anglais officiel. Chaque clause est notre traduction du critère coréen tel qu'imprimé dans le guide de KISA. Chaque critère ISMS-P associe une mesure technique à une procédure, une désignation ou une revue périodique, et les enregistrements d'un cluster ne répondent qu'à la partie technique : aucun critère ISMS-P n'est donc déclaré pleinement satisfait.

Cinq attestations alimentent les correspondances les plus récentes. `patch-window-48-hours` est satisfaite quand le délai déclaré pour les vulnérabilités critiques est d'au plus 48 heures et que les vulnérabilités exploitées sont refusées à l'admission. `patch-window-two-weeks` est satisfaite quand le délai déclaré pour les vulnérabilités hautes est d'au plus deux semaines. `runtime-observed` est satisfaite quand l'intégration Falco est active et qu'au moins un événement est arrivé dans la période. Elle est partielle quand l'intégration est active et que rien n'est arrivé, parce qu'un cluster calme et un Falco qui n'envoie plus se ressemblent vus de l'operator. C'est un écart quand l'intégration est désactivée. `producer-statement-honoured` est satisfaite quand la policy applique les déclarations VEX et fait confiance à au moins un émetteur. `context-aware-thresholds` est satisfaite quand l'escalade est active pour `exposed` et `privileged`, partielle avec l'un des deux, et un écart en `Ignore`. `remediation-windows` ne peut atteindre *met* qu'avec un gestionnaire de tickets connecté, car c'est seulement alors qu'il existe une trace de la date d'une correction.

Comment chaque exigence a reçu sa réponse

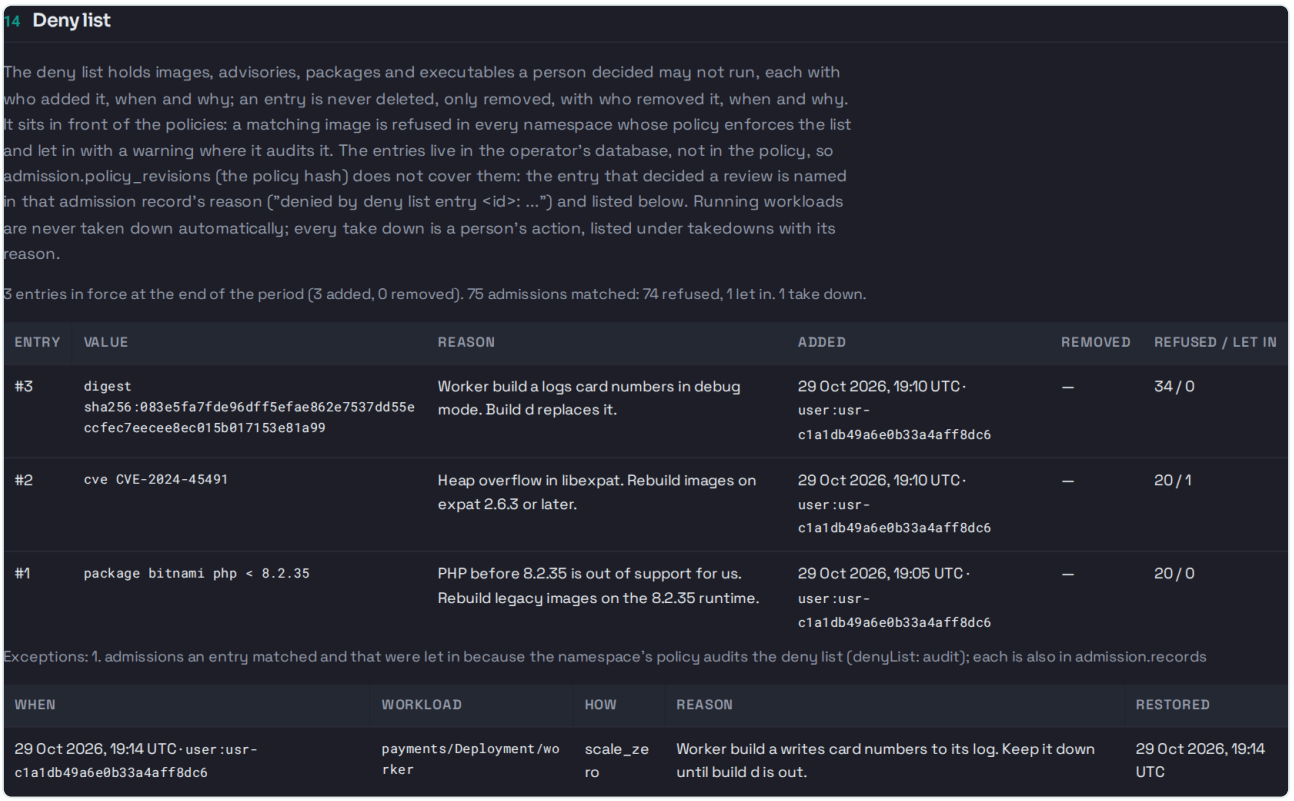
Chaque exigence liste les tests qui y ont répondu. Un test a une méthode (`TEST` quand il lit la trace de ce qui s'est passé, `EXAMINE` quand il lit une configuration), un résultat, une observation, des chemins dans le rapport, la source lue et le moment de la lecture. Chaque exigence indique aussi `result` et `implementation_status` au sens d'OSCAL. Le PDF imprime les tests sous chaque exigence.

Le bloc `generation` dit quand et par quoi le document a été produit, le système source, la fenêtre de requête (fin exclue, en UTC), la requête de chaque section avec son nombre de lignes et si une requête plafonnée a atteint son plafond, et comment les personnes sont nommées. À partir de la 1.3.0, il porte aussi `nodes {count, ceiling}` : les nodes Ready du cluster au moment de la production et le plafond de nodes de la licence, sous forme de nombres uniquement. `admission.records[]` porte chaque revue d'admission de la période, une ligne par revue et sans plafond, pour qu'un auditeur puisse échantillonner sur toute la population. `organisation.declarations` porte la source de temps et la règle de conservation de l'exploitant, imprimées comme déclarées par lui.

Les personnes sont nommées par compte, pas par nom. `generated_by`, `chain[].actor` pour les lignes du journal d'audit, `triage[].triaged_by` et `runtime.audit_entries[].actor` contiennent `user:<id de compte>`. Un pod arrêté par une policy runtime s'écrit `policy:<nom>`. Le journal d'audit de la console continue d'afficher des noms, et l'exploitant s'en sert pour résoudre un identifiant quand un auditeur le demande. Les lignes enregistrées avant la 1.2.0 qui ne portent qu'un nom sont rattachées à un compte par nom affiché ou e-mail exact, ou écrites `user:unresolved` et comptées. L'identité Kubernetes derrière chaque admission est écrite telle quelle si c'est une identité `system:`, sinon sous la forme `k8s-user:` suivie des 16 premiers caractères hexadécimaux de son SHA-256.

