Public
Cloud
Group

Image Signing and Supply Chain Security.

@ CNCF Meetup Linz, 27/01/2026

Andreas Wimmersberger
Lead Cloud Consultant
andreas.wimmersberger@pcg.io

Agenda.

01. —— **Introduction**

02. —— **CNCF Projects**

03. —— **Kubernetes Security**

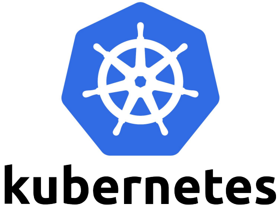
04. —— **Supply Chain Security Frameworks**

05. —— **Supply Chain Stages**

06. —— **Tools**

07. —— **Sign and Verify Container images**

CNCF Projects

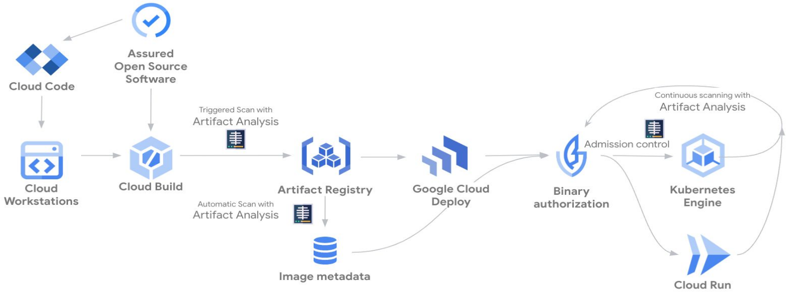


Kubernetes Security

- **Supply Chain Security**
  - SBOM / providence artifact generation
  - Vulnerability and compliance scanning (Grype)
  - Image signing (Notary & Ratify)
- **Cluster Security**
  - Secure API endpoint (authorized IP-ranges, mTLS)
  - Use robust RBAC for access control (OIDC)
  - Use cluster auto-upgrade (if possible)
  - Don’t install the dashboard (or if you do, do is the right way)
- **Node Security**
  - Automatically update node images
  - Disable SSH access
  - For potentially hostile workloads use compute isolation capabilities
    - Use confidential compute nodes (based on Intel SGX)
    - Confidential Containers - based on Kata Containers (using AMD SEV-SNP)
    - Pod Sandboxing
- **Network Security**
  - Deploy a network policy engine to secure pod network communications (Calico, Cilium, NPM)
  - Deploy WAF for ingress
- **Application Security**
  - Continuous scanning of running pods
  - Use a credential vault for storing secrets (CSI integration)

Container Secure Supply Chain Frameworks (Examples)

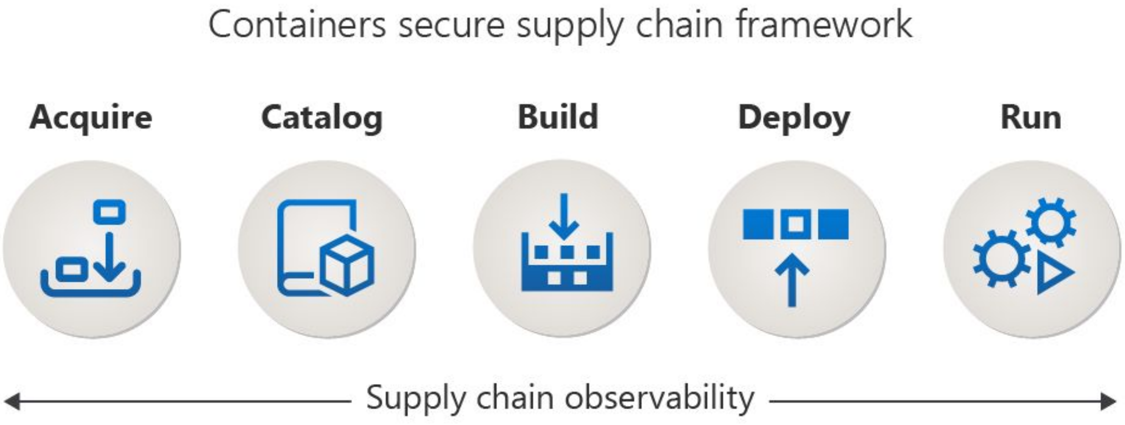
- **CNCF Software Supply Chain Best Practices**
  - Good overview of the topic
  - Unfortunately (very) outdated
  - Basis for many other frameworks
- **GCP Software supply chain security**
  - Comprehensive
  - Good documentation
  - Uses Google tools
- **Supply-chain Levels for Software Artifacts (SLSA)**
  - Part of the Open Source Security Foundation (OpenSSF)
  - Vendor neutral
  - Uses levels (get as much security as you need or can afford)
  - Has guidance for developers, organizations and infrastructure providers



