Attestkeep

Produkthandbuch

Container-Image-Sicherheit für Kubernetes. Attestkeep scannt Images, prüft ihre Signaturen, entscheidet bei der Admission, ob ein Pod starten darf, weist zurück, was Ihr Team auf die Deny List gesetzt hat, und führt über jede Entscheidung einen signierten Nachweis. Dieses Handbuch beschreibt, was installiert wird, wie man es betreibt und was es kostet.

Attestkeep 1.4.0 · Helm-Chart `oci://ghcr.io/attestkeep/charts/attestkeep` Version 1.4.0 · Stand 29. Oktober 2026.

Die Bildschirmfotos entstanden am 29. Oktober 2026 in einer Installation, deren Konsole Version 1.4.0 meldet, als Enterprise Edition auf Kubernetes v1.35.1 mit angebundenem Falco. Die Konsole war auf Englisch eingestellt, so wie dieses Handbuch ihre Bezeichnungen nennt. Mehr dazu im Anhang. Befehle, Einstellungen und Grenzwerte stammen aus der Produktdokumentation auf docs.attestkeep.com, aus dem Chart und aus der Konsole. Nichts in diesem Handbuch ist nur zur Illustration erfunden.

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

Inhalt

1. Was Attestkeep ist

2. Was in Ihrem Cluster läuft

3. Wie eine Admission-Entscheidung entsteht

4. Installation, Upgrade und erster Start

5. Die Konsole

6. Admission-Policies

7. Die Deny List

8. Runtime-Events aus Falco

9. Nachweisen, dass durchgesetzt wird

10. Nachweispakete und Vertrauensmodell

11. Betrieb, SIEM-Export und Backup

12. Lizenzierung

13. Editionen und Preise

14. Sicherheit, Support und Ihr Vertragspartner

Anhang. Wogegen dieses Handbuch geprüft wurde

1. Was Attestkeep ist

Attestkeep ist ein Kubernetes-Operator. Er scannt die Images Ihrer Workloads, prüft ihre Signaturen und Attestations und antwortet dem API-Server bei der Admission mit Zulassen oder Ablehnen, nach einer Policy, die Sie schreiben. Jede Entscheidung wird aufgezeichnet. Aus diesen Aufzeichnungen entsteht ein Bericht pro Zeitraum, den ein Auditor mit eigenen Werkzeugen prüfen kann.

Alles läuft in Ihrem Cluster. Ein Helm-Release installiert den Operator, seine API, die Konsole und die Datenbank in einen Namespace, der Ihnen gehört. Image-Namen, Befunde und die Topologie des Clusters verlassen den Cluster nicht. Die Lizenzprüfung sendet einmal am Tag drei Werte.

Was es tut

- **Schwachstellen-Scan.** Trivy ist im Image enthalten. Befunde werden nach Schweregrad aufgelistet, mit der korrigierten Version, sofern es eine gibt.
- **Signaturprüfung.** cosign-Prüfung per Schlüssel oder keyless und in-toto-Attestations mit Höchstalter. Ab 1.4.0 kann eine Policy die Public Keys auflisten, die sie akzeptiert. Dann zählt nur eine Signatur, die einer davon bestätigt.
- **Admission Control.** Schwellen nach Schweregrad, Regeln für Tags und Registries, Digest-Pinning, Pflicht zu Signatur und Attestation, fünfzehn Kontrollen zur Härtung von Workloads, eine Sperre auf Basis des CISA-Katalogs Known Exploited Vulnerabilities, eine Sperre für Basis-Betriebssysteme ohne Herstellersupport, eine SLSA-Build-Provenance-Sperre, die prüft, wer signiert hat, VEX-Aussagen des Herstellers und strengere Regeln für Pods, die privilegiert oder von außen erreichbar laufen.
- **Deny List.** Wenn jemand entscheidet, dass ein Image, eine CVE, eine Paketversion oder eine ausführbare Datei nicht laufen darf, hält ein einziger Eintrag das fest. Er wird vor Break-Glass und vor jeder Regel der Policy geprüft, in jedem Namespace, dessen Policy die Liste durchsetzt. Ein zweites Werkzeug liest die ausführbaren Dateien in jedem Image, sodass auch eine aus dem Quellcode gebaute Runtime ohne Paketeintrag gefunden wird. Ein laufender Workload wird nie automatisch gestoppt. Ein Administrator kann ihn von Hand einmal per Take Down herunterfahren und wiederherstellen. Kapitel 7 beschreibt das.
- **Triage mit Ablaufdatum.** Sie können ein Risiko mit Begründung und Enddatum akzeptieren. Break-Glass lässt bei einem Vorfall ein benanntes Image durch. Beides wird aufgezeichnet.
- **Nachweispakete.** Ein signiertes Dokument pro Auditzeitraum, abgebildet auf den Kontrollkatalog Ihrer Wahl. Es sagt, was es belegen kann, und ebenso, was nicht.
- **Audit-Log.** Policy-Änderungen, Ausnahmen bei der Admission, Triage-Entscheidungen, Exporte von Nachweisen und Anmeldungen in einem Protokoll.
- **Benachrichtigungen.** Slack, Teams, E-Mail und ein generischer Webhook. Eine Benachrichtigung nennt den Cluster, nie seine Topologie.
- **Issue-Tracker.** Ein neuer kritischer Befund öffnet ein Issue in Ihrem GitHub, GitLab oder Jira, eines pro Image-Digest und CVE. Geschlossen wird es mit einem Kommentar, der den Grund nennt: behoben, per Triage bewertet oder Image ausgemustert. Nur *behoben* zählt als Behebung.
- **Runtime-Events aus Falco.** Ihr eigenes Falco sendet seine Events per Mutual TLS an den Operator, direkt oder über Falcosidekick. Jedes Event wird dem Image und dem Pod zugeordnet, aus dem es kam. Runtime-Policies, die Sie in der Konsole schreiben, entscheiden, was danach geschieht: nur aufzeichnen, benachrichtigen, ein Issue öffnen oder den Pod stoppen. Ein Runtime-Event ändert nie eine Admission-Entscheidung. Kapitel 8 beschreibt das.
- **SIEM-Export.** Abgelehnte Admissions, Treffer und Take Downs der Deny List, Runtime-Events, Änderungen an Policies und Triage, Downloads von Nachweisen, fehlgeschlagene Anmeldungen und neue kritische Befunde gehen in dem Moment an Ihr SIEM, in dem sie entstehen: Splunk, Microsoft Sentinel, Elasticsearch, Syslog oder ein generischer HTTP-Collector. Kapitel 11 beschreibt das.
- **Single Sign-on und Rollen.** Google, GitHub, GitLab, Okta, Microsoft Entra ID, Keycloak oder jeder OIDC-Anbieter, in den kostenpflichtigen Editionen. Rollen legen fest, wer eine Policy ändern darf.

Was es nicht tut

Attestkeep ist kein Runtime-Sensor. Es beobachtet keine Syscalls, Netzwerkflüsse oder Prozesse in einem laufenden Container. Diese Erkennung leistet Falco, Attestkeep empfängt nur, was Falco meldet. Attestkeep liefert Falco nicht aus, installiert, konfiguriert und beschreibt es nicht.

Es fährt laufende Workloads nicht von sich aus herunter. Ein Eintrag in der Deny List und eine durchsetzende Policy lehnen neue Pods ab. Was bereits läuft, wird nur gestoppt, wenn ein Mensch das entscheidet oder eine Runtime-Policy, die Sie geschrieben und der das Chart die Berechtigung gegeben hat.

Es ist kein Werkzeug für Cloud Posture und liest weder Ihr Cloud-Konto noch Ihre IAM-Policies. Es prüft weder Terraform noch Manifeste noch die Konfiguration des Clusters gegen einen Benchmark. Es prüft nicht, ob eine verwundbare Funktion je aufgerufen wird, und es liest weder Ihre Repositories noch öffnet es Pull Requests für Abhängigkeiten. Auf docs.attestkeep.com steht, was Sie für jeden dieser Punkte neben Attestkeep betreiben können.

Warum das Tor nicht aus freien Werkzeugen bauen?

Das geht. Es gibt gute Open-Source-Engines für Admission-Policies und Scanner im Cluster. Wenn Sie nur `:latest` sperren und bei kritischen CVEs ablehnen wollen, brauchen Sie Attestkeep nicht. Was einem solchen Aufbau meist fehlt, ist der Nachweis, mit dem Audits und Vorfallanalysen arbeiten:

- **Ein Entscheidungsprotokoll mit Integrität.** Jede Admission-Entscheidung wird beim Schreiben gehasht und unter signierten, verketteten Checkpoints versiegelt. Wird ein Urteil in der Datenbank geändert, nennt das nächste Nachweisdokument den Datensatz.
- **Entscheidungen hängen an der Policy, die sie traf.** Jeder Datensatz trägt den Hash der Policy, wie sie in diesem Moment war.
- **Lücken werden gemeldet.** Zeiträume, in denen der Webhook nicht lief oder nicht geprüft werden konnte, werden aufgezeichnet und im Nachweis gedruckt. Der Abgleichlauf listet auf, was in diesen Zeiträumen hereinkam.
- **Ein Produkt.** Scan-Ergebnisse, Signaturprüfungen, Admission, Deny List, Ausnahmen und Nachweispaket werden gemeinsam gepflegt. Ihr Team muss den Klebstoff dazwischen nicht pflegen.

2. Was in Ihrem Cluster läuft

Ein Admission-Webhook redet bei jedem Pod mit. Das hier ist die Bestandsliste dessen, was dazu gehört.

Objekt	Standard	Aufgabe
Deployment `attestkeep`	2 Replicas, bevorzugte Anti-Affinity über Nodes, PodDisruptionBudget minAvailable 1	Der Operator: Admission-Webhook (8443), Konsole und API (8080), Metriken (9090) und der Falco-Listener für Integrationen (8444), wenn `integrations.falco.enabled` aktiv ist. Controller-Arbeit läuft seriell über einen PostgreSQL Advisory Lock, Replicas skalieren also ohne Leader-Election-Lease. Ab 1.4.0 verlässt eine stoppende Replica den Service, bevor sie aufhört zu antworten: Ihr preStop-Hook wartet `webhook.preStopSleepSeconds` (Standard 5), dann beendet sie die laufenden Prüfungen.
Deployment `attestkeep-scanner`	1 Replica	Führt Trivy-Scans, die cosign-Prüfung und, wenn die Deny List es braucht, die Prüfung der ausführbaren Dateien `attestkeep-binscan` aus, alles außerhalb des Admission-Pfads. Er bekommt nie Admission-Verkehr und kontaktiert ab 1.4.0 nie den Lizenzserver.
StatefulSet `attestkeep-postgresql`	1 Replica, 20 GiB, postgres:16.15-alpine	Die mitgelieferte Datenbank. Wenn Sie `externalDsn` auf Ihr eigenes PostgreSQL setzen, wird dieses Objekt nicht angelegt.

Alle Container des Produkts laufen ohne Root (uid 65532), mit schreibgeschütztem Root-Dateisystem, ohne Capabilities und mit RuntimeDefault-seccomp. Die Requests liegen standardmäßig bei 100m CPU und 256Mi pro Produkt-Workload.

Clusterweit kommen vier CRDs der Gruppe `attestkeep.com` hinzu (ImageSecurityPolicy, ImageSecurityReport, AttestationRenewal, ImageSecurityUsage) und eine ValidatingWebhookConfiguration. Der Webhook deckt CREATE und UPDATE von Pods ab, einschließlich der Subressource `pods/ephemeralcontainers`, in die `kubectl debug` schreibt, mit 15 Sekunden Timeout und standardmäßig `failurePolicy: Ignore`. Es gibt keinen mutierenden Webhook. Das Produkt schreibt Ihre Workloads nie um.

Berechtigungen, die Sie kennen sollten

- Die ClusterRole liest Nodes (get, list). So zählt der Operator die Ready-Nodes gegen die Node-Obergrenze der Lizenz. Er behält nur die Zahl, nie einen Node-Namen.
- Secrets sind clusterweit nur mit `get` lesbar, nicht mit `list`. Der Operator kann ein Pull-Secret lesen, dessen Namen er in einer Pod-Spec gefunden hat. Er kann nicht auflisten, was ein Namespace enthält.
- `pods/eviction` wird nur mit `rbac.allowEnforce: true` oder `rbac.denyList.takedown: true` gewährt (beide standardmäßig false). Eine Runtime-Policy, die Pods stoppt, und `runtimeReconciliation: enforce` brauchen `rbac.allowEnforce: true`, ganz gleich, was die ClusterRole sonst erlaubt. Ohne diesen Wert werden sie abgelehnt und protokolliert. Die Take-Down-Berechtigung dient nur der Deny List.
- Die Scale-Subressource von Deployments, StatefulSets und ReplicaSets (get, update) wird nur mit `rbac.denyList.takedown: true` gewährt. Damit kann ein Administrator einen Workload auf null skalieren und wiederherstellen. Sie ändert, wie viele Replicas laufen, nie, was sie ausführen.
- Deployments, ReplicaSets, StatefulSets, DaemonSets, Jobs und CronJobs sind ab 1.4.0 in jeder Installation lesbar (get, list). Der Operator liest die Eigentümerkette über einem Pod, sodass eine Ablehnung dem Deployment zugerechnet wird und nicht seinem ReplicaSet. Das ist nur Lesezugriff.
- Der Operator legt Kubernetes-Events an: eine Warning am Workload, den eine Ablehnung durch die Deny List gestoppt hat, und an einer Policy mit einem Signaturschlüssel, der kein PEM Public Key ist. Er schreibt außerdem den Status seiner eigenen Policies, in dem die Condition `SignatureKeysValid` steht.
- Eine Role im Namespace erlaubt das Schreiben dreier benannter Secrets im Namespace des Operators: das Serving-Zertifikat des Webhooks, `attestkeep-evidence-key` und `<release>-falco-mtls`, das nur bei aktiver Falco-Integration existiert. Ab 1.4.0 ist `attestkeep-evidence-key` auch die Lizenzidentität der Installation (Kapitel 12).

- Eine zweite Role, `<release>—rule-validation`, erlaubt dem Operator, im eigenen Namespace Pods und ConfigMaps anzulegen, zu lesen und zu löschen und Pod-Logs zu lesen. Der Falco-Regeleditor startet damit einen einmaligen Validierungs-Pod. In anderen Namespaces kann er keine Pods starten.

Jede ausgehende Verbindung

Diese Tabelle ist für Ihren Firewall-Antrag. Es gibt kein Analytics-, Crash-Reporting- oder Telemetrie-SDK.

Ziel	Erforderlich?	Einstellung	Wenn gesperrt
lic.attestkeep.com:443	Ja, außer bei Offline-Aktivierung	`license.serverUrl`; `license.caBundle` für eine private CA oder einen TLS-aufbrechenden Proxy auf dem Weg	Eine Online-Aktivierung ist nicht möglich. Eine aktivierte Installation läuft mit ihrem Zertifikat weiter und fällt auf Community zurück, wenn es abläuft. Ab 1.4.0 führt nur eine Operator-Replica die tägliche Prüfung aus.
ghcr.io (Trivy-Schwachstellendatenbank):443	Für Scans	`trivy.dbRepository`, für einen Mirror	Die vorhandene Datenbank arbeitet weiter und veraltet. `attestkeep_vulnerability_db_age_seconds` zeigt, wie stark.
Ihre Image-Registries:443	Für Scans	Pull-Secrets aus den Pod-Specs	Scans dieser Images schlagen fehl. Die Admission folgt der Einstellung der Policy für nicht gescannte Images.
ghcr.io/attestkeep/attestkeep-k8s:443	Bei der Installation	`image.repository`, für einen Mirror	Pods können nicht starten.
Sigstore TUF und öffentliches Rekor:443	Nein	`scan.cosignOffline: true`	Mit Schlüsselsignatur und Offline-Modus bleibt die Prüfung lokal.
Release-Feed	Nein, standardmäßig aus	`updates.endpoint`	Nichts. Eine Installation ohne Endpunkt schaut nie nach.
Ihre SIEM-Ziele	Nein, von Ihnen eingerichtet	Settings > SIEM export	Events warten und werden erneut versucht. Was nach 24 Stunden noch nicht zugestellt ist, wird aufgegeben und auf der Seite gezählt.
Ihr OTLP-Collector, Slack-, Teams- oder Webhook-URLs, SMTP-Server, SSO-Anbieter	Nein, von Ihnen eingerichtet	Je Funktion	Diese Funktion schlägt fehl und protokolliert es. Ein Fehler bei Benachrichtigungen blockiert nie die Admission.

Falls unser Lizenzserver kompromittiert wäre

Er könnte Ihrem Cluster nichts befehlen. Das Lizenzprotokoll kennt zwei Arten von Antwort: ein Zertifikat und einen Verlängerungsstatus. Kein Codepfad lässt eine Serverantwort eine Policy, einen Namespace, eine Image-Regel, eine URL oder ein auszuführendes Programm wählen. Zertifikate sind mit Ed25519 signiert, werden gegen einen in das Binary einkompilierten Public Key geprüft und sind an den Fingerabdruck Ihres Clusters gebunden.

Ein feindlicher Server könnte *revoked* antworten. Die Installation fiele dann auf eine Admission zurück, die nur warnt, und das ist in der Konsole und in jeder Admission-Antwort sichtbar. Einen Workload platzieren oder eine Policy ändern könnte er nicht. Antwortet der Server gar nicht, passiert nichts: Netzwerkfehler gelten als Ausfall, und die Installation läuft mit ihrem Zertifikat, bis es abläuft.

3. Wie eine Admission-Entscheidung entsteht

Der Webhook antwortet aus dem Cache. Gescannt wird nie im Admission-Pfad, eine langsame Registry bremst also kein Deployment.

1. Der API-Server fragt nach einem Pod: wenn er angelegt oder geändert wird und wenn `kubectl debug` einem laufenden Pod einen Container hinzufügt.
2. Der Image-Tag wird zu einem Digest aufgelöst. Das Ergebnis bleibt eine Minute im Cache, eine Welle gleicher Deployments verursacht also einen einzigen Registry-Aufruf.
3. Scan-Ergebnis und Signatururteil für den Digest werden aus dem Cache gelesen.
4. Zuerst kommen die Regeln, die keinen Registry-Aufruf brauchen: die Registry-Allowlist, `:latest` und ein verlangter Digest.
5. Die Deny List wird geprüft, vor Break-Glass und vor jeder anderen Regel der Policy. Ein gelistetes Image wird abgelehnt, auch wenn sein Scan sauber ist und auch während Break-Glass offen ist (Kapitel 7).
6. Die Policy wird ausgewertet. Die Antwort lautet Zulassen oder Ablehnen, mit Begründung.
7. Die Entscheidung wird aufgezeichnet, auch wenn sie Zulassen lautet. Ein Nachweis, der nur Ablehnungen enthält, kann nicht zeigen, dass eine Kontrolle gearbeitet hat.

Keiner dieser Schritte wartet auf einen Scan. Wurde ein Digest noch nie gesehen, hat Schritt drei keine Antwort, und die Policy entscheidet, was passiert.

Falco-Events sind nicht Teil dieses Pfads. Ein Runtime-Event ändert nie, ob ein Pod zugelassen wird.

Dry Runs. Ab 1.4.0 wird `kubectl apply --dry-run=server` wie eine echte Anfrage entschieden und beantwortet und hinterlässt nichts: keine Ledger-Zeile, keinen Treffer der Deny List, kein Kubernetes-Event, keine Benachrichtigung und keine Metrik, keine Zählung gegen das Image-Kontingent und keinen Scan in der Warteschlange.

Wenn die Datenbank nicht antwortet. Eine Admission-Prüfung gibt ihrer Datenbankarbeit insgesamt zwei Sekunden. Danach entscheiden sofort die statischen Regeln: `:latest`, die Registry-Allowlist und ein verlangter Digest lehnen weiter ab. Aufzeichnung und Urteile, die den Cache brauchen, fallen zurück, und der Ausfall wird einmal protokolliert. Vor 1.4.0 verließ eine Replica den Service, solange die Datenbank nicht erreichbar war, und Admission-Antworten gingen verloren. Kapitel 11 hat die Einzelheiten.

Debug-Container. Ab 1.3.7 geht auch ein mit `kubectl debug` hinzugefügter Container durch den Webhook. Entschieden wird nur über den hinzugefügten Container. Sein Image wird gescannt und an der Policy gemessen, und eine durchsetzende Policy lehnt es ab wie jedes andere Image. Die vorhandenen Container des Pods werden nicht erneut geprüft. Ein Debug-Image zählt weder gegen das monatliche Image-Kontingent noch gegen die Node-Obergrenze, und die Begründung seiner Ledger-Zeile endet auf `; subresource ephemeralcontainers`. Vor 1.3.7 wurde ein solcher Container nie an den Webhook geschickt, egal welches Image er ausführte.

Die zwei Einstellungen, die Ihre Haltung bestimmen

Einstellung	Standard	Bedeutung
`webhook.failurePolicy` (Chart)	`Ignore`	Was der API-Server tut, wenn der Operator nicht antwortet. `Ignore` lässt Pods während eines Ausfalls ungeprüft zu. `Fail` lehnt ab, bis der Operator antwortet.
`admissionTiming` (Policy)	`AllowAndScan`	Was mit einem erstmals gesehenen Digest geschieht. `AllowAndScan` lässt ihn zu und stellt einen Scan in die Warteschlange. `DenyUntilScanned` lehnt ihn ab, bis ein Scan-Datensatz existiert. Es ist der übliche Partner von `Fail`.

Keine der beiden Wahlen ist für alle richtig. `Ignore` hält den Cluster während eines Operator-Ausfalls deploybar und hinterlässt eine Lücke im Nachweis. Das Nachweispaket meldet diese Lücke. `Fail` ergibt eine Kontrolle ohne Lücken, und ein Operator-Ausfall stoppt dann die Deployments. Beide Einstellungen gibt es in jeder Edition, Community eingeschlossen.

Was während einer Lücke hereinkam, läuft danach weiter. Der Abgleichlauf zur Runtime findet es. Nach Zeitplan (standardmäßig stündlich) vergleicht er die Image-ID jedes laufenden Containers mit dem Admission-Ledger. Ein Digest ohne Admission dahinter wird gemeldet, benachrichtigt oder unter `runtimeReconciliation: enforce` gestoppt, sodass er erneut durch den Webhook kommt.

> **Der erste Scan eines neuen Digests läuft im Hintergrund.** In einem neuen Cluster kann er einige Minuten dauern, weil der Scanner das Image ziehen muss und die Schwachstellendatenbank noch nicht im Cache liegt. Bis er fertig ist, steht das Image auf der Seite Images im Zustand *Unscanned*. Spätere Admissions dieses Digests nutzen den fertigen Bericht. *Rescan* startet einen Scan auf Anforderung.

4. Installation, Upgrade und erster Start

Bevor Sie beginnen

- Kubernetes 1.27 oder neuer.
- Die Berechtigung, eine ValidatingWebhookConfiguration anzulegen, bei der ersten Installation also cluster-admin.
- Etwa 20 GiB für das mitgelieferte PostgreSQL oder der DSN eines PostgreSQL, das Sie bereits betreiben. Der SIEM-Export braucht PostgreSQL 13 oder neuer, das mitgelieferte ist 16.
- Ein Attestkeep-Konto auf attestkeep.com und einen Lizenzschlüssel: den kostenlosen Community-Schlüssel oder den Ihres gekauften Plans.

Das Release vor der Installation prüfen

Jedes Release ist auf seinem Digest signiert. Die Prüfung per Schlüssel braucht nur Ihre Registry und den Public Key unter 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
```

Installieren

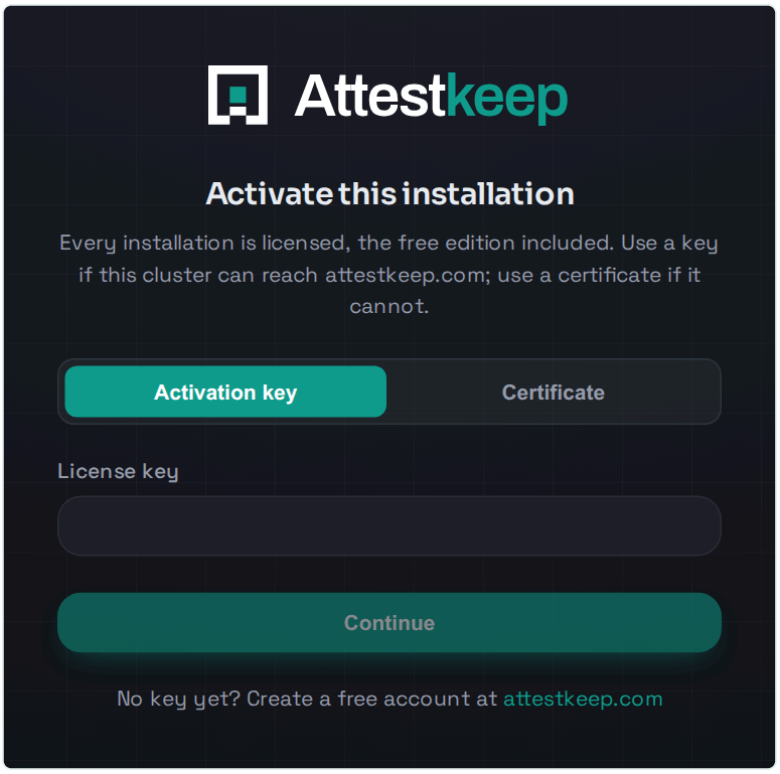
```
# installiert direkt aus der Registry; kein `helm repo add` nötig
helm install attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --version 1.4.0 \
  --namespace attestkeep --create-namespace \
  --set clusterName=production-eu \
  --wait
```

Setzen Sie `clusterName` schon bei der ersten Installation, auch für einen Test. Benachrichtigungen, Nachweisdokumente, das Dashboard und Settings > About benennen die Installation damit. Ohne den Wert greifen sie auf den Namen des ersten Nodes zurück. Den Namen wählen Sie selbst. Kein Hostname und keine Adresse verlassen den Cluster.

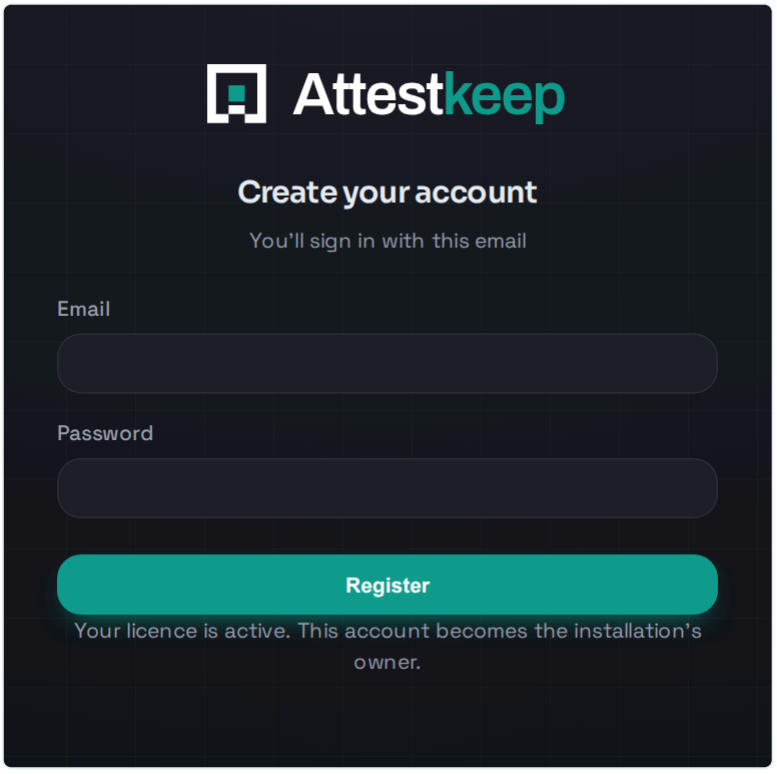
Prüfen, ob es läuft, und die Konsole öffnen

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

Öffnen Sie `http://localhost:8080`. Eine neue Installation zeigt einmalig zwei Einrichtungsseiten: die Aktivierung, dann das Owner-Konto. Danach öffnet die Konsole mit der Sicherheitsübersicht, die Kapitel 5 beschreibt.



Bildschirm 1, *Activate this installation*. Jede Installation ist lizenziert, die kostenlose Edition eingeschlossen. Im Reiter *Activation key* fügen Sie Ihren Schlüssel ein und wählen *Continue*. Der Schlüssel wird gegen ein signiertes Zertifikat getauscht, das der Operator im Cluster prüft. Ein Cluster ohne Verbindung zu attestkeep.com nutzt stattdessen den Reiter *Certificate* (Kapitel 12).



Bildschirm 2, *Create your account*. Die Zeile unter *Register* bestätigt, dass die Lizenz aktiv ist und dieses Konto Eigentümer der Installation wird. Sobald das Konto existiert, schließt sich der Bildschirm. Wer nach einer Installation mit Community-Schlüssel einen Plan kauft, aktiviert den bezahlten Schlüssel später auf der Seite Usage and License.

Erreicht die Aktivierung den Lizenzserver nicht, sagt die Seite das und nennt die Ursache. Ein Zertifikat, dem der Operator nicht vertraut (eine private CA ohne `license.caBundle`), eine abgelehnte oder abgelaufene Verbindung und ein Name, der sich nicht auflösen lässt, lauten alle "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." Vor 1.4.0 sahen dieselben Fehler wie ein ungültiger Schlüssel aus. Kapitel 12 führt die Meldungen auf.

Eine eigene Datenbank verwenden

Migrationen sind im Binary enthalten und werden bei der ersten Verbindung angewendet. Jede wird vermerkt und läuft genau einmal. Um eine bereits betriebene Datenbank zu verwenden, legen Sie den DSN in ein Secret statt auf die Kommandozeile:

