



Product handbook

Container image security for Kubernetes. Attestkeep scans images, verifies their signatures, decides at admission whether a pod may start, and keeps a signed record of each decision. This handbook covers what it installs, how to run it, and what it costs.

Attestkeep 1.3.7 · Helm chart `oci://ghcr.io/attestkeep/charts/attestkeep` version 1.3.7 · handbook dated 15 October 2026.

Most screenshots were taken on 3 October 2026 from an installation running 1.3.2 under Community Edition. The six screens that changed by 1.3.7 were taken again on 15 October 2026 from a second installation running the published 1.3.7 under Enterprise Edition. Both run on Kubernetes v1.35.1, and the appendix lists which screen comes from which. Two first-run screens were taken earlier and are marked where they appear. Commands, settings and limits come from the product documentation at docs.attestkeep.com, the chart and the console. Nothing in this handbook is illustrative.

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

Contents

1. What Attestkeep is
 2. What runs in your cluster
 3. How an admission decision is made
 4. Install, upgrade and first run
 5. Using the console
 6. Admission policies
 7. Runtime events from Falco
 8. Prove it is enforcing
 9. Evidence packages and the trust model
 10. Operations, alerting and backup
 11. Licensing
 12. Editions and prices
 13. Security, support and who you are dealing with
- Appendix. What this handbook was checked against

1. What Attestkeep is

Attestkeep is a Kubernetes operator. It scans the images your workloads use, verifies their signatures and attestations, and answers the API server at admission time with allow or deny, according to a policy you write. Every decision is recorded. The records become a period report that an auditor can check with their own tools.

Everything runs inside your cluster. One Helm release installs the operator, its API, the console and the database into a namespace you own. Image names, findings and cluster topology do not leave the cluster. The licence check sends three values once a day.

What it does

- **Vulnerability scanning.** Trivy is built into the image. Findings are listed by severity, with the fixed version where there is one.
- **Signature verification.** cosign verification by key or keyless, and in-toto attestations with a maximum age.
- **Admission control.** Severity thresholds, tag and registry rules, digest pinning, signature and attestation requirements, fifteen workload hardening controls, a gate on CISA's Known Exploited Vulnerabilities catalogue, a gate on base OS end of support, a SLSA build provenance gate that checks who signed, producer VEX statements, and stricter rules for pods that run privileged or exposed.
- **Triage with an expiry.** You can accept a risk with a reason and an end date. Break-glass lets a named image through during an incident. Both are recorded.
- **Evidence packages.** One signed document per audit period, mapped to the control set you choose. It states what it cannot prove as well as what it can.
- **Audit log.** Policy changes, admission overrides, triage decisions, evidence exports and sign-ins in one record.
- **Notifications.** Slack, Teams, e-mail and a generic webhook. A notification names the cluster, never its topology.
- **Issue trackers.** A new critical finding opens an issue in your GitHub, GitLab or Jira, one per image digest and CVE. The issue is closed with a comment that says why: fixed, triaged, or image retired. Only *fixed* counts as remediation.
- **Runtime events from Falco.** Your own Falco sends its events to the operator over mutual TLS, either directly or through Falcosidekick. Each event is stored against the image and pod it came from. Runtime policies you write in the console decide what happens next: keep a record, send a notification, open an issue, or stop the pod. A runtime event never changes an admission decision. Chapter 7 covers this.
- **Single sign-on and roles.** Google, GitHub, GitLab, Okta, Microsoft Entra ID, Keycloak or any OIDC provider, on paid editions. Roles decide who may change a policy.

What it does not do

Attestkeep is not a runtime sensor. It does not watch syscalls, network flows or processes inside a running container. Falco does that detection, and Attestkeep only receives what Falco reports. Attestkeep does not ship, install, configure or write to Falco.

It is not a cloud posture tool and does not read your cloud account or IAM policies. It does not scan Terraform, manifests or the cluster's own configuration against a benchmark. It does not check whether a vulnerable function is ever called, and it does not read your repositories or open dependency pull requests. docs.attestkeep.com lists what to run beside Attestkeep for each of these.

Why not build the gate from free tools?

You can. There are good open-source admission policy engines and in-cluster scanners. If you only need to block `:latest` and fail on critical CVEs, you can do that without Attestkeep. What such a setup usually lacks is the record that auditors and incident reviews use:

- **A decision record with integrity.** Each admission decision is hashed when it is written and sealed under signed, chained checkpoints. If one verdict is changed in the database, the next evidence document names the record.
- **Decisions tied to the policy that made them.** Each record carries the hash of the policy as it was at that moment.
- **Gaps that are reported.** Periods when the webhook was down or could not be verified are recorded and printed in the evidence. The reconciliation sweep lists what got in during those periods.
- **One product.** Scan results, signature checks, admission, exceptions and the evidence package are maintained together, so your team does not have to maintain the glue between them.

2. What runs in your cluster

An admission webhook gets a say over every pod. This is the inventory of what that involves.

Object	Default	What it does
Deployment <code>attestkeep</code>	2 replicas, anti-affinity preferred across nodes, PodDisruptionBudget minAvailable 1	The operator: admission webhook (8443), console and API (8080), metrics (9090), and the Falco integrations listener (8444) when <code>integrations.falco.enabled</code> is on. Controller work is serialised through a PostgreSQL advisory lock, so replicas scale without a leader-election lease.
Deployment <code>attestkeep-scanner</code>	1 replica	Runs Trivy scans and cosign verification outside the admission path. It never receives admission traffic.
StatefulSet <code>attestkeep-postgresql</code>	1 replica, 20 GiB, postgres:16.15-alpine	The bundled database. If you set <code>externalDsn</code> to your own PostgreSQL, this object is not created.

All product containers run as non-root (uid 65532) with a read-only root filesystem, all capabilities dropped and RuntimeDefault seccomp. Requests default to 100m CPU and 256Mi per product workload.

Cluster-scoped objects are four CRDs in the `attestkeep.com` group (ImageSecurityPolicy, ImageSecurityReport, AttestationRenewal, ImageSecurityUsage) and one ValidatingWebhookConfiguration. The webhook covers pod CREATE and UPDATE, including the `pods/ephemeralcontainers` subresource that `kubectl debug` writes to, with a 15-second timeout and `failurePolicy: Ignore` by default. There is no mutating webhook. The product never rewrites your workloads.

Permissions worth knowing about

- The ClusterRole reads nodes (`get`, `list`). This is how the operator counts Ready nodes against the licence's node ceiling. It keeps the count only, never a node name.
- Secrets are readable cluster-wide with `get` only, not `list`. The operator can read a pull secret whose name it found in a pod spec. It cannot list what a namespace holds.
- `pods/eviction` is granted only with `rbac.allowEnforce: true` (default false). Two features use it: a runtime policy that stops pods, and `runtimeReconciliation: enforce`. Without the grant, neither can evict anything.
- A namespaced Role allows writes to three named Secrets in the operator's namespace: the webhook serving certificate, `attestkeep-evidence-key`, and `<release>-falco-mtls`, which exists only when the Falco integration is on.
- A second Role, `<release>-rule-validation`, lets the operator create, read and delete pods and ConfigMaps in its own namespace and read pod logs. The Falco rule editor uses it to run a one-off validation pod. It cannot start pods in any other namespace.

Every outbound connection

Use this table for your firewall request. There is no analytics, crash reporting or telemetry SDK.

Destination	Required?	Setting	If blocked
lic.attestkeep.com:443	Yes, unless activated offline	<code>license.serverUrl</code>	Online activation is not possible. An activated install keeps running on the certificate it holds and falls back to Community when that certificate expires.
ghcr.io (Trivy vulnerability DB):443	For scanning	<code>trivy.dbRepository</code> , to use a mirror	The existing database keeps working and gets older. <code>attestkeep_vulnerability_db_age_seconds</code> shows by how much.
Your image registries:443	For scanning	Pull secrets from pod specs	Scans fail for those images. Admission follows the policy's setting for unscanned images.
ghcr.io/attestkeep/attestkeep-k8s:443	At install	<code>image.repository</code> , to use a mirror	Pods cannot start.
Sigstore TUF and public Rekor:443	No	<code>scan.cosignOffline: true</code>	With keyed signing and offline mode, verification stays local.
Release feed	No, off by default	<code>updates.endpoint</code>	Nothing. An installation that has no endpoint set never looks.
Your OTLP collector, Slack, Teams or webhook URLs, SMTP server, SSO provider	No, configured by you	Per feature	That feature fails and logs it. A notification failure never blocks admission.

If our licence server were compromised

It could not instruct your cluster. The licence protocol has two kinds of response: a certificate and a renewal status. No code path lets a server response choose a policy, a namespace, an image rule, a URL or a program to run. Certificates are signed with Ed25519, checked against a public key compiled into the binary, and bound to your cluster's fingerprint.

A hostile server could answer *revoked*. The installation would then fall back to admission with warnings only, which shows in the console and in every admission response. It could not place a workload or change a policy. If the server does not answer at all, nothing happens: network errors count as an outage, and the installation runs on its certificate until that certificate expires.

3. How an admission decision is made

The webhook answers from cache. Scanning never runs in the admission path, so a slow registry does not slow down a deployment.

1. The API server asks about a pod: when it is created or updated, and when `kubectl debug` adds a container to a running pod.
2. The image tag is resolved to a digest. The result is cached for a minute, so a burst of identical deployments makes one registry call.
3. The scan result and signature verdict for the digest are read from cache.
4. The policy is evaluated against them. The answer is allow or deny, with a reason.
5. The decision is recorded, allows included. Evidence that lists only denials cannot show that a control was working.

None of these steps waits for a scan. If a digest has never been seen, step three has no answer, and the policy decides what happens.

Falco events are not part of this path. A runtime event never changes whether a pod is admitted.

Debug containers. From 1.3.7 a container added with `kubectl debug` goes through the webhook too. Only the added container is decided. Its image is scanned and held to the policy, and an enforcing policy refuses it like any other image. The pod's existing containers are not reviewed again. A debug image is not counted against the monthly image allowance or the node ceiling, and its ledger row's reason ends in `; subresource ephemeralcontainers`. Before 1.3.7 such a container was never sent to the webhook, whatever image it ran.

The two settings that decide your posture

Setting	Default	What it means
<code>webhook.failurePolicy</code> (chart)	Ignore	What the API server does when the operator does not answer. <code>Ignore</code> admits pods unchecked during an outage. <code>Fail</code> denies until the operator answers.
<code>admissionTiming</code> (policy)	AllowAndScan	What happens to a digest seen for the first time. <code>AllowAndScan</code> admits it and queues a scan. <code>DenyUntilScanned</code> refuses it until a scan record exists. It is the usual partner of <code>Fail</code> .

Neither choice is right for everyone. `Ignore` keeps the cluster deployable during an operator outage, and leaves a gap in the evidence. The evidence package reports that gap. `Fail` gives a control with no gaps, and an operator outage then stops deployments. Both settings are available in every edition, Community included.

Whatever got in during a gap is still running afterwards. The runtime reconciliation sweep finds it. On a schedule (hourly by default) it checks every running container's image ID against the admission ledger. A digest with no admission behind it is reported, notified, or, under `runtimeReconciliation: enforce`, stopped so that it comes back through the webhook.

The first scan of a new digest runs in the background. On a new cluster it can take a few minutes, because the scanner has to pull the image and the vulnerability database is not yet cached. Until it finishes, the image is in the *Unscanned* state on the Images page. Later admissions of that digest use the finished report. *Rescan* runs a scan on demand.

4. Install, upgrade and first run

Before you start

- Kubernetes 1.27 or later.
- Permission to create a ValidatingWebhookConfiguration, which means cluster-admin on the first install.
- About 20 GiB for the bundled PostgreSQL, or the DSN of a PostgreSQL you already run.
- An Attestkeep account at attestkeep.com and a licence key: the free Community key or the plan you bought.

Verify the release before you install

Every release is signed on its digest. The keyed check needs only your registry and the public key at docs.attestkeep.com/cosign.pub.

```
curl -s0 https://docs.attestkeep.com/cosign.pub
cosign verify --key cosign.pub ghcr.io/attestkeep/attestkeep-k8s:1.3.7
```

Install

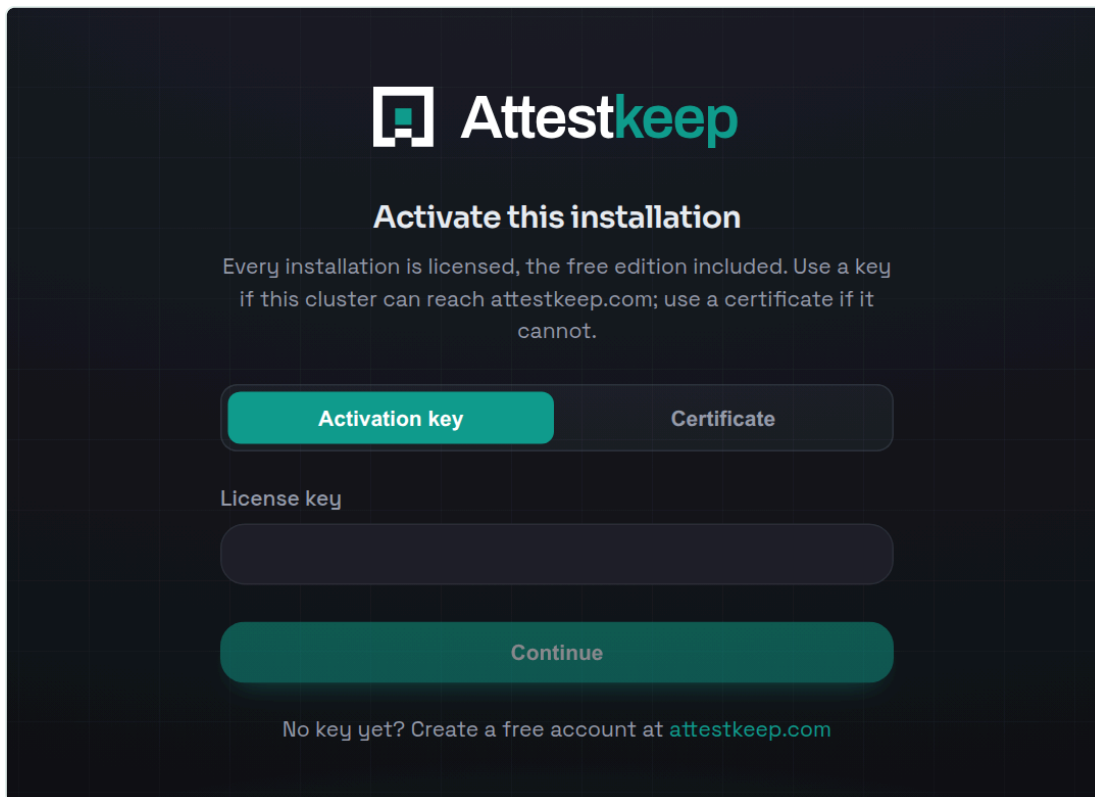
```
# installs from the registry; no `helm repo add` needed
helm install attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --version 1.3.7 \
  --namespace attestkeep --create-namespace \
  --set clusterName=production-eu \
  --wait
```

Set `clusterName` on the first install, even for a trial. Notifications, evidence documents, the dashboard and Settings > About use it to name the installation. Without it they fall back to the first node's name. It is a name you choose. No hostname or address leaves the cluster.

