Attestkeep

Product handbook

Container image security for Kubernetes: scan, verify signatures, decide at admission, and keep the signed evidence that it happened. What Attestkeep is, what it installs, how to run it, and what it costs.

Attestkeep 1.0.1 · Helm chart `oci://ghcr.io/attestkeep/charts/attestkeep` version 1.0.1 · handbook dated 6 September 2026.

Every screenshot in this document was taken on 6 September 2026 from live 1.0.1 installations on Kubernetes v1.35.1. Every figure and command comes from the product documentation at docs.attestkeep.com or from the installations themselves; nothing here is illustrative.

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

Contents

1. What Attestkeep is

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

It runs entirely inside your cluster. The operator, its API, the console and the database are installed by a single Helm release into a namespace you own. The only thing Attestkeep ever sends out is a daily licence check carrying three values; image names, findings and cluster topology never leave.

What it does

- **Vulnerability scanning** — Trivy, pinned into the image. Findings by severity, with the fixed version where one exists.
- **Signature verification** — cosign verification, by key or keyless, plus 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.
- **Triage with an expiry** — accept a risk with a reason and a deadline; break-glass when a deployment cannot wait. Both are recorded.
- **Evidence packages** — one consolidated, signed document per audit period, mapped to the control set you select, stating what it cannot prove as clearly as what it can.
- **Audit log** — policy changes, admission overrides, triage decisions, evidence exports and access events, in one record.
- **Notifications** — Slack, Teams, e-mail and a generic webhook. They name the cluster, never its topology.
- **Single sign-on and roles** — Google, GitHub, GitLab, Okta, Microsoft Entra ID, Keycloak or plain OIDC (paid editions). Roles decide who may change a policy.

What it does not do

It is not a runtime sensor: it does not watch syscalls, network flows or process trees inside a running container. It is not a cloud posture tool: it does not read your cloud account or IAM policies. What it does is narrower and provable — it decides what is allowed to start, and it keeps the record of what it decided.

Why not assemble the gate from free tools?

You can. Admission policy engines and in-cluster scanners exist as excellent open source, and if what you need is "block `:latest` and fail critical CVEs", you can build that without Attestkeep. What that assembly does not give you is the part auditors and incident reviews consume:

- **A decision record with integrity.** Every admission decision is content-hashed at write time and sealed under signed, chained checkpoints. Flip one verdict in the database and the next evidence document names the exact record.
- **Decisions tied to the policy that made them.** Each record carries the hash of the policy as it stood at that moment.
- **Honest gaps.** Windows where the webhook was down or unverifiable are recorded and printed in the evidence, and the reconciliation sweep names what got in during them.
- **One surface.** Scan results, signature verification, admission, exceptions and the evidence package, maintained as one product rather than glue code your team owns forever.

2. What runs in your cluster

You are being asked to give an admission webhook a say over every pod. This is the inventory of what that costs.

Object	Default	What it does
Deployment `attestkeep`	2 replicas, anti-affinity preferred across nodes, PodDisruptionBudget minAvailable 1	The operator: admission webhook (8443), console and API (8080), metrics (9090). Controller work is serialised through a PostgreSQL advisory lock, so replicas scale without a leader-election lease.
Deployment `attestkeep-scanner`	1 replica	Runs Trivy scans and cosign verification off the admission path. Never receives admission traffic.
StatefulSet `attestkeep-postgresql`	1 replica, 20 GiB, postgres:16.4-alpine	The bundled database. Point `externalDsn` at your own PostgreSQL and this object does not exist.

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 / 256Mi per product workload. Cluster-scoped objects: four CRDs in the `attestkeep.com` group (ImageSecurityPolicy, ImageSecurityReport, AttestationRenewal, ImageSecurityUsage) and one ValidatingWebhookConfiguration — pod CREATE and UPDATE only, 15-second timeout, `failurePolicy: Ignore` by default. There is no mutating webhook: the product never rewrites your workloads.

Every outbound connection

This is the table for your firewall ticket. There is no analytics, no crash reporting, no telemetry SDK.

Destination	Required?	Knob	If blocked
lic.attestkeep.com:443	Yes	`license.serverUrl`	Activation is impossible; an activated install keeps running on its held certificate and degrades to Community when that certificate expires.
ghcr.io (Trivy vulnerability DB):443	For scanning	`trivy.dbRepository` — mirror it	The existing database keeps working and ages; `attestkeep_vulnerability_db_age_seconds` tells you by how much.
Your image registries:443	For scanning	Pull secrets resolved from pod specs	Scans fail for images there; admission follows your policy's cold-image setting.
ghcr.io/attestkeep/attestkeep-k8s:443	At install	`image.repository` — mirror it	Pods cannot start.
Sigstore TUF + public Rekor:443	No	`scan.cosignOffline: true`	With keyed signing and offline mode on, verification stays local.
Your OTLP collector, Slack/Teams/webhook URLs, SMTP, SSO provider	No — off by default or configured by you	Per feature	That feature fails and is logged; admission is never blocked by a notification.

If our licence server were compromised

It cannot instruct your cluster: the licence protocol has two response shapes, a certificate and a renewal status, and no code path where a server response selects a policy, a namespace, an image rule, a URL or an executable. Certificates are Ed25519-signed and verified against a public key compiled into the binary, and bound to your cluster's fingerprint. What a hostile server could do is answer *revoked*: the result is a downgrade to warning-only admission, visible in the console and in every admission response — never a workload it placed or a policy it changed. Silence does nothing: network errors are read as outages, and your install runs on its held certificate until that certificate's own expiry.

3. How an admission decision is made

The webhook answers from cache. Scanning never runs in the admission path, which is why a slow registry cannot slow down a deployment.

1. The API server asks about a pod.
2. The tag is resolved to a digest. That resolution is cached for a minute, so a burst of identical deployments makes one registry call rather than a hundred.
3. The digest's scan result and signature verdict are read from cache.
4. The policy is evaluated against them and the answer is allow or deny, with a reason.
5. The decision is recorded — including the allows, because evidence that only lists denials cannot show a control was operating.

Nothing in that list waits on a scan. If the digest has never been seen, step three has no answer, and what happens next is a policy decision.

The two settings that decide your posture

Setting	Default	What it means
`webhook.failurePolicy` (chart)	`Ignore`	What the API server does when the operator does not answer. Ignore means an outage leaves you unguarded, never blocked. Fail denies until the operator answers.
`admissionTiming` (policy)	`AllowAndScan`	What happens to a digest never seen before. AllowAndScan admits it and queues the scan; DenyUntilScanned refuses it until a scan record exists, which is Fail's natural companion.

Neither answer is universally right. Ignore keeps your cluster deployable during an outage of ours and leaves a gap in the evidence — for that window nothing was reviewed, and the evidence package says so rather than quietly showing a clean quarter. Fail gives you a control that provably never had a gap, and it means an operator outage stops deployments. Both settings are available in every edition, Community included.

Whatever crossed during a gap is still running after it closes. The runtime reconciliation sweep is what finds it: on a schedule (hourly by default) every running container's observed imageID is checked against the admission ledger, and a digest with no admission event behind it is reported, announced, or — under `runtimeReconciliation: enforce` — stopped so it re-enters through the webhook.

> **The first scan of a new digest runs in the background.** On a fresh cluster it can take a few minutes — the scanner pulls the image and the vulnerability database is cold. Until it finishes the image shows as *Unknown* on the Images page with the reason "admitted before first scan". Every later admission of that digest decides on the finished report; *Rescan* runs a scan on demand.

4. Install and first run

Before you start

- Kubernetes 1.27 or later — any conformant distribution, including managed services, k3s and OpenShift.
- Permission to create a ValidatingWebhookConfiguration (cluster-admin on the first install).
- Roughly 20 GiB for the bundled PostgreSQL, or the DSN of one 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 nothing beyond your registry and the public key published at docs.attestkeep.com/cosign.pub.

```
curl -sO https://docs.attestkeep.com/cosign.pub
cosign verify --key cosign.pub ghcr.io/attestkeep/attestkeep-k8s:1.0.1
```

