UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIPLOMADO: INFRAESTRUCTURA EN TECNOLOGÍAS DE LA INFORMACIÓN.

TRIVY + COSIGN (IMAGE SECURITY)

ALUMNOS:

- PATIÑO OSEGUERA ALEXIS.
- COLIN MOSQUEDA EDUARDO

MÓDULO: SEGURIDAD INFORMÁTICA

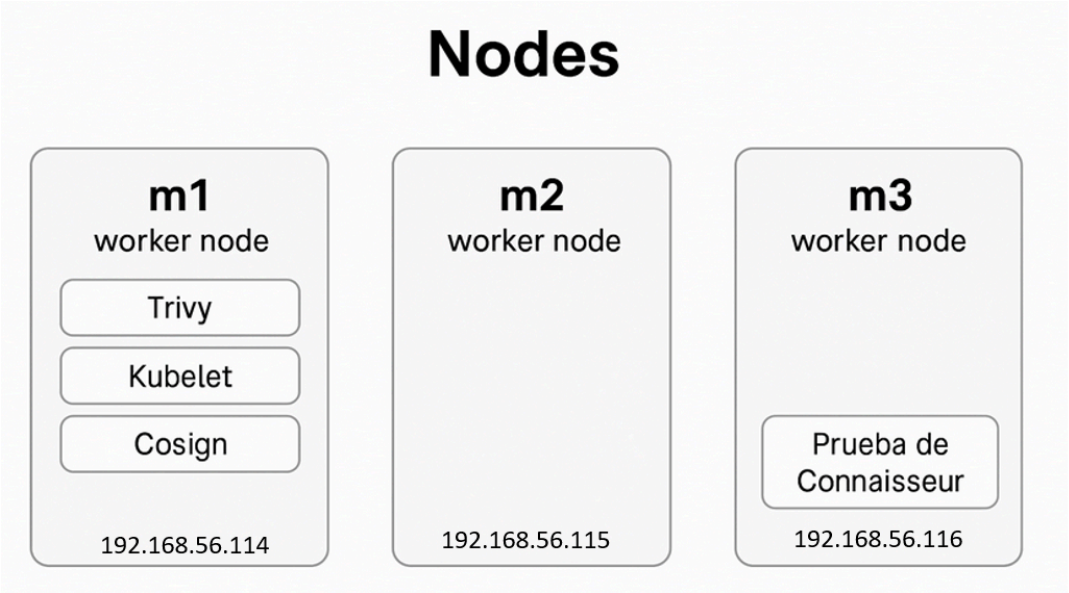
PROFESOR: ING. DANIEL GUERRERO RAMÍREZ

FECHA: 6 DE NOVIEMBRE DEL 2025.

Requisitos Previos

Requisito	Descripción	Verificación
Kubernetes ≥1.24	Clúster funcional ((1) control plane + nodos workers (2)).	kubectl get nodes -o wide
Red CNI Flannel o Calico	Configurada y en estado Ready.	kubectl get pods -n kube-flannel
Helm 3	Necesario para instalar Connaisseur.	helm version
kubectl	Configurado con contexto del clúster actual.	kubectl config current-context
Container Runtime (containerd o Docker)	Para manejo de imágenes locales.	systemctl status containerd
Podman o Docker CLI	Requerido para subir imágenes al registro local.	podman version
Acceso root o sudo	Permite modificar /etc/containerd y puertos.	whoami
Firewall abierto en puerto 5000	Registro local HTTP.	sudo firewall-cmd --list-ports

Entorno utilizado en guía de instalación



TRIVY

¿Qué es Trivy?

Trivy (de Aqua Security) es un escáner de vulnerabilidades de código abierto que analiza imágenes de contenedores, sistemas de archivos y dependencias de proyectos, comparando los paquetes detectados con bases de datos reconocidas como CVE (Common Vulnerabilities and Exposures) y CVSS (Common Vulnerability Scoring System). Cada hallazgo se clasifica según su nivel de severidad, proporcionando una visión clara del riesgo que implica cada componente del sistema. Trivy se ha consolidado como una herramienta fundamental para la detección temprana de vulnerabilidades en imágenes de contenedor, repositorios de código y configuraciones de infraestructura.

El funcionamiento de Trivy se basa en tres fases principales:

- Análisis de componentes: identifica los paquetes y dependencias incluidos en la imagen.
- Correlación con bases de datos de vulnerabilidades: consulta automáticamente las bases CVE/CVSS actualizadas.
- Generación de reportes: produce informes en distintos formatos (tabla, JSON o HTML) que permiten visualizar la severidad y descripción técnica de cada vulnerabilidad, junto con recomendaciones prácticas de mitigación.

El uso de Trivy no solo permite detectar vulnerabilidades antes del despliegue, sino que también facilita la gestión continua de seguridad dentro de los flujos DevOps. Al integrar sus escaneos en pipelines CI/CD o utilizarlos manualmente en un clúster Kubernetes, se promueve la creación de imágenes más seguras y el cumplimiento de buenas prácticas de ciberseguridad.

A continuación se mostrará una documentación sobre instalación, generación de reportes JSON + HTML y otros.

Guía de instalación de Trivy

1.Instalacion de Trivy (método oficial - local)

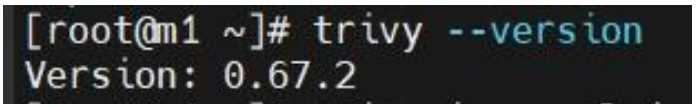
# prerequisitos (ejemplo en Rocky/Centos/Fedora)
sudo dnf -y install wget git tar curl

#tener instalado y en arranque docker o podman
sudo dnf install -y docker
sudo systemctl enable --now docker

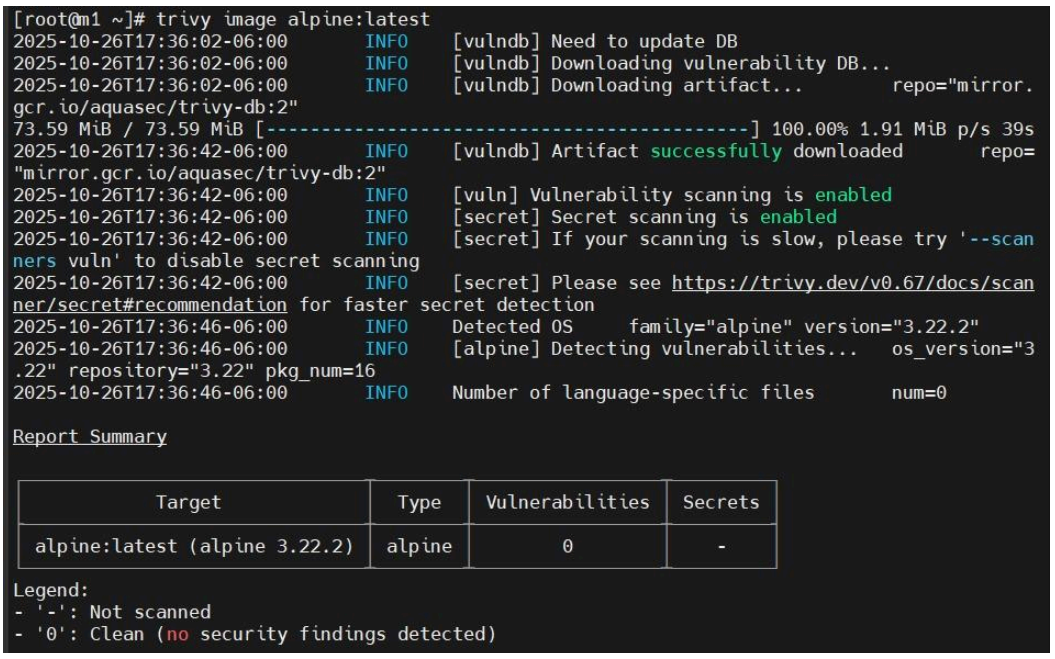
instalar binario (script oficial)
curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sudo sh -s -- -b /usr/local/bin

verificar binario
/usr/local/bin/trivy --version

si no está en PATH (opcional)
echo 'export PATH=$PATH:/usr/local/bin' | sudo tee -a /etc/profile
source /etc/profile
trivy --version
trivy -- download-db-only #Refresca la DB antes del primer uso



2. Prueba básica de testeo de escaneo
Test de escaneo de imagen de contenedor alpine:latest en busca de vulnerabilidades
trivy image alpine:latest



En el ejemplo anterior se visualiza que trivy entra como ejecutable de la herramienta para analizar imágenes de contenedores, sistemas de archivos, repositorios de código, dependencias, etc.

Image: es el subcomando que le indica a trivy que debe analizar una imagen de contenedor (Docker o Podman)

alpine:latest: Es el nombre de la imagen que se analiza, en este caso, se trata de la imagen base Alpine Linux, versión etiquetada como latest.