```
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
```

Private Registries brauchen keine Konfiguration. Der Operator verwendet für Scan und Signaturprüfung die Image-Pull-Secrets, auf die der Pod verweist, und die seines Service Accounts.

Deinstallieren geht mit `helm uninstall`. Die Webhook-Konfiguration wird mit dem Release entfernt, die Admission-Prüfung endet also in diesem Moment. Das Volume der Datenbank wird nicht gelöscht.

Upgrade

Ein Upgrade ist `helm upgrade`. Die eingebauten Migrationen wenden an, was neu ist, und überspringen den Rest. `helm upgrade --reuse-values` funktioniert über diese Versionen hinweg: Werte, die nach Ihrer installierten Version dazukamen, nehmen ihre Chart-Standards an. Migrationen laufen nur vorwärts. Wo ein Schritt unten Migrationen nennt, führt der Weg zurück zur früheren Version über ein Datenbank-Backup, nicht über `helm rollback`.

Von → nach	Migrationen	Was zu tun ist
1.3.7 → 1.4.0	0036 bis 0046	Führen Sie genau dieses Upgrade mit `--set webhook.failurePolicy=Fail` aus und setzen Sie den Wert danach zurück, oder führen Sie es während eines Change Freeze aus. Eine Replica von 1.3.7 stoppt, ohne sich zu leeren, und für etwa eine Sekunde wird eine Anfrage an sie unter `Ignore` ungeprüft zugelassen. Ab 1.4.0 leert sich jede Replica beim Upgrade. Jede Migration wartet höchstens fünf Sekunden auf ihren Lock, dann startet die Replica neu und versucht es erneut. Zurück zu 1.3.7 geht es nur über ein Backup. Die neuen Chart-Werte sind alle optional, und Take Down bleibt aus, bis Sie `rbac.denyList.takedown: true` setzen. Die ClusterRole erhält Lesezugriff auf die Workload-Objekte über einem Pod. Bestehende Benachrichtigungskanäle sind nicht für *Deny list hit* abonniert: Haken Sie es dort an, wo Ablehnungen gemeldet werden sollen. Eine externe Datenbank muss für den SIEM-Export PostgreSQL 13 oder neuer sein. Eine vor 1.4.0 offline aktivierte Installation braucht eine neue Aktivierungsanfrage, bevor signierte Nachweis-Downloads funktionieren (Kapitel 12).
1.3.6 → 1.3.7	0033 bis 0035	Mit `helm upgrade` aktualisieren, nicht nur das Image tauschen: Die neue Webhook-Regel für `pods/ephemeralcontainers` steckt im Chart. Ab dieser Version wird `kubectl debug` mit einem Image, das die Policy nicht erlaubt, abgelehnt. 0033 legt die Tabelle für What-if-Läufe an, 0034 ergänzt die Images-Tabelle um zwei Spalten und 0035 füllt sie aus dem Ledger. Zurück zu 1.3.6 geht es nur über ein Backup. Keine neuen Chart-Werte.
1.3.2 → 1.3.6	Keine	Nichts zu tun. 1.3.3, 1.3.4, 1.3.5 und 1.3.6 bringen weder Migrationen noch neue Chart-Werte.
1.3.1 → 1.3.2	Keine	Nichts für den Operator. `helm rollback` auf 1.3.1 funktioniert. Wenn Sie die falcoctl-Werte aus dem Regeleditor von 1.3.1 übernommen haben, nehmen Sie `falco-rules:5` wieder in `install.refs` und `follow.refs` auf. Ohne den Eintrag installiert falcoctl Falcos eigene `/etc/falco/falco_rules.yaml` nicht mehr.
1.3.0 → 1.3.1	0026 bis 0032	Die Policy-Ressource hat kein Feld `spec.runtime` mehr. Entfernen Sie es aus Ihren Policy-Dateien, falls Sie es gesetzt haben: `kubectl apply` lehnt eine Policy, die es noch enthält, mit `unknown field "spec.runtime"` ab. Migration 0031 stellt Runtime-Regeln mit der entfernten Aktion auf Benachrichtigen mit Issue um. Neue Benachrichtigungskanäle sind für Runtime alert abonniert. Bestehende Kanäle behalten ihre Abonnements.
1.2.x → 1.3.0	0025	Den Cluster neu aktivieren. Ein Operator ab 1.3.0 braucht ein Zertifikat mit Node-Obergrenze und lehnt ein älteres mit "re-activate with a key issued by licence service 1.3.0 or later" ab.
1.1.x → 1.2.0	0021 bis 0024	Falco-Events, die über Falcosidekick mit Bearer-Token gesendet werden, kommen nicht mehr an, bis Falco per Mutual TLS verbunden ist (Kapitel 8). Das Löschen eines Benutzers deaktiviert das Konto jetzt.

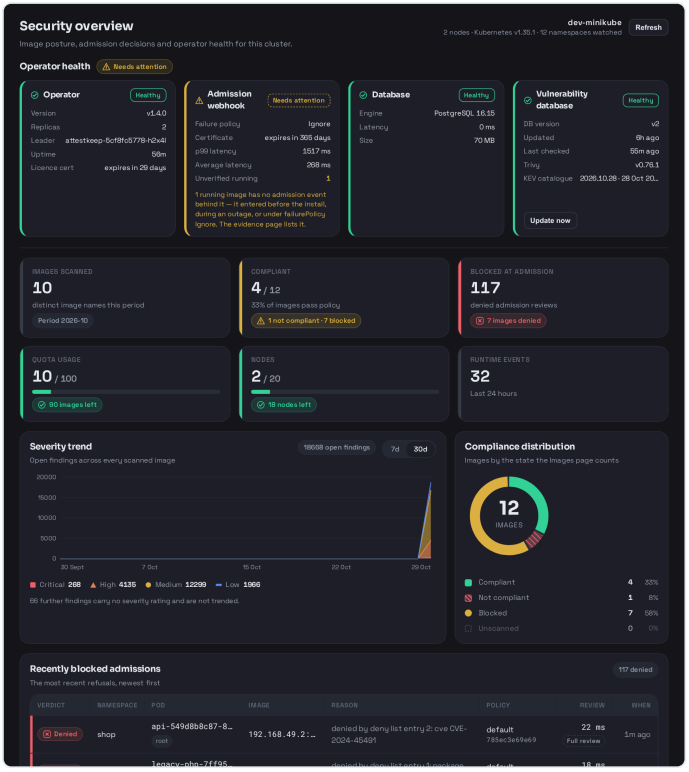
Eine Installation, die ihr RBAC selbst verwaltet, nimmt das Secret `<release>-falco-mtls` in die Role auf, bevor sie die Falco-Integration einschaltet.

5. Die Konsole

Die Konsole kommt mit dem Release. Dasselbe Helm-Release, das den Operator installiert, liefert sie aus. Sie lädt keine Schriften, Skripte oder Analytics von außen. Es gibt sie auf Englisch und Türkisch, mit hellem und dunklem Thema und einem Hilfebereich auf jeder Seite. Dieses Handbuch zeigt die englische Oberfläche und nennt Menüs und Schaltflächen so, wie sie dort stehen.

Die Seitenleiste hat vier Gruppen. *Security* enthält Images, Vulnerabilities, Policies, Deny list und What-if history. *Falco* enthält Policies, Rules, Events und Falco integration und erscheint, wenn die Falco-Integration aktiv ist. *Compliance* enthält Compliance evidence und das Audit Log. *Settings* enthält Organisation, Appearance, My Notifications, Webhooks, Scanning, SMTP, Single Sign-On, Logging & telemetry, SIEM export, Usage and License, Users, Backup und About.

Die Sicherheitsübersicht



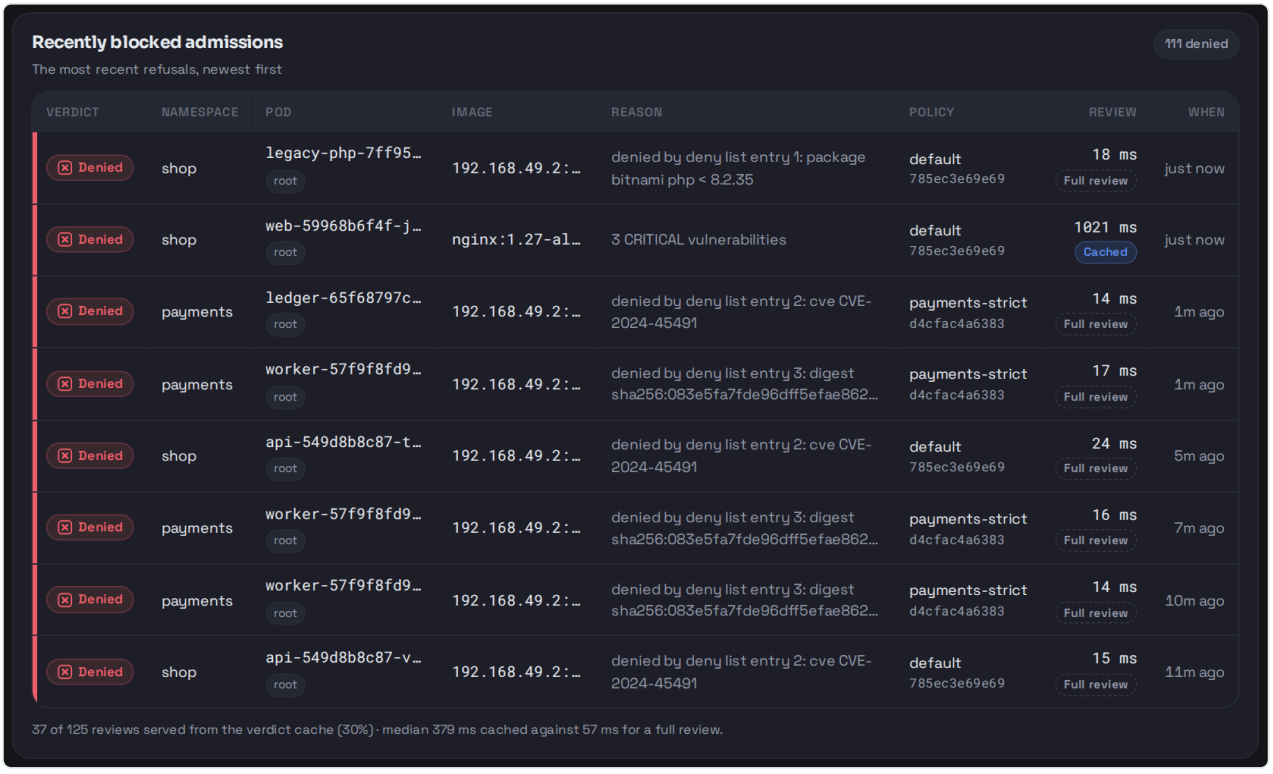
Die Sicherheitsübersicht der für dieses Handbuch fotografierten Installation: Version 1.4.0 mit zwei Replicas, zwei Nodes, Kubernetes v1.35.1. Die Kachel *Admission webhook* braucht Aufmerksamkeit, weil hinter einem laufenden Image kein Admission-Eintrag steht, und sie nennt die möglichen Ursachen. Darunter: 10 gescannte Image-Namen in diesem Zeitraum, 4 von 12 Images konform, 1 nicht konform und 7 blockiert, 117 abgelehnte Admission Reviews, *Quota usage* 10 von 100, *Nodes* 2 von 20 und 32 Runtime-Events in den letzten 24 Stunden. Die Compliance-Verteilung zählt dieselben vier Zustände wie die Seite Images.

Ab 1.3.5 lösen die beiden Kacheln keine Warnung mehr von sich aus. Die Webhook-Kachel vergleicht die Latenz nicht mit einem festen Budget, denn die erste Prüfung eines Images löst seinen Digest bei der Registry auf und dauert länger als spätere Prüfungen, die aus dem Cache antworten. Der Operator erneuert die Schwachstellendatenbank und den KEV-Katalog alle sechs Stunden. Ein Administrator kann *Update now* drücken, um sofort herunterzuladen. Die Kachel sagt dann, ob sich die Datenbank geändert hat, schon aktuell war oder warum der Download scheiterte, und die Anfrage landet als `vulnerability_db_refresh` im Audit-Log.

Die Karte Nodes kam mit 1.3.1. Solange der Cluster über seiner Obergrenze liegt, zeigt der Streifen, seit wann, und ab welchem Datum neue Image-Namen abgelehnt werden. Ist dieses Datum vorbei, meldet er, dass die Durchsetzung läuft. Was schon läuft, stoppt in keinem der beiden Fälle.

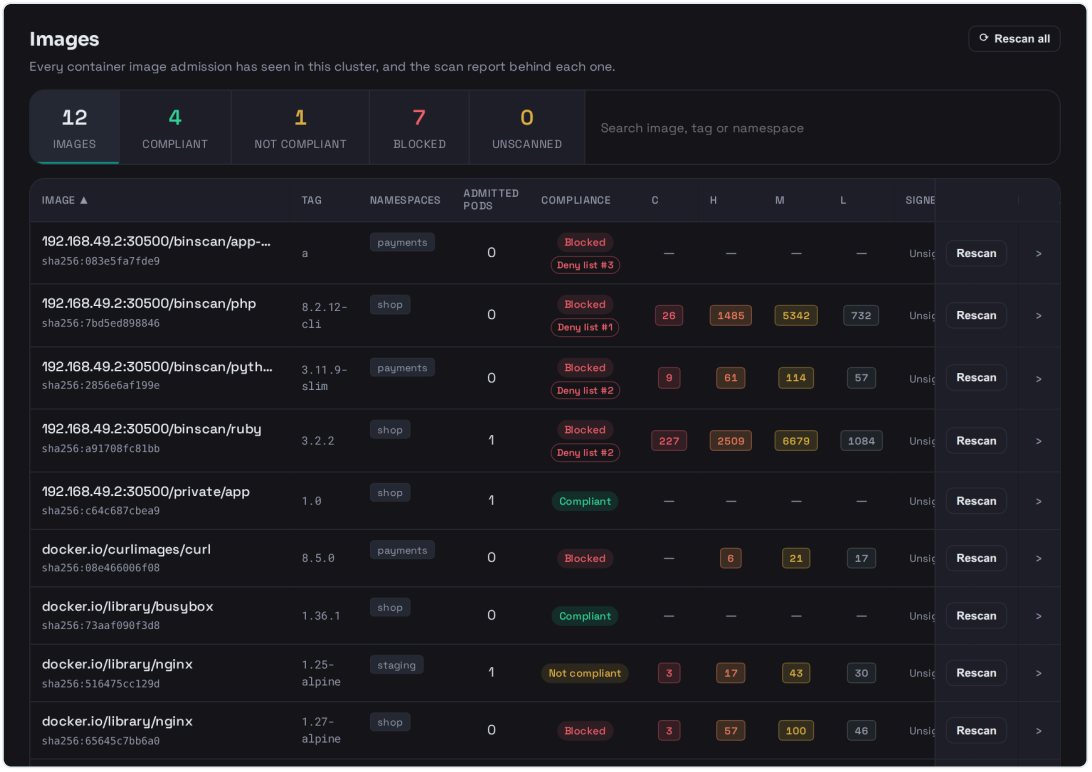
Ab 1.4.0 zählen die Compliance-Kachel, ihre Chips und die Verteilung dieselben vier Zustände wie die Seite Images, nach derselben Regel, sodass beide Seiten dieselben Zahlen zeigen. Ein Image, dessen letzte Admission abgelehnt wurde, ist

Blocked, auch wenn sein Scan innerhalb der Policy liegt, und eines, das unter Break-Glass zugelassen wurde, ist Not compliant. Die Segmente Break-glass und Unknown gibt es nicht mehr. Der *Severity trend* zählt die offenen Befunde aller vorhandenen Images, also die Zeilen der Seite Vulnerabilities, und sein Titel ist die Summe der vier Bänder darunter. *Leader* nennt die Replica, die den Controller-Lock hält.



Recently blocked admissions zeigt nur Ablehnungen, die neueste zuerst. Eine Ablehnung durch die Deny List nennt ihren Eintrag, etwa *denied by deny list entry 1: package bitnami php < 8.2.35*. Eine Ablehnung durch die Regeln einer Policy nennt die Regel, hier 3 CRITICAL vulnerabilities. Jede Zeile zeigt die Policy mit ihrem Hash, die Prüfdauer und ob das Urteil aus dem Cache kam. Die Zeile unter der Tabelle vergleicht Cache-Treffer und volle Prüfungen.

Images



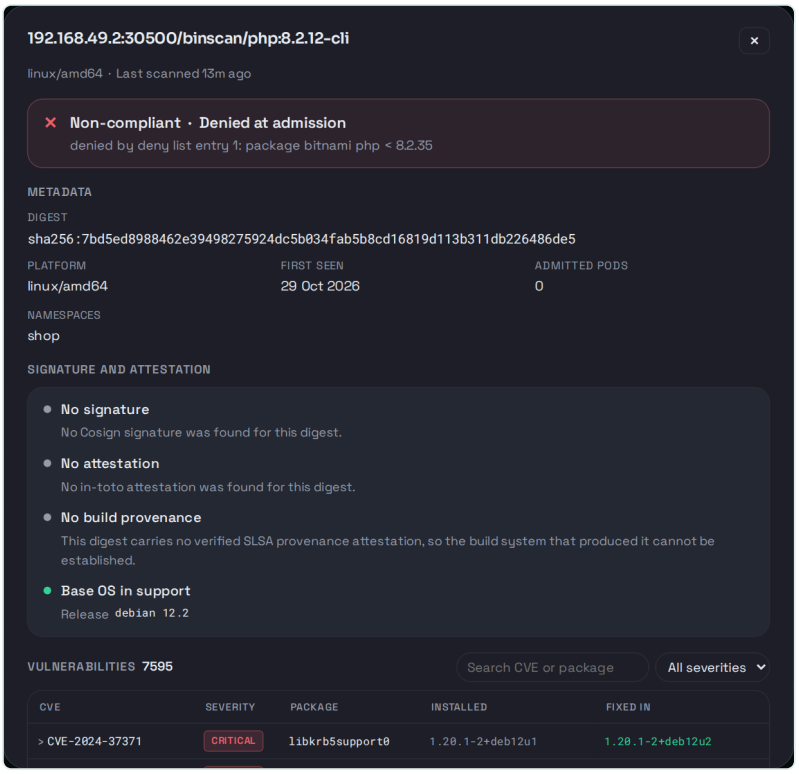
Jedes Image, das die Admission in diesem Cluster gesehen hat: 12 Images, 4 konform, 1 nicht konform, 7 blockiert, keines ungescannt. Die Kacheln ergeben zusammen die Liste. Eine von der Deny List abgelehnte Zeile trägt unter ihrem Zustand *Deny list #N*. Die Spalten zeigen Namespaces, zugelassene Pods, Befunde nach Schweregrad und den Signaturstatus. *Rescan* stellt den Scan eines Images ein, *Rescan all* den aller Images.

Ab 1.3.7 steht jedes Image in genau einem von vier Zuständen, sodass die Kacheln zusammen die Liste ergeben. Der erste zutreffende Zustand gilt:

- **Blocked**: Seine letzte Admission-Prüfung wurde abgelehnt.
- **Unscanned**: zugelassen, noch ohne Scan-Datensatz.
- **Compliant**: zugelassen, gescannt und innerhalb der Policy.
- **Not compliant**: zugelassen und gescannt, aber außerhalb der Policy, typischerweise unter einer Policy im Audit-Modus oder in einem Break-Glass-Fenster.

Blocked folgt der letzten Admission-Prüfung, nicht dem Urteil, zu dem der Scanner später kommt. Ein Image, das unter einer Audit-Policy vor seinem ersten Scan zugelassen wurde und danach kritische Befunde zeigt, ist Not compliant, und sein Detail lautet "Admitted; the next admission would be denied". Ab 1.4.0 nennt der Compliance-Chip jeder Zeile seinen Zustand mit denselben Worten, die Filter wählen danach aus, und das eigene Urteil des Scans steht im Image-Detail. Eine von der Deny List abgelehnte Zeile trägt das Abzeichen *Deny list #N*, mit der Ablehnung beim Darüberfahren. Die Spalte *Admitted pods* zählt die Pods, die mit dem Image seit seinem ersten Auftauchen zugelassen wurden. Sie sinkt nicht, wenn ein Pod stoppt, und eine abgelehnte Admission zählt nie mit.

Ab 1.3.1 werden die Listen Images und Vulnerabilities seitenweise gelesen. Filter, Suche und Sortierung laufen im Operator, und die Zahlen über jeder Tabelle umfassen alle Zeilen.

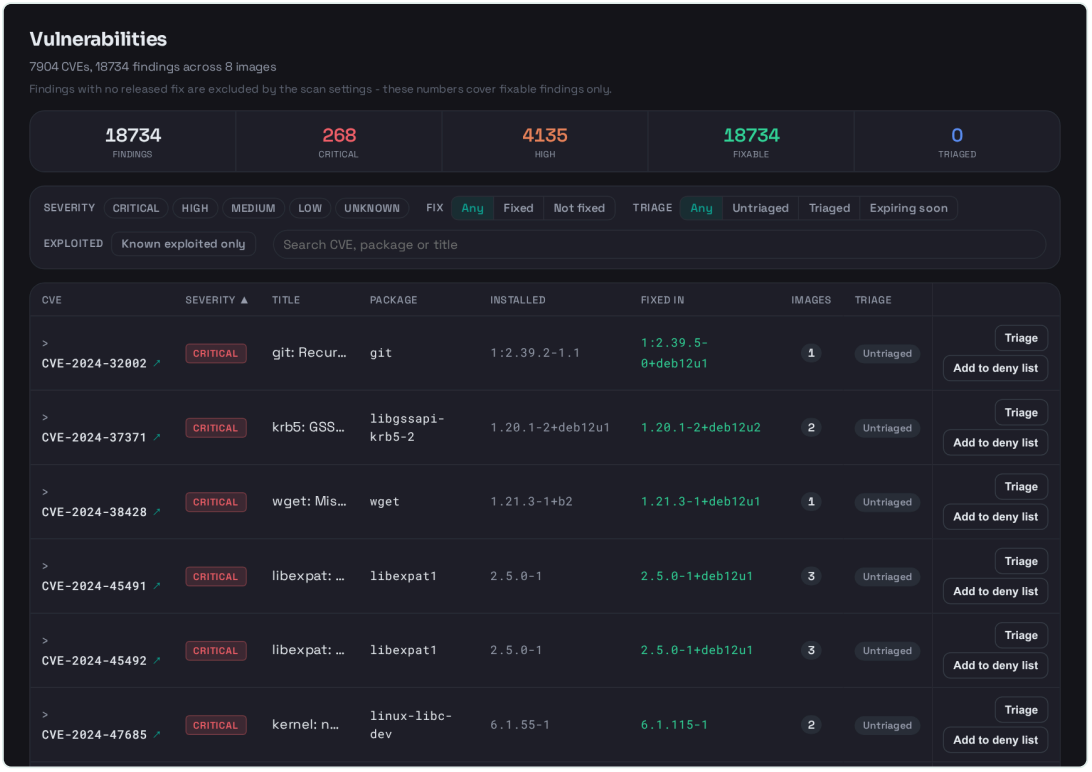


Das Fenster für `php:8.2.12-cli`. Die Admission hat es wegen Deny-List-Eintrag 1 abgelehnt, und das Fenster gibt die Ablehnung wörtlich wieder. Über den Befunden stehen vier Antworten: Gibt es eine cosign-Signatur, eine in-toto-Attestation, verifizierte SLSA Build Provenance, und wird das Basis-OS noch unterstützt. Hier wird Debian 12.2 unterstützt, und der Scan meldet 7595 Schwachstellen.

Das Detail bietet *Add to deny list* mit dem Digest des Images an und zeigt *Listed*, wenn der Digest schon auf der Liste steht. Enthält die Deny List einen Eintrag, den die Prüfung ausführbarer Dateien beantwortet, zeigt das Detail deren Antworten für diesen Digest unter *Deny list checks* (Kapitel 7).

Derselbe Scan steht für alle, die mit kubectl lesen, im Cluster. Jeder `ImageSecurityReport` trägt die Zusammenfassung in `status.summary`, die Befunde in `status.vulnerabilities` (die schwersten zuerst, höchstens 1000) und die Scanzeit in `status.lastScanned`. Eine Triage-Entscheidung wird mit `triagedBy` auf `user:<Konto-ID>` in den Bericht übernommen, nie mit einem Namen.

Vulnerabilities und Triage

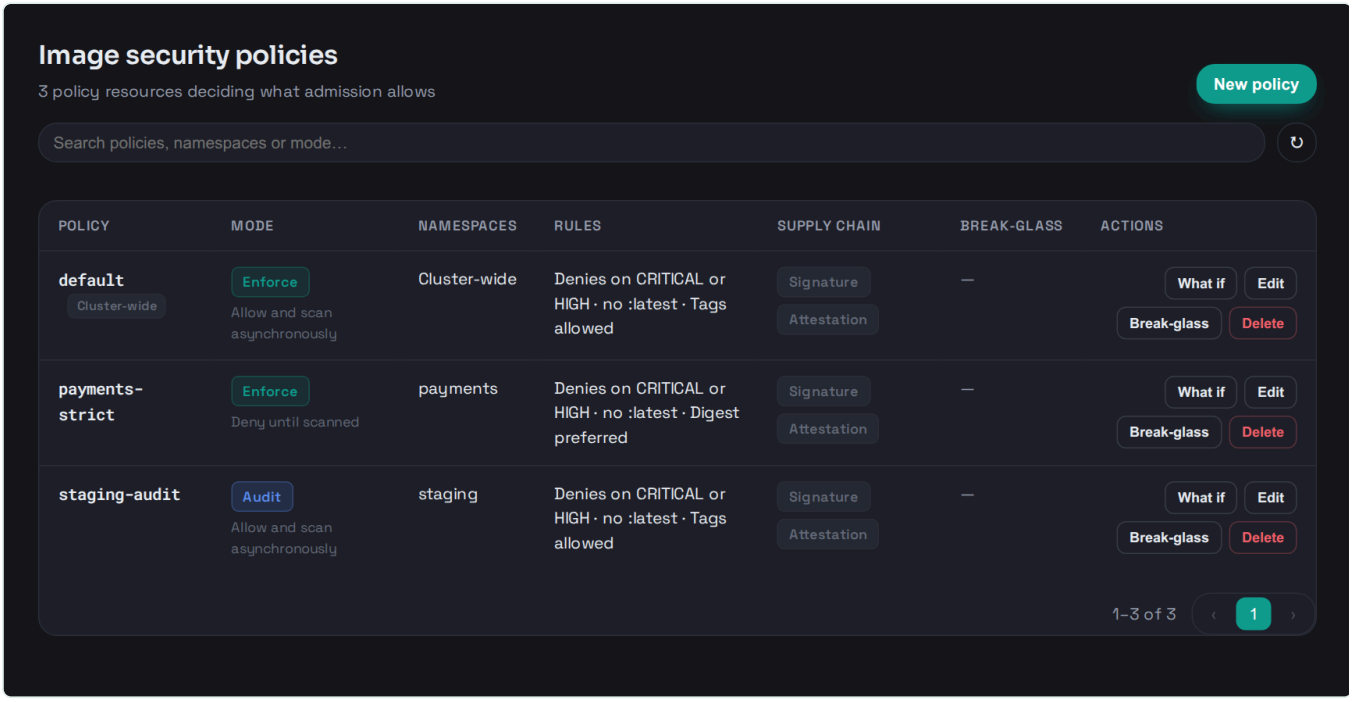


Befunde aller Images, nach CVE gruppiert, installierte und korrigierte Version nebeneinander: 7904 CVEs und 18734 Befunde in 8 Images. Die Zeile unter der Überschrift sagt, dass die Scan-Einstellungen dieser Installation Befunde ohne veröffentlichten Fix ausblenden. Jede Zeile hat *Triage* und seit 1.4.0 *Add to deny list*.

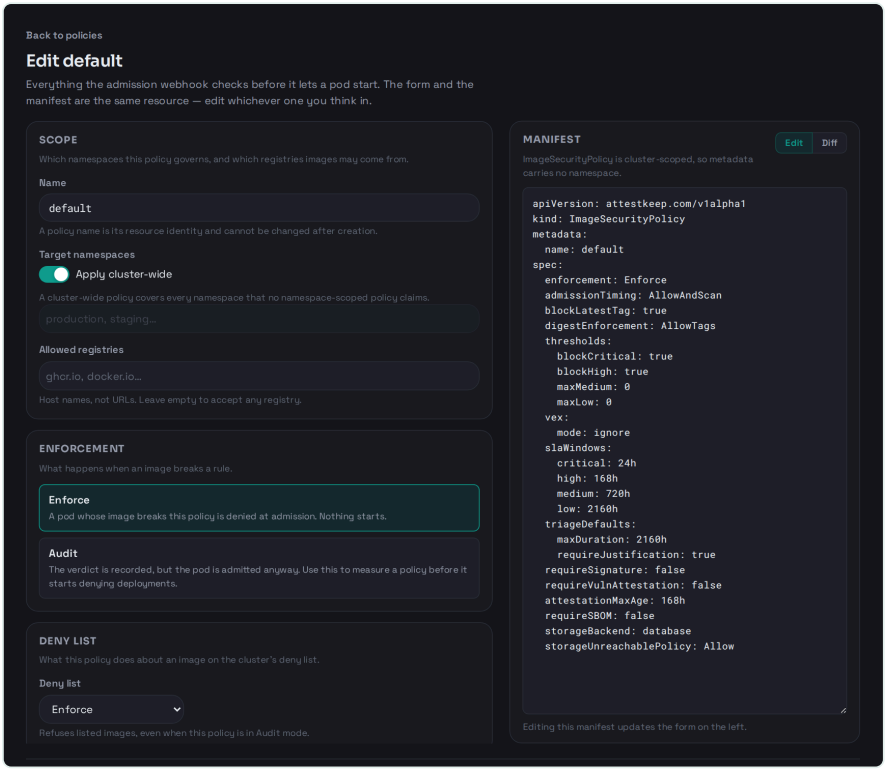
Ab 1.4.0 zählt der Titel jede CVE, die die Filter auswählen, nicht nur die 25 der Seite, und die Befunde und Images daneben werden unter denselben Filtern gezählt. Ein Digest, der über zwei Referenzen gezogen wird, seinen Tag und seinen Digest, ist ein Image: Seine Befunde zählen einmal, und es wird einmal mit allen Referenzen aufgeführt. *Add to deny list* in einer Zeile bietet die Advisory-ID an, für einen Paketeintrag das Paket und seine installierte Version.

Triage ist ein Datensatz, den ein Auditor liest, und kann daher kein stilles Stummschalten sein. Sie wählen einen Status (false positive, accepted risk oder under investigation), schreiben eine Begründung und setzen ein Ablaufdatum. Läuft es ab, gilt für den Befund wieder das Urteil der Policy. Jede Entscheidung hält fest, wer sie wann traf, und erscheint im Audit-Log und im nächsten Nachweispaket.

Admission-Policies



Drei Policies. `default` erzwingt clusterweit, `payments-strict` erzwingt in `payments` und lehnt ab, solange kein Scan vorliegt, `staging-audit` läuft in `staging` im Audit-Modus. Jede Zeile zeigt Modus, Namespaces, eine Regelübersicht und die geforderten Supply-Chain-Prüfungen, dazu *What if*, *Edit*, *Break-glass* und *Delete*.



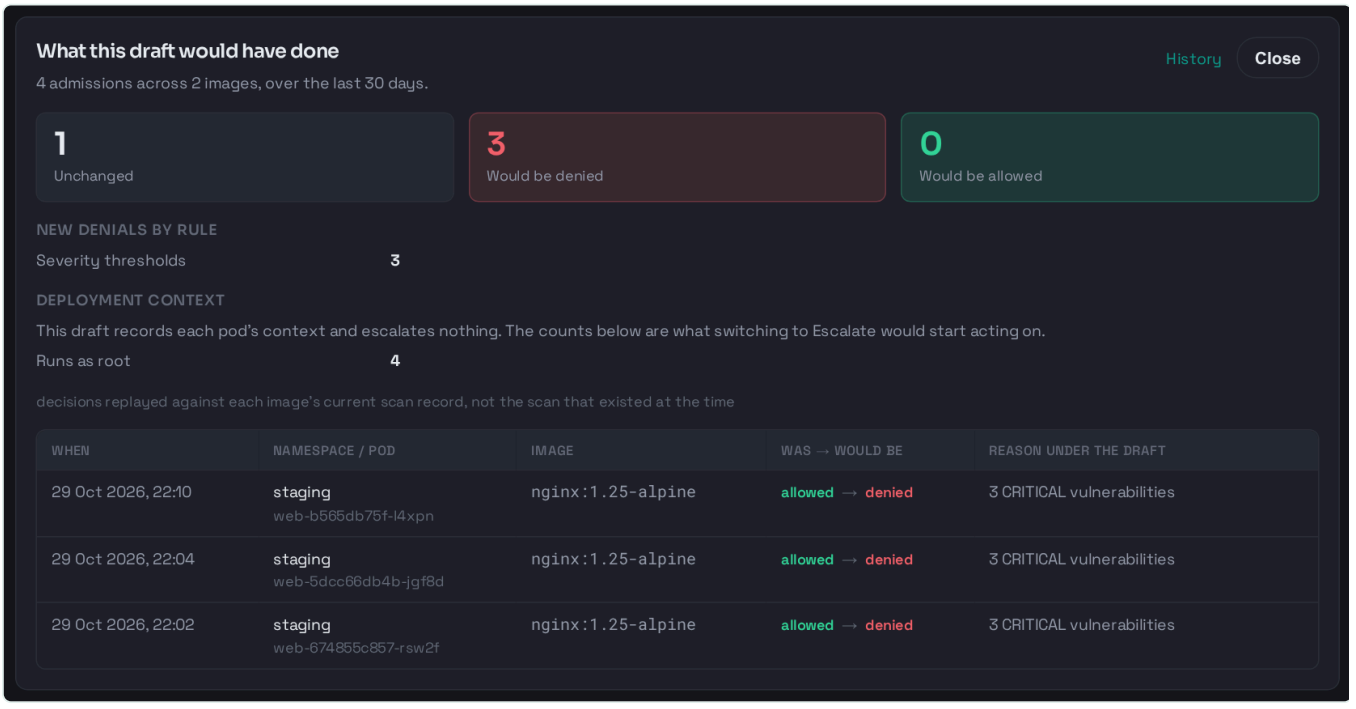
Der Editor für `default`. Formular und Manifest sind dieselbe Ressource, Sie bearbeiten, was Ihnen lieber ist. Der neue Abschnitt *Deny list* steht auf *Enforce*: Gelistete Images werden abgelehnt, auch wenn die Policy selbst im Audit-Modus läuft.

Ab 1.3.7 sehen nur Konten, die Policies schreiben dürfen, *Apply to cluster*. Alle anderen behalten das What-if-Ergebnis, Download und Copy for git. *Cancel* fragt nach, bevor ungespeicherte Änderungen verworfen werden. Ab 1.4.0 behält das Speichern die Teile der Ressource, die das Formular nicht bearbeitet, etwa den Datensatz eines geschlossenen Break-Glass-Fensters oder einen mit kubectl geschriebenen Block. Das Formular hat einen Abschnitt *Deny list* mit *Enforce*, *Audit* und *Off* (Kapitel 7). Eine neue Policy startet mit denselben SLA- und Triage-Fristen, die Operator und Dokumentation verwenden