Install

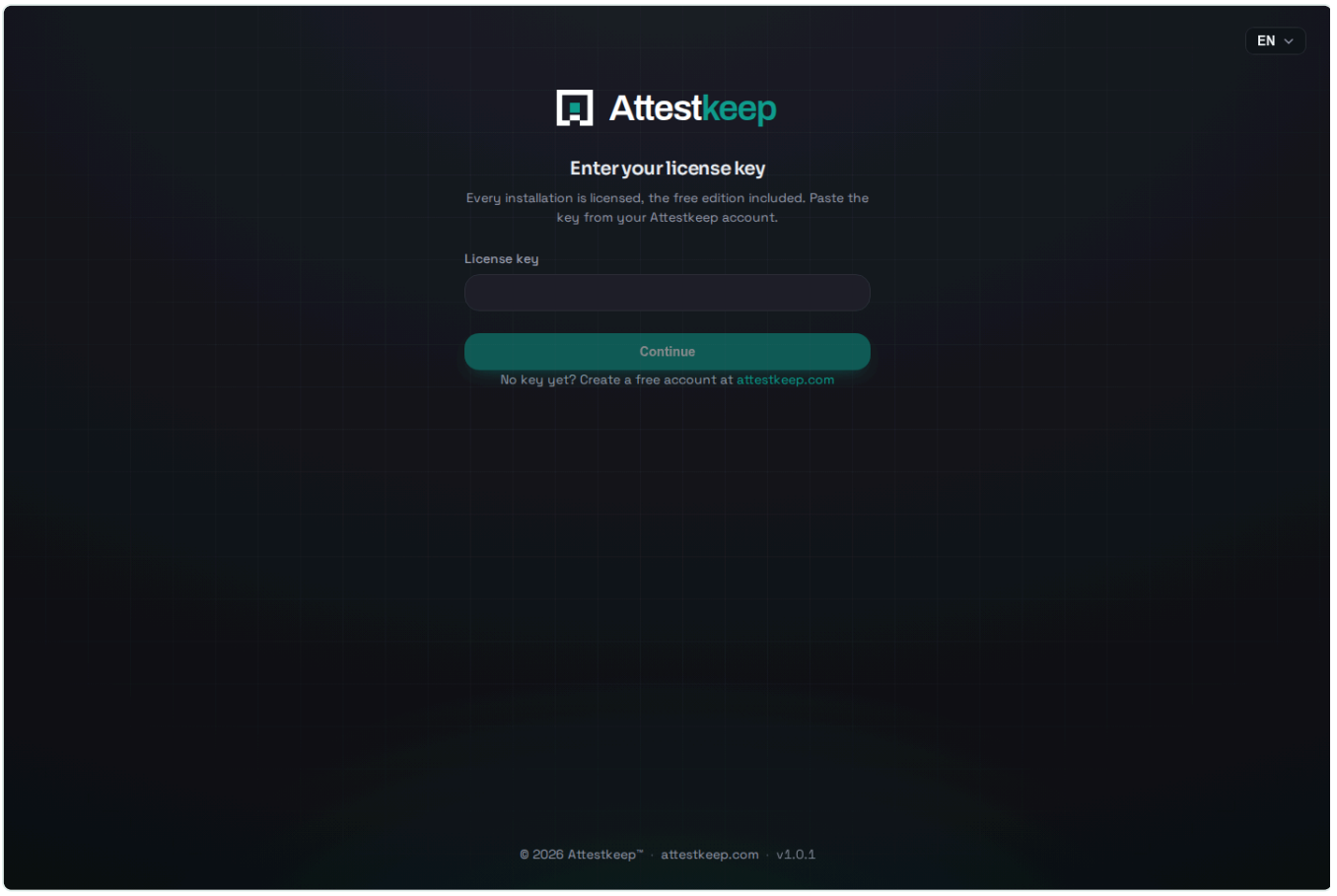
```
# installs from the registry; no `helm repo add` needed
helm install attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --version 1.0.1 \
  --namespace attestkeep --create-namespace \
  --set clusterName=production-eu \
  --wait
```

`clusterName` is the one value worth setting on the first install, even for a trial. It is the label every notification and every evidence document uses to say which installation it came from. Nothing topological leaves the cluster, so this is a name you choose, not a hostname.

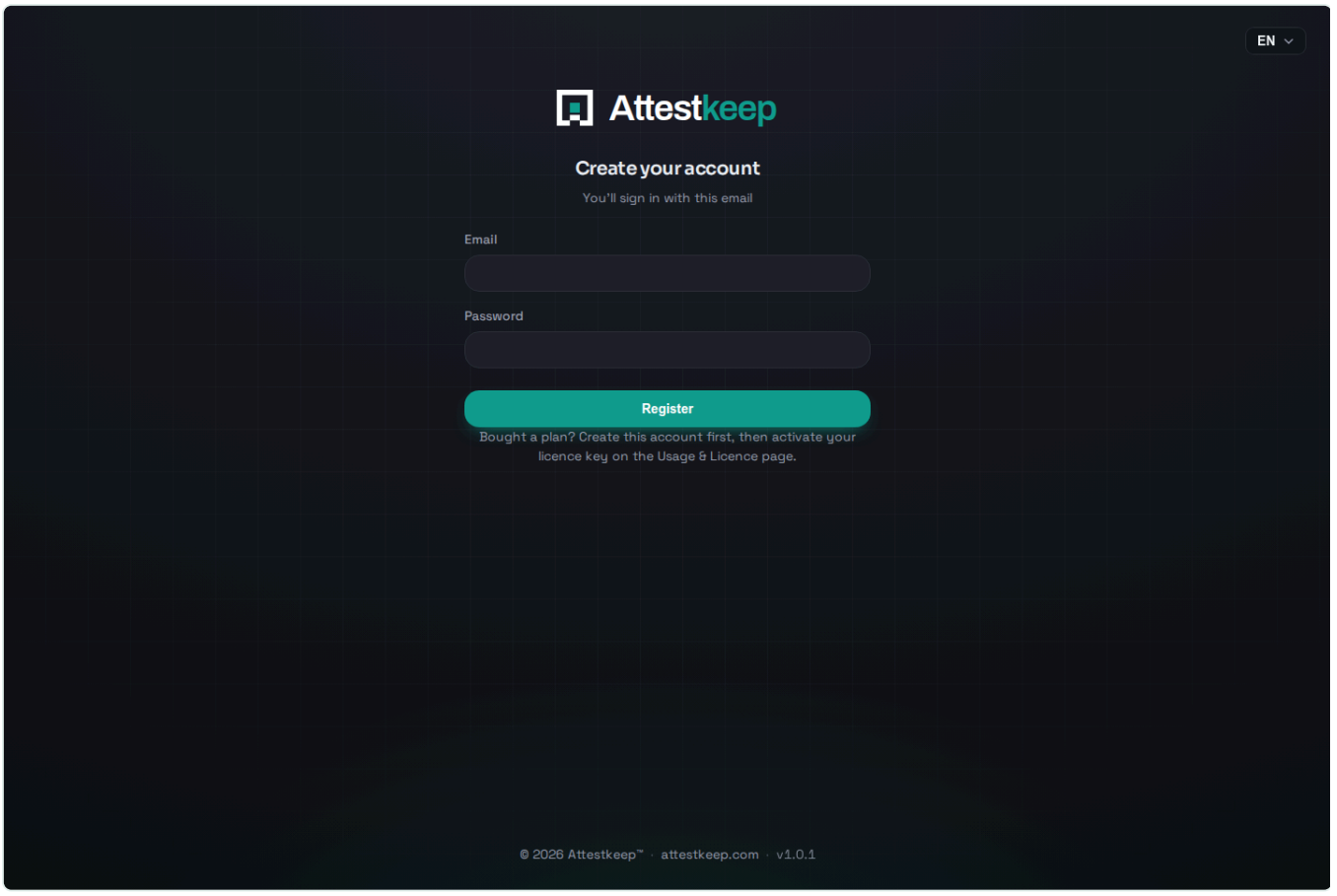
Confirm it is running and open the console

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

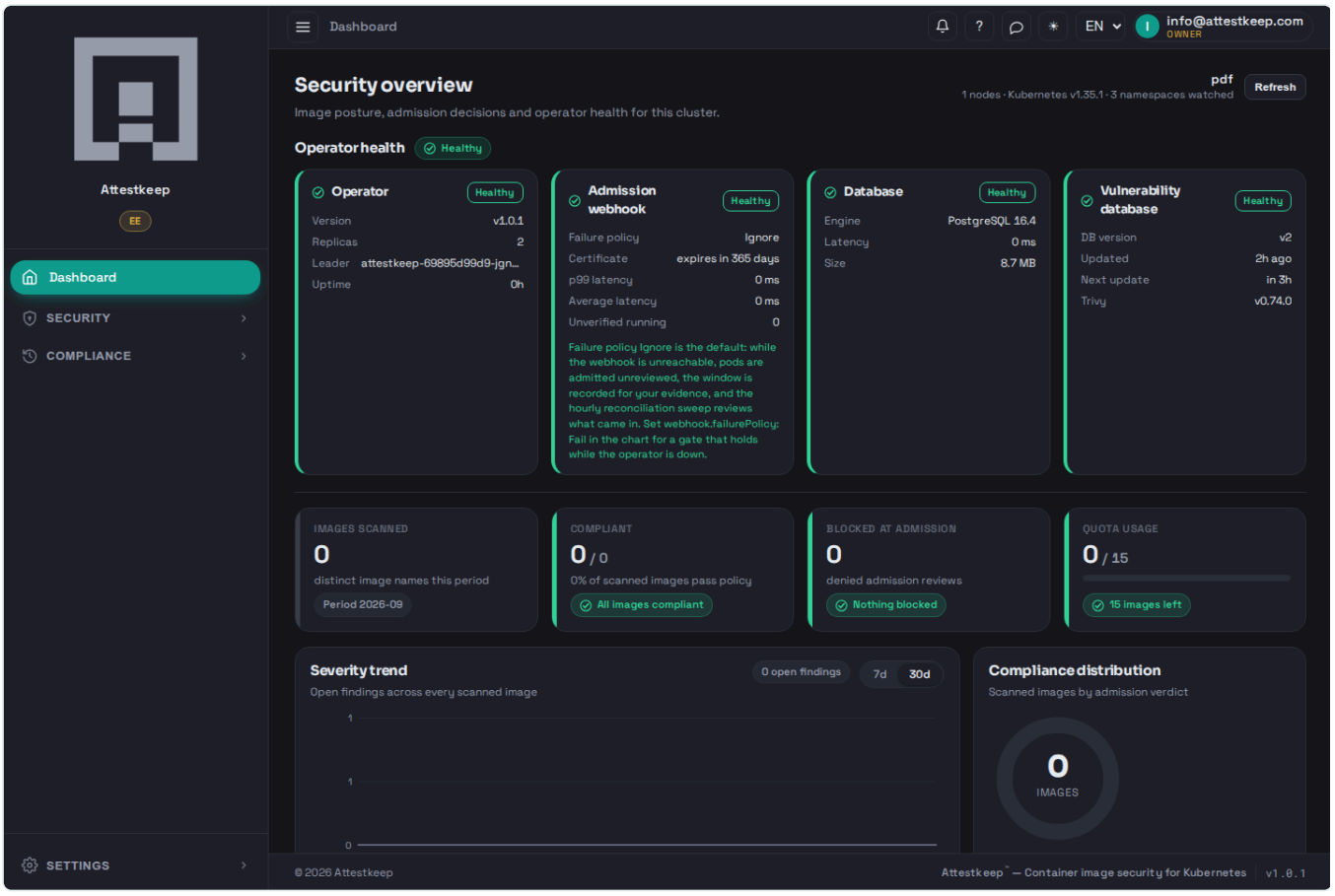
Then open `http://localhost:8080`. The console walks a new installation through three screens, once. After that it is a normal sign-in.