3. Instalación template HTML
sudo mkdir -p /root/reports
sudo mkdir -p /usr/local/share/trivy
sudo curl -fsSL -o /usr/local/share/trivy/html.tpl \
  https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/html.tpl

```
[root@m1 ~]# sudo mkdir -p /usr/local/share/trivy
sudo curl -fsSL -o /usr/local/share/trivy/html.tpl \
  https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/html.tpl
```

4.Generar reportes en formatos profesionales de 3 imagenes distintas
JSON + HTML
NGINX
trivy image nginx:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-nginx.html

trivy image nginx:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-nginx.json

```
[root@m1 ~]# sudo mkdir -p /usr/local/share/trivy
sudo curl -fsSL -o /usr/local/share/trivy/html.tpl \
  https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/html.tpl
[root@m1 ~]# # NGINX
trivy image nginx:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-nginx.ht
ml

trivy image nginx:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-nginx.json
2025-10-26T17:50:25-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:50:43-06:00       INFO    Detected OS     family="debian" version="13.1"
2025-10-26T17:50:43-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
3" pkg_num=150
2025-10-26T17:50:43-06:00       INFO    Number of language-specific files       num=0
2025-10-26T17:50:43-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
2025-10-26T17:50:43-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:50:44-06:00       INFO    Detected OS     family="debian" version="13.1"
2025-10-26T17:50:44-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
3" pkg_num=150
2025-10-26T17:50:45-06:00       INFO    Number of language-specific files       num=0
2025-10-26T17:50:45-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
```

ALPINE
trivy image alpine:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-alpine.html

trivy image alpine:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-alpine.json

```
[root@m1 ~]# # ALPINE
trivy image alpine:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-alpine.h
tml

trivy image alpine:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-alpine.json
2025-10-26T17:51:02-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:51:05-06:00       INFO    Detected OS     family="alpine" version="3.22.2"
2025-10-26T17:51:05-06:00       INFO    [alpine] Detecting vulnerabilities...   os_version="3
.22" repository="3.22" pkg_num=16
2025-10-26T17:51:05-06:00       INFO    Number of language-specific files       num=0
2025-10-26T17:51:05-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:51:07-06:00       INFO    Detected OS     family="alpine" version="3.22.2"
2025-10-26T17:51:07-06:00       INFO    [alpine] Detecting vulnerabilities...   os_version="3
.22" repository="3.22" pkg_num=16
2025-10-26T17:51:07-06:00       INFO    Number of language-specific files       num=0
```

REDIS
trivy image redis:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-redis.html

trivy image redis:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-redis.json

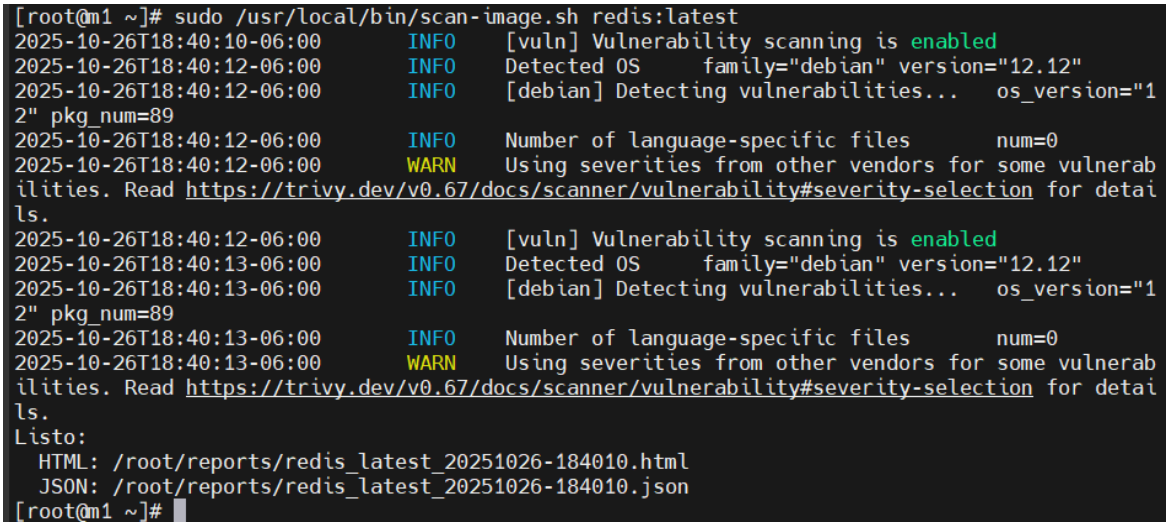
```
[root@m1 ~]# # REDIS
trivy image redis:latest \
  --scanners vuln --timeout 10m \
  --format template --template /usr/local/share/trivy/html.tpl -o /root/trivy-report-redis.ht
ml

trivy image redis:latest \
  --scanners vuln --timeout 10m \
  --format json -o /root/trivy-report-redis.json
2025-10-26T17:51:21-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:51:36-06:00       INFO    Detected OS     family="debian" version="12.12"
2025-10-26T17:51:36-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
2" pkg_num=89
2025-10-26T17:51:36-06:00       INFO    Number of language-specific files       num=0
2025-10-26T17:51:36-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
2025-10-26T17:51:37-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T17:51:39-06:00       INFO    Detected OS     family="debian" version="12.12"
2025-10-26T17:51:39-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
2" pkg_num=89
2025-10-26T17:51:39-06:00       INFO    Number of language-specific files       num=0
2025-10-26T17:51:39-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
[root@m1 ~]# ^C
[root@m1 ~]#
```

5. Script para escanear imágenes manualmente
```
sudo tee /usr/local/bin/scan-image.sh >/dev/null <<'EOF'
#!/usr/bin/env bash
set -euo pipefail

# Uso: scan-image.sh <imagen[:tag]> [directorio_salida] [opciones_trivy...]
# Ej:  scan-image.sh nginx:latest /root/reports --skip-db-update

img="${1:-}"
outdir="${2:-/root/reports}"
shift || true
shift || true

# Resto de argumentos opcionales para trivy (p.ej. --skip-db-update)
extra_args=("$@")

if [[ -z "${img}" ]]; then
  echo "Uso: $(basename "$0") <imagen[:tag]> [directorio_salida] [opciones_trivy...]"
  exit 1
fi

tpl="/usr/local/share/trivy/html.tpl"
if [[ ! -f "$tpl" ]]; then
  echo "Template HTML no encontrado en $tpl. Descargando..."
  sudo mkdir -p /usr/local/share/trivy
  sudo curl -fsSL -o "$tpl" \
    https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/html.tpl
fi

mkdir -p "$outdir"

# Normaliza nombre de archivo
ts="$(date +%Y%m%d-%H%M%S)"
base="$(echo "$img" | tr '/:' '__')"

html="$outdir/${base}_${ts}.html"
json="$outdir/${base}_${ts}.json"

# Escaneo a HTML (solo vulnerabilidades, con timeout razonable)
trivy image "$img" \
  --scanners vuln --timeout 10m \
  --format template --template "$tpl" -o "$html" \
  "${extra_args[@]}"

# Escaneo a JSON (mismo scope)
trivy image "$img" \
```

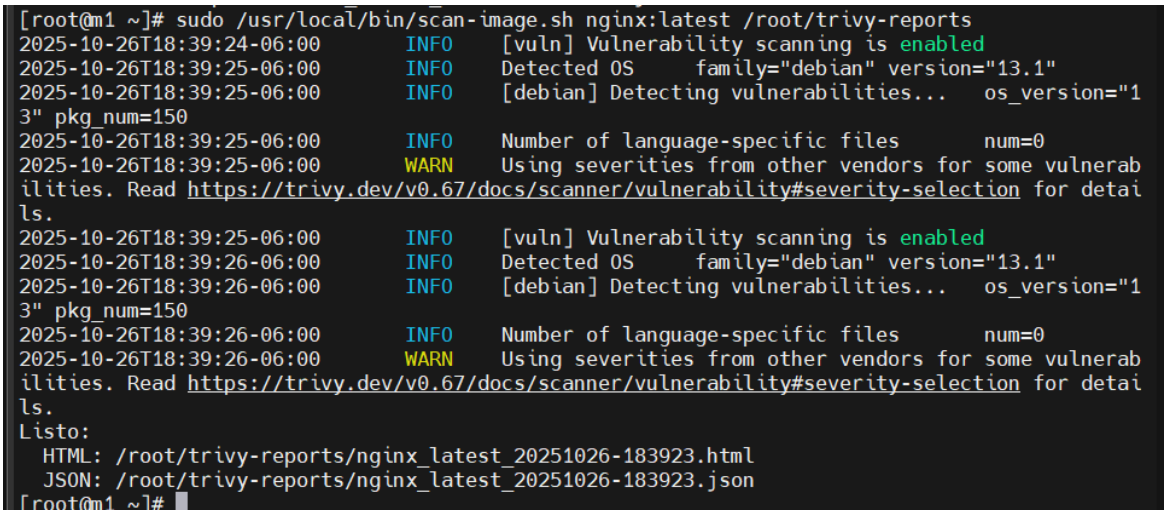
```
  --scanners vuln --timeout 10m \
  --format json -o "$json" \
  "${extra_args[@]}"

echo "Listo:"
echo "  HTML: $html"
echo "  JSON: $json"
EOF

sudo chmod +x /usr/local/bin/scan-image.sh
```

Ejemplos de ejecución:

# Ejemplo 1: escaneo simple (crea /root/reports si no existe)
sudo /usr/local/bin/scan-image.sh redis:latest



# Ejemplo 2: mismo escaneo, pero guardando en otro directorio
sudo /usr/local/bin/scan-image.sh nginx:latest /root/trivy-reports



Ejemplo 3: usando la DB ya descargada (más rápido en repeticiones)

sudo /usr/local/bin/scan-image.sh alpine:latest /root/reports --skip-db-update

```
[root@m1 ~]# sudo /usr/local/bin/scan-image.sh redis:latest  /root/reports --skip-db-update
2025-10-26T18:38:52-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T18:38:54-06:00       INFO    Detected OS     family="debian" version="12.12"
2025-10-26T18:38:54-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
2" pkg_num=89
2025-10-26T18:38:54-06:00       INFO    Number of language-specific files       num=0
2025-10-26T18:38:54-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
2025-10-26T18:38:54-06:00       INFO    [vuln] Vulnerability scanning is enabled
2025-10-26T18:38:56-06:00       INFO    Detected OS     family="debian" version="12.12"
2025-10-26T18:38:56-06:00       INFO    [debian] Detecting vulnerabilities...   os_version="1
2" pkg_num=89
2025-10-26T18:38:56-06:00       INFO    Number of language-specific files       num=0
2025-10-26T18:38:56-06:00       WARN    Using severities from other vendors for some vulnerab
ilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for detai
ls.
Listo:
  HTML: /root/reports/redis_latest_20251026-183852.html
  JSON: /root/reports/redis_latest_20251026-183852.json
[root@m1 ~]#
```