(critical 24h, high 168h, medium 720h, low 2160h, Triage 2160h). Eine mit kubectl geschriebene Policy, die eine Frist offenlässt, öffnet mit diesen Werten, und der Diff sagt das vor dem Anwenden.

Ob eine Policy in Ihrem Cluster etwas ablehnen würde, hängt von Ihren Images, Scans und Triage-Entscheidungen ab, nicht nur von ihrem Text. *What if* beantwortet das aus den eigenen Aufzeichnungen des Clusters. Der Entwurf wird gegen die Admissions der letzten 7, 30 oder 90 Tage ausgeführt, mit derselben Funktion wie der Admission-Webhook. Nichts wird angewendet: Kein Pod wird angefasst, keine Registry kontaktiert, und der Lizenzzähler bewegt sich nicht. Entschieden wird gegen den heutigen Scan jedes Images, nicht gegen den damaligen.

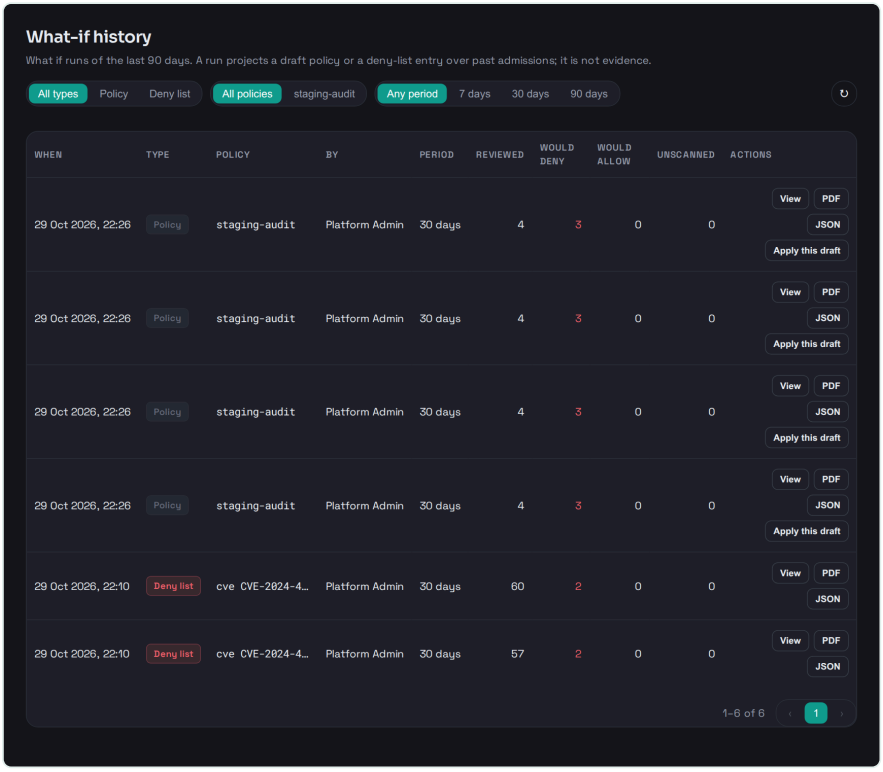
What if in einer Zeile der Policy-Liste ist der kurze Weg. Eine Audit-Policy wird so wiedergegeben, als würde sie durchsetzen, eine durchsetzende Policy so, wie sie ist, jeweils über die letzten dreißig Tage. Das Ergebnis öffnet im Editor, und nichts wird angewendet.



`staging-audit` als erzwingender Entwurf gegen die letzten 30 Tage abgespielt: 4 Admissions in 2 Images, 1 unverändert, 3 würden wegen der Severity-Schwellen abgelehnt, 0 würden zugelassen. Das Fenster nennt die Regeln hinter den neuen Ablehnungen, die Deployment-Context-Signale der abgespielten Pods (4 laufen als root) und jede geänderte Admission mit der Begründung des Entwurfs.

Scan now. Haben Images im Zeitraum keinen Scan-Datensatz, nennt das Ergebnis sie. Ein Deployer oder ein Administrator kann Scans für genau diese Digests einreihen, höchstens 500 pro Anfrage, und jede Anfrage landet als `rescan` im Audit-Log. Die Wiedergabe läuft nicht von selbst erneut. Drücken Sie *What if* noch einmal, wenn die Scans fertig sind.

What-if history



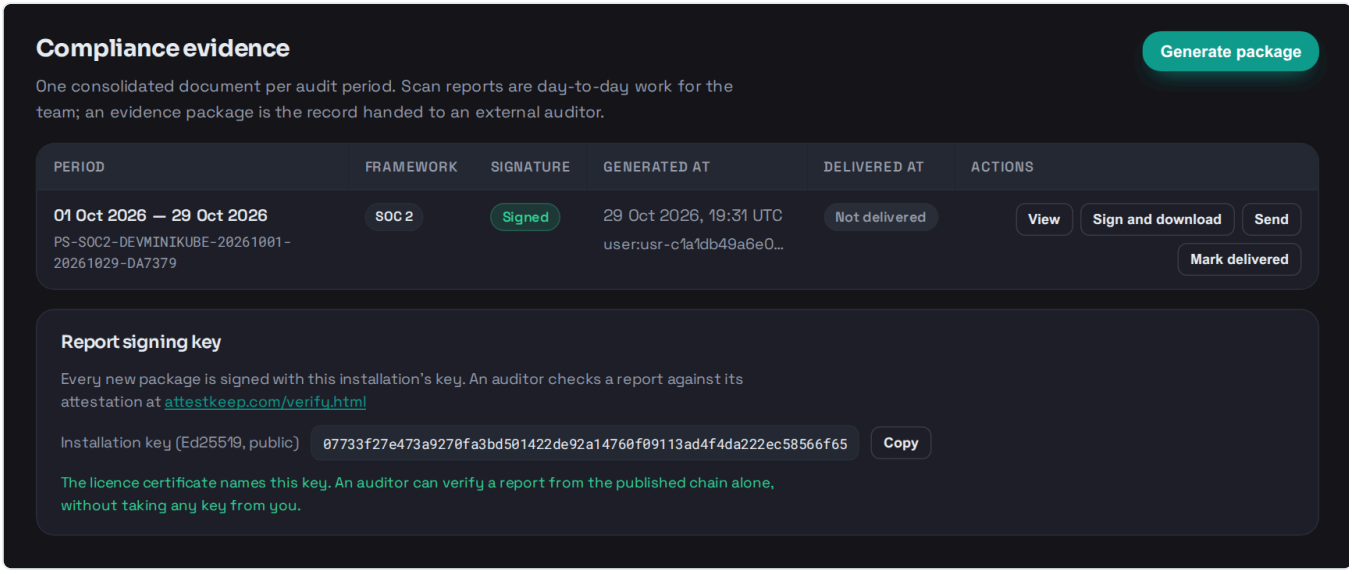
Security > What-if history hält beide Arten von Läufen. Spalte und Filter *Type* unterscheiden einen Policy-Entwurf von einem Deny-List-Eintrag: hier vier Läufe von `staging-audit` und zwei des Eintrags für CVE-2024-45491. Jeder Lauf zeigt, wer ihn gestartet hat, den Zeitraum und die Zahlen. *Apply this draft* gibt es nur bei Policy-Läufen.

Ab 1.4.0 hält die Historie zwei Arten von Läufen: einen Policy-Entwurf und einen Eintrag der Deny List, der vor dem Hinzufügen oder aus seiner Zeile ausprobiert wurde (Kapitel 7). Eine Spalte *Type* und ein Typfilter trennen sie. Eine Wiedergabe durch einen Deployer oder Administrator wird neunzig Tage aufbewahrt, höchstens fünfhundert Läufe je Art. Ein Viewer kann ebenfalls wiedergeben, die Antwort wird gezeigt, aber nicht aufbewahrt. Jeder Lauf lässt sich erneut ansehen, als PDF oder JSON herunterladen oder von einem Administrator mit *Apply this draft* im Editor öffnen. Das lädt den gespeicherten Entwurf und wendet nichts an.

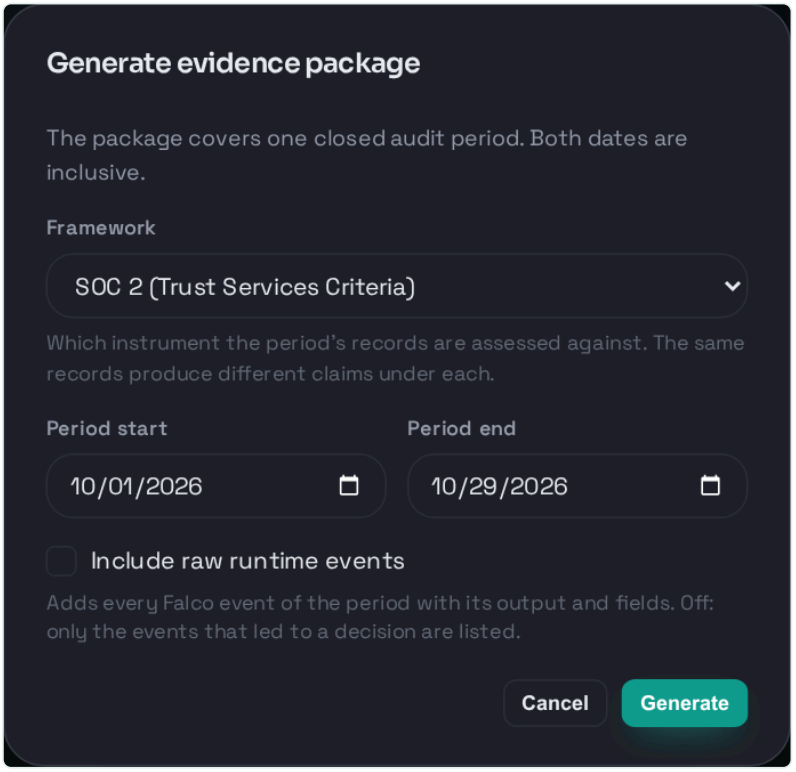
> **Ein Download ist eine Projektion, kein Nachweis.** Das PDF sagt das auf jeder Seite und ist weder signiert noch versiegelt. Ein Lauf gelangt nie in ein Nachweispaket, denn das Paket hält fest, was geschah, und eine Wiedergabe, was geschehen wäre. Jeder Download landet als `policy_whatif_download` im Audit-Log.

Die API ist `POST /api/v1/policies/whatif`. Sie nimmt optional `source`, die Policy, aus der der Entwurf bearbeitet wurde (ein unbekannter Name wird mit `400` beantwortet), und antwortet neben den Zahlen mit `kept`, `run_id`, `unscanned`, `unscanned_total` und `notes`. Ein Konto darf pro Operator-Replica höchstens zehn Wiedergaben in der Minute starten. Die nächste wird mit `429` und `Retry-After` beantwortet.

Compliance evidence

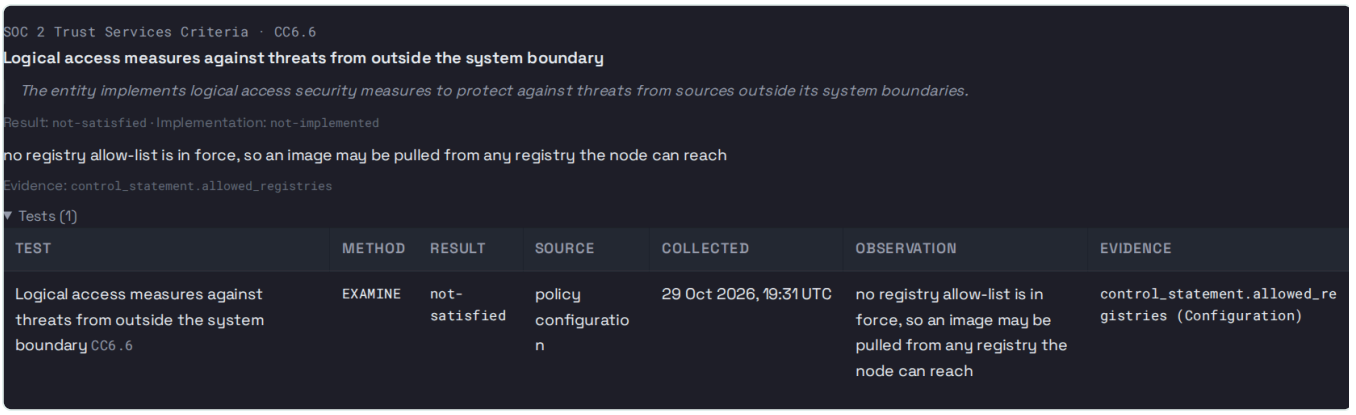


Ein Dokument pro Prüfzeitraum. Die Zeile zeigt Zeitraum, Framework, Signaturstatus, wann und von welchem Konto das Paket erzeugt wurde, geschrieben als `user:<Konto-ID>`, und ob es zugestellt wurde. Darunter der öffentliche Signaturschlüssel der Installation, gegen den ein Prüfer einen Bericht kontrolliert.



Generate evidence package. Die Liste *Framework* enthält die abgebildeten Kontrollsätze, die die Lizenz erlaubt. Der Zeitraum ist ein abgeschlossener Prüfzeitraum, beide Daten eingeschlossen. *Include raw runtime events* fügt jedes Falco-Event des Zeitraums hinzu. Ist es aus, stehen nur die Events im Paket, die zu einer Entscheidung geführt haben.

Deckt eine Lizenz weniger Frameworks ab, als die Installation schon genutzt hat, etwa nach dem Ende einer bezahlten Lizenz, wird der nächste Owner oder Administrator, der sich anmeldet, gefragt, welche Frameworks bleiben sollen. Kapitel 12 hat die Einzelheiten.



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

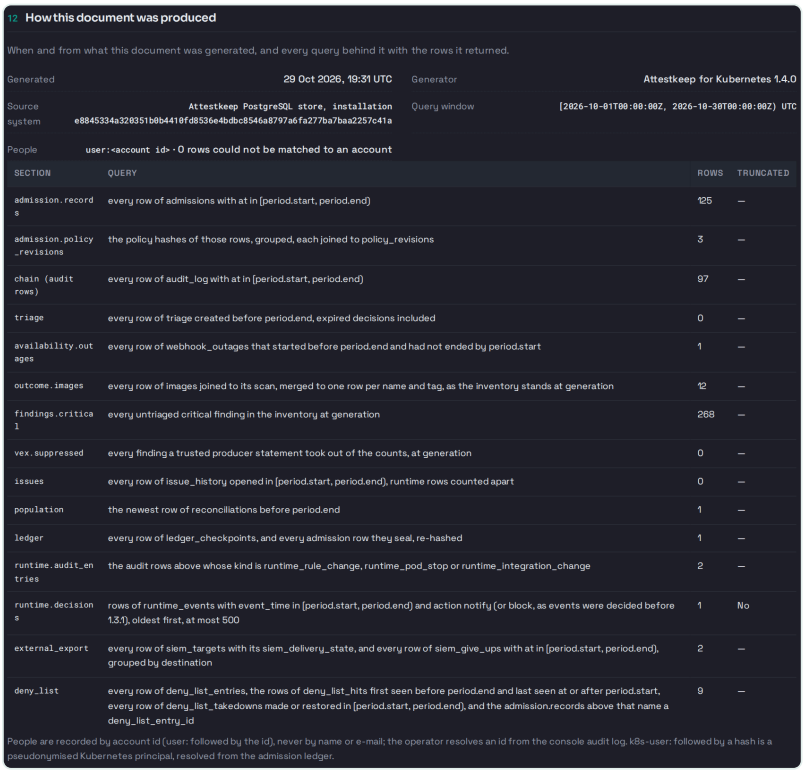
no registry allow-list is in force, so an image may be pulled from any registry the node can reach

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)

Eine Anforderung in der Berichtsansicht mit geöffnetem Test. CC6.6 ist nicht erfüllt, weil keine Registry-Allowlist gilt, und der Test sagt genau das. Die Methode ist EXAMINE, weil eine Konfiguration gelesen wird. Die Tabelle nennt außerdem Ergebnis, Quelle, Lesezeitpunkt, Beobachtung und den Pfad im Bericht.



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

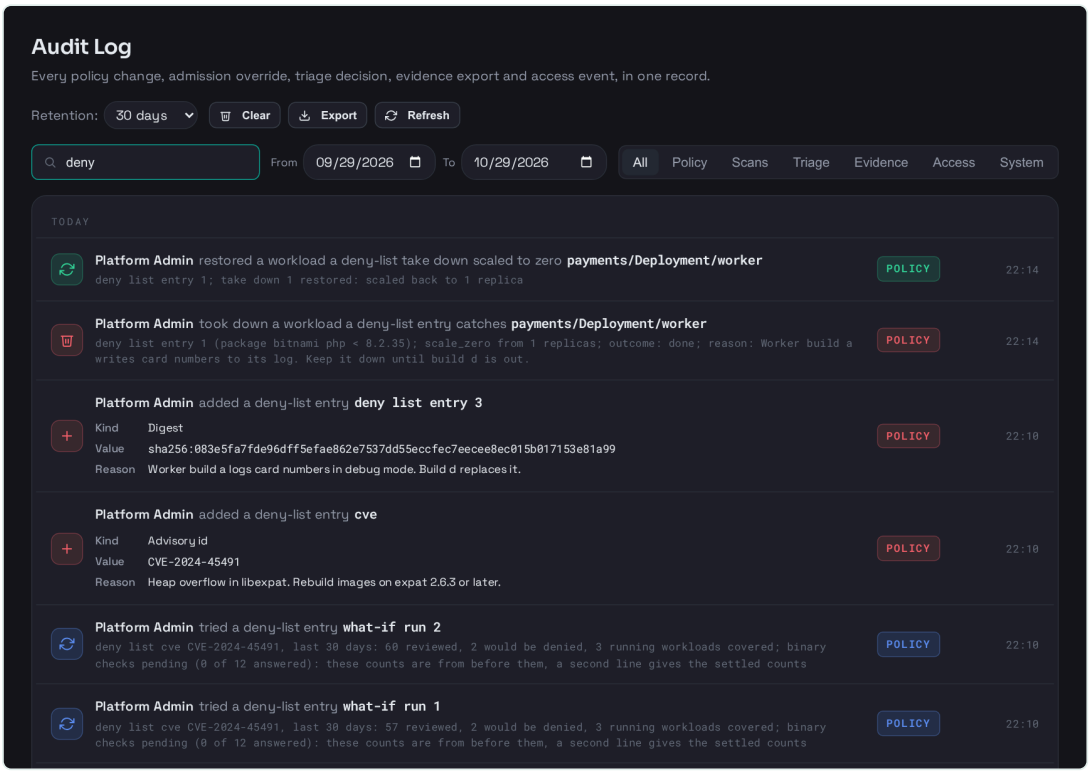
People user:<account id> · 0 rows could not be matched to an account

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 im hier gezeigten Paket, erzeugt von 1.4.0. Der Abschnitt nennt Erzeugungszeit, Generator, Quellsystem, Abfragefenster (Ende ausgeschlossen, UTC), wie Personen benannt werden, und die Abfrage jedes Abschnitts mit den gelieferten Zeilen. Die letzten beiden Zeilen, `external_export` und `deny_list`, sind neu in 1.4.0.

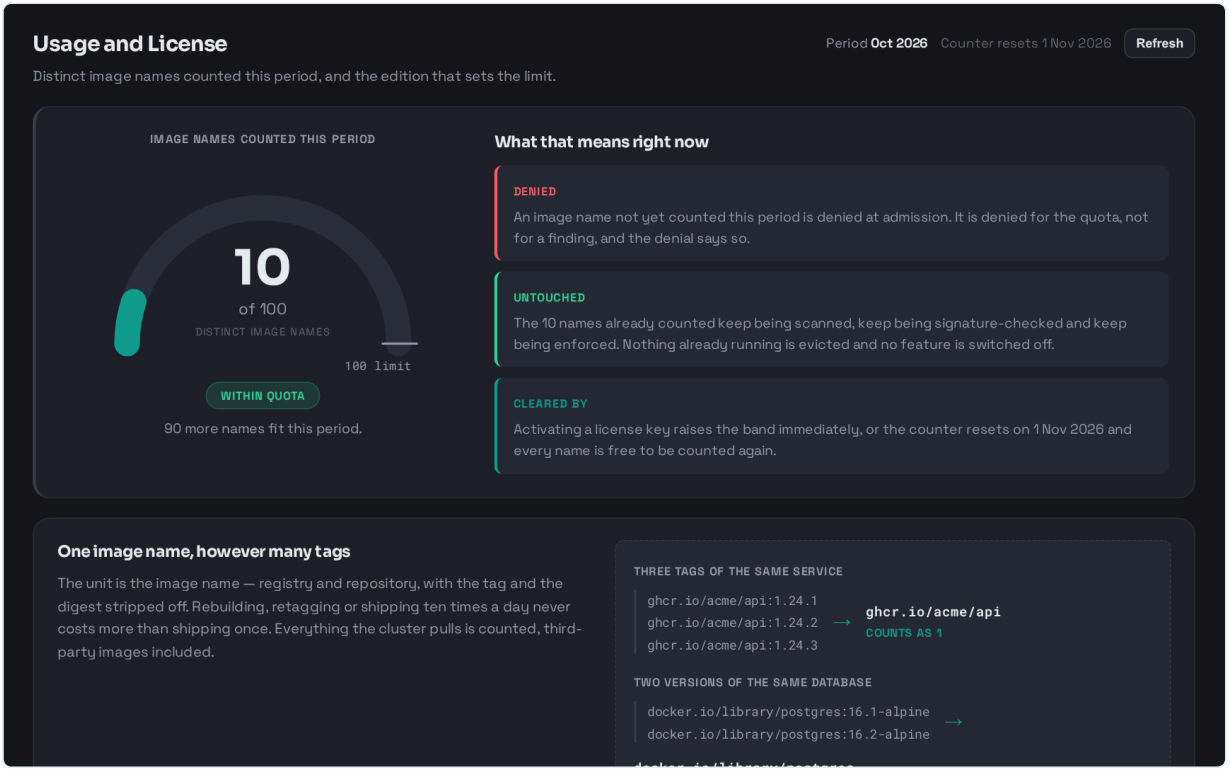
Audit Log



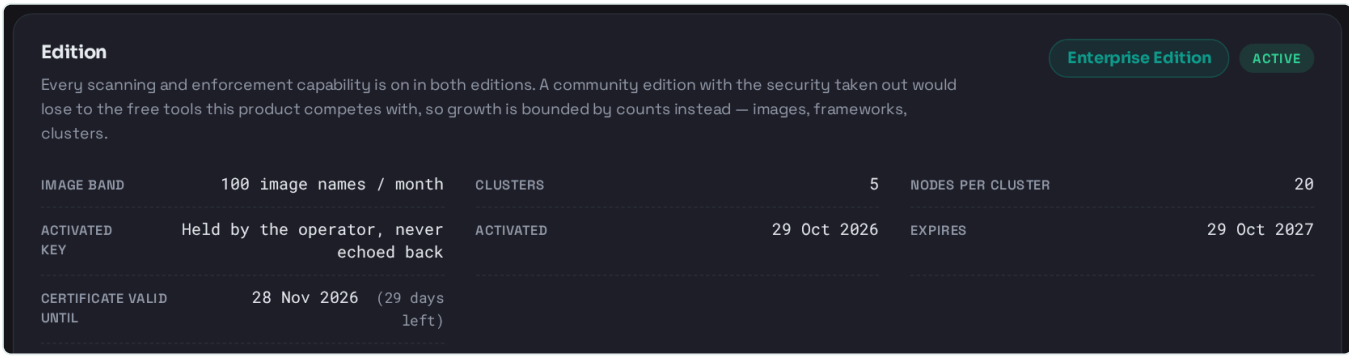
Das Audit-Log mit der Suche *deny*. Ganz oben eine Wiederherstellung und der Take Down davor, dann zwei hinzugefügte Einträge mit Art, Wert und Begründung und zwei What-if-Läufe eines Eintrags. Gesucht wird nach Akteur, Policy, Image oder CVE, gefiltert nach den Kategorien Policy, Scans, Triage, Evidence, Access und System.

Zeiten werden in Ihrer Ortszeit angezeigt, der CSV-Export behält seine ISO-Zeitstempel. Änderungen und Tests von Benachrichtigungskanälen, Änderungen an der Falco-Integration und an Runtime-Policies, von einer Runtime-Policy gestoppte Pods, die Aktionen Validieren, Speichern und Veröffentlichen des Regeleditors, What-if-Läufe und -Downloads, Scan now und Update now werden hier alle aufgezeichnet. Ab 1.4.0 auch hinzugefügte und entfernte Einträge der Deny List, Take Downs, Wiederherstellungen und Check again (`deny_list_add` , `deny_list_remove` , `deny_list_takedown` , `deny_list_restore` , `deny_list_check_again`) sowie hinzugefügte, geänderte, getestete und entfernte SIEM-Ziele (`siem_target_add` , `siem_target_update` , `siem_target_test` , `siem_target_remove`). Kanalaktionen gelangen nicht in Nachweispakete, denn sie beschreiben, was jemand an der Installation getan hat, nicht, was die Kontrolle über ein Image entschieden hat.

Usage and License



Unterschiedliche Image-Namen in diesem Zeitraum: 10 von 100, innerhalb der Quota, 90 weitere, bevor neue Namen abgelehnt werden. Die Seite erklärt, was an der Grenze geschieht. Seit 1.4.0 zählt ein Name, der wegen des Kontingents abgelehnt wurde, nicht mit, und ein volles Band zeigt *Allowance reached*.



Der Abschnitt *Edition* dieser Installation: Enterprise Edition, aktiv, mit 100 Image-Namen pro Monat, 5 Clustern und 20 Nodes pro Cluster. Die Lizenz läuft vom 29. Oktober 2026 bis zum 29. Oktober 2027. Das Zertifikat, das der Operator hält, gilt bis zum 28. November 2026. Ein Online-Zertifikat gilt höchstens dreißig Tage, und die tägliche Lizenzprüfung ersetzt es, bevor es abläuft (Kapitel 12).

Die Einheit ist der Image-Name: Registry und Repository, ohne Tag und Digest. Zehnmal am Tag neu taggen oder ausliefern kostet dasselbe wie einmal. Ab 1.3.1 werden die Zahlen des laufenden Monats auf dieser Seite und im Dashboard gegen die geltende Lizenz gelesen. Nach dem Ende einer Lizenz zeigen sie das Community-Kontingent, dieselben Zahlen, die die Admission verwendet.

Ab 1.4.0 zählt ein Image-Name, der wegen des Monatskontingents abgelehnt wurde, nicht mehr dazu. Der Zähler bleibt am Limit stehen und zeigt etwa 8 of 8 mit *Allowance reached*, wo 1.3.7 abgelehnte Namen über das Limit hinaus weiterzählte. Die Ablehnung und ihr Admission-Datensatz bleiben unverändert.

Organisation und Benutzer

Settings > Organisation enthält, was Nachweispakete über die Organisation hinter der Installation sagen. Unter *Operator declarations* geben Sie Ihre Zeitquelle und Ihre Aufbewahrungsregel an. Das Produkt misst beides nicht, und die Pakete drucken beides als Angabe des Betreibers.



Die beiden Erklärungsfelder. In dieser Installation sind sie leer. Der graue Text ist das Beispiel, das das Formular zeigt, bis Sie etwas eintragen.

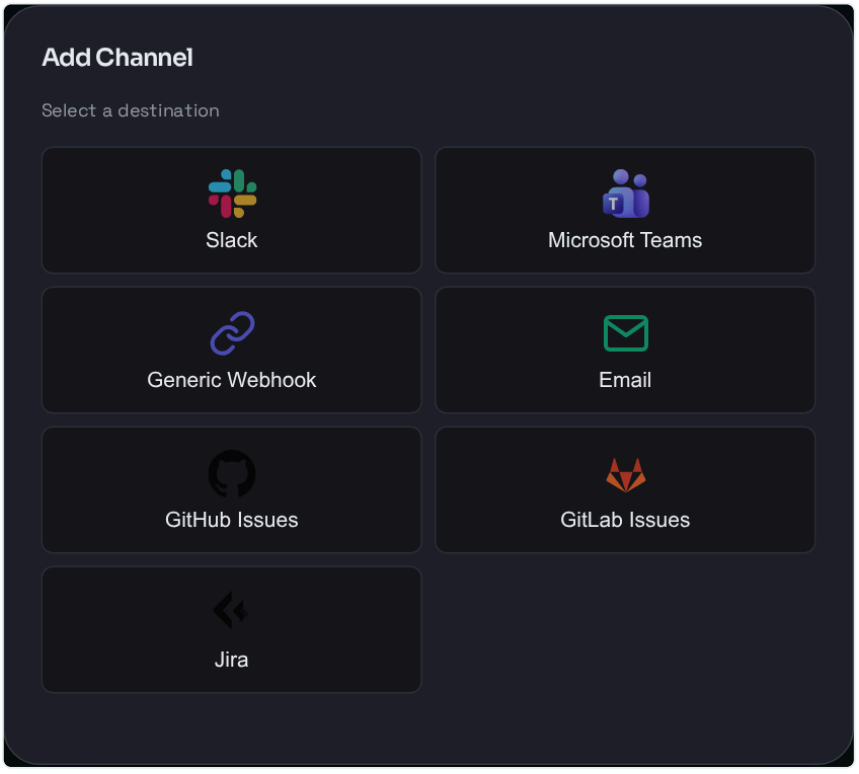
Das Löschen eines Benutzers deaktiviert das Konto, statt es zu entfernen. Ein deaktiviertes Konto kann sich nicht anmelden, und seine Sitzungen enden. Seine ID und seine Audit-Zeilen bleiben, sodass jedes `user:<id>` in einem ausgestellten Paket weiter auflösbar ist. Für einen Löschantrag kann ein Admin das deaktivierte Konto danach mit *Anonymise* anonymisieren. Das leert Name, E-Mail und Avatar und schreibt jede Kopie des Namens im Audit-Log, in Triage-Entscheidungen und in Runtime-Policies samt ihren Revisionen auf `user:<id>` um. Bereits signierte Pakete behalten, womit sie signiert wurden, und schon an einen Issue-Tracker gesendete Kommentare werden nicht geändert.



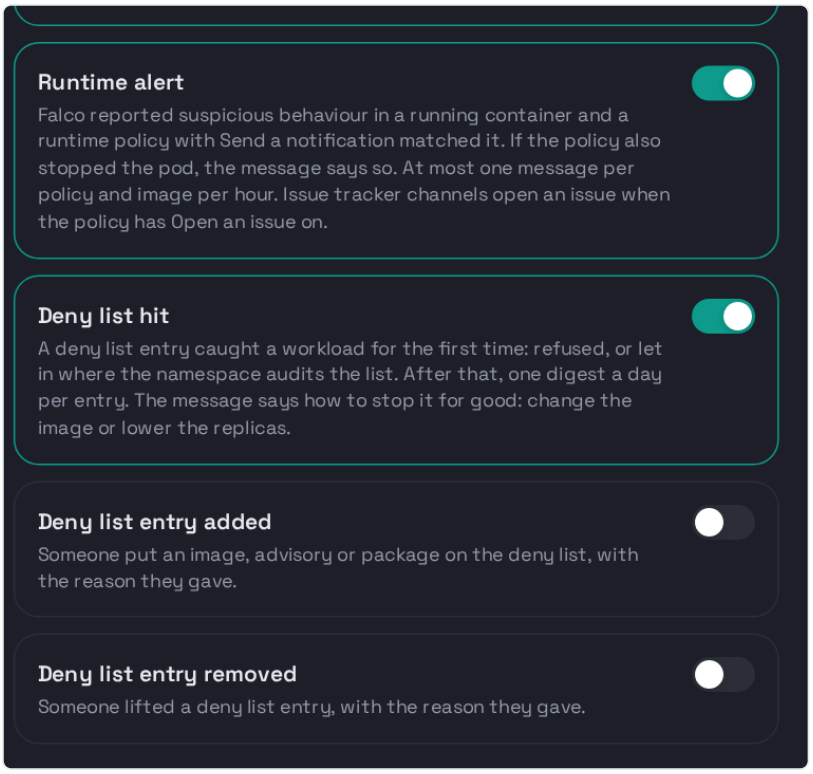
Ein anonymisiertes Konto bleibt deaktiviert in der Liste, mit seiner Konto-ID an der Stelle der E-Mail-Adresse. Das deaktivierte Konto darunter bietet noch *Anonymise* an.

Benachrichtigungskanäle

Kanäle werden unter Settings > Webhooks auf der Seite *Notification Channels* eingerichtet. Die eigenen E-Mail-Einstellungen jeder Person stehen unter Settings > My Notifications.



Add Channel. Ein Issue-Tracker ist ein Kanal wie die anderen, mit denselben Abonnements. *Test* schreibt bei einem Tracker-Kanal nichts. Er liest Repository oder Projekt einmal und meldet, ob die Zugangsdaten funktionieren.



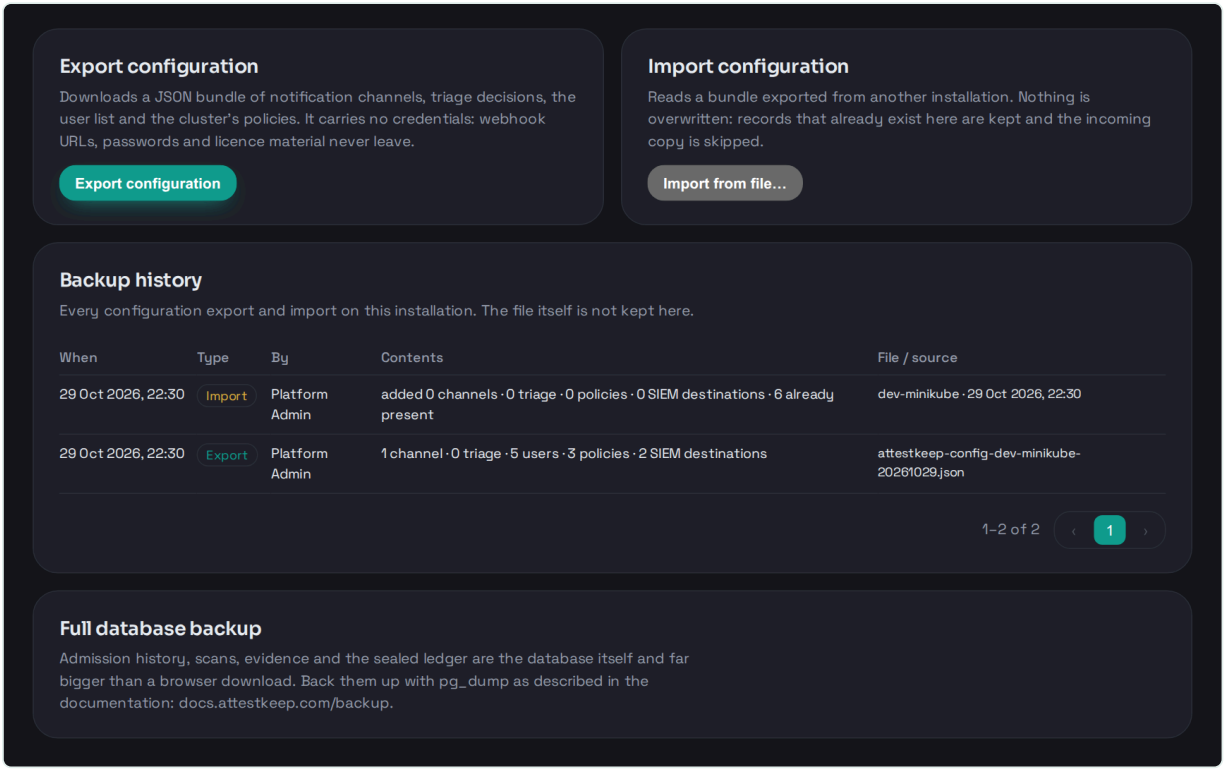
Ein Ausschnitt der Liste *Events* eines Kanals. *Deny list hit* ist an, wie bei jedem Kanal, der ab 1.4.0 angelegt wird: eine Nachricht, wenn ein Eintrag zum ersten Mal einen Workload trifft, danach eine Zusammenfassung pro Tag und Eintrag. *Deny list entry added* und *Deny list entry removed* sind standardmäßig aus. *Runtime alert* beschreibt Kapitel 8.

Ab 1.4.0 kann ein Kanal über die Deny List informiert werden. *Deny list hit* wird gesendet, wenn ein Eintrag zum ersten Mal einen Workload erfasst, mit dem Eintrag, seiner Begründung und dem Ausweg, danach einmal am Tag pro Eintrag als Zusammenfassung. Ein neuer Kanal hat es angehakt. Ein vor 1.4.0 angelegter Kanal behält seine Abonnements. Eine Ablehnung durch die Deny List wird nicht zusätzlich als *Admission blocked* gesendet, das gilt für Ablehnungen durch die eigenen Regeln einer Policy. *Deny list entry added* und *Deny list entry removed* sind standardmäßig aus.

Jeder Versand per Mail, Webhook, Slack oder Teams hat 30 Sekunden vom Verbindungsaufbau bis zum letzten Byte. Ein Versand, der die Zeit überschreitet, steht im Benachrichtigungsprotokoll als nicht zugestellt mit "no answer within 30s". Höchstens 64 Sendungen laufen gleichzeitig, und höchstens 64 Benachrichtigungen warten. Darüber hinaus wird eine Benachrichtigung verworfen, als nicht zugestellt vermerkt und in `attestkeep_notifications_dropped_total` gezählt.

Ein Token für einen Issue-Tracker ist die einzige Zugangsinformation in Attestkeep, die in ein System außerhalb Ihres Clusters schreiben kann. Die API gibt es nie zurück. Vor dem Speichern wird es mit AES-256-GCM verschlüsselt, unter einem Schlüssel, den das Chart im Secret `<release>-data-key` hält und nie in der Datenbank. Setzen Sie `api.dataKey.existingSecret`, um den Schlüssel aus Ihrem eigenen Secret-Manager zu liefern. Ohne den Schlüssel lässt sich kein Tracker-Kanal speichern. Sonst hängt nichts daran.

Backup history



Settings > Backup. *Export configuration* lädt ein JSON-Bundle mit Benachrichtigungskanälen, Triage-Entscheidungen, der Benutzerliste, den Policies und seit 1.4.0 den SIEM-Zielen herunter, ohne Zugangsdaten. *Backup history* listet jeden Export und Import mit Zeitpunkt, Person und Inhalt des Bundles. Der Operator behält die Datei nicht. Die Datenbanksicherung ist Ihre Aufgabe (Kapitel 11).

6. Admission-Policies

Eine clusterweite Ressource entscheidet, was laufen darf: `ImageSecurityPolicy` in der API-Gruppe `attestkeep.com`. Die Konsole bearbeitet sie direkt, `kubectl apply` bearbeitet sie aus einem Repository. Beide arbeiten am selben Objekt. Eine Änderung gilt ab der nächsten Admission-Entscheidung, ohne Neustart.

Wie ein Namespace zu seiner Policy kommt. Es gilt die Policy, deren `spec.targetNamespaces` den Namespace aufführt. Sonst gilt die Policy ohne `targetNamespaces`, der globale Standard. Üblich sind eine globale Policy und eine strengere für die Namespaces, die sie brauchen.

Der mitgelieferte Standard