Containers Secure Supply Chain Framework (CSSC)

- **The CSSC framework is built using the following steps:**
  - Identify the supply chain stages for containerized applications
  - Outline the risks and the required security controls in each stage
  - Describe the security objectives and goals in each stage
  - Identify security tools, processes, and best practices in each stage
  - Maintain security posture with metadata, logging, and reporting in each stage

- **Defines the following stages in supply chains:**


Containers secure supply chain framework
Acquire
Catalog
Build
Deploy
Run
Supply chain observability

Supply Chain Stages

- **Acquire**
  - Acquire container images from external sources or third-party vendors, e.g. os images, service proxies or logging and metric images.
- **Catalog**
  - Offer approved container images for internal consumption including builds and deployments.
- **Build**
  - Produce compliant service and application images and deployment artifacts.
- **Deploy**
  - Securely deploy containerized services and applications to the hosting environments.
- **Run**
  - Run containers created from compliant, latest, and secure container images executing the business logic for an application.

Types of Supply Chain Compromise

- **Dev Tooling**
  - Attack on development machine, SDK, tool chains, or build kit, Often results in backdoor access.
  - Mitigation: Use trusted binary repos and verify signatures and checksums.
- **Negligence**
  - Lack of adherence to best practices, e.g. missing dependency name checks etc.
- **Publishing Infrastructure**
  - The integrity or availability of shipment, publishing, or distribution mechanisms are compromised.
  - Mitigation: Code Signing
- **Source Code**
  - Source code repository (public or private) is manipulated maliciously.
- **Trust and Signing**
  - A signing key used is compromised.
- **Malicious Maintainer**
  - A maintainer, or an entity posing as a maintainer, deliberately injects a vulnerability
- **Technique: Attack Chaining**
  - Multiple attack vectors are chained together.

Acquire Stage

Possible workflow for the acquisition of external images:

- Import container images and cloud-native artifacts into an internal registry.
- Quarantine the images in the internal registry.
- Validate any signatures associated with the image.
- Validate any other metadata associated with the image including SBOMs and provenance.
- Scan the images for known vulnerabilities and malware.
- Attach the vulnerability and malware reports as image attestations.
- If not available, generate SBOM and provenance for the image and attach them .
- Sign the image and relevant metadata with enterprise keys to ensure integrity.
- If the image meets the internal policies, publish the image to the golden registry for internal use.

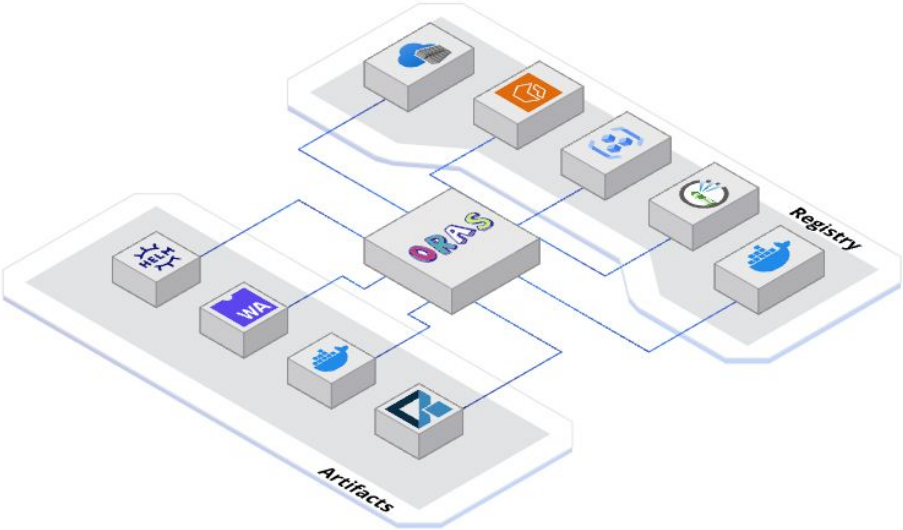
Tools used:

- An OCI-compliant container registry (Docker Hub, GitHub Container Registry (GHCR), Azure Container Registry (ACR), Google Container Registry (GCR), Amazon ECR, …)
- OCI Registry As Storage or ORAS (metadata enrichment for images)
- Notary & Notation (image signing)
- SBOM Tool (Creation of SBOMs)
- Alternative SBOM tools: CyclineDX, Sfyt (open-source), Fossa, Finite State, …

OCI Registry As Storage (ORAS)

The ORAS project provides a means to enable various client libraries with a way to push OCI Artifacts to OCI-conformant registries:

- Provides the ORAS CLI
- Github action is available for install
- Works with all OCI-compliant registries, see here.