Screen 1 — enter your licence key. Every installation is licensed, the free Community edition included. Activation happens once and binds the key to this cluster; the key is exchanged for a signed certificate the operator verifies inside the cluster.



Screen 2 — create the owner account. Whoever completes this first becomes the owner; the window closes the moment the account exists. Bought a plan after installing with a Community key? Activate the paid key later on the Usage & Licence page — the account and the licence are separate.



Screen 3 — you land on the security overview. This is a freshly activated Community installation minutes after install: operator, admission webhook, database and vulnerability database all healthy, 0 of 15 image names used this period. The webhook tile explains the default failure policy and how to change it.

The database builds itself

Migrations are compiled into the binary and applied on first connection, each recorded so it runs exactly 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
```

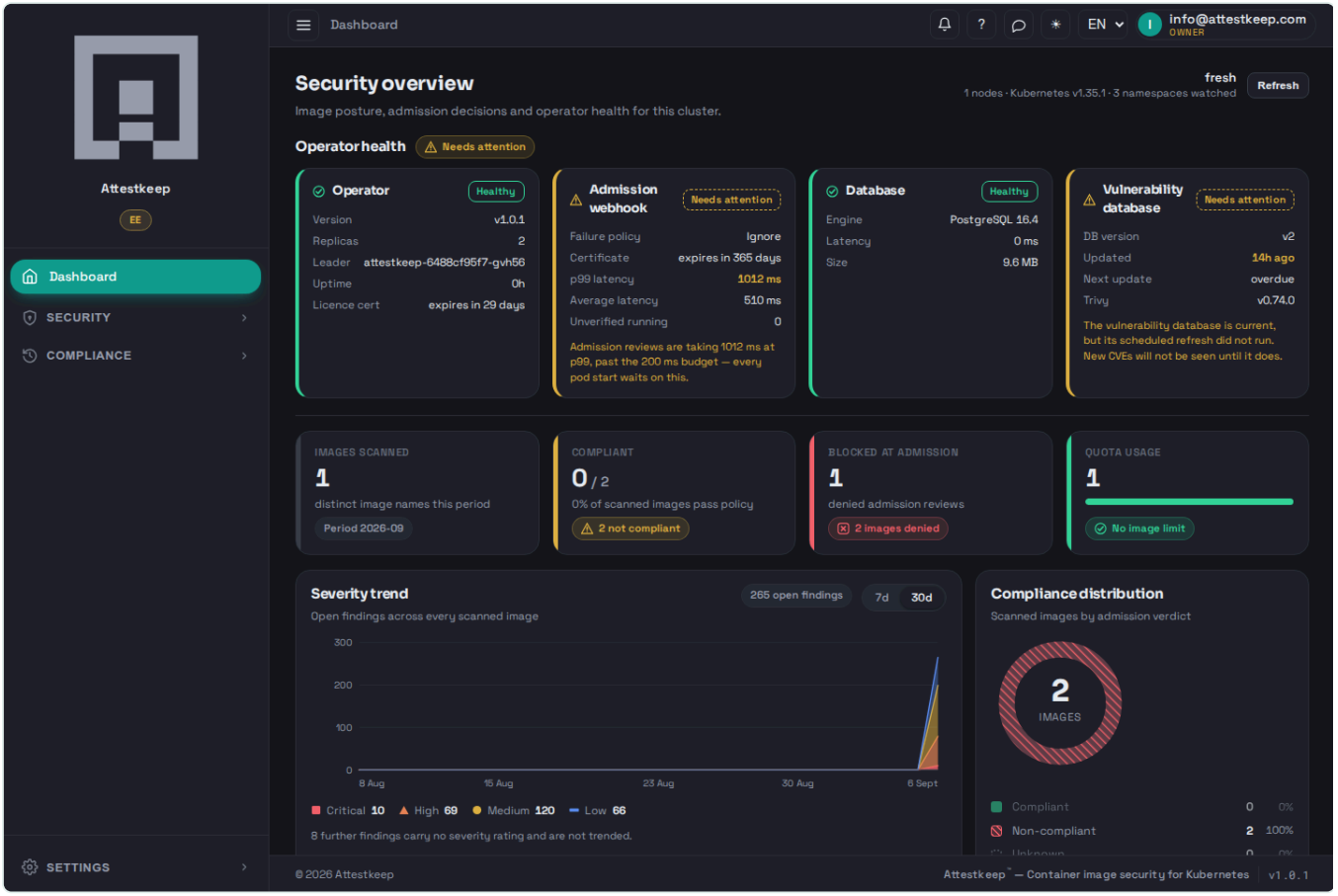
Private registries: nothing to configure. The operator resolves the same image pull secrets the pod references, and those on its service account, and uses them for both the scan and the signature check.

Upgrade is `helm upgrade`; embedded migrations apply the new steps and skip the rest. **Uninstall** is `helm uninstall`: the webhook configuration goes with the release, so admission stops being reviewed the moment it is gone. The database volume is not deleted.

5. Using the console

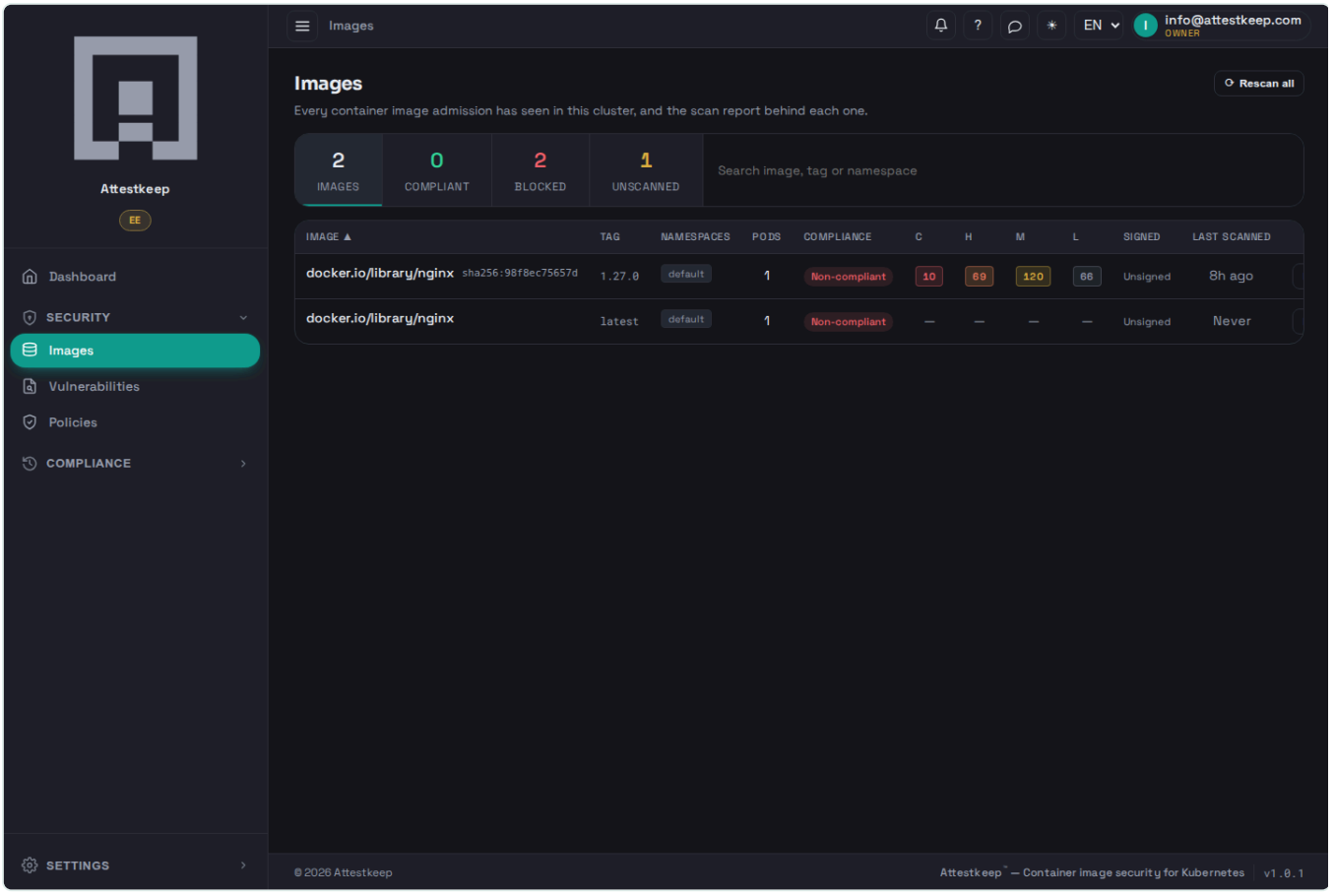
The console is part of the release, not a second product: the same Helm release that installs the operator installs the console it serves. It loads no fonts, scripts or analytics from anywhere. English and Turkish, light and dark, and a help panel on every screen.