La section runtime porte le nombre d'événements par priorité, les règles Falco les plus actives, les alertes envoyées, et les tickets runtime ouverts et fermés. Elle porte aussi le jeu de policies runtime en vigueur avec son hash de révision, les `audit_entries` des changements de policy, des pods arrêtés et des changements d'intégration de la période, les correspondances par policy et par action, les événements sur lesquels une policy a agi (jusqu'à 500, chacun avec le ticket qu'il a ouvert), et la couverture : intégration active ou non, dernier événement et plus long intervalle sans événement. `event_time` est l'horodatage propre de Falco. `received_at` est le moment où l'operator a reçu l'événement. La couverture utilise `received_at`, et `coverage.clock` le dit. `runtime.rules[].effect` et `runtime.decisions[].policy_effect` donnent l'action prévue par la policy. Le succès d'un arrêt se lit dans la ligne `runtime_pod_stop` de `runtime.audit_entries`.

Export externe et deny list. À partir de la 1.4.0, le paquet a deux sections de plus. *External export* liste les destinations SIEM, le dernier lot que chacune a accepté, et les événements abandonnés dans la période (`external_export` dans `report.json`). *Deny list* liste les entrées en vigueur et celles ajoutées ou retirées dans la période, les admissions qu'elles ont refusées et laissées passer, les exceptions, chaque take down et chaque rétablissement, et pour une entrée d'exécutable le fichier, son hash et la version (`deny_list` dans `report.json`).



14 Deny list

The deny list holds images, advisories, packages and executables a person decided may not run, each with who added it, when and why; an entry is never deleted, only removed, with who removed it, when and why. It sits in front of the policies: a matching image is refused in every namespace whose policy enforces the list and let in with a warning where it audits it. The entries live in the operator's database, not in the policy, so admission.policy_revisions (the policy hash) does not cover them: the entry that decided a review is named in that admission record's reason ("denied by deny list entry <id>: ...") and listed below. Running workloads are never taken down automatically; every take down is a person's action, listed under takedowns with its reason.

3 entries in force at the end of the period (3 added, 0 removed). 75 admissions matched: 74 refused, 1 let in. 1 take down.

ENTRY	VALUE	REASON	ADDED	REMOVED	REFUSED / LET IN
#3	digest sha256:083e5fa7fde96dff5efae862e7537dd55eccfec7eecee8ec015b017153e81a99	Worker build a logs card numbers in debug mode. Build d replaces it.	29 Oct 2026, 19:10 UTC · user:usr-c1a1db49a6e0b33a4aff8dc6	—	34 / 0
#2	cve CVE-2024-45491	Heap overflow in libexpat. Rebuild images on expat 2.6.3 or later.	29 Oct 2026, 19:10 UTC · user:usr-c1a1db49a6e0b33a4aff8dc6	—	20 / 1
#1	package bitnami php < 8.2.35	PHP before 8.2.35 is out of support for us. Rebuild legacy images on the 8.2.35 runtime.	29 Oct 2026, 19:05 UTC · user:usr-c1a1db49a6e0b33a4aff8dc6	—	20 / 0

Exceptions: 1. admissions an entry matched and that were let in because the namespace's policy audits the deny list (denyList: audit); each is also in admission.records

WHEN	WORKLOAD	HOW	REASON	RESTORED
29 Oct 2026, 19:14 UTC · user:usr-c1a1db49a6e0b33a4aff8dc6	payments/Deployment/worker	scale_zero	Worker build a writes card numbers to its log. Keep it down until build d is out.	29 Oct 2026, 19:14 UTC

La section 14, *Deny list*, du même paquet. Trois entrées étaient en vigueur en fin de période. Elles ont concerné 75 admissions : 74 refusées et 1 admise là où la policy du namespace audite la liste. Chaque entrée montre sa valeur, sa raison, qui l'a ajoutée et quand, et ses nombres de refus et d'admissions. Sous les exceptions figure le take down de `payments/Deployment/worker`, avec sa raison et l'heure de sa restauration.

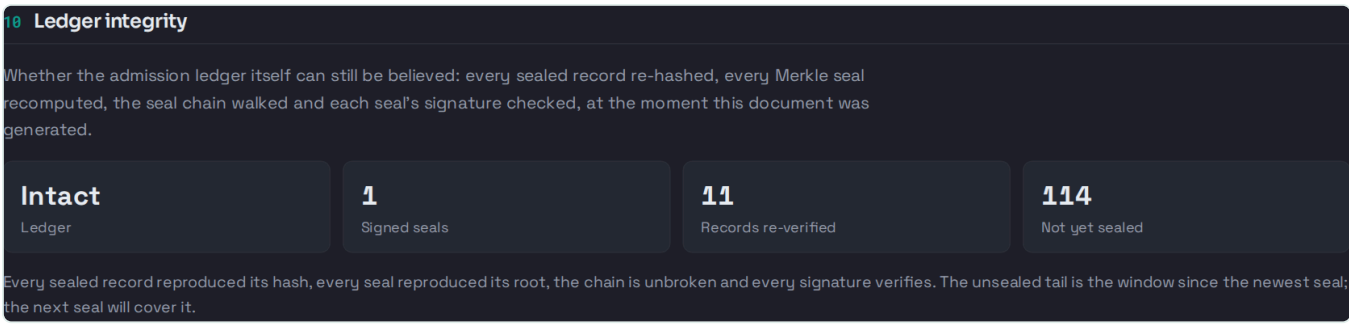
Les vulnérabilités sont comptées par digest. À partir de la 1.4.0, un digest tiré par deux références, son tag et son digest ou deux noms, forme une seule ligne de `findings.critical` qui les nomme tous deux sous `references`, et ses vulnérabilités comptent une fois dans `coverage.severity` et `coverage.findings_total`. Un paquet produit maintenant pour le même cluster peut afficher moins qu'un paquet de la 1.3.7, exactement de ces doublons.

Break-glass et triage. À partir de la 1.3.4, chaque activation de break-glass est lue dans la révision de policy sous laquelle ses admissions ont été décidées, si bien qu'un paquet dit ce qui était approuvé à ce moment même après la fermeture de la fenêtre. Produisez de nouveau tout paquet qui couvre une ancienne fenêtre de break-glass pour obtenir la trace complète. Les décisions de triage portent la gravité tirée des résultats de scan, la plus haute quand une CVE a été enregistrée avec plusieurs.

Intégrité du registre

Chaque décision d'admission est écrite dans PostgreSQL. Une base est un bon endroit pour interroger des décisions, mais ce n'est pas, à elle seule, une raison de leur faire confiance. Le registre est construit pour que la falsification se voie :

- **Chaque enregistrement porte son propre hash**, un SHA-256 de son contenu (heure, namespace, pod, image, digest, verdict, raison, nom et hash de la policy, indicateur de break-glass), calculé à l'écriture.
- **Les enregistrements sont scellés sous des checkpoints signés**, toutes les heures par défaut ou tous les 10 000 enregistrements. Le réplica leader construit une racine de Merkle sur la plage non scellée et signe un checkpoint qui inclut le hash du précédent, si bien que les sceaux forment une chaîne.
- **La clé de signature n'est pas dans la base.** C'est une clé Ed25519 générée dans votre cluster et gardée dans le Secret `attestkeep-evidence-key`. Réécrire des lignes du registre exige un accès en écriture à la base. Signer de nouveau les sceaux exige l'accès à ce Secret. L'un des deux ne suffit pas. À partir de la 1.4.0, la même clé est aussi l'identité de licence de l'installation (chapitre 12).
- **La même clé signe vos documents de preuve** sous forme d'enveloppes DSSE, et le certificat de licence nomme votre clé, si bien qu'un tiers peut rattacher un document à votre installation.