Check it is running and open the console

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

Open <http://localhost:8080>. A new installation shows two setup screens once: activation, then the owner account. After that the console opens on the security overview, described in chapter 5.



Attestkeep

Activate this installation

Every installation is licensed, the free edition included. Use a key if this cluster can reach attestkeep.com; use a certificate if it cannot.

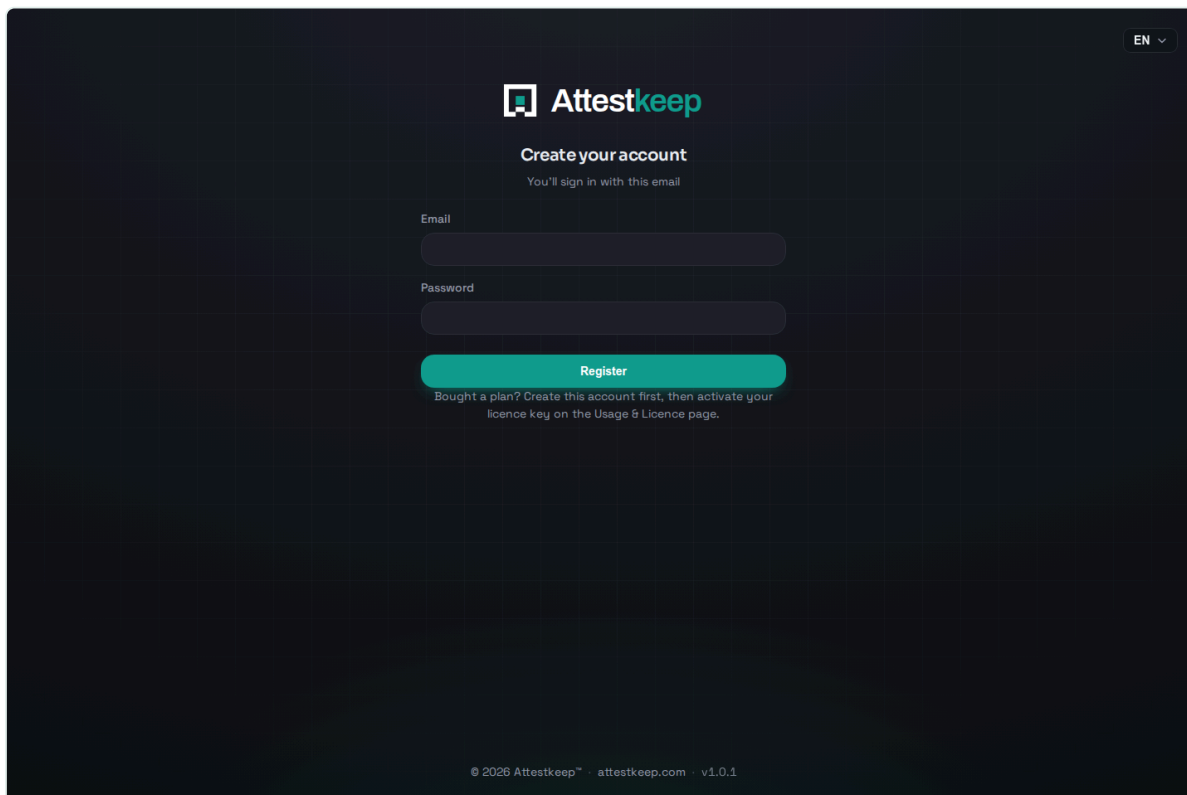
Activation key Certificate

License key

Continue

No key yet? Create a free account at attestkeep.com

Screen 1, *Activate this installation*. Every installation is licensed, Community included. On the *Activation key* tab you paste your key and choose *Continue*. The key is exchanged for a signed certificate that the operator checks inside the cluster. A cluster that cannot reach attestkeep.com uses the *Certificate* tab instead (chapter 11).



EN

Attestkeep

Create your account

You'll sign in with this email

Email

Password

Register

Bought a plan? Create this account first, then activate your licence key on the Usage & Licence page.

© 2026 Attestkeep™ · attestkeep.com · v1.0.1

Screen 2, *Create your account*. The first person to complete it becomes the owner, and the screen closes once the account exists. If you bought a plan after installing with a Community key, activate the paid key later on the Usage and License page. This screenshot was taken on version 1.0.1. The text on the screen is the same in 1.3.2.

Using your own database

Migrations are built into the binary and applied on first connection. Each one is recorded so it runs once. To use a database you already run, put the DSN in a Secret rather than on the command line:

```
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 need no configuration. The operator uses the image pull secrets the pod references, and those on its service account, for both the scan and the signature check.

Uninstall is `helm uninstall`. The webhook configuration is removed with the release, so admission review stops at that moment. The database volume is not deleted.

Upgrading

An upgrade is `helm upgrade`. Built-in migrations apply what is new and skip the rest. `helm upgrade --reuse-values` works across these versions: values added since your installed version take their chart defaults. Migrations only move forward. Where a step below lists migrations, going back to the earlier version means restoring a database backup, not `helm rollback`.

From → to	Migrations	What to do
1.3.6 → 1.3.7	0033 to 0035	Upgrade with <code>helm upgrade</code> , not by replacing the image: the new webhook rule for <code>pods/ephemeralcontainers</code> is in the chart. From this version <code>kubectl debug</code> with an image the policy does not allow is refused. 0033 adds the table that keeps What-if runs, 0034 adds two columns to the images table and 0035 fills them from the ledger. Going back to 1.3.6 is a restore from backup. No new chart values.
1.3.2 → 1.3.6	None	Nothing to do. 1.3.3, 1.3.4, 1.3.5 and 1.3.6 bring no migrations and no new chart values.
1.3.1 → 1.3.2	None	Nothing for the operator. <code>helm rollback</code> to 1.3.1 works. If you copied the <code>falcoctl</code> values from the 1.3.1 rule editor, add <code>falco-rules:5</code> back to <code>install.refs</code> and <code>follow.refs</code> . Without it, <code>falcoctl</code> stops installing Falco's own <code>/etc/falco/falco_rules.yaml</code> .
1.3.0 → 1.3.1	0026 to 0032	The policy resource no longer has a <code>spec.runtime</code> field. Remove it from your policy files if you set it: <code>kubectl apply</code> refuses a policy that still has one with <code>unknown field "spec.runtime"</code> . Migration 0031 changes runtime rules saved with the removed action to notify with an issue. New notification channels are subscribed to Runtime alert. Existing channels keep their subscriptions.
1.2.x → 1.3.0	0025	Re-activate the cluster. A 1.3.0 or later operator needs a certificate that carries the node ceiling and refuses an older one with "re-activate with a key issued by licence service 1.3.0 or later".
1.1.x → 1.2.0	0021 to 0024	Falco events sent through Falcosidekick with a bearer token stop arriving until Falco is connected over mutual TLS (chapter 7). Deleting a user now disables the account.

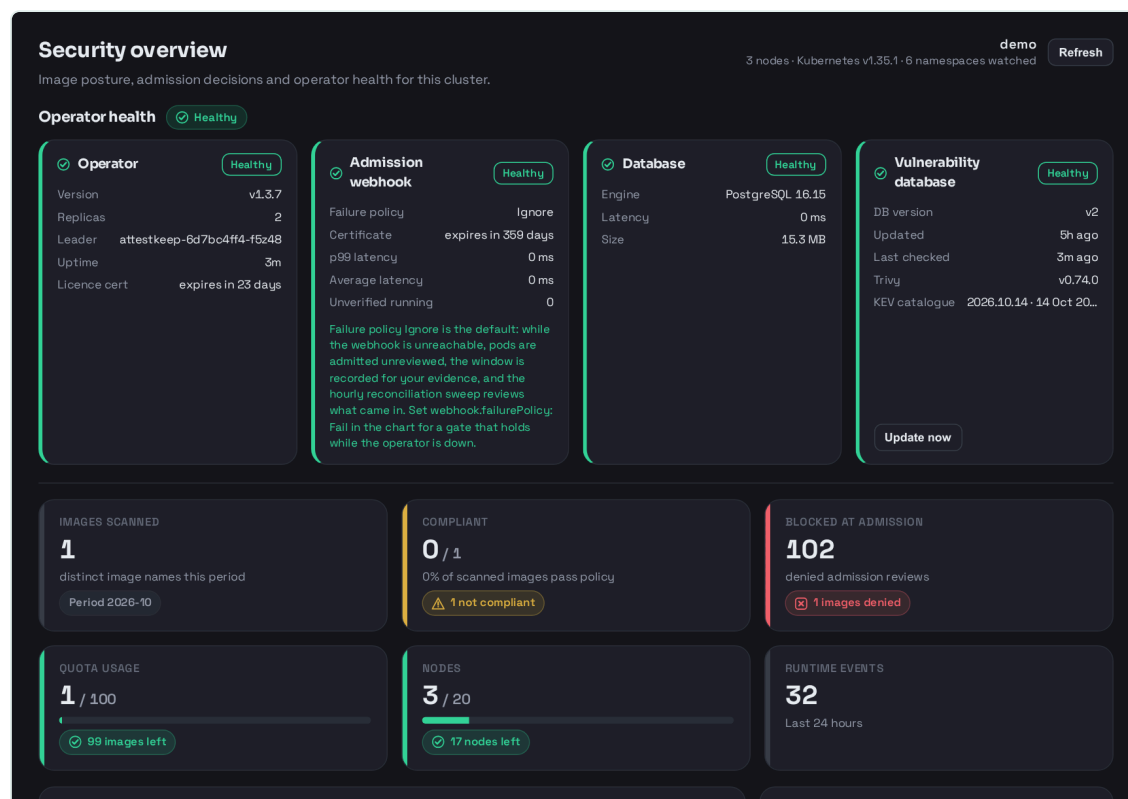
An installation that manages its own RBAC adds the `<release>-falco-mtls` Secret to the Role before turning on the Falco integration.

5. Using the console

The console comes with the release. The same Helm release that installs the operator serves it. It loads no fonts, scripts or analytics from outside. It is available in English and Turkish, in light and dark themes, with a help panel on each screen.

The sidebar has four groups. *Security* holds Images, Vulnerabilities, Policies and What-if history. *Falco* holds Policies, Rules, Events and Falco integration, and appears when the Falco integration is on. *Compliance* holds Compliance evidence and the Audit Log. *Settings* holds Organisation, Appearance, My Notifications, Webhooks, Scanning, SMTP, Single Sign-On, Logging & telemetry, Usage and License, Users, Backup and About.

The security overview



The dashboard of the Enterprise Edition installation running 1.3.7. All four health tiles are healthy. The *Admission webhook* tile shows the p99 and average latency of the last 24 hours. The *Vulnerability database* tile shows when the database was updated and when the scanner last checked, with *Update now*. Below them: 1 image name scanned this period, 102 denied admission reviews, *Quota usage* 1 of 100 and *Nodes* 3 of 20.

From 1.3.5 the two tiles no longer raise a warning on their own. The webhook tile does not compare latency with a fixed budget, because the first review of an image resolves its digest at the registry and takes longer than the later reviews, which answer from cache. The operator refreshes the vulnerability database and the KEV catalogue every six hours. An administrator can press *Update now* to download at once. The tile then says whether the database changed, was already current, or why the download failed, and the request is written to the audit log as `vulnerability_db_refresh`.

The Nodes card appeared in 1.3.1. While the cluster is over its ceiling, the strip shows since when and the date new image names will be denied from. Once that date has passed, it says enforcement is on. Nothing already running stops in either case.

Recently blocked admissions 271 denied

The most recent refusals, newest first

VERDICT	NAMESPACE	POD	IMAGE	REASON	POLICY	REVIEW	WHEN
Denied	attestkeep-e2e	bg-after root	nginx:latest	the :latest tag is blocked by policy	e2e-breakglass fa765f573847	9 ms Full review	1d ago
Denied	attestkeep-e2e	bg-before root	nginx:latest	the :latest tag is blocked by policy	e2e-breakglass 8a6312486629	2 ms Full review	1d ago
Denied	attestkeep-e2e	pol-digest root	docker.io/library/busybox...	policy requires images to be pinned by digest	e2e-policy 17327a38a8a9	1 ms Full review	1d ago
Denied	attestkeep-e2e	pol-registry root	docker.io/library/busybox...	registry "docker.io" is not in the policy's allow-list	e2e-policy 586b3de37d2a	4 ms Full review	1d ago
Denied	attestkeep-e2e	rollout-denied-6c8d75c65b-6xdjv root	nginx:1.27-alpine	2 CRITICAL vulnerabilities	default 2bfc5885a1a1	2 ms Cached	1d ago
Denied	attestkeep-e2e	rollout-denied-6c8d75c65b-89zfb root	nginx:1.27-alpine	2 CRITICAL vulnerabilities	default	2 ms Cached	1d ago

Recently blocked admissions lists refusals only, newest first, with the reason and the policy that gave it. The badge counts all denied reviews, here 271. Before 1.3.1 this table also listed allowed admissions.

Images

Images Rescan all

Every container image admission has seen in this cluster, and the scan report behind each one.

1 IMAGES	0 COMPLIANT	0 NOT COMPLIANT	1 BLOCKED	0 UNSCANNED	Search image, tag or namespace							
IMAGE ▲	TAG	NAMESPACES	PODS	COMPLIANCE	C	H	M	L	SIGNED	RUNTIME	LAST SCANNED	
ghcr.io/stefanprodan/podinfo sha256:ec73780a8425	6.15.0	gozlem uretim	1	Non-compliant	—	6	21	25	✓ Signed	1	just now	Rescan

1-1 of 1 1

Every image admission has seen in this cluster, with its compliance verdict, findings by severity, signature state, Falco events and last scan. The 1.3.7 installation has one image, counted under *Blocked* because its last admission review was refused. *Rescan* queues a scan on demand. The *Runtime* column counts each image's Falco events over the last 7 days and links to an open runtime issue. It appears only after the operator has stored at least one event.

From 1.3.7 each image is in exactly one of four states, so the tiles add up to the list. The first state that applies wins:

- **Blocked**: its last admission review was refused.
- **Unscanned**: admitted, with no scan record yet.
- **Compliant**: admitted, scanned and within the policy.
- **Not compliant**: admitted and scanned but outside the policy, typically under an audit-mode policy or a break-glass window.

Blocked follows the last admission review, not the verdict the scanner reaches later. An image admitted under an audit-mode policy before its first scan and found to carry criticals afterwards is Not compliant, and its detail reads "Admitted; the next admission would be denied". The dashboard's Compliant and Blocked figures use the same rule, so the two pages agree. The *Pods* column counts admitted pods only. A refused admission is not a running pod.

From 1.3.1 the Images and Vulnerabilities lists are read a page at a time. Filters, search and sorting run in the operator, and the counts above each table cover every row.

docker.io/library/nginx:1.27-alpine

linux/amd64 · Last scanned 19h ago

✗ Non-compliant · Denied at admission

2 CRITICAL vulnerabilities

METADATA

DIGEST

sha256:65645c7bb6a0661892a8b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10

PLATFORM

FIRST SEEN

PODS

linux/amd64

1 Oct 2026

168

NAMESPACES

acc24-ns, attestkeep-e2e

SIGNATURE AND ATTESTATION

No signature

No Cosign signature was found for this digest.

No attestation

No in-toto attestation was found for this digest.

No build provenance

This digest carries no verified SLSA provenance attestation, so the build system that produced it cannot be established.

Base OS in support

Release `alpine 3.21.3`

VULNERABILITIES 159

Search CVE or package

All severities

CVE	SEVERITY	PACKAGE	INSTALLED	FIXED IN
> CVE-2026-31789	CRITICAL	libssl3	3.3.3-r0	3.3.7-r0
> CVE-2026-31789	CRITICAL	libcrypto3	3.3.3-r0	3.3.7-r0
> CVE-2025-15467	HIGH	libcrypto3	3.3.3-r0	3.3.6-r0
> CVE-2025-15467	HIGH	libssl3	3.3.3-r0	3.3.6-r0

The panel for `nginx:1.27-alpine`. It is non-compliant and was denied at admission for 2 CRITICAL vulnerabilities. Above the findings the panel answers four questions: is there a cosign signature, an in-toto attestation, verified SLSA build provenance, and is the base OS still supported. Here the base OS, Alpine 3.21.3, is in support. The panel also lists the image's last 20 Falco events when there are any.

The same scan is on the cluster for anyone who reads it with `kubectl`. Each `ImageSecurityReport` carries the scan summary in `status.summary`, the findings in `status.vulnerabilities` (most severe first, at most 1000) and the scan time in `status.lastScanned`. A triage decision is copied onto the report with `triagedBy` set to `user:<account id>`, never a name.

15 October 2026

Page 13 of 52

Vulnerabilities and triage

Vulnerabilities

25 CVEs, 230 findings across 6 images

Findings with no released fix are excluded by the scan settings - these numbers cover fixable findings only.

230

FINDINGS

2

CRITICAL

69

HIGH

230

FIXABLE

0

TRIAGED

SEVERITY

CRITICAL

HIGH

MEDIUM

LOW

UNKNOWN

FIX

Any

Fixed

Not fixed

TRIAGE

Any

Untriaged

Triaged

Expiring soon

EXPLOITED

Known exploited only

Search CVE, package or title

CVE	SEVERITY	TITLE	PACKAGE	INSTALLED	FIXED IN	IMAGES	TRIAGE
> CVE-2026-31789	CRITICAL	openssl: OpenSSL: Heap buffer overflow ...	libcrypto3	3.3.3-r0	3.3.7-r0	2	Untriaged <button>Triage</button>
> CVE-2025-15467	HIGH	openssl: OpenSSL: Remote code executi...	libcrypto3	3.0.15-r0	3.0.19-r0	4	Untriaged <button>Triage</button>
> CVE-2025-26519	HIGH	musl libc 0.9.13 through 1.2.5 before 1.2.6 ...	musl	1.2.3-r3	1.2.3-r4	4	Untriaged <button>Triage</button>
> CVE-2025-32414	HIGH	libxml2: Out-of-Bounds Read in libxml2	libxml2	2.13.4-r5	2.13.4-r6	1	Untriaged <button>Triage</button>
> CVE-2025-32415	HIGH	libxml2: Out-of-bounds Read in xmlSche...	libxml2	2.13.4-r5	2.13.4-r6	1	Untriaged <button>Triage</button>
> CVE-2025-49794	HIGH	libxml: Heap use after free (UAF) leads to ...	libxml2	2.13.4-r5	2.13.9-r0	1	Untriaged <button>Triage</button>
> CVE-2025-49795	HIGH	libxml: Null pointer dereference leads to ...	libxml2	2.13.4-r5	2.13.9-r0	1	Untriaged <button>Triage</button>
> CVE-2025-49796	HIGH	libxml: Type confusion leads to Denial of ...	libxml2	2.13.4-r5	2.13.9-r0	1	Untriaged <button>Triage</button>
> CVE-2025-59375	HIGH	firefox: thunderbird: expat: libexpat In Ex...	libexpat	2.7.0-r0	2.7.2-r0	1	Untriaged <button>Triage</button>

Findings across all images, grouped by CVE, with installed and fixed versions side by side. Filters cover severity, whether a fix exists, triage state and *Known exploited only*. The list is read 25 CVEs at a time. Here: 25 CVEs, 230 findings across 6 images. The line under the heading says this installation's scan settings leave out findings with no released fix.

Triage is a record an auditor reads, so it cannot be a silent mute. You choose a status (false positive, accepted risk or under investigation), write a justification and set an expiry. When it expires, the finding goes back to the policy verdict. Each decision records who made it and when, and appears in the audit log and the next evidence package.

Admission policies

Image security policies

4 policy resource(s) deciding what admission allows

New policy

audit

POLICY	MODE	NAMESPACES	RULES	SUPPLY CHAIN	BREAK-GLASS	ACTIONS
gozlem-audit	Audit Allow and scan asynchronously	gozlem	Denies on CRITICAL or HIGH · no :latest · Tags allowed	Signature Attestation	—	What if Edit Break-glass Delete

1-1 of 1

< 1 >

The policy list on the 1.3.7 installation, searched for *audit*. Of its four policies, one runs in Audit mode: `gozlem-audit`. Each row shows the mode, the namespaces covered, a rule summary and the supply-chain checks required. From 1.3.7 every row also has *What if*.

[Back to policies](#)

Edit default

Everything the admission webhook checks before it lets a pod start. The form and the manifest are the same resource — edit whichever one you think in.

SCOPE

Which namespaces this policy governs, and which registries Images may come from.

Name

default

A policy name is its resource identity and cannot be changed after creation.

Target namespaces

☒ Apply cluster-wide

A cluster-wide policy covers every namespace that no namespace-scoped policy claims.

production, staging...

Allowed registries

ghcr.io, docker.io...

Host names, not URLs. Leave empty to accept any registry.

MANIFEST

[Edit](#) [Diff](#) +1 -1

ImageSecurityPolicy is cluster-scoped, so metadata carries no namespace.

