Profesor: Ing. Daniel Guerrero Ramírez
Diplomado: Infraestructura en T.I.

Trivy + Cosign

Image Security

Integrantes:
Colin Mosqueda Eduardo
Patiño Oseguera Alexis

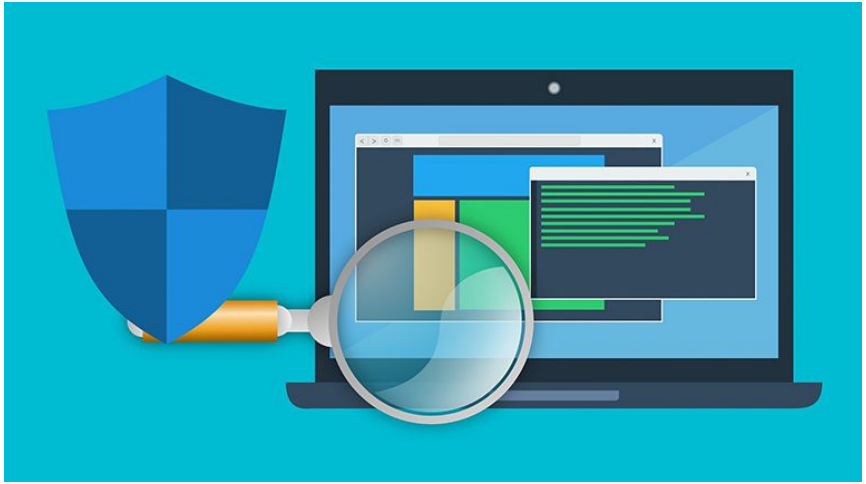
TRIVY

¿Qué es Trivy?

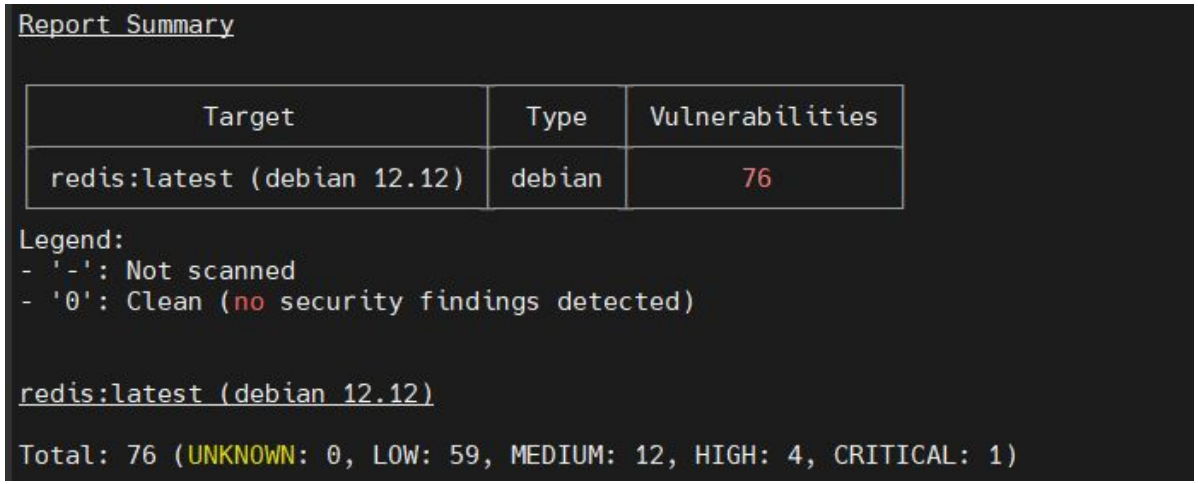
Herramienta de escaneo de vulnerabilidades desarrollada por Aqua Security.
Analiza imágenes de contenedores, archivos del sistema, repositorios de código y dependencias en busca de paquetes inseguros, configuraciones erróneas y secretos expuestos.
Es ampliamente usada en entornos desarrollo, seguridad y operaciones (DevSecOps) por su facilidad de integración con CI/CD (Integración/Entrega Continua), Kubernetes y registries locales.

Principales características:

- Escaneo de vulnerabilidades en sistemas, dependencias y contenedores.
- Compatible con Docker, Podman y Kubernetes.
- Genera reportes en formatos JSON y HTML.
- Soporte para múltiples sistemas operativos y distribuciones.



