Attestkeep

Manual del producto

Seguridad de las images de container para Kubernetes. Attestkeep escanea las images, verifica sus firmas, decide en la admission si un pod puede arrancar, rechaza lo que su equipo ha puesto en la deny list y guarda un registro firmado de cada decisión. Este manual explica qué se instala, cómo se opera y cuánto cuesta.

Attestkeep 1.4.0 · Chart de Helm `oci://ghcr.io/attestkeep/charts/attestkeep` versión 1.4.0 · manual con fecha del 29 de octubre de 2026.

Las capturas de pantalla se tomaron el 29 de octubre de 2026 en una instalación cuya consola indica la versión 1.4.0, como Enterprise Edition sobre Kubernetes v1.35.1 y con Falco conectado. La consola estaba en inglés, y este manual cita sus etiquetas tal como aparecen en ella. El anexo da más detalles. Los comandos, ajustes y límites proceden de la documentación del producto en docs.attestkeep.com, del chart y de la consola. Nada en este manual es ilustrativo.

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

Contenido

1. Qué es Attestkeep

2. Qué se ejecuta en su cluster

3. Cómo se toma una decisión de admission

4. Instalación, actualización y primer arranque

5. La consola

6. Policies de admission

7. La deny list

8. Eventos de runtime de Falco

9. Comprobar que el control se aplica

10. Paquetes de evidencia y modelo de confianza

11. Operación, exportación al SIEM y copias de seguridad

12. Licencias

13. Ediciones y precios

14. Seguridad, soporte y con quién trata

Anexo. Contra qué se comprobó este manual

1. Qué es Attestkeep

Attestkeep es un operator de Kubernetes. Escanea las images que usan sus workloads, verifica sus firmas y attestations, y responde al API server en el momento de la admission con permitir o denegar, según una policy que usted escribe. Cada decisión queda registrada. Esos registros se convierten en un informe por periodo que un auditor puede comprobar con sus propias herramientas.

Todo se ejecuta dentro de su cluster. Una sola release de Helm instala el operator, su API, la consola y la base de datos en un namespace que es suyo. Los nombres de las images, los hallazgos y la topología del cluster no salen del cluster. La comprobación de licencia envía tres valores una vez al día.

Qué hace

- **Escaneo de vulnerabilidades.** Trivy viene dentro de la image. Los hallazgos se listan por gravedad, con la versión corregida cuando existe.
- **Verificación de firmas.** Verificación de cosign por clave o keyless, y attestations in-toto con una antigüedad máxima. Desde la 1.4.0, una policy puede listar las claves públicas que acepta, y entonces solo cuenta una firma que verifique una de ellas.
- **Control en la admission.** Umbrales por gravedad, reglas de tags y de registries, fijación por digest, exigencia de firma y de attestation, quince controles de endurecimiento de workloads, un bloqueo basado en el catálogo Known Exploited Vulnerabilities de CISA, un bloqueo de sistemas base fuera de soporte, un control de procedencia SLSA que comprueba quién firmó, las declaraciones VEX del productor y reglas más estrictas para los pods privilegiados o expuestos.
- **Deny list.** Cuando alguien decide que una image, una CVE, una versión de un paquete o un ejecutable no debe ejecutarse, una sola entrada lo dice. Se consulta antes del break-glass y antes de cualquier regla de la policy, en cada namespace cuya policy aplica la lista. Una segunda herramienta lee los ejecutables dentro de cada image, de modo que también se encuentra un runtime compilado desde el código fuente sin registro de paquete. Un workload en marcha nunca se detiene de forma automática. Un administrador puede hacerle un take down a mano, una vez, y restaurarlo. El capítulo 7 lo explica.
- **Triage con caducidad.** Puede aceptar un riesgo con una justificación y una fecha de fin. El break-glass deja pasar una image concreta durante un incidente. Ambos quedan registrados.
- **Paquetes de evidencia.** Un documento firmado por periodo de auditoría, asignado al catálogo de controles que usted elija. Dice lo que no puede demostrar igual que lo que demuestra.
- **Registro de auditoría.** Cambios de policy, excepciones en la admission, decisiones de triage, exportaciones de evidencia e inicios de sesión en un solo registro.
- **Notificaciones.** Slack, Teams, correo electrónico y un webhook genérico. Una notificación nombra el cluster, nunca su topología.
- **Gestores de incidencias.** Un hallazgo crítico nuevo abre una incidencia en su GitHub, GitLab o Jira, una por digest de image y CVE. La incidencia se cierra con un comentario que da el motivo: corregido, valorado en triage o image retirada. Solo *corregido* cuenta como remediación.
- **Eventos de runtime de Falco.** Su propio Falco envía sus eventos al operator por TLS mutuo, directamente o a través de Falcosidekick. Cada evento se guarda junto a la image y el pod de los que procede. Las policies de runtime que usted escribe en la consola deciden qué ocurre después: dejar constancia, enviar una notificación, abrir una incidencia o detener el pod. Un evento de runtime nunca cambia una decisión de admission. El capítulo 8 lo explica.
- **Exportación al SIEM.** Las admissions rechazadas, las coincidencias y los take downs de la deny list, los eventos de runtime, los cambios de policy y de triage, las descargas de evidencia, los inicios de sesión fallidos y los hallazgos críticos nuevos llegan a su SIEM en el momento en que ocurren: Splunk, Microsoft Sentinel, Elasticsearch, syslog o un colector HTTP genérico. El capítulo 11 lo explica.
- **Inicio de sesión único y roles.** Google, GitHub, GitLab, Okta, Microsoft Entra ID, Keycloak o cualquier proveedor OIDC, en las ediciones de pago. Los roles deciden quién puede cambiar una policy.

Qué no hace

Attestkeep no es un sensor de runtime. No observa llamadas al sistema, flujos de red ni procesos dentro de un container en marcha. Esa detección la hace Falco, y Attestkeep solo recibe lo que Falco informa. Attestkeep no distribuye, instala, configura ni escribe en Falco.

No detiene por su cuenta los workloads en marcha. Una entrada de la deny list y una policy en modo Enforce rechazan los pods nuevos. Detener lo que ya se ejecuta es siempre decisión de una persona, o de una policy de runtime que usted escribió y que el chart permite.

No es una herramienta de postura en la nube y no lee su cuenta de nube ni sus políticas IAM. No escanea Terraform, ni manifiestos, ni la configuración del propio cluster frente a un benchmark. No comprueba si una función vulnerable llega a llamarse, no lee sus repositorios ni abre pull requests de dependencias. docs.attestkeep.com indica qué ejecutar junto a Attestkeep para cada una de estas necesidades.

¿Por qué no montar el control con herramientas libres?

Se puede. Hay buenos motores open source de policies de admission y escáneres dentro del cluster. Si solo necesita bloquear `:latest` y rechazar las CVE críticas, puede hacerlo sin Attestkeep. Lo que suele faltar en un montaje así es el registro que usan las auditorías y los análisis de incidentes:

- **Un registro de decisiones con integridad.** Cada decisión de admission se hashea al escribirse y se sella bajo checkpoints firmados y encadenados. Si se modifica un veredicto en la base de datos, el siguiente documento de evidencia señala el registro.
- **Decisiones ligadas a la policy que las tomó.** Cada registro lleva el hash de la policy tal como era en ese momento.
- **Huecos que se informan.** Los periodos en que el webhook no funcionaba o no pudo verificarse se registran y se imprimen en la evidencia. La conciliación lista lo que entró durante esos periodos.
- **Un solo producto.** Los resultados de escaneo, la verificación de firmas, la admission, la deny list, las excepciones y el paquete de evidencia se mantienen juntos, y su equipo no tiene que mantener el pegamento entre ellos.

2. Qué se ejecuta en su cluster

Un webhook de admission tiene voz en cada pod. Este es el inventario de lo que eso supone.

Objeto	Por defecto	Qué hace
Deployment `attestkeep`	2 réplicas, anti-afinidad preferente entre nodes, PodDisruptionBudget minAvailable 1	El operator: webhook de admission (8443), consola y API (8080), métricas (9090) y el receptor de integraciones de Falco (8444) cuando `integrations.falco.enabled` está activado. El trabajo del controlador se serializa con un advisory lock de PostgreSQL, así que las réplicas escalan sin lease de elección de líder. Desde la 1.4.0, una réplica que se detiene sale del Service antes de dejar de responder: su hook preStop espera `webhook.preStopSleepSeconds` (5 por defecto) y luego termina las revisiones en curso.
Deployment `attestkeep-scanner`	1 réplica	Ejecuta los escaneos de Trivy, la verificación de cosign y, cuando la deny list lo necesita, la comprobación de ejecutables `attestkeep-binscan`, todo fuera del camino de la admission. Nunca recibe tráfico de admission y, desde la 1.4.0, nunca contacta con el servidor de licencias.
StatefulSet `attestkeep-postgresql`	1 réplica, 20 GiB, postgres:16.15-alpine	La base de datos incluida. Si apunta `externalDsn` a su propio PostgreSQL, este objeto no se crea.

Todos los containers del producto se ejecutan sin root (uid 65532), con el sistema de archivos raíz de solo lectura, sin capabilities y con seccomp RuntimeDefault. Las requests por defecto son 100m de CPU y 256Mi por workload del producto.

Los objetos de ámbito de cluster son cuatro CRD del grupo `attestkeep.com` (ImageSecurityPolicy, ImageSecurityReport, AttestationRenewal, ImageSecurityUsage) y una ValidatingWebhookConfiguration. El webhook cubre CREATE y UPDATE de pods, incluido el subrecurso `pods/ephemeralcontainers` en el que escribe `kubectl debug`, con un timeout de 15 segundos y `failurePolicy: Ignore` por defecto. No hay webhook de mutación. El producto nunca reescribe sus workloads.

Permisos que conviene conocer

- El ClusterRole lee los nodes (get, list). Así cuenta el operator los nodes Ready frente al tope de nodes de la licencia. Guarda solo el número, nunca el nombre de un node.
- Los Secrets se pueden leer en todo el cluster solo con `get`, no con `list`. El operator puede leer un pull secret cuyo nombre encontró en la spec de un pod. No puede listar lo que contiene un namespace.
- `pods/eviction` solo se concede con `rbac.allowEnforce: true` o `rbac.denyList.takedown: true` (ambos false por defecto). Una policy de runtime que detiene pods y `runtimeReconciliation: enforce` necesitan `rbac.allowEnforce: true`, conceda lo que conceda el ClusterRole. Sin él se rechazan y se registran. El permiso de take down sirve solo a la deny list.
- El subrecurso scale de Deployments, StatefulSets y ReplicaSets (get, update) solo se concede con `rbac.denyList.takedown: true`. Permite a un administrador llevar un workload a cero réplicas y restaurarlo. Puede cambiar cuántas réplicas se ejecutan, nunca qué ejecutan.
- Desde la 1.4.0, los Deployments, ReplicaSets, StatefulSets, DaemonSets, Jobs y CronJobs se pueden leer (get, list) en toda instalación. El operator lee la cadena de propietarios sobre un pod, de modo que un rechazo se cuenta contra el Deployment y no contra su ReplicaSet. Es solo lectura.
- El operator crea eventos de Kubernetes: un Warning en el workload que un rechazo de la deny list detuvo, y en una policy con una clave de firma que no es una clave pública PEM. También escribe el estado de sus propias policies, donde está la condición `SignatureKeysValid`.

- Un Role de namespace permite escribir tres Secrets con nombre en el namespace del operator: el certificado del webhook, `attestkeep-evidence-key` y `<release>-falco-mtls`, que solo existe con la integración de Falco activada. Desde la 1.4.0, `attestkeep-evidence-key` es también la identidad de licencia de la instalación (capítulo 12).
- Un segundo Role, `<release>-rule-validation`, permite al operator crear, leer y borrar pods y ConfigMaps en su propio namespace y leer los logs de los pods. El editor de reglas de Falco lo usa para lanzar un pod de validación puntual. No puede arrancar pods en ningún otro namespace.

Todas las conexiones salientes

Use esta tabla para su solicitud de firewall. No hay SDK de analítica, de informes de fallos ni de telemetría.

Destino	¿Necesario?	Ajuste	Si está bloqueado
lic.attestkeep.com:443	Sí, salvo activación sin conexión	`license.serverUrl`; `license.caBundle` para una CA privada o un proxy que intercepta TLS en el camino	La activación en línea no es posible. Una instalación activada sigue funcionando con su certificado y pasa a Community cuando caduca. Desde la 1.4.0, solo una réplica del operator hace la comprobación diaria.
ghcr.io (base de vulnerabilidades de Trivy):443	Para escanear	`trivy.dbRepository`, para un espejo	La base existente sigue funcionando y envejece. `attestkeep_vulnerability_db_age_seconds` indica cuánto.
Sus registries de images:443	Para escanear	Los pull secrets de las specs de los pods	Los escaneos de esas images fallan. La admission sigue el ajuste de la policy para images sin escanear.
ghcr.io/attestkeep/attestkeep-k8s:443	Al instalar	`image.repository`, para un espejo	Los pods no pueden arrancar.
Sigstore TUF y Rekor público:443	No	`scan.cosignOffline: true`	Con firma por clave y modo sin conexión, la verificación es local.
Fuente de versiones	No, desactivada por defecto	`updates.endpoint`	Nada. Una instalación sin dirección configurada nunca consulta.
Sus destinos de SIEM	No, los configura usted	Settings > SIEM export	Los eventos esperan y se reintentan. Un evento que sigue sin entregarse a las 24 horas se abandona y se cuenta en la página.
Su colector OTLP, sus URL de Slack, Teams o webhook, su servidor SMTP, su proveedor de SSO	No, los configura usted	Según la función	Esa función falla y lo registra. Un fallo de notificación nunca bloquea la admission.

Si nuestro servidor de licencias se viera comprometido

No podría dar órdenes a su cluster. El protocolo de licencia tiene dos tipos de respuesta: un certificado y un estado de renovación. Ningún camino del código permite que una respuesta del servidor elija una policy, un namespace, una regla de image, una URL o un programa que ejecutar. Los certificados se firman con Ed25519, se verifican con una clave pública compilada en el binario y están ligados a la huella de su cluster.

Un servidor hostil podría responder *revoked*. La instalación pasaría entonces a una admission que solo avisa, lo que se ve en la consola y en cada respuesta de admission. No podría colocar un workload ni cambiar una policy. Si el servidor no responde, no pasa nada: los errores de red cuentan como una caída, y la instalación sigue con su certificado hasta que caduca.

3. Cómo se toma una decisión de admission

El webhook responde desde la caché. El escaneo nunca se ejecuta en el camino de la admission, así que un registry lento no retrasa un despliegue.

1. El API server pregunta por un pod: al crearlo o actualizarlo, y cuando `kubectl debug` añade un container a un pod en marcha.
2. El tag de la image se resuelve a un digest. El resultado se guarda un minuto en caché, de modo que una ráfaga de despliegues idénticos hace una sola llamada al registry.
3. El resultado del escaneo y el veredicto de firma del digest se leen de la caché.
4. Primero se comprueban las reglas que no necesitan llamar al registry: la lista de registries permitidos, `:latest` y la exigencia de digest.
5. Se consulta la deny list, antes del break-glass y antes de cualquier otra regla de la policy. Una image en la lista se rechaza aunque su escaneo esté limpio, e incluso con un break-glass abierto (capítulo 7).
6. Se evalúa la policy. La respuesta es permitir o denegar, con un motivo.
7. La decisión queda registrada, incluidas las que permiten. Una evidencia que solo lista rechazos no puede demostrar que un control funcionaba.

Ninguno de estos pasos espera a un escaneo. Si un digest nunca se ha visto, el paso tres no tiene respuesta y la policy decide qué ocurre.

Los eventos de Falco no forman parte de este camino. Un evento de runtime nunca cambia si un pod se admite.

Dry runs. Desde la 1.4.0, `kubectl apply --dry-run=server` se decide y se responde como lo haría una petición real y no deja nada atrás: ni fila en el registro, ni coincidencia de la deny list, ni evento de Kubernetes, ni notificación, ni métrica, nada contado en la cuota de images y ningún escaneo en cola.

Cuando la base de datos no responde. Una revisión de admission concede dos segundos en total a su trabajo en la base de datos. Después, las reglas estáticas deciden al momento: `:latest`, la lista de registries permitidos y la exigencia de digest siguen rechazando. El registro y los veredictos que dependen de la caché se degradan, y la caída se registra una vez. Antes de la 1.4.0, una réplica salía del Service mientras la base de datos no era alcanzable, y se perdían respuestas de admission. El capítulo 11 da los detalles.

Containers de depuración. Desde la 1.3.7, un container añadido con `kubectl debug` también pasa por el webhook. Solo se decide sobre el container añadido. Su image se escanea y se somete a la policy, y una policy en modo Enforce lo rechaza como a cualquier otra image. Los containers que el pod ya tenía no se revisan de nuevo. Una image de depuración no cuenta para la cuota mensual de images ni para el tope de nodes, y el motivo de su fila en el registro termina en `; subresource ephemeralcontainers`. Antes de la 1.3.7, un container así nunca se enviaba al webhook, fuera cual fuera su image.

Los dos ajustes que fijan su postura

Ajuste	Por defecto	Qué significa
`webhook.failurePolicy` (chart)	`Ignore`	Qué hace el API server cuando el operator no responde. `Ignore` admite los pods sin control durante una caída. `Fail` deniega hasta que el operator responde.
`admissionTiming` (policy)	`AllowAndScan`	Qué ocurre con un digest visto por primera vez. `AllowAndScan` lo admite y pone un escaneo en cola. `DenyUntilScanned` lo rechaza hasta que exista un escaneo. Es el compañero habitual de `Fail`.

Ninguna de las dos opciones sirve para todos. `Ignore` mantiene el cluster desplegable durante una caída del operator y deja un hueco en la evidencia. El paquete de evidencia informa de ese hueco. `Fail` da un control sin huecos, y entonces

una caída del operator detiene los despliegues. Los dos ajustes están en todas las ediciones, Community incluida.

Lo que entró durante un hueco sigue ejecutándose después. La conciliación de runtime lo encuentra. Según un calendario (cada hora por defecto), compara el identificador de image de cada container en marcha con el registro de admission. Un digest sin admission detrás se informa, se notifica o, con `runtimeReconciliation: enforce`, se detiene para que vuelva a pasar por el webhook.

> **El primer escaneo de un digest nuevo se hace en segundo plano.** En un cluster nuevo puede tardar unos minutos, porque el escáner tiene que descargar la image y la base de vulnerabilidades aún no está en caché. Hasta que termina, la image aparece como *Unscanned* en la página Images. Las admissions posteriores de ese digest usan el informe terminado. *Rescan* lanza un escaneo a petición.

4. Instalación, actualización y primer arranque

Antes de empezar

- Kubernetes 1.27 o posterior.
- Permiso para crear una ValidatingWebhookConfiguration, es decir, cluster-admin en la primera instalación.
- Unos 20 GiB para el PostgreSQL incluido, o el DSN de un PostgreSQL que ya opere. La exportación al SIEM necesita PostgreSQL 13 o posterior. El incluido es la versión 16.
- Una cuenta de Attestkeep en attestkeep.com y una clave de licencia: la clave gratuita de Community o la del plan que compró.

Verificar la release antes de instalar

Cada release se firma sobre su digest. La comprobación por clave solo necesita su registry y la clave pública de docs.attestkeep.com/cosign.pub.

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

Instalar

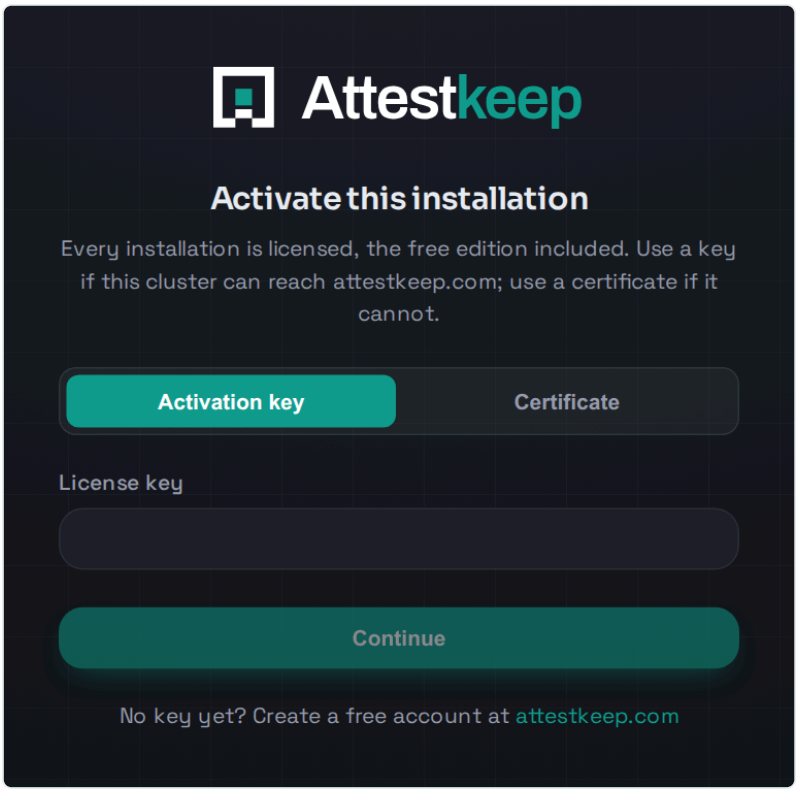
```
# instala desde el registry; no hace falta `helm repo add`
helm install attestkeep oci://ghcr.io/attestkeep/charts/attestkeep \
  --version 1.4.0 \
  --namespace attestkeep --create-namespace \
  --set clusterName=production-eu \
  --wait
```

Fije `clusterName` en la primera instalación, aunque sea de prueba. Las notificaciones, los documentos de evidencia, el panel y Settings > About lo usan para nombrar la instalación. Sin él recurren al nombre del primer node. Es un nombre que usted elige. Ningún nombre de host ni dirección sale del cluster.

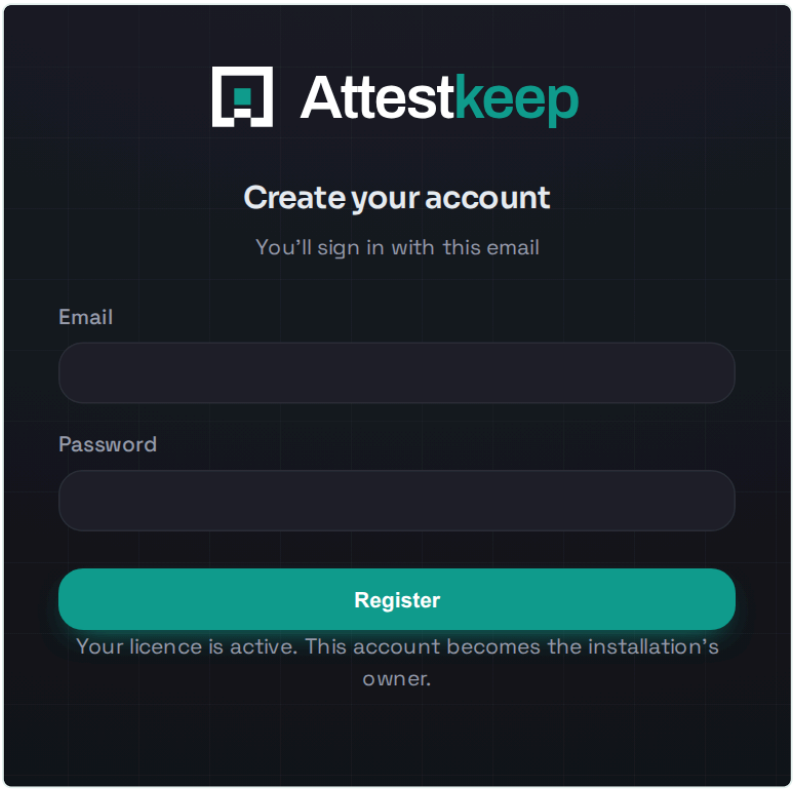
Comprobar que funciona y abrir la consola

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

Abra `http://localhost:8080`. Una instalación nueva muestra una sola vez dos pantallas de configuración: la activación y, después, la cuenta owner. Luego la consola abre en el resumen de seguridad que describe el capítulo 5.



Pantalla 1, *Activate this installation*. Toda instalación tiene licencia, la edición gratuita incluida. En la pestaña *Activation key* pega su clave y elige *Continue*. La clave se cambia por un certificado firmado que el operator comprueba dentro del cluster. Un cluster que no llega a attestkeep.com usa en su lugar la pestaña *Certificate* (capítulo 12).



Pantalla 2, *Create your account*. La línea bajo *Register* confirma que la licencia está activa y que esta cuenta pasa a ser la propietaria de la instalación. La pantalla se cierra en cuanto la cuenta existe. Si compró un plan después de instalar con una clave Community, active la clave de pago más tarde en la página Usage and License.

Si la activación no llega al servidor de licencias, la pantalla lo dice y da la causa. Un certificado en el que el operator no confía (una CA privada sin `license.caBundle`), una conexión rechazada o agotada y un nombre que no se resuelve muestran todos "The licence server could not be reached: its certificate is not trusted, or the connection failed. If it uses a

private CA, set license.caBundle in the chart." Antes de la 1.4.0, esos mismos fallos parecían una clave no válida. El capítulo 12 recoge los mensajes.

Usar su propia base de datos

Las migraciones vienen dentro del binario y se aplican en la primera conexión. Cada una se registra y se ejecuta una sola vez. Para usar una base de datos que ya opera, ponga el DSN en un Secret en lugar de en la línea de comandos:

```
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
```

Los registries privados no necesitan configuración. El operator usa, para el escaneo y para la comprobación de firma, los pull secrets que referencia el pod y los de su service account.

Desinstalar es `helm uninstall`. La configuración del webhook se elimina con la release, así que la revisión de admission se detiene en ese momento. El volumen de la base de datos no se borra.

Actualizar

Una actualización es `helm upgrade`. Las migraciones integradas aplican lo nuevo y omiten el resto. `helm upgrade --reuse-values` funciona entre estas versiones: los valores añadidos desde su versión instalada toman el valor por defecto del chart. Las migraciones solo avanzan. Donde un paso de la tabla lista migraciones, volver a la versión anterior significa restaurar una copia de la base de datos, no `helm rollback`.

De → a	Migraciones	Qué hacer
1.3.7 → 1.4.0	0036 a 0046	Haga esta actualización concreta con `--set webhook.failurePolicy=Fail` y vuelva después al valor anterior, o hágala durante una congelación de cambios. Una réplica 1.3.7 se detiene sin vaciarse, y durante cerca de un segundo una petición que recibe se admite sin control bajo `Ignore`. Desde la 1.4.0, cada actualización vacía las réplicas. Cada migración espera como mucho cinco segundos por su bloqueo, y luego la réplica se reinicia y lo vuelve a intentar. Volver a la 1.3.7 es restaurar una copia. Los valores nuevos del chart son todos opcionales, y el take down sigue desactivado hasta que fije `rbac.denyList.takedown: true`. El ClusterRole gana lectura sobre los objetos de workload por encima de un pod. Los canales de notificación existentes no se suscriben a *Deny list hit*: márquelo donde quiera que se avisen los rechazos. Una base de datos externa debe ser PostgreSQL 13 o posterior para la exportación al SIEM. Una instalación activada sin conexión antes de la 1.4.0 necesita una nueva solicitud de activación para que funcionen las descargas de evidencia firmada (capítulo 12).
1.3.6 → 1.3.7	0033 a 0035	Actualice con `helm upgrade`, no cambiando solo la image: la nueva regla del webhook para `pods/ephemeralcontainers` está en el chart. Desde esta versión, `kubectl debug` con una image que la policy no permite se rechaza. 0033 añade la tabla que guarda las ejecuciones de What if, 0034 añade dos columnas a la tabla de images y 0035 las rellena desde el registro. Volver a la 1.3.6 es restaurar una copia. No hay valores nuevos en el chart.
1.3.2 → 1.3.6	Ninguna	No hay nada que hacer. 1.3.3, 1.3.4, 1.3.5 y 1.3.6 no traen migraciones ni valores nuevos en el chart.
1.3.1 → 1.3.2	Ninguna	Nada para el operator. `helm rollback` a la 1.3.1 funciona. Si copió los valores de falcoctl del editor de reglas de la 1.3.1, vuelva a añadir `falco-rules:5` a `install.refs` y `follow.refs`. Sin él, falcoctl deja de instalar el `/etc/falco/falco_rules.yaml` propio de Falco.
1.3.0 → 1.3.1	0026 a 0032	El recurso de policy ya no tiene el campo `spec.runtime`. Quítelo de sus archivos de policy si lo había puesto: `kubectl apply` rechaza una policy que aún lo tenga con `unknown field "spec.runtime"`. La migración 0031 convierte las reglas de runtime guardadas con la acción eliminada en notificar con incidencia. Los canales de notificación nuevos se suscriben a Runtime alert. Los existentes conservan sus suscripciones.
1.2.x → 1.3.0	0025	Vuelva a activar el cluster. Un operator 1.3.0 o posterior necesita un certificado que lleve el tope de nodes y rechaza uno anterior con "re-activate with a key issued by licence service 1.3.0 or later".
1.1.x → 1.2.0	0021 a 0024	Los eventos de Falco enviados a través de Falcosidekick con un bearer token dejan de llegar hasta que Falco se conecte por TLS mutuo (capítulo 8). Borrar un usuario ahora desactiva la cuenta.