```

apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: default
spec:
  enforcement: Enforce
  admissionTiming: AllowAndScan
  blockLatestTag: true
  digestEnforcement: AllowTags
  thresholds:
    blockCritical: true
    blockHigh: true
    maxMedium: 0
    maxLow: 0
  vex:
    mode: ignore
  slaWindows:
    critical: 24h0m0s
    high: 168h0m0s
    medium: 720h0m0s
    low: 2160h0m0s

```

Editing this manifest updates the form on the left.

ENFORCEMENT

What happens when an image breaks a rule.

Enforce

A pod whose image breaks this policy is denied at admission. Nothing starts.

[Cancel](#)
[Download](#)
[Copy for git](#)
[Last 30 days](#)
[What if](#)
[Apply to cluster](#)

The editor, as an account that may write policies sees it. The form and the manifest are the same resource, so you can edit either. *Copy for git* gives you the manifest for your repository. *Apply to cluster* writes it to the cluster. The editor detects when Argo CD or Flux owns the object.

From 1.3.7 *Apply to cluster* is shown only to accounts that may write policies. Everyone else keeps the What-if result, Download and Copy for git. *Cancel* asks before it discards unsaved changes. A new policy starts with the same SLA and triage durations the operator and the documentation use (critical 24h, high 168h, medium 720h, low 2160h, triage 2160h). A policy written with kubectl that leaves a window unset opens with these values, and the diff says so before you apply.

Whether a policy would refuse anything in your cluster depends on your images, scans and triage decisions, not only on its text. *What if* answers that from the cluster's own records. The draft is run against the admissions of the last 7, 30 or 90 days using the same function as the admission webhook. Nothing is applied: no pod is touched, no registry is contacted and the licence count does not move. Decisions are replayed against each image's scan as it is now, not as it was at the time.

What if on a row of the policy list is the short way. An audit-mode policy is replayed as if it were enforcing, and an enforcing policy as it stands, over the last thirty days. The result opens in the editor, and nothing is applied.

What this draft would have done

9 admissions across 5 images, over the last 30 days.

7
Unchanged

2
Would be denied

0
Would be allowed

NEW DENIALS BY RULE

Severity thresholds 2

DEPLOYMENT CONTEXT

This draft records each pod's context and escalates nothing. The counts below are what switching to Escalate would start acting on.

Runs as root 9

4 images have no scan record
Scan now

decisions replayed against each image's current scan record, not the scan that existed at the time

WHEN	NAMESPACE / POD	IMAGE	WAS → WOULD BE	REASON UNDER THE DRAFT
9 Oct 2026, 14:11	gozlem podinfo-64674c494f-798p4	ghcr.io/stefanprodan/podinfo:6.15.0	allowed → denied	10 HIGH vulnerabilities
9 Oct 2026, 10:59	gozlem podinfo-64674c494f-nwc6s	ghcr.io/stefanprodan/podinfo:6.15.0	allowed → denied	10 HIGH vulnerabilities

A stored run of `gozlem-audit` as an enforcing draft, over the last 30 days: 9 admissions across 5 images, 7 unchanged, 2 would be denied by the severity thresholds and 0 would be allowed. The panel lists the rules behind the new denials, the deployment-context signals of the replayed pods and the changed admissions with the draft's reason for each. Four images in the period have no scan record, and *Scan now* is offered for them.

Scan now. When images in the period have no scan record, the result names them. A deployer or an administrator can queue scans for exactly those digests, at most 500 per request, and each request is written to the audit log as `rescan`. The replay does not run again on its own. Press *What if* again once the scans have finished.

What-if history

WHEN	POLICY	BY	PERIOD	REVIEWED	WOULD DENY	WOULD ALLOW	UNSCANNED	ACTIONS
15 Oct 2026, 03:39	gozlem-audit	Demo Operator	30 days	9	2	0	4	View PDF JSON Apply this draft
9 Oct 2026, 14:09	gozlem-audit	Demo Operator	30 days	8	0	0	5	View PDF JSON Apply this draft

Security > What-if history, added in 1.3.7, filtered to `gozlem-audit`. The image shows the newest two runs of the table. Each run shows when it ran, the policy the draft was edited from, who ran it, the period, how many admissions were reviewed, how many would be denied or allowed, and how many images had no scan record. The list can be filtered by policy and period.

A replay run by a deployer or an administrator is kept for ninety days, at most five hundred runs. A viewer can still replay, and the answer is shown but not kept. Each run can be viewed again, downloaded as PDF or JSON, or, by an administrator, opened in the editor with *Apply this draft*. That loads the stored draft and applies nothing.

A download is a projection, not evidence. The PDF says so on every page and is neither signed nor sealed. A run never enters an evidence package, because the package records what happened and a replay records what would have. Each download is written to the audit log as `policy_whatif_download`.

The API is `POST /api/v1/policies/whatif`. It takes an optional `source`, the policy the draft was edited from (an unknown name is answered with `400`), and answers with `kept`, `run_id`, `unscanned`, `unscanned_total` and `notes`.

beside the counts. One account may start at most ten replays a minute on each operator replica. The next one is answered with `429` and `Retry-After`.

Compliance evidence

Compliance evidence						Generate package
One consolidated document per audit period. Scan reports are day-to-day work for the team; an evidence package is the record handed to an external auditor.						
PERIOD	FRAMEWORK	SIGNATURE	GENERATED AT	GENERATED BY	RECIPIENT	
02 Oct 2026 — 02 Oct 2026 PS-SOC2-DEVMN3-20261002-20261002-56A369	SOC 2	Signed	02 Oct 2026, 14:03 UTC	user:usr-e0b083dc06719775f70ac465	—	
01 Oct 2026 — 02 Oct 2026 PS-SOC2-DEVMN3-20261001-20261002-6605CF	SOC 2	Unsigned	02 Oct 2026, 21:12 UTC	user:usr-e0b083dc06719775f70ac465	auditor@e2e.local	
01 Oct 2026 — 02 Oct 2026 PS-SOC2-DEVMN3-20261001-20261002-C9AEE4	SOC 2	Unsigned	02 Oct 2026, 19:23 UTC	user:usr-e0b083dc06719775f70ac465	—	
01 Oct 2026 — 02 Oct 2026 PS-SOC2-DEVMN3-20261001-20261002-3A160D	SOC 2	Unsigned	02 Oct 2026, 13:21 UTC	user:usr-e0b083dc06719775f70ac465	—	
01 Oct 2026 — 02 Oct 2026	ISMS-P	Unsigned	02 Oct 2026, 13:13 UTC	user:usr-	—	

One document per audit period. Each row shows the period, the framework, whether the package is signed, when it was generated, and the account that generated it, written as `user:<account id>`. From 1.3.7 the framework names in this list and in the reports come from one list, so both name a framework the same way.

Generate evidence package

The package covers one closed audit period. Both dates are inclusive.

Framework

SOC 2 (Trust Services Criteria) ▼

Which instrument the period's records are assessed against. The same records produce different claims under each.

Period start **Period end**

10/01/2026 10/03/2026

☐ Include raw runtime events

Adds every Falco event of the period with its output and fields. Off: only the events that led to a decision are listed.

[Cancel](#) [Generate](#)

Generate evidence package. The *Framework* list holds the mapped control sets the licence allows. The period is one closed audit period, both dates included. *Include raw runtime events* adds every Falco event of the period, up to 5000. When it is off, only the events that a runtime policy acted on are listed.

When a licence covers fewer frameworks than the installation has already used, for example after a paid licence ends, the next owner or administrator to sign in is asked which frameworks to keep. Chapter 11 has the details.

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	02 Oct 2026, 14:03 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)

A requirement in the report view with its test opened. CC6.6 is not satisfied because no registry allow-list is in force, and the test says so. The method is EXAMINE because it reads a configuration. The table also gives the result, the source, when the source was read, the observation and the path into the report.

12 How this document was produced

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

Generated02 Oct 2026, 14:03 UTCGeneratorAttestkeep for Kubernetes 1.3.1

SourceAttestkeep PostgreSQL store, installation system758bafbac1475c5e5f05bbd79c034f12d513e0a84013b95dc2b81e5fd5d862e

Query window[2026-10-02T00:00:00Z, 2026-10-03T00:00:00Z) UTC

Peopleuser:<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]	21	—
admission.policy_revisions	the policy hashes of those rows, grouped, each joined to policy_revisions	1	—
chain (audit rows)	every row of audit_log with at in [period.start, period.end]	298	—
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	2	—
outcome.images	every row of images joined to its scan, merged to one row per name and tag, as the inventory stands at generation	16	—
findings.critical	every untriaged critical finding in the inventory at generation	2	—
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	—

The *How this document was produced* section of a SOC 2 package generated on 2 October 2026 by version 1.3.1. It gives the generation time, the generator, the source system, the query window (end excluded, in UTC), how people are named, and each section's query in words with the rows it returned and whether a capped query reached its cap.

Audit log

Audit Log
Every policy change, admission override, triage decision, evidence export and access event, in one record.

Retention: 30 days ⚙️ 🗑️ Clear 📄 Export ⚙️ 🔄 Refresh

🔍 Search by actor, policy, image, From 09/03/2026 To 10/03/2026 All Policy Scans Triage Evidence Access System

TODAY

🔄	E2E Owner replayed a draft policy whatif	POLICY	10:21
→	E2E Owner signed in login	ACCESS	10:20
→	E2E Owner signed in login	ACCESS	10:18
→	E2E Owner signed in login	ACCESS	10:16
⚙️	E2E Owner updated their preferences appearance	SYSTEM	10:14
→	E2E Owner signed in login	ACCESS	10:13

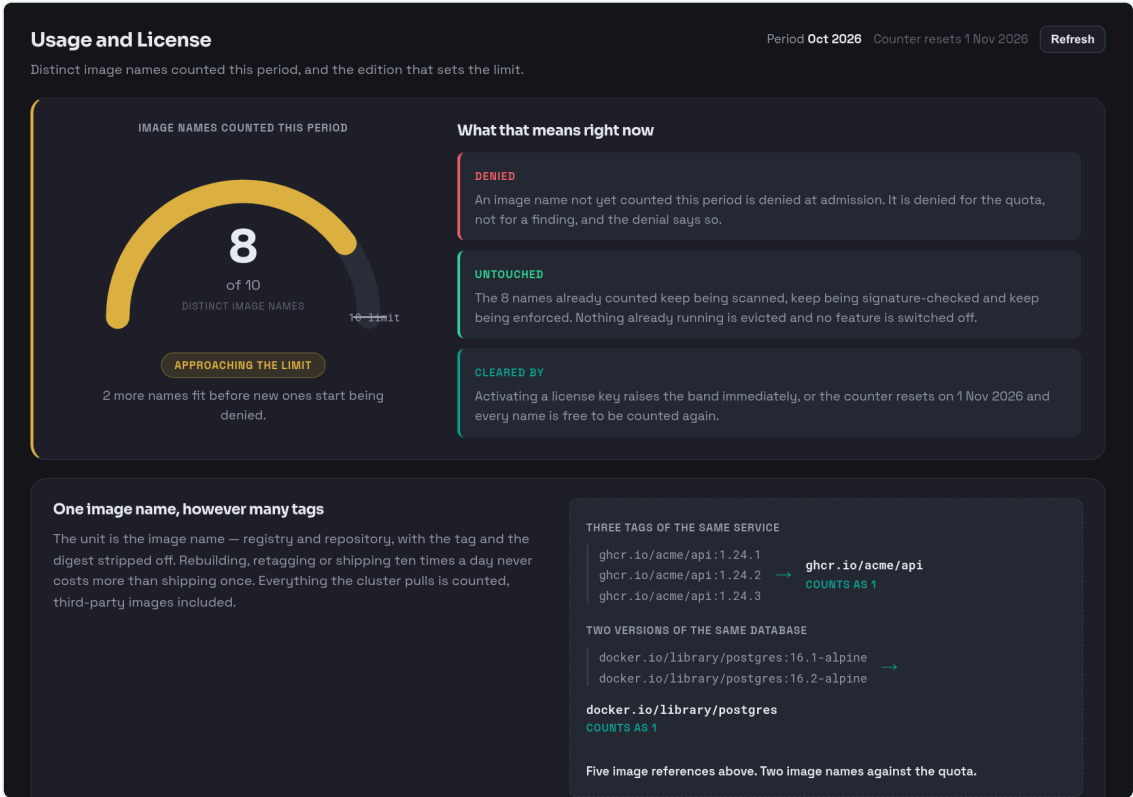
YESTERDAY

→	E2E Owner signed in login	ACCESS	22:31
→	E2E Owner signed in login	ACCESS	22:30

The audit log shows the last 30 days by default, 20 entries a page. *From* and *To* reach anything older. You can search by actor, policy, image or CVE, and filter by category: Policy, Scans, Triage, Evidence, Access or System.

Times are shown in your local time, and the CSV export keeps its ISO timestamps. Notification channel changes and tests, Falco integration changes, runtime policy changes, pods stopped by a runtime policy, the rule editor's validate, save and publish actions, What-if runs and downloads, Scan now and Update now are all recorded here. Channel actions do not go into evidence packages, because they describe what an operator did to the installation, not what the control decided about an image.

Usage and License



declared by the operator.

OPERATOR DECLARATIONS

Declared by the operator. Evidence packages print these as your statement; the product does not measure them.

TIME SOURCE

chrony on all nodes, synced to pool.ntp.org

RETENTION POLICY

admission ledger and audit log kept 5 years

Save

The two declaration fields. On this installation they are empty. The grey text is the example the form shows until you type.

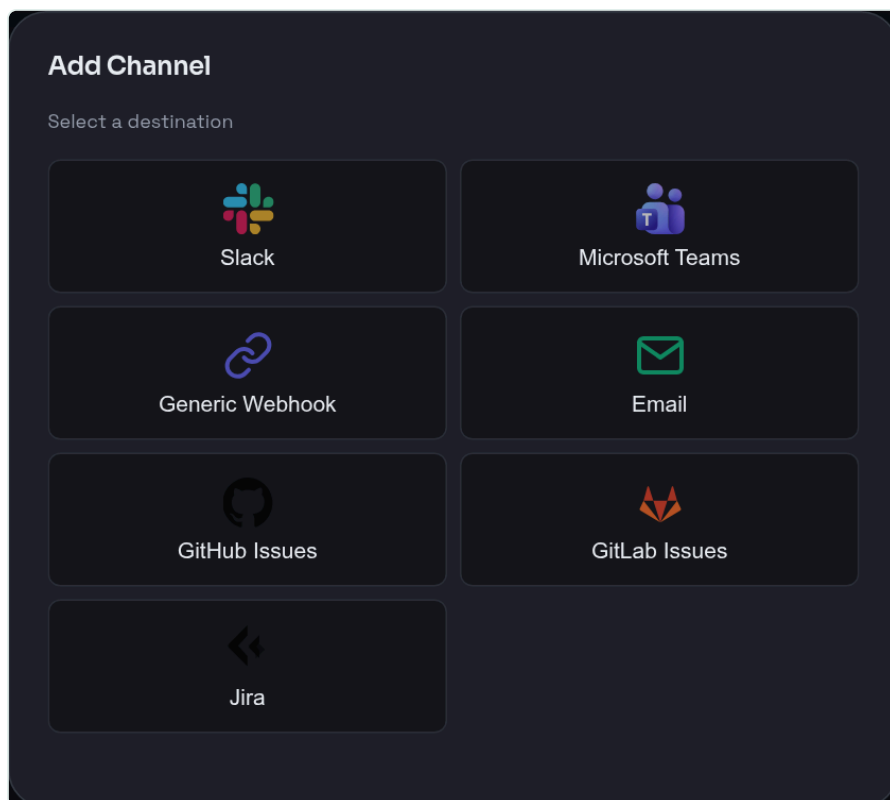
Deleting a user disables the account instead of removing it. A disabled account cannot sign in and its sessions end. Its id and audit rows stay, so every `user:<id>` in an issued package still resolves. For an erasure request an admin can then *Anonymise* the disabled account. This blanks its name, e-mail and avatar and rewrites every copy of its name in the audit log, triage decisions, and runtime policies and their revisions to `user:<id>`. Packages already signed keep what they were signed with, and comments already posted to an issue tracker are not changed.

	Anonymised account	Disabled	usr-b05c1da8af94170642f0752c	Viewer	1 Oct 2026	
	acc-viewer	Disabled	acc-viewer@acc.local	Viewer	1 Oct 2026	Anonymise

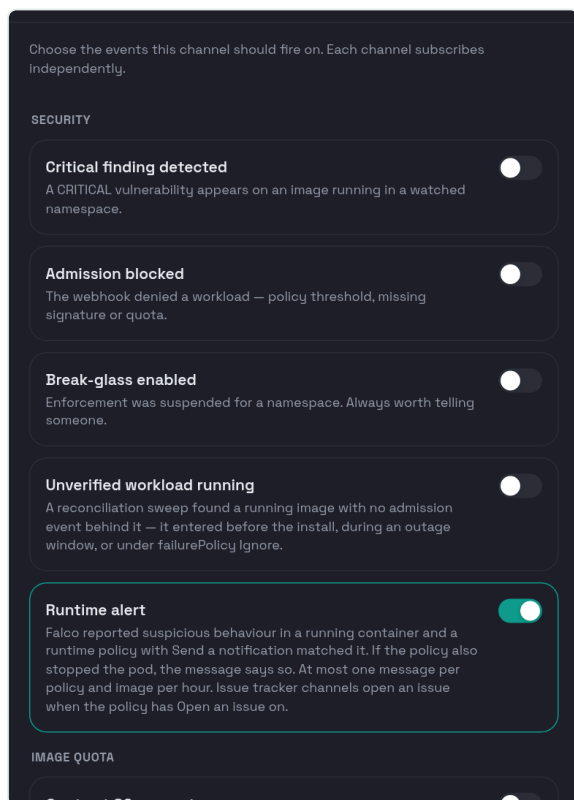
An anonymised account stays in the list as disabled, with its account id where the e-mail address was. The disabled account below it still offers *Anonymise*.

Notification channels

Channels are set up under Settings > Webhooks, on the *Notification Channels* page. Each person's own e-mail preferences are under Settings > My Notifications.



Add Channel. An issue tracker is a channel like the others, with the same subscriptions. *Test* on a tracker channel writes nothing. It reads the repository or project once and reports whether the credential works.



A channel's *Events* list. *Runtime alert* is on for every channel created from 1.3.1. Its description says what is sent: a message when a runtime policy with *Send a notification* matches, a note when the policy also stopped the pod, at most one message per policy and image per hour, and an issue when the policy has *Open an issue* on.

An issue-tracker token is the only credential Attestkeep holds that can write to a system outside your cluster. It is never returned by the API. It is encrypted with AES-256-GCM before it is stored, under a key the chart keeps in the `<release>`

`data-key` Secret and never in the database. Set `api.dataKey.existingSecret` to supply the key from your own secret manager. Without the key, a tracker channel cannot be saved. Nothing else depends on it.

Backup history

Export configuration

Downloads a JSON bundle of notification channels, triage decisions, the user list and the cluster's policies. It carries no credentials: webhook URLs, passwords and licence material never leave.

Export configuration

Import configuration

Reads a bundle exported from another installation. Nothing is overwritten: records that already exist here are kept and the incoming copy is skipped.

Import from file...

Backup history

Every configuration export and import on this installation. The file itself is not kept here.

When	Type	By	Contents	File / source
2 Oct 2026, 22:05	Import	E2E Owner	added 0 channels · 1 triage · 0 policies · 1 already present	dev-mn3 · 2 Oct 2026, 22:05
2 Oct 2026, 22:05	Export	E2E Owner	0 channels · 0 triage · 26 users · 1 policies	attestkeep-config-dev-mn3-20261002.json

1-2 of 2 < 1 >

Full database backup

Admission history, scans, evidence and the sealed ledger are the database itself and far bigger than a browser download. Back them up with `pg_dump` as described in the documentation: [docs.attestkeep.com/backup](#).

Settings > Backup. *Export configuration* downloads a JSON bundle of notification channels, triage decisions, the user list and policies, without credentials. *Backup history*, added in 1.3.1, lists every export and import with when, by whom and what the bundle carried. The operator does not keep the bundle file. The *Full database backup* card points to `pg_dump`, because the database backup is yours to run (chapter 10).

15 October 2026

Page 23 of 52

6. Admission policies

One cluster-scoped resource decides what may run: `ImageSecurityPolicy` in the `attestkeep.com` API group. The console edits it directly, and `kubectl apply` edits it from a repository. Both work on the same object. A change applies from the next admission decision, with no restart.

How a namespace gets its policy. A policy whose `spec.targetNamespaces` lists the namespace applies. Otherwise the policy without `targetNamespaces` applies, which is the global default. A common setup is one global policy and a stricter one for the namespaces that need it.

The shipped default

```
apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: default
spec:
  enforcement: Enforce           # Audit only records what it would have refused
  admissionTiming: AllowAndScan  # DenyUntilScanned refuses anything without a scan record
  blockLatestTag: true          # a mutable tag can change what runs without a deploy
  digestEnforcement: AllowTags  # AllowTags | PreferDigest | Required
  thresholds:
    blockCritical: true
    blockHigh: true
    maxMedium: 0                # 0 means unlimited here
    maxLow: 0
  slaWindows:
    critical: 24h
    high: 168h
    medium: 720h
    low: 2160h
  requireSignature: false       # turn on once your pipeline produces signatures
  requireSBOM: false
  requireVulnAttestation: false
  attestationMaxAge: 168h
  triageDefaults:
    maxDuration: 2160h
    requireJustification: true
```

A stricter policy for production 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: []
```

End-of-life base OS and known-exploited CVEs

Two gates look at something other than a severity count. Both accept `ignore`, `warn` or `block`, and both default to `warn`. A policy written before they existed keeps admitting what it admitted before, and now also says why it should not.

```
spec:
  # The base OS is past its vendor's support window, so no more security
  # fixes will be published for it. warn (default) admits and says so;
  # block refuses whatever the counts are; ignore says nothing.
  endOfLifeOS: warn

  # The image carries a CVE on CISA's Known Exploited Vulnerabilities
  # catalogue. warn (default) admits and names the CVEs; block refuses;
  # ignore says nothing.
  knownExploited: warn
```

The KEV catalogue is kept as a local copy and refreshed every six hours. A cluster without internet access points `kev.source` at a mirror, or sets it empty to keep the copy it has. Triaged CVEs are excluded from this gate, as they are from the severity thresholds. A refusal names the CVEs, so the person deploying knows what to fix. The dashboard shows the catalogue version in use, and evidence packages record it next to the vulnerability database version.