La vérification du registre dans le paquet. À la génération, chaque enregistrement scellé a été haché de nouveau, chaque sceau Merkle recalculé, la chaîne parcourue et chaque signature vérifiée. Ici : *Intact*, 1 sceau signé et 11 enregistrements revérifiés. 114 enregistrements ont été écrits après le dernier sceau et comptent comme pas encore scellés, d'où la lecture d'*Intact* avec le nombre de sceaux.

Où s'arrêtent les garanties

Ce dispositif rend la falsification visible. Il ne l'empêche pas. Les enregistrements écrits depuis le dernier checkpoint sont comptés mais pas encore scellés, sur une fenêtre d'au plus un intervalle de scellement. *Intact* parle de ce qui est scellé, lisez-le donc avec le nombre de checkpoints. Un attaquant qui a un accès en écriture à la base et l'accès à la clé de signature peut contourner le dispositif. L'affirmation est que cela demande deux accès distincts. L'auditeur reçoit un résultat signé et l'enveloppe DSSE. Refaire la vérification de Merkle de façon indépendante demanderait un accès à la base.

Signer un paquet pour un auditeur produit un PDF à côté du JSON, remis ensemble dans une archive signée. N'importe qui peut vérifier un paquet remis sur attestkeep.com/verify.html avec ses propres outils.

11. Exploitation, export SIEM et sauvegarde

Sondes

Point d'accès	Ce qu'il vérifie	Utilisé comme
`/healthz`	Seulement que le processus tourne.	Sonde liveness (délai initial 10s, période 20s)
`/readyz`	À partir de la 1.4.0, que le serveur répond et que son certificat est chargé. La base n'en fait plus partie : un réplica reste prêt pendant que la base est injoignable, et le webhook continue de refuser selon les règles statiques.	Sonde readiness (délai initial 5s, période 10s)

Ce que fait une panne de la base

- Les réplicas restent prêts et continuent de répondre. Une revue d'admission accorde deux secondes au total à son travail en base. Ensuite, les règles statiques tranchent aussitôt : `:latest`, la liste des registries autorisés et l'exigence de digest continuent de refuser. L'enregistrement et les verdicts qui dépendent du cache se dégradent, et la panne est journalisée une fois.
- La base est suivie dans `attestkeep_database_up` et sur la page de santé. Si vous surveilliez la readiness de l'operator pour détecter une panne de base avant la 1.4.0, surveillez cette métrique à la place.
- Ensuite, l'operator repère le trou dans le heartbeat et l'enregistre comme panne, et le rapprochement suivant liste chaque digest entré sans décision.
- Les notifications partent en dehors de la requête, 64 au plus en attente. Les autres sont abandonnées et comptées dans `attestkeep_notifications_dropped_total`.
- Le comptage du quota admet en cas d'erreur de base au lieu de bloquer les déploiements.
- Le scan s'arrête avec la base et reprend avec elle. La file d'attente est dans la base, rien n'est perdu.

Ce qui doit vous alerter

Les métriques sont servies sur le port 9090 sous `/metrics`. Chaque série étiquetée existe dès le démarrage, à zéro, si bien qu'une donnée absente et zéro refus sont deux choses différentes.

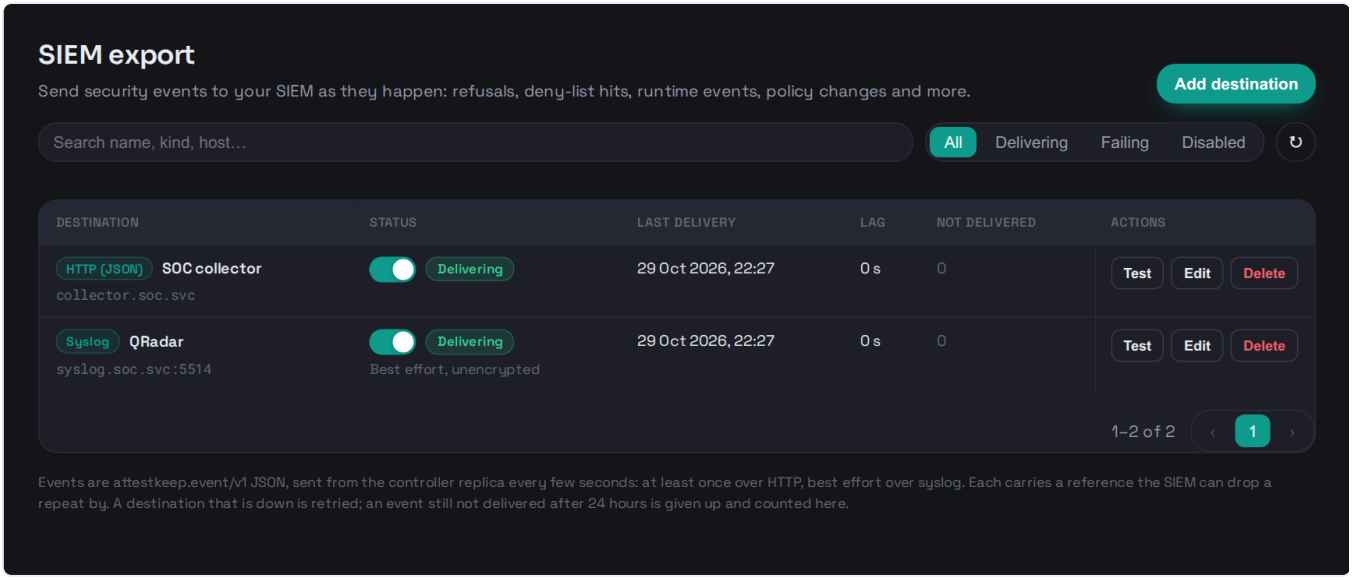
Situation	Signal	Pourquoi c'est important
L'admission est contournée	`attestkeep_webhook_distrusted == 1`	Tant que cela vaut 1 sous `failurePolicy: Ignore`, les déploiements passent sans contrôle. Traitez-le comme une alerte immédiate.
Quelque chose est entré	`attestkeep_runtime_unmatched_digests > 0`	Le rapprochement a trouvé des digests en cours sans admission autorisée derrière eux.
Problème de base	`attestkeep_database_up == 0`	À partir de la 1.4.0, les réplicas restent prêts pendant une panne, et les alertes de readiness ne la montrent plus.
Le scan prend du retard	`attestkeep_scan_queue_depth` qui reste haut	Une file qui ne fait que grandir signifie que le scanner n'atteint pas un registry ou sa base.
Les données de vulnérabilités vieillissent	`attestkeep_vulnerability_db_age_seconds`	Les verdicts ne sont pas plus à jour que la base qui les sous-tend.
Quota d'images près de la limite	`attestkeep_image_quota{kind="count"}` face à `{kind="limit"}`	À la limite, seuls les noms d'image pas encore vus dans la période sont refusés. Les workloads en cours continuent.
Une destination SIEM prend du retard	`attestkeep_siem_lag_seconds{target}`, `attestkeep_siem_events_total{outcome="gave_up"}`	Le retard est l'âge du plus vieil événement que la destination n'a pas reçu. Les événements non livrés au bout de 24 heures sont abandonnés.
Au-dessus du plafond de nodes	`attestkeep_nodes{kind="count"}` face à `{kind="ceiling"}`, `attestkeep_node_overage_seconds > 0`	La jauge de dépassement indique depuis combien de temps le cluster est au-dessus du plafond. Alertez bien avant les 14 jours au-delà desquels les nouveaux noms d'image sont refusés.