The security overview



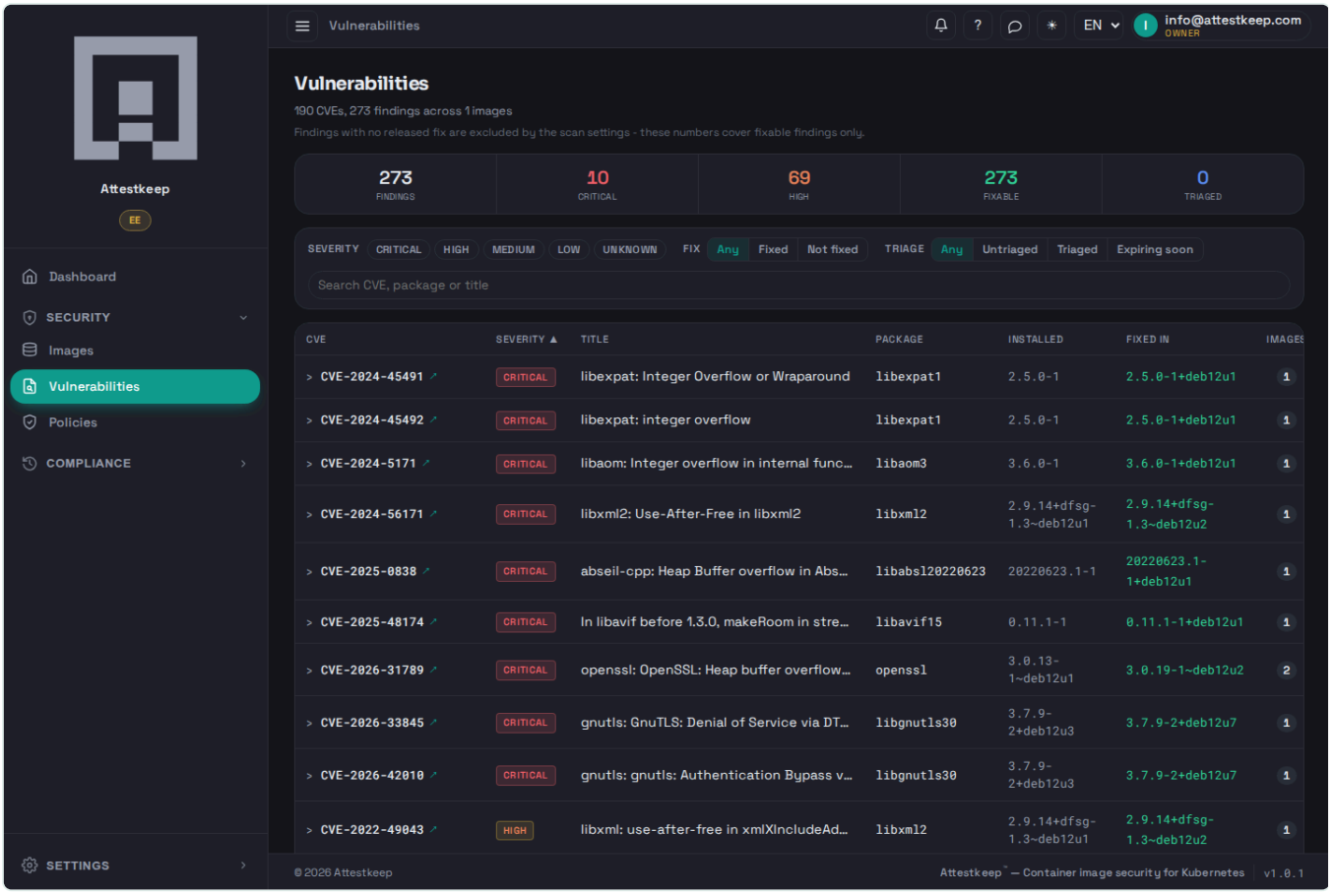
A Team installation after a few admissions: 1 distinct image name counted, 2 images scanned and non-compliant, 1 admission denied, 265 open findings. The health tiles are honest about what needs attention — here, cold-path admission latency on a small test cluster and a vulnerability-database refresh that missed its schedule while the cluster was stopped.

Images



Every container image admission has seen in this cluster, and the scan report behind each one: compliance verdict, findings by severity, signature state, last scan. *Rescan* queues a scan on demand.

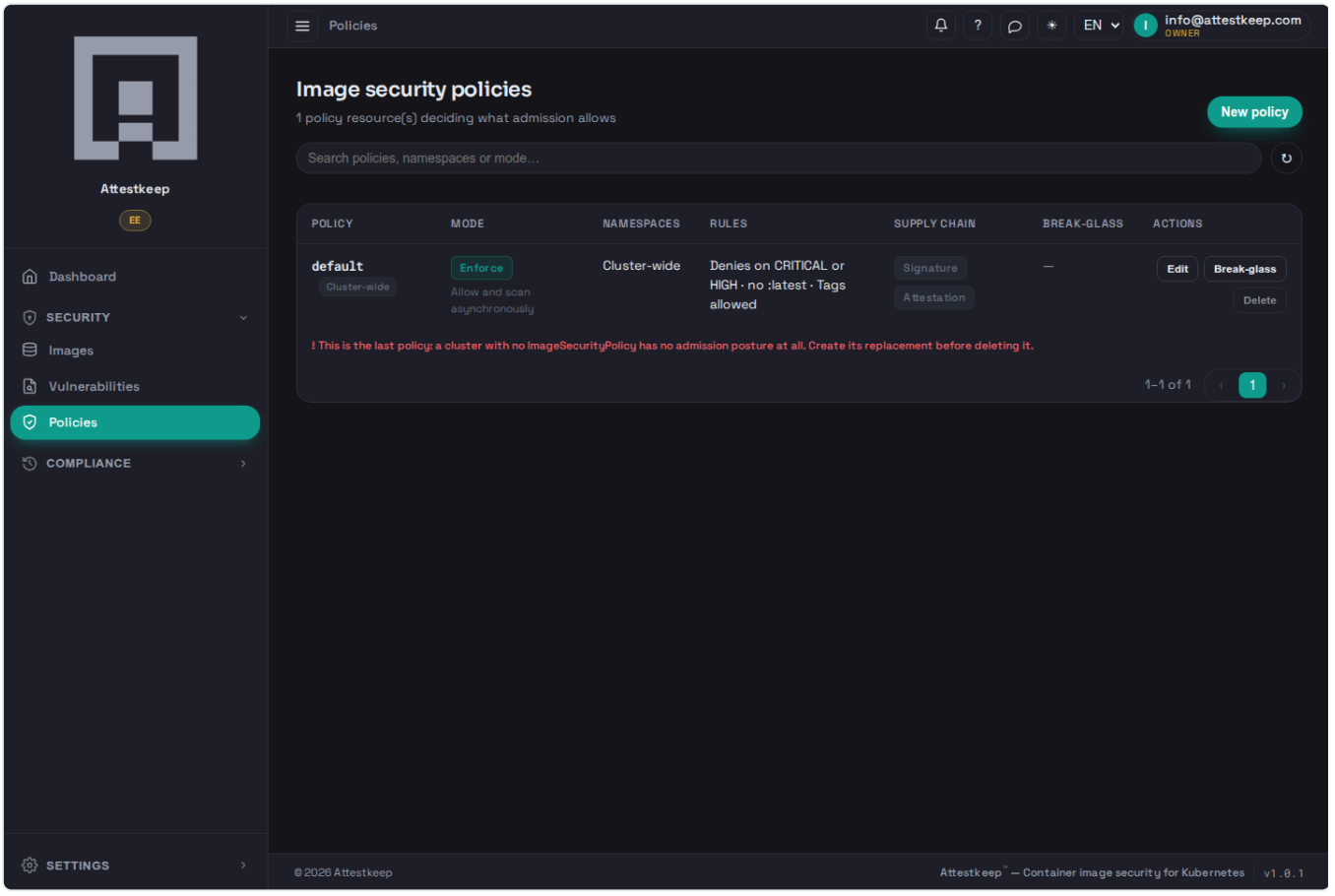
Vulnerabilities and triage



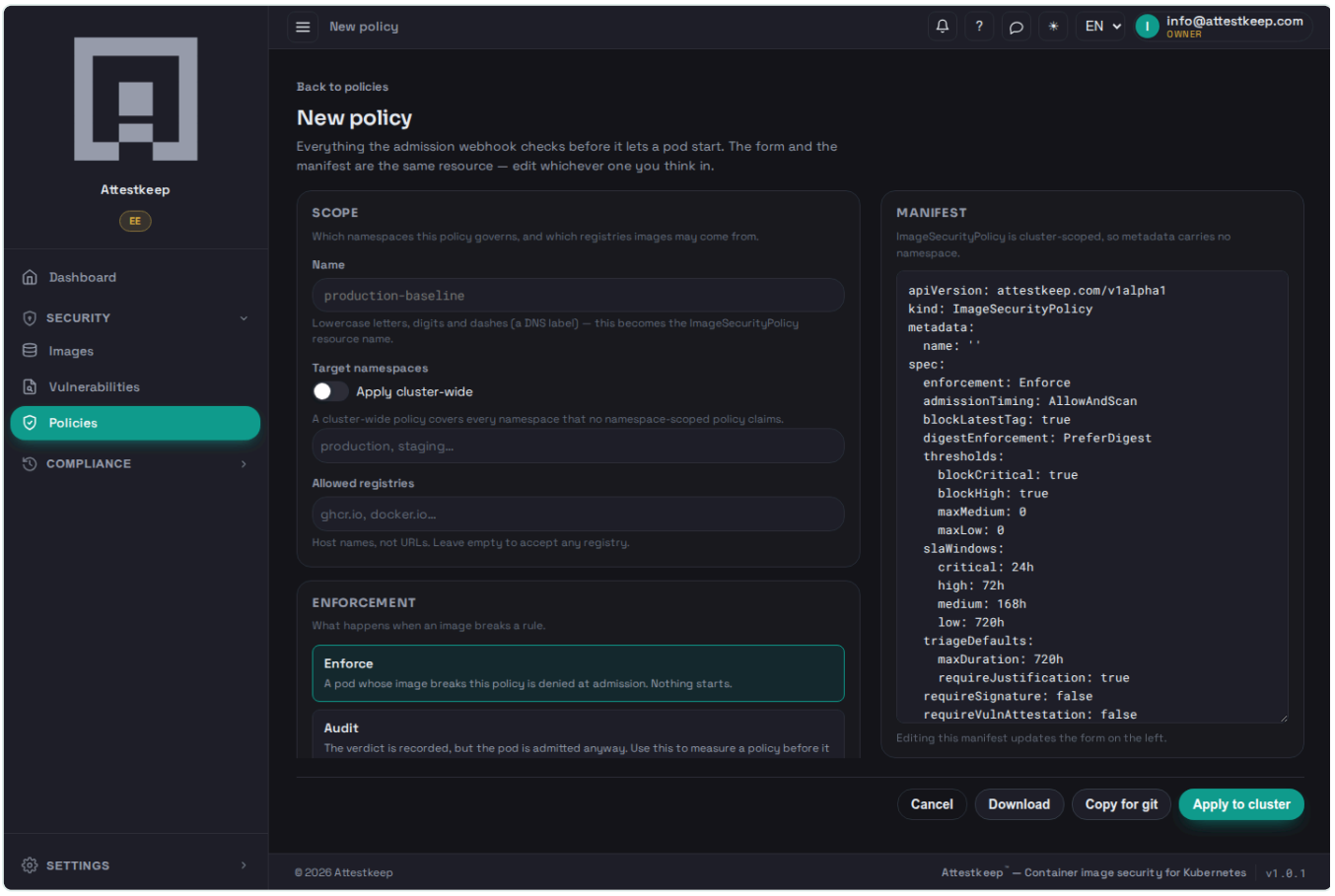
Findings across every image, grouped by CVE, with the installed and fixed versions side by side. Filters narrow by severity, by whether a fix exists, and by triage state. Here: 190 CVEs, 273 findings across one image (nginx 1.27.0), 10 critical, 69 high.

A triage decision is a record an auditor reads, so it is deliberately not a silent mute: choose the status (false positive, accepted risk, under investigation), write a justification, set an expiry. When it lapses, the finding returns to the policy verdict on its own. Every decision is stamped with who made it and when, and it lands in the audit log and the next evidence package.

Policies

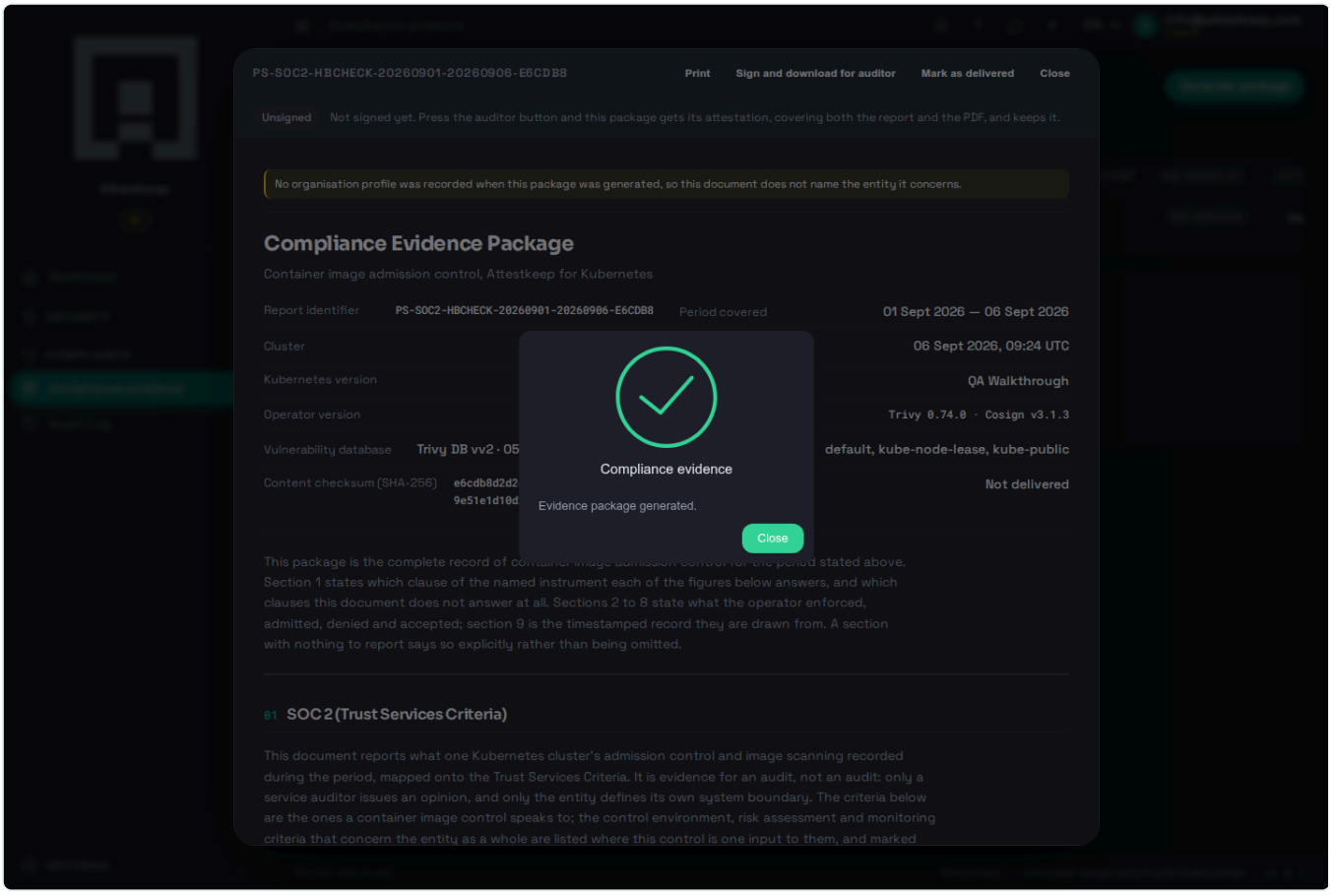


The whole rule set at a glance: each row states the mode, the namespaces it covers, the rule summary and the supply-chain checks it requires.



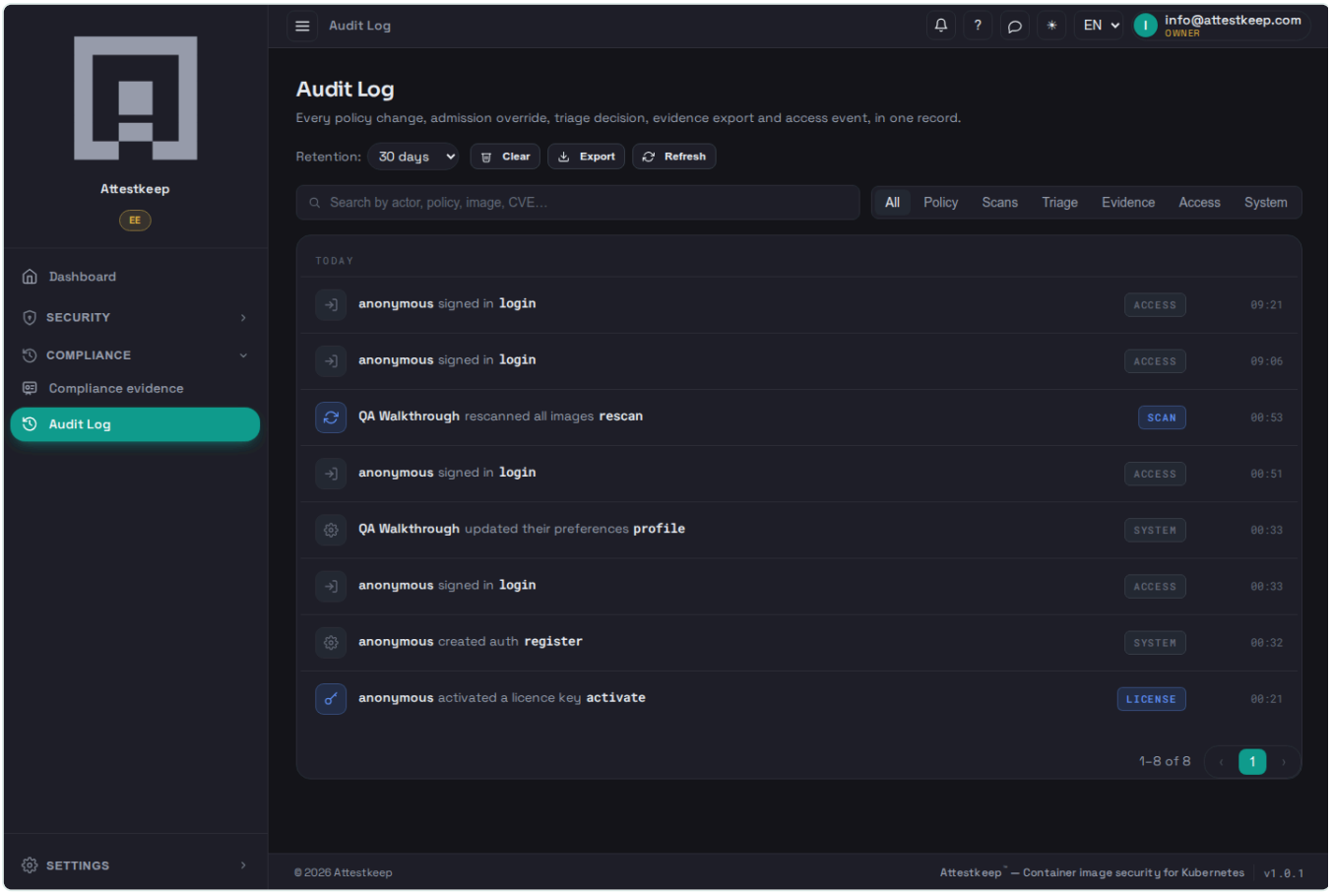
The editor: the form and the manifest are the same resource — edit whichever you think in. *Copy for git* hands the manifest to your repository; *Apply to cluster* writes it live. The editor knows when Argo CD or Flux owns the object.