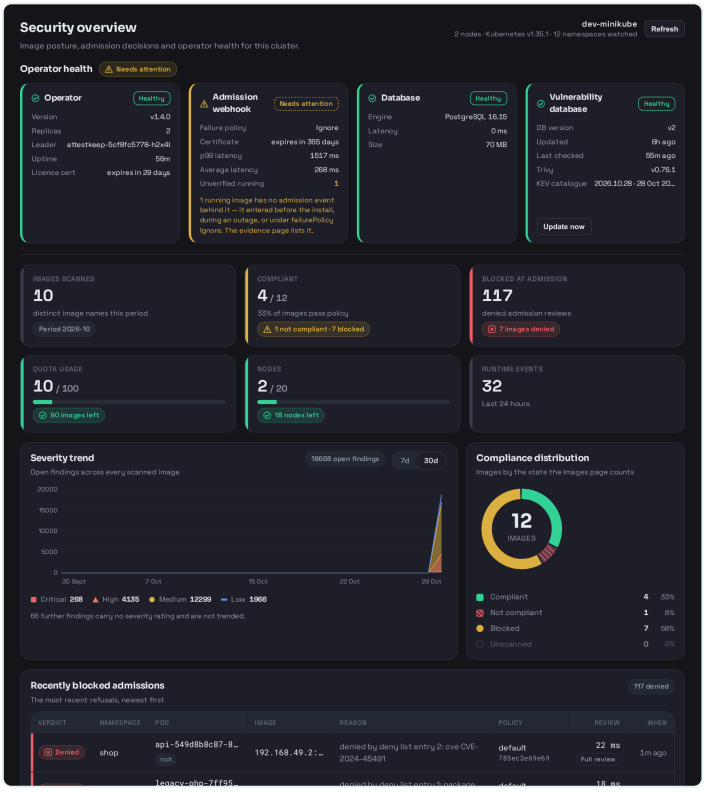
Una instalación que gestiona su propio RBAC añade el Secret `<release>-falco-mtls` al Role antes de activar la integración de Falco.

5. La consola

La consola viene con la release. La misma release de Helm que instala el operator la sirve. No carga fuentes, scripts ni analítica de fuera. Está en inglés y en turco, con tema claro y oscuro y un panel de ayuda en cada pantalla. Este manual muestra la interfaz en inglés y nombra menús y botones tal como aparecen en ella.

La barra lateral tiene cuatro grupos. *Security* contiene Images, Vulnerabilities, Policies, Deny list y What-if history. *Falco* contiene Policies, Rules, Events y Falco integration, y aparece cuando la integración de Falco está activada. *Compliance* contiene Compliance evidence y el Audit Log. *Settings* contiene Organisation, Appearance, My Notifications, Webhooks, Scanning, SMTP, Single Sign-On, Logging & telemetry, SIEM export, Usage and License, Users, Backup y About.

El resumen de seguridad



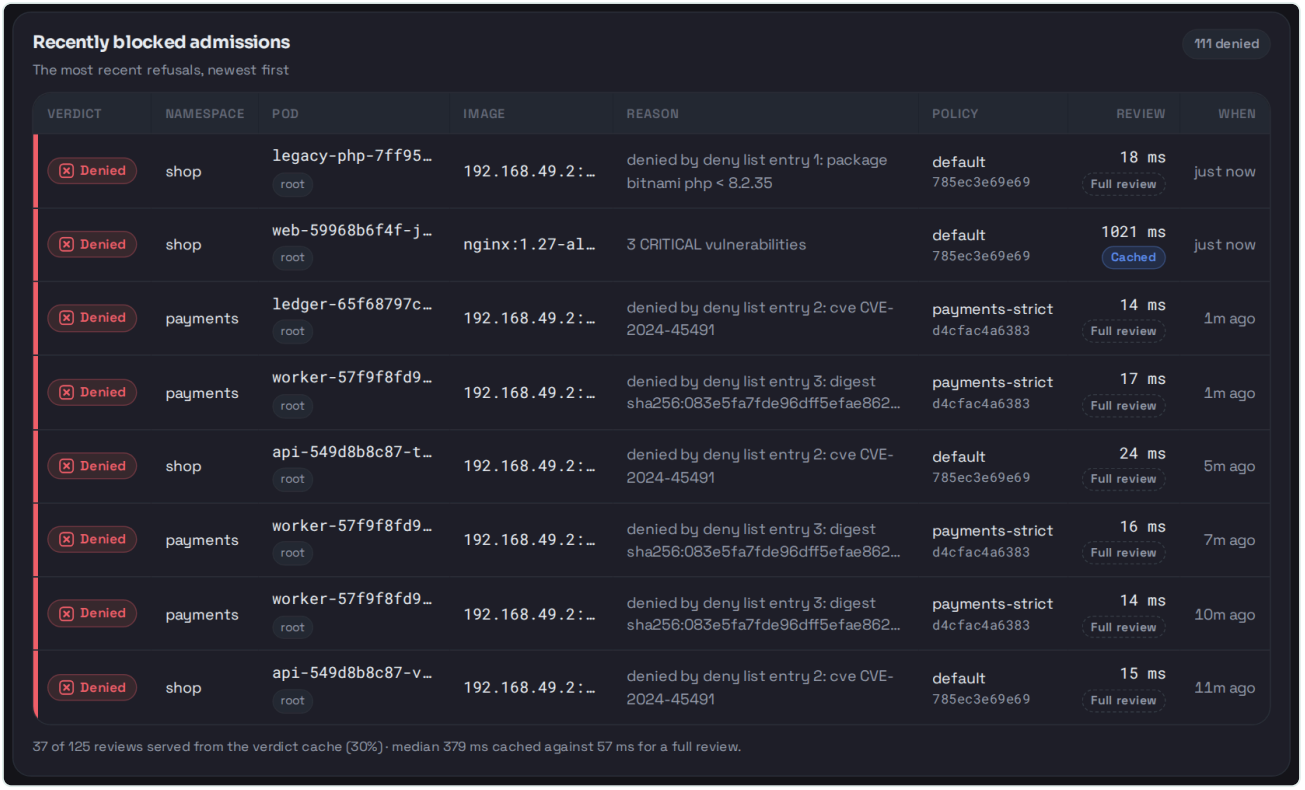
La vista de seguridad de la instalación fotografiada para este manual: versión 1.4.0 con dos réplicas, dos nodes, Kubernetes v1.35.1. El panel *Admission webhook* pide atención porque una image en ejecución no tiene registro de admission detrás, y el panel explica cómo puede ocurrir. Debajo: 10 nombres de image escaneados en el periodo, 4 de 12 images conformes, 1 no conforme y 7 bloqueadas, 117 revisiones de admission denegadas, *Quota usage* 10 de 100, *Nodes* 2 de 20 y 32 eventos de runtime en las últimas 24 horas. La distribución de cumplimiento cuenta los mismos cuatro estados que la página Images.

Desde la 1.3.5, las dos tarjetas ya no lanzan un aviso por sí solas. La tarjeta del webhook no compara la latencia con un presupuesto fijo, porque la primera revisión de una image resuelve su digest en el registry y tarda más que las siguientes, que responden desde la caché. El operator renueva la base de vulnerabilidades y el catálogo KEV cada seis horas. Un administrador puede pulsar *Update now* para descargar en el acto. La tarjeta dice entonces si la base cambió, ya estaba al día o por qué falló la descarga, y la petición se escribe en el registro de auditoría como `vulnerability_db_refresh`.

La tarjeta Nodes llegó en la 1.3.1. Mientras el cluster supera su tope, la franja indica desde cuándo y la fecha a partir de la cual se rechazarán nombres de image nuevos. Pasada esa fecha, indica que el control está activo. En ningún caso se detiene lo que ya se ejecuta.

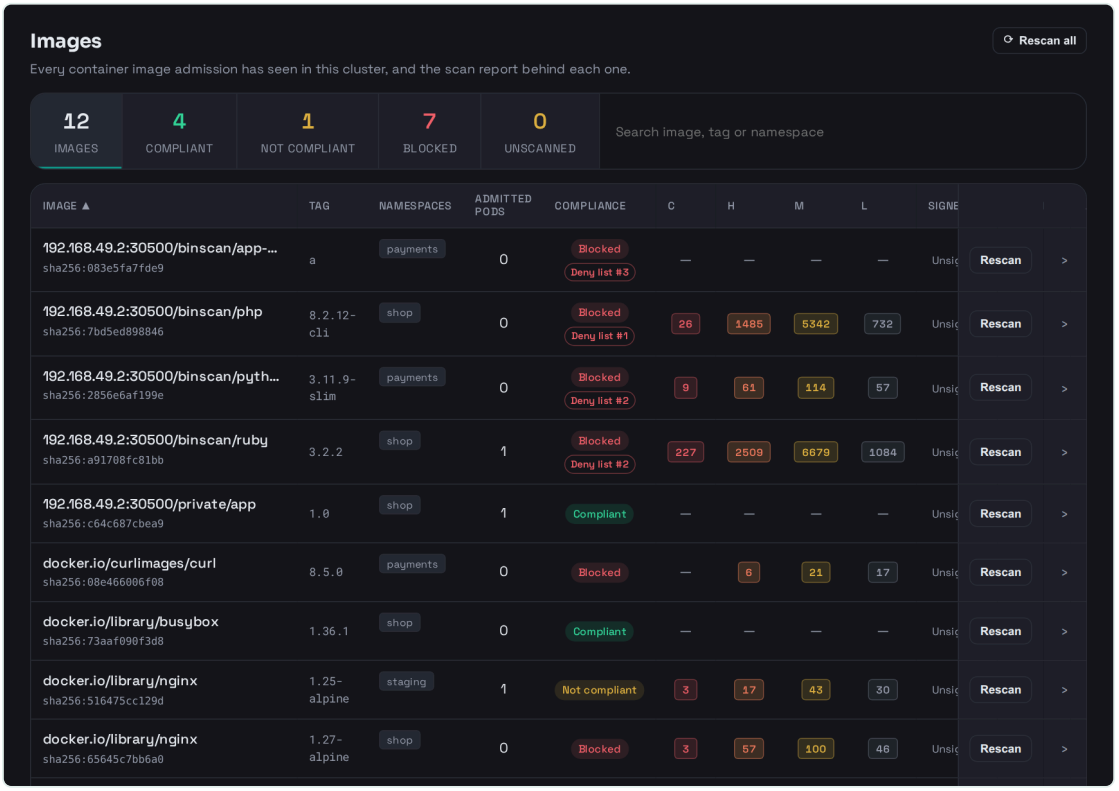
Desde la 1.4.0, la tarjeta de cumplimiento, sus etiquetas y la distribución cuentan los mismos cuatro estados que la página Images, con la misma regla, y las dos páginas dan las mismas cifras. Una image cuya última admission se rechazó es Blocked aunque su escaneo cumpla la policy, y una admitida bajo break-glass es Not compliant. Las porciones Break-glass

y Unknown han desaparecido. La *Severity trend* cuenta los hallazgos abiertos de todas las images registradas, las filas de la página Vulnerabilities, y su título es la suma de las cuatro bandas de debajo. *Leader* indica la réplica que tiene el bloqueo del controlador.



Recently blocked admissions lista solo rechazos, el más reciente primero. Un rechazo de la deny list nombra su entrada, por ejemplo *denied by deny list entry 1: package bitnami php < 8.2.35*. Un rechazo por las reglas de una policy indica la regla, aquí 3 CRITICAL vulnerabilities. Cada fila muestra la policy y su hash, el tiempo de revisión y si el veredicto salió de la caché. La línea bajo la tabla compara las revisiones servidas desde la caché con las completas.

Images



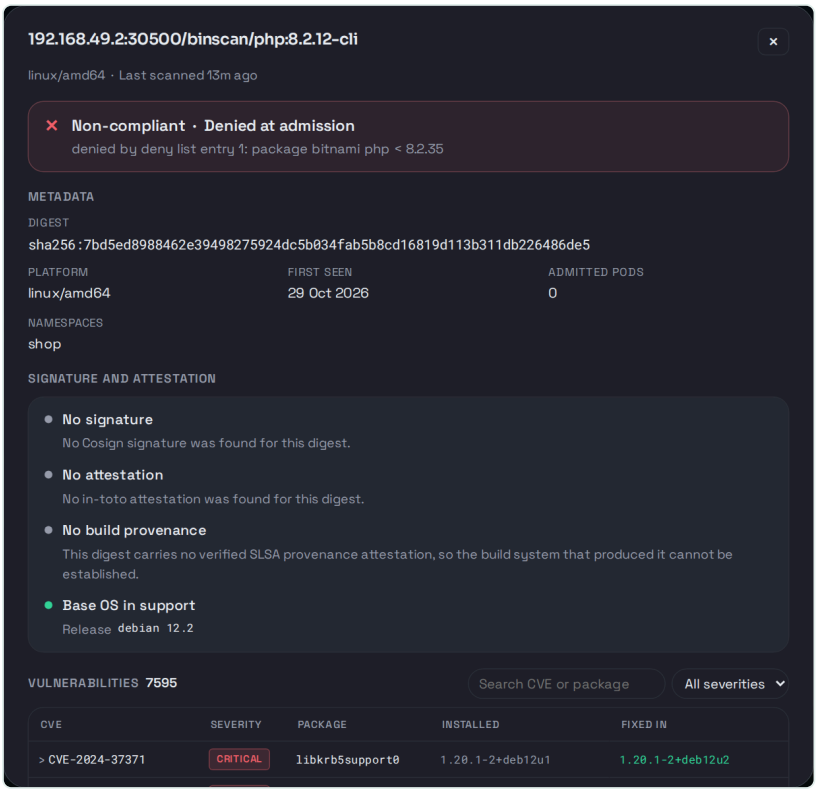
Todas las images que la admission ha visto en este cluster: 12 images, 4 conformes, 1 no conforme, 7 bloqueadas y ninguna sin escanear, así que los paneles suman la lista. Una fila rechazada por la deny list lleva *Deny list #N* bajo su estado. Las columnas dan los namespaces, los pods admitidos, los hallazgos por severidad y el estado de la firma. *Rescan* pone en cola el escaneo de una image y *Rescan all* el de todas.

Desde la 1.3.7, cada image está en uno solo de cuatro estados, de modo que las tarjetas suman la lista. Gana el primer estado que se cumple:

- **Blocked**: su última revisión de admission se rechazó.
- **Unscanned**: admitida, todavía sin escaneo.
- **Compliant**: admitida, escaneada y dentro de la policy.
- **Not compliant**: admitida y escaneada pero fuera de la policy, normalmente bajo una policy en modo Audit o durante un break-glass.

Blocked sigue a la última revisión de admission, no al veredicto al que llega el escáner después. Una image admitida bajo una policy en Audit antes de su primer escaneo, en la que luego aparecen hallazgos críticos, es Not compliant, y su detalle dice "Admitted; the next admission would be denied". Desde la 1.4.0, la etiqueta Compliance de cada fila nombra su estado con las mismas palabras, los filtros seleccionan por él y el veredicto propio del escaneo está en el detalle de la image. Una fila rechazada por la deny list lleva la insignia *Deny list #N*, con el rechazo al pasar el ratón. La columna *Admitted pods* cuenta los pods admitidos con la image desde que se vio por primera vez. No baja cuando un pod se detiene, y una admission rechazada nunca se cuenta.

Desde la 1.3.1, las listas Images y Vulnerabilities se leen por páginas. Los filtros, la búsqueda y el orden se ejecutan en el operator, y los totales sobre cada tabla cubren todas las filas.

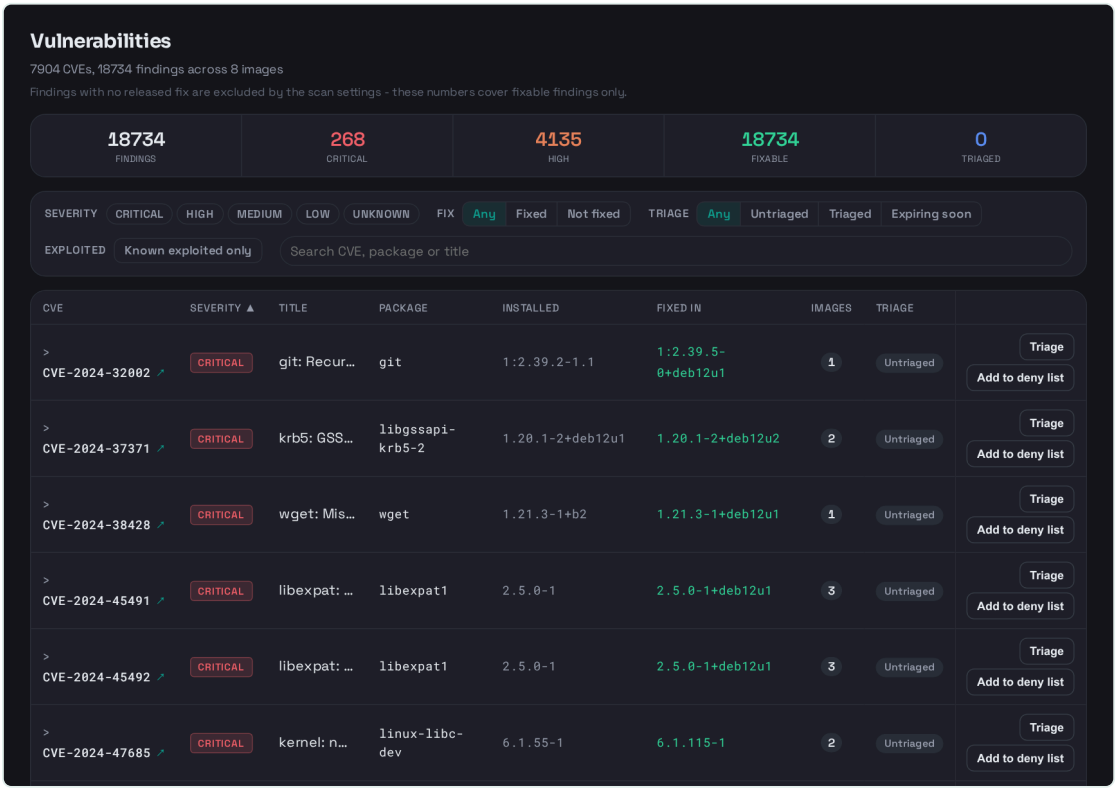


El panel de `php:8.2.12-cli`. La admission la rechazó por la entrada 1 de la deny list, y el panel cita el rechazo. Encima de los hallazgos responde a cuatro preguntas: si hay firma de cosign, attestation in-toto, build provenance SLSA verificada, y si el sistema base sigue con soporte. Aquí Debian 12.2 tiene soporte y el escaneo informa de 7595 vulnerabilidades.

El detalle ofrece *Add to deny list* con el digest de la image ya puesto, y muestra *Listed* para un digest que ya está en la lista. Cuando la deny list tiene una entrada que responde la comprobación de ejecutables, el detalle muestra además sus respuestas para ese digest en *Deny list checks* (capítulo 7).

El mismo escaneo está en el cluster para quien lo lea con kubectl. Cada `ImageSecurityReport` lleva el resumen en `status.summary`, los hallazgos en `status.vulnerabilities` (primero los más graves, 1000 como máximo) y la hora del escaneo en `status.lastScanned`. Una decisión de triage se copia al informe con `triagedBy` igual a `user:<id de cuenta>`, nunca un nombre.

Vulnerabilities y triage

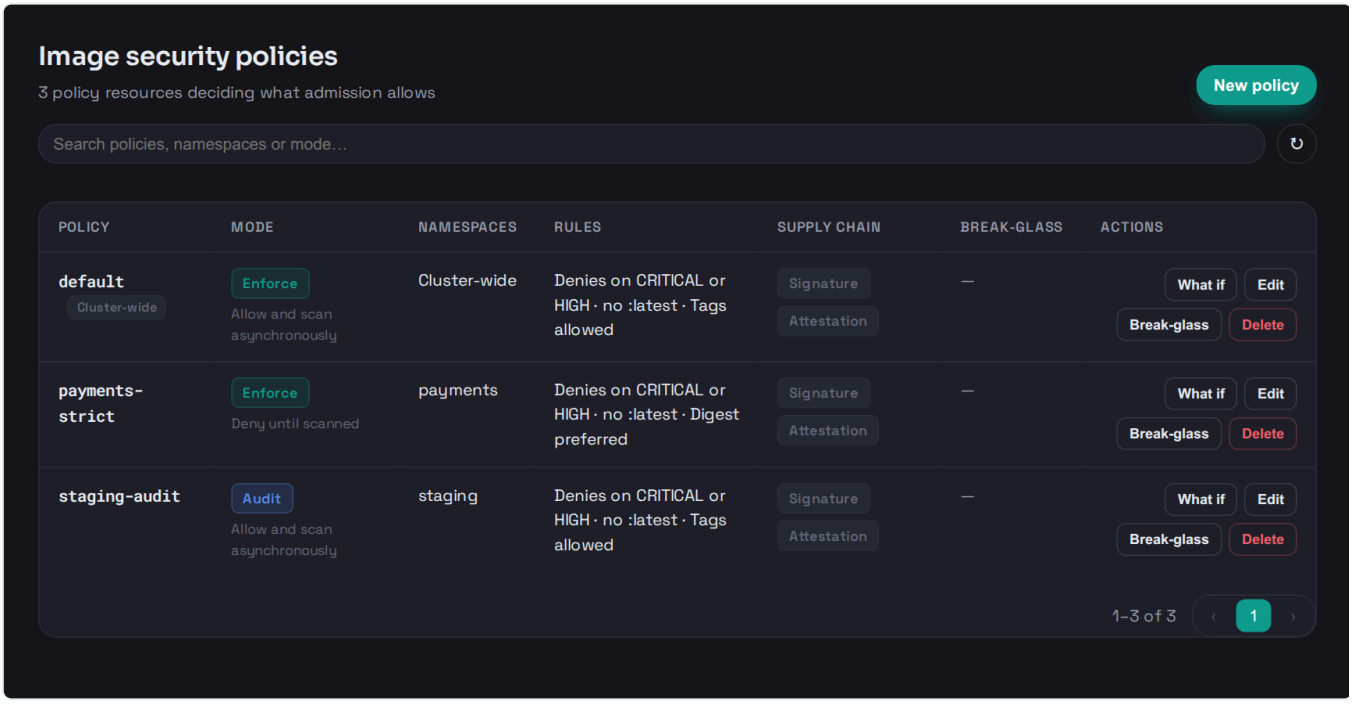


Los hallazgos de todas las images, agrupados por CVE, con la versión instalada y la corregida una junto a otra: 7904 CVE y 18734 hallazgos en 8 images. La línea bajo el título dice que los ajustes de escaneo de esta instalación dejan fuera los hallazgos sin corrección publicada. Cada fila tiene *Triage* y, desde 1.4.0, *Add to deny list*.

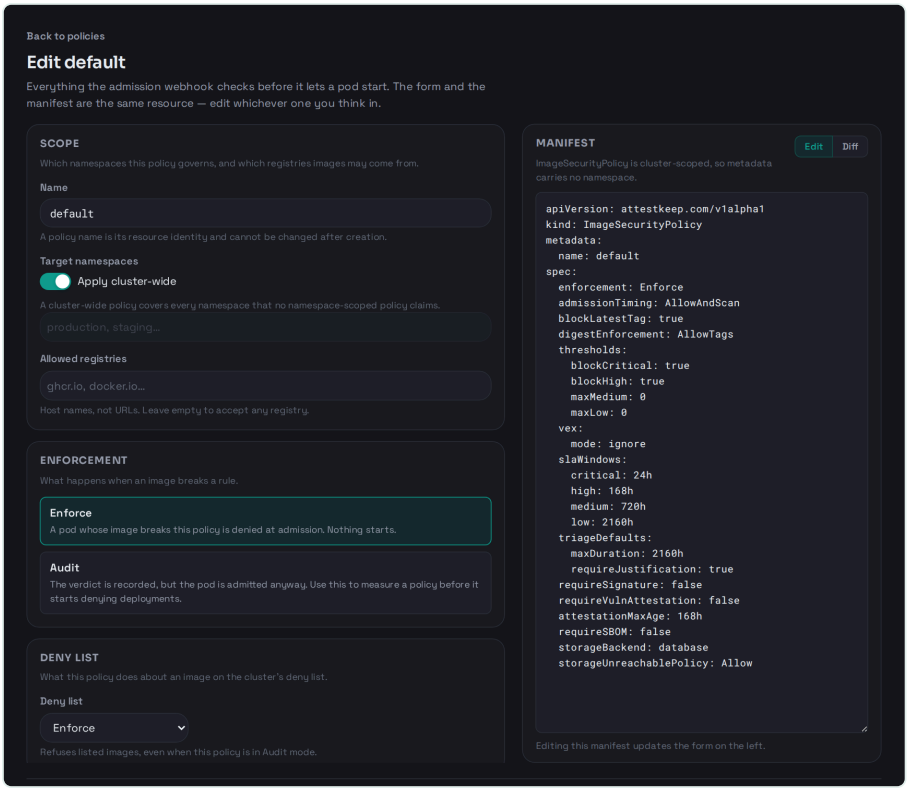
Desde la 1.4.0, el título cuenta todas las CVE que seleccionan los filtros, no solo las 25 de la página, y los hallazgos e images de al lado se cuentan con los mismos filtros. Un digest descargado con dos referencias, su tag y su digest, es una sola image: sus hallazgos cuentan una vez y aparece una vez con todas sus referencias. *Add to deny list* en una fila ofrece el identificador del aviso o, para una entrada de paquete, el paquete y su versión instalada.

El triage es un registro que lee un auditor, así que no puede ser un silencio sin rastro. Usted elige un estado (false positive, accepted risk o under investigation), escribe una justificación y fija una caducidad. Al caducar, el hallazgo vuelve al veredicto de la policy. Cada decisión registra quién la tomó y cuándo, y aparece en el registro de auditoría y en el siguiente paquete de evidencia.

Policies de admission



Tres policies. `default` aplica sus reglas en todo el cluster, `payments-strict` las aplica en `payments` y rechaza mientras no haya escaneo, y `staging-audit` funciona en modo Audit en `staging`. Cada fila muestra el modo, los namespaces cubiertos, un resumen de reglas y las comprobaciones de supply chain exigidas, con *What if*, *Edit*, *Break-glass* y *Delete*.



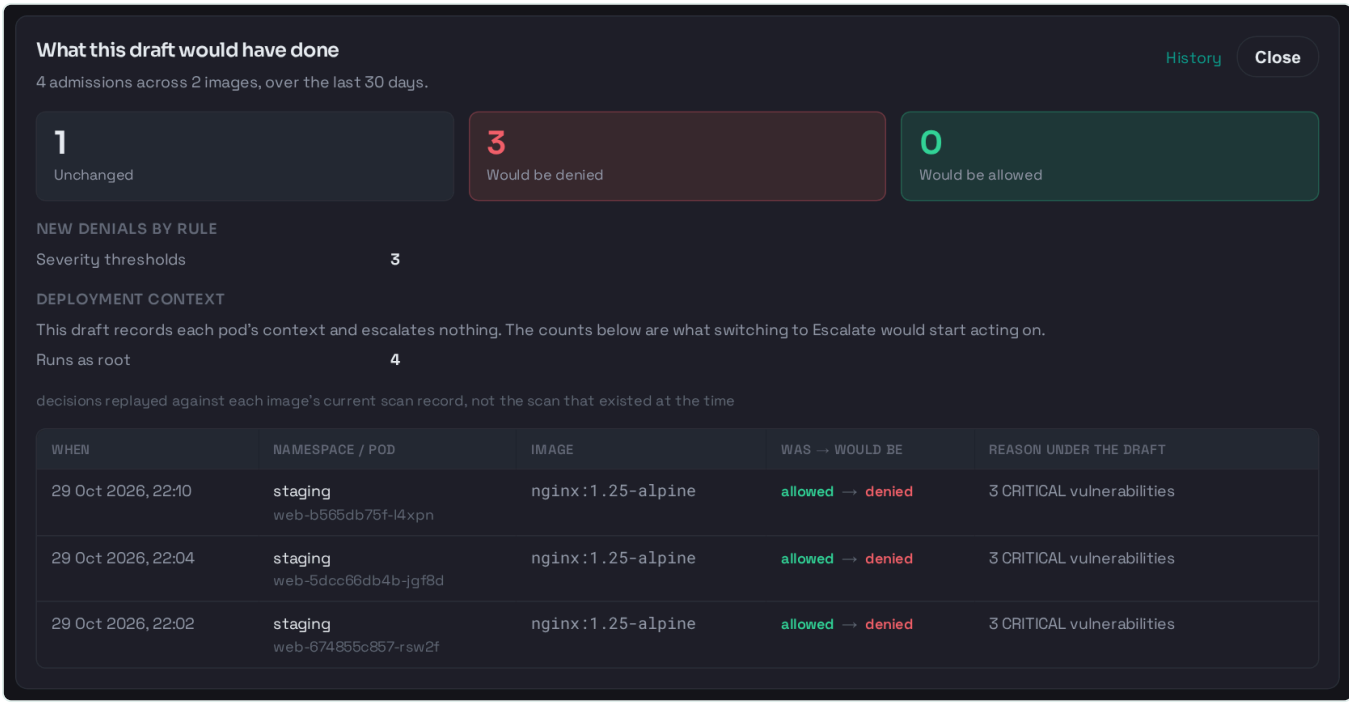
El editor de `default`. El formulario y el manifiesto son el mismo recurso, así que puede editar cualquiera de los dos. La nueva sección *Deny list* está en *Enforce*: las images de la lista se rechazan aunque la propia policy esté en modo Audit.