VERIFICACIÓN DE SALIDAS
ls -lh /root/reports/*.html /root/reports/*.json 2>/dev/null || true
ls -lh /root/trivy-reports/*.html /root/trivy-reports/*.json 2>/dev/null || true

```
[root@m1 ~]# ls -lh /root/reports/*.html /root/reports/*.json 2>/dev/null || true
ls -lh /root/trivy-reports/*.html /root/trivy-reports/*.json 2>/dev/null || true
-rw-r--r--. 1 root root   31 Oct 26 18:38 /root/reports/redis_latest_20251026-183852.html
-rw-r--r--. 1 root root 566K Oct 26 18:38 /root/reports/redis_latest_20251026-183852.json
-rw-r--r--. 1 root root   31 Oct 26 18:40 /root/reports/redis_latest_20251026-184010.html
-rw-r--r--. 1 root root 566K Oct 26 18:40 /root/reports/redis_latest_20251026-184010.json
-rw-r--r--. 1 root root   31 Oct 26 18:39 /root/trivy-reports/nginx_latest_20251026-183923.ht
ml
-rw-r--r--. 1 root root 706K Oct 26 18:39 /root/trivy-reports/nginx_latest_20251026-183923.js
on
[root@m1 ~]#
```

6.Interpretación con prueba

Al hacer escaneo de imagen con Trivy, hace un escaneo de vulnerabilidades, muestra los resultados en forma de tabla. Cada línea representa un paquete afectado, su CVE, su severidad y una descripción breve. Al final, Trivy clasifica cada vulnerabilidad según su nivel de severidad (severity level).

TABLA DE NIVELES DE SEVERIDAD

Nivel	Color en el reporte	Descripción técnica	Recomendación práctica
🟥 **CRITICAL**	Rojo	Vulnerabilidades explotables de forma inmediata. Permiten ejecución remota de código, escalamiento de privilegios o fuga crítica de datos.	🚨 Debe corregirse de inmediato. Actualiza la imagen o aplica parches antes de desplegar.

🟧 **HIGH**	Naranja	Fallos graves que pueden comprometer la seguridad, aunque requieran condiciones específicas o usuario local.	⚠ Prioridad alta: Mitigar o reemplazar la imagen tan pronto como sea posible.
🟨 **MEDIUM**	Amarillo	Problemas moderados: pueden revelar información o afectar estabilidad si se cumplen ciertas condiciones.	⚙️ Planifica parcheo o revisión en el siguiente ciclo de mantenimiento.
🟦 **LOW**	Azul / gris	Riesgo bajo o difícil de explotar. Generalmente no afecta a la seguridad directamente.	🕐 Puede posponerse, pero conviene monitorizar.
⚫ **UNKNOWN**	Gris claro	Trivy detectó un CVE, pero no hay datos suficientes o el proveedor no asignó severidad.	🔍 Revisa manualmente el CVE o consulta la base NVD/CVSS.

Trivy usa las bases de datos CVE y CVSS (Common Vulnerability Scoring System).
Cada CVE tiene una puntuación del 0 al 10, y Trivy lo mapea así:

CVSS Base Score	**Severidad Trivy**
9.0–10.0	CRITICAL
7.0–8.9	HIGH
4.0–6.9	MEDIUM
0.1–3.9	LOW
N/A	UNKNOWN

En este ejemplo, se muestra un comando donde lista las vulnerabilidades más graves de redis:latest de forma general.
trivy image redis:latest --scanners vuln --timeout 5m --severity HIGH,CRITICAL

Report Summary

Target	Type	Vulnerabilities
redis:latest (debian 12.12)	debian	5

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

redis:latest (debian 12.12)

Total: 5 (HIGH: 4, CRITICAL: 1)

Trivy analizó la imagen redis:latest y detectó que está basada en Debian 12.12.

Se encontraron 76 vulnerabilidades conocidas en los paquetes del sistema base.

Este análisis incluye librerías del sistema (APT, Bash, libc, PAM, zlib, etc.), no el binario principal de Redis.

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libpam-modules	CVE-2025-6020	HIGH	affected	1.5.2-6+deb12u1		linux-pam: Linux-pam directory Traversal https://avd.aquasec.com/nvd/cve-2025-6020
libpam-modules-bin						
libpam-runtime						
libpam0g						
zlib1g	CVE-2023-45853	CRITICAL	will_not_fix	1:1.2.13.dfsg-1		zlib: integer overflow and resultant heap-based buffer overflow in zipOpenNewFileInZip4_6 https://avd.aquasec.com/nvd/cve-2023-45853

[root@m1 ~]#

-Library: el paquete afectado en el sistema base.
-CVE: identificador público de la vulnerabilidad (permite consultar detalles en NVD o AquaSec).
-Status:
affected: el paquete está vulnerable y no actualizado.
will_not_fix: Debian no planea actualizarlo (semitolerado o sin parche disponible).
-Fixed Version: indica la versión corregida (si existe).
-Title: resumen del tipo de ataque o impacto.

Como buenas prácticas para mitigar están:

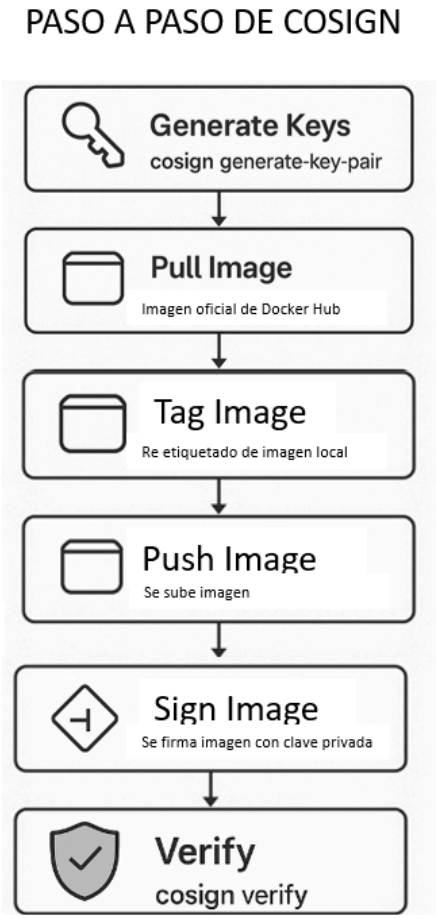
- Actualizar la imagen base
- Reeconstruir y reescanear
- Implementar políticas de firma: imágenes firmadas y verificadas.
- Automatizar el escaneo en pipeline CI/CD para prevenir despliegues con vulnerabilidades no corregidas.

COSIGN

¿Qué es Cosign?
Cosign es una herramienta de seguridad desarrollada por Sigstore, diseñada para firmar, verificar y almacenar firmas criptográficas de imágenes de contenedores (y otros artefactos OCI) de forma segura y automatizada. Su objetivo es garantizar la integridad, autenticidad y procedencia de las imágenes que se despliegan en entornos de producción, evitando la ejecución de software modificado o no autorizado.

Cosign ayuda a prevenir esto mediante un sistema de firma digital basada en claves criptográficas (RSA o Ed25519) y una verificación transparente antes de desplegar la imagen. En proyectos de seguridad como Connaisseur o Kyverno, Cosign es una pieza clave, ya que permite a estos validadores comprobar que las imágenes están firmadas y provengan de fuentes confiables, contribuyendo así al cumplimiento de buenas prácticas de supply chain security (SLSA, NIST 800-218, etc.).

A continuación en la siguiente imagen se tiene el paso a paso de como funciona Cosign y posteriormente la guía de instalación.



Guía de instalación de cosign
# Instalar desde repositorio
sudo dnf install -y cosign

Verificar versión
cosign version

#2do método de instalación binario más reciente de Github.
sudo curl -sSL -o /usr/local/bin/cosign \
https://github.com/sigstore/cosign/releases/latest/download/cosign-linux-amd64
sudo chmod +x /usr/local/bin/cosign

sudo install -m 0755 /usr/local/bin/cosind /usr/local/bin/cosign
cosign version #verificar version
Cosign instalado

```
[root@m1 connaisseur]# cosign version
  ______   ______        _______. __    _______ .__   __.
 /      | /      |      /       ||  |  /  _____||  \ |  |
|  ,----'|  ,----'     |   (----`|  | |  |  __  |   \|  |
|  |     |  |           \   \    |  | |  | |_ | |  . `  |
|  `----.|  `----.  .----)   |   |  | |  |__| | |  |\   |
 \______| \______/  |_______/    |__|  \______| |__| \__|
cosign: A tool for Container Signing, Verification and Storage in an OCI registry.

GitVersion:    v2.2.4
GitCommit:     fb651b4ddd8176bd81756fca2d988dd8611f514d
GitTreeState:  clean
BuildDate:     2024-04-10T21:57:27Z
GoVersion:     go1.21.8
Compiler:      gc
Platform:      linux/amd64
```

Importante utilizar versión 2.2 o similares, para evitar problemas de compatibilidad no utilice una versión mayor a esta debido a incompatibilidad con Connaiseur posteriormente.

Cosign configurado para firma manual
A) Subir un **registro local** en nodo padre para alojar imágenes y firmas
Así evitamos credenciales en Docker Hub y nos servirá para Connaisseur.