SEVERITY THRESHOLDS

CRITICAL and HIGH deny on the first finding; MEDIUM and LOW deny above a count.

☒ Deny on any CRITICAL finding
 ☒ Deny on any HIGH finding

Max MEDIUM findings

0 means no limit.

Max LOW findings

0 means no limit.

Base OS past end of life

Warn

The image is admitted and kubectl prints that its base OS receives no security fixes any more. Findings on it can only grow.

Known exploited vulnerability

Warn

The image is admitted and kubectl prints which of its CVEs are on CISA's Known Exploited Vulnerabilities catalogue.

The *Severity thresholds* section of the editor. *Base OS past end of life* and *Known exploited vulnerability* are both set to Warn, and the text under each says what that setting does.

Build provenance

This gate checks which build system produced the image and from what source. It is the only check that is tied to an identity. `requireSBOM` and `requireVulnAttestation` check that someone signed an attestation. This gate checks who.

```
spec:
  provenance:
    # 0 (default, also when the block is absent) turns the gate off.
    # Above 0 the image must carry verified SLSA provenance naming a
    # builder listed below, credited with at least this level.
    minLevel: 2
    trustedBuilders:
      # A trailing "*" matches by prefix; other ids match exactly.
      # An empty list with minLevel above 0 admits nothing.
      - id: https://github.com/acme/build/.github/workflows/release.yml@*
        # Your statement about this builder. SLSA provenance does not
        # carry a level, because a level describes how the builder is run.
        level: 3
        # Read from the signing certificate. If unset, it must equal the
        # builder id. If issuer is unset, any issuer is accepted.
        signer: https://github.com/acme/build/.github/workflows/release.yml@*
        issuer: https://token.actions.githubusercontent.com
```

An image is refused if its verified SLSA provenance (v1 or v0.2) does not name a trusted builder, if the signing certificate's identity or issuer is not the one the entry accepts, or if the builder's level is below `minLevel`. The reason names the builder, the signer or the level. Both identities are read from the certificate stored with the attestation. If that certificate cannot be fetched or parsed, the image is refused.

Producer statements (VEX)

A scanner reports what is installed in an image. It cannot tell whether the vulnerable code is reachable or ever runs. The party that built the image can, and OpenVEX is the format for that. Attestkeep reads a VEX statement only when it arrives as a signed in-toto attestation on the image and the policy names its signer. Unsigned statements, statements from signers the policy does not name, and statements about another product are recorded and ignored.

```
spec:
  vex:
    # ignore (default): trusted statements are verified, recorded against
    # the finding and shown in the console; no verdict changes.
    # apply: a trusted not_affected or fixed statement removes that finding
    # from the counts admission decides on.
    mode: apply
    trustedIssuers:
      # A keyless signer, by certificate identity. A trailing "*" matches
      # by prefix. issuer is optional and matched exactly.
      - identity: https://github.com/acme/images/*
        issuer: https://token.actions.githubusercontent.com
      # Or a keyed signer, by public key. An entry has an identity or a
      # key, not both.
      - key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE ...
          -----END PUBLIC KEY-----
```

`affected` and `under_investigation` never suppress a finding. A suppressed finding drops out of the counts that the severity thresholds and the known-exploited rule use, and the `allow` reason records how many findings were suppressed and by whom. `mode: apply` with an empty `trustedIssuers` list is refused, because it would look like a control and do nothing. A mode change applies from the next pod. A new issuer applies from the next scan of the digest.

SUPPLY CHAIN

☐ Require a valid signature

The image must carry a cosign signature this cluster trusts.

☐ Require a vulnerability attestation

The image must carry a signed scan result, not just a scan this cluster ran itself.

☐ Require an SBOM

Attestation max age

168h

Go duration: 168h, 24h, 90m.

Build provenance

Off

Nothing is asked about where the image was built.

Producer statements (VEX)

Record only

A trusted producer's statement is verified, recorded against the finding it names and shown here, and no verdict moves because of it.

Trusted issuers

Add issuer

Nobody is trusted yet, so no producer statement is read at all. Applying with an empty list is refused.

The *Supply chain* section. Signature, vulnerability attestation and SBOM requirements are off, *Build provenance* is Off, and *Producer statements (VEX)* is on *Record only*, the default. No issuer is trusted yet, so no statement is read.

Where the pod is going to run

The same HIGH finding means more in an internet-facing privileged pod than in a batch job nothing can reach. `deploymentContext` lets a policy treat those pods more strictly. It only ever makes a decision stricter. No signal or combination of signals admits a pod the policy would otherwise refuse.

```
spec:
  deploymentContext:
    mode: Escalate      # Ignore (default) | Escalate
    signals:            # empty means all six
      - exposed
      - privileged
```

Six signals are derived for every pod and recorded on every ledger row, in both modes: `privileged` , `root` , `hostNamespace` , `hostFilesystem` , `capabilities` and `exposed` . Only `exposed` looks outside the pod: `hostNetwork`, a `hostPort`, a `NodePort` or `LoadBalancer Service` that selects the pod, or a `ClusterIP Service` behind an `Ingress`. A `ClusterIP Service` with no `Ingress` does not count as exposure. Under `Escalate` , a pod with a selected signal is judged like this:

The policy says	A pod with a selected signal is judged as
<code>thresholds.blockCritical: true</code>	also blocking HIGH
<code>thresholds.blockHigh: true</code>	also blocking MEDIUM
<code>knownExploited: warn</code>	<code>block</code>
<code>endOfLifeOS: warn</code>	<code>block</code>

`maxMedium` and `maxLow` are not changed, and `ignore` stays `ignore` . The refusal names the signal and, for exposure, the `Service`, `Ingress` or `hostPort` that caused it.

DEPLOYMENT CONTEXT

Where a pod runs, as an input to the decision about its images. The same CVE is a different risk in an internet-facing privileged pod and in a batch job nothing can reach. Escalation only: a signal can make this policy stricter for a pod and can never make it looser.

What context does

Record only

▼

Every pod’s context is derived and written to the admission ledger, and no decision changes. Switch this on first: the ledger and the What if panel then say how much of the cluster Escalate would touch.

The *Deployment context* section set to *Record only*, which is `Ignore` . Each pod's context is written to the admission ledger and no decision changes. The ledger and the What if panel then show how many pods `Escalate` would affect.

Field reference

Field	Values	What it decides
enforcement	Enforce Audit	Refuse, or only record what would have been refused.
admissionTiming	AllowAndScan DenyUntilScanned	What happens to an image with no scan record yet.
blockLatestTag	bool	Refuse the <code>:latest</code> tag.
digestEnforcement	AllowTags PreferDigest Required	Whether images must be pinned by digest.
allowedRegistries	list of hosts	Empty allows any registry. Non-empty refuses all others.
targetNamespaces	list	Empty makes this the global policy. Non-empty limits it to those namespaces.
thresholds	blockCritical, blockHigh, maxMedium, maxLow	Refuse on any critical or high finding. Set ceilings for medium and low (0 means unlimited).
slaWindows	durations	How long a finding of each severity may stay open before reports call it overdue.
requireSignature / requireSBOM / requireVulnAttestation	bool	Refuse images without a verified cosign signature, an SBOM attestation or a vulnerability-scan attestation.
attestationMaxAge	duration	An older attestation no longer counts.
workloadHardening	mode, minSeverity, exempt	Whether privileged containers, hostPath mounts and similar settings refuse, warn or pass.
endOfLifeOS	ignore warn block	Images whose base OS is past its support window. Default <code>warn</code> .
knownExploited	ignore warn block	Images with a CVE on CISA's KEV catalogue. Default <code>warn</code> .
provenance	minLevel 0 to 3, trustedBuilders (id, level, signer, issuer)	Refuse images without verified provenance from a trusted builder at the required level. Default <code>minLevel: 0</code> , off.
vex	mode ignore apply, trustedIssuers	Whether a trusted producer's <code>not_affected</code> or <code>fixed</code> statement removes a finding from the counts. Default <code>ignore</code> .
deploymentContext	mode Ignore Escalate, signals	Judge pods with a selected signal more strictly. Default <code>Ignore</code> .
runtimeReconciliation	audit notify enforce	What the periodic sweep does about a running image with no admission behind it.
breakGlass	enabled, reason, approvedBy, auditRef, expiresAt, affectedImages	A named, approved exception that expires. Every use is recorded.
triageDefaults	maxDuration, requireJustification	How long an accepted risk may stand, and whether it needs a reason.

storageUnreachablePolicy	Allow Deny	The admission answer while the database is unreachable.
--------------------------	--------------	---

What a Falco event leads to is not part of the admission policy. It is set by runtime policies in the console (chapter 7).

Break-glass

When production is down and the fix is an image the policy refuses, break-glass lets the named images in, records each use, and expires on its own:

```
breakGlass:
  enabled: true
  reason: "INC-2417: rollback image carries a known high finding"
  approvedBy: "on-call, incident commander"
  auditRef: "INC-2417"
  expiresAt: "2026-09-01T06:00:00Z"
  affectedImages:
    - registry.example.com/payments/api:1.41.2
```

Every admission that passed only because of break-glass is marked that way in the audit trail and in compliance reports.

7. Runtime events from Falco

Falco watches what containers do after they start. Attestkeep can receive Falco's events, store each one against the image and pod that raised it, and act on it through runtime policies you write in the console.

Attestkeep does not ship, install, configure or write to Falco. Your Falco and its rules do the detection. A runtime event never changes an admission decision. What a runtime policy can do is notify, open an issue, and stop the pod the event came from.

How it works

1. A Falco rule fires on a node, for example when someone opens an interactive shell in a container.
2. The event is sent as JSON to the operator's integrations port over mutual TLS. Falco sends it from its own `http_output`, or Falcosidekick forwards it from its webhook output. Either way, the sender presents a client certificate the operator issued.
3. The operator stores the event and ties it to the image and digest the container runs.
4. Runtime policies are checked in order. The first enabled policy that matches decides what happens. If none matches, the event is kept as a record and nothing is sent.
5. The event and the decision appear under Falco > Events, next to the image on the Images page, and in the evidence package.

Requirements

- Falco installed with the upstream `falcosecurity/falco` Helm chart. The documented values are for chart 9.2.0 (Falco 0.45.0, Falcosidekick 2.32.0). Older versions have not been checked.
- `falco.json_output: true`.
- Network access from the Falco or Falcosidekick pods to the operator's Service on the integrations port, 8444 by default.
- An admin account. Only admins see the integration page, download certificates and edit policies.
- For a policy that stops pods: `rbac.allowEnforce: true` in the Attestkeep chart.
- For messages and issues: a notification channel, or a GitHub, GitLab or Jira channel, subscribed to Runtime alert.

Connecting Falco

Turn the integration on in the Attestkeep chart and choose how Falco connects:

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

With `mtls`, the default, Falco posts each event from its own `http_output` straight to the operator. Use `sidekick` if your Falco already sends to Falcosidekick. Sidekick's webhook output then forwards to the operator with the same client certificate. Any other value makes the chart fail.

When the integration is on, the operator creates its own small CA and one client certificate (`CN=falco`, valid one year) and keeps them in the Secret `<release>-falco-mtls`. The integrations port serves one route, `POST /api/v1/integrations/falco`, and accepts only a client certificate signed by that CA. There is no token.

Falco integration

Falco's events reach Attestkeep over mutual TLS, from Falco itself or through Falcosidekick.

Status	On
Connection	Falcosidekick (mTLS)
Endpoint	https://attestkeep.attestkeep.svc:8444/api/v1/integrations/falco Copy
Client certificate expires	1 Oct 2027, 18:47
Last event	3 Oct 2026, 13:11 via Falcosidekick
Download certificates ca.crt client.crt Rotate	

The bundle holds ca.crt, client.crt and client.key. Keep the key private.

1. CREATE THE SECRET IN THE FALCO NAMESPACE

```
tar xzf attestkeep-falco-mtls.tar.gz
kubectl -n falco create secret generic attestkeep-falco-client \
  --from-file=ca.crt --from-file=client.crt --from-file=client.key
```

[Copy](#)

2. FALCO CHART VALUES

Falco > Falco integration on the installation used for this handbook. Status is On, the connection is Falcosidekick over mTLS, and the last event arrived via Falcosidekick. *Download certificates* saves a bundle with `ca.crt`, `client.crt` and `client.key`. Below it, the page gives the command that creates the Secret in the Falco namespace, followed by the Falco chart values for the chosen connection.

Every bundle download is written to the audit log. `client.key` must be kept private, because anyone who holds it can post events that your policies act on. *Rotate* issues a new client certificate from the same CA. The old one keeps working until it expires, so events keep arriving while you replace the Secret. When the operator's own server certificate is renewed, the page shows "Certificates changed on <date>" until you download the bundle again.

Runtime policies

Falco > Policies lists your runtime policies in order. Each stored event is checked against them from the top, and the first enabled policy that matches decides. The arrows change the order. A change reaches the replica that saved it at once and the other replicas within 15 seconds.

Policies

[New policy](#)

Checked top to bottom. The first policy that matches an event decides what happens. No match means record.