Desde la 1.3.7, *Apply to cluster* solo aparece para las cuentas que pueden escribir policies. Las demás conservan el resultado de What if, Download y Copy for git. *Cancel* pregunta antes de descartar cambios sin guardar. Desde la 1.4.0, guardar conserva las partes del recurso que el formulario no edita, como el registro de una ventana de break-glass cerrada o un bloque escrito con kubectl. El formulario tiene una sección *Deny list* con *Enforce*, *Audit* y *Off* (capítulo 7). Una policy

nueva empieza con los mismos plazos de SLA y de triage que usan el operator y la documentación (critical 24h, high 168h, medium 720h, low 2160h, triage 2160h). Una policy escrita con kubectl que deja un plazo sin fijar se abre con estos valores, y el diff lo indica antes de aplicar.

Lo que una policy rechazaría en su cluster depende de sus images, escaneos y decisiones de triage, no solo de su texto. *What if* lo responde a partir de los propios registros del cluster. El borrador se ejecuta contra las admissions de los últimos 7, 30 o 90 días con la misma función que usa el webhook de admission. No se aplica nada: no se toca ningún pod, no se contacta ningún registry y el contador de la licencia no se mueve. Las decisiones se repiten con el escaneo actual de cada image, no con el de aquel momento.

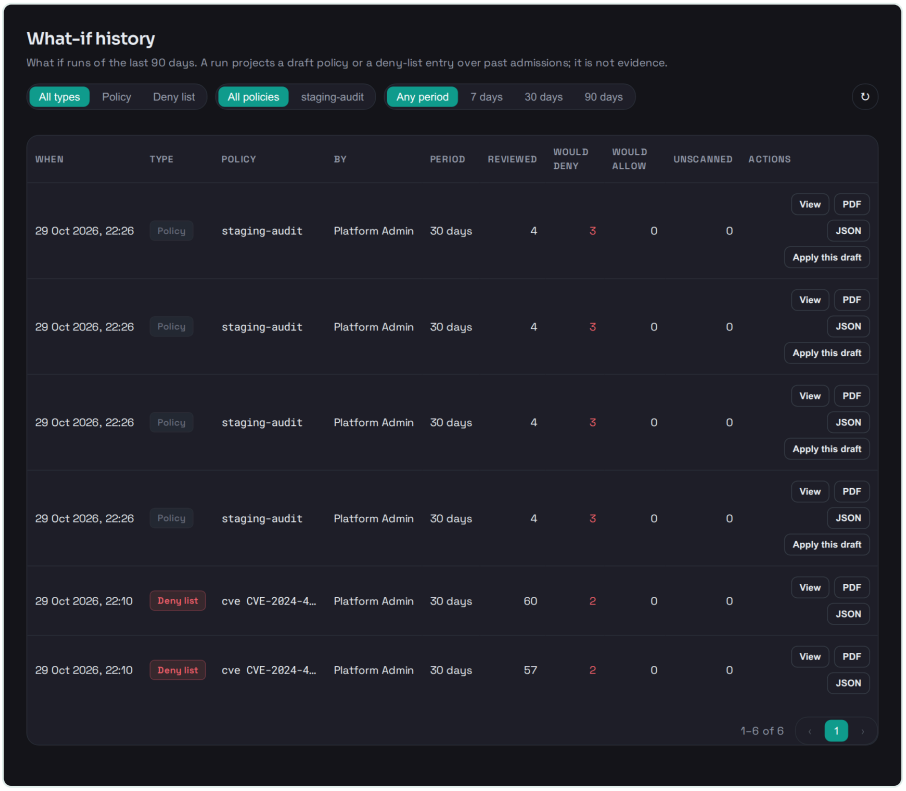
What if en una fila de la lista de policies es el atajo. Una policy en Audit se repite como si estuviera en Enforce, y una en Enforce tal como está, sobre los últimos treinta días. El resultado se abre en el editor y no se aplica nada.



`staging-audit` repetida como borrador aplicado sobre los últimos 30 días: 4 admissions en 2 images, 1 sin cambios, 3 se denegarían por los umbrales de severidad y 0 se permitirían. El panel lista las reglas detrás de las nuevas denegaciones, las señales de deployment context de los pods repetidos (4 se ejecutan como root) y cada admission que cambia con el motivo del borrador.

Scan now. Cuando hay images del periodo sin escaneo, el resultado las nombra. Un deployer o un administrador puede poner en cola escaneos exactamente para esos digests, 500 como máximo por petición, y cada petición se escribe en el registro de auditoría como `rescan`. La repetición no vuelve a ejecutarse sola. Pulse *What if* de nuevo cuando terminen los escaneos.

What-if history



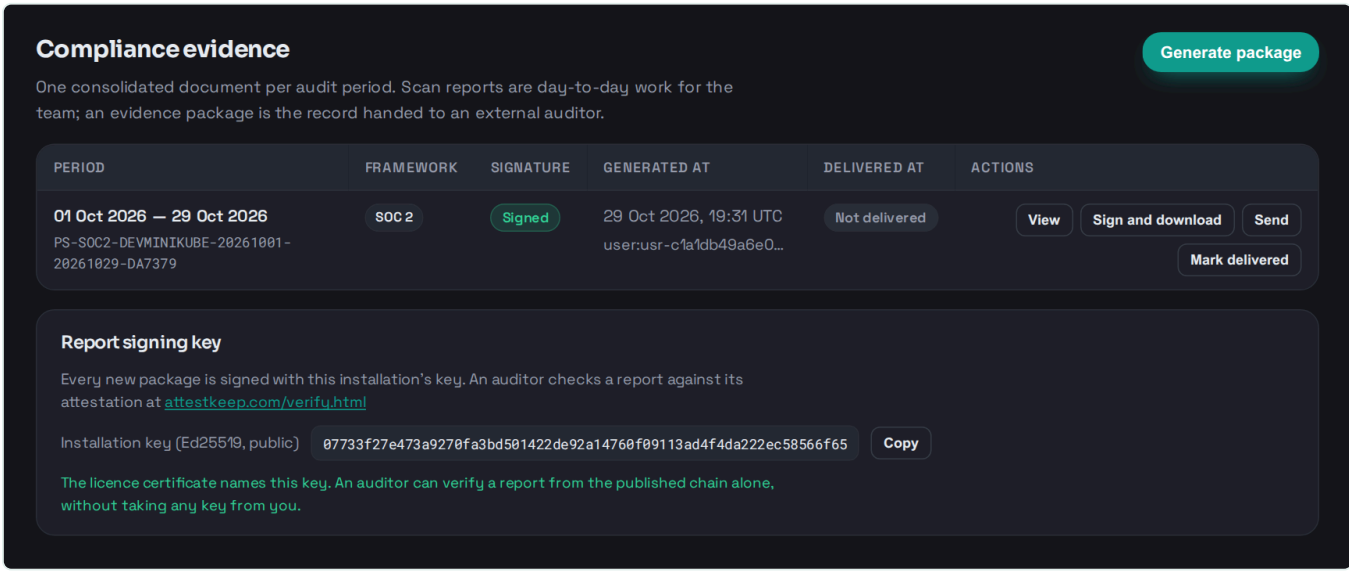
Security > What-if history guarda los dos tipos de ejecución. La columna y el filtro *Type* separan un borrador de policy de una entrada de la deny list: aquí cuatro ejecuciones de `staging-audit` y dos de la entrada para CVE-2024-45491. Cada ejecución muestra quién la lanzó, el periodo y los recuentos. *Apply this draft* solo aparece en las ejecuciones de policy.

Desde la 1.4.0, el historial guarda dos tipos de ejecución: un borrador de policy y una entrada de la deny list probada antes de añadirse o desde su fila (capítulo 7). Una columna *Type* y un filtro por tipo las distinguen. Una repetición lanzada por un deployer o un administrador se guarda noventa días, quinientas ejecuciones como máximo de cada tipo. Un viewer también puede repetir, y la respuesta se muestra pero no se guarda. Cada ejecución puede volver a verse, descargarse en PDF o JSON o, por un administrador, abrirse en el editor con *Apply this draft*. Eso carga el borrador guardado y no aplica nada.

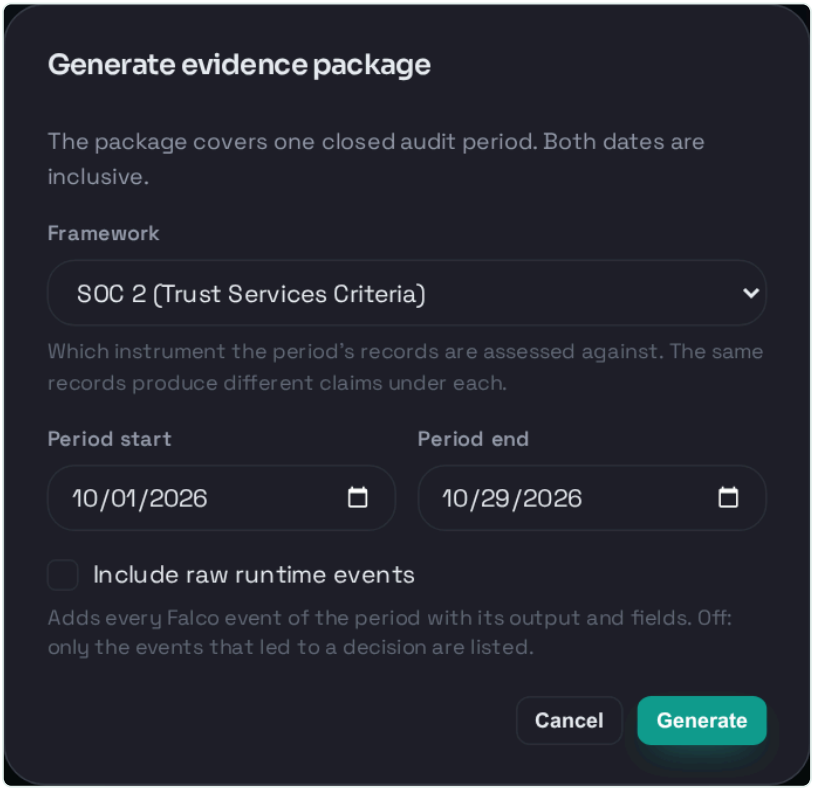
> **Una descarga es una proyección, no una evidencia.** El PDF lo dice en cada página y no está firmado ni sellado. Una ejecución nunca entra en un paquete de evidencia, porque el paquete registra lo que ocurrió y una repetición, lo que habría ocurrido. Cada descarga se escribe en el registro de auditoría como `policy_whatif_download`.

La API es `POST /api/v1/policies/whatif`. Acepta un `source` opcional, la policy desde la que se editó el borrador (un nombre desconocido recibe `400`), y responde con `kept`, `run_id`, `unscanned`, `unscanned_total` y `notes` junto a los totales. Una cuenta puede lanzar como mucho diez repeticiones por minuto en cada réplica del operator. La siguiente recibe `429` y `Retry-After`.

Compliance evidence

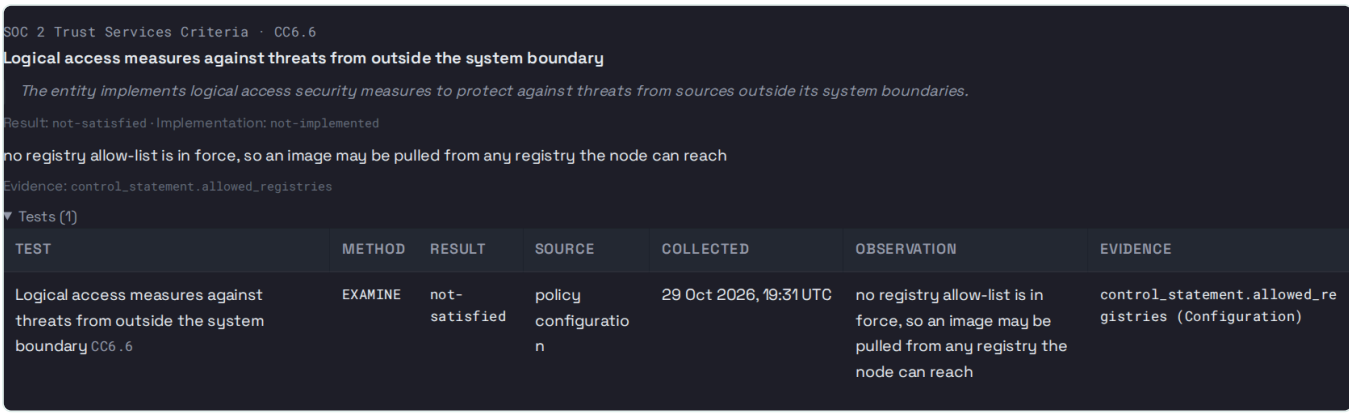


Un documento por periodo de auditoría. La fila da el periodo, el framework, el estado de la firma, cuándo y con qué cuenta se generó el paquete, escrita como `user:<id de cuenta>`, y si se entregó. Debajo, la clave pública de firma de la instalación, con la que un auditor comprueba un informe.



Generate evidence package. La lista *Framework* contiene los conjuntos de controles mapeados que permite la licencia. El periodo es un periodo de auditoría cerrado, ambas fechas incluidas. *Include raw runtime events* añade todos los eventos de Falco del periodo. Si está desmarcada, solo se listan los eventos que llevaron a una decisión.

Cuando una licencia cubre menos frameworks de los que la instalación ya ha usado, por ejemplo al terminar una licencia de pago, el siguiente owner o administrador que inicia sesión debe elegir cuáles conservar. El capítulo 12 da los detalles.



SOC 2 Trust Services Criteria · CC6.6

Logical access measures against threats from outside the system boundary

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

Result: not-satisfied · Implementation: not-implemented

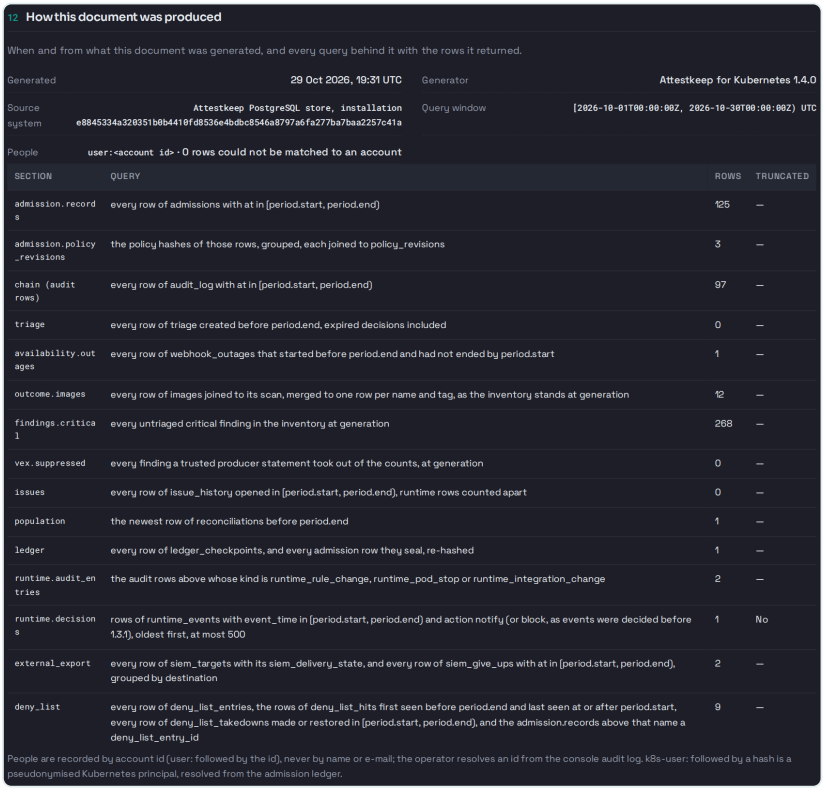
no registry allow-list is in force, so an image may be pulled from any registry the node can reach

Evidence: control_statement.allowed_registries

▼ Tests (1)

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

Un requisito en la vista del informe con su prueba abierta. CC6.6 no se cumple porque no hay en vigor una lista de registries permitidos, y la prueba lo dice. El método es EXAMINE porque lee una configuración. La tabla da también el resultado, la fuente, cuándo se leyó, la observación y la ruta dentro del informe.



12 How this document was produced

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

Generated 29 Oct 2026, 19:31 UTC — Generator Attestkeep for Kubernetes 1.4.0

Source system Attestkeep PostgreSQL store, installation e8845334a320351b0b4410fd8536e4bdbc8546a8797a6fa277ba7baa2257c41a — Query window [2026-10-01T00:00:00Z, 2026-10-30T00:00:00Z) UTC

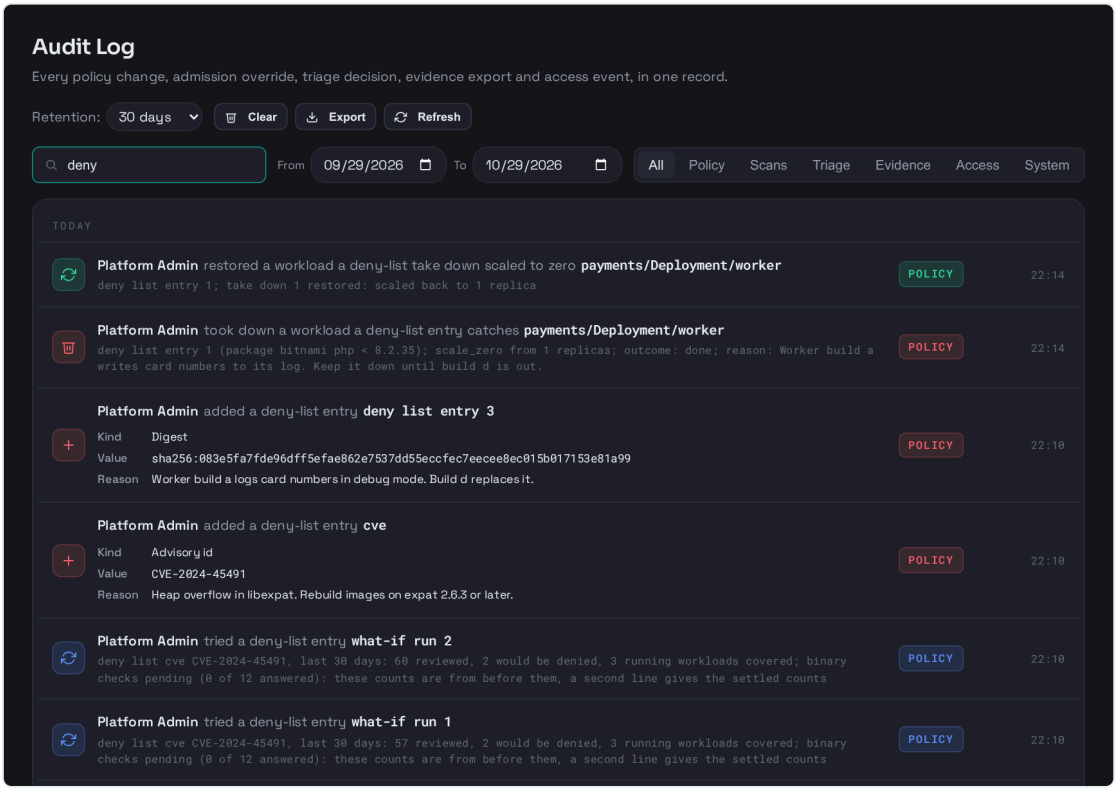
People user:<account id> · 0 rows could not be matched to an account

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

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

How this document was produced en el paquete que muestra este manual, generado por 1.4.0. Da la hora de generación, el generador, el sistema de origen, la ventana de consulta (fin excluido, UTC), cómo se nombra a las personas y la consulta de cada sección con las filas que devolvió. Las dos últimas filas, `external_export` y `deny_list`, son nuevas en 1.4.0.

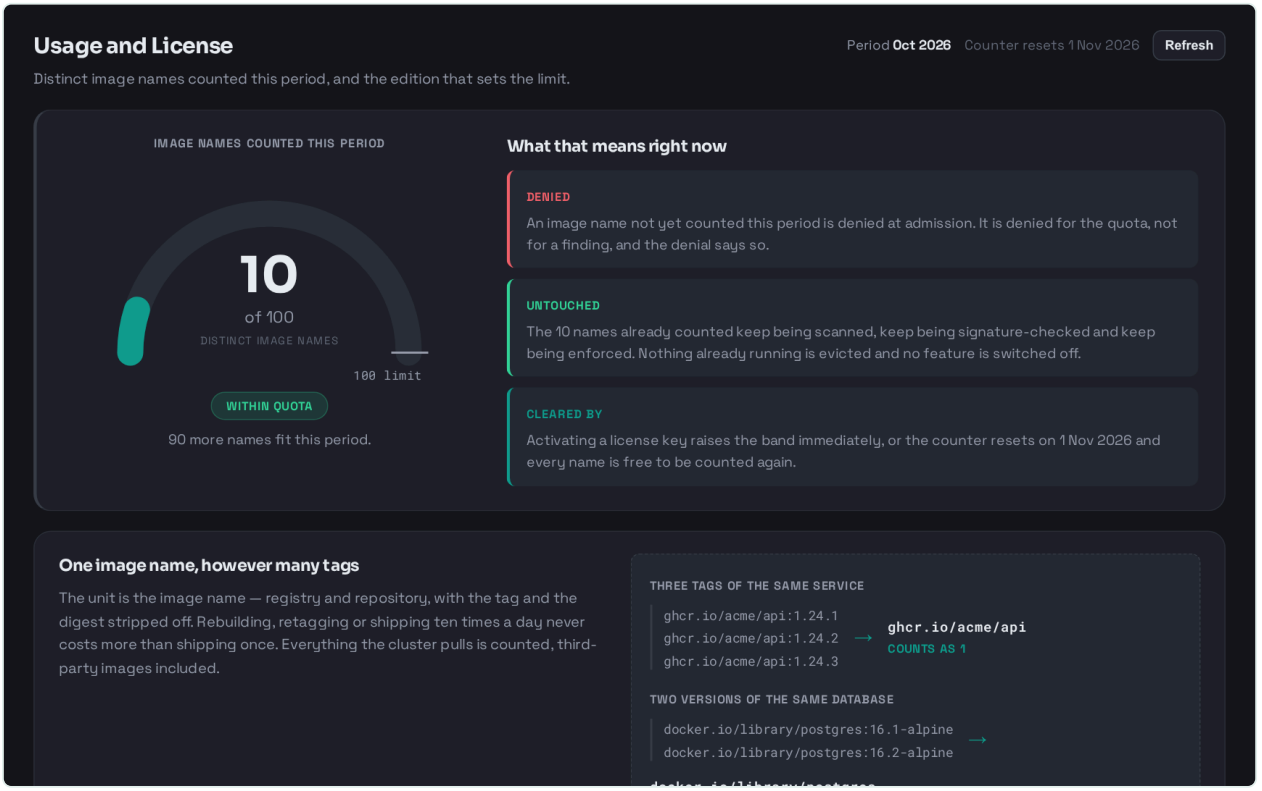
Audit Log



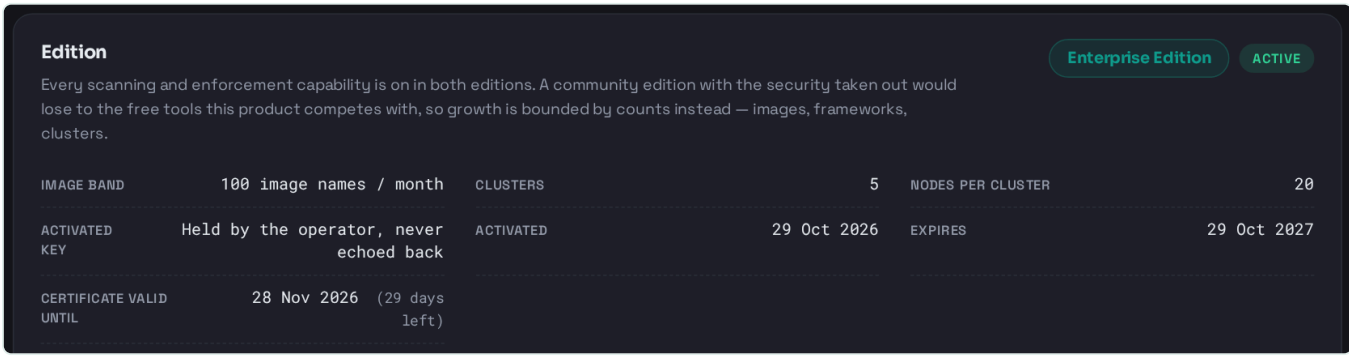
El registro de auditoría buscando *deny*. Arriba, una restauración y el take down anterior, luego dos entradas añadidas con su tipo, valor y motivo, y dos ejecuciones What if de una entrada. Se busca por actor, policy, image o CVE, y se filtra por categoría: Policy, Scans, Triage, Evidence, Access o System.

Las horas se muestran en su hora local, y la exportación CSV conserva sus marcas de tiempo ISO. Aquí quedan registrados los cambios y las pruebas de canales de notificación, los cambios de la integración de Falco y de las policies de runtime, los pods detenidos por una policy de runtime, las acciones de validar, guardar y publicar del editor de reglas, las ejecuciones y descargas de What if, Scan now y Update now. Desde la 1.4.0, también las entradas de la deny list añadidas y retiradas, los take downs, las restauraciones y Check again (`deny_list_add` , `deny_list_remove` , `deny_list_takedown` , `deny_list_restore` , `deny_list_check_again`), y los destinos de SIEM añadidos, cambiados, probados y retirados (`siem_target_add` , `siem_target_update` , `siem_target_test` , `siem_target_remove`). Las acciones sobre canales no entran en los paquetes de evidencia, porque describen lo que alguien hizo en la instalación, no lo que el control decidió sobre una image.

Usage and License



Nombres de image distintos contados en el periodo: 10 de 100, dentro de la cuota, con 90 más antes de que se rechacen nombres nuevos. La página explica qué pasa en el límite. Desde 1.4.0 un nombre rechazado por la asignación no se cuenta, y una franja llena muestra *Allowance reached*.



La sección *Edition* de esta instalación: Enterprise Edition, activa, con 100 nombres de image al mes, 5 clusters y 20 nodes por cluster. La licencia va del 29 de octubre de 2026 al 29 de octubre de 2027. El certificado que tiene el operator es válido hasta el 28 de noviembre de 2026. Un certificado en línea dura como mucho treinta días, y la comprobación diaria de la licencia lo sustituye antes de que caduque (capítulo 12).

La unidad es el nombre de image: registry y repositorio, sin tag ni digest. Reetiquetar o publicar diez veces al día cuesta lo mismo que publicar una vez. Desde la 1.3.1, las cifras del mes en curso de esta página y del panel se leen contra la licencia vigente. Cuando una licencia termina, muestran la cuota de Community, las mismas cifras que aplica la admission.

Desde la 1.4.0, un nombre de image rechazado por la cuota mensual no se cuenta en ella. El contador se detiene en el límite y muestra, por ejemplo, 8 of 8 con *Allowance reached*, donde la 1.3.7 seguía contando los nombres rechazados más allá. El rechazo y su registro de admission no cambian.

Organización y usuarios

Settings > Organisation contiene lo que los paquetes de evidencia dicen de la organización detrás de la instalación. En *Operator declarations* usted declara su fuente de hora y su norma de conservación. El producto no mide ninguna de las dos, y los paquetes las imprimen como declaradas por el operador.



Los dos campos de declaración. En esta instalación están vacíos. El texto gris es el ejemplo que muestra el formulario hasta que usted escribe.