Levanta registry:2 en el puerto 5000
#tener instalado podman o docker
sudo dnf install -y podman
sudo mkdir -p /var/lib/registry
sudo podman run -d --name registry --restart=always -p 5000:5000 \
  -v /var/lib/registry:/var/lib/registry registry:2

# Abre firewall en máquina nodo padre
sudo firewall-cmd --add-port=5000/tcp --permanent
sudo firewall-cmd --reload

Prueba rápida:
curl http://127.0.0.1:5000/v2/

Debe responder {}

salida esperada:

```
Installed:
  aardvark-dns-2:1.14.0-1.el9.x86_64
  conmon-3:2.1.12-1.el9.x86_64
  containers-common-2:1-117.el9_6.x86_64
  criu-3.19-1.el9.x86_64
  criu-libs-3.19-1.el9.x86_64
  crun-1.23.1-2.el9_6.x86_64
  fuse-common-3.10.2-9.el9.x86_64
  fuse-overlayfs-1.14-1.el9.x86_64
  fuse3-3.10.2-9.el9.x86_64
  fuse3-libs-3.10.2-9.el9.x86_64
  libnet-1.2-7.el9.x86_64
  libslirp-4.4.0-8.el9.x86_64
  netavark-2:1.14.1-1.el9_6.x86_64
  passt-0^20250217.ga1e48a0-10.el9_6.x86_64
  passt-selinux-0^20250217.ga1e48a0-10.el9_6.noarch
  podman-5:5.4.0-13.el9_6.x86_64
  protobuf-c-1.3.3-13.el9.x86_64
  shadow-utils-subid-2:4.9-12.el9.x86_64
  slirp4netns-1.3.2-1.el9.x86_64
  yajl-2.1.0-25.el9.x86_64

Complete!
Resolved "registry" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/registry:2...
Getting image source signatures
Copying blob 6d464ea18732 done   |
Copying blob 3493bf46cdec done   |
Copying blob bbbdd6c6894b done   |
Copying blob 44cf07d57ee4 done   |
Copying blob 8e82f80af0de done   |
Copying config 26b2eb0361 done   |
Writing manifest to image destination
5eae8da8621b7c8e2d8d183ef296f3e67c8d2ba31df21b2b860af360f3a1b68e
success
success
{}[m1@m1 ~]curl http://127.0.0.1:5000/v2/
{}[m1@m1 ~]$
```

B) Configurar containerd para “plain HTTP”

sudo mkdir -p /etc/containerd/certs.d/192.168.56.114:5000
sudo tee /etc/containerd/certs.d/192.168.56.114:5000/hosts.toml >/dev/null <<'EOF'
server = "http://192.168.56.114:5000"

[host."http://192.168.56.114:5000"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
EOF
sudo systemctl restart containerd

*Se ejecuta ese bloque en cluster o de manera local para que todos puedan hacer pull/push del registro local.

C) Generar el par de claves (privada/pública) de Cosign

mkdir -p /root/cosign
cd /root/cosign
cosign generate-key-pair

```
[root@m1 ~]# mkdir -p /root/cosign && cd /root/cosign
# Con passphrase vacía para laboratorio:
COSIGN_PASSWORD="" cosign generate-key-pair
ls -l /root/cosign/   # verás cosign.key y cosign.pub
Private key written to cosign.key
Public key written to cosign.pub
total 8
-rw-------. 1 root root 653 Oct 26 19:35 cosign.key
-rw-r--r--. 1 root root 178 Oct 26 19:35 cosign.pub
[root@m1 cosign]#
```

D) Preparar 3 imágenes de prueba en el registro local
Ya con el el registro local levantado pueden subirse imágenes aquí hay 3 ejemplos:
# NGINX
sudo podman pull docker.io/library/nginx:1.25
sudo podman tag docker.io/library/nginx:1.25 192.168.56.114:5000/demo/nginx:1.25
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/nginx:1.25

```
[root@m1 cosign]# # NGINX
sudo podman pull docker.io/library/nginx:1.25
sudo podman tag docker.io/library/nginx:1.25 192.168.56.114:5000/demo/nginx:1.25
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/nginx:1.25
Trying to pull docker.io/library/nginx:1.25...
Getting image source signatures
Copying blob 933cc8470577 done   |
Copying blob a11fc495bafd done   |
Copying blob 45337c09cd57 done   |
Copying blob 999643392fb7 done   |
Copying blob 971bb7f4fb12 done   |
Copying blob 09f376ebb190 done   |
Copying blob de3b062c0af7 done   |
Copying config e784f45604 done   |
Writing manifest to image destination
e784f4560448b14a66f55c26e1b4dad2c2877cc73d001b7cd0b18e24a700a070
Getting image source signatures
Copying blob 56f8fe6aedcd done   |
Copying blob fc1cf9ca5139 done   |
Copying blob 747b290aeba8 done   |
Copying blob 9f4d73e635f1 done   |
Copying blob 7d2fd59c368c done   |
Copying blob 5d4427064ecc done   |
Copying blob 14773070094d done   |
Copying config e784f45604 done   |
Writing manifest to image destination
```

# ALPINE
sudo podman pull docker.io/library/alpine:3.18
sudo podman tag docker.io/library/alpine:3.18 192.168.56.114:5000/demo/alpine:3.18
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/alpine:3.18

```
[root@m1 cosign]# # ALPINE
sudo podman pull docker.io/library/alpine:3.18
sudo podman tag docker.io/library/alpine:3.18 192.168.56.114:5000/demo/alpine:3.18
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/alpine:3.18
Trying to pull docker.io/library/alpine:3.18...
Getting image source signatures
Copying blob 44cf07d57ee4 skipped: already exists
Copying config 802c91d529 done   |
Writing manifest to image destination
802c91d5298192c0f3a08101aeb5f9ade2992e22c9e27fa8b88eab82602550d0
Getting image source signatures
Copying blob f44f286046d9 done   |
Copying config 802c91d529 done   |
Writing manifest to image destination
```

# REDIS
sudo podman pull docker.io/library/redis:7.2
sudo podman tag docker.io/library/redis:7.2 192.168.56.114:5000/demo/redis:7.2
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/redis:7.2

```
[root@m1 cosign]# # REDIS
sudo podman pull docker.io/library/redis:7.2
sudo podman tag docker.io/library/redis:7.2 192.168.56.114:5000/demo/redis:7.2
sudo podman push --tls-verify=false 192.168.56.114:5000/demo/redis:7.2
Trying to pull docker.io/library/redis:7.2...
Getting image source signatures
Copying blob a0ca0ee367c9 done   |
Copying blob 46b1dd0eeff1 done   |
Copying blob 68904d1222d2 done   |
Copying blob e52757c29af7 done   |
Copying blob abe1fea37542 done   |
Copying blob 32cc31543243 done   |
Copying blob 4f4fb700ef54 done   |
Copying blob 90645010d9fe done   |
Copying config 3bd16f9c28 done   |
Writing manifest to image destination
3bd16f9c2821e8836fe63ea422c8482096acbea0b0cbf35e6ef6a811a2a2baaa
Getting image source signatures
Copying blob d5f125df2046 done   |
Copying blob 3eef8a31472f done   |
Copying blob 9e36c753b08f done   |
Copying blob 4724408ab3be done   |
Copying blob a70d9b3548dd done   |
Copying blob d6d0586ce802 done   |
Copying blob 5f70bf18a086 done   |
Copying blob b7065059b716 done   |
Copying config 3bd16f9c28 done   |
Writing manifest to image destination
[root@m1 cosign]#
```

Verificar que el registry las “vea”
curl http://192.168.56.114:5000/v2/_catalog
curl http://192.168.56.114:5000/v2/demo/nginx/tags/list

```
[root@m1 cosign]# curl http://192.168.56.114:5000/v2/_catalog
curl http://192.168.56.114:5000/v2/demo/nginx/tags/list
{"repositories":["demo/alpine","demo/nginx","demo/redis"]}
{"name":"demo/nginx","tags":["1.25"]}
[root@m1 cosign]#
```

E) Firmar las 3 imágenes con Cosign (clave local)
Esto genera una firma OCI que se guarda como un artefacto asociado en el mismo registro.

```
# NGINX
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/nginx:1.25

# ALPINE
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/alpine:3.18

# REDIS
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/redis:7.2
```

```
[root@m1 cosign]# # NGINX
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/nginx:1.25
WARNING: Image reference 192.168.56.114:5000/demo/nginx:1.25 uses a tag, not a digest, to ide
ntify the image to sign.
    This can lead you to sign a different image than the intended one. Please use a
    digest (example.com/ubuntu@sha256:abc123...) rather than tag
    (example.com/ubuntu:latest) for the input to cosign. The ability to refer to
    images by tag will be removed in a future release.

[root@m1 cosign]# # ALPINE
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/alpine:3.18
WARNING: Image reference 192.168.56.114:5000/demo/alpine:3.18 uses a tag, not a digest, to id
entify the image to sign.
    This can lead you to sign a different image than the intended one. Please use a
    digest (example.com/ubuntu@sha256:abc123...) rather than tag
    (example.com/ubuntu:latest) for the input to cosign. The ability to refer to
    images by tag will be removed in a future release.

[root@m1 cosign]# # REDIS
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key \
  192.168.56.114:5000/demo/redis:7.2
WARNING: Image reference 192.168.56.114:5000/demo/redis:7.2 uses a tag, not a digest, to iden
tify the image to sign.
    This can lead you to sign a different image than the intended one. Please use a
    digest (example.com/ubuntu@sha256:abc123...) rather than tag
    (example.com/ubuntu:latest) for the input to cosign. The ability to refer to
    images by tag will be removed in a future release.
```

F) Verificar las firmas con la clave pública
La verificación se hace con la clave pública generada previamente:

```
cosign verify --key /root/cosign/cosign.pub \
  192.168.56.114:5000/demo/nginx:1.25
```

```
[root@m1 cosign]# cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/nginx:
1.25

Verification for 192.168.56.114:5000/demo/nginx:1.25 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"192.168.56.114:5000/demo/nginx:1.25"},"image":{
"docker-manifest-digest":"sha256:380f6eda6aca3bbc82ca52b88d7e3a46f6aa6bb5b0559a21a2c4379e6611
5833"},"type":"https://sigstore.dev/cosign/sign/v1"},"optional":null}]
[root@m1 cosign]#
[root@m1 cosign]#
```

cosign verify --key /root/cosign/cosign.pub \
  192.168.56.114:5000/demo/alpine:3.18

```
[root@m1 cosign]# cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/alpine
:3.18

Verification for 192.168.56.114:5000/demo/alpine:3.18 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"192.168.56.114:5000/demo/alpine:3.18"},"image":
{"docker-manifest-digest":"sha256:6c66d7635b9721dbdde41e62dcedf52a732c4588d235a2b85f896a2d804
f3bf2"},"type":"https://sigstore.dev/cosign/sign/v1"},"optional":null}]
[root@m1 cosign]#
```

cosign verify --key /root/cosign/cosign.pub \
  192.168.56.114:5000/demo/redis:7.2

```
[root@m1 cosign]# cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/redis:
7.2

Verification for 192.168.56.114:5000/demo/redis:7.2 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"192.168.56.114:5000/demo/redis:7.2"},"image":{"
docker-manifest-digest":"sha256:f8417a2945c9cea5ccc658aebfdcf3f799f6f886b0069d34cab2aadfb8baa
95b"},"type":"https://sigstore.dev/cosign/sign/v1"},"optional":null}]
[root@m1 cosign]#
[root@m1 cosign]#
```

Si la imagen no está firmada, Cosign mostrará un error indicando que no se encontró firma válida.