Falco connected · last event 15 Oct 2026, 03:46

1 **gozlem-kaydet** Record

Terminal shell in container · ns: gozlem

Events ☒ ↑ ↓ [Edit](#) [Delete](#)

2 **uretim-durdur** Stop the pod

Terminal shell in container · ns: uretim

Events ☒ ↑ ↓ [Edit](#) [Delete](#)

Two policies on the 1.3.7 installation. Each row shows what the policy does, the Falco rule it matches and the namespace it is limited to: `gozlem-kaydet` records, `uretim-durdur` stops the pod. From 1.3.5 a policy is labelled by its effect (Stop the pod, Open an issue, Notify, Record or Allow) here, in Events and in the evidence package.

Edit policy

Name *

Stop pods when someone opens a shell

WHICH EVENTS

Falco rules

critical

warning

notice

informational

Select all (28)

Terminal shell in container

notice

Search the catalogue or type a

Any of them. Pick from the catalogue, or type your own rule's exact name.

WHERE

All namespaces

Only in namespaces

ak-e2e

Type and press Enter

WHAT TO DO

☒ Send a notification

Sends the event to your notification channels. Nothing in the cluster changes.

☒ Open an issue

Opens an issue in your connected tracker for each new case. Nothing in the cluster changes.

☒ Stop the pod

☒ Kill at once (recommended)

The pod is evicted with a one-second grace period, so its containers are killed a second later. Whatever was started inside, a shell session included, ends there.

☐ Use the application's shutdown time

The pod gets its own termination grace period (30 s by default) to shut down cleanly. A session opened inside keeps running until then.

Pods will be deleted without asking. When Terminal shell in container fires, in ak-e2e, the pod it came from is deleted as soon as the event arrives. Its controller starts a fresh pod; a pod with no controller is gone for good. If the cause is in the image, the new pod can trigger the event again and be deleted again. Try the policy with only notifications first.

Cancel

Save

The policy form. This policy matches the Falco rule *Terminal shell in container* in one namespace, with *Send a notification*, *Open an issue* and *Stop the pod* switched on. Under *Stop the pod* you choose *Kill at once (recommended)* or *Use the application's shutdown time*. The red box warns that matching pods will be stopped without asking.

Field	What it does
Name	Required, at most 128 characters, unique.
Falco rules	The policy matches events from any of these rules, by exact name. Pick from the Rules list or type your own rule's name. At least one is required.
All namespaces / Only in namespaces	Limits the policy to events from pods in the listed namespaces.
Send a notification	Sends the event to every enabled channel subscribed to Runtime alert.
Open an issue	Opens an issue through the tracker channels subscribed to Runtime alert.
Stop the pod	Stops the pod the event came from, with <i>Kill at once</i> or <i>Use the application's shutdown time</i> .
State	On or Off. A new policy starts Off. An Off policy is kept and does nothing.

With any of the three actions on, the policy is saved with action `notify`. With all three off, it is saved as `record`: matching events are recorded against the policy and nothing else happens. The API (`/api/v1/runtime/rules`) accepts more conditions than the form shows: tags, priority, source, output fields, the image's last verdict and signature, and a window of `count` events within `duration`. It also accepts the action `allow`, which keeps the event and does nothing else. If a policy has settings the form does not show, the form says so and keeps them when you save.

Stopping the pod

A policy with *Stop the pod* on stops the pod named in the event as soon as the event arrives, or as soon as the window is reached if the policy has one. The operator asks the API server to evict the pod, so PodDisruptionBudgets apply. If a budget does not allow the eviction, the pod keeps running and the audit log records why.

15 October 2026

Page 33 of 52

- *Kill at once* (`stop_mode: immediate` , the default): the eviction uses a one-second grace period. Anything started inside the containers, an open shell included, ends there.
- *Use the application's shutdown time* (`stop_mode: graceful`): the pod gets its own termination grace period, 30 seconds unless the pod sets another value. A shell opened inside keeps running until then.

The pod's controller starts a new pod. A pod without a controller does not come back. If the cause is in the image, the new pod can trigger the same event and be stopped again.

- Stopping needs `rbac.allowEnforce: true` . Without it the attempt is refused and recorded.
- Pods in the operator's own namespace are never stopped. No other namespace is excluded, so use *Only in namespaces* to keep a stop policy away from system namespaces.
- The event must name the pod (`k8s.ns.name` and `k8s.pod.name`). With the Falco chart's defaults it does.
- A pod that is already terminating is left alone. Each operator replica asks to stop a given pod at most once every 10 minutes.

Every attempt is written to the audit log as `runtime_pod_stop` , with `policy:<name>` as the actor, the pod as the target, and the Falco rule, pod UID, container, image, digest, node, event id and result in the detail. Stopping a pod does not change admission. New pods of the same image are admitted as before.

Notifications and issues

A message is titled `Runtime alert: <Falco rule>` and carries Falco's output line, the rule, priority, source, namespace, pod, container, image, digest, node, time, the policy and its action. If the policy stopped the pod, the message says so. A policy sends at most one message per image digest per hour. That limit applies to messages and issues. It does not delay stopping a pod.

An issue is opened only when the event is tied to an image digest: one issue per digest and Falco rule, titled `RUNTIME: <rule> in <image>` and labelled `runtime` . It closes by itself as `quiet` after 7 days without that rule firing on that image, or as `image_retired` when no running pod has used the digest for 24 hours. Neither means the problem was fixed. An event counts as notified only when at least one enabled channel is subscribed to receive it.

Events

Events

Refresh

Everything your Falco sent, and what each event led to.

Falco connected · last event 3 Oct 2026, 13:11

Policy

Stop pods when someone of

Action

AllowRecordNotify

Search rule, pod, container or output

1h24h7dCustom

Priority

emergencyalertcriticalerrorwarningnoticeinformationaldebug

Namespace

Pod

Falco rule

Tag

Any

Choose a namespace first

Any

Any

Matching events: 11

Clear filters

Time	Policy	Action	Priority	Falco rule	Namespace / pod
3 Oct 2026, 00:25	Stop pods when someone opens a shell	Notify Sent	notice	Terminal shell in container	ak-e2e / web-7b98857f77-v4jpz docker.io/library/busybox:1.36
2 Oct 2026, 23:28	Stop pods when someone opens a shell	Notify Sent	notice	Terminal shell in container	ak-e2e / web-nginx-79766bdbfb-bn7b5 docker.io/library/nginx:1.27-alpine
2 Oct 2026, 23:27	Stop pods when someone opens a shell	Notify	notice	Terminal shell in container	ak-e2e / web-nginx-79766bdbfb-ml66m docker.io/library/nginx:1.27-alpine
2 Oct 2026, 23:17	Stop pods when someone opens a shell	Notify Sent	notice	Terminal shell in container	ak-e2e / web-7b98857f77-wvjdg docker.io/library/busybox:1.36
2 Oct 2026, 23:16	Stop pods when someone opens a shell	Notify	notice	Terminal shell in container	ak-e2e / web-7b98857f77-zbrdb docker.io/library/busybox:1.36
2 Oct 2026, 23:04	Stop pods when someone opens a shell	Notify	notice	Terminal shell in container	ak-e2e / web-7b98857f77-cv7zk

Falco > Events filtered to one policy over the last 7 days: 11 events from *Terminal shell in container*, each with the policy that matched, the action and whether a message was sent. Filters cover free text, policy, action, priority, namespace, pod, Falco rule and tag, over a time range. From 1.3.6 the action filter offers the five effects a policy can have: Allow, Record, Notify, Open an issue and Stop the pod. Results come a page at a time.

Terminal shell in container

Close

3 Oct 2026, 00:25

DECISION

Notify Stop pods when someone opens a shell

Sent 3 Oct 2026, 00:25

Policy hash 62c8e48ab453

Filter by this pod

Filter by this rule

Filter by this policy

OUTPUT

21:25:35.538967447: Notice A shell was spawned in a container with an attached terminal | evt_type=execve user=root user_uid=0 user_loginuid=-1 process=sh proc_exeppath=/bin/sh parent=containerd-shim command=sh -c i=0; while [\$i -lt 50]; do echo tick \$i; i=\$((i+1)); sleep 1; done terminal=34816 exe_flags=EXE_WRITABLE|EXE_LOWER_LAYER container_id=c0fad8bc8637 container_name=k8s_busybox_web-7b98857f77-v4jpz_ak-e2e_99051f1c-3e83-4ba8-b95b-0f390a6a9428_0 container_image_repository=busybox container_image_tag=1.36 k8s_pod_name=web-7b98857f77-v4jpz k8s_ns_name=ak-e2e

TAGS

T1059containermaturity_stablemitre_executionshell

DETAILS

Priority

notice

Source

syscall

Namespace / pod

ak-e2e / web-7b98857f77-v4jpz

Container

k8s_busybox_web-7b98857f77-v4jpz_ak-e2e_99051f1c-3e83-4ba8-b95b-0f390a6a9428_0

Image

docker.io/library/busybox:1.36

Digest

sha256:73aaf090f3d85aa34ee199857f03fa3a95c8ede2ffd4cc2c5b94e566b11662

One event opened. The panel shows the decision and the policy that made it, when the message was sent, the policy hash, Falco's output line, tags, and the namespace, pod, container, image and digest the event was filed against. The policy name and hash are stored with each decision, so renaming or deleting a policy later does not change what past events show.

Falco rules and the rule editor (beta)

Falco > Rules lists the Falco rules the operator knows: the 25 stable rules from `falcosecurity/rules` that ship with it, the rules in your saved rules files, and rule names seen in events. *Use in policy* starts a policy from a rule.

Rules New rule

The Falco rules this operator knows: the stable list it ships with, the rules in your saved rules files, and rules seen in your events.

Falco connected · last event 3 Oct 2026, 13:11

Rules files e2e-rules v2

Search name or description Any priority All Built in Custom Seen in events

Falco rule	Priority	Description	Events	Origin	
Clear Log Activities maturity_stable host container filesystem mitre_defense_evasion T1070 NIST_800-53_AU-10	warning	Detect clearing of critical access log files, typically done to erase evidence that could be attributed to an...	6	falco-rules-5.2.0	Use in policy
Contact K8S API Server From Container maturity_stable container network k8s mitre_discovery T1565	notice	Detect attempts to communicate with the K8S API Server from a container by non-profiled users. Kubernetes APIs...	237	falco-rules-5.2.0	Use in policy
Create Hardlink Over Sensitive Files maturity_stable host container filesystem mitre_credential_access T1555	warning	Detect hardlink created over a curated list of sensitive files or subdirectories under /etc/ or root directories. Can...	3	falco-rules-5.2.0	Use in policy
Create Symlink Over Sensitive Files maturity_stable host container filesystem mitre_credential_access T1555	warning	Detect symlinks created over a curated list of sensitive files or subdirectories under /etc/ or root directories. Can...	0	falco-rules-5.2.0	Use in policy
Debugfs Launched In Privileged Container maturity_stable container cis process mitre_privilege_escalation T1611	warning	Detect file system debugger debugfs launched inside a privileged container which might lead to container...	0	falco-rules-5.2.0	Use in policy
Detect release_agent File Container Escapes maturity_stable container process mitre_privilege_escalation T1611	critical	Detect an attempt to exploit a container escape using release_agent file. By running a container with certain...	0	falco-rules-5.2.0	Use in policy

Falco > Rules with the *Built in* filter. Each rule shows its priority, tags, description, how many events it has produced here, and its origin, `falco-rules-5.2.0`. The *Rules files* line lists the saved rules files, here one file at version 2.

New rule (admins) opens an editor for a Falco rules file: rules, macros, lists and overrides in Falco's own YAML. The editor is in beta. Completion and checks follow the Falco version running in your cluster, found by the label `app.kubernetes.io/name=falco`.

- *Validate with Falco* starts a one-off pod in the operator's namespace with the same image your Falco runs. It loads Falco's default rules and your file, runs Falco in validation mode, and is deleted once the result is read. Your running Falco is not touched.
- *Save* is enabled only after Falco has accepted exactly the text in the editor. The server validates it again before storing it. Each save is a new version, recorded with who saved it and which Falco version accepted it. A rule name already defined by another saved file is refused.
- *Publish* pushes a saved version to your own OCI registry as a `falcoctl rulesfile` artifact, tagged with the version number and `latest`. If the registry needs a login, you give the name of a `kubernetes.io/dockerconfigjson` Secret in the operator's namespace. Attestkeep reads it when it pushes and does not store the credential.

← Rules

Rules file e2e-rules BETA v2

Write a Falco rules file. Validate it with your cluster's Falco, save it as a new version, then publish it to Falco.

Falco in this cluster: Falco 0.45.0-3 pods

Rules file name *

e2e-rules

Lower case, digits, dots and dashes. It is how the file is kept and published.

Falco rules *

```

1 - rule: AK e2e marker exec
2   desc: e2e marker for the rule editor publish test.
3   condition: >
4     spawned_process and container and
5     proc.cmdline contains "ak-e2e-marker-v1"
6   output: >
7     ak e2e marker (cmd=%proc.cmdline ns=%k8s.ns.name pod=%k8s.pod.name
8     container_id=%container.id image=%container.image.repository)
9   priority: WARNING
10  tags: [ak e2e]
11 - rule: AK e2e marker two
12   desc: second rule, added in v2.
13   condition: spawned_process and container and proc.cmdline contains "ak-e2e-second"
14   output: ak e2e second (cmd=%proc.cmdline pod=%k8s.pod.name)
15   priority: NOTICE
16

```

Rules, macros and lists in Falco's own format. Ctrl+Space suggests what fits where the cursor is.

Red: certain mistake. Yellow: a name this editor does not know. A light bulb offers a fix when there is exactly one.

Validate with Falco Save as v3 No changes to save. Delete

Saved by E2E Owner, 2 Oct 2026, 20:08. Falco 0.45.0 accepted it.

Publish to Falco

The file is pushed to your OCI registry; your Falco picks it up through falcoctl. Attestkeep never writes to Falco.

A saved rules file at version 2. The line under the title shows the Falco found in the cluster, Falco 0.45.0 on 3 pods. Save as v3 stays disabled until the text changes and Falco accepts it. The last line says who saved version 2 and which Falco version accepted it.

Publish to Falco

The file is pushed to your OCI registry; your Falco picks it up through falcoctl. Attestkeep never writes to Falco.

Repository *

registry.registry-test.svc:5000/falco/e2e-rules

Registry and path, no tag. Each version is pushed as its number and as latest.

Push Secret

falco-rules-push

A kubernetes.io/dockerconfigjson Secret with the registry login, in the namespace Attestkeep runs in. Empty for a registry without login.