Demo en vivo:
Escanear imagen con vulnerabilidades



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.

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

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

-Library: el paquete afectado en el sistema base.

-Vulnerabilidad: CVE (Common Vulnerabilities and Exposures)
identificador único asignado a cada vulnerabilidad pública conocida.

-Status: affected: la versión instalada puede tener problemas
fixed: ya se tiene una versión corregida
will_not_fix: no tendra parche
unknown: trivy no puede determinar el estado

-Installed version / Fixed Version: indica la versión instalada y corregida

-Title: resumen del tipo de vulnerabilidad y componente afectado

SEVERIDAD

Nivel	Color en el reporte	Descripción técnica	Recomendación práctica
🟥 **CRITICAL**	Rojo	Falla muy grave que puede usarse para atacar el sistema (ejecución de código, robo de datos, etc.)	⚠️ Corregir de inmediato. Actualiza o parchea antes de usar la imagen.
🟧 **HIGH**	Naranja	Falla seria que puede comprometer la seguridad.	🔧 Reemplaza o mitiga el problema lo antes posible.
🟨 **MEDIUM**	Amarillo	Problema moderado; requiere condiciones especiales para explotar.	🕒 Planifica su corrección en la próxima actualización.
🟦 **LOW**	Azul / gris / verde	Riesgo bajo o difícil de aprovechar.	👀 Se puede dejar para después, pero hay que monitorear.
⚫ **UNKNOWN**	Gris claro	Se detectó una falla, pero no hay suficiente información.	🔍 Revisa manualmente el CVE o consulta la base de datos NVD/CVSS.

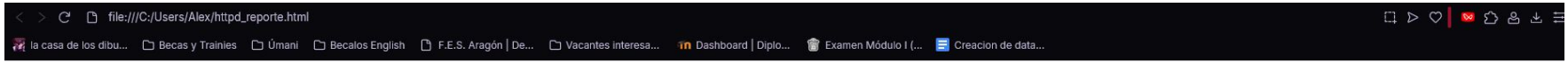
En Redis:latest se encontró:
→ Vulnerabilidad de traversal de directorio en el paquete linux-pam.
Permite acceder a archivos fuera del directorio permitido.
Se recomienda actualizar cuando se tenga el parche

→ Falla de desbordamiento de entero y de buffer en heap (riesgo de ejecución de código o crash).
Se recomienda usar versión segura o sustituta de librería

Otra forma de visualizar los reportes es hacerlo mediante el archivo HTML y un equipo con navegador web.

Gracias a los colores, se puede hacer un análisis de las vulnerabilidades de una manera más intuitiva.

```
[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="12" 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 vulnerabilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for details.
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="12" 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 vulnerabilities. Read https://trivy.dev/v0.67/docs/scanner/vulnerability#severity-selection for details.
Listo:
  HTML: /root/reports/redis_latest_20251026-183852.html
  JSON: /root/reports/redis_latest_20251026-183852.json
[root@m1 ~]#
```



httpd:2.4.41 (debian 10.3) - Trivy Report - 2025-11-03 21:39:38.592887451 -0600 CST m=+140.913862605