```
apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: default
spec:
  enforcement: Enforce          # Audit zeichnet nur auf, was abgelehnt worden wäre
  admissionTiming: AllowAndScan # DenyUntilScanned lehnt alles ohne Scan-Datensatz ab
  blockLatestTag: true          # ein veränderlicher Tag ändert, was läuft, ohne Deploy
  digestEnforcement: AllowTags  # AllowTags | PreferDigest | Required
  denyList: "enforce"           # enforce | audit | "off" (Kapitel 7)
  thresholds:
    blockCritical: true
    blockHigh: true
    maxMedium: 0                # 0 heißt hier unbegrenzt
    maxLow: 0
  slaWindows:
    critical: 24h
    high: 168h
    medium: 720h
    low: 2160h
  requireSignature: false       # einschalten, sobald Ihre Pipeline Signaturen erzeugt
  requireSBOM: false
  requireVulnAttestation: false
  attestationMaxAge: 168h
  triageDefaults:
    maxDuration: 2160h
    requireJustification: true
```

Eine strengere Policy für Produktions-Namespaces

```
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: []
```

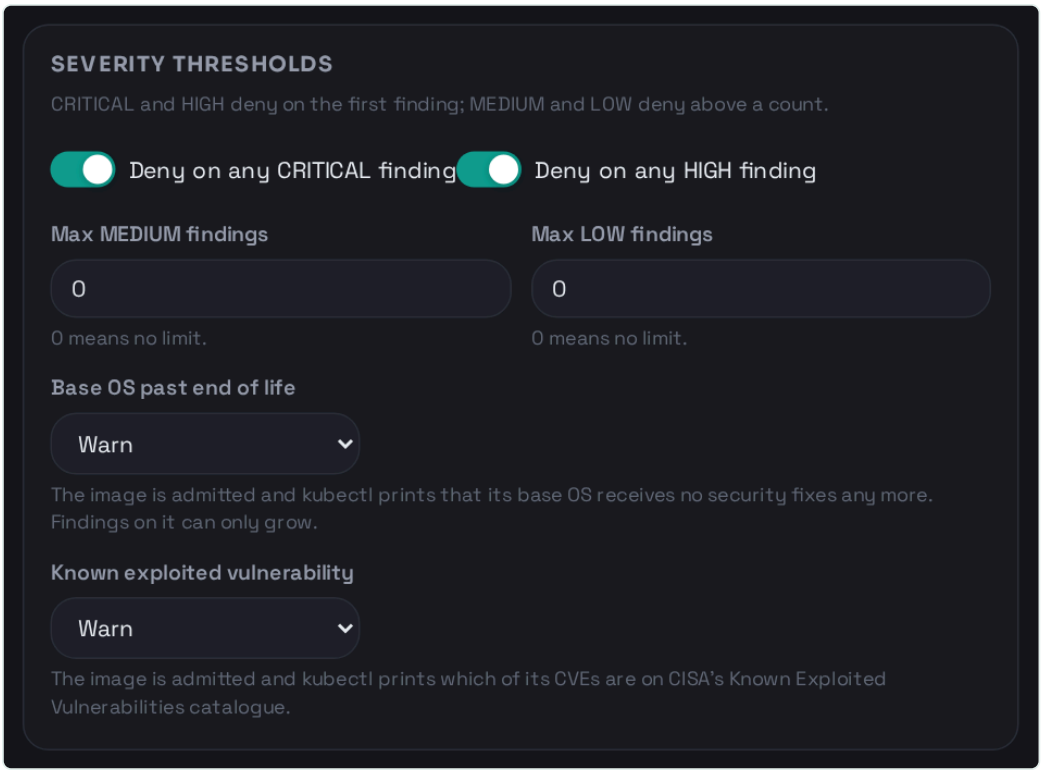
Basis-Betriebssystem ohne Support und ausgenutzte CVEs

Zwei Sperren schauen auf etwas anderes als die Zahl der Befunde je Schweregrad. Beide akzeptieren `ignore`, `warn` oder `block`, und beide stehen standardmäßig auf `warn`. Eine Policy, die vor ihnen geschrieben wurde, lässt weiter zu, was sie bisher zuließ, und sagt jetzt zusätzlich, warum sie es nicht sollte.

```
spec:
  # Das Basis-Betriebssystem liegt außerhalb des Supportzeitraums seines
  # Herstellers, es kommen keine Sicherheitskorrekturen mehr. warn (Standard)
  # lässt zu und sagt es; block lehnt ab, egal wie die Zahlen stehen;
  # ignore sagt nichts.
  endOfLifeOS: warn

  # Das Image enthält eine CVE aus dem CISA-Katalog Known Exploited
  # Vulnerabilities. warn (Standard) lässt zu und nennt die CVEs;
  # block lehnt ab; ignore sagt nichts.
  knownExploited: warn
```

Der KEV-Katalog wird als lokale Kopie gehalten und alle sechs Stunden erneuert. Ein Cluster ohne Internetzugang richtet `kev.source` auf einen Mirror oder lässt den Wert leer, um die vorhandene Kopie zu behalten. Per Triage bewertete CVEs sind von dieser Sperre ausgenommen, wie von den Schwellen. Eine Ablehnung nennt die CVEs, damit die Person, die deployt, weiß, was zu beheben ist. Das Dashboard zeigt die verwendete Katalogversion, und Nachweispakete halten sie neben der Version der Schwachstellendatenbank fest.



Der Abschnitt *Severity thresholds* im Editor. *Base OS past end of life* und *Known exploited vulnerability* stehen beide auf Warn, und der Text darunter sagt jeweils, was die Einstellung bewirkt.

Signaturschlüssel

Ab 1.4.0 kann eine Policy per Schlüssel sagen, wessen Signatur sie akzeptiert. `signatureKeys` listet PEM Public Keys auf, etwa Ihren `cosign.pub`. Mit `requireSignature: true` zählt nur eine Signatur, die einer davon bestätigt.

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

Der Scan hält den Signierenden als Fingerabdruck des Schlüssels fest, `key sha256:…`, und das Nachweispaket nennt ihn. Mit `scan.cosignOffline` braucht die Prüfung kein Transparenzlog. SBOM- und Schwachstellen-Attestations bleiben keyless und haben kein Schlüsselfeld. Der API-Server lehnt einen Eintrag ab, der nicht wie ein PEM Public Key aussieht. Einer, der trotzdem nicht lesbar ist, wird ignoriert und an drei Stellen genannt: in der Status-Condition `SignatureKeysValid` der Policy, in einem Warning-Event und auf der Policy-Seite.

Build-Provenance

Diese Sperre prüft, welches Build-System das Image aus welcher Quelle erzeugt hat. Sie ist die einzige Prüfung, die an eine Identität gebunden ist. `requireSBOM` und `requireVulnAttestation` prüfen, dass jemand eine Attestation signiert hat. Diese Sperre prüft, wer.

```
spec:
  provenance:
    # 0 (Standard, auch wenn der Block fehlt) schaltet die Sperre aus.
    # Über 0 muss das Image eine geprüfte SLSA-Provenance tragen, die einen
    # der unten aufgeführten Builder nennt, mit mindestens diesem Level.
    minLevel: 2
    trustedBuilders:
      # Ein "*" am Ende vergleicht per Präfix, andere IDs exakt.
      # Eine leere Liste mit minLevel über 0 lässt nichts zu.
      - id: https://github.com/acme/build/.github/workflows/release.yml@*
        # Ihre Aussage über diesen Builder. SLSA-Provenance trägt kein
        # Level, denn ein Level beschreibt, wie der Builder betrieben wird.
        level: 3
        # Aus dem Signaturzertifikat gelesen. Ohne Wert muss er der Builder-ID
        # entsprechen. Ohne issuer wird jeder Aussteller akzeptiert.
        signer: https://github.com/acme/build/.github/workflows/release.yml@*
        issuer: https://token.actions.githubusercontent.com
```

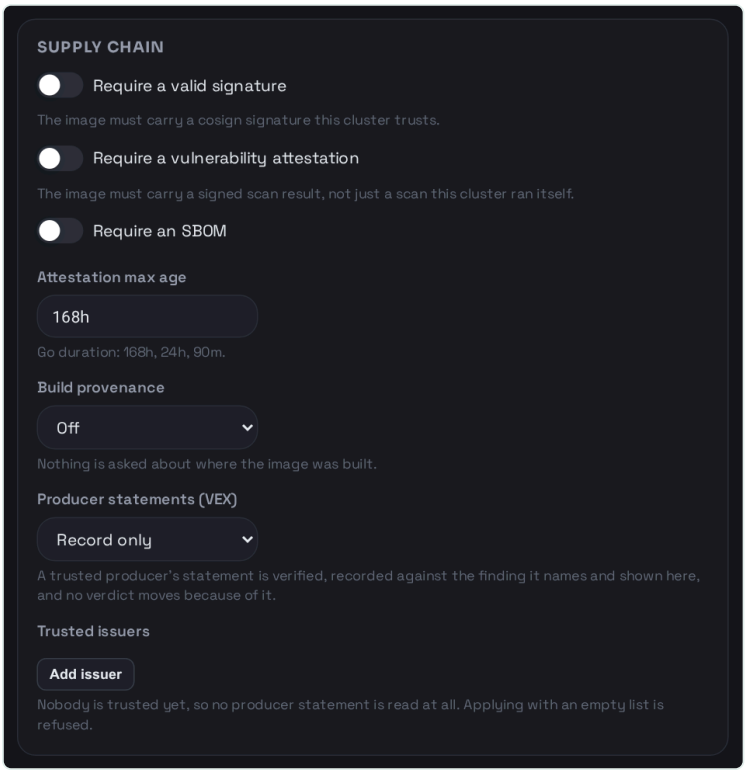
Abgelehnt wird ein Image, dessen geprüfte SLSA-Provenance (v1 oder v0.2) keinen vertrauenswürdigen Builder nennt, dessen Identität oder Aussteller im Signaturzertifikat nicht der des Eintrags entspricht oder dessen Builder-Level unter `minLevel` liegt. Die Begründung nennt Builder, Signierenden oder Level. Beide Identitäten werden aus dem Zertifikat gelesen, das mit der Attestation gespeichert ist. Lässt es sich nicht abrufen oder lesen, wird das Image abgelehnt.

Herstelleraussagen (VEX)

Ein Scanner meldet, was in einem Image installiert ist. Ob der verwundbare Code erreichbar ist oder je läuft, kann er nicht sagen. Wer das Image gebaut hat, kann es, und OpenVEX ist das Format dafür. Attestkeep liest eine VEX-Aussage nur, wenn sie als signierte in-toto-Attestation am Image hängt und die Policy den Signierenden nennt. Unsignierte Aussagen, Aussagen von Signierenden, die die Policy nicht nennt, und Aussagen über ein anderes Produkt werden aufgezeichnet und ignoriert.

```
spec:
  vex:
    # ignore (Standard): vertrauenswürdige Aussagen werden geprüft, beim
    # Befund vermerkt und in der Konsole gezeigt; kein Urteil ändert sich.
    # apply: eine vertrauenswürdige Aussage not_affected oder fixed nimmt
    # den Befund aus den Zahlen, nach denen die Admission entscheidet.
    mode: apply
    trustedIssuers:
      # Ein keyless Signierender, über die Zertifikatsidentität. Ein "*" am
      # Ende vergleicht per Präfix. issuer ist optional und wird exakt verglichen.
      - identity: https://github.com/acme/images/*
        issuer: https://token.actions.githubusercontent.com
      # Oder ein Signierender mit Schlüssel, über den Public Key. Ein Eintrag
      # hat eine Identität oder einen Schlüssel, nicht beides.
      - key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...
          -----END PUBLIC KEY-----
```

`affected` und `under_investigation` unterdrücken nie einen Befund. Ein unterdrückter Befund fällt aus den Zahlen, die Schwellen und die Regel für ausgenutzte CVEs verwenden, und die Begründung beim Zulassen hält fest, wie viele Befunde von wem unterdrückt wurden. `mode: apply` mit leerer Liste `trustedIssuers` wird abgelehnt, weil es wie eine Kontrolle aussähe und nichts täte. Ein Moduswechsel gilt ab dem nächsten Pod, ein neuer Aussteller ab dem nächsten Scan des Digests.



Der Abschnitt *Supply chain*. Signatur, Vulnerability-Attestation und SBOM sind nicht gefordert, *Build provenance* steht auf Off und *Producer statements (VEX)* auf *Record only*, der Voreinstellung. Noch ist kein Aussteller vertrauenswürdig, also wird keine Erklärung gelesen.

Wo der Pod laufen wird

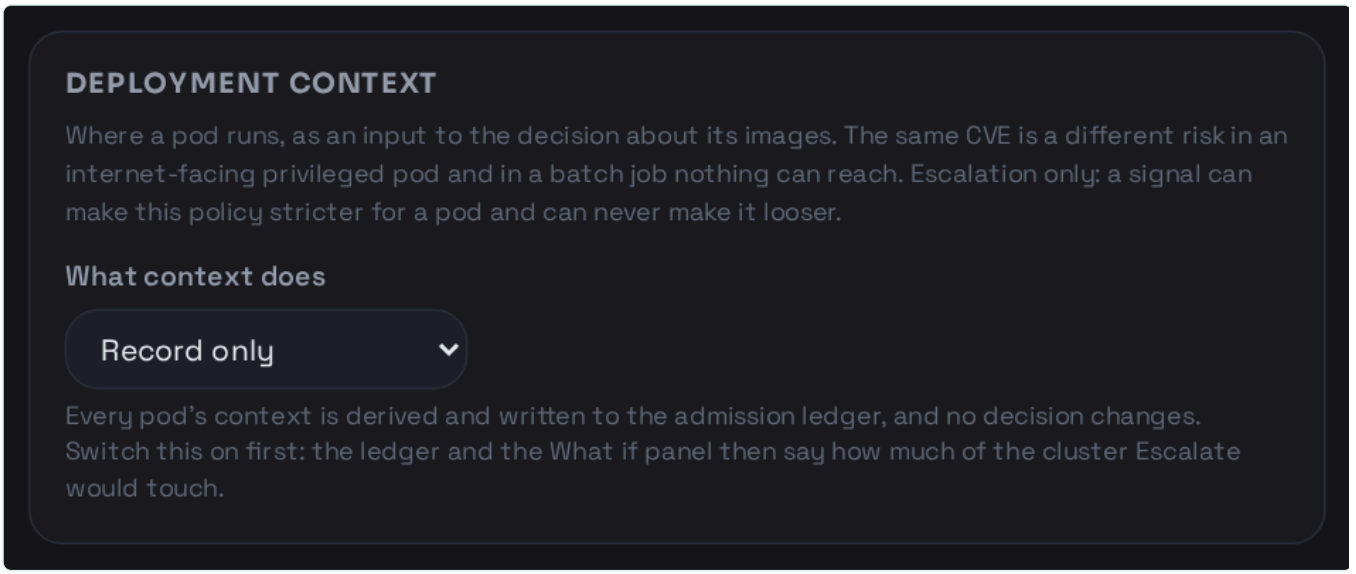
Derselbe HIGH-Befund wiegt in einem privilegierten, aus dem Internet erreichbaren Pod schwerer als in einem Batch-Job, den niemand erreicht. Mit `deploymentContext` kann eine Policy solche Pods strenger behandeln. Sie macht eine Entscheidung immer nur strenger. Kein Signal und keine Kombination von Signalen lässt einen Pod zu, den die Policy sonst ablehnen würde.

```
spec:
  deploymentContext:
    mode: Escalate      # Ignore (Standard) | Escalate
    signals:            # leer heißt alle sechs
      - exposed
      - privileged
```

Für jeden Pod werden sechs Signale abgeleitet und in jeder Ledger-Zeile festgehalten, in beiden Modi: `privileged`, `root`, `hostNamespace`, `hostFilesystem`, `capabilities` und `exposed`. Nur `exposed` schaut über den Pod hinaus: hostNetwork, ein hostPort, ein Service vom Typ NodePort oder LoadBalancer, der den Pod auswählt, oder ein ClusterIP-Service hinter einem Ingress. Ein ClusterIP-Service ohne Ingress gilt nicht als erreichbar. Unter `Escalate` wird ein Pod mit einem ausgewählten Signal so beurteilt:

Die Policy sagt	Ein Pod mit ausgewähltem Signal wird beurteilt als
`thresholds.blockCritical: true`	sperrt auch HIGH
`thresholds.blockHigh: true`	sperrt auch MEDIUM
`knownExploited: warn`	`block`
`endOfLifeOS: warn`	`block`

`maxMedium` und `maxLow` bleiben unverändert, und `ignore` bleibt `ignore`. Die Ablehnung nennt das Signal und bei Erreichbarkeit den Service, den Ingress oder den hostPort, der sie ausgelöst hat.



Der Abschnitt *Deployment context* auf *Record only*, also `Ignore`. Der Kontext jedes Pods wird ins Admission-Ledger geschrieben, und keine Entscheidung ändert sich. Ledger und What-if-Fenster zeigen dann, wie viele Pods `Escalate` betreffen würde.

Feldreferenz

Feld	Werte	Was es entscheidet
enforcement	Enforce | Audit	Ablehnen oder nur aufzeichnen, was abgelehnt worden wäre.
admissionTiming	AllowAndScan | DenyUntilScanned	Was mit einem Image ohne Scan-Datensatz geschieht.
blockLatestTag	bool	Den Tag `:latest` ablehnen.
digestEnforcement	AllowTags | PreferDigest | Required	Ob Images per Digest gepinnt sein müssen.
allowedRegistries	Liste von Hosts	Leer erlaubt jede Registry. Nicht leer lehnt alle anderen ab.
denyList	enforce | audit | "off"	Was ein Treffer der Deny List in den Namespaces dieser Policy bewirkt. Standard `enforce`, das auch unter `enforcement: Audit` ablehnt (Kapitel 7).
targetNamespaces	Liste	Leer macht dies zur globalen Policy. Nicht leer beschränkt sie auf diese Namespaces.
thresholds	blockCritical, blockHigh, maxMedium, maxLow	Bei jedem kritischen oder hohen Befund ablehnen. Obergrenzen für mittel und niedrig setzen (0 heißt unbegrenzt).
slaWindows	Dauern	Wie lange ein Befund je Schweregrad offen bleiben darf, bevor Berichte ihn als überfällig führen.
requireSignature / requireSBOM / requireVulnAttestation	bool	Images ohne geprüfte cosign-Signatur, ohne SBOM-Attestation oder ohne Attestation eines Schwachstellen-Scans ablehnen.
signatureKeys	Liste von PEM Public Keys	Mit `requireSignature` wird nur eine Signatur akzeptiert, die einer dieser Schlüssel bestätigt.
attestationMaxAge	Dauer	Eine ältere Attestation zählt nicht mehr.
workloadHardening	mode, minSeverity, exempt	Ob privilegierte Container, hostPath-Mounts und ähnliche Einstellungen ablehnen, warnen oder durchgehen.
endOfLifeOS	ignore | warn | block	Images, deren Basis-Betriebssystem den Supportzeitraum verlassen hat. Standard `warn`.
knownExploited	ignore | warn | block	Images mit einer CVE aus dem KEV-Katalog der CISA. Standard `warn`.
provenance	minLevel 0 bis 3, trustedBuilders (id, level, signer, issuer)	Images ohne geprüfte Provenance eines vertrauenswürdigen Builders auf dem verlangten Level ablehnen. Standard `minLevel: 0`, aus.
vex	mode ignore | apply, trustedIssuers	Ob die Aussage not_affected oder fixed eines vertrauenswürdigen Herstellers einen Befund aus den Zahlen nimmt. Standard `ignore`.
deploymentContext	mode Ignore | Escalate, signals	Pods mit ausgewähltem Signal strenger beurteilen. Standard `Ignore`.
runtimeReconciliation	audit | notify | enforce	Was der periodische Abgleich mit einem laufenden Image ohne Admission dahinter tut.
breakGlass	enabled, reason, approvedBy, auditRef, expiresAt,	Eine benannte, genehmigte Ausnahme, die abläuft. Jede Nutzung wird aufgezeichnet.

	affectedImages	
triageDefaults	maxDuration, requireJustification	Wie lange ein akzeptiertes Risiko stehen darf und ob es eine Begründung braucht.
storageUnreachablePolicy	Allow | Deny	Die Admission-Antwort, solange die Datenbank nicht erreichbar ist.

Was ein Falco-Event auslöst, gehört nicht zur Admission-Policy. Das legen Runtime-Policies in der Konsole fest (Kapitel 8).

Break-Glass

Wenn die Produktion steht und die Lösung ein Image ist, das die Policy ablehnt, lässt Break-Glass die benannten Images herein, zeichnet jede Nutzung auf und läuft von selbst ab:

```
breakGlass:
  enabled: true
  reason: "INC-2417: Rollback-Image hat einen bekannten High-Befund"
  approvedBy: "Bereitschaft, Incident Commander"
  auditRef: "INC-2417"
  expiresAt: "2026-09-01T06:00:00Z"
  affectedImages:
    - registry.example.com/payments/api:1.41.2
```

Jede Admission, die nur dank Break-Glass durchkam, ist im Audit-Trail und in Compliance-Berichten so gekennzeichnet.

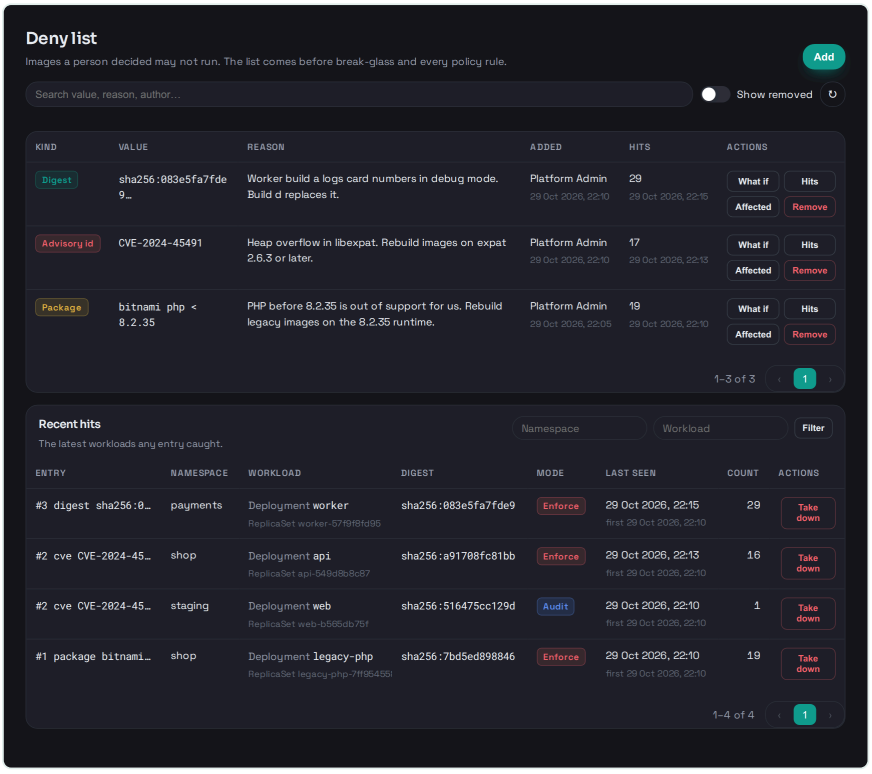
Die Images, die Break-Glass abdeckt, werden ohne die Image-Regeln der Policy zugelassen: die Registry-Allowlist, die Regeln zu `:latest` und zum Digest-Pinning, das Scan-Ergebnis und seine Regeln, Signaturen, Attestations und Build-Provenance, das Image-Kontingent und die Node-Obergrenze. Die Deny List und die Workload-Härtung gelten weiter. `affectedImages` wird über den Image-Namen verglichen, sodass `nginx`, `docker.io/library/nginx` und `nginx:1.27` dasselbe Image meinen und ein Tag in der Liste sie nicht einschränkt. Eine leere Liste deckt jedes Image ab. Vor 1.4.0 wurde die Liste beim Öffnen von Break-Glass nicht gespeichert, ein auf benannte Images begrenztes Fenster deckte daher jedes Image der Policy ab.

7. Die Deny List

Eine Policy beantwortet eine Frage automatisch: Erfüllt dieses Image die Regeln? Ein Eintrag in der Deny List beantwortet eine andere: Hat jemand entschieden, dass dieses Image nicht laufen darf? Die Deny List ist der Ort, an dem diese Entscheidung festgehalten wird, und die Admission prüft sie vor allem anderen.

Es gibt eine Liste pro Cluster, unter Security > Deny list. Das Feld `denyList` jeder Policy legt fest, was ihre Namespaces bei einem Treffer tun. Ein Eintrag wird nie gelöscht. Entfernen hebt ihn auf, und die Zeile bleibt mit der Angabe, wer ihn wann und warum entfernt hat. Hinzufügen und Entfernen sind Aufgaben eines Administrators, und beides verlangt eine Begründung.

> **Eine leere Liste ändert nichts.** Ohne aktiven Eintrag löst die Admission keinen zusätzlichen Digest auf und lehnt nichts ab. Das Upgrade auf 1.4.0 ändert keine Entscheidung, bis jemand einen Eintrag hinzufügt.



Security > Deny list mit drei Einträgen: ein Digest, die Advisory CVE-2024-45491 und das Paket `bitnami php < 8.2.35`. Jede Zeile nennt Begründung, wer den Eintrag wann hinzugefügt hat und wie viele Treffer er hatte. *Recent hits* darunter nennt die Workloads, die die Einträge erfasst haben, nach Namespace und Digest, und ob die Policy des Namespace die Liste erzwungen oder, wie in `staging`, nur auditiert hat.

Arten von Einträgen

Art	Trifft	Beispiel
Digest	Genau einen Image-Build, verglichen mit dem Digest, den die Admission für den Pod aufgelöst hat, und mit dem Image-Index oder den Plattform-Manifesten, zu denen er bekanntermaßen gehört.	`sha256:9f2a…d4e5`
Advisory-ID	Jedes Image, dessen letzter Scan das Advisory meldet, ob es eine Korrektur gibt oder nicht.	`CVE-2021-44228`, `GHSA-jfh8-c2jp-5v3q`
Paket	Jedes Image, dessen letzter Scan das Paket im Versionsbereich listet.	`maven org.apache.logging.log4j:log4j-core < 2.17.1`
Hash einer ausführbaren Datei	Jedes Image, das eine Datei mit diesem sha256 ihrer Bytes enthält, unter welchem Namen und Pfad auch immer.	ein sha256
Pfad einer ausführbaren Datei	Jedes Image mit einer ausführbaren Datei unter einem Pfad oder mit einem Dateinamen irgendwo. Eine Pfadregel trifft auch Basis-Images.	`/bin/sh`, `/usr/local/bin/*`, `nc`
Paket, Ökosystem `bitnami`	Jedes Image mit einer Runtime, deren aus der ausführbaren Datei selbst gelesene Version im Bereich liegt. Trifft nie ein Paket der Distribution.	`bitnami php < 8.2.35`

Ein Advisory-Eintrag akzeptiert die Schemata, unter denen Scanner Befunde melden, denn für manche Befunde ist diese ID der einzige Name: CVE, GHSA, GO, PYSEC, RUSTSEC, OSV, MAL, GSD, DLA, DSA, ELA, USN, RHSA, RHBA, ALSA, ALBA, ELSA, CGA, BIT, TEMP, SUSE-SU und openSUSE-SU. Alles andere wird beim Hinzufügen abgelehnt, denn eine ID, die kein Scanner meldet, träfe nichts.

Ein Paketeintrag nennt ein Ökosystem, ein Paket und einen Operator: unter, bis, genau, ab, über, zwischen (untere Grenze eingeschlossen, obere ausgeschlossen) oder jede Version. Zwei Bereiche sind zwei Einträge. Versionen werden mit dem Vergleicher verglichen, den der Scanner selbst für das Ökosystem verwendet: npm, Go, Cargo, Composer, NuGet, Maven, PyPI, RubyGems, deb, apk und rpm. Eine Version, die der Vergleicher nicht lesen kann, wird beim Hinzufügen abgelehnt, statt als Text verglichen zu werden, wobei 10.0 vor 9.0 einsortiert würde. Ein Eintrag mit *über* oder *ab* deckt auch jede Version ab, die nach dem Hinzufügen erscheint, und What if sagt das.

Wo sie in der Entscheidung steht

Die Reihenfolge ist: Deny List, dann Break-Glass, dann die Regeln der Policy. Ein gelistetes Image wird abgelehnt, auch wenn sein Scan sauber ist und auch während Break-Glass offen ist. Der Eintrag ist eine bewusste Entscheidung und steht über der Notfalltür. Die Ablehnung lautet:

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

Die Regeln ohne Registry-Aufruf, also die Registry-Allowlist, `blockLatestTag` und ein verlangter Digest, werden zuerst geprüft. Unter einer durchsetzenden Policy steht ihre Ablehnung für sich. Unter einer Audit-Policy geht die Prüfung weiter zur Deny List, sodass ein gelistetes Image auch dort abgelehnt wird.

`denyList` in der Policy	Was ein Treffer in ihren Namespaces bewirkt
`enforce` (Standard)	Lehnt den Pod ab, auch wenn die Policy im Modus Audit steht. Ein leeres oder fehlendes Feld gilt als `enforce`.
`audit`	Lässt den Pod mit der Warnung *would be denied by deny list entry …* zu und zeichnet den Treffer auf.
`"off"`	Die Liste wird nicht befragt.

Schreiben Sie `off` in YAML mit Anführungszeichen: Ein nacktes `off` ist für einen YAML-1.1-Leser ein Boolean. Nach dem Upgrade legt jede Policy eine neue Revision an, denn das Feld gehört zu dem Inhalt, den das Ledger hasht.

Welche Anfragen sie prüft. Jeden Container, Init-Container und Ephemeral Container eines neuen Pods. Bei `kubectl debug` nur den Container, den die Anfrage hinzufügt. Andere Änderungen an einem bestehenden Pod, ein Label, eine Annotation oder ein Finalizer, werden weder abgelehnt noch gezählt. Ein Dry Run wird wie eine echte Anfrage beantwortet und hinterlässt keinen Treffer.

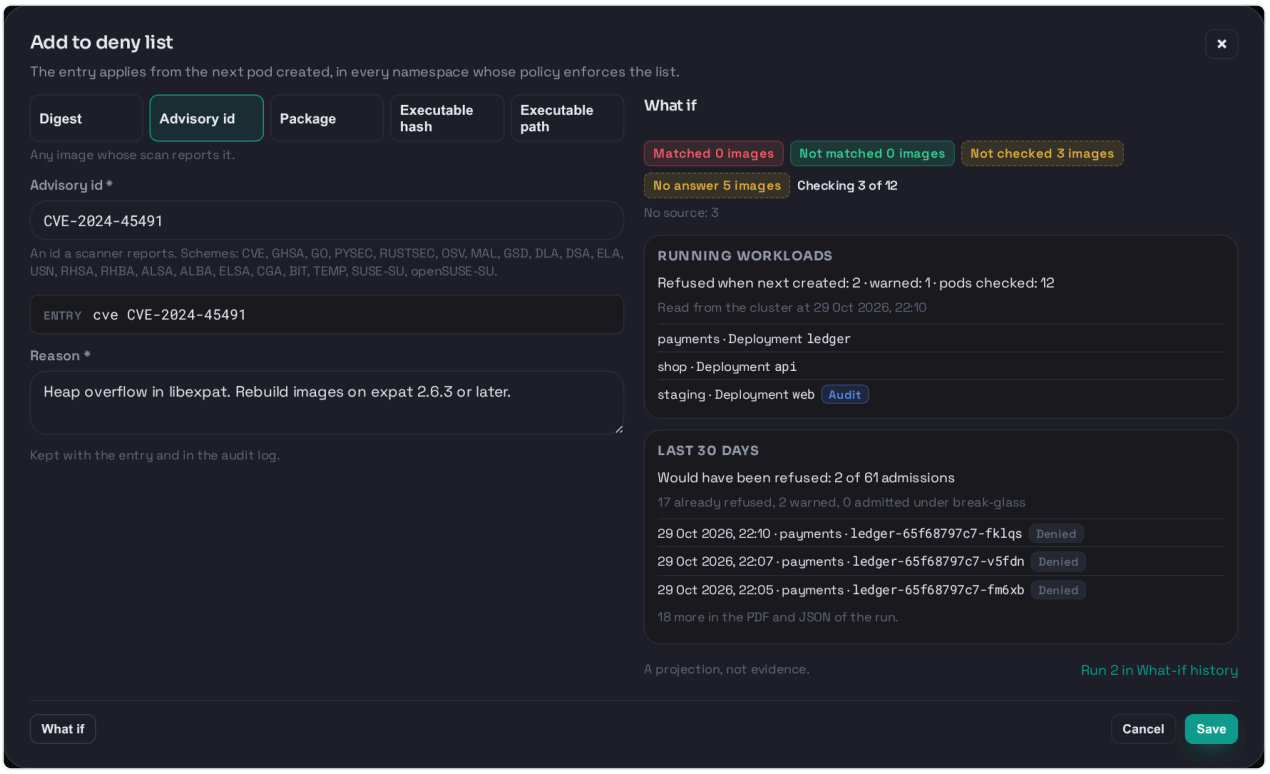
Nichts Laufendes wird gestoppt. Ein Eintrag lehnt einen Workload ab, wenn das nächste Mal einer seiner Pods angelegt wird. Der Controller des Workloads versucht es mit seinem eigenen Back-off weiter, bis sich sein Image ändert.

Wenn sich ein Digest nicht auflösen lässt. Solange die Liste einen Eintrag hat, löst die Admission den Digest des Pods vor Break-Glass auf. Antwortet die Registry nicht, wird ein Pod unter einer durchsetzenden Policy mit `denyList: enforce` mit *deny list could not be evaluated: digest unresolved* abgelehnt. Unter einer Policy im Audit-Modus oder mit `denyList: audit` wird er mit dieser Warnung zugelassen. Ein Image, das nur die Nodes ziehen können, über Node-Anmeldedaten statt über ein Pull-Secret, auf das der Pod oder sein Service Account verweist, kann der Operator nicht auflösen. Es wird in durchsetzenden Namespaces abgelehnt, solange die Liste nicht leer ist. Pinnen Sie es per Digest oder verweisen Sie im Pod auf ein Pull-Secret. Ist die Datenbank nicht lesbar, wird die Liste nicht angewendet, und der Pod trägt die Warnung *deny list unavailable*.

What if, bevor der Eintrag hinzukommt

What if probiert einen Eintrag aus, bevor jemand es bei der Admission erfährt: im Dialog Add vor dem Speichern oder aus der Zeile eines Eintrags. Es beantwortet zwei Dinge nebeneinander.

- **Laufende Workloads, die er abdeckt**, in diesem Moment aus dem Cluster gelesen und nach dem Workload gruppiert, zu dem jeder Pod gehört: das Deployment über seinem ReplicaSet, der CronJob über seinem Job, ein StatefulSet, ein DaemonSet oder der Pod selbst. Auf einem großen Cluster endet das Lesen nach 20 Sekunden, 20.000 Pods oder 2.000 verschiedenen Images und meldet dann, dass es unvollständig ist und wie viele Pods es geprüft hat.
- **Vergangene Admissions, die er abgelehnt hätte**, standardmäßig aus den letzten 30 Tagen, höchstens 90: Die Zeilen des Ledgers werden unter der heutigen Policy jedes Namespace neu entschieden, mit diesem Eintrag als ganzer Liste. Unter Break-Glass zugelassene Admissions werden mit entschieden, denn Break-Glass umgeht die Deny List nie.



What if im Dialog zum Hinzufügen, bevor der Eintrag gespeichert ist. Von den laufenden Workloads würden 2 beim nächsten Erstellen abgelehnt und 1 gewarnt, weil `staging` die Liste nur auditiert. In den letzten 30 Tagen wären 2 von 61 Admissions abgelehnt worden, neben 17 bereits abgelehnten und 2 gewarnten. Die Prüfung der ausführbaren Dateien lief noch, 3 von 12 Images, und die Chips zeigen, wie viele Images noch keine Antwort hatten. Der Lauf bleibt als Lauf 2 in der What-if history.

Nichts wird gestoppt, nichts gespeichert. Ein Lauf eines Deployers oder Administrators wird unter Security > What-if history mit dem Typ *Deny list* aufbewahrt, neunzig Tage wie jeder Lauf und höchstens die letzten fünfhundert Läufe der Deny List, getrennt von den Policy-Läufen gezählt. Er lässt sich als PDF oder JSON herunterladen, als Projektion, nicht als Nachweis, und kommt nie in ein Nachweispaket.

Treffer

Jede Ablehnung und jede Audit-Warnung eines Eintrags wird dem Workload zugerechnet: eine Zeile pro Eintrag, Namespace, Workload und Digest, mit dem ersten und letzten Auftreten und der Anzahl. Die Pods eines Deployments zählen für das Deployment, die eines CronJobs für den CronJob, ein Rollout beginnt also keine neue Zeile. Die Seite der Deny List zeigt unter der Liste die jüngsten Treffer aller Einträge, und *Hits* in einer Zeile zeigt die dieses Eintrags, filterbar nach Namespace oder Workload. Die Ledger-Zeile jeder Ablehnung nennt den Eintrag, der entschieden hat.

Ausführbare Dateien: was ein Scanner nicht listet

Trivy liest, was Paketmanager verzeichnet haben. Eine aus dem Quellcode gebaute und in ein Image kopierte Runtime hat keinen solchen Eintrag, Trivy listet für sie also kein Paket. Ein aus dem php.net-Archiv gebautes PHP 8.2.12 in einem Image ohne Labels zeigt 91 Debian-Pakete und kein php. Die Einträge für ausführbare Dateien und das Ökosystem `bitnami` beantwortet ein zweites Werkzeug im Operator-Image, `attestkeep-binscan`, das die ausführbaren Dateien jedes Images liest. Der Scanner startet es so, wie er Trivy startet.

- **Nur bei Bedarf.** Es läuft nur, solange die Liste einen Eintrag enthält, den es beantwortet: einen Hash oder Pfad einer ausführbaren Datei, ein `bitnami`-Paket oder eine Advisory-ID. Ohne solchen Eintrag läuft es nie, und kein Image wird ein zweites Mal gezogen.
- **Nur Bytes.** Das Image wird per Digest mit den eigenen Pull-Secrets des Workloads gezogen, seine Layer werden von oben nach unten gelesen, gelöschte Dateien entfernt, und jede ELF-Datei wird gehasht. Name, Labels, Umgebung und Basis-Image des Images werden nie gelesen.
- **Versionen aus der Datei selbst.** Die Version einer Runtime wird aus den schreibgeschützten Daten der ausführbaren Datei gelesen und durch eine zweite Zeichenkette in derselben Datei bestätigt. Die Produkte sind php, node, python, ruby,

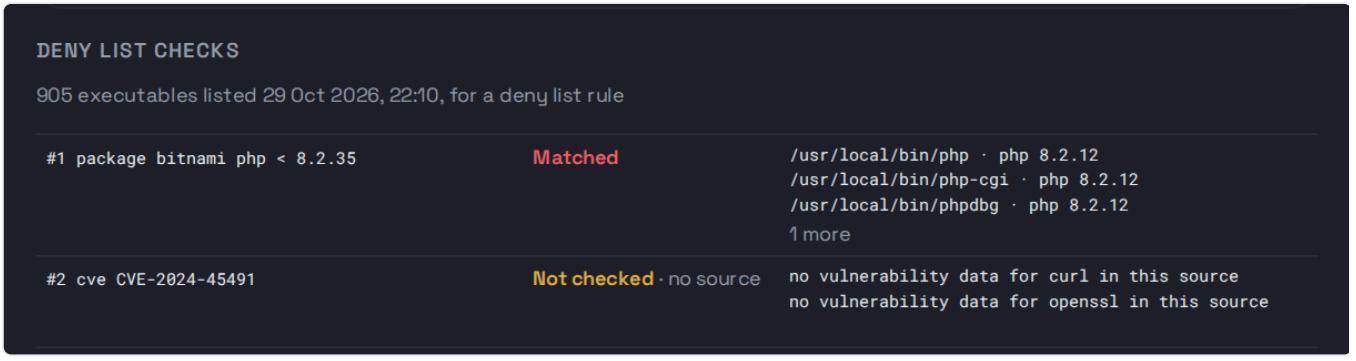
nginx, apache, postgresql, curl und openssl. Ein gestrippter Build wird trotzdem gelesen: `strip` entfernt Symbole, keine schreibgeschützten Daten.

- **Drei Antworten.** Getroffen, mit der Datei, ihrem sha256, der Version und der Signatur, die sie gefunden hat. Nicht getroffen. Oder nicht geprüft, mit dem Grund: Anmeldedaten, Registry, Image weg, Limit, Werkzeugfehler oder keine Quelle. Nicht geprüft zählt nie als sauber.

Grenzen des Durchlaufs. Eine ELF-Datei über 256 MiB wird gehasht, aber nicht auf Versionen gelesen, höchstens 4.000 ELF-Dateien werden pro Image gelesen, und ein Durchlauf endet nach 60 Sekunden. Diese Grenzen sind der Chart-Wert `binscanLimits`. Eine ganze Prüfung aus Ziehen, Durchlauf und Versionsabgleich hat eine eigene Frist, `scan.binscanTimeout` (10 Minuten). Vor dem Herunterladen addiert die Prüfung die Layer-Größen des Images, und ein Image über ihrem Plattenbudget wird nicht heruntergeladen. Das Budget ist die `/tmp`-Grenze des Scanners, `scan.tmpSizeLimit` (4Gi), geteilt durch seine Worker, oder `scan.binscanDiskBudget`, wenn gesetzt. Eine Antwort, die an einer Grenze oder der Frist endete, lautet *not checked (limit)* und nennt sie. Ein Administrator kann ein Image, das nicht geprüft werden konnte, mit *Check again* unter den geltenden Grenzen erneut prüfen lassen.

Wann es läuft. Ein neuer Eintrag wird zuerst gegen die Images laufender Pods geprüft, dann gegen die in den letzten 90 Tagen gescannten Digests. Ein neues Image wird bei seinem Scan geprüft, und laufende Images, die ein Eintrag noch nicht erreicht hat, werden stündlich geprüft. Ein Pod, dessen Image für einen Eintrag zu ausführbaren Dateien oder `bitnami` noch keine Antwort hat, wird wie ein nicht gescanntes Image entschieden: `DenyUntilScanned` lehnt ab, `AllowAndScan` lässt mit Warnung zu. Ein Advisory-Eintrag wartet nicht: Trivys Befunde entscheiden sofort, und ein Treffer, den die Prüfung später findet, kommt nur hinzu.

Advisory-IDs erreichen auch diese Runtimes. Mit einem Advisory-Eintrag auf der Liste werden die gefundenen Runtimes mit den Bitnami-Schwachstellenbereichen in der Trivy-Datenbank abgeglichen, die der Operator ohnehin hält. Eine Runtime, die diese Bereiche nicht abdecken, etwa curl oder openssl, lautet "no vulnerability data in this source", nie "not matched".



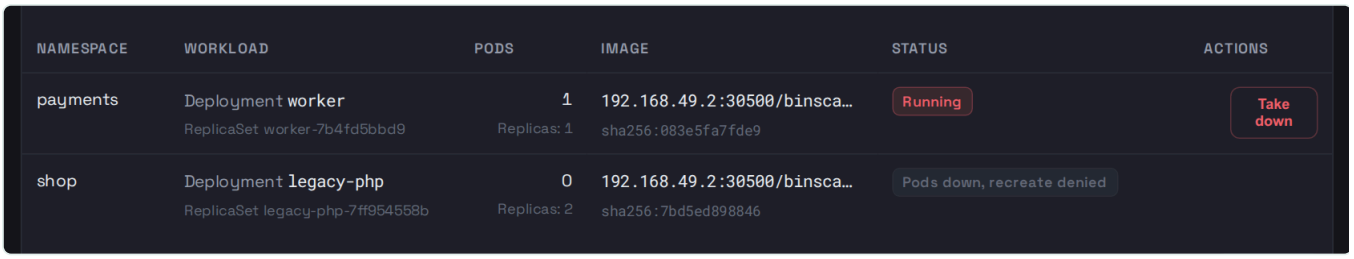
Deny list checks im Detail von `php:8.2.12-cli`, aus einer Liste von 905 ausführbaren Dateien. Eintrag 1, `bitnami php < 8.2.35`, trifft `/usr/local/bin/php` und drei weitere Dateien in Version 8.2.12. Eintrag 2, eine Advisory-ID, zeigt *Not checked*: Die Quelle hat keine Schwachstellendaten für das gefundene curl und openssl.

Was es nicht sieht: eine ausführbare Datei mit entfernter oder veränderter Versionszeichenkette, eine ohne Section Header, eine statisch gelinkte und gestrippte Datei, die nur über einen Symbolnamen erkennbar wäre, gepackte ausführbare Dateien, Runtimes, die beim Start des Containers heruntergeladen werden, Versionen außerhalb ausführbarer Dateien, interpretierten Anwendungscode und Images, die sich nicht ziehen lassen. Fehlende Section Header, eine gepackte Datei und ein nicht ziehbares Image erscheinen als nicht geprüft, mit der Ursache.

Betroffene Workloads und Take Down

Ein Eintrag lehnt neue Pods ab. Was schon läuft, stoppt er nie, und kein Policy-Modus tut das von selbst. *Affected* in der Zeile eines Eintrags listet live aus dem Cluster die Workloads, die er jetzt erfasst, und die, die er abgelehnt hat, jeweils mit einem Status:

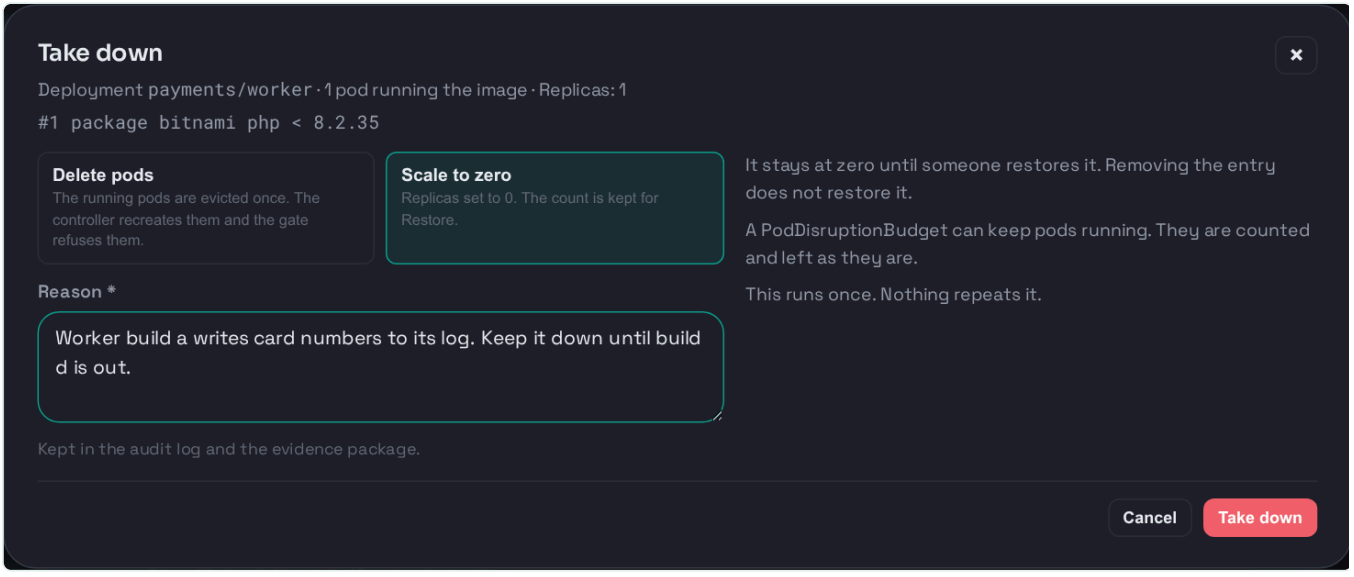
- **Running**: Pods mit einem gelisteten Image laufen.
- **Pods down, recreate denied**: Keiner läuft, und die neuen Pods des Controllers werden abgelehnt.
- **Scaled to zero**: Ein Take Down hat ihn auf null Replicas gesetzt, und er existiert noch.
- **Restored externally**: Er wurde hier auf null skaliert und verlangt wieder Replicas. Jemand oder ein GitOps-Werkzeug hat sie zurückgesetzt. Er wird angezeigt und in Ruhe gelassen.



Affected, live aus dem Cluster gelesen. Das Deployment `worker` in `payments` führt noch ein gelistetes Image aus, darum wird *Take down* angeboten. `legacy-php` in `shop` hat keinen laufenden Pod, und seine neuen Pods werden abgelehnt.

Mit `rbac.denyList.takedown: true` im Chart kann ein Administrator einen laufenden Workload mit Begründung per *Take down* herunterfahren. Das wirkt einmal, und nichts wiederholt es. Der Take Down wird aufgezeichnet, bevor der Cluster berührt wird, sein Ergebnis danach: erledigt, von einem Budget blockiert oder mit Fehler gescheitert.

- **Delete pods** (Standard) verdrängt die Pods mit einem gelisteten Image einmal. Der Sollzustand bleibt unberührt. Der Controller legt Ersatz an, die Admission lehnt ihn ab, solange der Eintrag steht, und der Workload erscheint als degradiert, bis sich sein Image ändert oder der Eintrag entfernt wird. Dann heilt er von selbst.
- **Scale to zero** setzt bei einem Deployment, StatefulSet oder ReplicaSet die Replicas über die Scale-Subressource auf 0 und merkt sich die vorherige Zahl. Es lehnt ab, wenn die Zahl nicht mehr die ist, die der Dialog zeigte. **Restore** setzt die Zahl zurück. Das Entfernen des Eintrags tut das nicht: Nichts skaliert einen Workload ohne einen Klick. Ein DaemonSet, ein Job, ein CronJob oder ein einzelner Pod wird durch Löschen seiner Pods heruntergefahren.

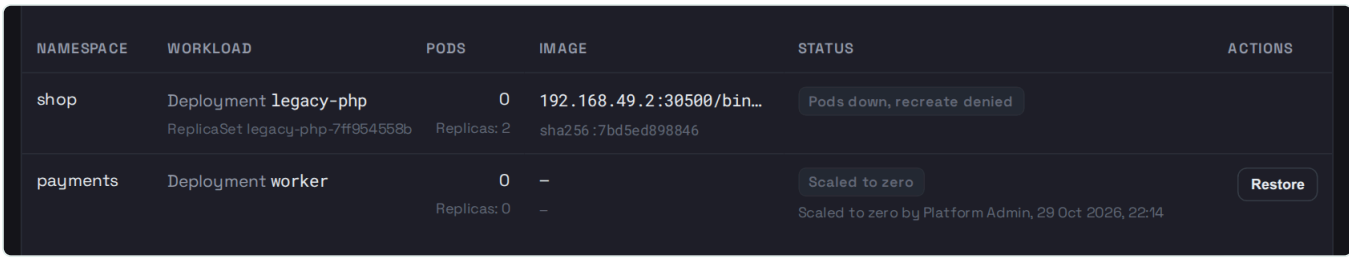


Take down für das Deployment `payments/worker` mit einem laufenden Pod. Gewählt ist *Scale to zero*, die Replica-Zahl wird für Restore aufbewahrt. Die Hinweise sagen, dass das Entfernen des Eintrags nichts wiederherstellt, dass ein PodDisruptionBudget Pods weiterlaufen lassen kann und dass die Aktion einmal läuft und nichts sie wiederholt. Die Begründung landet im Audit-Log und im Nachweispaket.

PodDisruptionBudgets werden respektiert. Pods werden über die Eviction-API verdrängt, und ein Pod, den ein Budget schützt, läuft weiter. Die Zeile lautet dann "Blocked by PodDisruptionBudget" mit der Zahl der Pods, die es behalten hat, und nichts erzwingt oder wiederholt es.