Les refus sont comptés dans `attestkeep_admission_denials_total{reason}`. Les raisons comprennent `threshold`, `signature`, `attestation`, `tag`, `digest`, `registry`, `quota`, `unscanned`, `scan_error`, `end_of_life`, `known_exploited`, `provenance` et `context`. Les refus dus au plafond de nodes comptent sous `quota`. Autres séries utiles pour un tableau de bord : `attestkeep_vex_statements_total`, `attestkeep_vex_rejected_total`, `attestkeep_issues_opened_total`, `attestkeep_issues_closed_total`, `attestkeep_issues_skipped_total`, `attestkeep_admission_context_total{signal}`, `attestkeep_policy_whatif_runs_total`, et les séries runtime du chapitre 8.

Les événements de notification couvrent le même terrain : `threshold_80` et `threshold_100` pour le quota d'images, `nodes_80`, `nodes_over`, `nodes_enforced` et `nodes_cleared` pour le plafond de nodes, `license_expiring` quand un certificat en ligne a moins d'une semaine devant lui, `trivy_db_failed`, `unverified_workload`, `runtime_alert`, et à partir de la 1.4.0 `deny_list_hit`, `deny_list_entry_added` et `deny_list_entry_removed`. Un nouveau canal a `nodes_over`, `nodes_enforced`, Runtime alert et Deny list hit cochés.

Export SIEM

Votre équipe sécurité regarde le SIEM, pas cette console. À partir de la 1.4.0, Settings > SIEM export y envoie les événements de sécurité d'Attestkeep au moment où ils se produisent, pour qu'ils s'inscrivent sur la même ligne de temps que tout ce que vous collectez. Rien ne sort tant que vous n'ajoutez pas de destination, et en ajouter une ne change aucune décision d'admission. Toute personne qui se connecte voit les destinations et leur état. Ajouter, modifier, tester et retirer une destination revient à un administrateur.



Settings > SIEM export avec deux destinations. *SOC collector* reçoit du HTTP JSON et *QRadar* du syslog au format LEEF sur TCP simple. Les deux affichent *Delivering*, sans retard et sans événement abandonné. La ligne syslog dit ce que signifie le TCP simple : best effort, non chiffré.

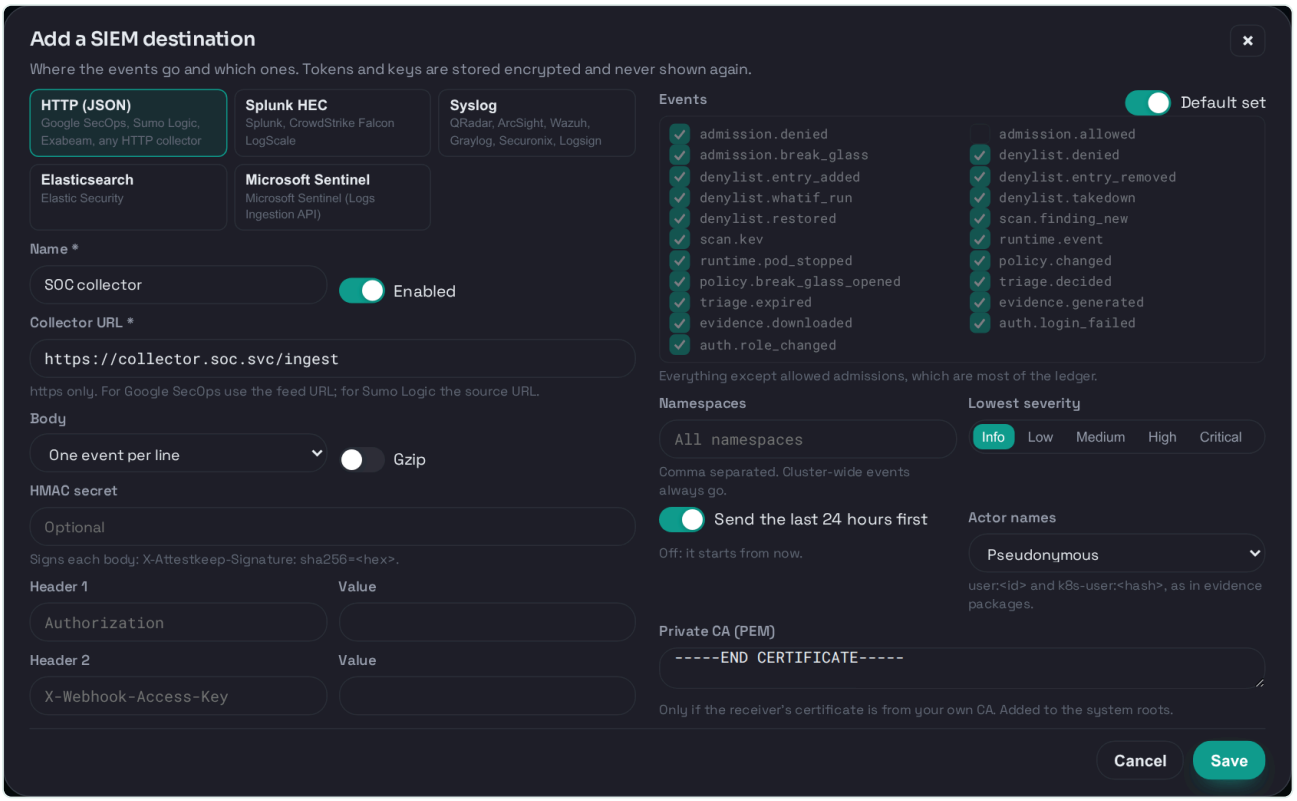
Ce qui est envoyé. Les admissions refusées, les correspondances de deny list, les entrées ajoutées et retirées, les take downs et rétablissements, les événements runtime et les pods arrêtés par une policy runtime, les changements de policy, de break-glass et de triage, la production et le téléchargement de preuves, les connexions échouées et les changements de rôle. Une vulnérabilité critique est envoyée une fois, par le scan qui l'a signalée le premier, et une vulnérabilité exploitée une fois, par le scan qui l'a vue le premier sur la liste KEV. Les admissions autorisées restent désactivées sauf si vous les cochez, parce qu'elles forment l'essentiel du registre. Chaque destination filtre par type d'événement, namespace et gravité minimale, et une nouvelle destination peut commencer par les 24 dernières heures.