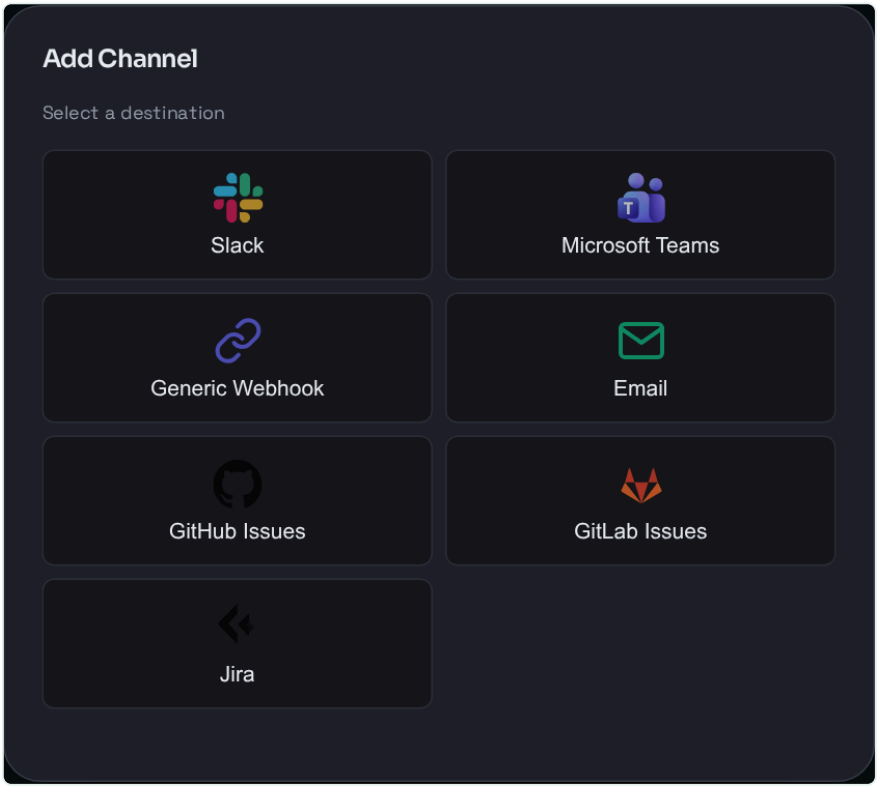
Borrar un usuario desactiva la cuenta en lugar de eliminarla. Una cuenta desactivada no puede iniciar sesión y sus sesiones terminan. Su identificador y sus filas de auditoría se conservan, de modo que cada `user:<id>` de un paquete ya emitido se sigue resolviendo. Ante una solicitud de supresión, un admin puede después aplicar *Anonymise* a la cuenta desactivada. Eso vacía su nombre, su correo y su avatar y reescribe como `user:<id>` cada copia de su nombre en el registro de auditoría, en las decisiones de triage y en las policies de runtime y sus revisiones. Los paquetes ya firmados conservan aquello con lo que se firmaron, y los comentarios ya publicados en un gestor de incidencias no cambian.



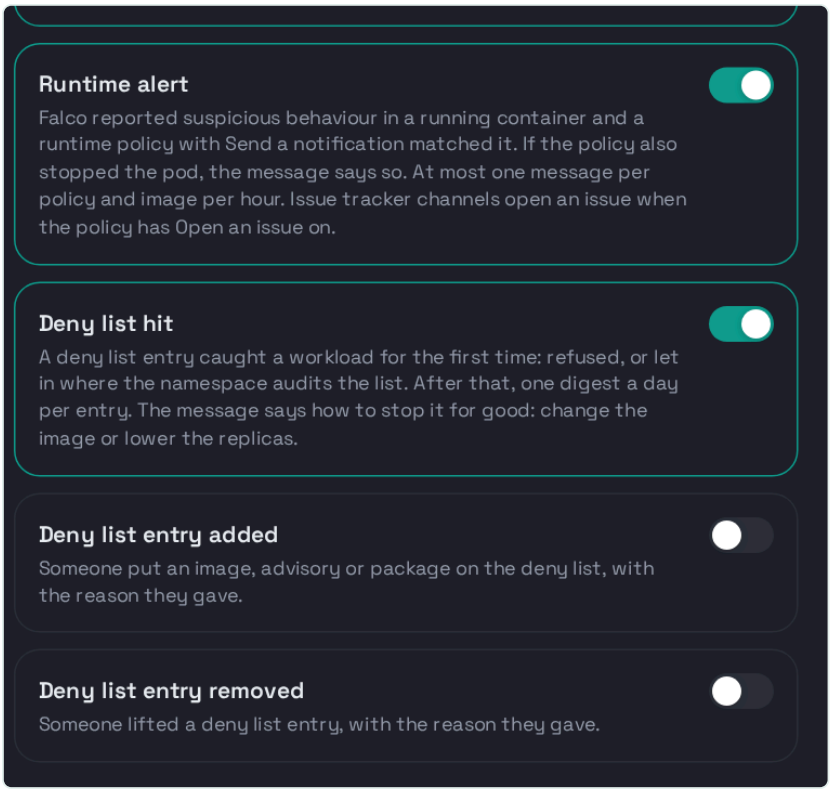
Una cuenta anonimizada sigue en la lista como desactivada, con su id de cuenta donde estaba la dirección de correo. La cuenta desactivada de debajo todavía ofrece *Anonymise*.

Canales de notificación

Los canales se configuran en Settings > Webhooks, en la página *Notification Channels*. Las preferencias de correo de cada persona están en Settings > My Notifications.



Add Channel. Un gestor de incidencias es un canal como los demás, con las mismas suscripciones. *Test* en un canal de ese tipo no escribe nada. Lee una vez el repositorio o el proyecto e indica si la credencial funciona.



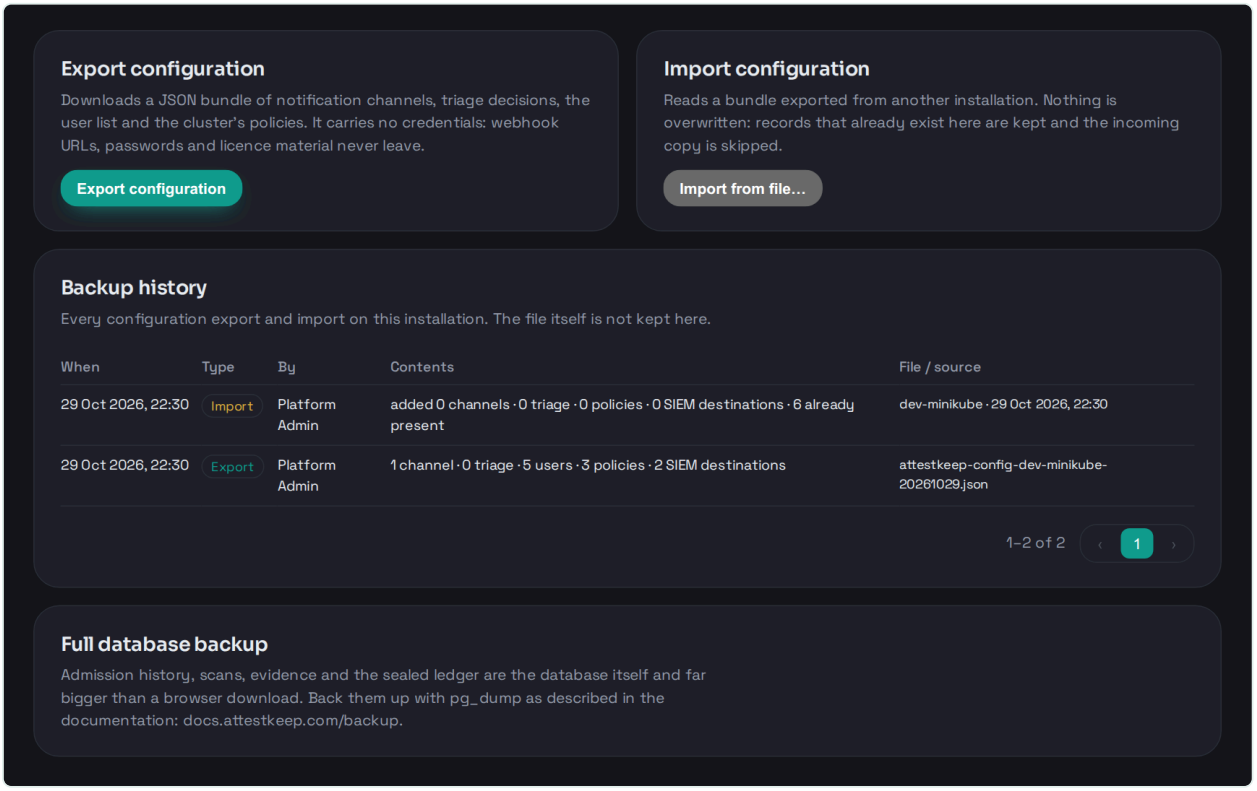
Parte de la lista *Events* de un canal. *Deny list hit* está activado, como en todo canal creado desde 1.4.0: un mensaje la primera vez que una entrada atrapa un workload y después un resumen al día por entrada. *Deny list entry added* y *Deny list entry removed* están desactivados por defecto. *Runtime alert* se describe en el capítulo 8.

Desde la 1.4.0, un canal puede recibir avisos de la deny list. *Deny list hit* se envía la primera vez que una entrada atrapa un workload, con la entrada, su motivo y la salida, y después una vez al día por entrada como resumen. Un canal nuevo lo trae marcado. Un canal creado antes de la 1.4.0 conserva sus suscripciones. Un rechazo de la deny list no se envía además como *Admission blocked*, que es para los rechazos de las reglas propias de una policy. *Deny list entry added* y *Deny list entry removed* están desactivados por defecto.

Cada envío, por correo, webhook, Slack o Teams, tiene 30 segundos desde la conexión hasta el último byte. Un envío que se queda sin tiempo figura en el registro de notificaciones como no entregado con "no answer within 30s". Como mucho se hacen 64 envíos a la vez y esperan 64 notificaciones. Por encima de eso, una notificación se descarta, se registra como no entregada y se cuenta en `attestkeep_notifications_dropped_total`.

El token de un gestor de incidencias es la única credencial que guarda Attestkeep capaz de escribir en un sistema fuera de su cluster. La API nunca lo devuelve. Se cifra con AES-256-GCM antes de guardarse, con una clave que el chart guarda en el Secret `<release>-data-key` y nunca en la base de datos. Fije `api.dataKey.existingSecret` para aportar la clave desde su propio gestor de secretos. Sin la clave, no se puede guardar un canal de incidencias. Nada más depende de ella.

Backup history



Settings > Backup. *Export configuration* descarga un paquete JSON con los canales de notificación, las decisiones de triage, la lista de usuarios, las policies y, desde 1.4.0, los destinos SIEM, sin credenciales. *Backup history* lista cada exportación e importación con cuándo, quién y qué llevaba el paquete. El operator no guarda el archivo. La copia de seguridad de la base de datos corre de su cuenta (capítulo 11).

6. Policies de admission

Un único recurso de ámbito de cluster decide qué puede ejecutarse: `ImageSecurityPolicy`, en el grupo de API `attestkeep.com`. La consola lo edita directamente, y `kubectl apply` lo edita desde un repositorio. Ambos trabajan sobre el mismo objeto. Un cambio se aplica desde la siguiente decisión de admission, sin reinicio.

Cómo obtiene su policy un namespace. Se aplica la policy cuyo `spec.targetNamespaces` nombra el namespace. Si no hay ninguna, se aplica la policy sin `targetNamespaces`, que es la global por defecto. Una configuración habitual es una policy global y otra más estricta para los namespaces que la necesitan.

La policy por defecto

```
apiVersion: attestkeep.com/v1alpha1
kind: ImageSecurityPolicy
metadata:
  name: default
spec:
  enforcement: Enforce          # Audit solo registra lo que habría rechazado
  admissionTiming: AllowAndScan # DenyUntilScanned rechaza lo que no tenga escaneo
  blockLatestTag: true          # un tag mutable puede cambiar lo que corre sin despliegue
  digestEnforcement: AllowTags  # AllowTags | PreferDigest | Required
  denyList: "enforce"           # enforce | audit | "off" (capítulo 7)
  thresholds:
    blockCritical: true
    blockHigh: true
    maxMedium: 0                # 0 aquí significa sin límite
    maxLow: 0
  slaWindows:
    critical: 24h
    high: 168h
    medium: 720h
    low: 2160h
  requireSignature: false       # actívelo cuando su pipeline produzca firmas
  requireSBOM: false
  requireVulnAttestation: false
  attestationMaxAge: 168h
  triageDefaults:
    maxDuration: 2160h
    requireJustification: true
```

Una policy más estricta para los namespaces de producción

```
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: []
```

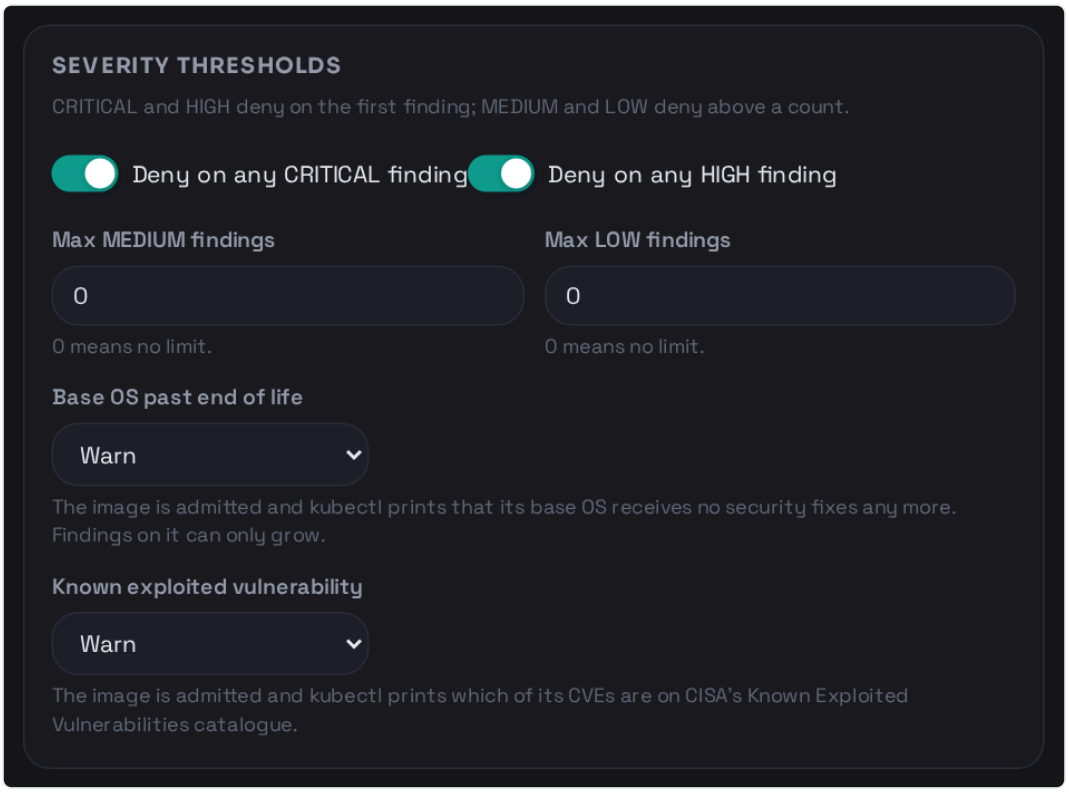
Sistema base sin soporte y CVE explotadas

Dos controles miran algo distinto del número de hallazgos por gravedad. Ambos aceptan `ignore`, `warn` o `block`, y ambos valen `warn` por defecto. Una policy escrita antes de que existieran sigue admitiendo lo que admitía, y ahora además dice por qué no debería.

```
spec:
  # El sistema base ha salido del periodo de soporte de su fabricante y
  # ya no se publicarán correcciones de seguridad. warn (por defecto)
  # admite y lo dice; block rechaza sean cuales sean los totales;
  # ignore no dice nada.
  endOfLifeOS: warn

  # La image lleva una CVE del catálogo Known Exploited Vulnerabilities
  # de CISA. warn (por defecto) admite y nombra las CVE; block rechaza;
  # ignore no dice nada.
  knownExploited: warn
```

El catálogo KEV se guarda como copia local y se renueva cada seis horas. Un cluster sin acceso a internet apunta `kev.source` a un espejo, o lo deja vacío para conservar la copia que tiene. Las CVE valoradas en triage quedan fuera de este control, igual que de los umbrales de gravedad. Un rechazo nombra las CVE, para que quien despliega sepa qué corregir. El panel muestra la versión del catálogo en uso, y los paquetes de evidencia la anotan junto a la versión de la base de vulnerabilidades.



La sección *Severity thresholds* del editor. *Base OS past end of life* y *Known exploited vulnerability* están ambos en Warn, y el texto bajo cada uno dice qué hace ese ajuste.

Claves de firma

Desde la 1.4.0, una policy puede decir por clave de quién acepta la firma. `signatureKeys` lista claves públicas PEM, como su `cosign.pub`. Con `requireSignature: true`, solo cuenta una firma que verifique una de ellas.

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

El escaneo registra al firmante como la huella de la clave, `key sha256:…`, y el paquete de evidencia la nombra. Con `scan.cosignOffline`, la comprobación no necesita el registro de transparencia. Las attestations de SBOM y de vulnerabilidades siguen siendo keyless y no tienen campo de clave. El API server rechaza una entrada que no tenga forma de clave pública PEM. Una que aun así no se pueda leer se ignora y se nombra en tres sitios: la condición de estado `SignatureKeysValid` de la policy, un evento Warning y la página de policies.

Procedencia del build

Este control comprueba qué sistema de build produjo la image y a partir de qué fuente. Es el único control ligado a una identidad. `requireSBOM` y `requireVulnAttestation` comprueban que alguien firmó una attestation. Este control comprueba quién.

```
spec:
  provenance:
    # 0 (por defecto, también si falta el bloque) desactiva el control.
    # Por encima de 0, la image debe llevar una procedencia SLSA verificada
    # que nombre un builder de la lista de abajo, con al menos este nivel.
    minLevel: 2
    trustedBuilders:
      # Un "*" al final compara por prefijo; los demás id, de forma exacta.
      # Una lista vacía con minLevel por encima de 0 no admite nada.
      - id: https://github.com/acme/build/.github/workflows/release.yml@*
        # Su declaración sobre este builder. La procedencia SLSA no lleva
        # nivel, porque un nivel describe cómo se opera el builder.
        level: 3
        # Se lee del certificado de firma. Si falta, debe ser igual al id
        # del builder. Si falta issuer, se acepta cualquier emisor.
        signer: https://github.com/acme/build/.github/workflows/release.yml@*
        issuer: https://token.actions.githubusercontent.com
```

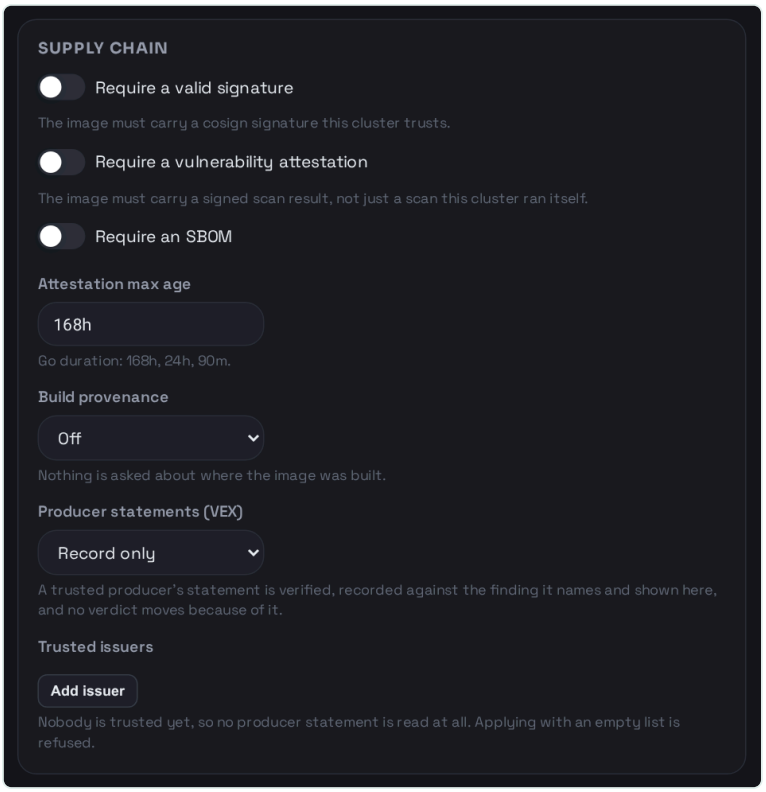
Una image se rechaza si su procedencia SLSA verificada (v1 o v0.2) no nombra un builder de confianza, si la identidad o el emisor del certificado de firma no es el que acepta la entrada, o si el nivel del builder está por debajo de `minLevel`. El motivo nombra el builder, el firmante o el nivel. Ambas identidades se leen del certificado guardado con la attestation. Si ese certificado no se puede obtener o leer, la image se rechaza.

Declaraciones del productor (VEX)

Un escáner informa de lo que hay instalado en una image. No puede saber si el código vulnerable es alcanzable o si llega a ejecutarse. Quien construyó la image sí puede, y OpenVEX es el formato para decirlo. Attestkeep solo lee una declaración VEX cuando llega como attestation in-toto firmada sobre la image y la policy nombra a su firmante. Las declaraciones sin firmar, las de firmantes que la policy no nombra y las que tratan de otro producto se registran y se ignoran.

```
spec:
  vex:
    # ignore (por defecto): las declaraciones de confianza se verifican,
    # se anotan en el hallazgo y se muestran en la consola; ningún
    # veredicto cambia.
    # apply: una declaración de confianza not_affected o fixed quita el
    # hallazgo de los totales con los que decide la admission.
    mode: apply
    trustedIssuers:
      # Un firmante keyless, por identidad del certificado. Un "*" al final
      # compara por prefijo. issuer es opcional y se compara de forma exacta.
      - identity: https://github.com/acme/images/*
        issuer: https://token.actions.githubusercontent.com
      # O un firmante por clave, por clave pública. Una entrada tiene una
      # identidad o una clave, no ambas.
      - key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...
          -----END PUBLIC KEY-----
```

`affected` y `under_investigation` nunca suprimen un hallazgo. Un hallazgo suprimido sale de los totales que usan los umbrales de gravedad y la regla de CVE explotadas, y el motivo de la admission anota cuántos hallazgos se suprimieron y por quién. `mode: apply` con una lista `trustedIssuers` vacía se rechaza, porque parecería un control sin hacer nada. Un cambio de modo se aplica desde el siguiente pod. Un emisor nuevo se aplica desde el siguiente escaneo del digest.



La sección *Supply chain*. Firma, attestation de vulnerabilidades y SBOM no se exigen, *Build provenance* está en Off y *Producer statements (VEX)* en *Record only*, el valor por defecto. Todavía no hay ningún emisor de confianza, así que no se lee ninguna declaración.

Dónde se va a ejecutar el pod

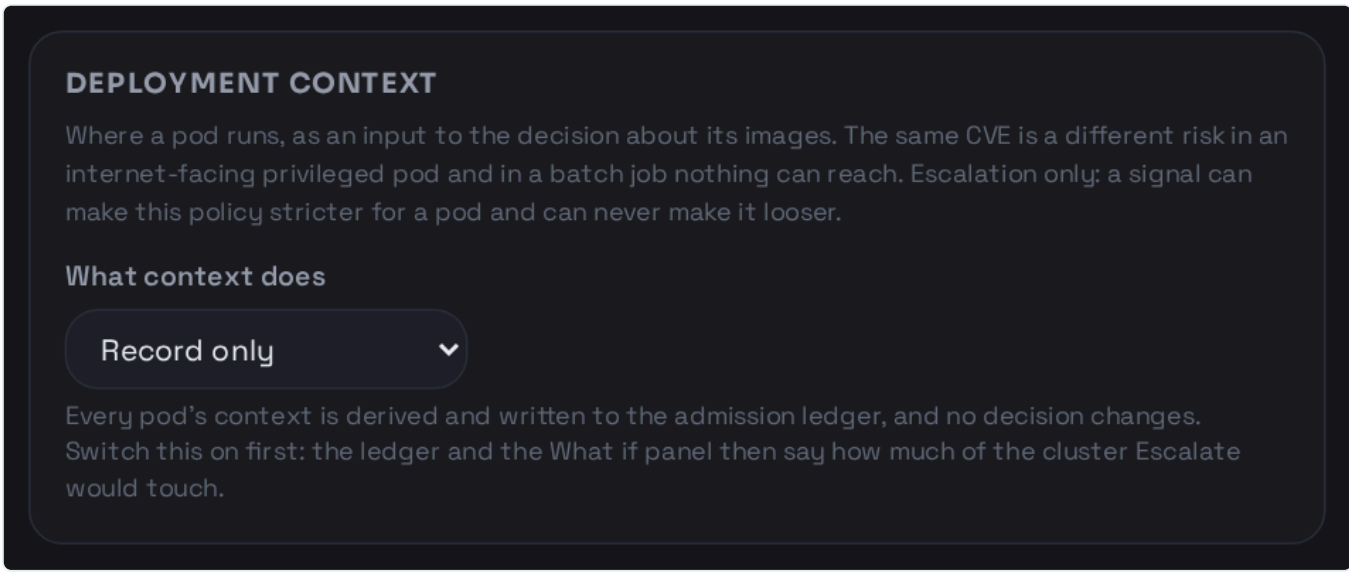
El mismo hallazgo HIGH pesa más en un pod privilegiado y expuesto a internet que en un batch al que nadie llega. `deploymentContext` permite a una policy tratar esos pods con más rigor. Solo endurece una decisión. Ninguna señal, ni combinación de señales, admite un pod que la policy rechazaría de otro modo.

```
spec:
  deploymentContext:
    mode: Escalate      # Ignore (por defecto) | Escalate
    signals:            # vacío significa las seis
      - exposed
      - privileged
```

Para cada pod se derivan seis señales, que quedan en cada fila del registro en los dos modos: `privileged`, `root`, `hostNamespace`, `hostFilesystem`, `capabilities` y `exposed`. Solo `exposed` mira fuera del pod: hostNetwork, un hostPort, un Service NodePort o LoadBalancer que selecciona el pod, o un Service ClusterIP detrás de un Ingress. Un Service ClusterIP sin Ingress no cuenta como exposición. Con `Escalate`, un pod con una señal seleccionada se juzga así:

La policy dice	Un pod con una señal seleccionada se juzga como
`thresholds.blockCritical: true`	bloquear también HIGH
`thresholds.blockHigh: true`	bloquear también MEDIUM
`knownExploited: warn`	`block`
`endOfLifeOS: warn`	`block`

`maxMedium` y `maxLow` no cambian, e `ignore` sigue siendo `ignore`. El rechazo nombra la señal y, si es exposición, el Service, el Ingress o el hostPort que la causó.



La sección *Deployment context* en *Record only*, es decir `Ignore`. El contexto de cada pod se escribe en el ledger de admission y ninguna decisión cambia. El ledger y el panel What if muestran después a cuántos pods afectaría `Escalate`.

Referencia de campos

Campo	Valores	Qué decide
enforcement	Enforce | Audit	Rechazar, o solo registrar lo que se habría rechazado.
admissionTiming	AllowAndScan | DenyUntilScanned	Qué ocurre con una image que aún no tiene escaneo.
blockLatestTag	bool	Rechazar el tag `:latest`.
digestEnforcement	AllowTags | PreferDigest | Required	Si las images deben fijarse por digest.
allowedRegistries	lista de hosts	Vacía, permite cualquier registry. Con valores, rechaza los demás.
denyList	enforce | audit | "off"	Qué hace una coincidencia de la deny list en los namespaces de esta policy. Por defecto `enforce`, que rechaza incluso con `enforcement: Audit` (capítulo 7).
targetNamespaces	lista	Vacía, es la policy global. Con valores, se limita a esos namespaces.
thresholds	blockCritical, blockHigh, maxMedium, maxLow	Rechazar ante cualquier hallazgo crítico o alto. Fijar topes para medium y low (0 significa sin límite).
slaWindows	duraciones	Cuánto puede seguir abierto un hallazgo de cada gravedad antes de que los informes lo den por vencido.
requireSignature / requireSBOM / requireVulnAttestation	bool	Rechazar images sin firma de cosign verificada, sin attestation de SBOM o sin attestation de escaneo de vulnerabilidades.
signatureKeys	lista de claves públicas PEM	Con `requireSignature`, solo se acepta una firma que verifique una de estas claves.
attestationMaxAge	duración	Una attestation más antigua deja de contar.
workloadHardening	mode, minSeverity, exempt	Si los containers privilegiados, los montajes hostPath y ajustes parecidos rechazan, avisan o pasan.
endOfLifeOS	ignore | warn | block	Images cuyo sistema base está fuera de soporte. Por defecto `warn`.
knownExploited	ignore | warn | block	Images con una CVE del catálogo KEV de CISA. Por defecto `warn`.
provenance	minLevel 0 a 3, trustedBuilders (id, level, signer, issuer)	Rechazar images sin procedencia verificada de un builder de confianza con el nivel exigido. Por defecto `minLevel: 0`, desactivado.
vex	mode ignore | apply, trustedIssuers	Si la declaración not_affected o fixed de un productor de confianza quita un hallazgo de los totales. Por defecto `ignore`.
deploymentContext	mode Ignore | Escalate, signals	Juzgar con más rigor los pods con una señal seleccionada. Por defecto `Ignore`.
runtimeReconciliation	audit | notify | enforce	Qué hace la conciliación periódica con una image en marcha sin admission detrás.

breakGlass	enabled, reason, approvedBy, auditRef, expiresAt, affectedImages	Una excepción con nombre y aprobada, que caduca. Cada uso queda registrado.
triageDefaults	maxDuration, requireJustification	Cuánto puede durar un riesgo aceptado y si necesita justificación.
storageUnreachablePolicy	Allow | Deny	La respuesta de admission mientras la base de datos no es alcanzable.

Lo que provoca un evento de Falco no forma parte de la policy de admission. Lo fijan las policies de runtime de la consola (capítulo 8).