SBOM Tool

The SBOM tool is a to create SPDX compatible SBOMs for any variety of artifacts:

- Supported on Windows, Linux and macOS
- Open-source (MIT) license
- Can create SBOMs in SPDX 2.2 or 3.0 formats
- Uses Component Detection for component detection and ClearlyDefined for license information.

```
- name: Generate SBOM
   run: |
    curl -Lo $RUNNER_TEMP/sbom-tool
https://github.com/microsoft/sbom-tool/releases/latest/download/sbom-tool-linux-x64
    chmod +x $RUNNER_TEMP/sbom-tool
    $RUNNER_TEMP/sbom-tool generate -b ./buildOutput -bc . -pn Test -pv 1.0.0 -ps MyCompany -nsb

https://sbom.mycompany.com -V Verbose
```

Catalog Stage

Possible workflow for cataloging images:

- Hosts the container images that pass quality checks in an internal staging registry.
- Catalog container images to enable internal teams to easily discover images.
- Schedule vulnerability and malware scans on a regular cadence
- Signs the reports with enterprise keys to ensure integrity and provide a trusted stamp of approval for internal use.
- Monitor the lifecycle of container images in the catalog and retire the images that are out of support.

Tools used:

- An OCI-compliant container registry (Docker Hub, GitHub Container Registry (GHCR), Azure Container Registry (ACR), Google Container Registry (GCR), Amazon ECR, …)
- Vulnerability scanning tool (Trivy, Grype, Defender for Cloud, ..)
- Notary & Notation (image signing)

Build Stage

Possible workflow for building & using images:

- Pull base images from the internal catalog only.
- Verify base images before using it to ensure it is trustworthy and compliant.
- Add additional frameworks, application code, and/or configurations from trusted sources on top of base images for build.
- Generate SBOM during the build process.
- Scan the resulting container image for known vulnerabilities.
- Patch the resulting container image to address known vulnerabilities and malware.
- Attach the vulnerability and malware reports as image attestations.
- Attach SBOM as image attestations to use in subsequent stages of the supply chain.
- Sign the resulting image and relevant metadata with enterprise keys to ensure integrity.

Tools used:

- Dependabot (automated dependency updates)
- Copacetic & copa (container patching)
- SBOM Tool
- Vulnerability scanning tool (Trivy, Grype, Defender for Cloud, …)
- Notary & Notation (image signing)

Deploy Stage

Possible workflow for deploying images:

- Implement Image integrity policy to verify image signatures before deployment.Implement Vulnerability scanning policy to scan container images for vulnerabilities.
- Implement Image lifecycle policy to ensure deployed images are within support and valid.
- Generate and sign vulnerability and malware reports for each image.
- Attach the signed reports to container images for visibility and compliance validation.
- Verify container image metadata.
- Implement admission control mechanisms to enforce deployment policies.
- Automate deployment processes with CI/CD pipelines, integrating image validation and verification checks.
- Continuously monitor deployed images and enforce compliance.
- Log deployment activities and conduct regular audits.
- Implement automated or manual remediation procedures to address security incidents.

Tools used:

- Ratify (verify image metadata according to admission policy)
- Gatekeeper (admission controller)
- Open Policy Agent (state admission policies)
- Vulnerability scanning tool (Trivy, Grype, Defender for Cloud, …)
- Notary & Notation (image signing)

Run Stage

Possible workflow for building & using images:

- Continuously scan for vulnerabilities and malware in containerized workloads.
- Regularly update containers and worker nodes.
- Check the image lifecycle metadata to identify outdated images.
- Regularly clean up stale images from the cache on the node.
- Configure strong authentication and authorization mechanisms on hosting environments and containers, as well as running containers.
- Reduce attack surface by restricting container and node ports, restricting network access of containers, enabling mutual TLS, and enforcing resource constraints to containers..