L'événement. Chaque événement est un objet JSON au schéma `attestkeep.event/v1`, avec l'heure, le type, la gravité, le cluster, le namespace, le pod, le workload, l'image, le digest, la policy, l'acteur, le résultat, la raison et une `reference` vers l'enregistrement d'origine. La référence reste la même quand un événement est renvoyé, et votre SIEM peut donc écarter les doublons avec elle. Un événement ne porte jamais de SBOM, de liste de vulnérabilités, d'identifiant secret, de données de licence ni ce qui a été saisi lors d'une connexion échouée. Une destination peut être réglée pour nommer les personnes en `user:<id de compte>`, comme le paquet de preuves.

Destination	Pour	Comment
HTTP (JSON)	Google SecOps, Sumo Logic, Exabeam, tout collecteur HTTP	Un POST HTTPS par lot, un tableau JSON ou un événement par ligne. Gzip facultatif, deux en-têtes de requête et une signature HMAC dans `X-Attestkeep-Signature`.
Splunk HEC	Splunk, CrowdStrike Falcon LogScale	Des événements HEC avec la source `attestkeep` et le sourcetype `attestkeep:event`.
Syslog	QRadar, ArcSight, Wazuh, Graylog, Securonix, Logsign	RFC 5424 sur TLS, ou TCP simple pour un récepteur sans TLS, avec un corps CEF, LEEF ou JSON.
Elasticsearch	Elastic Security	L'API bulk vers un index ou un data stream, avec une clé d'API.
Microsoft Sentinel	Microsoft Sentinel	La Logs Ingestion API via une data collection rule, avec une application Entra.

Livraison. Les événements partent au moins une fois en HTTP, et au mieux en syslog, qui n'a pas d'accusé de réception. Ils partent dans l'ordre des commits, toutes les quelques secondes, 100 au plus par requête, depuis le réplica qui fait le travail périodique. Une destination en échec est réessayée avec des attentes croissantes, ou après son `Retry-After`, et ne retient jamais une autre destination. Une destination bloquée derrière une transaction de base ouverte affiche *Waiting*. Un événement toujours pas livré 24 heures après s'être produit est abandonné et affiché comme "N events not delivered", et la section External export du paquet de preuves le liste. *Test* envoie un événement par la destination et montre la réponse.

Identifiants. Tokens, clés et secrets sont chiffrés avec la data key de l'installation et ne sont plus jamais affichés. Quand une modification change l'adresse d'une destination, les identifiants enregistrés sont supprimés, pour qu'ils ne partent jamais vers la nouvelle adresse sans que vous les y saisissiez. Les adresses loopback et link-local, dont le service de métadonnées du cloud, sont refusées. Les destinations HTTP doivent utiliser `https://`, et un récepteur dont le certificat vient de votre propre CA reçoit cette CA dans le formulaire.



Add a SIEM destination avec HTTP (JSON). L'URL du collecteur doit être en https. Le corps contient un événement par ligne, le jeu par défaut couvre tout sauf les admissions autorisées, et *Send the last 24 hours first* envoie d'abord la journée précédant l'ajout de la destination, puis les nouveaux événements. Les noms d'acteurs sont pseudonymes, comme dans les paquets de preuves. Un récepteur dont le certificat vient de votre propre CA reçoit ici cette CA, ajoutée aux racines du système. Les tokens et les clés sont stockés chiffrés et ne sont plus jamais affichés.

L'export SIEM exige PostgreSQL 13 ou plus récent, parce qu'il lit dans l'ordre des commits. Le PostgreSQL fourni est en version 16. Sur un serveur externe plus ancien, aucune destination ne peut être activée, et la page affiche *Database too old*. Même sans destination, les logs, traces et métriques de l'operator peuvent partir vers un collecteur OpenTelemetry avec `observability.otlp.endpoint`. docs.attestkeep.com/siem donne la configuration de chaque produit SIEM.

Ce qu'il faut sauvegarder

Attestkeep ne fait pas de sauvegarde de base. Lancer, planifier et conserver le dump vous revient. Tout ce que sait l'operator est dans PostgreSQL et dans les ressources du cluster. La sauvegarde qui compte est la base : vulnérabilités, décisions, exceptions, journal d'audit, archive des preuves, utilisateurs et certificat de licence y sont tous.

```
kubectl -n attestkeep exec statefulset/attestkeep-postgresql -- \
  sh -c 'pg_dump -U "$POSTGRES_USER" "$POSTGRES_DB"' > attestkeep.sql
```

Si vous utilisez votre propre PostgreSQL, vos sauvegardes existantes le couvrent déjà. Un snapshot du PVC de la base fournie fonctionne aussi si votre storage class le permet. À côté de la base, sauvegardez trois Secrets :

- `attestkeep-evidence-key`, la clé de signature des preuves, créée par l'operator. Si elle est perdue, une nouvelle clé est générée, et les checkpoints et documents signés avec l'ancienne ne se vérifient plus avec la clé actuelle de l'installation. L'historique reste. À partir de la 1.4.0, c'est aussi l'identité de licence : la perdre signifie que le contrôle quotidien suivant reçoit `401` et que le cluster tourne en Community jusqu'à une nouvelle activation.
- `attestkeep-data-key`, générée par le chart sauf si vous avez fourni `api.dataKey.existingSecret`. Si elle est perdue, les tokens de gestionnaires de tickets stockés ne se déchiffrent plus, et chaque canal de tickets doit recevoir de nouveau son token.
- `attestkeep-session`, la clé de session. Si elle est perdue, tout le monde se reconnecte.

Le certificat du webhook n'est pas dans la liste, parce que l'operator le réémet. La clé d'activation non plus : réinstaller dans le même cluster donne la même empreinte et réutilise l'emplacement de licence existant. Avec l'intégration Falco

active, l'operator crée aussi `<release>-falco-mtls`. Son certificat client se renouvelle depuis Falco > Falco integration.

Settings > Backup exporte aussi un bundle JSON de la configuration que vous avez montée à la main : canaux de notification, décisions de triage, liste des utilisateurs, policies et destinations SIEM. Il sert à monter un second cluster à l'identique. Les identifiants ne sont jamais dans le bundle, et les destinations SIEM importées arrivent désactivées. L'import montre d'abord un aperçu, les enregistrements existants l'emportent toujours, et les utilisateurs sont listés, pas créés. À partir de la 1.3.1, chaque export et import figure sous *Backup history*. Le bundle ne remplace pas le dump de la base.

Une restauration ne cache pas ce qui a été perdu. Au premier passage après une restauration, l'operator trouve le trou de heartbeat depuis le dump et enregistre toute la durée comme une période où le contrôle ne fonctionnait pas. Un paquet de preuves pour cette période montre le trou comme toute autre panne. Restaurer un dump dans la même version de l'operator ou une plus récente est pris en charge. Restaurer un dump plus récent dans un operator plus ancien ne l'est pas.