Break-glass

Cuando producción está caída y la solución es una image que la policy rechaza, el break-glass deja entrar las images nombradas, registra cada uso y caduca solo:

```
breakGlass:
  enabled: true
  reason: "INC-2417: la image de vuelta atrás tiene un hallazgo high conocido"
  approvedBy: "guardia, responsable del incidente"
  auditRef: "INC-2417"
  expiresAt: "2026-09-01T06:00:00Z"
  affectedImages:
    - registry.example.com/payments/api:1.41.2
```

Cada admission que pasó solo gracias al break-glass queda marcada así en la pista de auditoría y en los informes de cumplimiento.

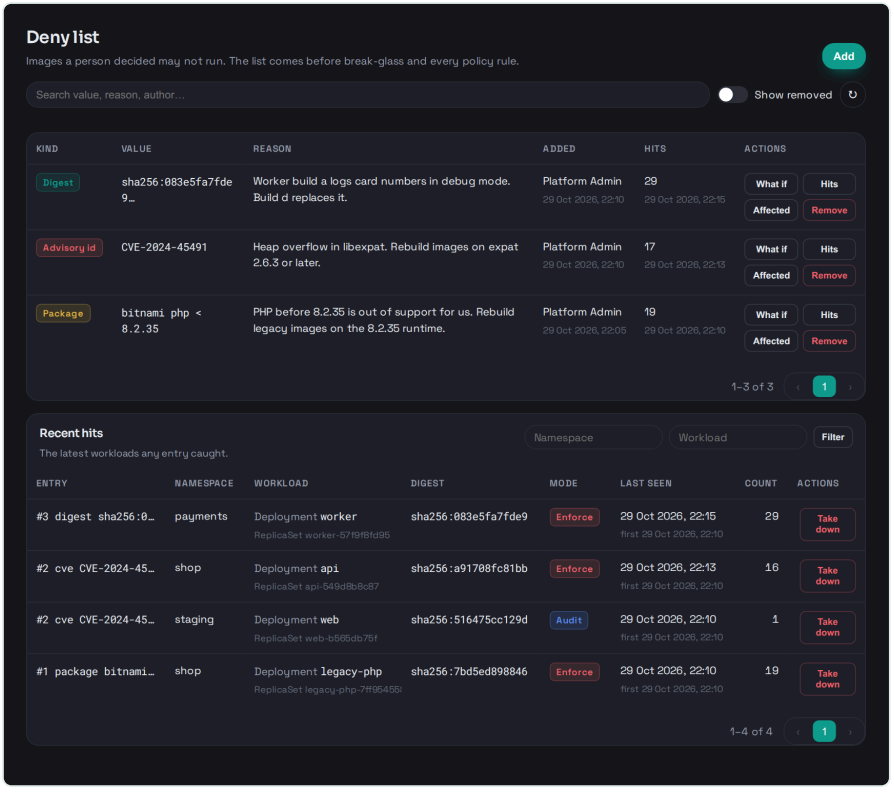
Las images que cubre el break-glass se admiten sin las reglas de image de la policy: la lista de registries permitidos, las reglas de `:latest` y de fijación por digest, el resultado del escaneo y sus reglas, las firmas, las attestations y la procedencia del build, la cuota de images y el tope de nodes. La deny list y el endurecimiento de workloads siguen aplicándose. `affectedImages` se compara por nombre de image, así que `nginx`, `docker.io/library/nginx` y `nginx:1.27` nombran una misma image, y un tag en la lista no la restringe. Una lista vacía cubre todas las images. Antes de la 1.4.0, la lista no se guardaba al abrir el break-glass, de modo que una ventana limitada a images concretas cubría todas las images de la policy.

7. La deny list

Una policy responde de forma automática a una pregunta: ¿cumple esta image las reglas? Una entrada de la deny list responde a otra: ¿ha decidido alguien que esta image no debe ejecutarse? La deny list es donde se deja escrita esa decisión, y la admission la consulta antes que nada.

Hay una lista por cluster, en Security > Deny list. El campo `denyList` de cada policy decide qué hacen sus namespaces ante una coincidencia. Una entrada nunca se borra. Retirarla la levanta, y la fila queda con quién la retiró, cuándo y por qué. Añadir y retirar son acciones de administrador, y ambas exigen un motivo.

> **Una lista vacía no cambia nada.** Sin entradas activas, la admission no resuelve ningún digest de más y no rechaza nada. Actualizar a la 1.4.0 no cambia ninguna decisión hasta que alguien añade una entrada.



Security > Deny list con tres entradas: un digest, el aviso CVE-2024-45491 y el paquete `bitnami php < 8.2.35`. Cada fila da el motivo, quién la añadió y cuándo, y cuántas coincidencias ha tenido. *Recent hits* nombra después los workloads atrapados, por namespace y digest, y si la policy del namespace aplicaba la lista o solo la auditaba, como en `staging`.

Tipos de entrada

Tipo	Coincide con	Ejemplo
Digest	Un build exacto de una image, comparado con el digest que la admission resolvió para el pod y con el índice de image o los manifiestos de plataforma a los que se sabe que pertenece.	`sha256:9f2a…d4e5`
Identificador de aviso	Toda image cuyo último escaneo informa del aviso, haya corrección o no.	`CVE-2021-44228`, `GHSA-jfh8-c2jp-5v3q`
Paquete	Toda image cuyo último escaneo lista el paquete en el rango de versiones.	`maven org.apache.logging.log4j:log4j-core < 2.17.1`
Hash de ejecutable	Toda image que lleva un archivo, por el sha256 de sus bytes, sea cual sea su nombre o ruta.	un sha256
Ruta de ejecutable	Toda image con un ejecutable en una ruta, o con un nombre de archivo en cualquier lugar. Una regla de ruta también alcanza a las images base.	`/bin/sh`, `/usr/local/bin/*`, `nc`
Paquete, ecosistema `bitnami`	Toda image con un runtime cuya versión, leída del propio ejecutable, está en el rango. Nunca coincide con un paquete de la distribución.	`bitnami php < 8.2.35`

Una entrada de aviso acepta los esquemas con los que los escáneres informan de sus hallazgos, porque para algunos ese identificador es su único nombre: CVE, GHSA, GO, PYSEC, RUSTSEC, OSV, MAL, GSD, DLA, DSA, ELA, USN, RHSA, RHBA, ALSA, ALBA, ELSA, CGA, BIT, TEMP, SUSE-SU y openSUSE-SU. Cualquier otra cosa se rechaza al añadirla, ya que un identificador que ningún escáner informa no coincidiría con nada.

Una entrada de paquete nombra un ecosistema, un paquete y un operador: por debajo, hasta, exactamente, desde, por encima, entre (límite inferior incluido, superior excluido) o cualquier versión. Dos rangos son dos entradas. Las versiones se comparan con el comparador que usa el propio escáner para el ecosistema: npm, Go, Cargo, Composer, NuGet, Maven, PyPI, RubyGems, deb, apk y rpm. Una versión que el comparador no sabe leer se rechaza al añadirla, en lugar de compararse como texto, donde 10.0 quedaría por debajo de 9.0. Una entrada con *por encima* o *desde* cubre también cada versión publicada después de añadirla, y What if lo dice.

Su lugar en la decisión

El orden es la deny list, después el break-glass y después las reglas de la policy. Una image en la lista se rechaza aunque su escaneo esté limpio, e incluso con un break-glass abierto: la entrada es una decisión deliberada y está por encima de la puerta de emergencia. El rechazo dice:

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

Las reglas que no necesitan llamar al registry, la lista de registries permitidos, `blockLatestTag` y la exigencia de digest, se comprueban primero. Con una policy en Enforce, su rechazo basta. Con una policy en Audit, la revisión sigue hasta la deny list, de modo que allí también se rechaza una image en la lista.

`denyList` en la policy	Qué hace una coincidencia en sus namespaces
`enforce` (por defecto)	Rechaza el pod, aunque la policy esté en modo Audit. Un campo vacío o ausente equivale a `enforce`.
`audit`	Admite el pod con el aviso *would be denied by deny list entry …* y registra la coincidencia.
`"off"`	No se consulta la lista.

Escriba `off` entre comillas en YAML: un `off` sin comillas es un booleano para un lector de YAML 1.1. Tras la actualización, cada policy registra una revisión nueva, porque el campo forma parte del contenido que hashea el registro.

Qué peticiones revisa. Cada container, init container y ephemeral container de un pod nuevo. En `kubectl debug`, solo el container que añade la petición. Otras actualizaciones de un pod existente, una etiqueta, una anotación o un finalizer, ni se rechazan ni se cuentan. Un dry run se responde como una petición real y no registra coincidencias.

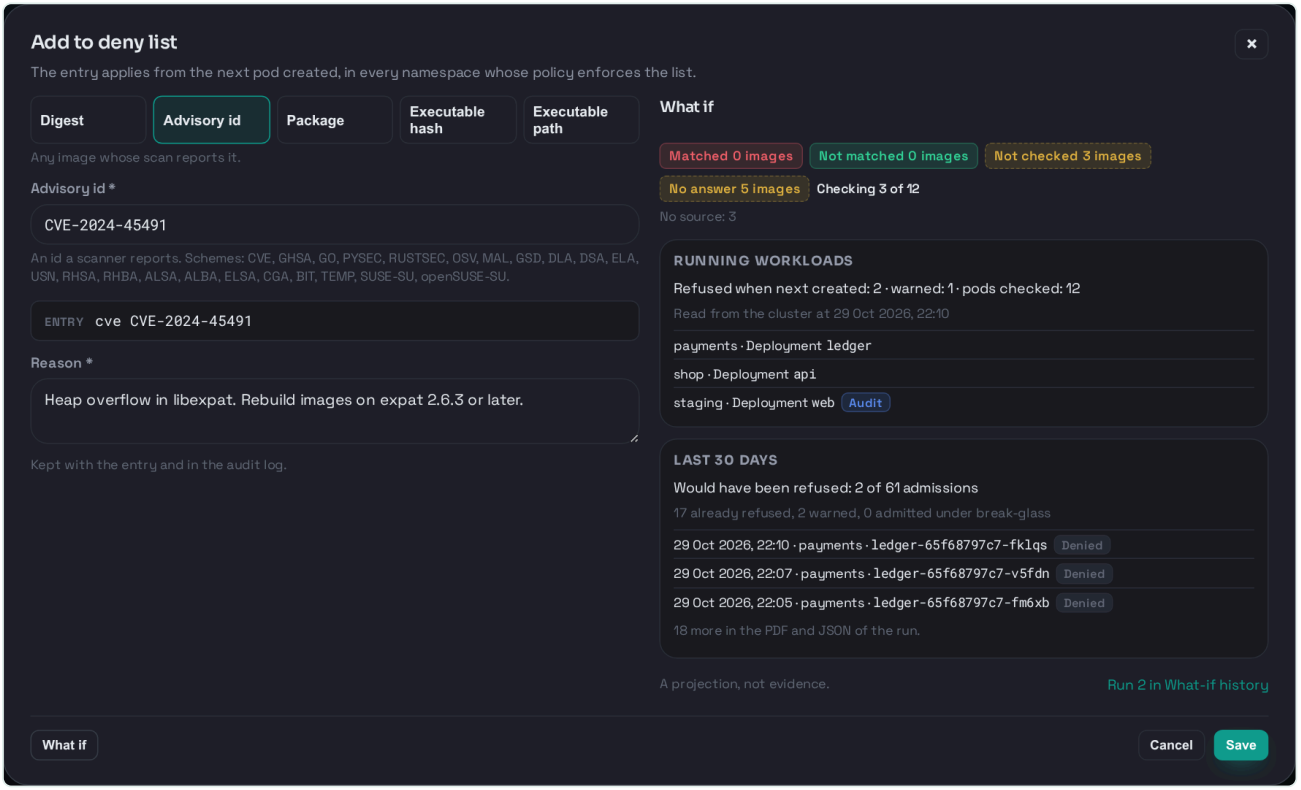
No se detiene nada de lo que se ejecuta. Una entrada rechaza un workload la próxima vez que se crea uno de sus pods. El controlador del workload sigue intentándolo, a su propio ritmo, hasta que cambie su image.

Cuando no se puede resolver un digest. Mientras la lista tiene una entrada, la admission resuelve el digest del pod antes del break-glass. Si el registry no responde, un pod con una policy en Enforce y `denyList: enforce` se rechaza con *deny list could not be evaluated: digest unresolved*. Con una policy en Audit, o con `denyList: audit`, se admite con ese aviso. Una image que solo pueden descargar los nodes, con credenciales del node y no con un pull secret que referencie el pod o su service account, no la puede resolver el operator, y se rechaza en los namespaces en Enforce mientras la lista no esté vacía. Fíjela por digest o referencie un pull secret desde el pod. Si no se puede leer la base de datos, la lista no se aplica y el pod lleva el aviso *deny list unavailable*.

What if, antes de añadir la entrada

What if prueba una entrada antes de que nadie se entere en la admission, en el diálogo Add antes de guardar o desde la fila de una entrada. Responde dos cosas, una junto a otra.

- **Los workloads en marcha que cubre**, leídos del cluster en ese momento y agrupados por el workload al que pertenece cada pod: el Deployment sobre su ReplicaSet, el CronJob sobre su Job, un StatefulSet, un DaemonSet o el propio pod. En un cluster grande, la lectura se detiene tras 20 segundos, 20.000 pods o 2.000 images distintas, y entonces indica que es parcial y cuántos pods revisó.
- **Las admissions pasadas que habría rechazado**, de los últimos 30 días por defecto y hasta 90: las filas del registro decididas de nuevo con la policy actual de cada namespace, con esta entrada como lista completa. Las admissions que entraron con break-glass también se deciden, porque el break-glass nunca salta la deny list.



What if en el diálogo de alta, antes de guardar la entrada. De los workloads en marcha, 2 se rechazarían la próxima vez que se creen y 1 recibiría un aviso, porque `staging` audita la lista. En los últimos 30 días se habrían rechazado 2 de 61 admissions, además de 17 ya rechazadas y 2 avisadas. La comprobación de ejecutables seguía en curso, 3 de 12 images, y las etiquetas indican cuántas images no tenían respuesta todavía. La ejecución se guarda como ejecución 2 en What-if history.

No se detiene ni se guarda nada. Una ejecución de un deployer o de un administrador se guarda en Security > What-if history con el tipo *Deny list*, noventa días como cualquier ejecución y como mucho las últimas quinientas ejecuciones de la deny list, contadas aparte de las de policy. Se descarga en PDF o JSON como proyección, no como evidencia, y nunca entra en un paquete de evidencia.

Coincidencias

Cada rechazo y cada aviso de auditoría que provoca una entrada se cuenta contra el workload: una fila por entrada, namespace, workload y digest, con la primera y la última vez que se vio y cuántas veces. Los pods de un Deployment cuentan para el Deployment y los de un CronJob para el CronJob, así que un rollout no abre una fila nueva. La página de la deny list muestra bajo la lista las últimas coincidencias de todas las entradas, y *Hits* en una fila muestra las de esa entrada, filtrables por namespace o workload. La fila del registro de cada rechazo nombra la entrada que lo decidió.

Ejecutables: lo que un escáner no lista

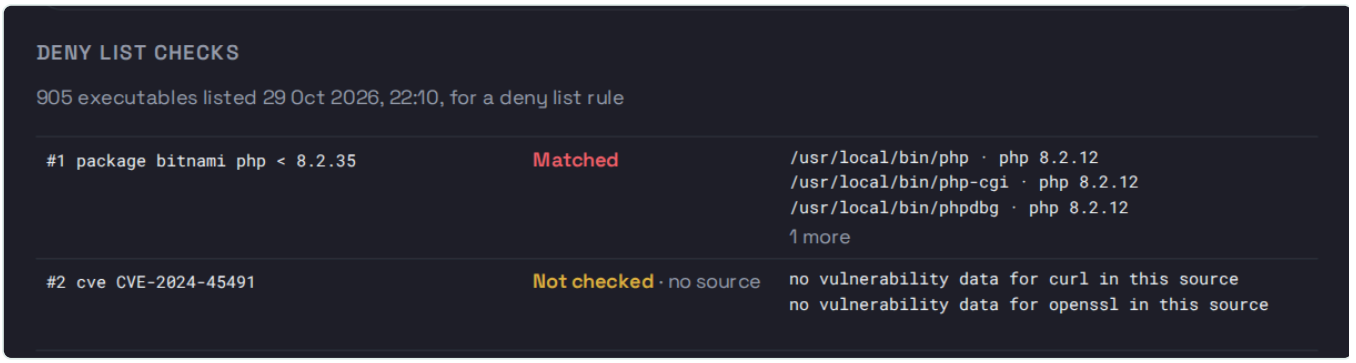
Trivy lee lo que registraron los gestores de paquetes. Un runtime compilado desde el código fuente y copiado en una image no tiene ese registro, así que Trivy no lista ningún paquete para él. Un PHP 8.2.12 compilado desde el archivo de php.net, en una image sin etiquetas, muestra 91 paquetes de Debian y ningún php. Las entradas de ejecutables y el ecosistema `bitnami` los responde una segunda herramienta de la image del operator, `attestkeep-binscan`, que lee los ejecutables de cada image. El escáner la ejecuta igual que ejecuta Trivy.

- **Solo cuando hace falta.** Solo se ejecuta mientras la lista tiene una entrada que responde: un hash o una ruta de ejecutable, un paquete `bitnami` o un identificador de aviso. Sin ellas nunca se ejecuta y ninguna image se descarga dos veces.
- **Solo bytes.** La image se descarga por digest con los pull secrets del propio workload, sus capas se leen de arriba abajo quitando los archivos borrados y cada archivo ELF se hashea. El nombre de la image, sus etiquetas, su entorno y su image base nunca se leen.
- **La versión sale del propio archivo.** La versión de un runtime se lee de los datos de solo lectura del ejecutable y se confirma con una segunda cadena del mismo archivo. Los productos son php, node, python, ruby, nginx, apache, postgresql, curl y openssl. Un build pasado por `strip` se lee igualmente: `strip` quita los símbolos, no los datos de solo lectura.
- **Tres respuestas.** Coincide, con el archivo, su sha256, la versión y la firma que lo encontró. No coincide. O no comprobado, con el motivo: credenciales, registry, image desaparecida, límite, error de la herramienta o sin fuente. No comprobado nunca cuenta como limpio.

Límites del recorrido. Un archivo ELF de más de 256 MiB se hashea pero no se lee para buscar versiones, se leen como mucho 4.000 archivos ELF por image y un recorrido se detiene a los 60 segundos. Estos topes son el valor del chart `binscanLimits`. Una comprobación completa, descarga, recorrido y comparación de versiones, tiene su propio plazo, `scan.binscanTimeout` (10 minutos). Antes de descargar, la comprobación suma el tamaño de las capas de la image, y una image mayor que su presupuesto de disco no se descarga. El presupuesto es el límite de `/tmp` del escáner, `scan.tmpSizeLimit` (4Gi), repartido entre sus workers, o `scan.binscanDiskBudget` si se fija. Una respuesta cortada por un tope o por el plazo dice *not checked (limit)* y lo nombra. Un administrador puede usar *Check again* con una image que no se pudo comprobar, con los topes vigentes.

Cuándo se ejecuta. Una entrada nueva se comprueba primero contra las images de los pods en marcha y después contra los digests escaneados en los últimos 90 días. Una image nueva se comprueba en su escaneo, y las images en marcha a las que una entrada aún no ha llegado se comprueban cada hora. Un pod cuya image aún no tiene respuesta para una entrada de ejecutable o `bitnami` se decide como una image sin escanear: `DenyUntilScanned` rechaza y `AllowAndScan` admite con aviso. Una entrada de aviso no espera: los hallazgos de Trivy deciden al momento, y una coincidencia que la comprobación encuentre después solo se suma a ellos.

Los identificadores de aviso también llegan a estos runtimes. Con una entrada de aviso en la lista, los runtimes encontrados se comparan con los rangos de vulnerabilidades de Bitnami de la base de Trivy que el operator ya mantiene. Un runtime que esos rangos no cubren, como curl u openssl, dice "no vulnerability data in this source", nunca "not matched".



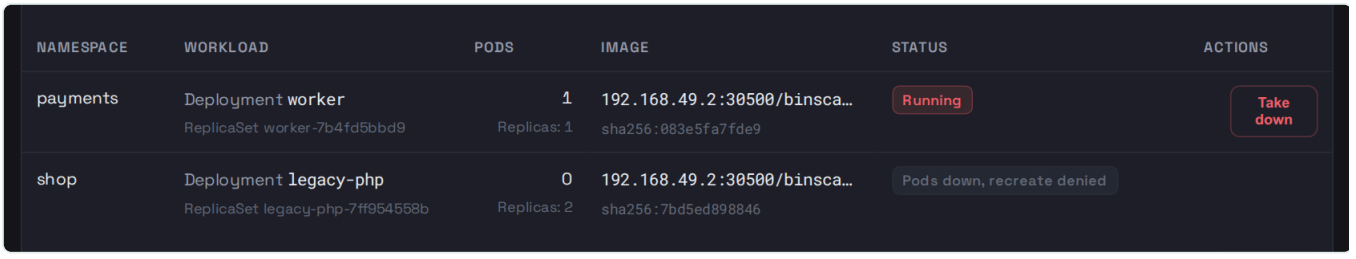
Deny list checks en el detalle de `php:8.2.12-cli`, a partir de una lista de 905 ejecutables. La entrada 1, `bitnami php < 8.2.35`, coincide con `/usr/local/bin/php` y otros tres archivos de la versión 8.2.12. La entrada 2, un id de aviso, indica *Not checked*: la fuente no tiene datos de vulnerabilidades para el curl y el openssl encontrados.

Lo que no ve: un ejecutable cuya cadena de versión se quitó o se modificó, uno sin cabeceras de sección, un ejecutable enlazado estáticamente y pasado por strip que solo se reconocería por un nombre de símbolo, los ejecutables empaquetados, los runtimes que se descargan al arrancar el container, las versiones escritas fuera de los ejecutables, el código de aplicación interpretado y las images que no se pueden descargar. La falta de cabeceras de sección, un ejecutable empaquetado y una image que no se puede descargar aparecen como no comprobados, con la causa.

Workloads afectados y take down

Una entrada rechaza pods nuevos. Nunca detiene lo que ya se ejecuta, y ningún modo de policy lo hace por su cuenta. *Affected* en la fila de una entrada lista, en directo desde el cluster, los workloads que atrapa ahora y los que rechazó, cada uno con un estado:

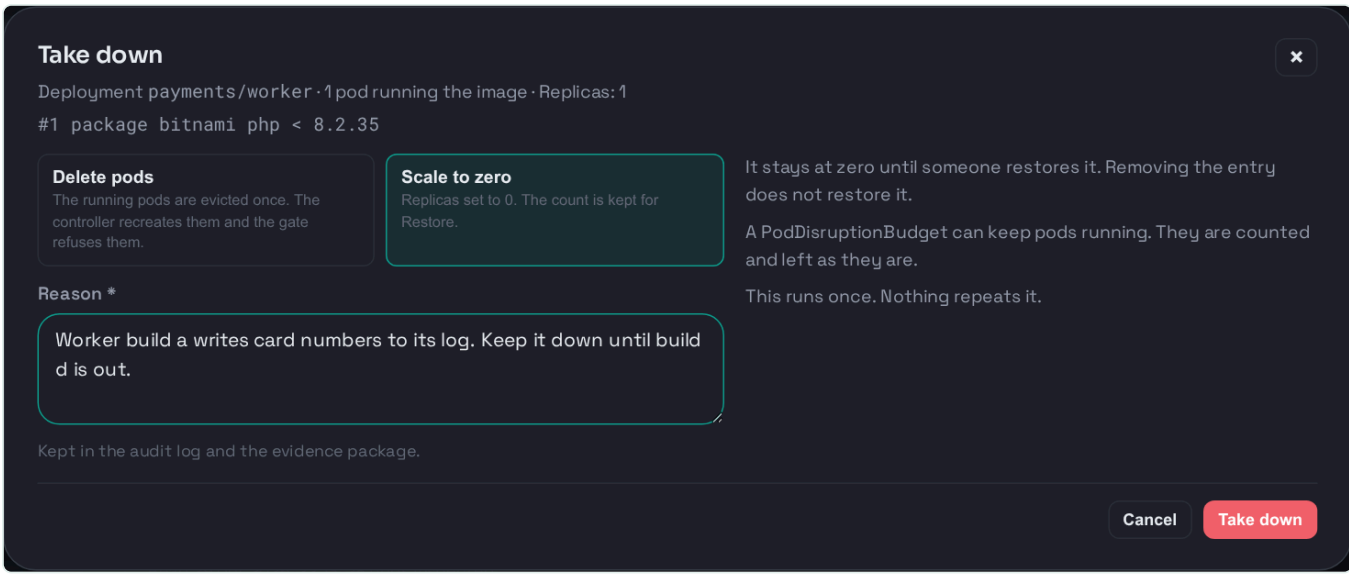
- **Running**: hay pods en marcha con una image de la lista.
- **Pods down, recreate denied**: no hay ninguno en marcha y los pods nuevos del controlador se rechazan.
- **Scaled to zero**: un take down lo llevó a cero réplicas y sigue existiendo.
- **Restored externally**: aquí se llevó a cero y vuelve a pedir réplicas. Alguien, o una herramienta GitOps, las repuso. Se muestra y se deja como está.



Affected, leído en directo del cluster. El Deployment `worker` de `payments` todavía ejecuta una image de la lista, así que se ofrece *Take down*. `legacy-php` en `shop` no tiene ningún pod en marcha y sus pods nuevos se rechazan.

Con `rbac.denyList.takedown: true` en el chart, un administrador puede hacer *Take down* de un workload en marcha, con un motivo. Actúa una vez y nada lo repite. El take down se registra antes de tocar el cluster, y su resultado después: hecho, bloqueado por un presupuesto o fallido con el error.

- **Delete pods** (por defecto) desaloja una vez los pods con una image de la lista. El estado deseado no se toca. El controlador crea sustitutos, la admission los rechaza mientras siga la entrada, y el workload aparece degradado hasta que cambie su image o se retire la entrada. Entonces se recupera solo.
- **Scale to zero**, para un Deployment, un StatefulSet o un ReplicaSet, pone las réplicas a 0 mediante el subrecurso scale y guarda el número anterior. Se niega si el número ya no es el que mostraba el diálogo. **Restore** devuelve el número. Retirar la entrada no lo hace: nada cambia la escala de un workload sin un clic. A un DaemonSet, un Job, un CronJob o un pod suelto se le hace take down borrando sus pods.

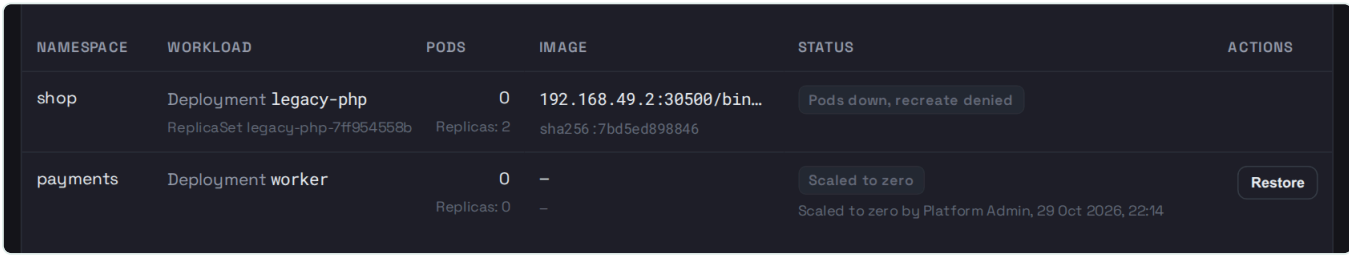


Take down del Deployment `payments/worker`, con un pod en marcha. Está elegido *Scale to zero*, así que el número de réplicas se guarda para Restore. Las notas dicen que quitar la entrada no restaura nada, que un PodDisruptionBudget puede mantener pods en marcha y que la acción se ejecuta una vez y nada la repite. El motivo queda en el registro de auditoría y en el paquete de evidencia.

Se respetan los PodDisruptionBudgets. Los pods se desalojan con la API de desalojo, y un pod que protege un presupuesto sigue en marcha. La fila dice entonces "Blocked by PodDisruptionBudget" con el número de pods que conservó, y nada fuerza ni reintenta.

GitOps. Un workload que siguen Argo CD o Flux aparece marcado en el panel, y el diálogo avisa de que la herramienta repondrá un scale to zero. Cuando lo hace, la fila dice *Restored externally* y Attestkeep no lo vuelve a reducir. El arreglo definitivo está en el origen: cambiar la image o bajar las réplicas.

Sin `rbac.denyList.takedown: true`, el botón aparece desactivado y la API responde `409`. Un namespace que la admission no revisa, el del propio operator o uno que el webhook omite, nunca recibe un take down: sus pods recreados no se comprobarían, y un take down solo reiniciaría la misma image.



Tras el take down la fila muestra *Scaled to zero*, con quién lo hizo y cuándo, y ofrece *Restore*. Restore devuelve el número de réplicas guardado. Nada más vuelve a escalar el workload.

Notificaciones, eventos y evidencia

Un rechazo deja un evento Warning de Kubernetes, con motivo `DenyListRefused`, que nombra la entrada, sobre el workload, de modo que `kubectl describe` dice por qué no arrancan sus pods. Para un pod sin propietario va al namespace. Hay como mucho uno por workload y entrada cada cinco minutos. El ImageSecurityReport de una image rechazada lleva `status.denyList` con la entrada y el rechazo. Los canales suscritos a *Deny list hit* reciben el aviso como describe el capítulo 5.