GitOps. Ein Workload, den Argo CD oder Flux verfolgt, ist im Bereich markiert, und der Dialog warnt, dass das Werkzeug ein Scale to zero zurücksetzen wird. Geschieht das, lautet die Zeile *Restored externally*, und Attestkeep skaliert ihn nie wieder herunter. Die dauerhafte Lösung liegt in der Quelle: das Image ändern oder die Replicas senken.

Ohne `rbac.denyList.takedown: true` ist die Schaltfläche deaktiviert, und die API antwortet mit `409`. Ein Namespace, den die Admission nicht prüft, der eigene des Operators oder einer, den der Webhook auslässt, wird nie heruntergefahren: Seine neu angelegten Pods würden nicht geprüft, ein Take Down startete also nur dasselbe Image neu.



Nach dem Take Down zeigt die Zeile *Scaled to zero*, wer es wann getan hat, und bietet *Restore* an. Restore setzt die aufbewahrte Replica-Zahl zurück. Sonst skaliert nichts den Workload wieder hoch.

Benachrichtigungen, Events und Nachweis

Eine Ablehnung hinterlässt ein Kubernetes-Warning-Event mit dem Grund `DenyListRefused`, das den Eintrag nennt, am Workload, sodass `kubectl describe` sagt, warum seine Pods nicht starten. Bei einem Pod ohne Eigentümer landet es am Namespace. Es gibt höchstens eines pro Workload und Eintrag alle fünf Minuten. Der ImageSecurityReport eines abgelehnten Images trägt `status.denyList` mit Eintrag und Ablehnung. Für *Deny list hit* abonnierte Kanäle werden informiert, wie Kapitel 5 beschreibt.

Das Nachweispaket hat einen Abschnitt *Deny list*: die Einträge, die am Ende des Zeitraums gelten, und die im Zeitraum hinzugefügten oder entfernten, mit wer, wann und warum; die Admissions, die die Einträge abgelehnt und durchgelassen haben; die Ausnahmen, also Treffer, die durchkamen, weil ein Namespace die Liste nur auditiert; jeden Take Down und jede Wiederherstellung mit Begründung; und bei einem Eintrag für ausführbare Dateien die Datei, ihren Hash und die Version. Der Policy-Hash deckt die Liste nicht ab. Der Eintrag, der über eine Admission entschied, steht in deren Datensatz.

Wer was darf

Die Liste, ihre Treffer und ihre betroffenen Workloads lesen und What if ausführen verlangt dieselbe Berechtigung wie das Lesen einer Policy. Einen Eintrag hinzufügen und entfernen, einen Workload herunterfahren und wiederherstellen sind Aufgaben eines Administrators, und alles außer der Wiederherstellung verlangt eine Begründung. Auch Check again ist eine Aufgabe eines Administrators. Das Audit-Log führt sie als `deny_list_add`, `deny_list_remove`, `deny_list_takedown`, `deny_list_restore` und `deny_list_check_again`. Die API liegt unter `/api/v1/deny-list`. Ein Feld, das der Server ablehnt, kommt als `400` mit dem Feldnamen zurück, und ein zweiter aktiver Eintrag mit demselben Wert ergibt `409`.

8. Runtime-Events aus Falco

Falco beobachtet, was Container nach dem Start tun. Attestkeep kann Falcos Events empfangen, jedes dem Image und Pod zuordnen, aus dem es kam, und über Runtime-Policies, die Sie in der Konsole schreiben, darauf reagieren.

Attestkeep liefert Falco nicht aus, installiert, konfiguriert und beschreibt es nicht. Die Erkennung leisten Ihr Falco und seine Regeln. Ein Runtime-Event ändert nie eine Admission-Entscheidung. Eine Runtime-Policy kann benachrichtigen, ein Issue öffnen und den Pod stoppen, aus dem das Event kam.

So funktioniert es

1. Auf einem Node feuert eine Falco-Regel, etwa wenn jemand eine interaktive Shell in einem Container öffnet.
2. Das Event geht als JSON per Mutual TLS an den Integrationsport des Operators. Falco sendet es aus seinem eigenen `http_output`, oder Falcosidekick leitet es aus seinem Webhook-Ausgang weiter. In beiden Fällen legt der Absender ein Client-Zertifikat vor, das der Operator ausgestellt hat.
3. Der Operator speichert das Event und verknüpft es mit Image und Digest, die der Container ausführt.
4. Die Runtime-Policies werden der Reihe nach geprüft. Die erste aktive Policy, die passt, entscheidet. Passt keine, wird das Event als Datensatz behalten, und es wird nichts gesendet.
5. Event und Entscheidung erscheinen unter Falco > Events, neben dem Image auf der Seite Images und im Nachweispaket.

Voraussetzungen

- Falco, installiert mit dem Upstream-Helm-Chart `falcosecurity/falco`. Die dokumentierten Werte gelten für Chart 9.2.0 (Falco 0.45.0, Falcosidekick 2.32.0). Ältere Versionen wurden nicht geprüft.
- `falco.json_output: true`.
- Netzwerkzugriff von den Falco- oder Falcosidekick-Pods auf den Service des Operators am Integrationsport, standardmäßig 8444.
- Ein Admin-Konto. Nur Admins sehen die Integrationsseite, laden Zertifikate herunter und bearbeiten Policies.
- Für eine Policy, die Pods stoppt: `rbac.allowEnforce: true` im Attestkeep-Chart. `rbac.denyList.takedown` zählt dafür nicht.
- Für Nachrichten und Issues: ein Benachrichtigungskanal oder ein GitHub-, GitLab- oder Jira-Kanal, der Runtime alert abonniert hat.

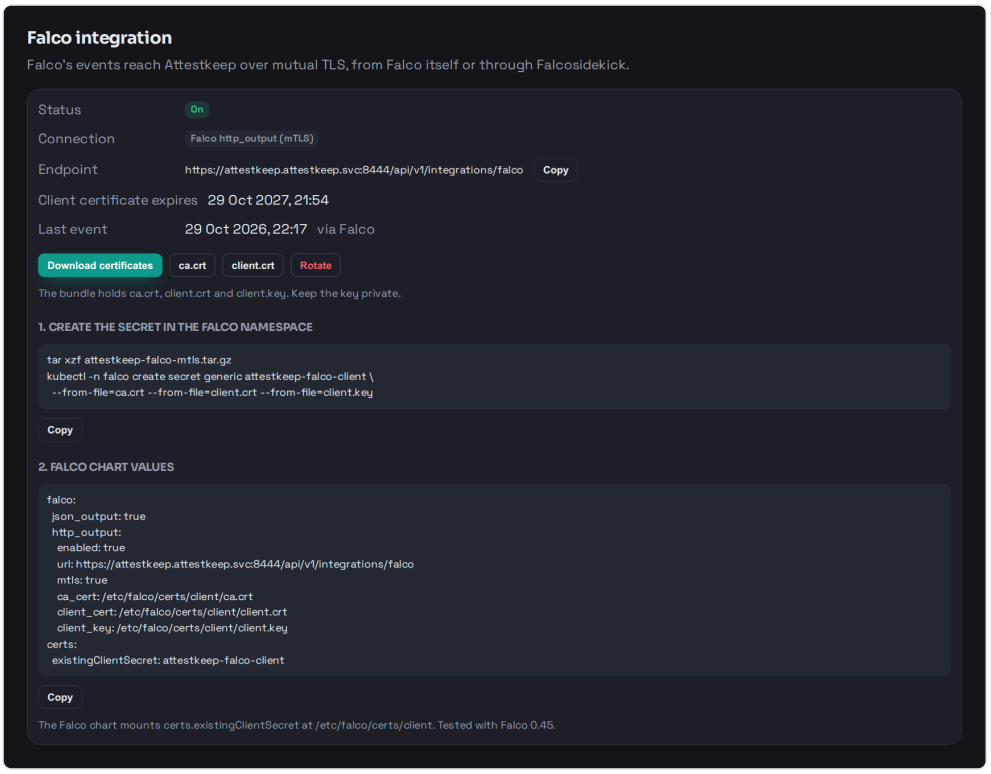
Falco verbinden

Schalten Sie die Integration im Attestkeep-Chart ein und wählen Sie, wie Falco sich verbindet:

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

Mit `mtls`, dem Standard, sendet Falco jedes Event aus seinem eigenen `http_output` direkt an den Operator. Nehmen Sie `sidekick`, wenn Ihr Falco bereits an Falcosidekick sendet. Dessen Webhook-Ausgang leitet dann mit demselben Client-Zertifikat an den Operator weiter. Jeder andere Wert lässt das Chart scheitern.

Ist die Integration aktiv, legt der Operator eine eigene kleine CA und ein Client-Zertifikat an (`CN=falco`, ein Jahr gültig) und hält beide im Secret `<release>-falco-mtls`. Der Integrationsport bedient eine einzige Route, `POST /api/v1/integrations/falco`, und akzeptiert nur ein von dieser CA signiertes Client-Zertifikat. Es gibt kein Token.

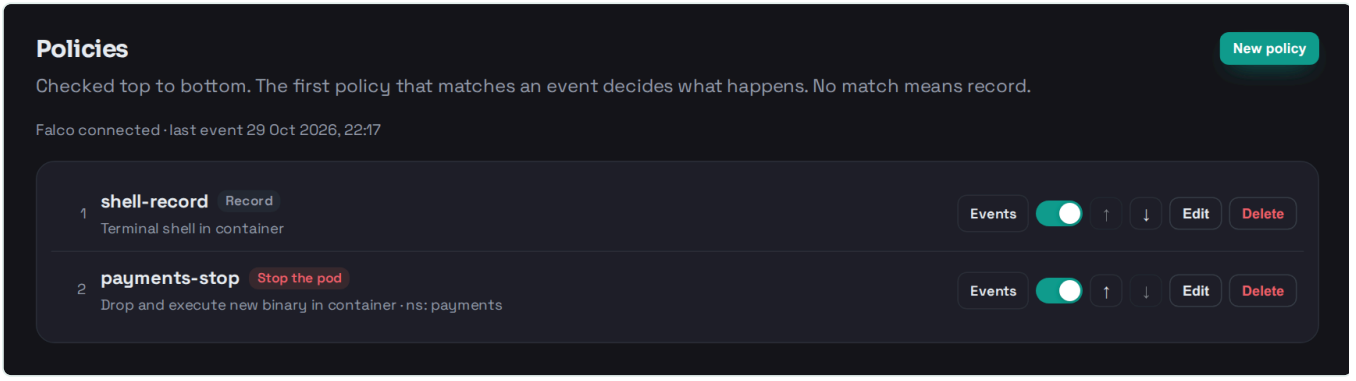


Falco > Falco integration in der Installation dieses Handbuchs. Der Status ist On, die Verbindung läuft über Falcos eigenes `http_output` mit mTLS, und das letzte Event kam direkt von Falco. *Download certificates* speichert ein Bundle mit `ca.crt`, `client.crt` und `client.key`. Darunter stehen der Befehl, der das Secret im Falco-Namespace anlegt, und die Werte des Falco-Charts für die gewählte Verbindung.

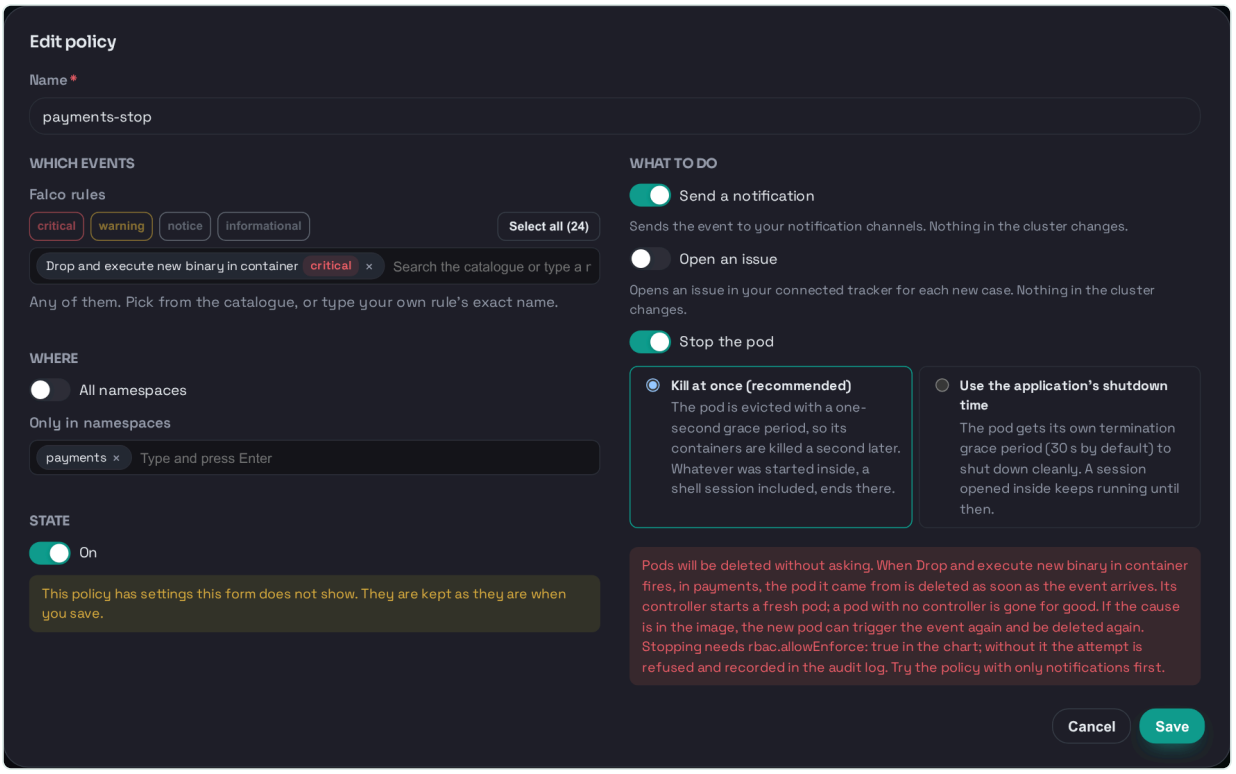
Jeder Download des Pakets landet im Audit-Log. `client.key` muss geheim bleiben, denn wer ihn besitzt, kann Events senden, auf die Ihre Policies reagieren. *Rotate* stellt aus derselben CA ein neues Client-Zertifikat aus. Das alte funktioniert bis zu seinem Ablauf weiter, Events kommen also weiter an, während Sie das Secret austauschen. Wird das eigene Serverzertifikat des Operators erneuert, zeigt die Seite "Certificates changed on <Datum>", bis Sie das Paket erneut herunterladen.

Runtime-Policies

Falco > Policies listet Ihre Runtime-Policies in Reihenfolge. Jedes gespeicherte Event wird von oben gegen sie geprüft, und die erste aktive Policy, die passt, entscheidet. Die Pfeile ändern die Reihenfolge. Eine Änderung gilt sofort auf der Replica, die sie gespeichert hat, und innerhalb von 15 Sekunden auf den anderen.



Zwei Runtime-Policies, von oben nach unten geprüft. `shell-record` zeichnet *Terminal shell in container* auf. `payments-stop` stoppt bei *Drop and execute new binary in container* in `payments` den Pod. Eine Policy trägt hier, unter Events und im Nachweispaket das Etikett ihrer Wirkung.



Das Formular für `payments-stop`: eine Falco-Regel, ein Namespace, *Send a notification* und *Stop the pod* an, *Kill at once (recommended)* gewählt. Der rote Kasten warnt, dass passende Pods ohne Rückfrage gelöscht werden, und sagt seit 1.4.0, dass das Stoppen `rbac.allowEnforce: true` im Chart braucht.

Feld	Wirkung
Name	Pflicht, höchstens 128 Zeichen, eindeutig.
Falco rules	Die Policy trifft Events jeder dieser Regeln, über den exakten Namen. Wählen Sie aus der Liste Rules oder tippen Sie den Namen Ihrer eigenen Regel. Mindestens eine ist Pflicht.
All namespaces / Only in namespaces	Beschränkt die Policy auf Events von Pods in den genannten Namespaces.
Send a notification	Sendet das Event an jeden aktiven Kanal, der Runtime alert abonniert hat.
Open an issue	Öffnet ein Issue über die Tracker-Kanäle, die Runtime alert abonniert haben.
Stop the pod	Stoppt den Pod, aus dem das Event kam, mit *Kill at once* oder *Use the application's shutdown time*.
State	On oder Off. Eine neue Policy startet auf Off. Eine Policy auf Off bleibt gespeichert und tut nichts.

Ist eine der drei Aktionen an, wird die Policy mit der Aktion `notify` gespeichert. Sind alle drei aus, wird sie als `record` gespeichert: Passende Events werden gegen die Policy aufgezeichnet, sonst passiert nichts. Die API (`/api/v1/runtime/rules`) akzeptiert mehr Bedingungen, als das Formular zeigt: Tags, Priorität, Quelle, Ausgabefelder, das letzte Urteil und die Signatur des Images und ein Fenster von `count` Events innerhalb von `duration`. Sie akzeptiert auch die Aktion `allow`, die das Event behält und sonst nichts tut. Hat eine Policy Einstellungen, die das Formular nicht zeigt, sagt das Formular das und behält sie beim Speichern.

Den Pod stoppen

Eine Policy mit *Stop the pod* stoppt den im Event genannten Pod, sobald das Event eintrifft, oder sobald das Fenster erreicht ist, wenn die Policy eines hat. Der Operator bittet den API-Server, den Pod zu verdrängen, PodDisruptionBudgets gelten also. Lässt ein Budget die Verdrängung nicht zu, läuft der Pod weiter, und das Audit-Log hält den Grund fest.

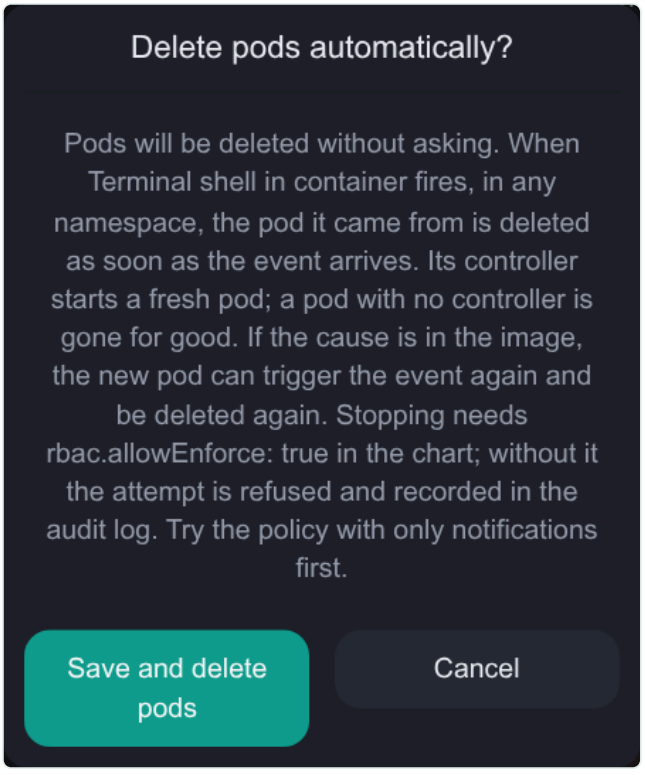
- *Kill at once* (`stop_mode: immediate`, Standard): Die Verdrängung nutzt eine Gnadenfrist von einer Sekunde. Alles, was in den Containern gestartet wurde, auch eine offene Shell, endet dort.

- *Use the application's shutdown time* (`stop_mode: graceful`): Der Pod bekommt seine eigene Termination Grace Period, 30 Sekunden, sofern der Pod keinen anderen Wert setzt. Eine darin geöffnete Shell läuft bis dahin weiter.

Der Controller des Pods startet einen neuen Pod. Ein Pod ohne Controller kommt nicht zurück. Liegt die Ursache im Image, kann der neue Pod dasselbe Event auslösen und erneut gestoppt werden.

- Stoppen braucht `rbac.allowEnforce: true`, ganz gleich, was die ClusterRole sonst erlaubt. Ab 1.4.0 gilt das auch dann, wenn `rbac.denyList.takedown: true` dem Operator die Pod-Verdrängung für Take Down gegeben hat. Ohne den Wert wird das Stoppen abgelehnt, bevor der API-Server gefragt wird, das Audit-Log hält "not stopped: the chart has not granted pod eviction (set rbac.allowEnforce)" fest, und die Meldung sagt, dass der Pod noch läuft. Der Abgleich unter `runtimeReconciliation: enforce` folgt derselben Regel.
- Pods im eigenen Namespace des Operators werden nie gestoppt. Kein anderer Namespace ist ausgenommen, halten Sie eine Stopp-Policy also mit *Only in namespaces* von System-Namespaces fern.
- Das Event muss den Pod nennen (`k8s.ns.name` und `k8s.pod.name`). Mit den Standardwerten des Falco-Charts tut es das.
- Ein Pod, der bereits beendet wird, bleibt unberührt. Jede Operator-Replica bittet höchstens alle 10 Minuten darum, einen bestimmten Pod zu stoppen.

Das Speichern einer Policy mit eingeschaltetem *Stop the pod* fragt vorher nach. Der Dialog sagt, dass Pods ohne Rückfrage gelöscht werden, was mit einem Pod mit und ohne Controller geschieht und dass das Stoppen `rbac.allowEnforce: true` im Chart braucht.



Die Frage, die beim Speichern einer Policy erscheint, in der *Stop the pod* neu eingeschaltet ist. Sie nennt Regel und Namespaces, sagt, was mit einem Pod mit und ohne Controller geschieht, und dass das Stoppen `rbac.allowEnforce: true` braucht. Fehlt es, wird der Versuch abgelehnt und im Audit-Log festgehalten.

Jeder Versuch wird als `runtime_pod_stop` ins Audit-Log geschrieben, mit `policy:<Name>` als Akteur, dem Pod als Ziel und Falco-Regel, Pod-UID, Container, Image, Digest, Node, Event-ID und Ergebnis im Detail. Einen Pod zu stoppen ändert die Admission nicht. Neue Pods desselben Images werden wie bisher zugelassen.

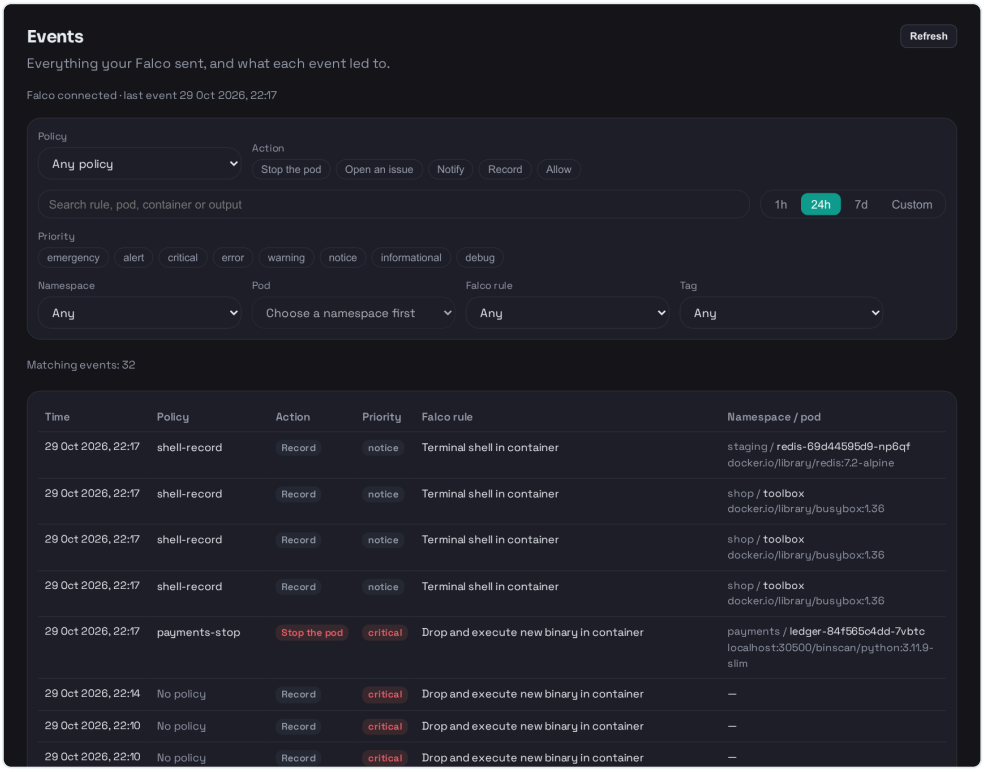
Benachrichtigungen und Issues

Eine Nachricht trägt den Titel `Runtime alert: <Falco-Regel>` und enthält Falcos Ausgabezeile, Regel, Priorität, Quelle, Namespace, Pod, Container, Image, Digest, Node, Zeit, die Policy und ihre Aktion. Hat die Policy den Pod gestoppt, sagt die

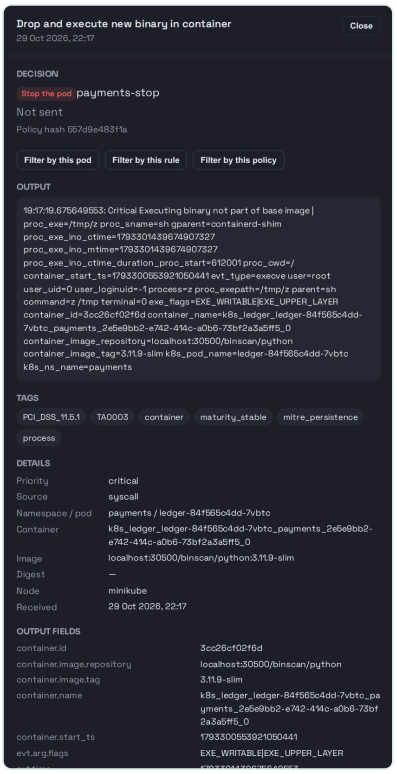
Nachricht das. Eine Policy sendet höchstens eine Nachricht pro Image-Digest und Stunde. Diese Grenze gilt für Nachrichten und Issues. Das Stoppen eines Pods verzögert sie nicht.

Ein Issue wird nur geöffnet, wenn das Event an einen Image-Digest gebunden ist: ein Issue pro Digest und Falco-Regel, mit dem Titel `RUNTIME: <rule> in <image>` und dem Label `runtime`. Es schließt sich selbst als `quiet` nach 7 Tagen, in denen die Regel auf diesem Image nicht feuerte, oder als `image_retired`, wenn 24 Stunden lang kein laufender Pod den Digest verwendete. Keines von beiden heißt, dass das Problem behoben ist. Ein Event gilt nur dann als benachrichtigt, wenn mindestens ein aktiver Kanal es abonniert hat.

Events



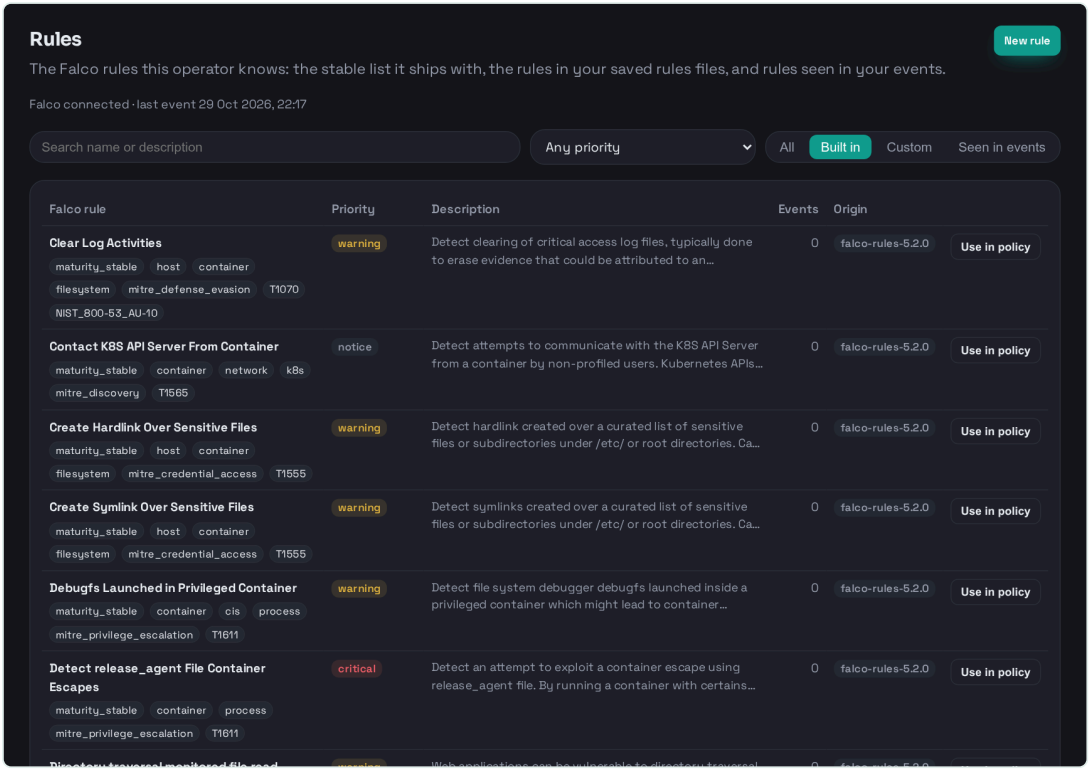
Falco > Events der letzten 24 Stunden: 32 Events. Jede Zeile nennt die passende Policy und ihre Aktion, die Priorität, die Falco-Regel sowie Namespace und Pod. Events ohne passende Policy zeigen *No policy* und bleiben als Aufzeichnung erhalten. Gefiltert wird nach Policy, Aktion, Freitext, Priorität, Namespace, Pod, Falco-Regel und Tag in einem Zeitraum.



Ein geöffnetes Event: *Drop and execute new binary in container* in `payments`, entschieden als *Stop the pod* durch `payments-stop`. Das Fenster zeigt Entscheidung, ob eine Nachricht verschickt wurde, Policy-Hash, Falcos Ausgabezeile, Tags sowie Namespace, Pod, Container, Image und Node, denen das Event zugeordnet ist. In dieser Installation setzt das Chart `rbac.allowEnforce` nicht, also wurde das Stoppen abgelehnt, und der Pod lief weiter.

Falco-Regeln und der Regeleditor (Beta)

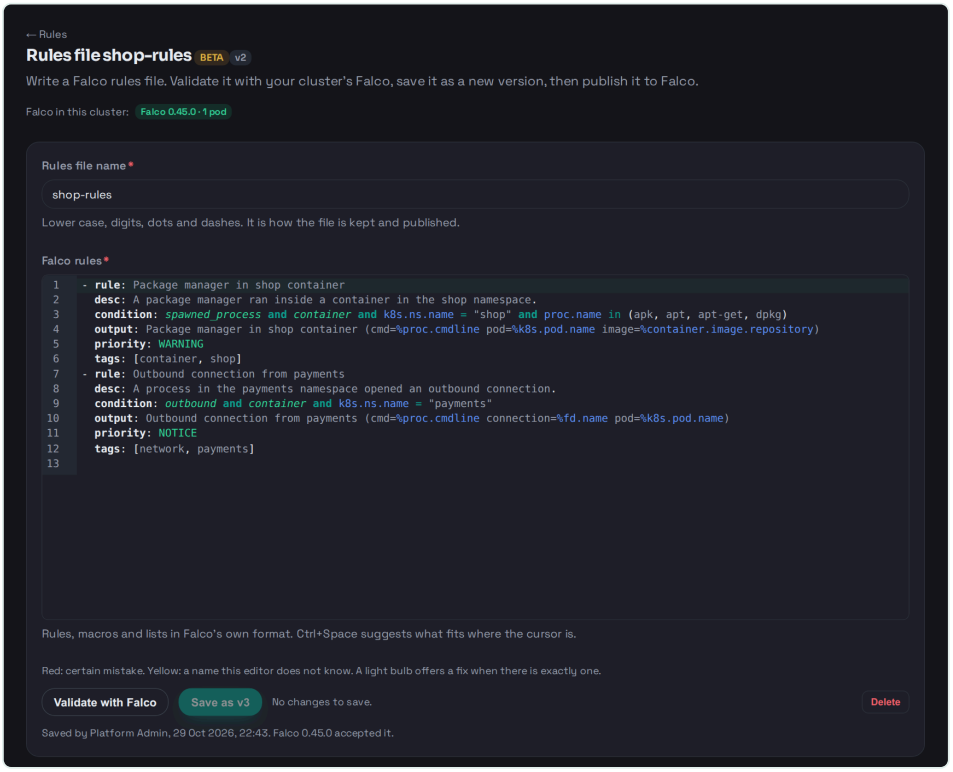
Falco > Rules listet die Falco-Regeln, die der Operator kennt: die 25 stabilen Regeln aus `falcosecurity/rules`, die mit ihm ausgeliefert werden, die Regeln in Ihren gespeicherten Regeldateien und in Events gesehene Regelnamen. *Use in policy* beginnt eine Policy aus einer Regel.



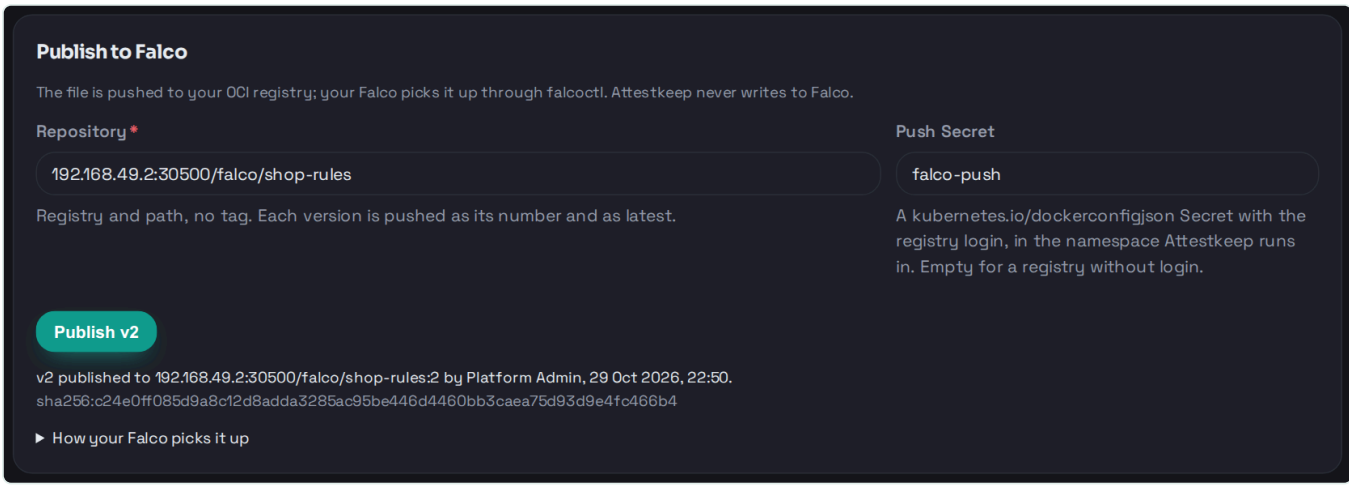
Falco > Rules mit dem Filter *Built in*. Jede Regel zeigt Priorität, Tags, Beschreibung, wie viele Events sie hier erzeugt hat, und ihre Herkunft, `falco-rules-5.2.0`. *Use in policy* beginnt eine Runtime-Policy mit dieser Regel.

New rule (Admins) öffnet einen Editor für eine Falco-Regeldatei: Regeln, Makros, Listen und Overrides in Falcos eigenem YAML. Der Editor ist in der Beta. Vervollständigung und Prüfungen richten sich nach der Falco-Version, die in Ihrem Cluster läuft, gefunden über das Label `app.kubernetes.io/name=falco`.

- *Validate with Falco* startet im Namespace des Operators einen einmaligen Pod mit demselben Image, das Ihr Falco verwendet. Er lädt Falcos Standardregeln und Ihre Datei, startet Falco im Validierungsmodus und wird gelöscht, sobald das Ergebnis gelesen ist. Ihr laufendes Falco wird nicht berührt.
- Speichern ist erst möglich, wenn Falco genau den Text im Editor akzeptiert hat. Der Server prüft ihn vor dem Speichern noch einmal. Jedes Speichern ist eine neue Version, festgehalten mit der Person, die gespeichert hat, und der Falco-Version, die akzeptiert hat. Ein Regelname, den schon eine andere gespeicherte Datei definiert, wird abgelehnt.
- Publish schiebt eine gespeicherte Version als falcoctl-Artefakt `rulesfile` in Ihre eigene OCI-Registry, getaggt mit der Versionsnummer und `latest`. Braucht die Registry eine Anmeldung, geben Sie den Namen eines Secrets vom Typ `kubernetes.io/dockerconfigjson` im Namespace des Operators an. Attestkeep liest es beim Push und speichert die Zugangsdaten nicht.



Die gespeicherte Regeldatei `shop-rules` in Version 2. Die Zeile unter dem Titel zeigt das im Cluster gefundene Falco: Falco 0.45.0 auf 1 Pod. *Save as v3* bleibt deaktiviert, bis sich der Text ändert und Falco ihn annimmt. Die letzte Zeile sagt, wer Version 2 gespeichert hat und welche Falco-Version sie angenommen hat.



Publish to Falco: das Repository in Ihrer eigenen Registry, der Name des Push-Secrets und der Eintrag, dass Version 2 mit ihrem Digest veröffentlicht wurde. *How your Falco picks it up* öffnet die falcoctl-Werte für Ihr Falco-Chart.

Ihr Falco lädt die Datei über falcoctl. Nehmen Sie in den Werten des Falco-Charts Ihre Referenz in `install.refs` und `follow.refs` auf und behalten Sie `falco-rules:5`, die Standardreferenz des Charts. Helm ersetzt Listen, statt sie zusammenzuführen, und ohne diesen Eintrag installiert falcoctl `/etc/falco/falco_rules.yaml` nicht mehr. Fügen Sie dann Ihre Datei nach Falcos eigenen Regeln zu `rules_files` hinzu. Wird eine Datei in der Konsole gelöscht, verschwinden ihre Regeln aus der Liste Rules. Was schon veröffentlicht wurde, bleibt in der Registry und in jedem Falco, das es installiert hat, bis Sie die Referenz aus falcoctl entfernen.

Aufbewahrung und Metriken

Jedes Event wird gespeichert. Attestkeep hat keine eigene Prioritätsschwelle. Gespeicherte Events, die älter als `runtime.retentionDays` sind (standardmäßig 90), werden stündlich entfernt, und `0` behält sie unbegrenzt. Issue-Historie, Audit-Einträge und Policy-Revisionen werden nicht entfernt. Damit Falco gar nicht erst Events sendet, die Sie nie behalten wollen, setzen Sie Falcos eigenes `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}
```