EJEMPLO FIRMAR IMAGEN MANUALMENTE

Ejemplo con redis
echo "=== Antes de firmar ==="
cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/redis:7.2 || echo "❌ No firmada todavía"

```
[root@m1 cosign]# echo "=== Antes de firmar ==="
cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/redis:7.2 || echo "✗ No
 firmada todavía"
=== Antes de firmar ===

Verification for 192.168.56.114:5000/demo/redis:7.2 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"192.168.56.114:5000/demo/redis:7.2"},"image":{"
docker-manifest-digest":"sha256:f8417a2945c9cea5ccc658aebfdcf3f799f6f886b0069d34cab2aadfb8baa
95b"},"type":"https://sigstore.dev/cosign/sign/v1"},"optional":null}]
[root@m1 cosign]#
```

echo "=== Firmando imagen ==="
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key 192.168.56.114:5000/demo/redis:7.2

```
[root@m1 cosign]# echo "=== Firmando imagen ==="
COSIGN_PASSWORD="" cosign sign --key /root/cosign/cosign.key 192.168.56.114:5000/demo/redis:7
.2
=== Firmando imagen ===
WARNING: Image reference 192.168.56.114:5000/demo/redis:7.2 uses a tag, not a digest, to iden
tify the image to sign.
    This can lead you to sign a different image than the intended one. Please use a
    digest (example.com/ubuntu@sha256:abc123...) rather than tag
    (example.com/ubuntu:latest) for the input to cosign. The ability to refer to
    images by tag will be removed in a future release.

[root@m1 cosign]#
```

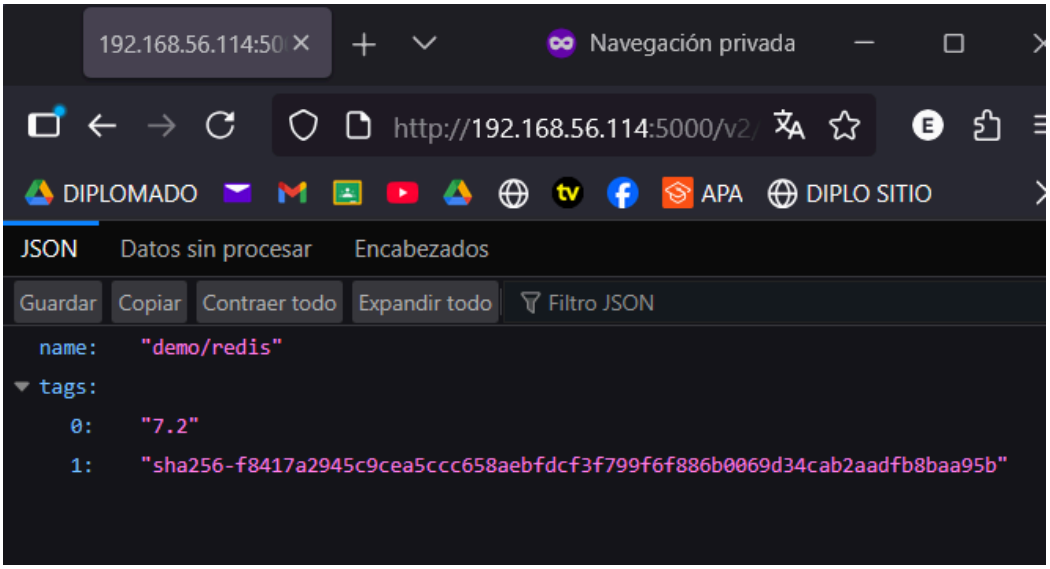
echo "=== Verificando después de firmar ==="
cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/redis:7.2

```
[root@m1 cosign]# echo "=== Verificando después de firmar ==="
cosign verify --key /root/cosign/cosign.pub 192.168.56.114:5000/demo/redis:7.2
=== Verificando después de firmar ===

Verification for 192.168.56.114:5000/demo/redis:7.2 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"192.168.56.114:5000/demo/redis:7.2"},"image":{"
docker-manifest-digest":"sha256:f8417a2945c9cea5ccc658aebfdcf3f799f6f886b0069d34cab2aadfb8baa
95b"},"type":"https://sigstore.dev/cosign/sign/v1"},"optional":null},{"critical":{"identity":
{"docker-reference":"192.168.56.114:5000/demo/redis:7.2"},"image":{"docker-manifest-digest":"
sha256:f8417a2945c9cea5ccc658aebfdcf3f799f6f886b0069d34cab2aadfb8baa95b"},"type":"https://sig
store.dev/cosign/sign/v1"},"optional":null}]
[root@m1 cosign]#
```

Ver detalles de la firma

Puedes mostrar que la firma se almacenó como una imagen OCI separada en el mismo registry:

curl -s http://192.168.56.114:5000/v2/demo/redis/tags/list



CONNAISEUR

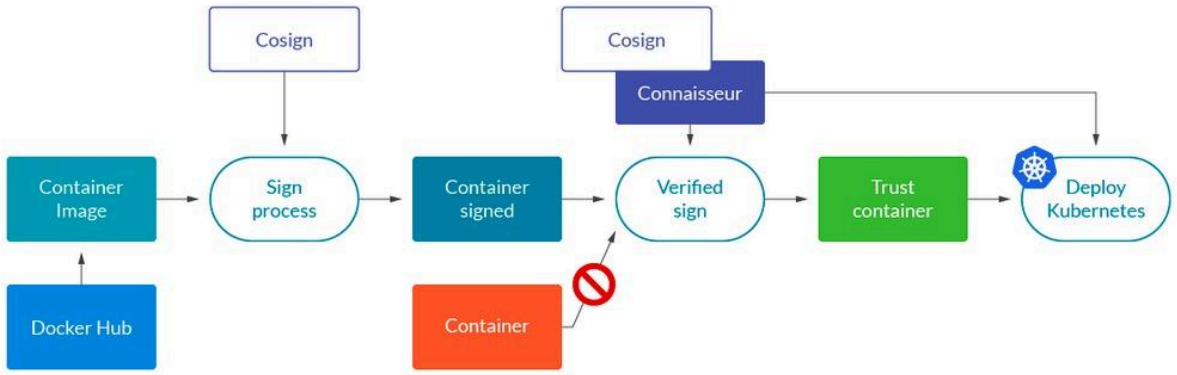
Connaisseur es un admission controller para Kubernetes que intercepta solicitudes de despliegue y verifica firmas de imágenes antes de permitir su ejecución.
Trabaja junto con Cosign para validar la firma digital de cada imagen y bloquear aquellas no firmadas o alteradas.

Objetivo principal es evitar que se ejecuten imágenes no verificadas o manipuladas, protegiendo el clúster de:

- Imágenes comprometidas o alteradas.
- Errores humanos (por ejemplo, usar una imagen equivocada).
- Ataques de “supply chain” (cadena de suministro).

¿Cómo funciona?
1.Se instala en el clúster de Kubernetes como un admission controller mutating/validating webhook.
2.Cuando se envía un Deployment, Pod, etc., Connaisseur intercepta la solicitud.
3.Extrae la referencia de la imagen (image: registry.io/repo:tag).
4.Consulta el registro de firmas (por ejemplo, Notary, Cosign o Docker Content Trust).
5.Verifica la firma contra una clave pública configurada previamente.
6.Si la firma es válida → deja pasar el despliegue.
Si no es válida o no existe → bloquea el despliegue y devuelve un error.

Diagrama esperado para Cosign y Connaisseur.