Publish v2

v2 published to registry.registry-test.svc:5000/falco/e2e-rules:2 by E2E Owner, 2 Oct 2026, 20:33.
sha256:5b704416b24e77aada5f54c31fbd4f7afc84e5c38900af8288beee02646fbc20

▶ How your Falco picks it up

Publish to Falco: the repository, the push Secret's name, and the record of version 2 being published with its digest. *How your Falco picks it up* opens the falcoctl values for your Falco chart.

Your Falco loads the file through falcoctl. In the Falco chart values, add your reference to both `install.refs` and `follow.refs`, and keep `falco-rules:5`, the chart's default reference. Helm replaces lists instead of merging them, so leaving it out stops falcoctl from installing `/etc/falco/falco_rules.yaml`. Then add your file to `rules_files` after Falco's own rules. Deleting a file in the console removes its rules from the Rules list. What was already published stays in the registry and in every Falco that installed it until you remove the reference from falcoctl.

Retention and metrics

Every event is stored. Attestkeep has no priority threshold of its own. Stored events older than `runtime.retentionDays` (90 by default) are removed every hour, and `0` keeps them indefinitely. Issue history, audit entries and policy revisions are not removed. To stop Falco from sending events you never want to keep, set Falco's own `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`, `rule` is the runtime policy's name. `unresolved_image` does not mean the event was lost. Falco also watches processes on the host, and such an event is stored without an image.

8. Prove it is enforcing

An admission webhook that is not running looks the same as a compliant cluster. This check takes about five minutes, uses only `kubectl`, and ends with a denial you have seen yourself. Run it after installing, after upgrades, and whenever you need to confirm enforcement.

1. Try a pod the policy should refuse

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

The API server refuses it and quotes the webhook. This is the output from an installation running the default policy:

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

The message starts with *Error from server*, so the refusal came from the API server's admission chain. The reason names the policy rule, so a developer reading it in a CI log knows what to change.

2. Confirm nothing was created

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

3. Check that the decision was recorded

The dashboard's *Recently blocked admissions* table now starts with this denial and the same reason. The metric `attestkeep_admission_decisions_total{decision="denied"}` has increased.

4. If the pod was admitted

- The namespace is bypassed: `kube-system`, `cert-manager`, the operator's own namespace, or a namespace labelled `attestkeep.com/webhook=bypass`.
- The policy does not cover what you tried. Check `enforcement: Enforce` and `blockLatestTag: true`.
- The webhook is failing open. Check that `attestkeep_webhook_distrusted` is 0 and that the pods are ready.
- The installation is not activated. Until a key is entered nothing is enforced, and every admission response says so in a warning.

With deny-until-scanned

Under `admissionTiming: DenyUntilScanned` a new image is refused until its scan exists. Output from a 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