In `attestkeep_runtime_rule_matches_total` ist `rule` der Name der Runtime-Policy. `unresolved_image` heißt nicht, dass das Event verloren ging. Falco beobachtet auch Prozesse auf dem Host, und ein solches Event wird ohne Image gespeichert.

9. Nachweisen, dass durchgesetzt wird

Ein Admission-Webhook, der nicht läuft, sieht genauso aus wie ein regelkonformer Cluster. Diese Prüfung dauert etwa fünf Minuten, braucht nur kubectl und endet mit einer Ablehnung, die Sie selbst gesehen haben. Führen Sie sie nach der Installation aus, nach Upgrades und immer, wenn Sie die Durchsetzung bestätigen müssen.

1. Einen Pod versuchen, den die Policy ablehnen sollte

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

Der API-Server lehnt ab und zitiert den Webhook. So sieht die Ausgabe einer Installation mit der Standard-Policy aus:

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

Die Meldung beginnt mit *Error from server*, die Ablehnung kam also aus der Admission-Kette des API-Servers. Die Begründung nennt die Regel der Policy, und wer sie in einem CI-Log liest, weiß, was zu ändern ist.

2. Bestätigen, dass nichts angelegt wurde

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

3. Prüfen, dass die Entscheidung aufgezeichnet wurde

Die Tabelle *Recently blocked admissions* im Dashboard beginnt jetzt mit dieser Ablehnung und derselben Begründung. Die Metrik `attestkeep_admission_decisions_total{decision="denied"}` ist gestiegen.

4. Wenn der Pod zugelassen wurde

- Der Namespace wird umgangen: kube-system, cert-manager, der eigene Namespace des Operators oder ein Namespace mit dem Label `attestkeep.com/webhook=bypass`.
- Die Policy deckt den Versuch nicht ab. Prüfen Sie `enforcement: Enforce` und `blockLatestTag: true`.
- Der Webhook lässt offen durch. Prüfen Sie, dass `attestkeep_webhook_distrusted` 0 ist und die Pods bereit sind.
- Die Installation ist nicht aktiviert. Bis ein Schlüssel eingegeben ist, wird nichts durchgesetzt, und jede Admission-Antwort sagt das in einer Warnung.

Mit DenyUntilScanned

Unter `admissionTiming: DenyUntilScanned` wird ein neues Image abgelehnt, bis sein Scan vorliegt. Ausgabe einer Community-Installation:

```
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

# kurz danach, derselbe Befehl:
pod/cold-check created
```

10. Nachweispakete und Vertrauensmodell

Ein Nachweispaket ist ein signiertes Dokument pro Auditzeitraum. Es entsteht im Cluster aus Ihren eigenen Aufzeichnungen, und ein Auditor kann es mit eigenen Werkzeugen prüfen. Es ist ein Nachweis für ein Audit, keine Zertifizierung, und das Dokument sagt das.

Kontrollkataloge, auf die es abbildet

Regelwerk	Was abgebildet ist
SOC 2, AICPA Trust Services Criteria	13 Kriterien aus CC6 bis CC9 und Availability A1.
EU CRA und NIS2: (EU) 2024/2847 und Artikel 21(2)	34 Anforderungen aus beiden Rechtsakten.
DORA, Verordnung (EU) 2022/2554	22 Anforderungen aus den Artikeln 8 bis 11 und Artikel 28.
NIST SP 800-190	15 Gegenmaßnahmen für Images, Registries, Orchestratoren und Container.
SSDF, NIST SP 800-218 v1.1	13 Praktiken aus PO, PS, PW und RV.
NIST SP 800-53 Rev. 5	15 ausgewählte Kontrollen aus CM, RA, SI, SR, AU und CP.
ISO/IEC 27001:2022 Anhang A	12 Klauseln: 10 Kontrollen aus Anhang A und als außerhalb des Umfangs markierte Klauseln.
DSGVO Artikel 32	6 Klauseln zur Sicherheit der Verarbeitung.
PCI DSS v4.0.1 (PCI SSC, Juni 2024)	13 Klauseln aus den Anforderungen 6.3, 6.5, 10.2 und 11.3.
MAS Technology Risk Management Guidelines (18. Januar 2021)	30 Klauseln aus den Abschnitten 3.3, 6.1, 6.3, 7.2 bis 7.6, 11.3, 11.4, 12.1, 12.2, 13.1, 13.6 und 15.
ASD Essential Eight Maturity Model (November 2023)	46 Klauseln: Patchen von Anwendungen, Patchen von Betriebssystemen und Application Control, Maturity Level eins bis drei.
ISMAP Control Criteria (japanische Ausgabe vom 1. August 2024)	42 Klauseln aus den Kontrollkriterien der beigefügten Tabelle 3.
ISMS-P Certification Criteria (Bekanntmachung von PIPC und MSIT, Anhang 7, KISA-Leitfaden vom November 2023)	32 Klauseln aus den Bereichen 1 und 2.

Insgesamt dreizehn Regelwerke und 293 Klauseln, jede mit eigener Fundstelle. Die Abbildung steckt im Operator, ein Air-Gapped-Cluster berichtet also genauso wie ein verbundener. Jede Kontrolle endet in einem von vier Zuständen: *met*, *partial*, *gap* oder *outside_scope*. Eine Klausel ist nur dann erfüllt, wenn die Aufzeichnungen des Zeitraums die Kontrolle bei der Arbeit zeigen, nie nur, weil das Produkt die Funktion hat. Eine Klausel, die ein Cluster nicht beantworten kann, wird mit Begründung als außerhalb des Umfangs gemeldet und ist in der Zahl oben enthalten.

Für wie viele Frameworks Sie Pakete erzeugen können, hängt von der Lizenz ab (Kapitel 13).

Übersetzte Quellen. Zwei Abbildungen beruhen auf Texten, die nicht auf Englisch veröffentlicht sind, und jede Klausel sagt das. Für **ISMAP** gibt es eine englische Referenzübersetzung. Wo ihr Eintrag nur auf ISO/IEC 27002 verweist, wird die Pflicht aus der japanischen Ausgabe vom 1. August 2024 übersetzt, und die Fundstelle sagt, welche. Für **ISMS-P** gibt es keinen offiziellen englischen Text. Jede Klausel ist unsere Übersetzung des koreanischen Kriteriums, wie es im KISA-Leitfaden steht. Jedes ISMS-P-Kriterium verbindet eine technische Maßnahme mit einem Verfahren, einer Benennung oder einer regelmäßigen Überprüfung. Die Aufzeichnungen eines Clusters beantworten nur den technischen Teil, kein ISMS-P-Kriterium wird daher als voll erfüllt gemeldet.

Fünf Attestations speisen die neueren Abbildungen. `patch-window-48-hours` ist erfüllt, wenn das angegebene Fenster für kritische Befunde höchstens 48 Stunden beträgt und ausgenutzte Schwachstellen bei der Admission abgelehnt werden. `patch-window-two-weeks` ist erfüllt, wenn das angegebene Fenster für hohe Befunde höchstens zwei Wochen beträgt. `runtime-observed` ist erfüllt, wenn die Falco-Integration aktiv ist und im Zeitraum mindestens ein Event ankam. Es ist partial, wenn die Integration aktiv ist und nichts ankam, denn ein ruhiger Cluster und ein Falco, das nicht mehr sendet, sehen für den Operator gleich aus. Es ist eine Lücke, wenn die Integration aus ist. `producer-statement-honoured` ist erfüllt, wenn die Policy VEX-Aussagen anwendet und mindestens einem Aussteller vertraut. `context-aware-thresholds` ist erfüllt, wenn die Eskalation für `exposed` und `privileged` aktiv ist, partial mit einem der beiden und eine Lücke unter `Ignore`. `remediation-windows` kann *met* nur mit verbundenem Issue-Tracker erreichen, denn nur dann gibt es einen Nachweis, wann eine Korrektur kam.

Wie jede Anforderung beantwortet wurde

Jede Anforderung listet die Tests, die sie beantwortet haben. Ein Test hat eine Methode (`TEST` , wenn er einen Datensatz über Geschehenes liest, `EXAMINE` , wenn er eine Konfiguration liest), ein Ergebnis, eine Beobachtung, Pfade in den Bericht, die gelesene Quelle und den Zeitpunkt. Jede Anforderung nennt außerdem `result` und `implementation_status` in OSCAL-Begriffen. Das PDF druckt die Tests unter jeder Anforderung.

Der Block `generation` sagt, wann und womit das Dokument erzeugt wurde, das Quellsystem, das Abfragefenster (Ende ausgeschlossen, UTC), die Abfrage jedes Abschnitts mit ihrer Zeilenzahl und ob eine begrenzte Abfrage ihr Limit erreichte, und wie Personen benannt werden. Ab 1.3.0 trägt er auch `nodes {count, ceiling}` : die Ready-Nodes des Clusters bei der Erzeugung und die Node-Obergrenze der Lizenz, nur als Zahlen. `admission.records[]` enthält jede Admission-Prüfung des Zeitraums, je eine Zeile und ohne Obergrenze, sodass ein Auditor aus der ganzen Grundgesamtheit Stichproben ziehen kann. `organisation.declarations` enthält Zeitquelle und Aufbewahrungsangaben des Betreibers, gedruckt als Angabe des Betreibers.

Personen werden über ihr Konto benannt, nicht über ihren Namen. `generated_by` , `chain[].actor` für Zeilen des Audit-Logs, `triage[].triaged_by` und `runtime.audit_entries[].actor` enthalten `user:<Konto-ID>` . Ein von einer Runtime-Policy gestoppter Pod wird als `policy:<Name>` geschrieben. Das Audit-Log der Konsole zeigt weiter Namen, und der Betreiber löst darüber eine ID auf, wenn ein Auditor fragt. Vor 1.2.0 aufgezeichnete Zeilen, die nur einen Namen enthalten, werden über den exakten Anzeigenamen oder die E-Mail einem Konto zugeordnet oder als `user:unresolved` geschrieben und gezählt. Die Kubernetes-Identität hinter jeder Admission wird unverändert geschrieben, wenn sie eine `system:` -Identität ist, sonst als `k8s-user:` gefolgt von den ersten 16 Hex-Zeichen ihres SHA-256.

Der Runtime-Abschnitt enthält die Zahl der Events nach Priorität, die häufigsten Falco-Regeln, gesendete Meldungen und geöffnete und geschlossene Runtime-Issues. Er enthält außerdem den geltenden Satz Runtime-Policies mit Revisions-Hash, `audit_entries` für Policy-Änderungen, gestoppte Pods und Änderungen an der Integration im Zeitraum, Trefferzahlen nach Policy und Aktion, die Events, auf die eine Policy reagierte (bis 500, jedes mit dem geöffneten Issue), und die Abdeckung: ob die Integration aktiv ist, das letzte Event und die längste Lücke ohne Events. `event_time` ist Falcos eigener Zeitstempel. `received_at` ist der Zeitpunkt, an dem der Operator das Event empfing. Die Abdeckung verwendet `received_at` , und `coverage.clock` sagt das. `runtime.rules[].effect` und `runtime.decisions[].policy_effect` geben die Aktion an, auf die die Policy eingestellt war. Ob ein Stopp gelang, steht in der Zeile `runtime_pod_stop` in `runtime.audit_entries` .

Externer Export und Deny List. Ab 1.4.0 hat das Paket zwei weitere Abschnitte. *External export* listet die SIEM-Ziele, den letzten Stapel, den jedes angenommen hat, und die im Zeitraum aufgegebenen Events (`external_export` in `report.json`). *Deny list* listet die geltenden Einträge und die im Zeitraum hinzugefügten oder entfernten, die Admissions, die sie abgelehnt und durchgelassen haben, die Ausnahmen, jeden Take Down und jede Wiederherstellung und bei einem Eintrag für ausführbare Dateien die Datei, ihren Hash und die Version (`deny_list` in `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

Abschnitt 14, *Deny list*, desselben Pakets. Am Ende des Zeitraums galten drei Einträge. Sie trafen 75 Admissions: 74 abgelehnt, 1 zugelassen, weil die Policy des Namespace die Liste nur auditiert. Jeder Eintrag zeigt Wert, Begründung, wer ihn wann hinzugefügt hat und wie viele Admissions er abgelehnt und durchgelassen hat. Unter den Ausnahmen steht der Take Down von `payments/Deployment/worker` mit Begründung und Zeitpunkt der Wiederherstellung.

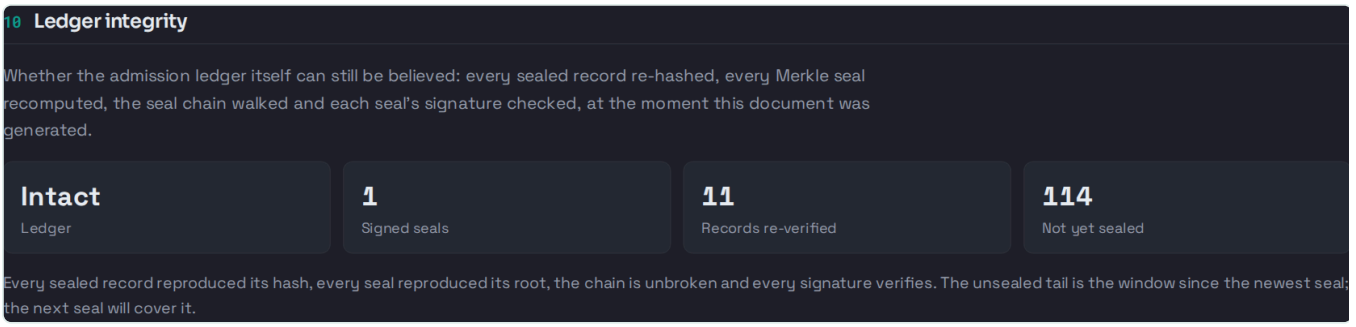
Befunde werden je Digest gezählt. Ab 1.4.0 ist ein Digest, der über zwei Referenzen gezogen wird, Tag und Digest oder zwei Namen, eine einzige Zeile in `findings.critical`, die beide unter `references` nennt, und seine Befunde zählen einmal in `coverage.severity` und `coverage.findings_total`. Ein jetzt für denselben Cluster erzeugtes Paket kann um genau diese Doppelungen niedriger ausfallen als eines von 1.3.7.

Break-Glass und Triage. Ab 1.3.4 wird jede Break-Glass-Aktivierung aus der Policy-Revision gelesen, unter der ihre Admissions entschieden wurden, ein Paket sagt also, was damals genehmigt war, auch nachdem das Fenster abgeschaltet wurde. Erzeugen Sie jedes Paket, das ein vergangenes Break-Glass-Fenster abdeckt, neu, um den vollständigen Datensatz zu erhalten. Triage-Entscheidungen tragen den Schweregrad aus den Scan-Befunden, den höchsten, wenn eine CVE mit mehreren aufgezeichnet wurde.

Integrität des Ledgers

Jede Admission-Entscheidung wird nach PostgreSQL geschrieben. Eine Datenbank ist gut, um Entscheidungen abzufragen, aber für sich allein kein Grund, ihnen zu vertrauen. Das Ledger ist so gebaut, dass Manipulation sichtbar wird:

- **Jeder Datensatz trägt seinen eigenen Hash**, ein SHA-256 über seinen Inhalt (Zeit, Namespace, Pod, Image, Digest, Urteil, Begründung, Name und Hash der Policy, Break-Glass-Kennzeichen), berechnet beim Schreiben.
- **Datensätze werden unter signierten Checkpoints versiegelt**, standardmäßig stündlich oder alle 10.000 Datensätze. Die Leader-Replica bildet eine Merkle-Wurzel über den unversiegelten Bereich und signiert einen Checkpoint, der den Hash des vorherigen enthält, sodass die Siegel eine Kette bilden.
- **Der Signaturschlüssel liegt nicht in der Datenbank.** Er ist ein in Ihrem Cluster erzeugter Ed25519-Schlüssel im Secret `attestkeep-evidence-key`. Ledger-Zeilen umzuschreiben braucht Schreibzugriff auf die Datenbank. Die Siegel neu zu signieren braucht Zugriff auf dieses Secret. Eines von beiden reicht nicht. Ab 1.4.0 ist derselbe Schlüssel auch die Lizenzidentität der Installation (Kapitel 12).
- **Derselbe Schlüssel signiert Ihre Nachweisdokumente** als DSSE-Umschläge, und das Lizenzzertifikat nennt Ihren Schlüssel, sodass ein Dritter ein Dokument Ihrer Installation zuordnen kann.



Die Ledger-Prüfung im Paket. Bei der Erzeugung wurde jeder versiegelte Eintrag neu gehasht, jedes Merkle-Siegel neu berechnet, die Kette abgelaufen und jede Signatur geprüft. Hier: *Intact*, 1 signiertes Siegel und 11 erneut geprüfte Einträge. 114 Einträge entstanden nach dem letzten Siegel und zählen als noch nicht versiegelt, darum liest man *Intact* zusammen mit der Zahl der Siegel.

Wo die Garantien enden

Dieses Verfahren macht Manipulation sichtbar. Es verhindert sie nicht. Datensätze, die seit dem jüngsten Checkpoint geschrieben wurden, sind gezählt, aber noch nicht versiegelt, ein Fenster von höchstens einem Siegelintervall. *Intact* spricht über das Versiegelte, lesen Sie es daher mit der Zahl der Checkpoints. Ein Angreifer mit Schreibzugriff auf die Datenbank und Zugriff auf den Signaturschlüssel kann das Verfahren aushebeln. Die Aussage ist, dass das zwei getrennte Zugriffe braucht. Der Auditor erhält ein signiertes Ergebnis und den DSSE-Umschlag. Die Merkle-Prüfung unabhängig zu wiederholen bräuchte Zugriff auf die Datenbank.

Wird ein Paket für einen Auditor signiert, entsteht neben dem JSON ein PDF, beide zusammen in einem signierten Archiv. Jeder kann ein geliefertes Paket unter attestkeep.com/verify.html mit eigenen Werkzeugen prüfen.

11. Betrieb, SIEM-Export und Backup

Probes

Endpunkt	Was er prüft	Verwendet als
`/healthz`	Nur, dass der Prozess läuft.	Liveness-Probe (Startverzögerung 10s, Intervall 20s)
`/readyz`	Ab 1.4.0, dass der Server antwortet und sein Zertifikat geladen ist. Die Datenbank gehört nicht mehr dazu, eine Replica bleibt also bereit, während die Datenbank nicht erreichbar ist, und der Webhook lehnt weiter nach den statischen Regeln ab.	Readiness-Probe (Startverzögerung 5s, Intervall 10s)

Was ein Datenbankausfall bewirkt

- Replicas bleiben bereit und antworten weiter. Eine Admission-Prüfung gibt ihrer Datenbankarbeit insgesamt zwei Sekunden. Danach entscheiden sofort die statischen Regeln: `:latest`, die Registry-Allowlist und ein verlangter Digest lehnen weiter ab. Aufzeichnung und cache-abhängige Urteile fallen zurück, und der Ausfall wird einmal protokolliert.
- Die Datenbank wird in `attestkeep_database_up` und auf der Health-Seite gemeldet. Wenn Sie vor 1.4.0 die Bereitschaft des Operators beobachtet haben, um einen Datenbankausfall zu erkennen, beobachten Sie stattdessen diese Metrik.
- Danach findet der Operator die Lücke im Heartbeat und hält sie als Ausfall fest, und der nächste Abgleichlauf listet jeden Digest, der ohne Entscheidung hereinkam.
- Benachrichtigungen gehen außerhalb der Anfrage hinaus, höchstens 64 warten. Der Rest wird verworfen und in `attestkeep_notifications_dropped_total` gezählt.
- Die Mengenzählung lässt bei einem Datenbankfehler zu, statt Deployments zu blockieren.
- Scans stoppen mit der Datenbank und laufen mit ihr weiter. Die Warteschlange liegt in der Datenbank, nichts geht verloren.

Was Sie alarmieren sollte

Metriken liegen an Port 9090 unter `/metrics`. Jede gelabelte Reihe existiert ab dem Start mit Null, fehlende Daten und null Ablehnungen sind also verschiedene Dinge.

Zustand	Signal	Warum es zählt
Die Admission wird umgangen	`attestkeep_webhook_distrusted == 1`	Solange der Wert unter `failurePolicy: Ignore` 1 ist, gehen Deployments ungeprüft durch. Behandeln Sie es als Alarm.
Etwas ist hereingekommen	`attestkeep_runtime_unmatched_digests > 0`	Der Abgleich hat laufende Digests ohne zugelassene Admission dahinter gefunden.
Probleme mit der Datenbank	`attestkeep_database_up == 0`	Ab 1.4.0 bleiben die Replicas während eines Ausfalls bereit, Readiness-Alarme zeigen ihn also nicht mehr.
Scans fallen zurück	`attestkeep_scan_queue_depth` bleibt hoch	Eine Warteschlange, die nur wächst, heißt, dass der Scanner eine Registry oder seine Datenbank nicht erreicht.
Schwachstellendaten veralten	`attestkeep_vulnerability_db_age_seconds`	Urteile sind nur so aktuell wie die Datenbank dahinter.
Image-Kontingent fast erreicht	`attestkeep_image_quota{kind="count"}` gegen `{kind="limit"}`	Am Limit werden nur Image-Namen abgelehnt, die in diesem Zeitraum noch nicht vorkamen. Laufende Workloads laufen weiter.
Ein SIEM-Ziel fällt zurück	`attestkeep_siem_lag_seconds{target}`, `attestkeep_siem_events_total{outcome="gave_up"}`	Die Verzögerung ist das Alter des ältesten Events, das das Ziel noch nicht erhalten hat. Nach 24 Stunden nicht zugestellte Events werden aufgegeben.
Über der Node-Obergrenze	`attestkeep_nodes{kind="count"}` gegen `{kind="ceiling"}`, `attestkeep_node_overage_seconds > 0`	Der Überschreitungswert zeigt, wie lange der Cluster schon über der Obergrenze liegt. Alarmieren Sie lange vor 14 Tagen, ab denen neue Image-Namen abgelehnt werden.

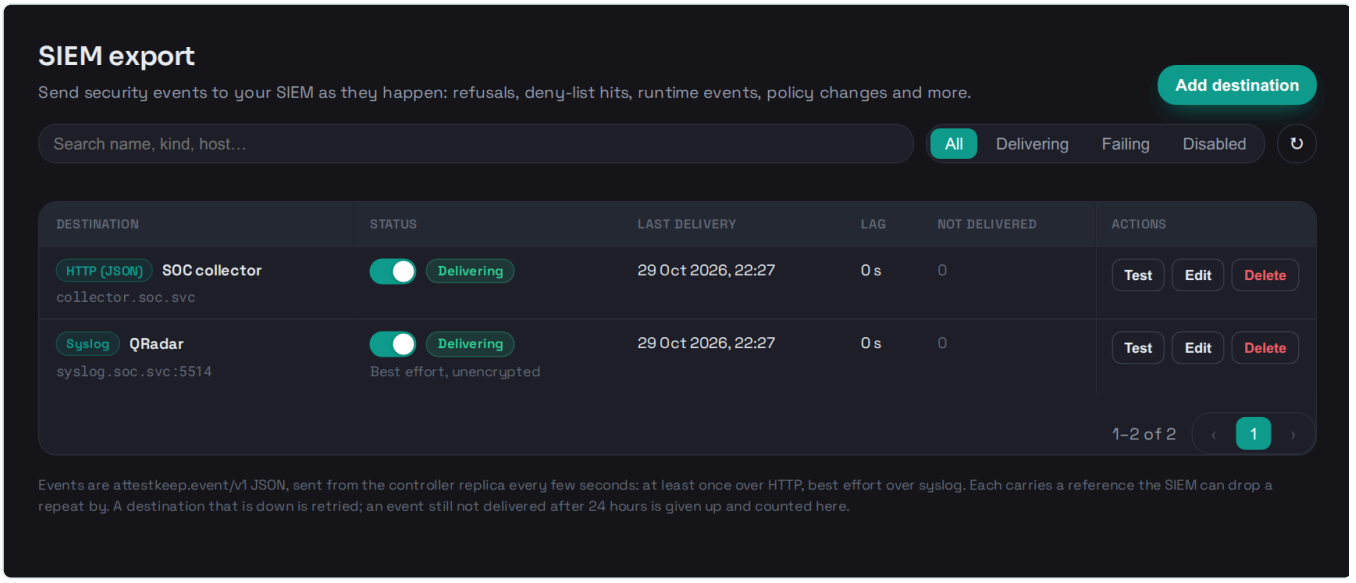
Ablehnungen werden in `attestkeep_admission_denials_total{reason}` gezählt. Gründe sind unter anderem `threshold`, `signature`, `attestation`, `tag`, `digest`, `registry`, `quota`, `unscanned`, `scan_error`, `end_of_life`, `known_exploited`, `provenance` und `context`. Ablehnungen wegen der Node-Obergrenze zählen unter `quota`. Weitere nützliche Reihen für ein Dashboard: `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` und die Runtime-Reihen aus Kapitel 8.

Benachrichtigungs-Events decken dasselbe ab: `threshold_80` und `threshold_100` für das Image-Kontingent, `nodes_80`, `nodes_over`, `nodes_enforced` und `nodes_cleared` für die Node-Obergrenze, `license_expiring`, wenn ein Online-Zertifikat weniger als eine Woche gilt, `trivy_db_failed`, `unverified_workload`, `runtime_alert` und ab 1.4.0 `deny_list_hit`, `deny_list_entry_added` und `deny_list_entry_removed`. Ein neuer Kanal hat `nodes_over`, `nodes_enforced`, Runtime alert und Deny list hit angehakt.

SIEM-Export

Ihr Sicherheitsteam schaut ins SIEM, nicht in diese Konsole. Ab 1.4.0 sendet Settings > SIEM export die Sicherheitsereignisse von Attestkeep dorthin, sobald sie entstehen, sodass sie auf derselben Zeitachse liegen wie alles andere, was Sie sammeln. Nichts verlässt den Cluster, solange Sie kein Ziel anlegen, und ein Ziel anzulegen ändert keine Admission-Entscheidung. Alle,

die sich anmelden, sehen die Ziele und ihren Zustand. Ziele anlegen, ändern, testen und entfernen ist Sache eines Administrators.



Settings > SIEM export mit zwei Zielen. *SOC collector* empfängt HTTP JSON, *QRadar* Syslog im LEEF-Format über einfaches TCP. Beide zeigen *Delivering*, ohne Verzug und ohne aufgegebene Events. Die Syslog-Zeile sagt, was einfaches TCP bedeutet: best effort, unverschlüsselt.

Was gesendet wird. Abgelehnte Admissions, Treffer der Deny List, hinzugefügte und entfernte Einträge, Take Downs und Wiederherstellungen, Runtime-Events und von einer Runtime-Policy gestoppte Pods, Änderungen an Policy, Break-Glass und Triage, Erzeugung und Download von Nachweisen, fehlgeschlagene Anmeldungen und Rollenwechsel. Ein kritischer Befund wird einmal gesendet, von dem Scan, der ihn zuerst meldete, ein ausgenutzter einmal von dem Scan, der ihn zuerst auf der KEV-Liste sah. Zugelassene Admissions sind aus, solange Sie sie nicht anhaken, denn sie machen den Großteil des Ledgers aus. Jedes Ziel filtert nach Ereignistyp, Namespace und niedrigstem Schweregrad, und ein neues Ziel kann mit den letzten 24 Stunden beginnen.

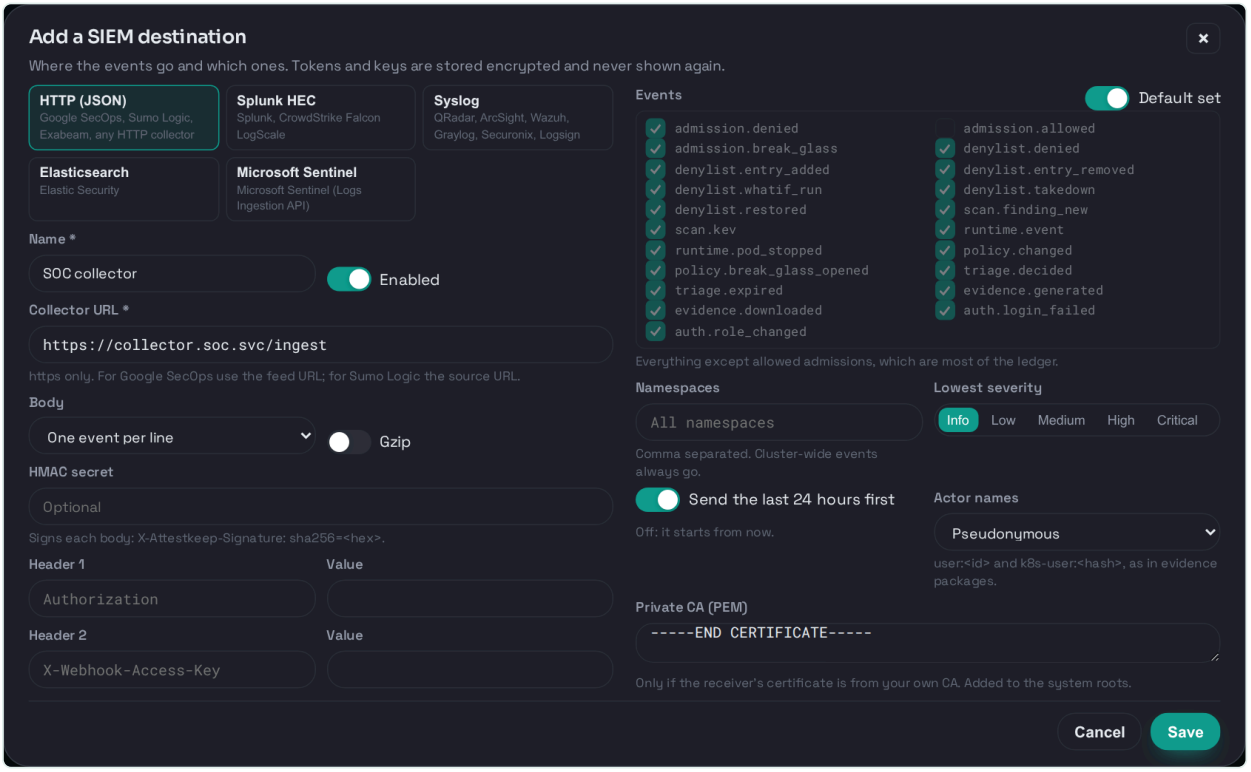
Das Ereignis. Jedes Ereignis ist ein JSON-Objekt im Schema `attestkeep.event/v1` mit Zeit, Typ, Schweregrad, Cluster, Namespace, Pod, Workload, Image, Digest, Policy, Akteur, Ergebnis, Begründung und einer `reference` auf den Datensatz, aus dem es stammt. Die Referenz bleibt gleich, wenn ein Ereignis erneut gesendet wird, Ihr SIEM kann Wiederholungen also danach verwerfen. Ein Ereignis enthält nie ein SBOM, eine Befundliste, Zugangsdaten, Lizenzdaten oder das, was bei einer fehlgeschlagenen Anmeldung eingetippt wurde. Ein Ziel lässt sich so einstellen, dass es Personen wie das Nachweispaket als `user:<Konto-ID>` benennt.