Disponibilité

Operator et webhook : 2 réplicas, anti-affinité préférée, PodDisruptionBudget minAvailable 1. À partir de la 1.4.0, un réplica qui s'arrête se vide : son hook preStop attend `webhook.preStopSleepSeconds` (5 par défaut) pendant que le cluster cesse de lui envoyer du trafic, puis il garde son écouteur ouvert cinq secondes de plus et termine les revues en cours. Le délai de terminaison du pod est cette valeur plus 30 secondes. Augmentez `replicaCount` pour monter en charge. Le travail du contrôleur est sérialisé par un advisory lock, c'est donc sans risque. Scanner : 1 réplica. Le chart refuse plusieurs réplicas sur un volume de cache ReadWriteOnce et dit pourquoi. PostgreSQL fourni : un seul réplica, par conception. Pour la réplication et des sauvegardes gérées, pointez `externalDsn` vers votre propre PostgreSQL.

GitOps. Le contrôle a lieu à l'admission du pod, peu importe donc que kubectl, Argo CD, Flux ou un Job applique le changement. Si une synchronisation GitOps crée un pod dont l'image ne passe pas la policy, la synchronisation échoue avec le message de refus comme erreur.

Clusters isolés. Trivy et cosign sont intégrés à l'image. Copiez la base de vulnérabilités avec `oras copy ghcr.io/aquasecurity/trivy-db:2 registry.internal/mirror/trivy-db:2` et renseignez `trivy.dbRepository`. Activez `scan.cosignOffline` pour vérifier avec des clés sans le journal de transparence public. Une copie du catalogue KEV peut être compressée en gzip dans une ConfigMap nommée par `kev.existingConfigMap`, avec `kev.source` pointant vers `file:///etc/kev/<key>`. La licence demande soit une route vers lic.attestkeep.com:443, soit l'activation hors ligne (chapitre 12).

12. Licences

Une clé d'activation par achat, et une clé Community par compte. Une clé se lie à autant de clusters que son offre le permet, et vous pouvez déplacer un emplacement vous-même.

Activation

L'activation se fait une fois par cluster et occupe un emplacement. Au premier démarrage, l'operator génère sa clé de signature dans le cluster, garde la partie privée dans le Secret `attestkeep-evidence-key`, partagé par tous les réplicas, et n'envoie que la partie publique avec la demande d'activation. À partir de la 1.4.0, c'est la même clé qui signe les paquets de preuves, si bien que le certificat nomme la clé avec laquelle un auditeur les vérifie. Une installation en ligne activée avant la 1.4.0 continue avec la paire de clés de son activation et passe à la clé unique à sa prochaine activation. La demande porte la clé d'activation, une empreinte du cluster (un hash de l'UID du namespace kube-system, ni un nom ni une adresse), le nom du cluster s'il est défini, et la version de l'operator. La réponse est un certificat signé, valable trente jours au plus. L'operator le vérifie dans le cluster avec une clé publique intégrée au binaire.

À partir de la 1.3.0, l'operator n'accepte qu'un certificat qui porte le plafond de nodes. Un certificat émis par un ancien service de licences est refusé avec "re-activate with a key issued by licence service 1.3.0 or later".

Si l'operator n'atteint pas le serveur de licences, il le dit au lieu de déclarer la clé invalide. Un certificat auquel l'operator ne fait pas confiance, une connexion refusée ou expirée et un nom qui ne se résout pas reçoivent un `502` avec le serveur et la cause. La console affiche "The licence server could not be reached: its certificate is not trusted, or the connection failed. If it uses a private CA, set license.caBundle in the chart." Une réponse `5xx` du serveur de licences devient un `503` avec "The licence server answered with an error. Try again later." La clé n'apparaît jamais dans un message. `license.caBundle` reçoit des certificats PEM auxquels le client de licence fait confiance en plus des racines du système, pour un proxy qui inspecte le TLS ou un serveur de licences sous CA privée. Posez-le avec `--set-file license.caBundle=ca.pem`. La vérification des certificats reste active.

Le contrôle quotidien

Une fois par jour, l'installation envoie trois valeurs : l'identifiant de licence, l'empreinte du cluster et la version de l'operator, signées avec la clé de l'activation. À partir de la 1.4.0, un seul réplica de l'operator fait le contrôle, et le scanner ne contacte plus le serveur de licences. La réponse est un état, plus un nouveau certificat si quelque chose a changé ou si le certificat en cours arrive à terme. Rien sur vos images, workloads, policies ou vulnérabilités n'est envoyé. Il n'y a pas de canal de télémétrie ni de réglage pour en ajouter un. Quel que soit le nombre de pods de l'operator, un cluster fait cet appel une fois par jour.

Quand le service de licences est injoignable

L'installation continue avec son certificat jusqu'à son expiration. Un contrôle échoué est retenté au bout de quatre heures. Quand un certificat en ligne a moins d'une semaine devant lui, la console avertit et la notification `license_expiring` part. Si le certificat expire, l'installation continue en Community Edition. Elle ne s'arrête pas et ne se met pas à refuser des déploiements. Une réponse `revoked` ou `released` s'applique au contrôle suivant.

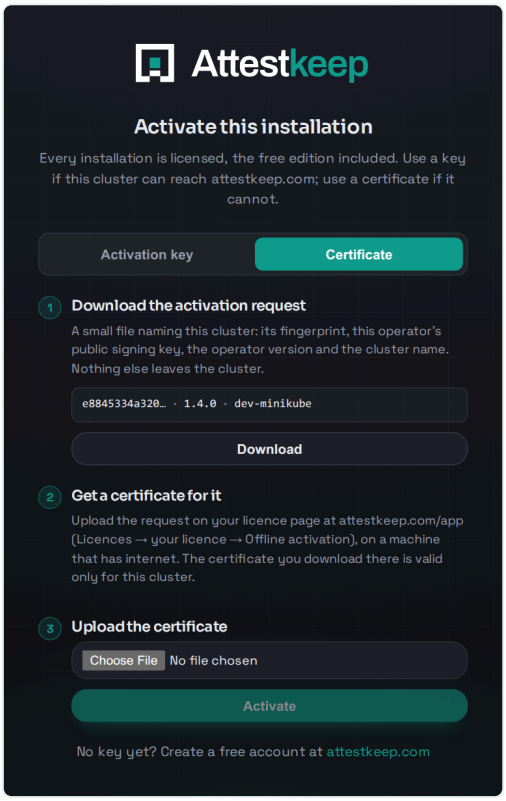
Emplacements de cluster

Chaque activation occupe un emplacement jusqu'à sa libération. Vous libérez les emplacements vous-même depuis votre page de licences sur attestkeep.com. Un emplacement libéré est disponible tout de suite, et l'installation qui l'occupait sort à son prochain contrôle quotidien. Réinstaller dans le même cluster n'occupe pas de second emplacement, l'empreinte étant la même. Si vous n'en avez plus, l'erreur nomme les clusters qui occupent vos emplacements et la date d'activation de chacun.

La console peut aussi rendre une licence : *Deactivate* sur la page Usage and License, après confirmation. En ligne, cela libère l'emplacement du cluster. Hors ligne, cela supprime le certificat du cluster et laisse au support Attestkeep le soin de libérer l'emplacement, parce qu'un cluster hors ligne ne se manifeste jamais.

Activation hors ligne

Un cluster qui n'atteindra jamais le service de licences est activé avec un certificat au lieu d'une clé. L'écran d'activation et la page Usage and License ont tous deux un onglet *Certificate* pour cela.