# a short while later, the same command:
pod/cold-check created
```

9. Evidence packages and the trust model

An evidence package is one signed document per audit period. It is generated inside the cluster from your own records, and an auditor can verify it with their own tools. It is evidence for an audit, not a certification, and the document says so.

Control sets it reports against

Instrument	What is mapped
SOC 2, AICPA Trust Services Criteria	13 criteria across CC6 to CC9 and Availability A1.
EU CRA and NIS2: (EU) 2024/2847 and Article 21(2)	34 requirements across the two instruments.
DORA, Regulation (EU) 2022/2554	22 requirements from Articles 8 to 11 and Article 28.
NIST SP 800-190	15 countermeasures for images, registries, orchestrators and containers.
SSDF, NIST SP 800-218 v1.1	13 practices from PO, PS, PW and RV.
NIST SP 800-53 Rev. 5	15 selected CM, RA, SI, SR, AU and CP controls.
ISO/IEC 27001:2022 Annex A	12 clauses: 10 Annex A controls plus clauses marked outside scope.
GDPR Article 32	6 clauses on security of processing.
PCI DSS v4.0.1 (PCI SSC, June 2024)	13 clauses from Requirements 6.3, 6.5, 10.2 and 11.3.
MAS Technology Risk Management Guidelines (18 January 2021)	30 clauses from sections 3.3, 6.1, 6.3, 7.2 to 7.6, 11.3, 11.4, 12.1, 12.2, 13.1, 13.6 and 15.
ASD Essential Eight Maturity Model (November 2023)	46 clauses: patch applications, patch operating systems and application control, Maturity Levels One to Three.
ISMAP Control Criteria (Japanese edition of 1 August 2024)	42 clauses from the control criteria of attached table 3.
ISMS-P Certification Criteria (PIPC and MSIT notification Annex 7, KISA guide of November 2023)	32 clauses from domains 1 and 2.

Thirteen instruments and 293 clauses in all, each with its own citation. The mapping ships inside the operator, so an air-gapped cluster reports the same way as a connected one. Each control ends in one of four states: *met*, *partial*, *gap* or *outside_scope*. A clause is met only when the period's records show the control working, never just because the product has the feature. A clause a cluster cannot answer is reported as outside scope with the reason, and is included in the count above.

How many frameworks you can generate packages for depends on the licence (chapter 12).

Translated sources. Two mappings are read from texts not published in English, and each clause says so. For **ISMAP**, an English reference translation exists, but where its entry only points to ISO/IEC 27002, the obligation is translated from the Japanese edition of 1 August 2024 and the citation says which. For **ISMS-P** there is no official English text. Each clause is our translation of the Korean criterion as printed in KISA's guide. Each ISMS-P criterion combines a technical measure with a procedure, a designation or a periodic review, and a cluster's records answer only the technical part, so no ISMS-P criterion is reported as fully met.

Five attestations feed the newer mappings. `patch-window-48-hours` is met when the declared window for critical findings is at most 48 hours and known-exploited vulnerabilities are refused at admission. `patch-window-two-weeks` is met when the declared window for high findings is at most two weeks. `runtime-observed` is met when the Falco integration is on and at least one event arrived in the period. It is partial when the integration is on and nothing arrived,

because a quiet cluster and a Falco that stopped sending look the same from the operator. It is a gap when the integration is off. `producer-statement-honoured` is met when the policy applies VEX statements and trusts at least one issuer. `context-aware-thresholds` is met when escalation is on for both `exposed` and `privileged`, partial with one of the two, and a gap under `Ignore`. `remediation-windows` can reach *met* only with an issue tracker connected, because only then is there a record of when a fix landed.

How each requirement was answered

Each requirement lists the tests that answered it. A test has a method (`TEST` when it reads a record of what happened, `EXAMINE` when it reads a configuration), a result, an observation, paths into the report, the source it read and when. Each requirement also states `result` and `implementation_status` in OSCAL terms. The PDF prints the tests under each requirement.

The `generation` block says when and by what the document was generated, the source system, the query window (end excluded, UTC), each section's query with its row count and whether a capped query hit its cap, and how people are named. From 1.3.0 it also carries `nodes {count, ceiling}`: the cluster's Ready nodes at generation and the licence's node ceiling, numbers only. `admission.records[]` carries every admission review of the period, one row each and with no cap, so an auditor can sample from the whole population. `organisation.declarations` carries the operator's time source and retention statements, printed as declared by the operator.

People are named by account, not by name. `generated_by`, `chain[].actor` for audit-log rows, `triage[].triaged_by` and `runtime.audit_entries[].actor` hold `user:<account id>`. A pod stopped by a runtime policy is written as `policy:<name>`. The console's audit log still shows names, and the operator uses it to resolve an id when an auditor asks. Rows recorded before 1.2.0 that hold only a name are matched to one account by exact display name or e-mail, or written as `user:unresolved` and counted. The Kubernetes identity behind each admission is written as given when it is a `system: identity`, and otherwise as `k8s-user:` followed by the first 16 hex characters of its SHA-256.

The runtime section carries the event count by priority, the busiest Falco rules, alerts sent, and runtime issues opened and closed. It also carries the runtime policy set in force with its revision hash, `audit_entries` for policy changes, pods stopped and integration changes in the period, match counts by policy and action, the events a policy acted on (up to 500, each with the issue it opened), and coverage: whether the integration is on, the last event and the longest gap without events. `event_time` is Falco's own timestamp. `received_at` is when the operator received the event. Coverage uses `received_at`, and `coverage.clock` says so. `runtime.rules[].effect` and `runtime.decisions[].policy_effect` give the action the policy was set to take. Whether a stop went through is the `runtime_pod_stop` row in `runtime.audit_entries`.

Break-glass and triage. From 1.3.4 each break-glass activation is read from the policy revision its admissions were decided under, so a package says what was approved at the time even after the window was switched off. Regenerate any package that covers a past break-glass window to get the full record. Triage decisions carry the severity from the scan findings, the highest where a CVE was recorded at more than one.

Ledger integrity

Every admission decision is written to PostgreSQL. A database is a good place to query decisions but not, on its own, a reason to trust them. The ledger is built so that tampering shows:

- **Every record carries its own hash**, a SHA-256 over its content (time, namespace, pod, image, digest, verdict, reason, policy name and hash, break-glass flag), computed when it is written.
- **Records are sealed under signed checkpoints**, hourly by default or every 10,000 records. The leader replica builds a Merkle root over the unsealed range and signs a checkpoint that includes the previous checkpoint's hash, so the seals form a chain.
- **The signing key is not in the database.** It is an Ed25519 key generated in your cluster and kept in the `attestkeep-evidence-key` Secret. Rewriting ledger rows needs database write access. Re-signing the seals needs access to that Secret. One of the two is not enough.

- **The same key signs your evidence documents** as DSSE envelopes, and the licence certificate names your key, so a third party can tie a document to your installation.

10 Ledger integrity

Whether the admission ledger itself can still be believed: every sealed record re-hashed, every Merkle seal recomputed, the seal chain walked and each seal's signature checked, at the moment this document was generated.

Intact

Ledger

13

Signed seals

432

Records re-verified

0

Not yet sealed

Every sealed record reproduced its hash, every seal reproduced its root, the chain is unbroken and every signature verifies. The unsealed tail is the window since the newest seal; the next seal will cover it.

The ledger check inside a package. At generation every sealed record was hashed again, every Merkle seal recomputed, the chain walked and every signature checked. Here: *Intact*, 13 signed seals, 432 records re-verified, 0 not yet sealed.

Where the guarantees end

This scheme makes tampering visible. It does not prevent it. Records written since the newest checkpoint are counted but not yet sealed, a window of at most one seal interval. *Intact* speaks about what is sealed, so read it with the checkpoint count. An attacker with database write access and access to the signing key can defeat the scheme. The claim is that this takes two separate accesses. The auditor receives a signed result and the DSSE envelope. Re-running the Merkle check independently would need database access.

Signing a package for an auditor produces a PDF alongside the JSON, delivered together in one signed archive. Anyone can check a delivered package at attestkeep.com/verify.html with their own tools.

10. Operations, alerting and backup

Probes

Endpoint	What it checks	Used as
<code>/healthz</code>	Only that the process is up.	Liveness probe (initial delay 10s, period 20s)
<code>/readyz</code>	The database, with a 3-second timeout. If it is unreachable, the probe returns 503 and the replica leaves the Service, because a replica that cannot read the verdict cache would admit everything.	Readiness probe (initial delay 5s, period 10s)

What a database outage does

- Replicas become unready and leave the Service. They do not crash-loop, and they come back when the database does.
- With the default `failurePolicy: Ignore`, the API server admits pods unchecked during the outage. Afterwards the operator finds the heartbeat gap and records it as an outage, and the next reconciliation sweep lists every digest that came in without a decision.
- Quota metering admits on a database error rather than blocking deployments.
- Scanning stops with the database and resumes with it. The queue is in the database, so nothing is lost.

What should page you

Metrics are served on port 9090 at `/metrics`. Every labelled series exists from startup at zero, so missing data and zero denials are different things.

Condition	Signal	Why it matters
Admission is being bypassed	<code>attestkeep_webhook_distrusted = 1</code>	While this is 1 under <code>failurePolicy: Ignore</code> , deployments go through unchecked. Treat it as a page.
Something got in	<code>attestkeep_runtime_unmatched_digests > 0</code>	The reconciliation sweep found running digests with no allowed admission behind them.
Database trouble	Pods unready, <code>/readyz</code> returns 503	Your existing <code>kube_pod_status_ready</code> alerts cover it.
Scanning is falling behind	<code>attestkeep_scan_queue_depth</code> staying high	A queue that only grows means the scanner cannot reach a registry or its database.
Vulnerability data getting old	<code>attestkeep_vulnerability_db_age_seconds</code>	Verdicts are only as current as the database behind them.
Image quota close to the limit	<code>attestkeep_image_quota{kind="count"}</code> against <code>{kind="limit"}</code>	At the limit only image names not seen this period are denied. Running workloads keep running.
Over the node ceiling	<code>attestkeep_nodes{kind="count"}</code> against <code>{kind="ceiling"}</code> , <code>attestkeep_node_overage_seconds > 0</code>	The overage gauge is how long the cluster has been over its ceiling. Alert well before 14 days, when new image names start being denied.

Denials are counted in `attestkeep_admission_denials_total{reason}`. Reasons include `threshold`, `signature`, `attestation`, `tag`, `digest`, `registry`, `quota`, `unscanned`, `scan_error`, `end_of_life`, `known_exploited`, `provenance` and `context`. Node-ceiling denials count under `quota`. Other useful series for a 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`, and the runtime series listed in chapter 7.

Notification events cover the same ground: `threshold_80` and `threshold_100` for the image quota, `nodes_80`, `nodes_over`, `nodes_enforced` and `nodes_cleared` for the node ceiling, `license_expiring` when an online certificate has less than a week left, `trivy_db_failed`, `unverified_workload` and `runtime_alert`. A new channel has `nodes_over`, `nodes_enforced` and Runtime alert ticked.

What to back up

Attestkeep does not take database backups. Running, scheduling and keeping the dump is your job. Everything the operator knows is in PostgreSQL and in cluster resources. The database is the backup that matters: findings, decisions, exceptions, the audit log, the evidence archive, users and the licence certificate are all in it.

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

If you use your own PostgreSQL, your existing backups already cover it. A volume snapshot of the bundled database's PVC also works if your storage class supports it. Next to the database, back up three Secrets:

- `attestkeep-evidence-key`, the evidence signing key, created by the operator. If it is lost, a new key is generated, and checkpoints and documents signed with the old key no longer verify against the installation's current key. The history stays.

- `attestkeep-data-key` , generated by the chart unless you supplied `api.dataKey.existingSecret` . If it is lost, the stored tracker tokens cannot be decrypted and each tracker channel needs its token entered again.
- `attestkeep-session` , the session key. If it is lost, everyone signs in again.

The webhook serving certificate is not on the list, because the operator reissues it. The activation key is not either: reinstalling into the same cluster gives the same fingerprint and reuses the existing licence slot. With the Falco integration on, the operator also creates `<release>-falco-mtls` . Its client certificate is rotated from Falco > Falco integration.

Settings > Backup also exports a JSON bundle of the configuration you built by hand: notification channels, triage decisions, the user list and policies. It is for setting up a second cluster the same way. Credentials are never in the bundle. Import shows a preview first, existing records always win, and users are listed, not created. From 1.3.1 every export and import is listed under *Backup history*. The bundle does not replace the database dump.

A restore does not hide what was lost. On the first pass after a restore, the operator finds the heartbeat gap since the dump and records the whole span as a period in which the control was not operating. An evidence package for that period shows the gap like any other outage. Restoring a dump into the same or a newer operator version is supported. Restoring a newer dump into an older operator is not.

Availability

Operator and webhook: 2 replicas, preferred anti-affinity, PodDisruptionBudget minAvailable 1. Raise `replicaCount` to scale out. Controller work is serialised through an advisory lock, so this is safe. Scanner: 1 replica. The chart refuses several replicas on a ReadWriteOnce cache volume and says why. Bundled PostgreSQL: one replica by design. For replication and managed backups, point `externalDsn` at your own PostgreSQL.