debian

Package	Vulnerability ID	Severity	Installed Version	Fixed Version	Links
apt	CVE-2020-27350	MEDIUM	1.8.2	1.8.2.2	https://bugs.launchpad.net/bugs/1899193 https://security.netapp.com/advisory/ntap-20210108-0005/ https://ubuntu.com/security/notices/USN-4667-1 Toggle more links
apt	CVE-2020-3810	MEDIUM	1.8.2	1.8.2.1	https://bugs.launchpad.net/bugs/1878177 https://github.com/Debian/apt/issues/111 https://github.com/julian-klode/apt/commit/de4efadc3c92e26d37272fd310be148ec61dcf36 Toggle more links
bsdutils	CVE-2021-37600	MEDIUM	1:2.33.1-0.1	2.33.1-0.1+deb10u1	https://access.redhat.com/security/cve/CVE-2021-37600 https://github.com/karelzak/util-linux/commit/1c9143d0c1f979c3daf10e1c37b5b1e916c22a1c https://github.com/karelzak/util-linux/issues/1395 Toggle more links
bsdutils	CVE-2024-28085	MEDIUM	1:2.33.1-0.1	2.33.1-0.1+deb10u1	http://www.openwall.com/lists/oss-security/2024/03/27/5 http://www.openwall.com/lists/oss-security/2024/03/27/6 http://www.openwall.com/lists/oss-security/2024/03/27/7 Toggle more links
coreutils	CVE-2016-2781	LOW	8.30-3		http://seclists.org/oss-sec/2016/q1/452 http://www.openwall.com/lists/oss-security/2016/02/28/2 http://www.openwall.com/lists/oss-security/2016/02/28/3 Toggle more links
debian-archive-keyring	DLA-3482-1	UNKNOWN	2019.1	2019.1+deb10u2	
dpkg	CVE-2022-1664	CRITICAL	1.19.7	1.19.8	https://git.dpkg.org/cgit/dpkg/dpkg.git/commit/?id=1f23dddc17f69c9598477098c7fb9936e15fa495 https://git.dpkg.org/cgit/dpkg/dpkg.git/commit/?id=58814cacee39c4ce9e2cd0e3a3b9b57ad437eff5 https://git.dpkg.org/cgit/dpkg/dpkg.git/commit/?id=7a6c03cb34d4a09f35df2f10779cbf1b70a5200b Toggle more links
e2fsprogs	CVE-2022-1304	HIGH	1.44.5-1+deb10u3		https://access.redhat.com/errata/RHSA-2022:8361 https://access.redhat.com/security/cve/CVE-2022-1304 https://bugzilla.redhat.com/2069726 Toggle more links
fdisk	CVE-2021-37600	MEDIUM	2.33.1-0.1	2.33.1-0.1+deb10u1	https://access.redhat.com/security/cve/CVE-2021-37600 https://github.com/karelzak/util-linux/commit/1c9143d0c1f979c3daf10e1c37b5b1e916c22a1c https://github.com/karelzak/util-linux/issues/1395 Toggle more links

Buenas prácticas para mitigar son:

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

COSIGN

¿Qué es la firma de imágenes?

La firma de imágenes (image signing) es un proceso criptográfico mediante el cual una persona o sistema garantiza la autenticidad e integridad de una imagen de contenedor.

¿Por qué es importante firmar imágenes?

1. Verificar la identidad del autor
2. Garantizar la integridad
3. Permitir políticas de seguridad automatizadas

¿Qué es cosign?

Cosign, parte del proyecto Sigstore, permite firmar y verificar imágenes de contenedor utilizando criptografía.

Su objetivo es garantizar la autenticidad, integridad y procedencia de las imágenes que se despliegan en producción.

sigstore

cosign

Principales características:

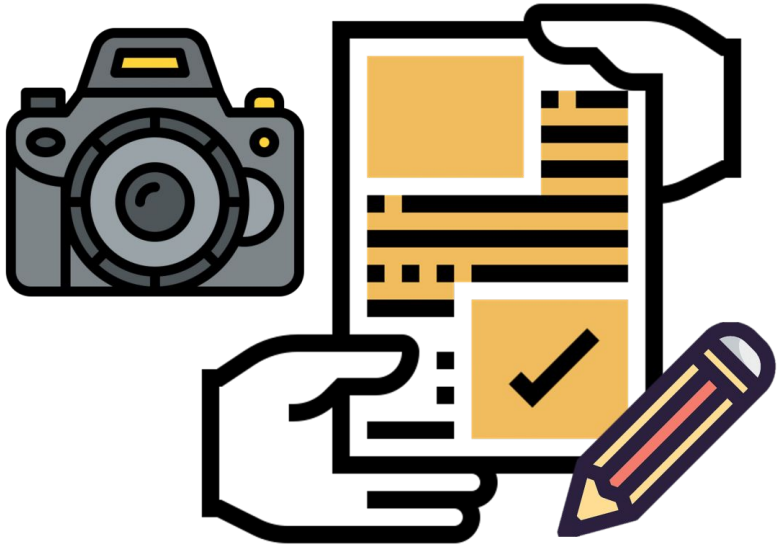
- Firma digital de imágenes con claves públicas y privadas.
- Almacenamiento de firmas en el mismo registry de contenedores.
- Verificación automática de integridad antes de desplegar.
- Compatible con Kubernetes, Docker, Podman y CI/CD pipelines.