Compliance evidence



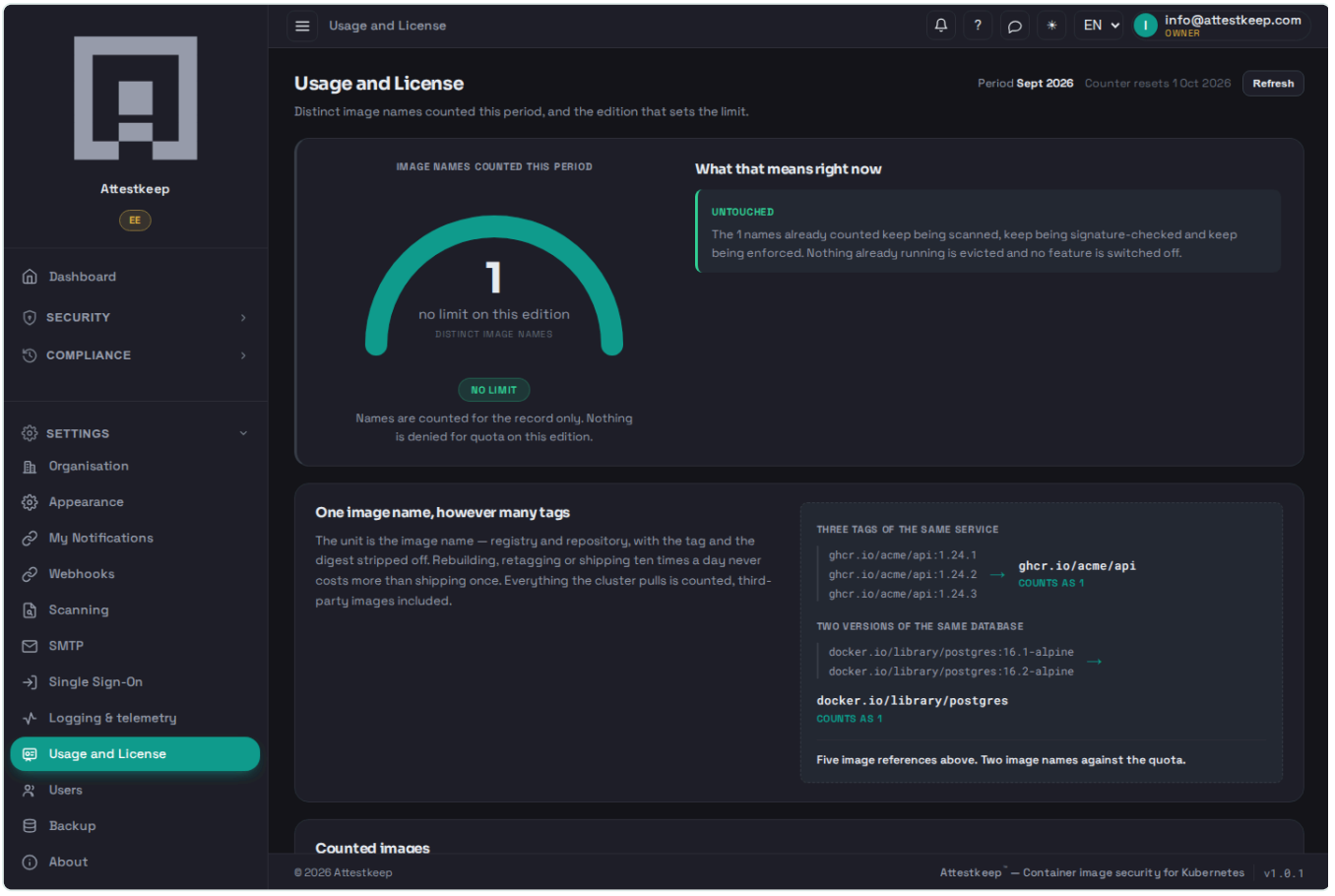
One consolidated document per audit period. The package `PS-SOC2-HBCHECK-20260901-20260906-E6CDB8` was generated for this handbook from the installation's own records. The signing key at the bottom is what an auditor verifies a package against.

Audit log



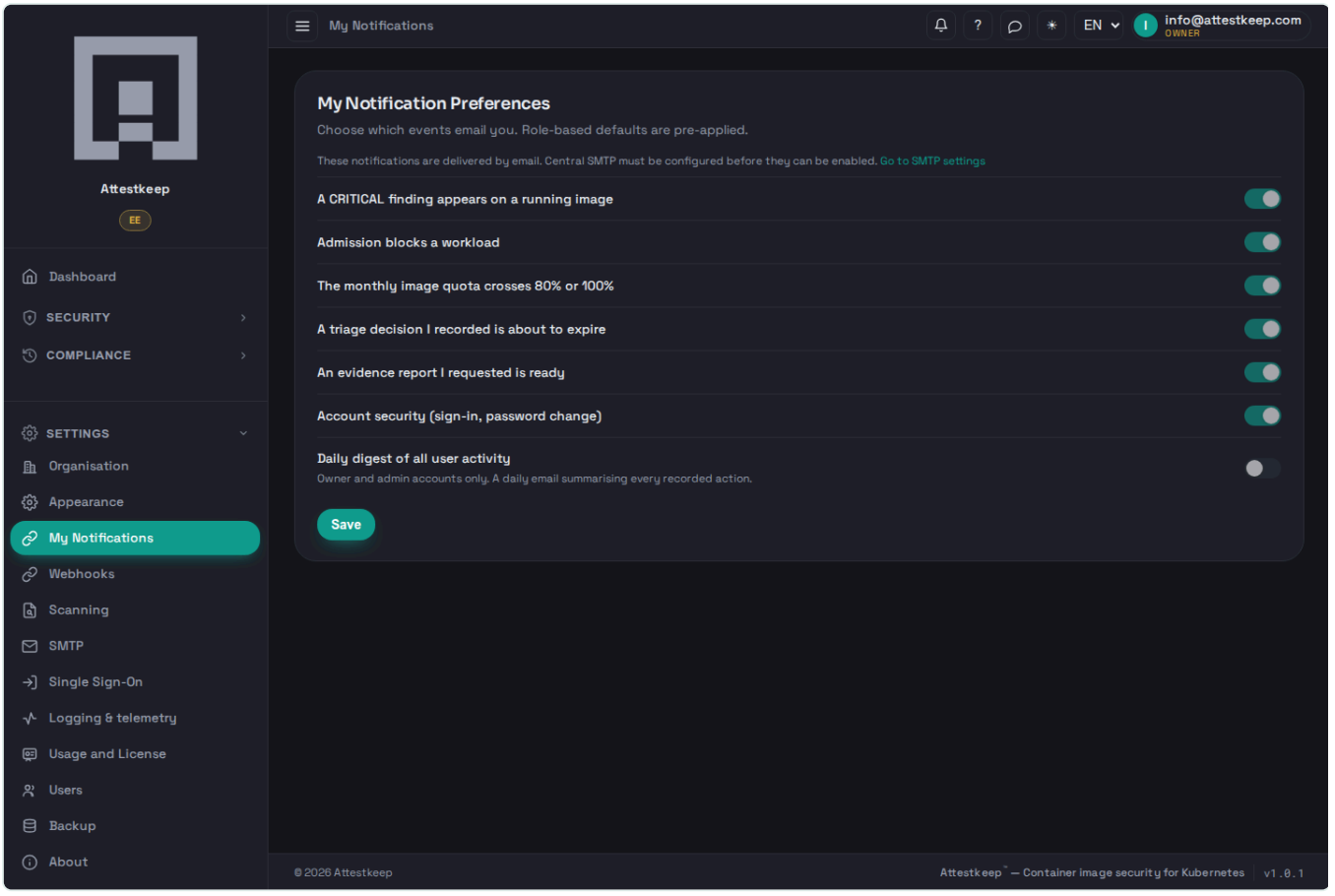
Every policy change, admission override, triage decision, evidence export and access event, searchable by actor, policy, image or CVE, with a retention you set.

Usage and licence



Distinct image names counted this period, the edition that sets the limit, the certificate's runway, and where to replace the key. The unit is the image name — registry and repository with the tag and digest stripped — so retagging and shipping ten times a day never costs more than shipping once.

Notifications



Per-role e-mail preferences; channels (Slack, Teams, webhook, SMTP) are configured under Settings. A notification names the installation, the image reference and the verdict — never a node name, an internal address or an endpoint.

6. Policies

One cluster-scoped resource decides what may run: `ImageSecurityPolicy` in the `attestkeep.com` API group. The console edits it live; `kubectl apply` edits it from a repository. They are the same object, so GitOps and the UI never fight over a second source of truth. A policy edit takes effect on the next admission decision — no redeploy, no restart.

How a namespace gets its policy: a policy whose `spec.targetNamespaces` lists the namespace wins; otherwise the policy with no `targetNamespaces` applies — the global default. The usual arrangement is one global policy plus a stricter one naming the namespaces that have earned it.

The shipped default, annotated