Guía de instalación Connaiseur

1) Instala Connaisseur (vía Helm)

#Creacion de namespace connaisseur
kubectl create ns connaisseur

```
[root@m1 ~]# kubectl create ns connaiseur
namespace/connaiseur created
[root@m1 ~]# helm repo add connaisseur https://sse-secure-systems.github.io/conn
aisseur/charts
```

#Instalación de Helm desde repositorio oficial
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

```
[root@m1 ~]# curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-h
elm-3 | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11928  100 11928    0     0  45876      0 --:--:-- --:--:-- --:--:-- 45876
Downloading https://get.helm.sh/helm-v3.19.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

#Verificar helm
helm version

```
[root@m1 ~]# helm version
version.BuildInfo{Version:"v3.19.0", GitCommit:"3d8990f0836691f0229297773f352459
8f46bda6", GitTreeState:"clean", GoVersion:"go1.24.7"}
[root@m1 ~]#
```

#agregar repositorio de helm para connaisseur
helm repo add connaisseur https://sse-secure-systems.github.io/connaisseur/chart

#actualizar la repo de helm
helm repo update

```
[root@m1 ~]# helm repo add connaisseur https://sse-secure-systems.github.io/conn
aisseur/charts
"connaisseur" has been added to your repositories
[root@m1 ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "connaisseur" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@m1 ~]#
```

Crea un values.yaml minimalista para:

- Validar con Cosign usando tu clave pública previamente creada.
- Exigir firma a todas las imágenes de tu registro 192.168.56.114:5000 (puedes ampliar el patrón).

- Permitir registry HTTP (inseguro) si tu registro corre en http://…:5000.

- Pega el contenido de tu /root/cosign/cosign.pub (bloque completo -----BEGIN PUBLIC KEY----- …) donde se indica

#Sugerencia tener creado el directorio de connaiseur donde este guardado values.yaml
mkdir -p /root/connaisseur
cd /root/connaiseur

#ARCHIVO VALUES.YAML
cat <<'EOF' > /root/connaisseur/values.yaml
values.yaml mínimo para Connaisseur + Cosign local
insecureRegistries:
  - "192.168.56.114:5000"

trustRoots:
  - name: cosign-pubkey
    key: |
      -----BEGIN PUBLIC KEY-----
      AGREGAR TU CLAVE PUBLICA DE /root/cosign
      -----END PUBLIC KEY-----

validators:
  - name: cosign-validator
    type: cosign
    trustRoots:
      - cosign-pubkey

policy:
  - pattern: "192.168.56.114:5000/*:*"
    validator: cosign-validator
EOF

El archivo values.yaml es importante ya que será el archivo de configuración de políticas y claves de confianza usadas por el webhook.
insecureRegistries → Permite trabajar con registro HTTP sin TLS.
trustRoots → Claves públicas confiables (en tu caso cosign-pubkey).
validators → Define el tipo de verificación (Cosign).
policy → Determina qué rutas o patrones de imágenes deben validarse.

#Instalar helm dentro del directorio
helm install connaisseur connaisseur/connaisseur -n connaisseur -f values.yaml

```
[root@m1 connaisseur]# helm install connaisseur connaisseur/connaisseur -n conna
isseur -f values.yaml
NAME: connaisseur
LAST DEPLOYED: Thu Oct 30 10:17:28 2025
NAMESPACE: connaisseur
STATUS: deployed
REVISION: 1
TEST SUITE: None
[root@m1 connaisseur]#
```

#verificar que los pods de connaisseur esten corriendo
kubectl -n connaisseur get pods

```
[root@m1 connaisseur]# kubectl -n connaisseur get pods
NAME                                READY   STATUS    RESTARTS   AGE
connaisseur-57b7cdbb4d-hq76d        1/1     Running   0          3m11s
connaisseur-57b7cdbb4d-t9mzn        1/1     Running   0          3m11s
connaisseur-57b7cdbb4d-vdrpl        1/1     Running   0          3m11s
connaisseur-redis-5c756cd5fc-h5frh  1/1     Running   0          3m11s
[root@m1 connaisseur]#
```

MODIFICACIONES PARA VISUALIZACIÓN DE WEBHOOK
Esto es importante ya que el Webhook es un mecanismo de control del API Server de Kubernetes. Cuando se intenta crear, actualizar o eliminar un recurso, el API Server llama al webhook para pedirle permiso antes de continuar.

Connaisseur instala un MutatingAdmissionWebhook llamado: connaisseur-webhook
Usuario hace kubectl apply.
El API Server envía la solicitud al webhook de Connaisseur.
Connaisseur consulta el registro y verifica la firma.
Si la firma es válida → responde “allow”, si no → “deny”.

1)Etiquetar el namespace para exlcluirlo del webhook
kubectl label namespace connaisseur \
  securesystemsengineering.connaisseur/webhook=ignore --overwrite

2)verificar el webhook, service y endpoints
Webhook (mutating)
kubectl get mutatingwebhookconfigurations | grep -i connaisseur

```
[root@m1 connaisseur]# kubectl get mutatingwebhookconfigurations | grep -i connaisseur
connaisseur-webhook   1          5m25s
```

kubectl describe mutatingwebhookconfiguration connaisseur-webhook

```
[root@m1 connaisseur]# kubectl describe mutatingwebhookconfiguration connaisseur-webhook
Name:         connaisseur-webhook
Namespace:
Labels:       app.kubernetes.io/instance=connaisseur
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/name=connaisseur
              app.kubernetes.io/version=3.8.5
              helm.sh/chart=connaisseur-2.8.5
Annotations:  helm.sh/hook: post-install, post-upgrade, post-rollback
API Version:  admissionregistration.k8s.io/v1
Kind:         MutatingWebhookConfiguration
Metadata:
  Creation Timestamp:  2025-10-30T18:20:45Z
  Generation:          1
  Resource Version:    31911
  UID:                 6a1cbabc-9742-4551-8a62-d839776a1f42
Webhooks:
  Admission Review Versions:
    v1
  Client Config:
    Ca Bundle:  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURyRENDQXBTZ0F3SUJBZ0lRQ2NOb2wx
K3dNT1k1NDdMaGpUU0lhVEFOQmdrcWhraUc5dzBCQVFzRkFEQXEKTVNnd0pnWURWUVFERXg5amIyNXVZV2x6YzJW
MWNpMXpkbU11WTI5dWJtRnBjM05sZFhJdWMzWmpNQ0FYRFRJMQpNVEF6TURFNE1UUXpObG9ZRHpJeE1qVXhNREEy
TVRneE5ETTJXakFxTVNnd0pnWURWUVFERXg5amIyNXVZV2x6CmMyVjFjaTF6ZG1NdVkyOXVibUZwYzNObGRYSXVj
M1pqTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEEKTUlJQkNnS0NBUUVBcDV4SEphbUpRVmw4aUoveUds
MUJQVDAySTF1RlpJMWl0ckpDcnN1WXM1cjFOV3lkenpWbQpNQ2U3aWhNNVh3OFByQU42VFNRWW9qekVKdW9yeC9W
M1BmSFp6bFhYdEtuYWNhKzA5WVpkMURleUtndXFzdGRkCjc1cTNVZTBTemhRcjdCbU9TRmdudUJQMDlOdll6c1k0
amZPdEx0STQ0UTE5Z3MwaW1IeGpWNFJMclRxZDI1WmsKMk1tTHhrY2lFNU1GR1ZUdXR1aHl0NTRJS0V3U1I3b2Iw
UFVBZ2I5TjlpTnRVUFNQbXJsUHFQSHhZaE5VeGg4Vgo5NjdheW5qODBhYUNKL3QzSWpuRjl5d3lRSWphUE5zRGlH
aEg3NTJlMXIyb3BkMFNjUWFHeFBFdFVWYjFCOHR1CjJXeU04T2VsR0NQT2d6OGJ0dndmY2Rwc3FVeGRUY2c0cXdJ
REFRQUJvNEhMTUlISU1BNEdBMVVkRHdFQi93UUUKQXdJRm9EQWRCZ05WSFNVRUZqQVVCZ2dyQmdFRkJRY0RBUVlJ
S3dZQkJRVUhBd0l3REFZRFZSMFRBUUgvQkFJdwpBRENCaUFZRFZSMFJCSUdBTUg2Q0QyTnZibTVoYVhOelpYVnlM
WE4yWTRJYlkyOXVibUZwYzNObGRYSXRjM1pqCkxtTnZibTVoYVhOelpYVnlnaDlqYjI1dVlXbHpjMlYxY2kxemRt
TXVZMjl1Ym1GcGMzTmxkWEl1YzNaamdpMWoKYjI1dVlXbHpjMlYxY2kxemRtTXVZMjl1Ym1GcGMzTmxkWEl1YzNa
akxtTnNkWE4wWlhJdWJHOWpZV3d3RFFZSgpLb1pJaHZjTkFRRUxCUUFEZ2dFQkFDSUNKcEptdUVLSk9GZlZLZzJz
K3Qxei9BOWFrV0gvdURBK2JZQU4wVTBZCk1qVXI0TkJFK3g4ZTBINHFneXJCa0dnTmpZSmhuVExpOFhwb1FIV2tq
cGVWQWl2S1p0VUFIY1pnaG1uaVhDdGUKV2tUcnNVWXF2RFBIaUJLZDBoY1lpdFlhZVF5QkZkUzl4blg1Mk9TZmVx
ZWFvNW05d3phbVhua1YxSFB4S1czVApsZTYwTGk0VDV4RlZjNy9HcmtBQkN1SllvSHY4eUJWNS91NWVzSi9GUG1G
N0N6cS9ramZRRzloeFdrQ1l0MksyCmtYZ2V6cnlkNUzVUtvVWFlMXhxUklMNDlpMW14WEk3T1FIV1VrSEhxRGk0
R2tYaUNBdnZTYno2ZDlKREhPU0cKSkl3T1F6UklWM0hMSG0vWjdvbmFKWEw1dDJNSGYyZGd5eitBU0YrZnV0MD0K
LS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    Service:
      Name:        connaisseur-svc
      Namespace:   connaisseur
```

```
      Namespace:   connaisseur
      Path:        /mutate
      Port:        443
  Failure Policy:  Fail
  Match Policy:    Equivalent
  Name:            connaisseur-svc.connaisseur.svc
  Namespace Selector:
    Match Expressions:
      Key:       securesystemsengineering.connaisseur/webhook
      Operator:  NotIn
      Values:
        ignore
  Object Selector:
  Reinvocation Policy:  Never
  Rules:
    API Groups:
      apps
    API Versions:
      v1
    Operations:
      CREATE
      UPDATE
    Resources:
      deployments
      replicasets
      daemonsets
      statefulsets
    Scope:  *
    API Groups:

    API Versions:
      v1
    Operations:
      CREATE
      UPDATE
    Resources:
      pods
      pods/ephemeralcontainers
      replicationcontrollers
    Scope:  *
    API Groups:
      batch
    API Versions:
      v1
    Operations:
      CREATE
      UPDATE
    Resources:
      pods
      pods/ephemeralcontainers
      replicationcontrollers
    Scope:  *
    API Groups:
      batch
    API Versions:
      v1
    Operations:
      CREATE
      UPDATE
    Resources:
      jobs
      cronjobs
    Scope:          *
  Side Effects:     None
  Timeout Seconds:  30
Events:             <none>
```

Webhook apunta a service.name: connaisseur-svc, port: 443, path: /mutate.

kubectl -n connaisseur get svc -o wide