Tipos de firmas en Cosign

Cosign soporta varios modos o tipos de firmas, que dependen de cómo y dónde se almacenan en el registro de contenedores

Modo	Descripción breve	Uso típico
Legacy	Firma almacenada como una imagen separada (`sha256-<digest/huella>.sig`) en el registro.	Registries antiguos sin soporte OCI.
OCI Artifacts / Referrers	Firma vinculada usando el estándar OCI(Open Container Initiative).	Entornos modernos con soporte OCI v1.1+.
Bundle / Transparency (Rekor)	Firma se guarda a su vez en un registro público de transparencia (Rekor) o en un `.bundle`.	Auditoría, verificación y trazabilidad.
Keyless	Firma sin clave privada; usa cuenta o identidad OIDC (OpenID Connect)	Pipelines CI/CD y proyectos open source.

Chain of thrust



Una cadena de confianza es una secuencia de certificados o claves donde cada uno verifica la autenticidad del siguiente, hasta llegar a una raíz de confianza (trust root) que todos consideran legítima.

Cosign:

1.Cosign utiliza una cadena de confianza para garantizar que una imagen de contenedor fue firmada por alguien legítimo.

2.La firma se puede verificar automáticamente con certificados o identidades.

3.La verificación no depende solo de una clave manual, sino de una infraestructura confiable.

Demo en vivo:
Firmar una imagen manualmente

Pasos para firmar una imagen manualmente

Paso 1 – Generar un par de claves Cosign (una sola vez)

cosign generate-key-pair

```
[root@m1 connaisseur]# ls
ca.cnf  ca.key  cosign.key
ca.crt  ca.srl  cosign.pub
```

Paso 2 – Pull, tag y push de imagen con registro local

PULL
Se descarga la imagen oficial desde Docker Hub.

TAG
Se re-etiqueta la imagen local con el nombre de destino: 192.168.56.114:5000/demo/redis:7.2.

PUSH
Se sube la imagen a destino.

Resultado: ahora la imagen está disponible en el repositorio local (/demo/redis) lista para firmarse.

```
[root@m1 ~]# export REG=192.168.56.114:5000
export IMG_REDIS=$REG/demo/redis
export COSIGN=/usr/local/bin/cosign
export COSIGN_KEY=/root/connaisseur/cosign.key
export COSIGN_PUB=/root/connaisseur/cosign.pub
[root@m1 ~]# sudo podman pull docker.io/library/redis:7.2
sudo podman tag  docker.io/library/redis:7.2 $IMG_REDIS:7.2
sudo podman push --tls-verify=false $IMG_REDIS:7.2
Trying to pull docker.io/library/redis:7.2...
Getting image source signatures
Copying blob a0ca0ee367c9 skipped: already exists
Copying blob 46b1dd0eeff1 skipped: already exists
Copying blob 32cc31543243 skipped: already exists
Copying blob 68904d1222d2 skipped: already exists
Copying blob e52757c29af7 skipped: already exists
Copying blob 4f4fb700ef54 skipped: already exists
Copying blob abe1fea37542 skipped: already exists
Copying blob 90645010d9fe skipped: already exists
Copying config 3bd16f9c28 done   |
Writing manifest to image destination
3bd16f9c2821e8836fe63ea422c8482096acbea0b0cbf35e6ef6a811a2a2baaa
Getting image source signatures
Copying blob 602f0874279d skipped: already exists
Copying blob a309ee7e5eb4 skipped: already exists
Copying blob 4de204883b8a skipped: already exists
Copying blob bd9ddc54bea9 skipped: already exists
Copying blob a5f9baba06a1 skipped: already exists
Copying blob e20e29cbc8ce skipped: already exists
Copying blob 5cf55faf6dad skipped: already exists
Copying blob 83255d51549e skipped: already exists
Copying config 3bd16f9c28 done   |
Writing manifest to image destination
```

Paso 3 – Firmar la imagen

Comando sign

- define la imagen objetivo (192.168.56.114:5000/demo/redis)
- inspecciona la imagen con skopeo, extrae huella digital
- activa modo compatibilidad: legacy
- --allow-insecure-registry: permite firmar en http sobre registro local
- --key: usará llave privada

Escribimos contraseña

Después de la ejecución:

- Confirmación y políticas de sigstore.
- Genera un registro (tlog entry) y firma se habrá creado

```
[root@m1 ~]# export DIGEST_REDIS=$(skopeo inspect --tls-verify=false docker://$IMG_REDIS:7.2 | jq -r .Digest)
COSIGN_REGISTRY_REFERRERS_MODE=legacy $COSIGN sign --allow-insecure-registry \
  --key "$COSIGN_KEY" --yes \
  $IMG_REDIS@$DIGEST_REDIS
Enter password for private key:

        The sigstore service, hosted by sigstore a Series of LF Projects, LLC, is provided pursuant to the Hosted Proje
cies/hosted-project-tools-terms-of-use/.
        Note that if your submission includes personal data associated with this signed artifact, it will be part of an
        This may include the email address associated with the account with which you authenticate your contractual Agr
        This information will be used for signing this artifact and will be stored in public transparency logs and cann
ice at https://lfprojects.org/policies/hosted-project-tools-immutable-records/.

By typing 'y', you attest that (1) you are not submitting the personal data of any other person; and (2) you understand
 listed above.
tlog entry created with index: 660736649
Pushing signature to: 192.168.56.114:5000/demo/redis
```

Paso 4 – Verificar imagen

Cosign verify:

- Verifica una firma digital existente en el registro
- Usará la clave pública, comparando con clave privada correcta.

Al terminar ejecución

- Cosign verifica la imagen
- Muestra la información firmada
- Validando que la firma es válida y la imagen es auténtica

```
[root@m1 ~]# $COSIGN verify --allow-insecure-registry --key "$COSIGN_PUB" $IMG_REDIS@$DIGEST_REDIS

Verification for 192.168.56.114:5000/demo/redis@sha256:f8417a2945c9cea5ccc658aebfdcf3f799f6f886b0069d34cab2aadfb8baa95b
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"},"image":{"docker-manifest-digest":"sha256
95b"},"type":"cosign container image signature"},"optional":{"Bundle":{"SignedEntryTimestamp":"MEQCIE7ytcOHuglBD2tVCTbrm
0eAzjXw==","Payload":{"body":"eyJhcGlWZXJzaW9uIjoiMC4wLjEiLCJraW5kIjoiaGFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIjp7Imhhc2giOn
ZGY1ODM3NTFlNjkxNTgzMWZiYjI3ZGExOTEyNzNiODU3NjQ1NDY2NjRkNDBlNjBmIn19LCJzaWduYXR1cmUiOnsiY29udGVudCI6Ik1FWUNJUUNMS1RLZU16
ZiRHhvODAxRnVWN1phUjQrTmMvYi9tUWdpWDBwM2NuZm1VdCIsInB1YmxpY0tleSI6eyJjb250ZW50IjoiTFMwdExTMUNSVWRKVGlCUVZVSk1TVU1nUzBWWk
UlkwUlJaMEZGUzJOTU9VRTNjbWxEV0Rka05XWnpkbFZFYTJoR2NVb3JkR3QyUlFwclIwWjJhbkJ3VW1Kak1rRnJTRkptUzBwcVNFcHRabmRoTkVoNU9VOVVV
ZUa1FnVUZWQ1RFbERJRXRGV1MwdExTMHRDZz09In19fX0=","integratedTime":1762028454,"logIndex":660736649,"logID":"c0d23d6ad40697
[root@m1 ~]#
```

CONNAISSEUR

¿Qué es Connaisseur?

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.

Ventajas

- Refuerza la seguridad en el clúster Kubernetes.
- Automatiza la validación de imágenes firmadas.
- Evita despliegues no autorizados.

Desventajas

- Requiere una configuración detallada (certificados, claves, políticas)
- Puede añadir algo de latencia al proceso de admisión de pods.

¿Qué es un Webhook?

Un *webhook de admisión* (Admission 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

El webhook es el "**guardia del clúster**": intercepta las peticiones al API Server, revisa las imágenes con base en la política, y decide si autoriza o bloquea la creación de los Pods según su firma digital.

Núcleo de la configuración de Connaisseur

El núcleo de la configuración es el archivo “values.yaml” Archivo de configuración que define políticas, validadores, y claves de confianza usadas por el webhook. Este esta dividido por varias secciones:

insecureRegistries:Permite validar imágenes alojadas en un registro HTTP (sin TLS).

trustRoots: Define las raíces de confianza, es decir, las claves públicas autorizadas. (Cosign.pub)

validators: Especifica qué tipo de validador se usará (Cosign)

policy: Define las reglas, qué imágenes deben ser verificadas, y con qué validador.