```
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 production namespace that has earned strictness

```
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: []
```

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 one tag that changes underneath you.
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 scopes it to those namespaces.
thresholds	—	blockCritical/blockHigh refuse on any such finding; maxMedium/maxLow set ceilings.
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, a vulnerability-scan attestation.
attestationMaxAge	duration	An attestation older than this no longer counts.
workloadHardening	mode, minSeverity, exempt	Whether privileged containers, hostPath mounts and similar refuse, warn, or pass.
runtimeReconciliation	audit | notify | enforce	What the periodic sweep does about a running image with no admission event behind it.
breakGlass	—	A named, approved, expiring exception; every use is recorded.
triageDefaults	maxDuration, requireJustification	How long an accepted risk may stand, and whether it must say why.
storageUnreachablePolicy	Allow | Deny	The admission answer while the database is unreachable.

Break-glass

Production is down, the fix is an image the policy refuses, and the incident channel has agreed to let it through. Break-glass admits the named images, records every use of the exception, 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 as such in the audit trail and in compliance reports — the exception is evidence too.

7. Prove it is enforcing

An admission webhook that is silently not running looks exactly like a compliant cluster. This runbook takes five minutes, uses nothing but kubectl, and ends with you having seen a denial with your own eyes. Run it after install, after upgrades, and any time trust needs re-earning.