```
[root@m1 connaisseur]# kubectl -n connaisseur get svc -o wide
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   S
ELECTOR
connaisseur-redis-service  ClusterIP   10.98.153.72    <none>        6379/TCP   14m   a
pp.kubernetes.io/instance=redis,app.kubernetes.io/name=connaisseur
connaisseur-svc            ClusterIP   10.101.92.166   <none>        443/TCP    14m   a
pp.kubernetes.io/instance=connaisseur,app.kubernetes.io/name=connaisseur
[root@m1 connaisseur]#
```

kubectl -n connaisseur get endpoints -o wide

```
[root@m1 connaisseur]# kubectl -n connaisseur get endpoints -o wide
NAME                        ENDPOINTS                                                  AGE
connaisseur-redis-service   10.244.1.10:6379                                           15m
connaisseur-svc             10.244.1.11:5000,10.244.2.10:5000,10.244.2.11:5000         15m
```

kubectl -n connaisseur get deploy connaisseur -o yaml | egrep -n 'containerPort|name: https|image:'

```
[root@m1 connaisseur]# kubectl -n connaisseur get deploy connaisseur -o yaml | egrep -n
'containerPort|name: https|image:'
64:        image: docker.io/securesystemsengineering/connaisseur:v3.8.5
77:        - containerPort: 5000
78:          name: https
[root@m1 connaisseur]#
```

connaisseur-svc tiene endpoints (IPs) y mapea a containerPort: 5000.
Si el Service no tuviera endpoints (selector no coincide), parchea el selector del Service para que apunte a los labels del Deployment:

kubectl -n connaisseur patch svc connaisseur-svc --type='json' \

-p='[{"op":"replace","path":"/spec/selector","value":{"app.kubernetes.io/instance":"connaisseur","app.kubernetes.io/name":"connaisseur"}}]'

3) (Opcional) Reintentar upgrade ahora que el ns está excluido
helm upgrade --install connaisseur connaisseur/connaisseur \
  -n connaisseur -f /root/connaisseur/values.yaml \
  --set webhook.failurePolicy=Fail

*Si antes salía el timeout failed calling webhook ... /mutate ..., debería desaparecer tras etiquetar el namespace.

4) Comprobación final del webhook
El webhook se dispara al crear/actualizar pods/deployments en otros ns
kubectl -n connaisseur logs deploy/connaisseur --tail=50

```
[root@m1 connaisseur]# kubectl -n connaisseur get deploy connaisseur -o yaml | egrep -n
'containerPort|name: https|image:'
64:        image: docker.io/securesystemsengineering/connaisseur:v3.8.5
77:        - containerPort: 5000
78:          name: https
[root@m1 connaisseur]# kubectl -n connaisseur logs deploy/connaisseur --tail=50
Found 3 pods, using pod/connaisseur-57b7cdbb4d-pgd8b
{
  "level": "info",
  "msg": "Starting server at 127.0.0.1:5000...",
  "time": "2025-10-30T18:14:40Z"
```

Estructura sugerida para archivos, carpetas y entornos de prueba.

Carpeta	Contenido	Descripción
`/root/connaisseur`	`values.yaml`	Configuración del Helm Chart.
`/root/cosign/`	`cosign.key`, `cosign.pub`	Claves de firma Cosign.
`/root/connaisseur/tests/`	`fail-unsigned.yaml`, `ok-signed.yaml`	Manifiestos de prueba.

5) Ejercicios de intentar deployar los pods al firmar o no estar firmados
Intentar desplegar imagen sin firma → debe ser rechazada.

```
[root@m1 bin]# cat > /root/connaisseur/fail-unsigned.yaml <<'EOF'
apiVersion: v1
kind: Pod
metadata:
  name: fail-unsigned
  namespace: default
spec:
  containers:
  - name: c
    image: 192.168.56.114:5000/demo/nginx:1.26
    imagePullPolicy: IfNotPresent
EOF
[root@m1 bin]# kubectl apply -f /root/connaisseur/fail-unsigned.yaml
Error from server: error when creating "/root/connaisseur/fail-unsigned.yaml": a
dmission webhook "connaisseur-svc.connaisseur.svc" denied the request: error dur
ing cosign validation of image 192.168.56.114:5000/demo/nginx:1.26: error valida
ting image: [no matching signatures: signature layer sha256:87d09fa7ec9c612ea14a
7a0388db9a29ea6bc5b617cfc56c53a7ea9951b00ecb is missing "dev.cosignproject.cosig
n/signature" annotation]
```

Se tiene de un archivo fail-unsigned.yaml
cat >/root/connaisseur/fail-unsigned.yaml <<'EOF'
apiVersion: v1
kind: Pod
metadata:
  name: fail-unsigned
  namespace: default
spec:
  containers:
  - name: c
    image: 192.168.56.114:5000/demo/nginx:1.26
    imagePullPolicy: IfNotPresent
EOF

#DEBE FALLAR
kubectl apply -f /root/connaisseur/fail-unsigned.yaml

Webhook de Connaisseur deniega el deploy con error "no signatures found" o "no matching signatures".

Kubernetes intenta crear un Pod llamado fail-unsigned. El webhook de Connaisseur intercepta la solicitud antes de que el Pod se ejecute. Connaisseur revisa la política que se tiene configurada (solo aceptar imágenes firmadas).

Como nginx:1.26 no tiene una firma Cosign asociada, Connaisseur rechaza la creación del Pod.

Resultado visible en el log:
error validating image: [no matching signatures ... missing "dev.cosignproject.cosign/signature" annotation]

El Pod fue bloqueado porque la imagen no tiene firma digital ni metadatos de autenticidad. Connaisseur impidió ejecutar código potencialmente alterado o no verificado.

Desplegar imagen firmada con Cosign → debe ser aceptada.

```
[root@m1 bin]# cat >/root/connaisseur/ok-signed.yaml <<'EOF'
apiVersion: v1
kind: Pod
metadata:
  name: ok-signed
  namespace: default
spec:
  nodeName: m3
  containers:
  - name: c
    image: 192.168.56.114:5000/demo/nginx-ok@sha256:380f6eda6aca3bbc82ca52b88d7e
3a46f6aa6bb5b0559a21a2c4379e66115833
    imagePullPolicy: IfNotPresent
EOF
[root@m1 bin]# kubectl delete pod ok-signed --ignore-not-found
pod "ok-signed" deleted
[root@m1 bin]# kubectl apply -f /root/connaisseur/ok-signed.yaml
pod/ok-signed created
[root@m1 bin]# kubectl get pod ok-signed -w
NAME        READY   STATUS    RESTARTS   AGE
ok-signed   1/1     Running   0          6s
^C[root@m1 bin]# ^C
[root@m1 bin]# kubectl get pods -A
NAMESPACE           NAME                                 READY   STATUS    RESTAR
TS      AGE
connaisseur         connaisseur-d765d97b9-2rpxn          2/2     Running   0
        9m12s
connaisseur         connaisseur-d765d97b9-jnnqt          2/2     Running   0
        9m14s
connaisseur         connaisseur-d765d97b9-krbps          2/2     Running   0
        9m9s
connaisseur         connaisseur-redis-5fd7cdf5c-8hf7t    1/1     Running   2 (65m
 ago)   42h
default             ok-signed                            1/1     Running   0
        44s
```

Se tiene el archivo ok-signed.yaml

```
cat >/root/connaisseur/ok-signed.yaml <<'EOF'
apiVersion: v1
kind: Pod
metadata:
  name: ok-signed
  namespace: default
spec:
  nodeName: m3
  containers:
  - name: c
                                                                    image:
192.168.56.114:5000/demo/nginx-ok@sha256:380f6eda6aca3bbc82ca52b88d7e3a46f6aa6
bb5b0559a21a2c4379e66115833
    imagePullPolicy: IfNotPresent
EOF