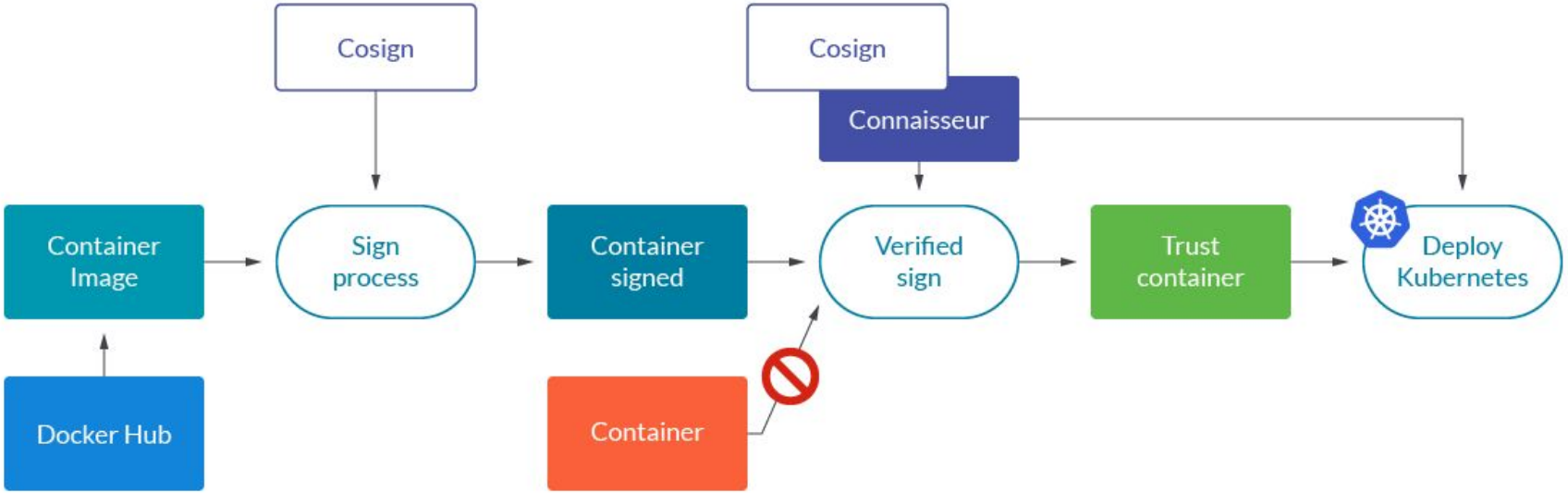
```
[root@m1 connaisseur]# cat values.yaml
# values.yaml mínimo para Connaisseur + Cosign local
insecureRegistries:
  - "192.168.56.114:5000"

trustRoots:
  - name: cosign-pubkey
    key: |
      -----BEGIN PUBLIC KEY-----
      MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAED4t3z23dJp0HLhr+ZPD4iQS5um4G
      OK7kS8E7vHmeGJ8CU8CtP0JzPd12SE0Y9Bg07qBJrY/pmpMOJnXXCyDzJw==
      -----END PUBLIC KEY-----

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

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

Etapa	Herramienta	Función
Previa al despliegue	**Trivy**	Escanea las imágenes para detectar vulnerabilidades.
Antes de publicar	**Cosign**	Firma la imagen validada para garantizar su origen y autenticidad.
En ejecución (opcional)	**Connaisseur**	Verifican que solo se desplieguen imágenes firmadas y seguras.


Cosign
Cosign
Connaisseur
Container Image
Sign process
Container signed
Verified sign
Trust container
Deploy Kubernetes
Docker Hub
Container

Demo en vivo:

Intentar deployar imagen sin firma (debe fallar)

Deployar imagen firmada (debe funcionar)

Intentar deployar imagen sin firma (debe fallar)

Aplicando manifiesto de Kubernetes que intenta crear un Pod con una imagen no firmada.

Contenido:
Define un Pod llamado fail-unsigned en el namespace default, que usa la imagen
192.168.56.114:5000/demo/nginx:1.26.
Esta imagen no tiene firma Cosign, por lo que Connaisseur la rechaza durante la validación.

Resultado:
El Pod no se crea. Connaisseur muestra el error:
no matching signatures → imagen sin firma digital válida.