1. Throw a bad pod at it

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

Expected — the API server itself refuses, quoting the webhook. This is the verbatim output from the installation this handbook was written against:

```
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 error comes from *Error from server* — the deny happened inside the API server's admission chain, not in some agent that might have been sleeping. And the reason names the policy rule, so a developer reading it in a CI log knows what to fix.

2. Confirm nothing was created

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

3. Check the decision was recorded

The dashboard's *Recently blocked admissions* panel now leads with this denial, same reason string, with the review's hash beside it. The metrics tell the same story: `attestkeep_admission_decisions_total{decision="denied"}` has moved.

4. If the pod went through

- 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 block what you threw — check `enforcement: Enforce` and `blockLatestTag: true`.
- The webhook is failing open — check `attestkeep_webhook_distrusted` (it should be 0) and pod readiness.
- The install 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 never-seen image is refused until its scan exists. Verbatim, 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
```

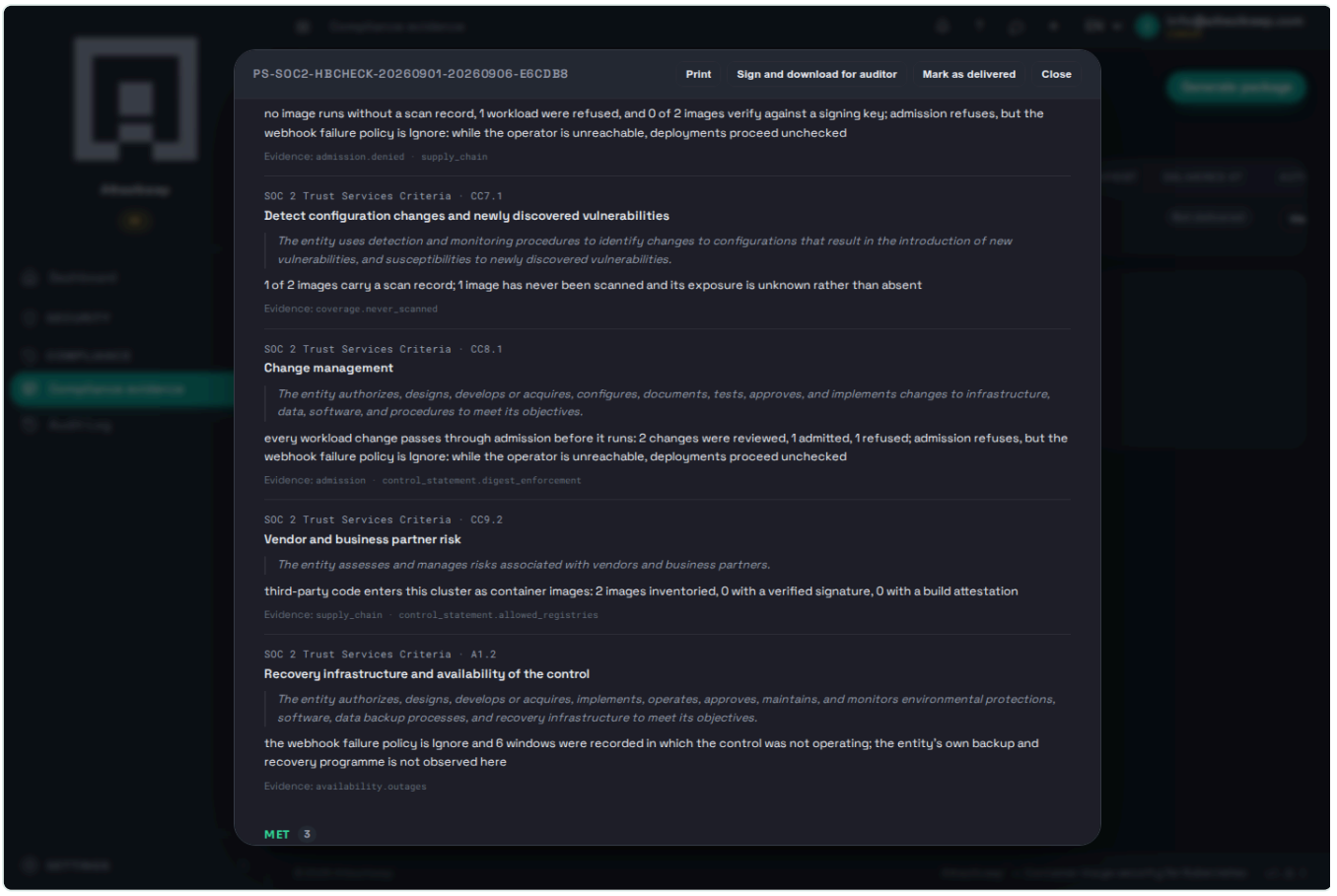
8. Evidence packages and the trust model

Day-to-day scanning is one thing; the record you hand an external auditor is another. An evidence package is one consolidated, signed document per audit period, generated inside the cluster and verifiable by the auditor with their own tools. It is evidence produced from your own records; it is not a certification, and it says so on the document.

Control sets it reports against

Instrument	What is mapped
SOC 2 — AICPA Trust Services Criteria	13 criteria across CC6–CC9 and Availability A1.
EU CRA · NIS2 — (EU) 2024/2847 and Article 21(2)	34 requirements across two instruments that bind different subjects.
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, named individually.
ISO/IEC 27001:2022 Annex A	12 clauses: 10 Annex A controls plus the clauses marked outside scope rather than claimed.
GDPR Article 32	6 clauses on security of processing, paragraph by paragraph.

130 clauses in all, each with its own citation. The mapping ships inside the operator, so an air-gapped cluster reports what a connected one does. Every control lands in one of four states — *met*, *partial*, *gap*, *outside_scope* — and a clause is met only where the period's own records show the control operating, never because the product has the feature.

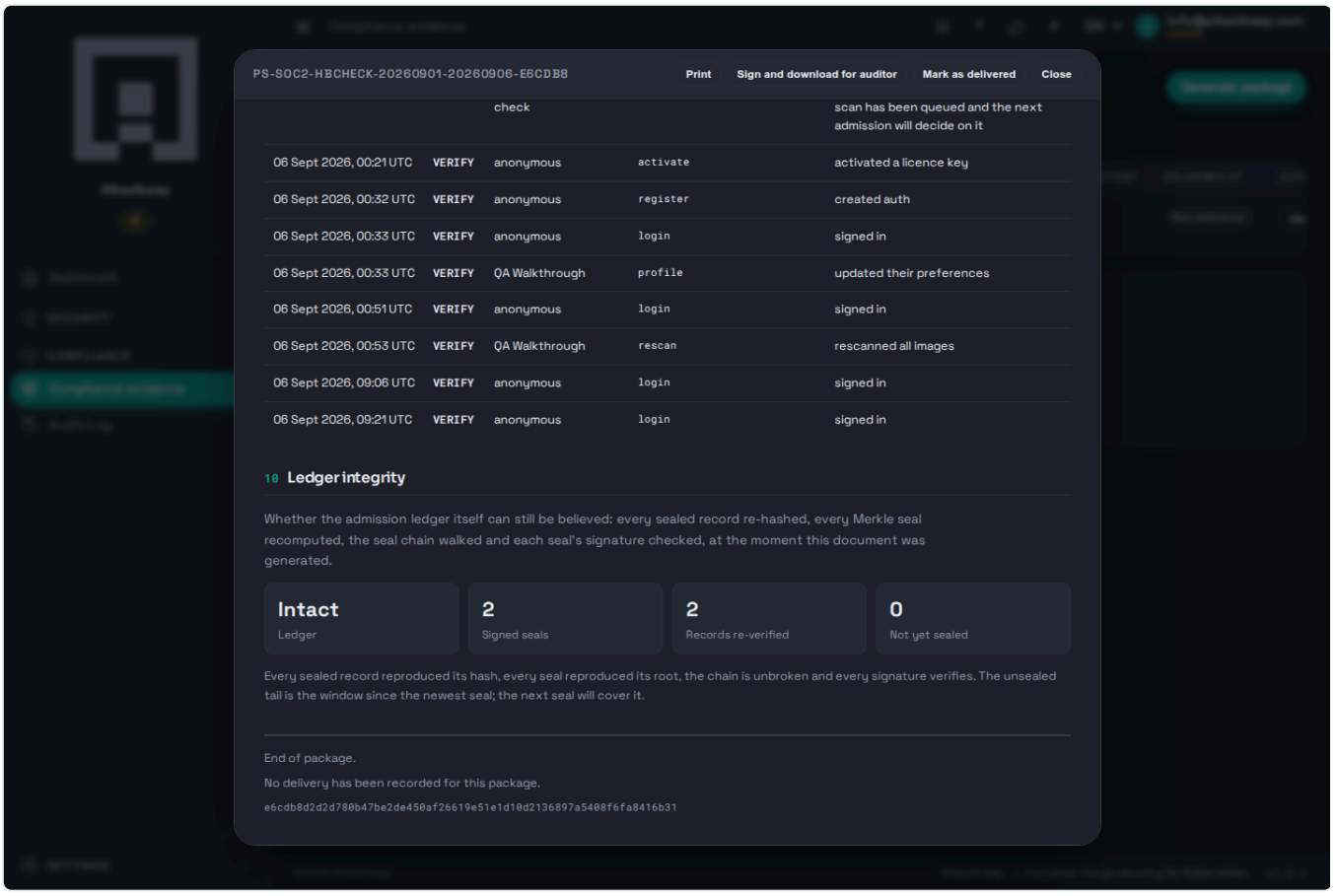


Inside the package generated for this handbook (SOC 2, 1–6 September 2026). Each clause quotes the criterion and answers it with the period's own numbers: "2 changes were reviewed, 1 admitted, 1 refused"; "2 images inventoried, 0 with a verified signature"; and, under Availability, that the failure policy is Ignore and six windows were recorded in which the control was not operating — this test cluster was stopped and started several times that day, and the document says so.