Ziel	Für	Wie
HTTP (JSON)	Google SecOps, Sumo Logic, Exabeam, jeder HTTP-Collector	Ein HTTPS-POST pro Stapel, ein JSON-Array oder ein Ereignis pro Zeile. Optional gzip, zwei Request-Header und eine HMAC-Signatur in `X-Attestkeep-Signature`.
Splunk HEC	Splunk, CrowdStrike Falcon LogScale	HEC-Ereignisse mit source `attestkeep` und sourcetype `attestkeep:event`.
Syslog	QRadar, ArcSight, Wazuh, Graylog, Securonix, Logsign	RFC 5424 über TLS oder einfaches TCP für einen Empfänger ohne TLS, mit CEF-, LEEF- oder JSON-Inhalt.
Elasticsearch	Elastic Security	Die Bulk-API in einen Index oder Data Stream, mit API-Key.
Microsoft Sentinel	Microsoft Sentinel	Die Logs Ingestion API über eine Data Collection Rule, mit einer Entra-Anwendung.

Zustellung. Ereignisse gehen über HTTP mindestens einmal hinaus und über Syslog, das keine Quittung kennt, nach bestem Bemühen. Sie gehen in Commit-Reihenfolge, alle paar Sekunden, höchstens 100 pro Anfrage, von der Replica, die die periodische Arbeit erledigt. Ein fehlschlagendes Ziel wird mit wachsenden Pausen oder nach seinem `Retry-After` erneut versucht und hält nie ein anderes auf. Ein Ziel, das hinter einer offenen Datenbanktransaktion wartet, zeigt *Waiting*. Ein

Ereignis, das 24 Stunden nach seinem Entstehen noch nicht zugestellt ist, wird aufgegeben und als "N events not delivered" angezeigt, und der Abschnitt External export des Nachweispakets listet es. *Test* sendet ein Ereignis über das Ziel und zeigt die Antwort.

Zugangsdaten. Tokens, Schlüssel und Secrets werden mit dem Data Key der Installation verschlüsselt und nie wieder angezeigt. Ändert eine Bearbeitung, wohin ein Ziel zeigt, werden die gespeicherten Zugangsdaten entfernt, damit sie nie an die neue Adresse gehen, außer Sie geben sie dort ein. Loopback- und Link-Local-Adressen, darunter der Metadatendienst der Cloud, werden abgelehnt. HTTP-Ziele müssen `https://` verwenden, und ein Empfänger mit einem Zertifikat Ihrer eigenen CA bekommt diese CA im Formular.



Add a SIEM destination mit HTTP (JSON). Die Collector-URL muss https sein. Der Body enthält ein Event pro Zeile, das Standard-Set umfasst alles außer zugelassenen Admissions, und *Send the last 24 hours first* schickt zuerst den Tag vor dem Anlegen des Ziels und dann die neuen Events. Akteursnamen sind pseudonym, wie in Nachweispaketen. Hat der Empfänger ein Zertifikat Ihrer eigenen CA, tragen Sie diese CA hier ein, sie kommt zu den Systemwurzeln hinzu. Tokens und Schlüssel werden verschlüsselt gespeichert und nie wieder angezeigt.

Der SIEM-Export braucht PostgreSQL 13 oder neuer, weil er in Commit-Reihenfolge liest. Das mitgelieferte PostgreSQL ist 16. Auf einem älteren externen Server lässt sich kein Ziel einschalten, und die Seite zeigt *Database too old*. Auch ohne Ziel können die eigenen Logs, Traces und Metriken des Operators mit `observability.otlp.endpoint` an einen OpenTelemetry-Collector gehen. docs.attestkeep.com/siem beschreibt die Einrichtung für jedes SIEM-Produkt.

Was zu sichern ist

Attestkeep macht keine Datenbank-Backups. Den Dump auszuführen, zu planen und aufzubewahren ist Ihre Aufgabe. Alles, was der Operator weiß, liegt in PostgreSQL und in Cluster-Ressourcen. Das Backup, auf das es ankommt, ist die Datenbank: Befunde, Entscheidungen, Ausnahmen, das Audit-Log, das Nachweisarchiv, Benutzer und das Lizenzzertifikat liegen alle darin.

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

Verwenden Sie Ihr eigenes PostgreSQL, decken Ihre bestehenden Backups es schon ab. Ein Volume-Snapshot des PVC der mitgelieferten Datenbank funktioniert ebenfalls, wenn Ihre Storage Class das unterstützt. Neben der Datenbank sichern Sie drei Secrets:

- `attestkeep-evidence-key`, den Schlüssel zum Signieren der Nachweise, vom Operator angelegt. Geht er verloren, wird ein neuer erzeugt, und mit dem alten signierte Checkpoints und Dokumente lassen sich nicht mehr gegen den aktuellen

Schlüssel der Installation prüfen. Die Historie bleibt. Ab 1.4.0 ist er auch die Lizenzidentität: Geht er verloren, wird die nächste tägliche Prüfung mit `401` beantwortet, und der Cluster läuft als Community, bis er neu aktiviert wird.

- `attestkeep-data-key`, vom Chart erzeugt, sofern Sie nicht `api.dataKey.existingSecret` geliefert haben. Geht er verloren, lassen sich die gespeicherten Tracker-Tokens nicht mehr entschlüsseln, und jeder Tracker-Kanal braucht sein Token neu.
- `attestkeep-session`, den Sitzungsschlüssel. Geht er verloren, melden sich alle neu an.

Das Serving-Zertifikat des Webhooks steht nicht auf der Liste, weil der Operator es neu ausstellt. Der Aktivierungsschlüssel auch nicht: Eine Neuinstallation im selben Cluster ergibt denselben Fingerabdruck und nutzt den vorhandenen Lizenz-Slot. Mit aktiver Falco-Integration legt der Operator außerdem `<release>-falco-mtls` an. Sein Client-Zertifikat wird unter Falco > Falco integration erneuert.

Settings > Backup exportiert außerdem ein JSON-Bündel der Konfiguration, die Sie von Hand aufgebaut haben: Benachrichtigungskanäle, Triage-Entscheidungen, die Benutzerliste, Policies und SIEM-Ziele. Es dient dazu, einen zweiten Cluster genauso einzurichten. Zugangsdaten sind nie im Bündel, und importierte SIEM-Ziele kommen deaktiviert an. Der Import zeigt zuerst eine Vorschau, vorhandene Datensätze gewinnen immer, und Benutzer werden aufgeführt, nicht angelegt. Ab 1.3.1 steht jeder Export und Import unter *Backup history*. Das Bündel ersetzt den Datenbank-Dump nicht.

Eine Wiederherstellung verschweigt nicht, was verloren ist. Beim ersten Durchlauf nach einer Wiederherstellung findet der Operator die Heartbeat-Lücke seit dem Dump und hält die ganze Spanne als Zeitraum fest, in dem die Kontrolle nicht arbeitete. Ein Nachweispaket für diesen Zeitraum zeigt die Lücke wie jeden anderen Ausfall. Einen Dump in dieselbe oder eine neuere Operator-Version zurückzuspielen wird unterstützt, einen neueren Dump in einen älteren Operator nicht.

Verfügbarkeit

Operator und Webhook: 2 Replicas, bevorzugte Anti-Affinity, PodDisruptionBudget minAvailable 1. Ab 1.4.0 leert sich eine stoppende Replica: Ihr preStop-Hook wartet `webhook.preStopSleepSeconds` (standardmäßig 5), während der Cluster aufhört, sie anzusteuern, dann hält sie ihren Listener fünf Sekunden länger offen und beendet die laufenden Prüfungen. Die Termination Grace Period des Pods ist dieser Wert plus 30 Sekunden. Erhöhen Sie `replicaCount`, um zu skalieren. Controller-Arbeit läuft über einen Advisory Lock seriell, das ist also sicher. Scanner: 1 Replica. Das Chart lehnt mehrere Replicas auf einem ReadWriteOnce-Cache-Volume ab und sagt, warum. Mitgeliefertes PostgreSQL: bewusst eine Replica. Für Replikation und verwaltete Backups richten Sie `externalDsn` auf Ihr eigenes PostgreSQL.

GitOps. Die Durchsetzung geschieht bei der Pod-Admission, es ist also gleich, ob kubectl, Argo CD, Flux oder ein Job die Änderung anwendet. Legt ein GitOps-Sync einen Pod an, dessen Image die Policy nicht erfüllt, scheitert der Sync mit der Ablehnung als Fehlermeldung.

Air-Gapped-Cluster. Trivy und cosign sind im Image enthalten. Spiegeln Sie die Schwachstellendatenbank mit `oras copy ghcr.io/aquasecurity/trivy-db:2 registry.internal/mirror/trivy-db:2` und setzen Sie `trivy.dbRepository`. Setzen Sie `scan.cosignOffline`, um gegen Schlüssel ohne das öffentliche Transparenzlog zu prüfen. Ein gespiegelter KEV-Katalog kann gzip-komprimiert in eine ConfigMap gelegt werden, die `kev.existingConfigMap` nennt, mit `kev.source` auf `file:///etc/kev/<key>`. Die Lizenzierung braucht entweder eine Route zu lic.attestkeep.com:443 oder die Offline-Aktivierung (Kapitel 12).

12. Lizenzierung

Ein Aktivierungsschlüssel pro Kauf und ein Community-Schlüssel pro Konto. Ein Schlüssel bindet sich an so viele Cluster, wie sein Plan erlaubt, und einen Slot können Sie selbst verschieben.

Aktivierung

Die Aktivierung geschieht einmal pro Cluster und belegt einen Slot. Beim ersten Start erzeugt der Operator seinen Signaturschlüssel im Cluster, hält den privaten Teil im Secret `attestkeep-evidence-key`, das alle Replicas teilen, und sendet mit der Aktivierungsanfrage nur den öffentlichen Teil. Ab 1.4.0 ist es derselbe Schlüssel, der Nachweispakete signiert, sodass das Zertifikat den Schlüssel nennt, gegen den ein Auditor sie prüft. Eine vor 1.4.0 aktivierte Online-Installation signiert weiter mit dem Schlüsselpaar, mit dem sie aktiviert wurde, und wechselt bei ihrer nächsten Aktivierung zum einen Schlüssel. Die Anfrage enthält den Aktivierungsschlüssel, einen Fingerabdruck des Clusters (einen Hash der UID des Namespace kube-system, weder Name noch Adresse), den Clusternamen, sofern gesetzt, und die Operator-Version. Die Antwort ist ein signiertes Zertifikat, höchstens dreißig Tage gültig. Der Operator prüft es im Cluster gegen einen in das Binary eingebauten Public Key.

Ab 1.3.0 akzeptiert der Operator nur ein Zertifikat, das die Node-Obergrenze trägt. Ein Zertifikat eines älteren Lizenzdienstes wird mit "re-activate with a key issued by licence service 1.3.0 or later" abgelehnt.

Erreicht der Operator den Lizenzserver nicht, sagt er das, statt den Schlüssel für ungültig zu erklären. Ein Zertifikat, dem der Operator nicht vertraut, eine abgelehnte oder abgelaufene Verbindung und ein nicht auflösbarer Name werden mit `502`, dem Server und der Ursache beantwortet. Die Konsole zeigt "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." Eine `5xx`-Antwort des Lizenzservers wird zu `503` mit "The licence server answered with an error. Try again later." Der Schlüssel steht nie in einer Meldung. `license.caBundle` nimmt PEM-Zertifikate auf, denen der Lizenz-Client neben den System-Roots vertraut, für einen TLS-aufbrechenden Proxy oder einen Lizenzserver unter einer privaten CA. Setzen Sie es mit `--set-file license.caBundle=ca.pem`. Die Zertifikatsprüfung bleibt eingeschaltet.

Die tägliche Prüfung

Einmal am Tag sendet die Installation drei Werte: Lizenz-ID, Fingerabdruck des Clusters und Operator-Version, signiert mit dem Schlüssel aus der Aktivierung. Ab 1.4.0 führt eine Operator-Replica die Prüfung aus, und der Scanner kontaktiert den Lizenzserver nicht mehr. Die Antwort ist ein Status, dazu ein neues Zertifikat, wenn sich etwas geändert hat oder das aktuelle bald abläuft. Nichts über Ihre Images, Workloads, Policies oder Befunde wird gesendet. Es gibt keinen Telemetriekanal und keine Einstellung, die einen hinzufügt. Wie viele Operator-Pods auch laufen, ein Cluster macht diesen Aufruf einmal am Tag.

Wenn der Lizenzdienst nicht erreichbar ist

Die Installation läuft mit ihrem Zertifikat weiter, bis es abläuft. Eine gescheiterte Prüfung wird nach vier Stunden wiederholt. Hat ein Online-Zertifikat weniger als eine Woche übrig, warnt die Konsole, und die Benachrichtigung `license_expiring` geht hinaus. Läuft das Zertifikat ab, arbeitet die Installation als Community Edition weiter. Sie stoppt nicht und beginnt nicht, Deployments abzulehnen. Eine Antwort `revoked` oder `released` gilt bei der nächsten Prüfung.

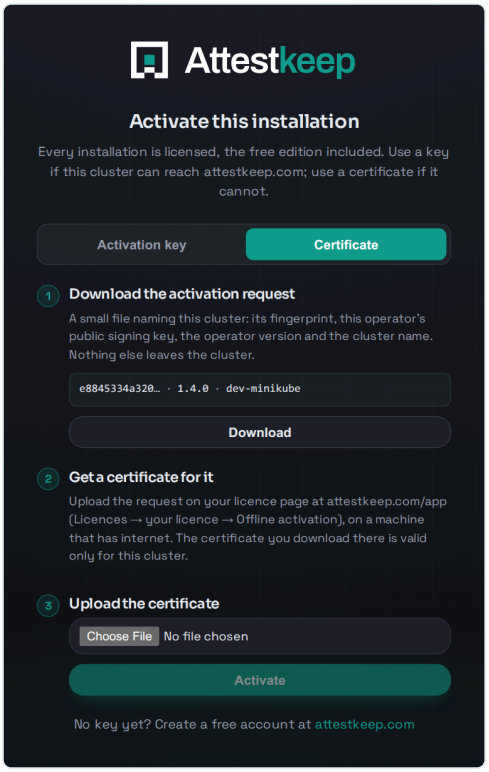
Cluster-Slots

Jede Aktivierung hält einen Slot, bis er freigegeben wird. Slots geben Sie selbst auf Ihrer Lizenzseite auf attestkeep.com frei. Ein freigegebener Slot ist sofort frei, und die Installation, die ihn hielt, fällt bei ihrer nächsten täglichen Prüfung heraus. Eine Neuinstallation im selben Cluster belegt keinen zweiten Slot, denn der Fingerabdruck ist derselbe. Sind alle Slots belegt, nennt die Fehlermeldung die Cluster, die sie halten, und wann jeder aktiviert wurde.

Auch die Konsole kann eine Lizenz zurückgeben: *Deactivate* auf der Seite Usage and License, nach einer Bestätigung. Auf dem Online-Weg wird so der Slot des Clusters frei. Auf dem Offline-Weg wird das Zertifikat aus dem Cluster gelöscht, und den Slot gibt der Attestkeep-Support frei, denn ein Offline-Cluster meldet sich nie zurück.

Offline-Aktivierung

Ein Cluster, der den Lizenzdienst nie erreichen wird, wird mit einem Zertifikat statt mit einem Schlüssel aktiviert. Die Aktivierungsseite und die Seite Usage and License haben dafür beide einen Reiter *Certificate*.



Der Reiter *Certificate* in drei Schritten. Laden Sie die Aktivierungsanfrage herunter, die diesen Cluster mit Fingerprint, öffentlichem Signaturschlüssel des Operators, Operator-Version und Cluster-Namen benennt. Laden Sie sie von einem Rechner mit Internetzugang auf Ihrer Lizenzseite bei attestkeep.com hoch. Laden Sie das Zertifikat hoch, das Sie zurückbekommen.

Eine vor 1.4.0 offline aktivierte Installation behält ihre Lizenz, aber ihr Zertifikat nennt das alte Schlüsselpaar, ein signierter Nachweis-Download wird daher abgelehnt, bis eine neue Aktivierungsanfrage aus Usage and License gegen ein neues Zertifikat getauscht ist. Die Seite Evidence und die Ablehnung sagen das.

Das Zertifikat ist dasselbe signierte Dokument, das der Online-Weg speichert, als offline gekennzeichnet. Der Operator prüft Signatur, Fingerabdruck des Clusters und seinen eigenen Schlüssel, bevor er es speichert. Für dieses Zertifikat gibt es keine tägliche Prüfung, und er warnt dreißig Tage vor dem Ablauf. Ein Zertifikat für einen anderen Cluster oder Schlüssel oder ein abgelaufenes wird mit Begründung abgelehnt. Für eine Installation, deren Konsole niemand öffnet, nennt `license.existingCertificateSecret` ein Secret, das das Zertifikat unter dem Schlüssel `certificate` enthält.

Ob die Offline-Aktivierung für Ihre Lizenz verfügbar ist, zeigt die Lizenzseite auf attestkeep.com. Laut veröffentlichter Preisliste gehört sie zu Enterprise mit jährlicher Laufzeit oder zu einer Vereinbarung.

Die Node-Obergrenze

Jeder Plan erlaubt eine Zahl von Ready-Nodes pro Cluster. Der Operator zählt die Ready-Nodes des Clusters alle `nodes.pollInterval` (standardmäßig 5 Minuten). Die Zahl bleibt im Cluster. Die tägliche Prüfung sendet weiter dieselben drei Werte. Die Obergrenze selbst steht im Zertifikat.

Die Obergrenze zu überschreiten stoppt nichts. Konsole und eine Benachrichtigung sagen es, mit einem Datum 14 Tage in der Zukunft (`nodes.graceDays`). Liegt der Cluster nach diesen 14 Tagen noch darüber, werden in diesem Zeitraum noch nicht gesehene Image-Namen bei der Admission mit "node ceiling of N exceeded" abgelehnt, und ein abgelehnter Name wird nicht gezählt. Dasselbe gilt ab 1.4.0 am Image-Kontingent: Ein Name, der abgelehnt wird, weil das Monatskontingent erreicht ist, wird nicht gezählt, und der Zähler bleibt mit *Allowance reached* am Limit stehen. Auf der Lizenzkarte der Seite Usage and License zeigt ein volles Image-Band *Allowance reached* mit den eigenen Daten der Lizenz statt *Expired*, und die Mail zum Kontingent sagt, dass neue Image-Namen bei der Admission abgelehnt werden, nicht, dass sie nicht mehr gescannt werden. Alles bereits Gezählte läuft weiter und wird weiter gescannt und durchgesetzt. Zurück unter die Obergrenze, mit weniger

Nodes oder mehr Nodes auf der Lizenz, stellt die Uhr sofort zurück. Eine Zählung, die die Nodes nicht erfassen kann, ändert nichts. Keiner der beiden Chart-Werte lockert die Obergrenze.

Wenn eine Lizenz endet

Die Installation arbeitet als Community Edition weiter, wenn das Abonnement endet und das Zertifikat abläuft, wenn das Zertifikat sich nicht mehr prüfen lässt oder sich als das eines anderen Clusters erweist, oder wenn Sie es deaktivieren. Community heißt 1 Cluster, 1 Framework, 10 Image-Namen im Monat und 1 Node pro Cluster. Nichts wird heruntergeladen, und kein neues Zertifikat wird ausgestellt. Scans, Admission und Nachweise arbeiten weiter. Ab diesem Moment zeigen die Seite Usage and License und das Dashboard das Community-Kontingent für den laufenden Monat, dieselben Zahlen, die die Admission anwendet. Frühere Monate behalten ihr damaliges Kontingent. Eine Kündigung mitten in der Laufzeit schaltet am Tag der Kündigung nichts ab. Sie behalten, wofür Sie bezahlt haben, bis die Laufzeit endet.

Frameworks wählen. Wurden Nachweise für mehr Frameworks erzeugt, als die Lizenz jetzt abdeckt, wird der nächste Owner oder Administrator, der sich anmeldet, gefragt, welche bleiben sollen, bis zur Zahl der Lizenz. Bis jemand wählt, ist keines der bereits genutzten Frameworks für ein neues Paket offen, und anderen Benutzern wird gesagt, dass ein Administrator wählen muss. Die nicht behaltenen Frameworks sind für neue Pakete geschlossen. Jedes bereits erzeugte Paket bleibt herunterladbar. Die Wahl gilt, bis sich Plan oder Zahl der Frameworks der Lizenz ändert, und sie wird ins Audit-Log geschrieben.

13. Editionen und Preise

Jede Sicherheitsfunktion steckt in der kostenlosen Edition. Scans, Signaturprüfung, Durchsetzung bei der Admission und Nachweispakete sind in Community vollständig, und jede Kontrolle der Policy, auch Deny until scanned und `failurePolicy: Fail`, funktioniert dort. Ebenso die Deny List, Take Down, What if und der SIEM-Export. Die bezahlten Pläne bringen Kapazität und Single Sign-on.

Edition	Pro Monat, zzgl. USt.	Pro Jahr, zzgl. USt.	Cluster	Frameworks	Image-Namen / Monat, pro Cluster	Nodes / Cluster	Außerdem
Community	0 USD	0 USD	1	1, nach Wahl	10	1	Öffentlicher Issue-Tracker
Team	600 USD	6.000 USD	1	1	30	6	Single Sign-on, Support per E-Mail
Business	1.800 USD	18.000 USD	3	3	50	10	Alles aus Team, vorrangiger Support per E-Mail
Enterprise	3.600 USD	36.000 USD	5	Alle dreizehn	75	20	Offline-Aktivierung für Air-Gapped-Cluster, Support per E-Mail

Ein Jahr kostet zehn Monate. Zusatzoptionen der bezahlten Editionen, pro Monat: zusätzlicher Cluster +300 USD (bis zu zehn Cluster in einem Checkout), zusätzliches Framework +120 USD, 10 weitere Image-Namen im Monat +30 USD für jeden von der Lizenz abgedeckten Cluster, zusätzlicher Node +50 USD für jeden von der Lizenz abgedeckten Cluster, einzeln, bis 50 Nodes pro Cluster. Jahrespreise sind das Zehnfache der Monatspreise. In Community kostet eine Mengenstufe +300 USD im Monat für jeden Cluster. Zusätzliche Nodes gibt es in Community nicht.

Mehr als zehn Cluster, keine Cluster-Obergrenze oder keine Node-Obergrenze sind eine individuelle Vereinbarung, kein Listenpreis.

Preise verstehen sich ohne Umsatzsteuer. Checkout und Zahlung laufen über Polar als Merchant of Record. Die Karte wird im Checkout von Polar eingegeben und erreicht nie die Server von Attestkeep. Fällige Steuern werden dort anhand Ihrer Rechnungsadresse berechnet und vor der Zahlung angezeigt. Ein umsatzsteuerlich registriertes Unternehmen in der EU führt die Umsatzsteuer im Reverse-Charge-Verfahren selbst ab. Lizenz und Support kommen von Attestkeep.

Es gibt keine befristete Testphase. Community Edition hat jede Sicherheitsfunktion und läuft nicht ab. Sie können das Produkt also so lange in Ihrem eigenen Cluster testen, wie Sie brauchen.

Gezählt wird der Image-Name: Registry und Repository, ohne Tag und Digest. Alles, was im Cluster läuft, zählt mit, auch Images Dritter. Ist das Kontingent aufgebraucht, werden bei der Admission nur neue Image-Namen abgelehnt. Jedes in diesem Zeitraum schon gezählte Image wird weiter gescannt und durchgesetzt.

14. Sicherheit, Support und Ihr Vertragspartner

Integrität der Releases

Jedes Release-Image wird für linux/amd64 und linux/arm64 gebaut und mit cosign auf seinem Digest signiert, keyless über GitHubs OIDC und mit dem unter docs.attestkeep.com/cosign.pub veröffentlichten Schlüssel. Scanner und Signaturprüfer stecken im Image und werden nicht erst im Betrieb nachgeladen. Der Quellcode ist nicht öffentlich.

Einen Fehler oder eine Schwachstelle melden

Fehler gehen an den öffentlichen Tracker unter github.com/attestkeep/attestkeep-docs, für jede Edition. Schwachstellen gehen über das private Advisory-Formular desselben Repositorys oder an security@attestkeep.com. Ein Mensch bestätigt eine Meldung innerhalb von drei Arbeitstagen, und behobene Probleme werden als Advisories veröffentlicht, sobald ein Release die Korrektur enthält. Die Verordnung (EU) 2024/2847 verlangt von einem Hersteller eine zentrale Anlaufstelle für Schwachstellenmeldungen. Unsere ist security@attestkeep.com, und dieselben Angaben stehen auf jedem Nachweispaket.

Support je Plan

Plan	Support
Community	Öffentlicher Issue-Tracker
Team	E-Mail
Business	E-Mail, vorrangig
Enterprise	E-Mail
Individuelle Vereinbarung	Wie in der Vereinbarung festgelegt

Attestkeep veröffentlicht für keinen Plan eine garantierte Reaktionszeit.

Mit wem Sie es zu tun haben

Handelsname	Attestkeep
Unternehmensname	Ecavist Technologies, ein in der Türkei eingetragenes Unternehmen
Eingetragene Anschrift	Dörtyol Küme Evleri No: 369, İç Kapı No: 3, Işıklar, Merkez, Çanakkale, Türkiye
Rechtliche Mitteilungen, Verträge, Rechnungen	legal@attestkeep.com
Support und Vorabfragen	support@attestkeep.com
Sicherheitsmeldungen	security@attestkeep.com

Die verbindlichen Dokumente (Nutzungsbedingungen, Fernabsatzvertrag, Datenschutzhinweis, Bedingungen zur Auftragsverarbeitung, Unterauftragsverarbeiter) stehen auf attestkeep.com. Weichen eine Seite in einfacher Sprache und eine Vereinbarung voneinander ab, gilt die Vereinbarung.

Anhang. Wogegen dieses Handbuch geprüft wurde

Produktversion	Attestkeep 1.4.0 (Chart 1.4.0, appVersion 1.4.0). Release Notes vom 28. Oktober 2026, Image und Chart veröffentlicht am 29. Oktober 2026
Veröffentlichtes Image	`ghcr.io/attestkeep/attestkeep-k8s:1.4.0` `sha256:bfe0299695e38779f02ac24d805088a373d0a754fd7f0f552751e0d9fa3f05e8`
Veröffentlichtes Chart	`oci://ghcr.io/attestkeep/charts/attestkeep` Version 1.4.0 `sha256:8d6c4ba1e3ff790f6394e83eb34e5a740eeeecb0b4148e5c7e48683ca9bc610f`
Fotografierte Installation	Eine Installation, deren Konsole Version 1.4.0 meldet, auf Kubernetes v1.35.1 mit zwei Nodes, als Enterprise Edition, mit Falco 0.45.0, das über sein eigenes `http_output` per Mutual TLS angebunden ist. Sie läuft mit einem aus dem 1.4.0-Quellcode gebauten Image, nicht mit dem oben genannten veröffentlichten Image. Fotografiert am 29. Oktober 2026 im dunklen Design mit englischer Konsole.
Gezeigtes Nachweispaket	`PS-SOC2-DEVMINIKUBE-20261001-20261029-DA7379`, erzeugt am 29. Oktober 2026 um 19:31 UTC von Version 1.4.0
Dokumentation	Seiten von docs.attestkeep.com für 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 und operations
Preise	attestkeep.com/pricing, USD, ohne Umsatzsteuer. Die Zahlen sind seit der Ausgabe 1.3.2 unverändert und wurden am 29. Oktober 2026 mit der Preisliste der Website abgeglichen.

Dokumentation: docs.attestkeep.com · Release Notes: docs.attestkeep.com/releases · Prüfwerkzeug für gelieferte Nachweispakete: attestkeep.com/verify.html

Was sich seit 1.3.2 geändert hat

Version	Was sich geändert hat
1.4.0, 28. Oktober 2026	Die Deny List, geprüft vor Break-Glass und jeder Regel der Policy, mit einem Modus `denyList` in jeder Policy, What if für einen Eintrag, Treffern je Workload, betroffenen Workloads und Take Down von Hand mit Wiederherstellung (Kapitel 7). Die Prüfung `attestkeep-binscan` liest die ausführbaren Dateien in Images, innerhalb fester Grenzen. SIEM-Export an fünf Arten von Zielen (Kapitel 11). Signaturprüfung per Schlüssel mit `signatureKeys`. `license.caBundle`, und Fehler des Lizenzservers werden als solche gemeldet. Ein Runtime-Stopp braucht `rbac.allowEnforce`, ganz gleich, was sonst gewährt ist. Replicas bleiben bei einem Datenbankausfall bereit und leeren sich beim Stoppen. Dry Runs hinterlassen nichts. Ein wegen des Kontingents abgelehnter Image-Name zählt nicht, und ein volles Band zeigt Allowance reached. Abschnitte 13 und 14 im Nachweis, Befunde einmal je Digest. Ein Schlüssel für Lizenz und Nachweis. Go 1.26.10, Trivy 0.76.1, cosign 3.1.4. Migrationen 0036 bis 0046.
1.3.7, 15. Oktober 2026	Mit `kubectl debug` hinzugefügte Container werden bei der Admission geprüft (Kapitel 3). Die Seite Images setzt jedes Image in einen von vier Zuständen, mit der neuen Kachel Not compliant, und Blocked folgt der letzten Admission-Prüfung. What-if-Läufe bleiben neunzig Tage unter Security > What-if history, mit PDF- und JSON-Downloads, die eine Projektion sind, kein Nachweis. What if in jeder Policy-Zeile, Scan now für die Images, die eine Wiedergabe ohne Scan fand, eine Rückfrage, bevor Cancel Änderungen verwirft, Apply to cluster nur für Konten, die Policies schreiben dürfen, und eine einzige Liste der Framework-Namen. Der Operator wird mit Go 1.26.9 und `golang.org/x/net` 0.60.0 gebaut, die veröffentlichte Schwachstellen schließen, die der Operator erreicht. Migrationen 0033 bis 0035.
1.3.6, 8. Oktober 2026	Die Seite Images zählt nur zugelassene Pods. Der Aktionsfilter in Falco Events bietet die fünf Wirkungen. Layoutkorrekturen für Seitenleiste und Nachweisliste bei 1366 und 1600 px und für die Nutzungsanzeige. Das Formular New policy schlägt die dokumentierten SLA- und Triage-Standards vor.
1.3.5, 8. Oktober 2026	Eine Falco-Policy auf Stop the pod ist überall so beschriftet, und ein Pod-Stopp steht im Nachweispaket als `policy:<Name>`. Der Cluster wird über `clusterName` des Charts benannt. Das Audit Log verwendet die Ortszeit. Die Kachel Vulnerability database zeigt, wann zuletzt geprüft wurde, und hat Update now. Die Webhook-Kachel vergleicht die Latenz nicht mehr mit einem festen Budget.
1.3.4, 8. Oktober 2026	Korrekturen am Nachweis: Angaben zu Break-Glass kommen aus der damals geltenden Policy-Revision, Triage-Entscheidungen tragen den Schweregrad der CVE, und die Paketansicht der Konsole zählt abgelehnte Prüfungen richtig.
1.3.3, 7. Oktober 2026	Migrationen laufen unter einem Datenbank-Lock, eine Neuinstallation oder ein Upgrade mit Migrationen startet also keinen Pod mehr einmal neu.