L'onglet *Certificate* en trois étapes. Téléchargez la demande d'activation, qui désigne ce cluster par son empreinte, la clé de signature publique de l'opérateur, la version de l'opérateur et le nom du cluster. Déposez-la sur votre page de licence sur attestkeep.com depuis une machine qui a accès à Internet. Chargez le certificat que vous recevez en retour.

Une installation activée hors ligne avant la 1.4.0 garde sa licence, mais son certificat nomme l'ancienne paire de clés, et un téléchargement de preuve signée est donc refusé jusqu'à ce qu'une nouvelle demande d'activation, faite depuis Usage and License, soit échangée contre un nouveau certificat. La page Evidence et le refus le disent.

Le certificat est le même document signé que celui du chemin en ligne, marqué hors ligne. L'operator vérifie la signature, l'empreinte du cluster et sa propre clé avant de le stocker. Il ne fait aucun contrôle quotidien pour lui et avertit trente jours avant son expiration. Un certificat pour un autre cluster ou une autre clé, ou un certificat expiré, est refusé avec la raison. Pour une installation dont personne n'ouvre la console, `license.existingCertificateSecret` nomme un Secret qui contient le certificat sous la clé `certificate`.

La page de licences sur attestkeep.com indique si l'activation hors ligne est disponible pour votre licence. Dans la grille tarifaire publiée, elle vient avec Enterprise en engagement annuel, ou dans le cadre d'un accord.

Le plafond de nodes

Chaque offre autorise un nombre de nodes Ready par cluster. L'operator compte les nodes Ready du cluster tous les `nodes.pollInterval` (5 minutes par défaut). Le nombre reste dans le cluster. Le contrôle quotidien envoie toujours les trois mêmes valeurs. Le plafond lui-même est dans le certificat.

Dépasser le plafond n'arrête rien. La console et une notification le disent, avec une date 14 jours plus tard (`nodes.graceDays`). Si le cluster est toujours au-dessus au bout de ces 14 jours, les noms d'image pas encore vus dans la période sont refusés à l'admission avec "node ceiling of N exceeded", et un nom refusé n'est pas compté. Il en va de même à partir de la 1.4.0 pour le quota d'images : un nom refusé parce que le quota du mois est atteint n'est pas compté, et le compteur s'arrête à la limite avec *Allowance reached*. Sur la carte de licence de la page Usage and License, une tranche d'images pleine affiche *Allowance reached* avec les dates propres de la licence, et non *Expired*, et l'e-mail du

quota dit que les nouveaux noms d'image sont refusés à l'admission, pas qu'ils ne sont plus scannés. Tout ce qui est déjà compté continue de tourner, d'être scanné et contrôlé. Revenir sous le plafond, avec moins de nodes ou plus de nodes sur la licence, remet le compteur à zéro tout de suite. Un relevé qui ne parvient pas à compter les nodes ne change rien. Aucune des deux valeurs du chart n'assouplit le plafond.

Quand une licence prend fin

L'installation continue en Community Edition quand l'abonnement se termine et que le certificat expire, quand le certificat ne se vérifie plus ou s'avère appartenir à un autre cluster, ou quand vous le désactivez. Community, c'est 1 cluster, 1 framework, 10 noms d'image par mois et 1 node par cluster. Rien n'est téléchargé et aucun nouveau certificat n'est émis. Le scan, l'admission et les preuves continuent de fonctionner. Dès ce moment, la page Usage and License et le tableau de bord affichent le quota Community du mois en cours, les mêmes chiffres que ceux qu'applique l'admission. Les mois précédents gardent le quota qu'ils avaient. Résilier en cours de période ne coupe rien le jour de la résiliation. Vous gardez ce que vous avez payé jusqu'à la fin de la période.

Choisir les frameworks. Si des preuves ont été produites pour plus de frameworks que la licence n'en couvre désormais, le prochain owner ou administrateur qui se connecte doit choisir lesquels garder, dans la limite du nombre de la licence. Tant que personne n'a choisi, aucun des frameworks déjà utilisés n'est ouvert à un nouveau paquet, et les autres utilisateurs sont informés qu'un administrateur doit choisir. Les frameworks non retenus sont fermés aux nouveaux paquets. Chaque paquet déjà produit reste téléchargeable. Le choix tient jusqu'à ce que l'offre ou le nombre de frameworks de la licence change, et il est inscrit au journal d'audit.

13. Éditions et prix

Chaque fonction de sécurité est dans l'édition gratuite. Le scan, la vérification des signatures, le contrôle à l'admission et les paquets de preuves sont complets en Community, et chaque contrôle de policy, y compris deny until scanned et `failurePolicy: Fail`, y fonctionne. La deny list, le take down, What if et l'export SIEM aussi. Les offres payantes ajoutent de la capacité et l'authentification unique.

Édition	Par mois, HT	Par an, HT	Clusters	Frameworks	Noms d'image / mois, par cluster	Nodes / cluster	En plus
Community	0 USD	0 USD	1	1, au choix	10	1	Gestionnaire de tickets public
Team	600 USD	6 000 USD	1	1	30	6	Authentification unique, support par e-mail
Business	1 800 USD	18 000 USD	3	3	50	10	Tout Team, support prioritaire par e-mail
Enterprise	3 600 USD	36 000 USD	5	Les treize	75	20	Activation hors ligne pour les clusters isolés, support par e-mail

Une année coûte dix mois. Options des éditions payantes, par mois : cluster supplémentaire +300 USD (jusqu'à dix clusters par commande), framework supplémentaire +120 USD, 10 noms d'image de plus par mois +30 USD pour chaque cluster couvert par la licence, node supplémentaire +50 USD pour chaque cluster couvert par la licence, un à la fois, jusqu'à 50 nodes par cluster. Les prix annuels valent dix fois les prix mensuels. En Community, un palier de volume coûte +300 USD par mois et par cluster. Les nodes supplémentaires ne sont pas vendus en Community.

Plus de dix clusters, aucun plafond de clusters ou aucun plafond de nodes relèvent d'un accord sur mesure, pas d'un prix catalogue.

Les prix s'entendent hors TVA. La commande et le paiement passent par Polar, merchant of record. La carte est saisie sur la page de paiement de Polar et n'atteint jamais les serveurs d'Attestkeep. Les taxes dues y sont calculées d'après votre adresse de facturation et affichées avant le paiement. Une entreprise assujettie à la TVA dans l'UE déclare elle-même la TVA en autoliquidation. La licence et le support viennent d'Attestkeep.

Il n'y a pas d'essai limité dans le temps. Community Edition a toutes les fonctions de sécurité et n'expire pas, si bien que vous pouvez tester le produit sur votre propre cluster aussi longtemps que nécessaire.

L'unité comptée est le nom d'image : registry et dépôt, sans tag ni digest. Tout ce qui tourne dans le cluster compte, images tierces comprises. Quand le quota est épuisé, seuls les nouveaux noms d'image sont refusés à l'admission. Chaque image déjà comptée dans la période continue d'être scannée et contrôlée.