Ledger integrity — can the record itself be believed

Every admission decision lands in PostgreSQL. A database is the right place to query decisions and the wrong place to trust them, so the ledger is made tamper-evident:

- **Every record carries its own hash.** A SHA-256 over the record's content — timestamp, namespace, pod, image, digest, verdict, reason, policy name and policy hash, break-glass flag — computed by the operator at write time.
- **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 also carries the hash of the previous one, so the seals form a chain back to the first.
- **The signing key does not live in the database.** It is an Ed25519 key generated inside your cluster and kept in the `attestkeep-evidence-key` Secret. Rewriting ledger rows needs database write access; re-signing the seals needs cluster access to the Secret. One access 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 without trusting us to say so.



The package re-verifies the whole ledger at the moment it is generated: every sealed record re-hashed, every Merkle seal recomputed, the chain walked, every signature checked. Here: *Intact*, 2 signed seals, 2 records re-verified, 0 not yet sealed. The document's own digest is printed at the end.

Where the guarantees end

This is a tamper-evidence scheme, not a tamper-proofing one. Records written since the newest checkpoint are counted, not protected — a window of at most the seal interval. *Intact* speaks about what is sealed, not about coverage: read it together with the checkpoint count. An attacker with both database write access and cluster access to the signing key defeats the scheme — the claim is that this requires compromising two separately audited surfaces, not that it is impossible. The auditor receives a signed verdict plus the DSSE envelope; independent re-execution of the Merkle verification would require database access.

Signing a package for an auditor produces a PDF alongside the JSON, and both are delivered in one signed archive. Anyone can check a delivered package at attestkeep.com/verify.html, with their own tools, against the published certificate chain.

9. Operations and alerting

Probes

Endpoint	What it checks	Used as
`/healthz`	Nothing beyond the process being up.	Liveness probe (initial delay 10s, period 20s)
`/readyz`	The database, with a 3-second timeout. Unreachable → 503 and the replica leaves the Service — a replica that cannot read the verdict cache would admit everything.	Readiness probe (initial delay 5s, period 10s)

What a database outage does — and does not do

- Replicas turn unready and leave the Service. They do not crash-loop; they come back the moment the database does.
- With the default `failurePolicy: Ignore`, the API server admits pods unchecked for the duration. On recovery the operator finds the heartbeat gap and writes it to the outage record, and the next reconciliation sweep names every digest that entered without a decision.
- Admission never fails closed because of the database alone: quota metering admits on a database error rather than blocking deploys.
- Scanning stops with the database and resumes with it — the queue is in the database, so nothing is lost, only delayed.

What should page you

Metrics are served on port 9090 at `/metrics`. Series for every label combination exist from startup at zero, so absence of data is distinguishable from absence of denials.

Condition	Signal	Why it matters
Admission is being bypassed	`attestkeep_webhook_distrusted == 1`	While this gauge is 1 under failurePolicy Ignore, deployments are proceeding unchecked. Treat as a page, not a ticket.
Something got in	`attestkeep_runtime_unmatched_digests > 0`	The reconciliation sweep found running digests with no allowed admission behind them.
Database trouble	Pods unready on `/readyz` 503	Your existing kube_pod_status_ready alerting already covers it.
Scanning is falling behind	`attestkeep_scan_queue_depth` sustained high	A queue that only grows means the scanner cannot reach a registry or its database.
Vulnerability data going stale	`attestkeep_vulnerability_db_age_seconds`	Verdicts are only as fresh as the database behind them.
Licence quota approaching	`attestkeep_image_quota{kind="count"}` vs `{kind="limit"}`	Exhaustion denies only image names never seen this period — running workloads keep running.

Availability of the pieces

Operator/webhook: 2 replicas, preferred anti-affinity, PodDisruptionBudget minAvailable 1 — raise `replicaCount`; controller work is serialised through an advisory lock, so scaling out is safe. Scanner: 1 replica; the chart refuses multiple

replicas on a ReadWriteOnce cache volume and tells you why. Bundled PostgreSQL: single replica by design — point `externalDsn` at the PostgreSQL you already operate with replication and backups.

GitOps: enforcement sits at pod admission, so it does not matter who does the applying — kubectl, Argo CD, Flux, a Job. If your GitOps controller tries to sync a workload whose image fails policy, the sync fails with the denial message as the error.

Air-gapped clusters: Trivy and cosign are pinned 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. Licensing needs one route out: lic.attestkeep.com:443, once a day, three values, tolerating up to thirty days of outage on the current certificate.

10. Licensing

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

Activation

Activation happens once per cluster and spends one slot. On the first run the operator generates a keypair inside the cluster, keeps the private half in its own database, and sends only the public half with the activation request. What is sent: the activation key, a cluster fingerprint (a hash of the kube-system namespace UID — not a name and not an address), the cluster name if you set one, and the operator version. What comes back is a signed certificate valid for at most thirty days, verified inside the cluster against a public key built into the operator.

The daily check

Once a day the installation sends three values — licence id, cluster fingerprint, operator version — signed with the keypair from activation. The answer is a status, and a fresh certificate only 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 would add one. However many operator pods run and however often they restart, a cluster makes this call once a day.

Being unreachable is not being denied

If the licence service cannot be reached, the installation keeps running on the certificate it already holds, for that certificate's remaining term — between ten days and a month. Failed checks are retried within hours. After a week without a route the console says so and counts down; before the certificate lapses the administrators are e-mailed. Past thirty days a paid installation falls back to Community Edition: scanning, signature verification, admission enforcement and evidence carry on under the free limits. It does not stop, lock you out, or start denying deployments.

Cluster slots

Each activation holds a slot until it is released. You release one yourself, from your licences page. Reinstalling into the same cluster does not spend a second slot — the fingerprint is the same. If you have run out, the error names the clusters that are holding your slots and when each one activated.

When a licence ends

Expiry and cancellation behave like a certificate that lapsed: the installation returns to Community Edition and keeps working under the free limits. Cancelling mid-term does not switch anything off on the day you cancel; you keep what you paid for until the term ends.

11. Editions and prices

Every security feature is in the free edition. Scanning, signature verification, admission enforcement and evidence packages are not held back, and every policy control — deny until scanned and `failurePolicy: Fail` included — is in Community. What a key buys is scale, and single sign-on.

Edition	Per month	Per year	Clusters	Frameworks	Image names / month	Also
Community	$0	$0, forever	1	1, your choice	15	Public issue tracker
Team	$480	$4,800	1	1	100	Single sign-on, e-mail support
Business	$1,440	$14,400	3	3	200	Everything in Team, priority e-mail support
Enterprise	$3,000	$30,000	5	all eight	to fit your estate	Support terms and onboarding set in the agreement, invoicing and procurement paperwork

A year costs ten months. Add-ons on paid editions: additional cluster +$300 / month, additional framework +$120 / month, 10 more image names a month +$10 / month, uncapped volume +$600 / month — yearly, ten times each. On Community a volume step is +$120 / month and uncapped volume is not sold. Beyond five clusters is a custom agreement rather than a listed price.

Prices exclude VAT. Checkout and payment run on Polar Software, Inc., the merchant of record: the card is entered on Polar's checkout and never reaches Attestkeep's servers, any tax due is worked out there from your billing address, and the invoice is Polar's. A VAT-registered business in the EU accounts for VAT itself under the reverse charge. The licence and the support come from Attestkeep.

There is no time-limited trial, and there is no need for one. Community Edition carries every security feature and never expires, so the question a trial usually answers — does this work on my cluster — is one you can answer on your own cluster, for as long as you like.

The metered unit is the image name: registry and repository, with the tag and digest stripped. Everything running in the cluster counts, third-party images included. When the quota is exhausted, only new image names are denied at admission; every image already counted this period keeps being scanned and enforced.

12. 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 against the public key at docs.attestkeep.com/cosign.pub. The scanner and the signature verifier are pinned into the image rather than pulled at runtime.

Reporting a bug or a vulnerability

Bugs go in 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 a single point of contact for vulnerability reports: it is security@attestkeep.com, and the same details are printed on every evidence package the product generates.

Support by plan

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

Attestkeep does not publish a response-time guarantee for the plans below Enterprise.

Who you are dealing with

Trading name	Attestkeep
Registered name	Arif Hacı Bayram Kurnaz
Legal form	Sole trader (şahıs işletmesi), Türkiye
Registered address	Dörtyol Küme Evleri No: 369, İç Kapı No: 3, Işıklar, Merkez, Çanakkale, Türkiye
Tax registration	Çanakkale Tax Office · Tax no 5940929524
Field of activity	Computer programming (NACE 62.10)
Legal notices, contracts, invoicing	legal@attestkeep.com
Support and pre-sales	support@attestkeep.com
Security reports	security@attestkeep.com
Telephone	+90 541 546 34 58

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 binds.

Appendix — what this handbook was verified against

Product version	Attestkeep 1.0.1 (chart 1.0.1, appVersion 1.0.1), released 6 September 2026
Image	`ghcr.io/attestkeep/attestkeep-k8s:1.0.1` `sha256:a00befd84230ac81db36cd22091e9418251a9ecf6ef9d5ca7f92e4aa40be3f34`
Kubernetes	v1.35.1 (minikube, single node), Kubernetes 1.27+ supported
Installations photographed	A Team-licensed cluster upgraded 0.3.6 → 1.0.0 → 1.0.1 (dashboard, images, vulnerabilities, policies, evidence, audit log, usage); a Community-licensed cluster installed clean at 1.0.1 (activation, account creation, first dashboard, policy editor)
Evidence package shown	`PS-SOC2-HBCHECK-20260901-20260906-E6CDB8` , generated 6 September 2026 09:24 UTC, ledger intact, 2 signed seals
Commands and outputs	Copied verbatim from the installations above and from docs.attestkeep.com
Prices	attestkeep.com/pricing.html as of 6 September 2026, USD, excluding VAT

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