kubectl apply -f /root/connaisseur/ok-signed.yaml
kubectl get pod ok-signed -w
```

Se solicita crear el Pod ok-signed. Connaisseur intercepta nuevamente la solicitud.

Verifica que la imagen nginx-ok@sha256:... posee una firma Cosign válida (la misma clave pública que fuw configurada en Connaisseur). La validación es exitosa, y Connaisseur autoriza la creación del Pod. Kubernetes descarga la imagen y el contenedor arranca correctamente.
A diferencia del pod unsigned, aquí incluye un digest SHA256 firmado con Cosign. Esto identifica el contenido exacto de la imagen y permite que Connaisseur lo valide contra la clave pública configurada.

Resultado visible:
ok-signed 0/1 ContainerCreating → 1/1 Running

El Pod fue autorizado y desplegado porque la imagen está firmada digitalmente y el digest coincide con una firma válida registrada en el repositorio.

Ambos Pods se definen igual, pero la diferencia está en la autenticidad de la imagen.
Connaisseur actúa como un guardia de seguridad del clúster:

Rechaza cualquier imagen sin firma digital válida (fail-unsigned).
Acepta e implementa las imágenes firmadas y verificadas (ok-signed).

De esta forma, se garantiza que todo lo que se ejecuta en Kubernetes proviene de una fuente confiable y verificable.

TROUBLESHOOTING COMÚN

Este apartado tiene como objetivo ayudar a identificar y resolver los problemas más comunes que pueden presentarse durante la instalación, configuración o ejecución de los tres componentes principales del entorno: Trivy, Cosign y Connaisseur.

1. Problemas comunes con Trivy

1.1. Error: “FATAL database initialization error”
Síntoma: Trivy no logra actualizar o descargar la base de datos de vulnerabilidades.

Causa probable:

- Falta de conexión a internet o proxy bloqueando la descarga.
- Cache local corrupto en ~/.cache/trivy.

Solución:

1. Borra la cache local:
   rm -rf ~/.cache/trivy

2. Fuerza una actualización manual de la base:
   trivy --download-db-only

3. Si hay un proxy corporativo, configura las variables:
   export HTTP_PROXY=http://proxy:8080
   export HTTPS_PROXY=http://proxy:8080

Esto se puede prevenir en actualizar la base de datos regularmente con un cron diario o en cada ejecución del pipeline CI/CD.

1.2. Reportes vacíos o sin vulnerabilidades
Síntoma: El reporte HTML se genera, pero no muestra resultados.

Causa probable:

- Imagen base no contiene binarios analizables.
- Escaneo incompleto o con opción --skip-db-update.

Solución:
Asegura usar un tag válido:
trivy image nginx:latest

- No usar --skip-db-update en el primer análisis.
- Valida que la imagen tenga sistema de archivos (usa alpine, ubuntu, etc.).

Prevención:

Evitar imágenes “distroless” sin empaquetado de paquetes del sistema (no generan datos útiles para Trivy).

2. Problemas comunes con Cosign

2.1. Error: “no matching signatures” o “missing signature”
Síntoma: Cosign no encuentra una firma asociada a la imagen.

Causa probable:
- La imagen fue firmada con otro tag o digest.
- No se subió el .sig al registro.
- La clave pública (cosign.pub) no corresponde a la privada usada.

Solución:

1. Verifica el digest de la imagen:
   podman inspect IMAGE | grep -i digest

2. Firma usando el digest en lugar del tag:
   cosign sign --key cosign.key 192.168.56.114:5000/demo/nginx@sha256:<digest>

3. Confirma que los archivos .sig estén en el registro local:
   curl -X GET http://192.168.56.114:5000/v2/_catalog

Esto puede prevenirse si se mantiene consistencia de tags o usa siempre la firma por digest (@sha256:).
Guarda copia de las claves generadas con cosign generate-key-pair.

2.2. Error: “permission denied” o “no space left on device”
Síntoma: Cosign no puede escribir o subir la firma al registro.

Causa probable:
- Permisos incorrectos en /etc/containerd o /var/lib/registry.
- Disco del registro lleno.

Solución:
1. Verifica permisos:
   sudo chmod -R 755 /var/lib/registry

2. Revisa espacio disponible:
   df -h

3. Elimina imágenes antiguas
   podman rmi $(podman images -q)

3. Problemas comunes con Connaisseur

3.1. Error: “failed calling webhook ... x509: certificate signed by unknown authority”
Síntoma: Kubernetes no puede validar el webhook de Connaisseur.

Causa probable:

- El caBundle en el MutatingWebhookConfiguration no coincide con el certificado TLS usado por el servicio.
- El Secret connaisseur-tls fue regenerado sin actualizar el webhook.

Solución:

1. Verificar el contenido de la CA:

```
kubectl -n connaisseur get secret connaisseur-tls -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -subject
```

2. Reinyectar el nuevo caBundle:

```
CAB=$(kubectl -n connaisseur get secret connaisseur-tls -o jsonpath='{.data.ca\.crt}')
kubectl patch mutatingwebhookconfiguration connaisseur-webhook \
                                                                --type='json'
-p='[{"op":"replace","path":"/webhooks/0/clientConfig/caBundle","value":"'"$CAB"'"}]'
```

3. Esperar a que el webhook reinicie:
   kubectl rollout status deploy/connaisseur -n connaisseur

Prevención:
Mantener respaldos de la CA (ca.crt) y actualizar el webhook cada vez que regeneres certificados.
3.2. Error: “no matching signatures” o “invalid trust root”
Síntoma: El webhook rechaza imágenes firmadas correctamente.

Causa probable:

- La configuración trustRoot en el values.yaml no contiene la clave pública correcta.
- El digest o tag de imagen no coincide con la firma registrada.

Solución:

1. Verifica el valor de trustRoot:

```
kubectl -n connaisseur get configmap connaisseur-app-config -o yaml | grep -A3 trustRoot
```

2. Sustituye el contenido por la clave pública cosign.pub válida.
3. Reinstala o actualiza Connaisseur:

```
helm upgrade connaisseur connaisseur/connaisseur -n connaisseur -f /root/connaisseur/values.yaml
```

Prevención:
Usar nombres consistentes en las imágenes firmadas y mantener una única clave pública por entorno.

3.3. Error: “connection refused” o “context deadline exceeded”
Síntoma: Los pods del clúster no pueden comunicarse con el webhook.

Causa probable:
- El servicio connaisseur-svc no tiene endpoints válidos.
- Puerto incorrecto o el sidecar socat no redirige correctamente al backend.

Solución:
1. Verifica endpoints activos:
   kubectl -n connaisseur get endpoints connaisseur-svc -o wide
2. Asegúrate de que apunten a puertos 5000 o 5001.
3. Si hay inconsistencias, reinicia el despliegue:
   kubectl -n connaisseur rollout restart deploy connaisseur

Prevención:
Evita modificar manualmente los puertos del servicio y haz snapshots del entorno antes de aplicar cambios.

3.4. Error: “CrashLoopBackOff” en pods Connaisseur
Síntoma: El pod entra en un ciclo de reinicios constantes.

Causa probable:
- El contenedor principal no encuentra el binario uvicorn.
- Comandos command o args fueron sobreescritos incorrectamente.
- Certificados faltantes o Secret eliminado.

Solución:
1. Elimina configuraciones manuales:

```
kubectl -n connaisseur patch deploy connaisseur --type=json -p='[
  {"op":"remove","path":"/spec/template/spec/containers/0/command"},
  {"op":"remove","path":"/spec/template/spec/containers/0/args"}
]'
```

2. Reinstala desde Helm:

```
helm upgrade --install connaisseur connaisseur/connaisseur -n connaisseur -f /root/connaisseur/values.yaml
```

Prevención:
Nunca modifiques directamente los command o args del pod principal. Si es necesario usar uvicorn, hazlo en una capa de pruebas separada.

3.5. Error: “unable to verify the first certificate” (en test TLS)

Síntoma:
La verificación de TLS falla al usar openssl s_client.

Causa probable:

- El pod de prueba no tiene acceso a la CA (/tls/ca.crt).
- El Secret no fue montado correctamente.
- El archivo ca.crt está vacío o corrupto.

Solución:

1. Monta el Secret dentro del pod de prueba:

```
kubectl -n connaisseur run tlsprobe --image=alpine --restart=Never --overrides='{
  "apiVersion": "v1",
  "spec": {
    "containers": [{
      "name": "test",
      "image": "alpine",
      "command": ["sh","-lc"],
      "args": [
        "apk add --no-cache openssl >/dev/null 2>&1; \
          echo | openssl s_client -connect connaisseur-svc.connaisseur.svc:443 \
          -CAfile /tls/ca.crt 2>/dev/null | egrep -i \"subject=|issuer=|Verify return code\""
      ],
      "volumeMounts": [{"name":"tls","mountPath":"/tls","readOnly":true}]
    }],
    "volumes": [{"name":"tls","secret":{"secretName":"connaisseur-tls"}}]
  }
}'
```

2. Verifica que el resultado contenga:
   Verify return code: 0 (ok)

Prevención:
Mantén sincronizados los certificados CA (ca.crt) entre el Secret, el webhook y el servicio.

Como comentario para englobar lo de los troubleshootings encontrados, la mayoría de los problemas en este entorno provienen de inconsistencias entre el certificado TLS, el caBundle y el registro de imágenes firmado. Una vez que esos tres elementos están correctamente alineados, Connaisseur funciona de manera estable y confiable, garantizando que solo se desplieguen imágenes firmadas y verificadas.

Buenas prácticas para la instalación en general

- Realizar snapshots antes de cambios en TLS/webhook
- Mantener un backup de las claves Cosign y CA TLS.
- Usar digest SHA256 en todas las políticas de Connaisseur
- Verificar logs en pods al hacer cambios rigurosos o fozosos en Cosign o Connaisseur
- Desplegar Connaisseur en un namespace dedicado
- Tener un buen directorio de archivos de pruebas en cada uno de los casos Trivy, Cosign y Connaisseur.
- Utilizar la version 2.2.x de Cosign, ya que otras se presentan fallas al hacer las pruebas de Deploy con firma y sin firma

Referencias:

- Sigstore. (s. f.). Sigstore Quickstart with Cosign. https://docs.sigstore.dev/quickstart/quickstart-cosign/

- Aqua Security. (s. f.). Trivy Documentation. Recuperado de https://aquasecurity.github.io/trivy/v0.56

- Sigstore. (s. f.). Cosign Documentation. Recuperado de https://docs.sigstore.dev/cosign/

- SSE Secure Systems. (s. f.). Connaisseur — Verify Container Image Signatures in Kubernetes Clusters. Recuperado de https://sse-secure-systems.github.io/connaisseur/v2.5.3/

- The Hacker Way. (2022, 5 mayo). «DevSecOps y detección de vulnerabilidades con Trivy». The Hacker Way. Recuperado de https://thehackerway.es/2022/05/05/devsecops-y-deteccion-de-vulnerabilidades-con-trivy/

- Keyfactor, Equipo técnico. (2023, 26 enero). Protección de contenedores con SignServer y Cosign. Keyfactor Blog. Recuperado de https://www.keyfactor.com/es/blog/securing-containers-with-signserver-and-cosign/

- Trivy / Cosign Team. (s.f.). Cosign Vulnerability Scan Record. In Trivy Documentation. Retrieved [date], from https://trivy.dev/v0.67/docs/supply-chain/attestation/vuln/

- ASCIT Group. (2023, 26 septiembre). «Trivy: Escáner de seguridad versátil para contenedores y más». ASCIT Group. Recuperado de https://www.ascitgroup.com/2023/09/26/trivy-escaner-de-seguridad-versatil-para-contenedores-y-mas/