14. Sécurité, support et votre interlocuteur

Intégrité des releases

Chaque image de release est construite pour linux/amd64 et linux/arm64 et signée avec cosign sur son digest, en keyless via l'OIDC de GitHub et avec la clé publiée sur docs.attestkeep.com/cosign.pub. Le scanner et le vérificateur de signatures sont intégrés à l'image et ne sont pas téléchargés pendant le fonctionnement. Le code source n'est pas public.

Signaler un bug ou une vulnérabilité

Les bugs vont sur le gestionnaire de tickets public github.com/attestkeep/attestkeep-docs, pour toutes les éditions. Les vulnérabilités passent par le formulaire d'avis privé du même dépôt ou par security@attestkeep.com. Un signalement reçoit un accusé de réception d'une personne sous trois jours ouvrés, et les problèmes corrigés sont publiés comme avis dès qu'une release contient la correction. Le règlement (UE) 2024/2847 demande au fabricant de désigner un point de contact unique pour les signalements de vulnérabilités. Le nôtre est security@attestkeep.com, et les mêmes coordonnées figurent sur chaque paquet de preuves.

Support selon l'offre

Offre	Support
Community	Gestionnaire de tickets public
Team	E-mail
Business	E-mail, prioritaire
Enterprise	E-mail
Accord sur mesure	Selon l'accord

Attestkeep ne publie de délai de réponse garanti pour aucune offre.

À qui vous avez affaire

Nom commercial	Attestkeep
Raison sociale	Ecavist Technologies, entreprise enregistrée en Turquie
Adresse enregistrée	Dörtyol Küme Evleri No: 369, İç Kapı No: 3, Işıklar, Merkez, Çanakkale, Türkiye
Notifications juridiques, contrats, facturation	legal@attestkeep.com
Support et avant-vente	support@attestkeep.com
Signalements de sécurité	security@attestkeep.com

Les documents contractuels (conditions d'utilisation, contrat de vente à distance, avis de confidentialité, conditions de traitement des données, sous-traitants) sont sur attestkeep.com. En cas de désaccord entre une page en langage clair et un accord, l'accord prévaut.

Annexe. Sources vérifiées pour ce manuel

Version du produit	Attestkeep 1.4.0 (chart 1.4.0, appVersion 1.4.0). Notes de version datées du 28 octobre 2026, image et chart publiés le 29 octobre 2026
Image publiée	`ghcr.io/attestkeep/attestkeep-k8s:1.4.0` `sha256:bfe0299695e38779f02ac24d805088a373d0a754fd7f0f552751e0d9fa3f05e8`
Chart publié	`oci://ghcr.io/attestkeep/charts/attestkeep` version 1.4.0 `sha256:8d6c4ba1e3ff790f6394e83eb34e5a740eeeecb0b4148e5c7e48683ca9bc610f`
Installation photographiée	Une installation dont la console indique la version 1.4.0, sur Kubernetes v1.35.1 avec deux nodes, en Enterprise Edition, avec Falco 0.45.0 connecté en mutual TLS depuis son propre `http_output`. Elle exécute une image construite à partir des sources de la 1.4.0, pas l'image publiée ci-dessus. Photographiée le 29 octobre 2026 en thème sombre, console en anglais.
Paquet de preuves montré	`PS-SOC2-DEVMINIKUBE-20261001-20261029-DA7379`, généré le 29 octobre 2026 à 19:31 UTC par la version 1.4.0
Documentation	Pages docs.attestkeep.com pour la 1.4.0 : releases, install, upgrade, what runs, admission, configuration, deny list, runtime events, SIEM export, licensing, backup, using the console, policies, try a policy before enforcing it, compliance, issue trackers et operations
Prix	attestkeep.com/pricing, USD, hors TVA. Les chiffres n'ont pas changé depuis l'édition 1.3.2 et ont été comparés à la grille du site le 29 octobre 2026.

Documentation : docs.attestkeep.com · Notes de version : docs.attestkeep.com/releases · Vérificateur des paquets de preuves remis : attestkeep.com/verify.html

Ce qui a changé depuis la 1.3.2

Version	Ce qui a changé
1.4.0, 28 octobre 2026	La deny list, vérifiée avant le break-glass et avant toute règle de policy, avec un mode `denyList` sur chaque policy, What if pour une entrée, des correspondances par workload, les workloads concernés et le take down à la main avec rétablissement (chapitre 7). Le contrôle `attestkeep-binscan` lit les exécutables dans les images, dans des limites de parcours. L'export SIEM vers cinq types de destination (chapitre 11). La vérification de signature par clé avec `signatureKeys`. `license.caBundle`, et les erreurs du serveur de licences signalées comme telles. Un arrêt runtime exige `rbac.allowEnforce` quoi qu'il soit accordé par ailleurs. Les réplicas restent prêts pendant une panne de base et se vident à l'arrêt. Les dry runs ne laissent rien. Un nom d'image refusé pour le quota n'est pas compté, et une tranche pleine affiche Allowance reached. Sections 13 et 14 dans la preuve, vulnérabilités comptées une fois par digest. Une seule clé pour la licence et la preuve. Go 1.26.10, Trivy 0.76.1, cosign 3.1.4. Migrations 0036 à 0046.
1.3.7, 15 octobre 2026	Les containers ajoutés avec `kubectl debug` passent par l'admission (chapitre 3). La page Images place chaque image dans l'un de quatre états, avec la nouvelle tuile Not compliant, et Blocked suit la dernière revue d'admission. Les exécutions What if sont conservées quatre-vingt-dix jours sous Security > What-if history, avec des téléchargements PDF et JSON qui sont une projection, pas une preuve. What if sur chaque ligne de policy, Scan now pour les images qu'un rejeu a trouvées sans scan, une confirmation avant que Cancel ne jette des modifications, Apply to cluster réservé aux comptes autorisés à écrire des policies, et une liste unique de noms de frameworks. L'operator est compilé avec Go 1.26.9 et `golang.org/x/net` 0.60.0, qui corrigent des vulnérabilités publiées que l'operator atteint. Migrations 0033 à 0035.
1.3.6, 8 octobre 2026	La page Images ne compte que les pods admis. Le filtre d'action de Falco Events propose les cinq effets. Corrections de mise en page de la barre latérale et de la liste des preuves à 1366 et 1600 px, et de la jauge d'usage. Le formulaire New policy propose les délais de SLA et de triage documentés.
1.3.5, 8 octobre 2026	Une policy Falco réglée sur Stop the pod est étiquetée ainsi partout, et un arrêt de pod s'écrit `policy:<nom>` dans le paquet de preuves. Le cluster est nommé par `clusterName` du chart. L'Audit Log utilise l'heure locale. La tuile Vulnerability database indique quand elle a vérifié pour la dernière fois et propose Update now. La tuile du webhook ne compare plus la latence à un budget fixe.
1.3.4, 8 octobre 2026	Corrections de la preuve : les détails du break-glass viennent de la révision de policy en vigueur à l'époque, les décisions de triage portent la gravité de la CVE, et la vue de paquet de la console compte correctement les revues refusées.
1.3.3, 7 octobre 2026	Les migrations tournent sous un verrou de base, si bien qu'une installation neuve ou une mise à niveau avec migrations ne redémarre plus un pod une fois.