El paquete de evidencia tiene una sección *Deny list*: las entradas vigentes al final del periodo y las añadidas o retiradas en él, con quién, cuándo y por qué; las admissions que las entradas rechazaron y dejaron pasar; las excepciones, es decir, coincidencias admitidas porque un namespace audita la lista; cada take down y cada restauración con su motivo; y, para una entrada de ejecutable, el archivo, su hash y la versión. El hash de la policy no cubre la lista. La entrada que decidió una admission se nombra en el registro de esa admission.

Quién puede hacer qué

Leer la lista, sus coincidencias y sus workloads afectados, y ejecutar What if, requieren la misma autoridad que leer una policy. Añadir y retirar una entrada, hacer take down de un workload y restaurarlo son acciones de administrador, y todas salvo la restauración exigen un motivo. Check again también es una acción de administrador. El registro de auditoría las anota como `deny_list_add`, `deny_list_remove`, `deny_list_takedown`, `deny_list_restore` y `deny_list_check_again`. La API está en `/api/v1/deny-list`. Un campo que el servidor rechaza vuelve como `400` con el nombre del campo, y una segunda entrada activa con el mismo valor da `409`.

8. Eventos de runtime de Falco

Falco observa lo que hacen los containers después de arrancar. Attestkeep puede recibir los eventos de Falco, guardar cada uno junto a la image y el pod que lo produjeron y actuar sobre él mediante policies de runtime que usted escribe en la consola.

Attestkeep no distribuye, instala, configura ni escribe en Falco. La detección la hacen su Falco y sus reglas. Un evento de runtime nunca cambia una decisión de admission. Una policy de runtime puede notificar, abrir una incidencia y detener el pod del que vino el evento.

Cómo funciona

1. Una regla de Falco salta en un node, por ejemplo cuando alguien abre una shell interactiva en un container.
2. El evento se envía en JSON al puerto de integraciones del operator por TLS mutuo. Falco lo envía desde su propio `http_output`, o Falcosidekick lo reenvía desde su salida de webhook. En ambos casos, el emisor presenta un certificado de cliente que emitió el operator.
3. El operator guarda el evento y lo asocia a la image y al digest que ejecuta el container.
4. Las policies de runtime se comprueban en orden. La primera policy activa que coincide decide. Si ninguna coincide, el evento se guarda como registro y no se envía nada.
5. El evento y la decisión aparecen en Falco > Events, junto a la image en la página Images y en el paquete de evidencia.

Requisitos

- Falco instalado con el chart de Helm original `falcosecurity/falco`. Los valores documentados son para el chart 9.2.0 (Falco 0.45.0, Falcosidekick 2.32.0). No se han comprobado versiones anteriores.
- `falco.json_output: true`.
- Acceso de red desde los pods de Falco o Falcosidekick al Service del operator en el puerto de integraciones, 8444 por defecto.
- Una cuenta de admin. Solo los admins ven la página de integración, descargan certificados y editan policies.
- Para una policy que detiene pods: `rbac.allowEnforce: true` en el chart de Attestkeep. `rbac.denyList.takedown` no cuenta para esto.
- Para mensajes e incidencias: un canal de notificación, o un canal de GitHub, GitLab o Jira, suscrito a Runtime alert.

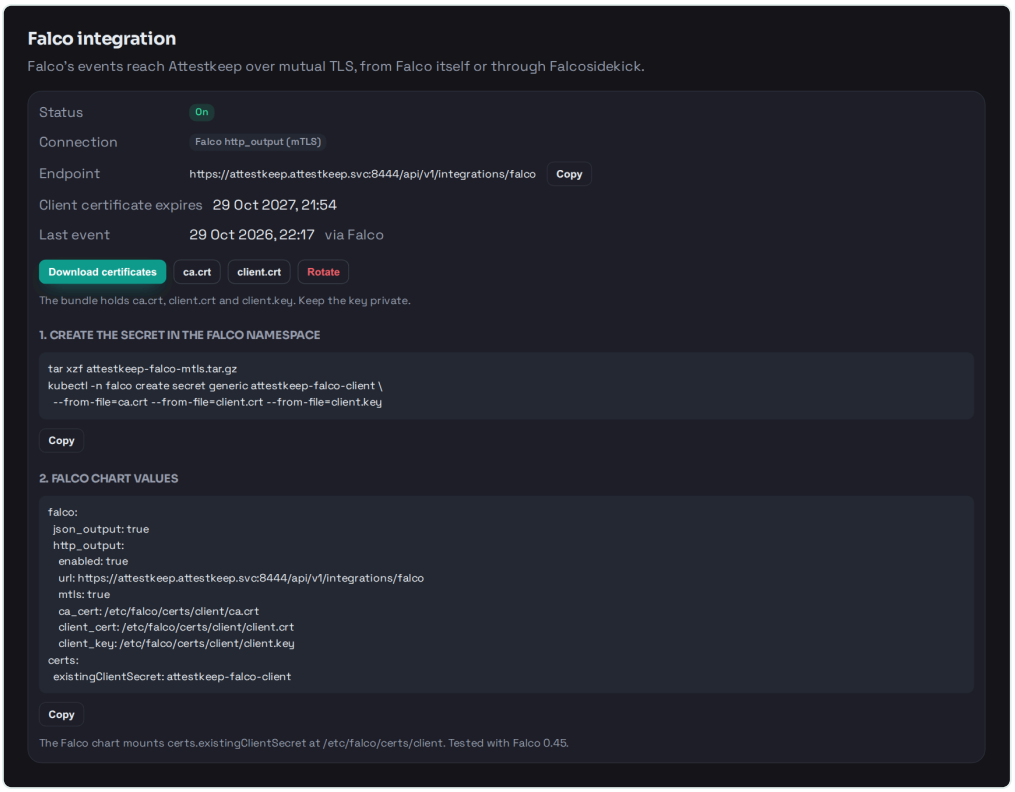
Conectar Falco

Active la integración en el chart de Attestkeep y elija cómo se conecta Falco:

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

Con `mtls`, el valor por defecto, Falco envía cada evento desde su propio `http_output` directamente al operator. Use `sidekick` si su Falco ya envía a Falcosidekick. La salida de webhook de Sidekick reenvía entonces al operator con el mismo certificado de cliente. Cualquier otro valor hace fallar el chart.

Con la integración activada, el operator crea su propia CA pequeña y un certificado de cliente (`CN=falco`, válido un año) y los guarda en el Secret `<release>-falco-mtls`. El puerto de integraciones sirve una sola ruta, `POST /api/v1/integrations/falco`, y solo acepta un certificado de cliente firmado por esa CA. No hay token.

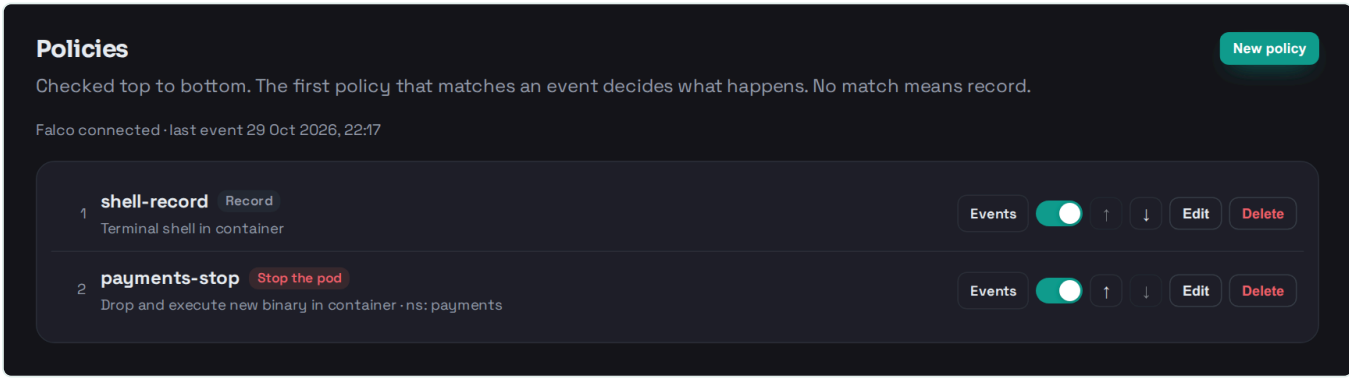


Falco > Falco integration en la instalación de este manual. El estado es On, la conexión es el propio `http_output` de Falco con mTLS, y el último evento llegó vía Falco. *Download certificates* guarda un paquete con `ca.crt`, `client.crt` y `client.key`. Debajo están el comando que crea el Secret en el namespace de Falco y los valores del chart de Falco para la conexión elegida.

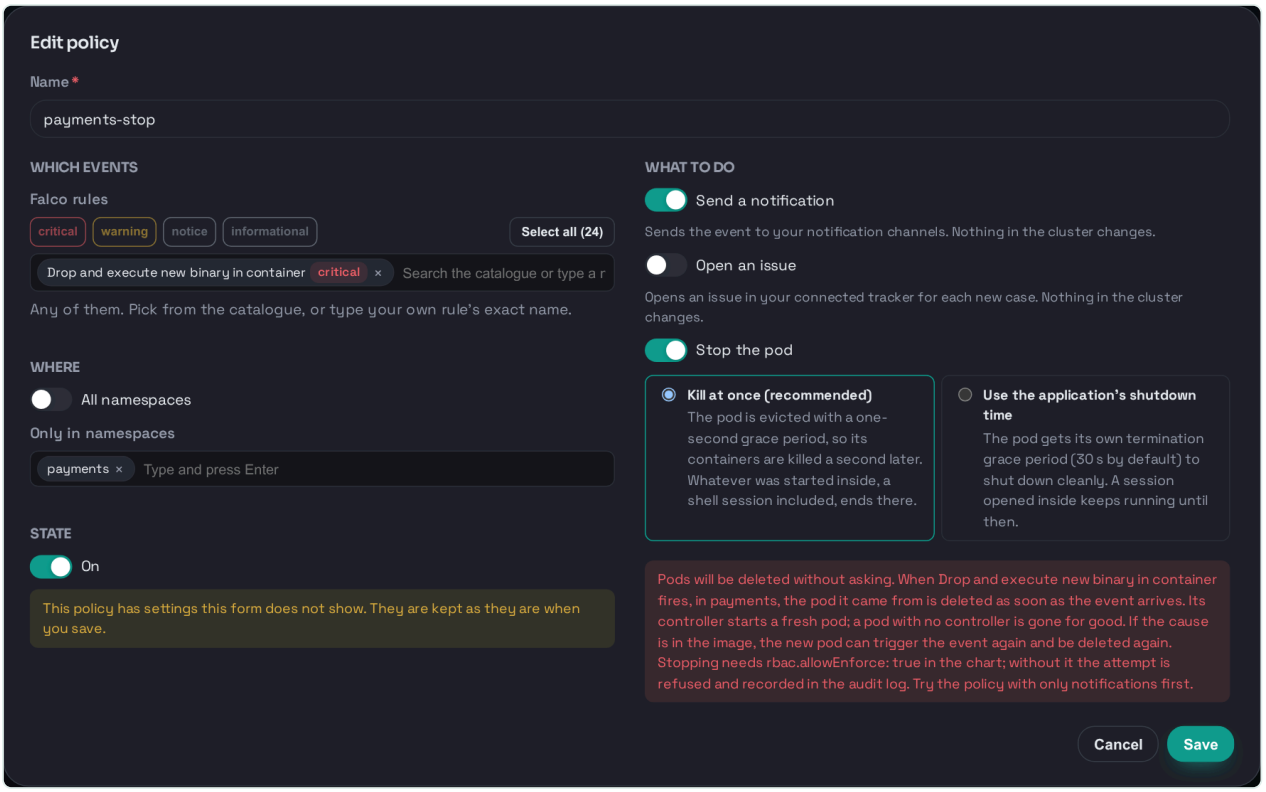
Cada descarga del paquete de certificados se escribe en el registro de auditoría. `client.key` debe mantenerse en privado, porque quien lo tenga puede enviar eventos a los que reaccionarán sus policies. *Rotate* emite un certificado de cliente nuevo desde la misma CA. El anterior sigue funcionando hasta que caduca, así que los eventos siguen llegando mientras sustituye el Secret. Cuando se renueva el certificado de servidor del propio operator, la página muestra "Certificates changed on <fecha>" hasta que vuelva a descargar el paquete.

Policies de runtime

Falco > Policies lista sus policies de runtime en orden. Cada evento guardado se compara con ellas desde arriba, y decide la primera policy activa que coincide. Las flechas cambian el orden. Un cambio llega al momento a la réplica que lo guardó y en menos de 15 segundos a las demás.



Dos policies de runtime, revisadas de arriba abajo. `shell-record` registra *Terminal shell in container*. `payments-stop` detiene el pod ante *Drop and execute new binary in container* en `payments`. Cada policy lleva la etiqueta de su efecto aquí, en Events y en el paquete de evidencia.



El formulario de `payments-stop`: una regla de Falco, un namespace, *Send a notification* y *Stop the pod* activados, *Kill at once (recommended)* elegido. El recuadro rojo avisa de que los pods que coincidan se borrarán sin preguntar y, desde 1.4.0, de que detenerlos requiere `rbac.allowEnforce: true` en el chart.

Campo	Qué hace
Name	Obligatorio, 128 caracteres como máximo, único.
Falco rules	La policy coincide con eventos de cualquiera de estas reglas, por nombre exacto. Elija de la lista Rules o escriba el nombre de su propia regla. Hace falta al menos una.
All namespaces / Only in namespaces	Limita la policy a los eventos de pods de los namespaces listados.
Send a notification	Envía el evento a cada canal activo suscrito a Runtime alert.
Open an issue	Abre una incidencia a través de los canales de incidencias suscritos a Runtime alert.
Stop the pod	Detiene el pod del que vino el evento, con *Kill at once* o *Use the application's shutdown time*.
State	On u Off. Una policy nueva empieza en Off. Una policy en Off se conserva y no hace nada.

Si alguna de las tres acciones está activada, la policy se guarda con la acción `notify`. Si las tres están desactivadas, se guarda como `record`: los eventos que coinciden se registran contra la policy y no ocurre nada más. La API (`/api/v1/runtime/rules`) acepta más condiciones de las que muestra el formulario: tags, prioridad, fuente, campos de salida, el último veredicto y la firma de la image, y una ventana de `count` eventos en `duration`. También acepta la acción `allow`, que guarda el evento y no hace nada más. Si una policy tiene ajustes que el formulario no muestra, el formulario lo dice y los conserva al guardar.

Detener el pod

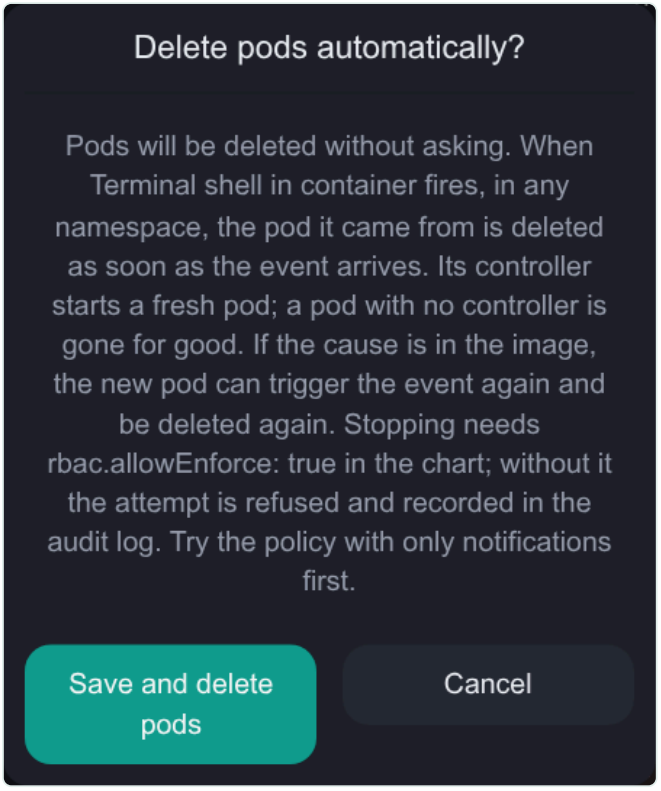
Una policy con *Stop the pod* detiene el pod nombrado en el evento en cuanto llega el evento, o en cuanto se alcanza la ventana si la policy tiene una. El operator pide al API server que desaloje el pod, así que se aplican los PodDisruptionBudgets. Si un presupuesto no permite el desalojo, el pod sigue en marcha y el registro de auditoría anota el motivo.

- *Kill at once* (`stop_mode: immediate` , por defecto): el desalojo usa un periodo de gracia de un segundo. Todo lo que se haya iniciado dentro de los containers, una shell abierta incluida, termina ahí.
- *Use the application's shutdown time* (`stop_mode: graceful`): el pod recibe su propio periodo de terminación, 30 segundos salvo que el pod fije otro valor. Una shell abierta dentro sigue hasta entonces.

El controlador del pod arranca un pod nuevo. Un pod sin controlador no vuelve. Si la causa está en la image, el pod nuevo puede provocar el mismo evento y detenerse otra vez.

- Detener necesita `rbac.allowEnforce: true` , conceda lo que conceda el ClusterRole. Desde la 1.4.0 es así incluso cuando `rbac.denyList.takedown: true` ha dado al operator el desalojo de pods para el take down. Sin él, la detención se rechaza antes de preguntar al API server, el registro de auditoría anota "not stopped: the chart has not granted pod eviction (set rbac.allowEnforce)" y la alerta dice que el pod sigue en marcha. La conciliación con `runtimeReconciliation: enforce` sigue la misma regla.
- Los pods del namespace del propio operator nunca se detienen. Ningún otro namespace queda excluido, así que use *Only in namespaces* para mantener una policy de detención lejos de los namespaces del sistema.
- El evento debe nombrar el pod (`k8s.ns.name` y `k8s.pod.name`). Con los valores por defecto del chart de Falco, lo hace.
- Un pod que ya se está terminando no se toca. Cada réplica del operator pide detener un mismo pod como mucho una vez cada 10 minutos.

Guardar una policy con *Stop the pod* activado pide confirmación antes. El diálogo dice que los pods se borrarán sin preguntar, qué le pasa a un pod con y sin controlador, y que detener necesita `rbac.allowEnforce: true` en el chart.



La pregunta que aparece al guardar una policy en la que *Stop the pod* se acaba de activar. Nombra la regla y los namespaces, explica qué pasa con un pod con y sin controlador, y recuerda que detener pods requiere `rbac.allowEnforce: true` . Sin ese ajuste, el intento se rechaza y queda en el registro de auditoría.

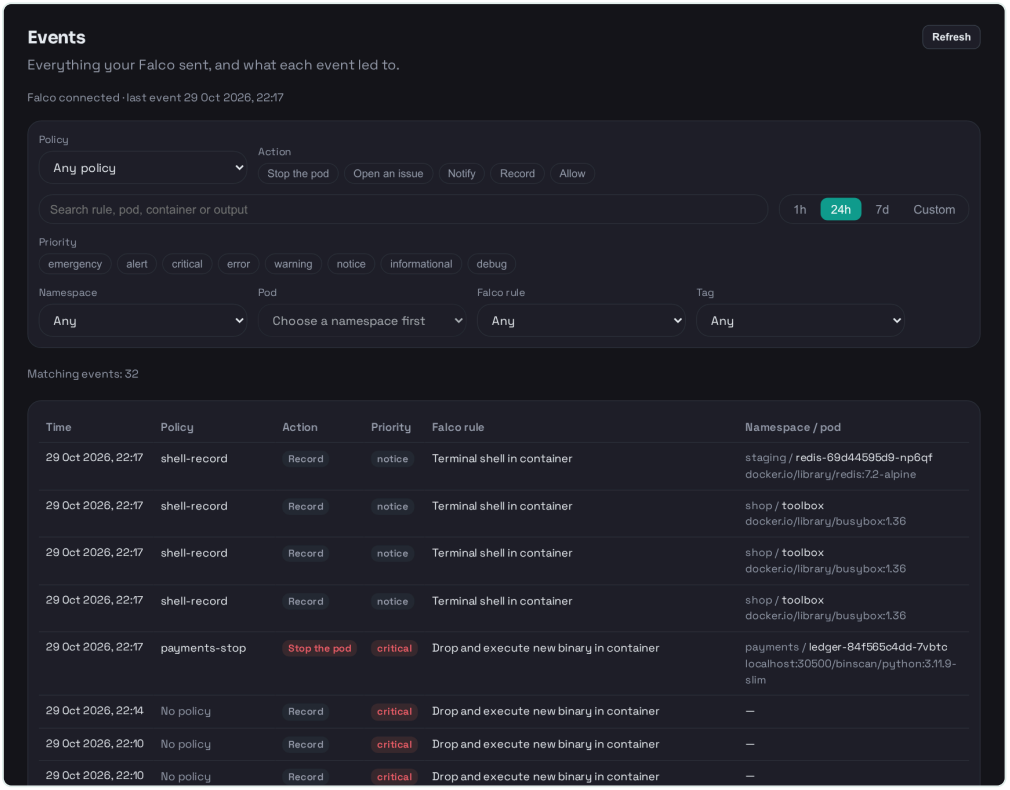
Cada intento se escribe en el registro de auditoría como `runtime_pod_stop` , con `policy:<nombre>` como actor, el pod como objetivo y, en el detalle, la regla de Falco, el UID del pod, el container, la image, el digest, el node, el identificador del evento y el resultado. Detener un pod no cambia la admission. Los pods nuevos de la misma image se admiten como antes.

Notificaciones e incidencias

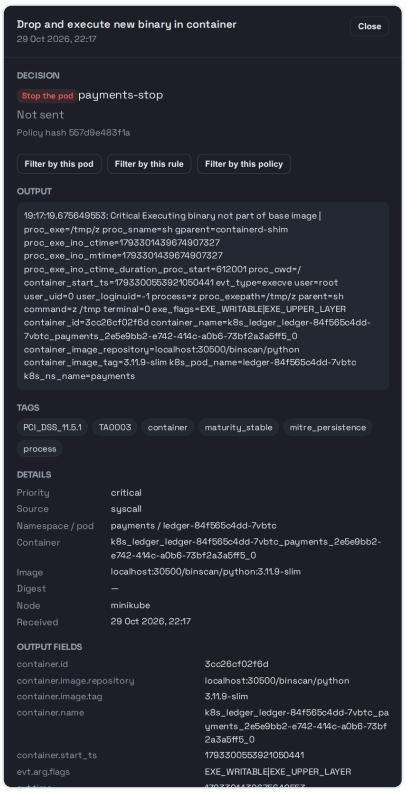
Un mensaje se titula `Runtime alert: <regla de Falco>` y lleva la línea de salida de Falco, la regla, la prioridad, la fuente, el namespace, el pod, el container, la image, el digest, el node, la hora, la policy y su acción. Si la policy detuvo el pod, el mensaje lo dice. Una policy envía como mucho un mensaje por digest de image y hora. Ese límite vale para mensajes e incidencias. No retrasa la detención de un pod.

Una incidencia solo se abre cuando el evento está ligado a un digest de image: una por digest y regla de Falco, titulada `RUNTIME: <rule> in <image>` y con la etiqueta `runtime`. Se cierra sola como `quiet` tras 7 días sin que esa regla salte en esa image, o como `image_retired` cuando ningún pod en marcha ha usado el digest en 24 horas. Ninguna de las dos cosas significa que el problema se corrigiera. Un evento solo cuenta como notificado si al menos un canal activo está suscrito para recibirlo.

Events



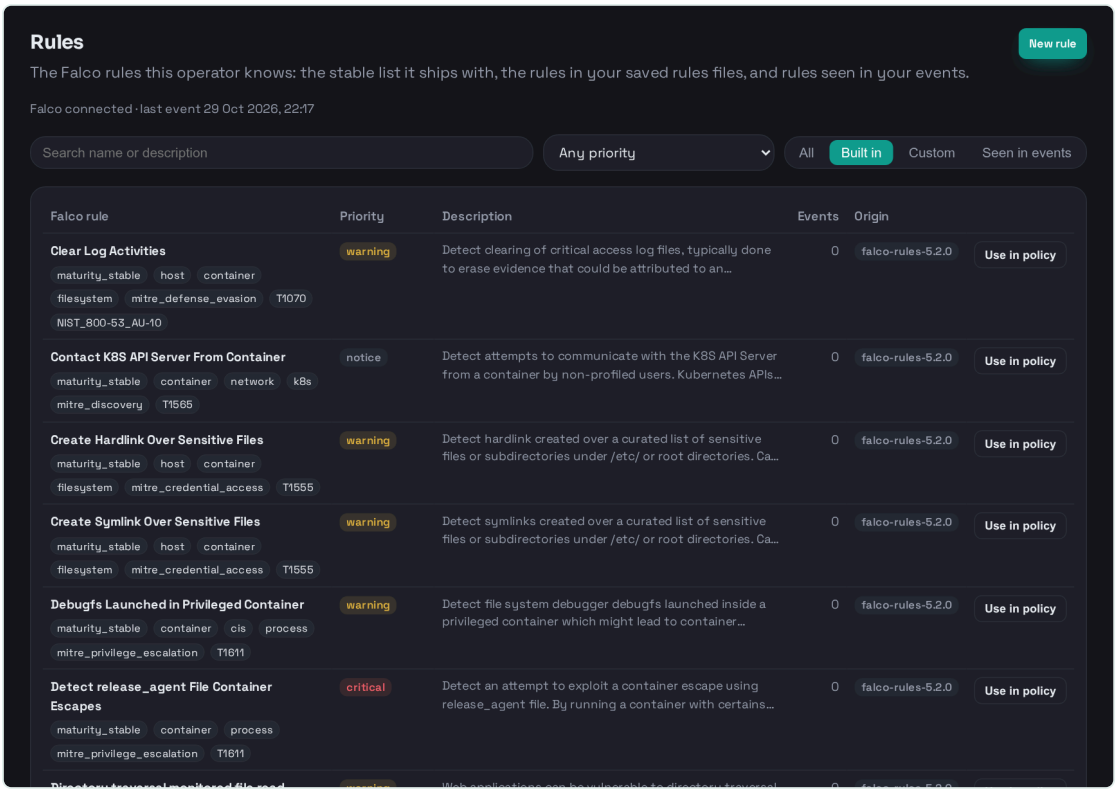
Falco > Events de las últimas 24 horas: 32 eventos. Cada fila da la policy que coincidió y su acción, la prioridad, la regla de Falco, el namespace y el pod. Los eventos sin policy muestran *No policy* y se guardan como registro. Los filtros cubren policy, acción, texto libre, prioridad, namespace, pod, regla de Falco y etiqueta, en un intervalo de tiempo.



Un evento abierto: *Drop and execute new binary in container* en `payments`, con la decisión *Stop the pod* de `payments-stop`. El panel muestra la decisión, si se envió un mensaje, el hash de la policy, la línea de salida de Falco, las etiquetas y el namespace, el pod, el container, la image y el node a los que se asocia el evento. En esta instalación el chart no define `rbac.allowEnforce`, así que la detención se rechazó y el pod siguió en marcha.

Reglas de Falco y el editor de reglas (beta)

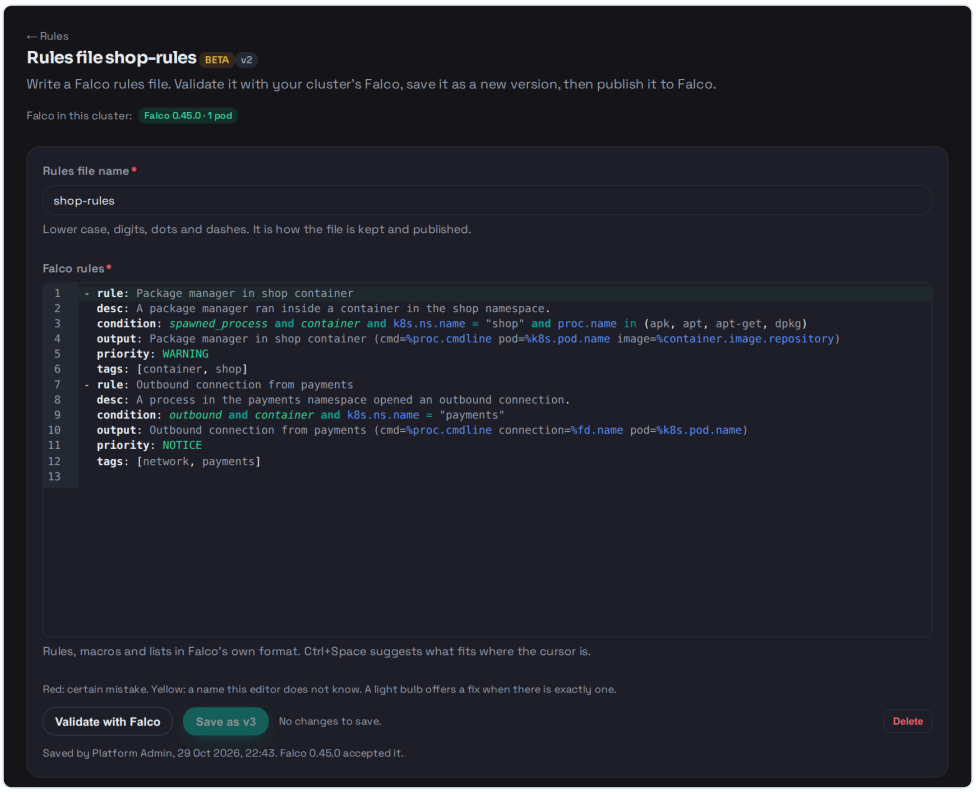
Falco > Rules lista las reglas de Falco que conoce el operator: las 25 reglas estables de `falcosecurity/rules` que vienen con él, las reglas de sus archivos de reglas guardados y los nombres de reglas vistos en eventos. *Use in policy* crea una policy a partir de una regla.