```
[root@m1 connaisseur]# 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 connaisseur]# kubectl apply -f /root/connaisseur/fail-unsigned.yaml
Error from server: error when creating "/root/connaisseur/fail-unsigned.yaml": admission
 webhook "connaisseur-svc.connaisseur.svc" denied the request: error during cosign valid
ation of image 192.168.56.114:5000/demo/nginx:1.26: error validating image: [no matching
 signatures: signature layer sha256:87d09fa7ec9c612ea14a7a0388db9a29ea6bc5b617cfc56c53a7
ea9951b00ecb is missing "dev.cosignproject.cosign/signature" annotation]
```

Intentar deployar imagen con firma

Aplicando manifiesto de Kubernetes que crea un Pod con una imagen firmada digitalmente.

Qué contiene:
Define un Pod llamado ok-signed (también en default), que usa la imagen 192.168.56.114:5000/demo/nginx-ok@sha256:..., la cual sí está firmada con Cosign.
Además, especifica que se ejecute en el nodo m3.

Resultado:
Connaisseur verifica la firma con la clave pública configurada y permite el despliegue.
El Pod pasa a estado Running.

```
[root@m1 connaisseur]# 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@sha256:380f6eda6aca3bbc82ca52b88d7e3a46f6aa
6bb5b0559a21a2c4379e66115833
    imagePullPolicy: IfNotPresent
EOF
[root@m1 connaisseur]# kubectl apply -f /root/connaisseur/ok-signed.yaml
kubectl get pod ok-signed -w
pod/ok-signed created
NAME        READY   STATUS              RESTARTS   AGE
ok-signed   0/1     ContainerCreating   0          1s
ok-signed   1/1     Running             0          2s
^C[root@m1 connaisseur]#
```

BONUS

Kyverno

Kyverno es un motor de políticas (Policy Engine) nativo de Kubernetes desarrollado por Nirmata, que permite definir políticas de seguridad y cumplimiento en forma de manifiestos YAML.

En otras palabras, controla **que se puede** y **que no se puede ejecutar en el cluster**, al verificar cosas como:

- Todos los labels deben tener límites de recursos (CPU, memoria)
- **Integridad de imágenes**: Solo permite imágenes firmadas

Cómo funciona.

Cuando se aplica un manifiesto (un pod o deployment), Kyverno **intercepta la solicitud**, evalúa las reglas declaradas en sus políticas y según la configuración (enforce o audit):

- **Permite** la solicitud si cumple
- **Rechaza** la solicitud si no cumple
- **Modifica el manifiesto** si la política incluye mutaciones

Kyverno vs Connaisseur

Kyverno

Ventajas:

- Muy flexible: permite validar, mutar y generar recursos (no solo imágenes)
- Fácil de integrar con políticas de seguridad y buenas prácticas (labels, limits, namespaces, etc.).
- Buen soporte y comunidad grande

Desventajas:

- Tiene más sobrecarga si se aplican muchas políticas complejas

Connaisseur

Ventajas:

- Especializado en verificar firmas de imágenes (como las de Cosign).
- Configuración simple y directa.
- Está enfocado en garantizar la integridad de imágenes antes de ejecutar.

Desventajas:

- Solo se enfoca en firmas (no puede validar etiquetas, limits, ni mutar recursos)

Cuando usar uno u otro.

Kyverno.

Cuando se requiere un control de seguridad integral del clúster (firmas, configuraciones, normas internas, etc.).

Connaisseur.

Cuando solamente se busca verificar firmas de imágenes y mantener el flujo de seguridad simple

Conclusiones

Trivy identifica vulnerabilidades antes del despliegue.

Cosign garantiza que las imágenes no hayan sido modificadas.

Connaisseur verifica que solo se usen imágenes confiables en el clúster.

Kyverno aplica reglas que fortalecen la seguridad y el cumplimiento.

En conjunto, crean un flujo seguro para el desarrollo y despliegue.
Usarlas ayuda a prevenir fallos y mantener la integridad del sistema.

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. (n.d.). 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/