GitOps. Enforcement happens at pod admission, so it does not matter whether kubectl, Argo CD, Flux or a Job applies the change. If a GitOps sync creates a pod whose image fails policy, the sync fails with the denial message as the error.

Air-gapped clusters. Trivy and cosign are built into the image. Mirror the vulnerability database with `oras copy ghcr.io/aquasecurity/trivy-db:2 registry.internal/mirror/trivy-db:2` and set `trivy.dbRepository` . Set `scan.cosignOffline` to verify against keys without the public transparency log. A mirrored KEV catalogue can be gzipped into a ConfigMap named by `kev.existingConfigMap` , with `kev.source` pointing at `file:///etc/kev/<key>` . Licensing needs either one route out to `lic.attestkeep.com:443` or offline activation (chapter 11).

11. Licensing

One activation key per purchase, and one Community key per account. A key binds to as many clusters as its plan allows, and you can move a slot yourself.

Activation

Activation happens once per cluster and uses one slot. On first run the operator generates a key pair inside the cluster, keeps the private key in its own database, and sends only the public key with the activation request. The request carries the activation key, a cluster fingerprint (a hash of the kube-system namespace UID, not a name or address), the cluster name if you set one, and the operator version. The answer is a signed certificate valid for at most thirty days. The operator verifies it inside the cluster against a public key built into the binary.

From 1.3.0 the operator accepts only a certificate that carries the node ceiling. A certificate issued by an older licence service is refused with "re-activate with a key issued by licence service 1.3.0 or later".

The daily check

Once a day the installation sends three values: licence id, cluster fingerprint and operator version, signed with the key pair from activation. The answer is a status, plus a new certificate when something changed or the current one is running out. Nothing about your images, workloads, policies or findings is sent. There is no telemetry channel and no setting that adds one. However many operator pods run, a cluster makes this call once a day.

When the licence service cannot be reached

The installation keeps running on the certificate it holds until that certificate expires. A failed check is retried after four hours. When an online certificate has less than a week left, the console warns and the `license_expiring` notification is sent. If the certificate runs out, the installation carries on as Community Edition. It does not stop and does not start denying deployments. A `revoked` or `released` answer applies at the next check.

Cluster slots

Each activation holds a slot until it is released. You release slots yourself from your licences page at attestkeep.com. A released slot is free at once, and the installation that held it drops out at its next daily check. Reinstalling into the same cluster does not use a second slot, because the fingerprint is the same. If you run out, the error names the clusters holding your slots and when each was activated.

The console can hand a licence back too: *Deactivate* on the Usage and License page, after a confirmation. On the online path this frees the cluster's slot. On the offline path it deletes the certificate from the cluster and leaves the slot for Attestkeep support to release, because an offline cluster never reports back.

Offline activation

A cluster that will never reach the licence service is activated with a certificate instead of a key. The activation screen and the Usage and License page both have a *Certificate* tab for this.

Attestkeep

Activate this installation

Every installation is licensed, the free edition included. Use a key if this cluster can reach attestkeep.com; use a certificate if it cannot.

Activation key **Certificate**

- Download the activation request**
A small file naming this cluster: its fingerprint, this operator's public signing key, the operator version and the cluster name. Nothing else leaves the cluster.
`758bafbac147... · 1.3.2 · dev-mn3`
Download
- Get a certificate for it**
Upload the request on your licence page at attestkeep.com/app (Licences → your licence → Offline activation), on a machine that has internet. The certificate you download there is valid only for this cluster.
- Upload the certificate**
Choose File No file chosen
Activate

No key yet? Create a free account at attestkeep.com

The *Certificate* tab in three steps. Download the activation request, which names this cluster by its fingerprint, the operator's public signing key, the operator version and the cluster name. Upload it on your licence page at attestkeep.com from a machine with internet access. Upload the certificate you get back.

The certificate is the same signed document the online path stores, marked as offline. The operator checks the signature, the cluster fingerprint and its own key before storing it. It runs no daily check for it and warns thirty days before it expires. A certificate for another cluster or key, or an expired one, is refused with the reason. For an installation whose console nobody opens, `license.existingCertificateSecret` names a Secret holding the certificate under the key `certificate`.

Whether offline activation is available for your licence is shown on the licence page at attestkeep.com. On the published price list it comes with Enterprise on annual terms, or under an agreement.

The node ceiling

Each plan allows a number of Ready nodes per cluster. The operator counts the cluster's Ready nodes every `nodes.pollInterval` (5 minutes by default). The count stays in the cluster. The daily check still sends the same three values. The ceiling itself is in the certificate.

Going over the ceiling stops nothing. The console and a notification say so, with a date 14 days ahead (`nodes.graceDays`). If the cluster is still over when those 14 days have passed, image names not yet seen this period are denied at admission with "node ceiling of N exceeded", and a denied name is not counted. Everything already counted keeps running and keeps being scanned and enforced. Coming back within the ceiling, with fewer nodes or more nodes on the licence, clears the clock at once. A poll that cannot count the nodes changes nothing. Neither chart value loosens the ceiling.

When a licence ends

The installation carries on as Community Edition when the subscription ends and the certificate runs out, when the certificate stops verifying or turns out to belong to another cluster, or when you deactivate it. Community means 1 cluster, 1 framework, 10 image names a month and 1 node per cluster. Nothing is downloaded and no new certificate is issued. Scanning, admission and evidence keep working. From that moment the Usage and License page and the dashboard show the Community allowance for the current month, the same numbers admission applies. Earlier months keep the

allowance they had. Cancelling mid-term does not switch anything off on the day you cancel. You keep what you paid for until the term ends.

Choosing frameworks. If evidence has been generated for more frameworks than the licence now covers, the next owner or administrator to sign in is asked which to keep, up to the licence's count. Until someone chooses, none of the frameworks already used is open for a new package, and other users are told an administrator has to choose. The frameworks not kept are closed for new packages. Every package already generated stays downloadable. The choice holds until the licence's plan or framework count changes, and it is written to the audit log.

12. Editions and prices

Every security feature is in the free edition. Scanning, signature verification, admission enforcement and evidence packages are complete in Community, and every policy control, including deny until scanned and `failurePolicy: Fail`, works there. Paid plans add capacity and single sign-on.

Edition	Per month, excl. VAT	Per year, excl. VAT	Clusters	Frameworks	Image names / month, per cluster	Nodes / cluster	Also
Community	\$0	\$0	1	1, your choice	10	1	Public issue tracker
Team	\$600	\$6,000	1	1	50	6	Single sign-on, e-mail support
Business	\$1,800	\$18,000	3	3	75	10	Everything in Team, priority e-mail support
Enterprise	\$3,600	\$36,000	5	All thirteen	100	20	Offline activation for air-gapped clusters, e-mail support

A year costs ten months. Add-ons on paid editions, per month: additional cluster +\$300 (up to ten clusters in one checkout), additional framework +\$120, 10 more image names a month +\$30 for each cluster the licence covers, additional node +\$50 for each cluster the licence covers, one at a time, up to 50 nodes per cluster. Yearly prices are ten times the monthly ones. On Community, one volume step is +\$150 a month for each cluster. Additional nodes are not sold on Community.

More than ten clusters, no cluster ceiling, or no node ceiling is a custom agreement rather than a listed price.

Prices exclude VAT. Checkout and payment run on Polar, the merchant of record. The card is entered on Polar's checkout and never reaches Attestkeep's servers. Any tax due is calculated there from your billing address and shown before you pay. A VAT-registered business in the EU accounts for VAT itself under the reverse charge. The licence and support come from Attestkeep.

There is no time-limited trial. Community Edition has every security feature and does not expire, so you can test the product on your own cluster for as long as you need.

The metered unit is the image name: registry and repository, without tag or digest. Everything running in the cluster counts, third-party images included. When the quota is used up, only new image names are denied at admission. Every image already counted this period keeps being scanned and enforced.

13. Security, support and who you are dealing with

Release integrity

Every release image is built for linux/amd64 and linux/arm64 and signed with cosign on its digest, keyless through GitHub's OIDC and with the key published at docs.attestkeep.com/cosign.pub. The scanner and the signature verifier are built into the image, not downloaded at runtime. The source code is not public.

Reporting a bug or a vulnerability

Bugs go to the public tracker at github.com/attestkeep/attestkeep-docs, for every edition. Vulnerabilities go through the private advisory form on the same repository or to security@attestkeep.com. A report is acknowledged by a person within three working days, and fixed issues are published as advisories once a release carries the fix. Regulation (EU) 2024/2847 asks a manufacturer to name one contact point for vulnerability reports. Ours is security@attestkeep.com, and the same details are printed on every evidence package.

Support by plan

Plan	Support
Community	Public issue tracker
Team	E-mail
Business	E-mail, prioritised
Enterprise	E-mail
Custom agreement	As set in the agreement

Attestkeep does not publish a response-time guarantee for any plan.

Who you are dealing with

Trading name	Attestkeep
Business name	Ecavist Technologies, a business registered in Türkiye
Registered address	Dört Yol Küme Evleri No: 369, İç Kapı No: 3, Işıklar, Merkez, Çanakkale, Türkiye
Legal notices, contracts, invoicing	legal@attestkeep.com
Support and pre-sales	support@attestkeep.com
Security reports	security@attestkeep.com

The binding documents (terms of service, distance sales agreement, privacy notice, data processing terms, sub-processors) are at attestkeep.com. Where a plain-language page and an agreement disagree, the agreement applies.

Appendix. What this handbook was checked against

Product version	Attestkeep 1.3.7 (chart 1.3.7, appVersion 1.3.7), released 15 October 2026
Published image	<code>ghcr.io/attestkeep/attestkeep-k8s:1.3.7</code> <code>sha256:1fbf69c8900369e76479e2aaf6ab80e96cb7e65ff01741ba0bb3a13c85ef5bb2</code>
Published chart	<code>oci://ghcr.io/attestkeep/charts/attestkeep</code> version 1.3.7 <code>sha256:843e6b26850ac6a3f8740a7834aae79990d8c29400d6c2f0faffc81db6a502c6</code>
Installation photographed, 1.3.7	An installation running the published 1.3.7 image (digest above, checked on the running pods) on Kubernetes v1.35.1 with three nodes, as Enterprise Edition, with Falco connected. Photographed on 15 October 2026 in the dark theme: the security overview, Images, the policy list, the What if result, What-if history and Falco Policies.
Installation photographed, 1.3.2	An installation whose console reports version 1.3.2, on Kubernetes v1.35.1 with three nodes, running as Community Edition, with Falco 0.45.0 connected through Falcosidekick over mutual TLS. Photographed on 3 October 2026 in the dark theme: every other screen. Its image digest was not checked against the published 1.3.2 digest. The Create your account screen in chapter 4 was taken on version 1.0.1. Its text is unchanged in 1.3.2.
Evidence package shown	<code>PS-S0C2-DEVMN3-20261002-20261002-56A369</code> , generated 2 October 2026 14:03 UTC by version 1.3.1
Documentation	docs.attestkeep.com pages for 1.3.7: releases, install, upgrade, what runs, admission, configuration, runtime events, licensing, backup, using the console, policies, try a policy before enforcing it, compliance, issue trackers and operations
Prices	attestkeep.com/pricing , USD, excluding VAT. The figures are unchanged since the 1.3.2 edition and were checked against the site's price list on 15 October 2026.

Documentation: docs.attestkeep.com · Release notes: docs.attestkeep.com/releases · Verifier for delivered evidence packages: attestkeep.com/verify.html

What changed since 1.3.2

Version	What changed
1.3.7, 15 October 2026	Containers added with <code>kubectl debug</code> are reviewed at admission (chapter 3). The Images page puts each image in one of four states, with the new Not compliant tile, and Blocked follows the last admission review. What-if runs are kept for ninety days under Security > What-if history, with PDF and JSON downloads that are a projection, not evidence. What if on every policy row, Scan now for the images a replay found unscanned, a confirmation before Cancel discards changes, Apply to cluster only for accounts that may write policies, and one list of framework names. The operator is built with Go 1.26.9 and <code>golang.org/x/net</code> 0.60.0, which close published vulnerabilities the operator reaches. Migrations 0033 to 0035.
1.3.6, 8 October 2026	The Images page counts admitted pods only. The Falco Events action filter offers the five effects. Layout fixes for the sidebar and the evidence list at 1366 and 1600 px, and for the usage gauge. The New policy form proposes the documented SLA and triage defaults.
1.3.5, 8 October 2026	A Falco policy set to Stop the pod is labelled so everywhere, and a pod stop is written as <code>policy:<name></code> in the evidence package. The cluster is named by the chart's <code>clusterName</code> . The Audit Log uses local time. The Vulnerability database tile shows when it last checked and has Update now. The webhook tile no longer compares latency with a fixed budget.
1.3.4, 8 October 2026	Evidence fixes: break-glass details come from the policy revision in force at the time, triage decisions carry the CVE's severity, and the console's package view counts refused reviews correctly.
1.3.3, 7 October 2026	Migrations run under a database lock, so a fresh install or an upgrade with migrations no longer restarts a pod once.