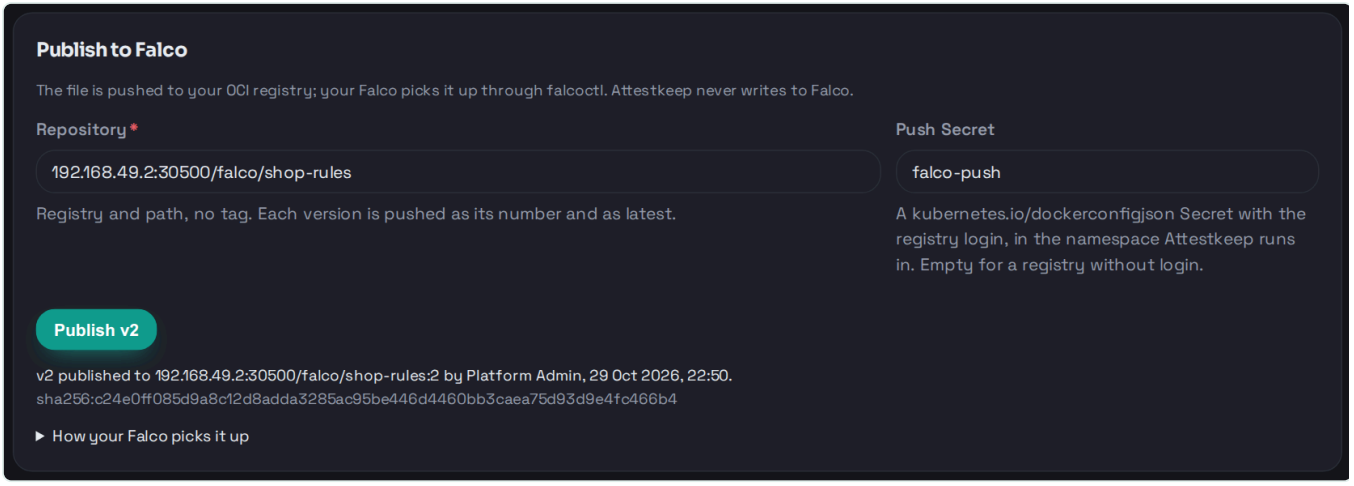
Falco > Rules con el filtro *Built in*. Cada regla muestra su prioridad, sus etiquetas, su descripción, cuántos eventos ha producido aquí y su origen, `falco-rules-5.2.0`. *Use in policy* empieza una policy de runtime a partir de la regla.

New rule (admins) abre un editor para un archivo de reglas de Falco: reglas, macros, listas y overrides en el YAML propio de Falco. El editor está en beta. El autocompletado y las comprobaciones siguen la versión de Falco que corre en su cluster, encontrada por la etiqueta `app.kubernetes.io/name=falco`.

- *Validate with Falco* arranca en el namespace del operator un pod puntual con la misma image que usa su Falco. Carga las reglas por defecto de Falco y su archivo, ejecuta Falco en modo de validación y se borra en cuanto se lee el resultado. No se toca el Falco que tiene en marcha.
- Guardar solo se habilita cuando Falco ha aceptado exactamente el texto del editor. El servidor lo vuelve a validar antes de guardarlo. Cada guardado es una versión nueva, registrada con quién la guardó y qué versión de Falco la aceptó. Se rechaza un nombre de regla que ya define otro archivo guardado.
- Publish sube una versión guardada a su propio registry OCI como artefacto `rulesfile` de falcoctl, con el número de versión y `latest` como tags. Si el registry pide inicio de sesión, usted da el nombre de un Secret `kubernetes.io/dockerconfigjson` del namespace del operator. Attestkeep lo lee al subir y no guarda la credencial.



El archivo de reglas guardado `shop-rules` en su versión 2. La línea bajo el título muestra el Falco encontrado en el cluster: Falco 0.45.0 en 1 pod. *Save as v3* sigue desactivado hasta que el texto cambie y Falco lo acepte. La última línea dice quién guardó la versión 2 y qué versión de Falco la aceptó.



Publish to Falco: el repositorio en su propio registry, el nombre del Secret de push y la constancia de que la versión 2 se publicó con su digest. *How your Falco picks it up* abre los valores de falcoctl para su chart de Falco.

Su Falco carga el archivo mediante falcoctl. En los valores del chart de Falco, añada su referencia a `install.refs` y a `follow.refs`, y conserve `falco-rules:5`, la referencia por defecto del chart. Helm sustituye las listas en lugar de fusionarlas, y sin ella falcoctl deja de instalar `/etc/falco/falco_rules.yaml`. Después añada su archivo a `rules_files`, detrás de las reglas propias de Falco. Borrar un archivo en la consola quita sus reglas de la lista Rules. Lo ya publicado sigue en el registry y en cada Falco que lo instaló hasta que retire la referencia de falcoctl.

Retención y métricas

Todos los eventos se guardan. Attestkeep no tiene un umbral de prioridad propio. Los eventos guardados con más antigüedad que `runtime.retentionDays` (90 por defecto) se borran cada hora, y `0` los conserva sin límite. El historial de incidencias, las entradas de auditoría y las revisiones de policies no se borran. Para que Falco no envíe eventos que nunca quiere guardar, ajuste su propio `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}
```

En `attestkeep_runtime_rule_matches_total`, `rule` es el nombre de la policy de runtime. `unresolved_image` no significa que el evento se perdiera. Falco también vigila procesos en el host, y un evento así se guarda sin image.

9. Comprobar que el control se aplica

Un webhook de admission que no funciona se ve igual que un cluster que cumple. Esta comprobación dura unos cinco minutos, solo usa kubectl y termina con un rechazo que usted mismo ha visto. Hágala tras instalar, tras cada actualización y siempre que necesite confirmar que el control se aplica.

1. Probar un pod que la policy debería rechazar

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

El API server lo rechaza y cita el webhook. Esta es la salida de una instalación con la policy por defecto:

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

El mensaje empieza por *Error from server*, así que el rechazo vino de la cadena de admission del API server. El motivo nombra la regla de la policy, y un desarrollador que lo lea en un log de CI sabe qué cambiar.

2. Confirmar que no se creó nada

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

3. Comprobar que la decisión quedó registrada

La tabla *Recently blocked admissions* del panel empieza ahora por este rechazo, con el mismo motivo. La métrica `attestkeep_admission_decisions_total{decision="denied"}` ha subido.

4. Si el pod se admitió

- El namespace se omite: kube-system, cert-manager, el namespace del propio operator o un namespace con la etiqueta `attestkeep.com/webhook=bypass`.
- La policy no cubre lo que probó. Compruebe `enforcement: Enforce` y `blockLatestTag: true`.
- El webhook está dejando pasar al fallar. Compruebe que `attestkeep_webhook_distrusted` vale 0 y que los pods están listos.
- La instalación no está activada. Hasta que se introduce una clave no se aplica nada, y cada respuesta de admission lo dice en un aviso.

Con DenyUntilScanned

Con `admissionTiming: DenyUntilScanned`, una image nueva se rechaza hasta que existe su escaneo. Salida de una instalación Community:

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

# poco después, el mismo comando:
pod/cold-check created
```

10. Paquetes de evidencia y modelo de confianza

Un paquete de evidencia es un documento firmado por periodo de auditoría. Se genera dentro del cluster a partir de sus propios registros, y un auditor puede verificarlo con sus propias herramientas. Es evidencia para una auditoría, no una certificación, y el documento lo dice.

Los textos a los que se asigna

Texto	Qué se asigna
SOC 2, AICPA Trust Services Criteria	13 criterios de CC6 a CC9 y Availability A1.
CRA y NIS2 de la UE: (UE) 2024/2847 y artículo 21(2)	34 requisitos entre los dos textos.
DORA, Reglamento (UE) 2022/2554	22 requisitos de los artículos 8 a 11 y del artículo 28.
NIST SP 800-190	15 contramedidas para images, registries, orquestadores y containers.
SSDF, NIST SP 800-218 v1.1	13 prácticas de PO, PS, PW y RV.
NIST SP 800-53 Rev. 5	15 controles seleccionados de CM, RA, SI, SR, AU y CP.
ISO/IEC 27001:2022 anexo A	12 cláusulas: 10 controles del anexo A y cláusulas marcadas fuera de alcance.
RGPD artículo 32	6 cláusulas sobre la seguridad del tratamiento.
PCI DSS v4.0.1 (PCI SSC, junio de 2024)	13 cláusulas de los requisitos 6.3, 6.5, 10.2 y 11.3.
MAS Technology Risk Management Guidelines (18 de enero de 2021)	30 cláusulas de las secciones 3.3, 6.1, 6.3, 7.2 a 7.6, 11.3, 11.4, 12.1, 12.2, 13.1, 13.6 y 15.
ASD Essential Eight Maturity Model (noviembre de 2023)	46 cláusulas: parches de aplicaciones, parches de sistemas operativos y application control, Maturity Levels uno a tres.
ISMAP Control Criteria (edición japonesa del 1 de agosto de 2024)	42 cláusulas de los criterios de control de la tabla adjunta 3.
ISMS-P Certification Criteria (notificación de PIPC y MSIT, anexo 7, guía de KISA de noviembre de 2023)	32 cláusulas de los dominios 1 y 2.

Trece textos y 293 cláusulas en total, cada una con su propia referencia. La asignación viene dentro del operator, así que un cluster aislado informa igual que uno conectado. Cada control termina en uno de cuatro estados: *met*, *partial*, *gap* u *outside_scope*. Una cláusula solo se cumple cuando los registros del periodo muestran el control funcionando, nunca solo porque el producto tenga la función. Una cláusula que un cluster no puede responder se informa como fuera de alcance, con el motivo, y está incluida en el total anterior.

El número de frameworks para los que puede generar paquetes depende de la licencia (capítulo 13).

Fuentes traducidas. Dos asignaciones se leen de textos no publicados en inglés, y cada cláusula lo dice. Para **ISMAP** existe una traducción inglesa de referencia, pero donde su entrada solo remite a ISO/IEC 27002, la obligación se traduce de la edición japonesa del 1 de agosto de 2024 y la referencia indica cuál. Para **ISMS-P** no hay texto inglés oficial. Cada cláusula es nuestra traducción del criterio coreano tal como aparece en la guía de KISA. Cada criterio de ISMS-P combina una medida técnica con un procedimiento, una designación o una revisión periódica, y los registros de un cluster solo responden a la parte técnica, así que ningún criterio de ISMS-P se informa como cumplido del todo.

Cinco attestations alimentan las asignaciones más recientes. `patch-window-48-hours` se cumple cuando el plazo declarado para los hallazgos críticos es de 48 horas como máximo y las vulnerabilidades explotadas se rechazan en la

admission. `patch-window-two-weeks` se cumple cuando el plazo declarado para los hallazgos altos es de dos semanas como máximo. `runtime-observed` se cumple cuando la integración de Falco está activada y llegó al menos un evento en el periodo. Es parcial cuando la integración está activada y no llegó nada, porque un cluster tranquilo y un Falco que dejó de enviar se ven igual desde el operator. Es una carencia cuando la integración está desactivada. `producer-statement-honoured` se cumple cuando la policy aplica las declaraciones VEX y confía en al menos un emisor. `context-aware-thresholds` se cumple cuando la escalada está activada para `exposed` y `privileged`, es parcial con una de las dos y una carencia con `Ignore`. `remediation-windows` solo puede llegar a *met* con un gestor de incidencias conectado, porque solo entonces hay registro de cuándo llegó una corrección.

Cómo se respondió cada requisito

Cada requisito lista las pruebas que lo respondieron. Una prueba tiene un método (`TEST` cuando lee un registro de lo ocurrido, `EXAMINE` cuando lee una configuración), un resultado, una observación, rutas dentro del informe, la fuente que leyó y cuándo. Cada requisito indica además `result` e `implementation_status` en términos de OSCAL. El PDF imprime las pruebas bajo cada requisito.

El bloque `generation` dice cuándo y con qué se generó el documento, el sistema de origen, la ventana de consulta (fin excluido, en UTC), la consulta de cada sección con su número de filas y si una consulta con tope alcanzó su tope, y cómo se nombra a las personas. Desde la 1.3.0 lleva también `nodes {count, ceiling}`: los nodes Ready del cluster al generarse y el tope de nodes de la licencia, solo como números. `admission.records[]` lleva cada revisión de admission del periodo, una fila cada una y sin tope, para que un auditor pueda muestrear sobre toda la población. `organisation.declarations` lleva la fuente de hora y la norma de conservación del operador, impresas como declaradas por él.

A las personas se las nombra por cuenta, no por nombre. `generated_by`, `chain[].actor` en las filas del registro de auditoría, `triage[].triaged_by` y `runtime.audit_entries[].actor` contienen `user:<id de cuenta>`. Un pod detenido por una policy de runtime se escribe como `policy:<nombre>`. El registro de auditoría de la consola sigue mostrando nombres, y el operador lo usa para resolver un identificador cuando un auditor pregunta. Las filas anteriores a la 1.2.0 que solo llevan un nombre se asocian a una cuenta por nombre visible o correo exactos, o se escriben como `user:unresolved` y se cuentan. La identidad de Kubernetes detrás de cada admission se escribe tal cual cuando es una identidad `system:`, y en otro caso como `k8s-user:` seguido de los primeros 16 caracteres hexadecimales de su SHA-256.

La sección de runtime lleva el número de eventos por prioridad, las reglas de Falco más activas, las alertas enviadas y las incidencias de runtime abiertas y cerradas. También lleva el conjunto de policies de runtime vigente con su hash de revisión, las `audit_entries` de los cambios de policy, los pods detenidos y los cambios de integración del periodo, las coincidencias por policy y acción, los eventos sobre los que actuó una policy (hasta 500, cada uno con la incidencia que abrió) y la cobertura: si la integración está activada, el último evento y el hueco más largo sin eventos. `event_time` es la marca de tiempo propia de Falco. `received_at` es cuándo recibió el operator el evento. La cobertura usa `received_at`, y `coverage.clock` lo dice. `runtime.rules[].effect` y `runtime.decisions[].policy_effect` dan la acción que tenía configurada la policy. Si una detención se llevó a cabo lo dice la fila `runtime_pod_stop` de `runtime.audit_entries`.

Exportación externa y deny list. Desde la 1.4.0, el paquete tiene dos secciones más. *External export* lista los destinos de SIEM, el último lote que aceptó cada uno y los eventos abandonados en el periodo (`external_export` en `report.json`). *Deny list* lista las entradas vigentes y las añadidas o retiradas en el periodo, las admissions que rechazaron y dejaron pasar, las excepciones, cada take down y cada restauración y, para una entrada de ejecutable, el archivo, su hash y la versión (`deny_list` en `report.json`).



14 Deny list

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

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

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

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

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

La sección 14, *Deny list*, del mismo paquete. Al final del periodo había tres entradas en vigor. Afectaron a 75 admissions: 74 rechazadas y 1 admitida donde la policy del namespace audita la lista. Cada entrada muestra su valor, su motivo, quién la añadió y cuándo, y cuántas admissions rechazó y cuántas dejó pasar. Bajo las excepciones aparece el take down de `payments/Deployment/worker` con su motivo y la hora en que se restauró.

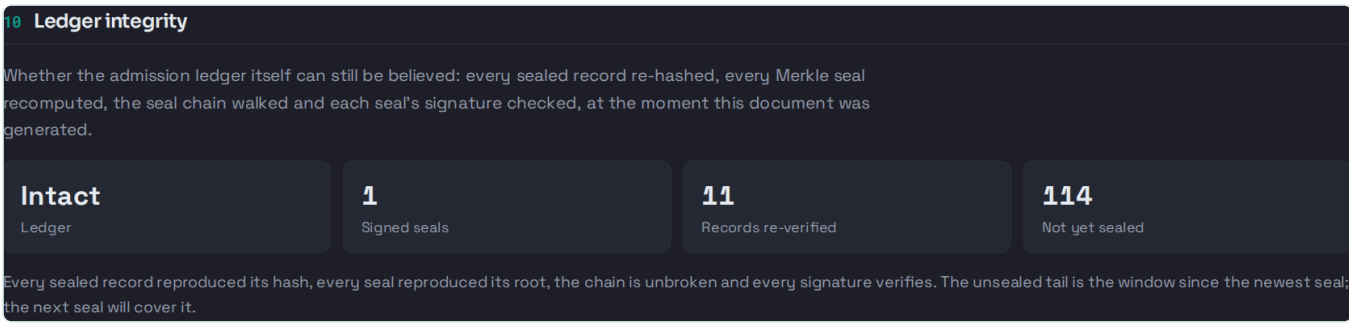
Los hallazgos se cuentan por digest. Desde la 1.4.0, un digest descargado con dos referencias, su tag y su digest o dos nombres, es una sola fila en `findings.critical` que nombra ambas en `references`, y sus hallazgos cuentan una vez en `coverage.severity` y `coverage.findings_total`. Un paquete generado ahora para el mismo cluster puede salir más bajo que uno de la 1.3.7, exactamente por esos duplicados.

Break-glass y triage. Desde la 1.3.4, cada activación de break-glass se lee de la revisión de policy con la que se decidieron sus admissions, de modo que un paquete dice lo que estaba aprobado entonces aunque la ventana ya se haya cerrado. Vuelva a generar cualquier paquete que cubra una ventana de break-glass pasada para obtener el registro completo. Las decisiones de triage llevan la gravedad de los hallazgos del escaneo, la más alta cuando una CVE se registró con varias.

Integridad del registro

Cada decisión de admission se escribe en PostgreSQL. Una base de datos es un buen sitio para consultar decisiones, pero no es, por sí sola, motivo para confiar en ellas. El registro está construido para que una manipulación se note:

- **Cada registro lleva su propio hash**, un SHA-256 de su contenido (hora, namespace, pod, image, digest, veredicto, motivo, nombre y hash de la policy, indicador de break-glass), calculado al escribirse.
- **Los registros se sellan bajo checkpoints firmados**, cada hora por defecto o cada 10.000 registros. La réplica líder construye una raíz de Merkle sobre el tramo sin sellar y firma un checkpoint que incluye el hash del anterior, de modo que los sellos forman una cadena.
- **La clave de firma no está en la base de datos.** Es una clave Ed25519 generada en su cluster y guardada en el Secret `attestkeep-evidence-key`. Reescribir filas del registro exige acceso de escritura a la base de datos. Volver a firmar los sellos exige acceso a ese Secret. Uno solo de los dos no basta. Desde la 1.4.0, la misma clave es también la identidad de licencia de la instalación (capítulo 12).
- **La misma clave firma sus documentos de evidencia** como sobres DSSE, y el certificado de licencia nombra su clave, de modo que un tercero puede ligar un documento a su instalación.



La comprobación del ledger dentro del paquete. Al generarlo se volvió a calcular el hash de cada registro sellado, se recalculó cada sello Merkle, se recorrió la cadena y se comprobó cada firma. Aquí: *Intact*, 1 sello firmado y 11 registros verificados de nuevo. 114 registros se escribieron después del último sello y cuentan como todavía sin sellar, por eso *Intact* se lee junto con el número de sellos.

Dónde acaban las garantías

Este esquema hace visible la manipulación. No la impide. Los registros escritos desde el último checkpoint están contados pero aún no sellados, una ventana de como mucho un intervalo de sellado. *Intact* habla de lo sellado, así que léalo junto al número de checkpoints. Un atacante con acceso de escritura a la base de datos y acceso a la clave de firma puede burlar el esquema. Lo que se afirma es que eso exige dos accesos distintos. El auditor recibe un resultado firmado y el sobre DSSE. Repetir la comprobación de Merkle de forma independiente exigiría acceso a la base de datos.

Firmar un paquete para un auditor produce un PDF junto al JSON, entregados juntos en un archivo firmado. Cualquiera puede comprobar un paquete entregado en attestkeep.com/verify.html con sus propias herramientas.

11. Operación, exportación al SIEM y copias de seguridad

Sondas

Endpoint	Qué comprueba	Se usa como
`/healthz`	Solo que el proceso está vivo.	Sonda liveness (retardo inicial 10s, periodo 20s)
`/readyz`	Desde la 1.4.0, que el servidor responde y su certificado está cargado. La base de datos ya no forma parte de ella: una réplica sigue lista mientras la base de datos no es alcanzable, y el webhook sigue rechazando con las reglas estáticas.	Sonda readiness (retardo inicial 5s, periodo 10s)

Qué hace una caída de la base de datos

- Las réplicas siguen listas y respondiendo. Una revisión de admission concede dos segundos en total a su trabajo en la base de datos. Después, las reglas estáticas deciden al momento: `:latest`, la lista de registries permitidos y la exigencia de digest siguen rechazando. El registro y los veredictos que dependen de la caché se degradan, y la caída se registra una vez.
- La base de datos se informa en `attestkeep_database_up` y en la página de salud. Si antes de la 1.4.0 vigilaba la disponibilidad del operator para detectar una caída de la base de datos, vigile esta métrica en su lugar.
- Después, el operator encuentra el hueco en el heartbeat y lo registra como caída, y la siguiente conciliación lista cada digest que entró sin decisión.
- Las notificaciones salen fuera de la petición, con 64 como máximo en espera. Las demás se descartan y se cuentan en `attestkeep_notifications_dropped_total`.
- El conteo de la cuota admite ante un error de la base de datos en lugar de bloquear despliegues.
- El escaneo se detiene con la base de datos y se reanuda con ella. La cola está en la base de datos, así que no se pierde nada.

Qué debería avisarle

Las métricas se sirven en el puerto 9090, en `/metrics`. Cada serie con etiquetas existe desde el arranque a cero, así que la falta de datos y cero rechazos son cosas distintas.

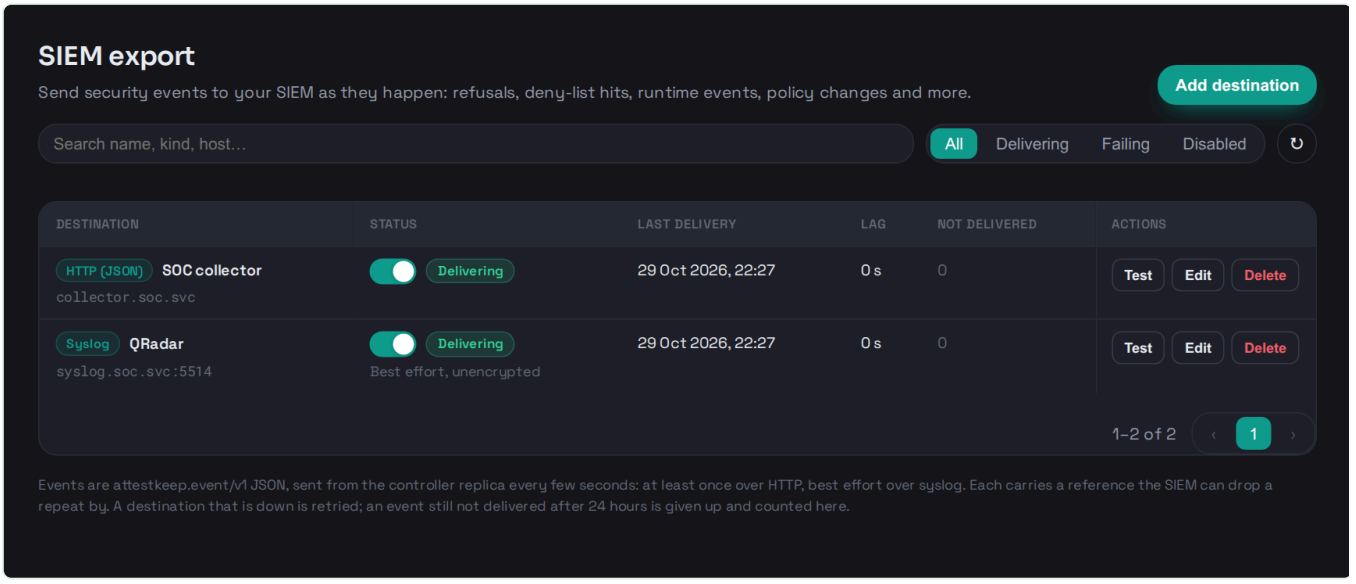
Situación	Señal	Por qué importa
Se está saltando la admission	`attestkeep_webhook_distrusted == 1`	Mientras valga 1 con `failurePolicy: Ignore`, los despliegues pasan sin control. Trátelo como una alerta inmediata.
Algo entró	`attestkeep_runtime_unmatched_digests > 0`	La conciliación encontró digests en marcha sin una admission permitida detrás.
Problemas con la base de datos	`attestkeep_database_up == 0`	Desde la 1.4.0, las réplicas siguen listas durante una caída, así que las alertas de readiness ya no la muestran.
El escaneo se queda atrás	`attestkeep_scan_queue_depth` que se mantiene alto	Una cola que solo crece significa que el escáner no llega a un registry o a su base de datos.
Los datos de vulnerabilidades envejecen	`attestkeep_vulnerability_db_age_seconds`	Los veredictos están tan al día como la base de datos que los respalda.
Cuota de images cerca del límite	`attestkeep_image_quota{kind="count"}` frente a `{kind="limit"}`	En el límite solo se rechazan los nombres de image no vistos en el periodo. Los workloads en marcha siguen.
Un destino de SIEM se queda atrás	`attestkeep_siem_lag_seconds{target}`, `attestkeep_siem_events_total{outcome="gave_up"}`	El retraso es la antigüedad del evento más viejo que el destino no ha recibido. Los eventos sin entregar a las 24 horas se abandonan.
Por encima del tope de nodes	`attestkeep_nodes{kind="count"}` frente a `{kind="ceiling"}`, `attestkeep_node_overage_seconds > 0`	El indicador de exceso dice cuánto tiempo lleva el cluster por encima de su tope. Avise mucho antes de los 14 días, cuando empiezan a rechazarse nombres de image nuevos.

Los rechazos se cuentan en `attestkeep_admission_denials_total{reason}`. Entre los motivos están `threshold`, `signature`, `attestation`, `tag`, `digest`, `registry`, `quota`, `unscanned`, `scan_error`, `end_of_life`, `known_exploited`, `provenance` y `context`. Los rechazos por el tope de nodes cuentan como `quota`. Otras series útiles para un panel: `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` y las series de runtime del capítulo 8.

Los eventos de notificación cubren lo mismo: `threshold_80` y `threshold_100` para la cuota de images, `nodes_80`, `nodes_over`, `nodes_enforced` y `nodes_cleared` para el tope de nodes, `license_expiring` cuando a un certificado en línea le queda menos de una semana, `trivy_db_failed`, `unverified_workload`, `runtime_alert` y, desde la 1.4.0, `deny_list_hit`, `deny_list_entry_added` y `deny_list_entry_removed`. Un canal nuevo trae marcados `nodes_over`, `nodes_enforced`, Runtime alert y Deny list hit.

Exportación al SIEM

Su equipo de seguridad mira el SIEM, no esta consola. Desde la 1.4.0, Settings > SIEM export le envía los eventos de seguridad de Attestkeep en el momento en que ocurren, para que queden en la misma línea de tiempo que todo lo demás que recoge. No sale nada hasta que añade un destino, y añadirlo no cambia ninguna decisión de admission. Todos los que inician sesión ven los destinos y cómo va cada uno. Añadir, cambiar, probar y retirar un destino corresponde a un administrador.



Settings > SIEM export con dos destinos. *SOC collector* recibe HTTP JSON y *QRadar* recibe syslog en formato LEEF sobre TCP simple. Los dos muestran *Delivering*, sin retraso y sin eventos abandonados. La fila de syslog dice lo que significa TCP simple: best effort, sin cifrar.

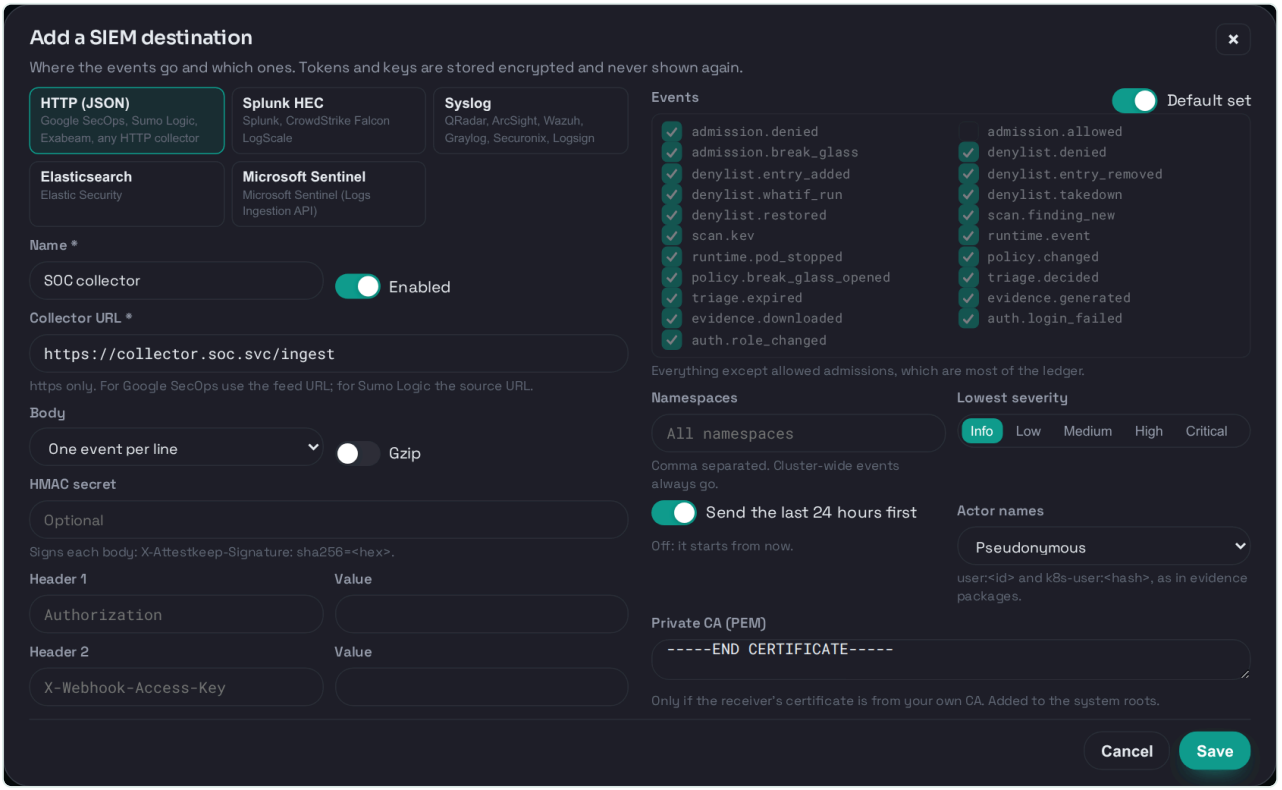
Qué se envía. Las admissions rechazadas, las coincidencias de la deny list, las entradas añadidas y retiradas, los take downs y las restauraciones, los eventos de runtime y los pods detenidos por una policy de runtime, los cambios de policy, de break-glass y de triage, la generación y la descarga de evidencias, los inicios de sesión fallidos y los cambios de rol. Un hallazgo crítico se envía una vez, con el escaneo que lo informó primero, y uno explotado una vez, con el escaneo que lo vio primero en la lista KEV. Las admissions permitidas están desactivadas salvo que las marque, porque son la mayor parte del registro. Cada destino filtra por tipo de evento, namespace y gravedad mínima, y uno nuevo puede empezar por las últimas 24 horas.

El evento. Cada evento es un objeto JSON con el esquema `attestkeep.event/v1`: hora, tipo, gravedad, cluster, namespace, pod, workload, image, digest, policy, actor, resultado, motivo y una `reference` al registro del que procede. La referencia es la misma cuando un evento se reenvía, así que su SIEM puede descartar repeticiones con ella. Un evento nunca lleva un SBOM, una lista de hallazgos, una credencial, datos de licencia ni lo que se escribió en un inicio de sesión fallido. Un destino puede configurarse para nombrar a las personas como `user:<id de cuenta>`, igual que el paquete de evidencia.

Destino	Para	Cómo
HTTP (JSON)	Google SecOps, Sumo Logic, Exabeam, cualquier colector HTTP	Un POST HTTPS por lote, un array JSON o un evento por línea. Gzip opcional, dos cabeceras de petición y una firma HMAC en `X-Attestkeep-Signature`.
Splunk HEC	Splunk, CrowdStrike Falcon LogScale	Eventos HEC con source `attestkeep` y sourcetype `attestkeep:event`.
Syslog	QRadar, ArcSight, Wazuh, Graylog, Securonix, Logsign	RFC 5424 sobre TLS, o TCP simple para un receptor sin TLS, con cuerpo CEF, LEEF o JSON.
Elasticsearch	Elastic Security	La API bulk hacia un índice o un data stream, con una API key.
Microsoft Sentinel	Microsoft Sentinel	La Logs Ingestion API a través de una data collection rule, con una aplicación de Entra.

Entrega. Los eventos se envían al menos una vez por HTTP y con el mejor esfuerzo por syslog, que no tiene acuse de recibo. Salen en orden de commit, cada pocos segundos, como mucho 100 por petición, desde la réplica que hace el trabajo periódico. Un destino que falla se reintenta con esperas crecientes, o tras su `Retry-After`, y nunca retiene a otro. Un destino retenido detrás de una transacción abierta en la base de datos muestra *Waiting*. Un evento que sigue sin entregarse 24 horas después de ocurrir se abandona y se muestra como "N events not delivered", y la sección External export del paquete de evidencia lo lista. *Test* envía un evento por el destino y muestra la respuesta.

Credenciales. Los tokens, las claves y los secretos se cifran con la data key de la instalación y no se vuelven a mostrar. Cuando una edición cambia adónde apunta un destino, las credenciales guardadas se eliminan, para que nunca se envíen al sitio nuevo salvo que las introduzca allí. Se rechazan las direcciones loopback y link-local, entre ellas el servicio de metadatos de la nube. Los destinos HTTP deben usar `https://`, y un receptor con un certificado de su propia CA recibe esa CA en el formulario.



Add a SIEM destination con HTTP (JSON). La URL del colector tiene que ser https. El cuerpo lleva un evento por línea, el conjunto por defecto es todo salvo las admissions permitidas, y *Send the last 24 hours first* envía primero el día anterior al alta del destino y luego sigue con los eventos nuevos. Los nombres de actor son seudónimos, como en los paquetes de evidencia. Si el certificado del receptor es de su propia CA, esa CA se pone aquí y se añade a las raíces del sistema. Los tokens y las claves se guardan cifrados y no se vuelven a mostrar.

La exportación al SIEM necesita PostgreSQL 13 o posterior, porque lee en orden de commit. El PostgreSQL incluido es la versión 16. En un servidor externo más antiguo no se puede activar ningún destino, y la página muestra *Database too old*. Sin destino, los propios logs, trazas y métricas del operator pueden ir igualmente a un colector OpenTelemetry con `observability.otlp.endpoint`. docs.attestkeep.com/siem explica la configuración de cada producto de SIEM.

Qué copiar

Attestkeep no hace copias de la base de datos. Ejecutar, programar y guardar el volcado es tarea suya. Todo lo que sabe el operator está en PostgreSQL y en recursos del cluster. La copia que importa es la de la base de datos: hallazgos, decisiones, excepciones, el registro de auditoría, el archivo de evidencias, los usuarios y el certificado de licencia están ahí.

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

Si usa su propio PostgreSQL, sus copias actuales ya lo cubren. Una instantánea del volumen del PVC de la base de datos incluida también sirve si su storage class lo admite. Junto a la base de datos, copie tres Secrets:

- `attestkeep-evidence-key`, la clave de firma de las evidencias, creada por el operator. Si se pierde, se genera una nueva, y los checkpoints y documentos firmados con la anterior ya no se verifican con la clave actual de la instalación. El historial se conserva. Desde la 1.4.0 es también la identidad de licencia: perderla significa que la siguiente comprobación diaria recibe `401` y el cluster funciona como Community hasta que se vuelva a activar.
- `attestkeep-data-key`, generada por el chart salvo que haya aportado `api.dataKey.existingSecret`. Si se pierde, los tokens guardados de los gestores de incidencias no se pueden descifrar y cada canal de incidencias necesita que se vuelva a introducir su token.
- `attestkeep-session`, la clave de sesión. Si se pierde, todos vuelven a iniciar sesión.

El certificado del webhook no está en la lista, porque el operator lo vuelve a emitir. La clave de activación tampoco: reinstalar en el mismo cluster da la misma huella y reutiliza la plaza de licencia existente. Con la integración de Falco activada, el operator crea además `<release>-falco-mtls`. Su certificado de cliente se renueva en Falco > Falco integration.

Settings > Backup exporta además un bundle JSON de la configuración que montó a mano: canales de notificación, decisiones de triage, la lista de usuarios, las policies y los destinos de SIEM. Sirve para montar un segundo cluster igual. Las credenciales nunca van en el bundle, y los destinos de SIEM importados llegan desactivados. La importación muestra antes una vista previa, los registros existentes siempre ganan y los usuarios se listan, no se crean. Desde la 1.3.1, cada exportación e importación aparece en *Backup history*. El bundle no sustituye al volcado de la base de datos.

Una restauración no oculta lo perdido. En la primera pasada tras una restauración, el operator encuentra el hueco de heartbeat desde el volcado y registra todo ese tramo como un periodo en el que el control no funcionaba. Un paquete de evidencia de ese periodo muestra el hueco como cualquier otra caída. Restaurar un volcado en la misma versión del operator o en una más nueva está admitido. Restaurar un volcado más nuevo en un operator más antiguo no lo está.

Disponibilidad

Operator y webhook: 2 réplicas, anti-afinidad preferente, PodDisruptionBudget minAvailable 1. Desde la 1.4.0, una réplica que se detiene se vacía: su hook preStop espera `webhook.preStopSleepSeconds` (5 por defecto) mientras el cluster deja de enviarle tráfico, después mantiene su receptor abierto cinco segundos más y termina las revisiones en curso. El periodo de terminación del pod es ese valor más 30 segundos. Suba `replicaCount` para escalar. El trabajo del controlador se serializa con un advisory lock, así que es seguro. Escáner: 1 réplica. El chart rechaza varias réplicas sobre un volumen de caché ReadWriteOnce y dice por qué. PostgreSQL incluido: una réplica por diseño. Para replicación y copias gestionadas, apunte `externalDsn` a su propio PostgreSQL.

GitOps. El control ocurre en la admission del pod, así que da igual que el cambio lo aplique kubectl, Argo CD, Flux o un Job. Si una sincronización de GitOps crea un pod cuya image no pasa la policy, la sincronización falla con el mensaje de rechazo como error.

Clusters aislados. Trivy y cosign vienen dentro de la image. Copie la base de vulnerabilidades con `oras copy ghcr.io/aquasecurity/trivy-db:2 registry.internal/mirror/trivy-db:2` y fije `trivy.dbRepository`. Active `scan.cosignOffline` para verificar con claves sin el registro público de transparencia. Una copia del catálogo KEV puede comprimirse con gzip en una ConfigMap que nombra `kev.existingConfigMap`, con `kev.source` apuntando a `file:///etc/kev/<key>`. La licencia necesita una ruta de salida a lic.attestkeep.com:443 o la activación sin conexión (capítulo 12).

12. Licencias

Una clave de activación por compra, y una clave Community por cuenta. Una clave se liga a tantos clusters como permita su plan, y usted mismo puede mover una plaza.

Activación

La activación se hace una vez por cluster y ocupa una plaza. En el primer arranque, el operator genera su clave de firma dentro del cluster, guarda la parte privada en el Secret `attestkeep-evidence-key`, que comparten todas las réplicas, y envía solo la parte pública con la solicitud de activación. Desde la 1.4.0 es la misma clave que firma los paquetes de evidencia, de modo que el certificado nombra la clave con la que un auditor los verifica. Una instalación en línea activada antes de la 1.4.0 sigue con el par de claves con el que se activó y pasa a la clave única en su siguiente activación. La solicitud lleva la clave de activación, una huella del cluster (un hash del UID del namespace kube-system, ni un nombre ni una dirección), el nombre del cluster si lo ha fijado y la versión del operator. La respuesta es un certificado firmado válido durante treinta días como máximo. El operator lo verifica dentro del cluster con una clave pública integrada en el binario.

Desde la 1.3.0, el operator solo acepta un certificado que lleve el tope de nodes. Un certificado emitido por un servicio de licencias anterior se rechaza con "re-activate with a key issued by licence service 1.3.0 or later".

Si el operator no llega al servidor de licencias, lo dice en lugar de declarar la clave no válida. Un certificado en el que el operator no confía, una conexión rechazada o agotada y un nombre que no se resuelve reciben `502`, con el servidor y la causa. La consola muestra "The licence server could not be reached: its certificate is not trusted, or the connection failed. If it uses a private CA, set license.caBundle in the chart." Una respuesta `5xx` del servidor de licencias se convierte en `503` con "The licence server answered with an error. Try again later." La clave nunca forma parte de un mensaje. `license.caBundle` recibe certificados PEM en los que el cliente de licencias confía además de las raíces del sistema, para un proxy que intercepta TLS o un servidor de licencias bajo una CA privada. Se fija con `--set-file license.caBundle=ca.pem`. La verificación de certificados sigue activada.

La comprobación diaria

Una vez al día, la instalación envía tres valores: el identificador de licencia, la huella del cluster y la versión del operator, firmados con la clave de la activación. Desde la 1.4.0, la comprobación la hace una sola réplica del operator, y el escáner ya no contacta con el servidor de licencias. La respuesta es un estado, más un certificado nuevo cuando algo cambió o el actual está por acabarse. No se envía nada sobre sus images, workloads, policies ni hallazgos. No hay canal de telemetría ni ajuste que lo añada. Corran los pods del operator que corran, un cluster hace esta llamada una vez al día.

Cuando el servicio de licencias no es alcanzable

La instalación sigue con el certificado que tiene hasta que caduca. Una comprobación fallida se reintenta a las cuatro horas. Cuando a un certificado en línea le queda menos de una semana, la consola avisa y se envía la notificación `license_expiring`. Si el certificado caduca, la instalación continúa como Community Edition. No se detiene ni empieza a rechazar despliegues. Una respuesta `revoked` o `released` se aplica en la siguiente comprobación.

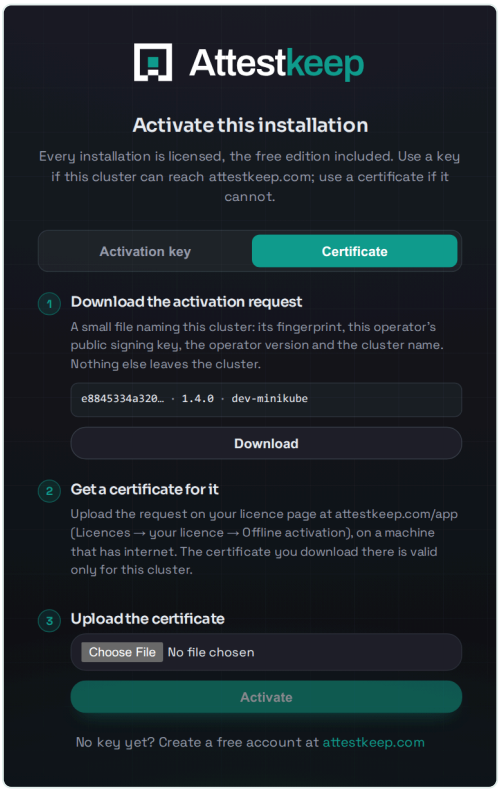
Plazas de cluster

Cada activación ocupa una plaza hasta que se libera. Las plazas las libera usted desde su página de licencias en attestkeep.com. Una plaza liberada queda libre al momento, y la instalación que la ocupaba se cae en su siguiente comprobación diaria. Reinstalar en el mismo cluster no ocupa una segunda plaza, porque la huella es la misma. Si se queda sin plazas, el error nombra los clusters que las ocupan y cuándo se activó cada uno.

La consola también puede devolver una licencia: *Deactivate* en la página Usage and License, tras una confirmación. En línea, eso libera la plaza del cluster. Sin conexión, borra el certificado del cluster y deja la plaza para que la libere el soporte de Attestkeep, porque un cluster sin conexión nunca vuelve a informar.

Activación sin conexión

Un cluster que nunca llegará al servicio de licencias se activa con un certificado en lugar de una clave. La pantalla de activación y la página Usage and License tienen ambas una pestaña *Certificate* para ello.



La pestaña *Certificate* en tres pasos. Descargue la solicitud de activación, que identifica este cluster por su huella, la clave pública de firma del operator, la versión del operator y el nombre del cluster. Súbala en su página de licencia en attestkeep.com desde un equipo con acceso a Internet. Suba el certificado que reciba a cambio.

Una instalación activada sin conexión antes de la 1.4.0 conserva su licencia, pero su certificado nombra el par de claves anterior, así que una descarga de evidencia firmada se rechaza hasta que una nueva solicitud de activación, hecha desde Usage and License, se cambie por un certificado nuevo. La página Evidence y el rechazo lo dicen.

El certificado es el mismo documento firmado que guarda la vía en línea, marcado como sin conexión. El operator comprueba la firma, la huella del cluster y su propia clave antes de guardarlo. No hace ninguna comprobación diaria para él y avisa treinta días antes de que caduque. Un certificado para otro cluster u otra clave, o uno caducado, se rechaza con el motivo. Para una instalación cuya consola nadie abre, `license.existingCertificateSecret` nombra un Secret que guarda el certificado bajo la clave `certificate`.

Si la activación sin conexión está disponible para su licencia lo indica la página de licencias en attestkeep.com. En la lista de precios publicada viene con Enterprise en contrato anual, o con un acuerdo.

El tope de nodes

Cada plan permite un número de nodes Ready por cluster. El operator cuenta los nodes Ready del cluster cada `nodes.pollInterval` (5 minutos por defecto). El número se queda en el cluster. La comprobación diaria sigue enviando los mismos tres valores. El tope en sí está en el certificado.

Superar el tope no detiene nada. La consola y una notificación lo dicen, con una fecha 14 días después (`nodes.graceDays`). Si el cluster sigue por encima al cumplirse esos 14 días, los nombres de image no vistos en el periodo se rechazan en la admission con "node ceiling of N exceeded", y un nombre rechazado no se cuenta. Desde la 1.4.0 pasa lo mismo con la cuota de images: un nombre rechazado porque se alcanzó la cuota del mes no se cuenta, y el contador se detiene en el límite con *Allowance reached*. En la tarjeta de licencia de la página Usage and License, una franja de images llena muestra *Allowance reached* con las fechas propias de la licencia, no *Expired*, y el correo de la cuota dice que los nombres de image nuevos se rechazan en la admission, no que dejan de escanearse. Todo lo ya contado

sigue ejecutándose, escaneándose y controlándose. Volver por debajo del tope, con menos nodes o con más nodes en la licencia, reinicia la cuenta al momento. Un sondeo que no consigue contar los nodes no cambia nada. Ninguno de los dos valores del chart afloja el tope.

Cuando termina una licencia

La instalación continúa como Community Edition cuando termina la suscripción y caduca el certificado, cuando el certificado deja de verificarse o resulta ser de otro cluster, o cuando usted lo desactiva. Community significa 1 cluster, 1 framework, 10 nombres de image al mes y 1 node por cluster. No se descarga nada ni se emite un certificado nuevo. El escaneo, la admission y la evidencia siguen funcionando. Desde ese momento, la página Usage and License y el panel muestran la cuota de Community del mes en curso, las mismas cifras que aplica la admission. Los meses anteriores conservan la cuota que tenían. Cancelar a mitad de periodo no desactiva nada el día de la cancelación. Conserva lo que pagó hasta el final del periodo.

Elegir frameworks. Si se ha generado evidencia para más frameworks de los que ahora cubre la licencia, el siguiente owner o administrador que inicia sesión debe elegir cuáles conservar, hasta el número de la licencia. Mientras nadie elija, ninguno de los frameworks ya usados está abierto a un paquete nuevo, y a los demás usuarios se les dice que un administrador tiene que elegir. Los frameworks no conservados quedan cerrados para paquetes nuevos. Cada paquete ya generado sigue pudiendo descargarse. La elección se mantiene hasta que cambien el plan o el número de frameworks de la licencia, y se escribe en el registro de auditoría.

13. Ediciones y precios

Todas las funciones de seguridad están en la edición gratuita. El escaneo, la verificación de firmas, el control en la admission y los paquetes de evidencia están completos en Community, y todos los controles de policy, incluidos deny until scanned y `failurePolicy: Fail`, funcionan ahí. También la deny list, el take down, What if y la exportación al SIEM. Los planes de pago añaden capacidad e inicio de sesión único.

Edición	Al mes, sin IVA	Al año, sin IVA	Clusters	Frameworks	Nombres de image / mes, por cluster	Nodes / cluster	Además
Community	0 USD	0 USD	1	1, a elegir	10	1	Gestor de incidencias público
Team	600 USD	6.000 USD	1	1	30	6	Inicio de sesión único, soporte por correo
Business	1.800 USD	18.000 USD	3	3	50	10	Todo lo de Team, soporte prioritario por correo
Enterprise	3.600 USD	36.000 USD	5	Los trece	75	20	Activación sin conexión para clusters aislados, soporte por correo

Un año cuesta diez meses. Complementos de las ediciones de pago, al mes: cluster adicional +300 USD (hasta diez clusters por compra), framework adicional +120 USD, 10 nombres de image más al mes +30 USD por cada cluster que cubre la licencia, node adicional +50 USD por cada cluster que cubre la licencia, de uno en uno, hasta 50 nodes por cluster. Los precios anuales son diez veces los mensuales. En Community, un escalón de volumen cuesta +300 USD al mes por cada cluster. En Community no se venden nodes adicionales.

Más de diez clusters, sin tope de clusters o sin tope de nodes es un acuerdo a medida, no un precio de lista.

Los precios no incluyen el IVA. La compra y el pago se hacen en Polar, el merchant of record. La tarjeta se introduce en la página de pago de Polar y nunca llega a los servidores de Attestkeep. Los impuestos que correspondan se calculan allí a partir de su dirección de facturación y se muestran antes de pagar. Una empresa registrada a efectos del IVA en la UE lo declara ella misma por inversión del sujeto pasivo. La licencia y el soporte los presta Attestkeep.

No hay prueba con límite de tiempo. Community Edition tiene todas las funciones de seguridad y no caduca, así que puede probar el producto en su propio cluster todo el tiempo que necesite.

La unidad medida es el nombre de image: registry y repositorio, sin tag ni digest. Cuenta todo lo que se ejecuta en el cluster, images de terceros incluidas. Cuando se agota la cuota, solo se rechazan en la admission los nombres de image nuevos. Cada image ya contada en el periodo sigue escaneándose y controlándose.

14. Seguridad, soporte y con quién trata

Integridad de las releases

Cada image de release se construye para linux/amd64 y linux/arm64 y se firma con cosign sobre su digest, keyless a través del OIDC de GitHub y con la clave publicada en docs.attestkeep.com/cosign.pub. El escáner y el verificador de firmas vienen dentro de la image, no se descargan durante el funcionamiento. El código fuente no es público.

Informar de un error o una vulnerabilidad

Los errores van al gestor de incidencias público de github.com/attestkeep/attestkeep-docs, para todas las ediciones. Las vulnerabilidades van por el formulario privado de avisos del mismo repositorio o a security@attestkeep.com. Una persona acusa recibo de cada informe en tres días laborables, y los problemas corregidos se publican como avisos en cuanto una release lleva la corrección. El Reglamento (UE) 2024/2847 pide al fabricante un único punto de contacto para los informes de vulnerabilidades. El nuestro es security@attestkeep.com, y los mismos datos figuran en cada paquete de evidencia.

Soporte por plan

Plan	Soporte
Community	Gestor de incidencias público
Team	Correo electrónico
Business	Correo electrónico, con prioridad
Enterprise	Correo electrónico
Acuerdo a medida	Según el acuerdo

Attestkeep no publica un tiempo de respuesta garantizado para ningún plan.

Con quién trata

Nombre comercial	Attestkeep
Razón social	Ecavist Technologies, empresa registrada en Turquía
Domicilio registrado	Dörtyol Küme Evleri No: 369, İç Kapı No: 3, Işıklar, Merkez, Çanakkale, Türkiye
Avisos legales, contratos, facturación	legal@attestkeep.com
Soporte y preventa	support@attestkeep.com
Informes de seguridad	security@attestkeep.com

Los documentos vinculantes (condiciones de servicio, contrato de venta a distancia, aviso de privacidad, condiciones de tratamiento de datos, subencargados) están en attestkeep.com. Si una página en lenguaje llano y un acuerdo no coinciden, prevalece el acuerdo.

Anexo. Contra qué se comprobó este manual

Versión del producto	Attestkeep 1.4.0 (chart 1.4.0, appVersion 1.4.0). Notas de la versión con fecha del 28 de octubre de 2026, image y chart publicados el 29 de octubre de 2026
Image publicada	`ghcr.io/attestkeep/attestkeep-k8s:1.4.0` `sha256:bfe0299695e38779f02ac24d805088a373d0a754fd7f0f552751e0d9fa3f05e8`
Chart publicado	`oci://ghcr.io/attestkeep/charts/attestkeep` versión 1.4.0 `sha256:8d6c4ba1e3ff790f6394e83eb34e5a740eeeecb0b4148e5c7e48683ca9bc610f`
Instalación fotografiada	Una instalación cuya consola indica la versión 1.4.0, sobre Kubernetes v1.35.1 con dos nodes, como Enterprise Edition, con Falco 0.45.0 conectado por mutual TLS desde su propio `http_output`. Ejecuta una image compilada a partir del código fuente de 1.4.0, no la image publicada indicada arriba. Fotografiada el 29 de octubre de 2026 con el tema oscuro y la consola en inglés.
Paquete de evidencia mostrado	`PS-SOC2-DEVMINIKUBE-20261001-20261029-DA7379`, generado el 29 de octubre de 2026 a las 19:31 UTC por la versión 1.4.0
Documentación	Páginas de docs.attestkeep.com para la 1.4.0: releases, install, upgrade, what runs, admission, configuration, deny list, runtime events, SIEM export, licensing, backup, using the console, policies, try a policy before enforcing it, compliance, issue trackers y operations
Precios	attestkeep.com/pricing, USD, sin IVA. Las cifras no han cambiado desde la edición 1.3.2 y se compararon con la lista de precios del sitio el 29 de octubre de 2026.

Documentación: docs.attestkeep.com · Notas de versión: docs.attestkeep.com/releases · Verificador de paquetes de evidencia entregados: attestkeep.com/verify.html

Qué cambió desde la 1.3.2

Versión	Qué cambió
1.4.0, 28 de octubre de 2026	La deny list, consultada antes del break-glass y de cada regla de policy, con un modo `denyList` en cada policy, What if para una entrada, coincidencias por workload, workloads afectados y take down a mano con restauración (capítulo 7). La comprobación `attestkeep-binscan` lee los ejecutables dentro de las images, con límites de recorrido. Exportación al SIEM a cinco tipos de destino (capítulo 11). Verificación de firmas por clave con `signatureKeys`. `license.caBundle`, y los errores del servidor de licencias se informan como tales. Una detención de runtime necesita `rbac.allowEnforce`, se conceda lo que se conceda además. Las réplicas siguen listas durante una caída de la base de datos y se vacían al detenerse. Los dry runs no dejan rastro. Un nombre de image rechazado por la cuota no se cuenta, y una franja llena muestra Allowance reached. Secciones 13 y 14 en la evidencia, y hallazgos contados una vez por digest. Una sola clave para la licencia y la evidencia. Go 1.26.10, Trivy 0.76.1, cosign 3.1.4. Migraciones 0036 a 0046.
1.3.7, 15 de octubre de 2026	Los containers añadidos con `kubectl debug` se revisan en la admission (capítulo 3). La página Images pone cada image en uno de cuatro estados, con la nueva tarjeta Not compliant, y Blocked sigue a la última revisión de admission. Las ejecuciones de What if se guardan noventa días en Security > What-if history, con descargas en PDF y JSON que son una proyección, no evidencia. What if en cada fila de policy, Scan now para las images que una repetición encontró sin escanear, una confirmación antes de que Cancel descarte cambios, Apply to cluster solo para las cuentas que pueden escribir policies, y una sola lista de nombres de frameworks. El operator se compila con Go 1.26.9 y `golang.org/x/net` 0.60.0, que cierran vulnerabilidades publicadas a las que llega el operator. Migraciones 0033 a 0035.
1.3.6, 8 de octubre de 2026	La página Images cuenta solo los pods admitidos. El filtro de acción de Falco Events ofrece los cinco efectos. Correcciones de maquetación de la barra lateral y de la lista de evidencias a 1366 y 1600 px, y del indicador de uso. El formulario New policy propone los plazos de SLA y de triage documentados.
1.3.5, 8 de octubre de 2026	Una policy de Falco configurada como Stop the pod aparece así en todas partes, y la detención de un pod se escribe como `policy:<nombre>` en el paquete de evidencia. El cluster se nombra con el `clusterName` del chart. El Audit Log usa la hora local. La tarjeta Vulnerability database muestra cuándo comprobó por última vez y tiene Update now. La tarjeta del webhook ya no compara la latencia con un presupuesto fijo.
1.3.4, 8 de octubre de 2026	Correcciones de la evidencia: los detalles del break-glass vienen de la revisión de policy vigente en su momento, las decisiones de triage llevan la gravedad de la CVE, y la vista de paquete de la consola cuenta bien las revisiones rechazadas.
1.3.3, 7 de octubre de 2026	Las migraciones se ejecutan bajo un bloqueo de la base de datos, así que una instalación nueva o una actualización con migraciones ya no reinicia un pod una vez.