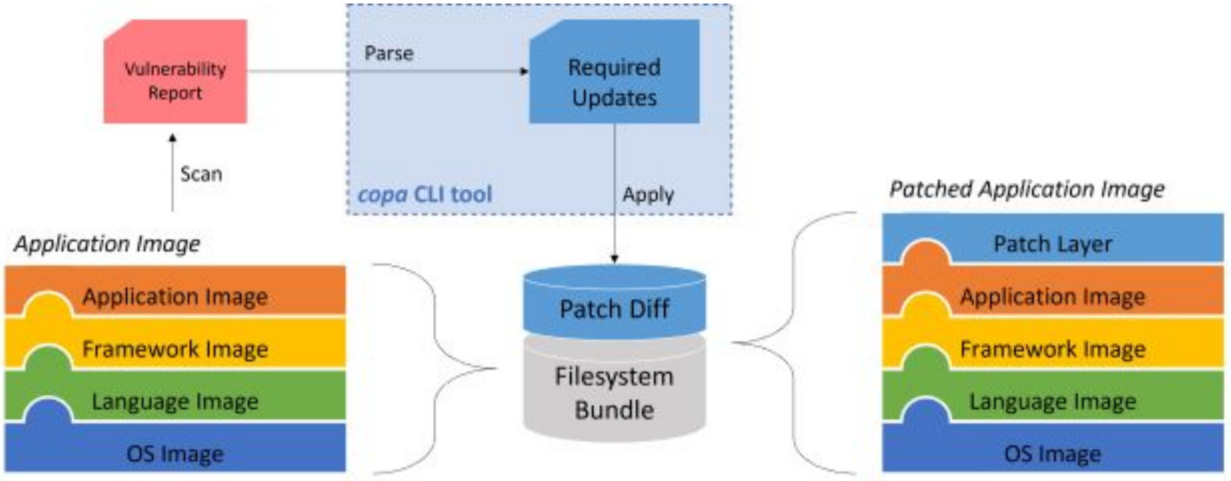
Tools used:

- Dependabot (automated dependency updates)
- Copacetic & copa (container patching)
- SBOM Tool
- Vulnerability scanning tool (Trivy, Grype, Defender for Cloud, ..)
- Notary & Notation (image signing)
- Eraser (image cleaning)

Copacetic (copa)

copa is a CLI tool written in Go and based on buildkit that can be used to directly patch container images given the vulnerability scanning results from popular tools like Trivy.

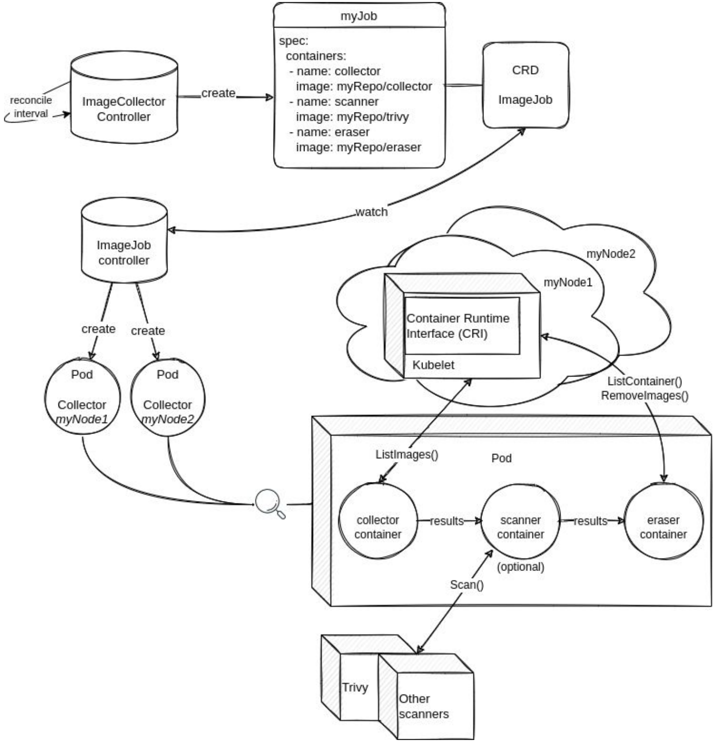
- CNCF sandbox project
- Supports patching existing container images
- Supports containers **without** package managers including distroless containers
- Works with the **existing** vulnerability scanning and mitigation ecosystems, e.g. Trivy, Grype
- GitHub Action and Docker Desktop extensions available



Eraser

Eraser aims to provide a simple way to determine the state of an image, and delete it if it meets the specified criteria.

- CNCF sandbox project
- Simple
- Installed via manifest file or Helm chart
- Configuration via configmap eraser-manager-config
- Two modes of operation: manual or automated
- Targets no-running images only
- Exclusions can be registries or specific images
- Supports Trivy or custom scanners



myJob

```
spec:
  containers:
   - name: collector
     image: myRepo/collector
   - name: scanner
     image: myRepo/trivy
   - name: eraser
     image: myRepo/eraser
```

Signing and verifying OCI artifacts

Types of OCI artifacts

- Container Images
- Software bills of materials (SBOMs)
- Helm charts
- Configuration bundles
- Ai models

Signing and verification ensures:

- **Integrity:** The artifact that you use is exactly the same as the one that was published.
- **Authenticity**: The artifact truly came from the expected publisher.

Processes used are

- **Signing:** Produces cryptographic signatures that bind a publisher's identity to an artifact descriptor, including the digest.
- **Verification:** Checks that a signature is valid, the publisher's identity is trusted, and the artifact isn't altered.

Tools:

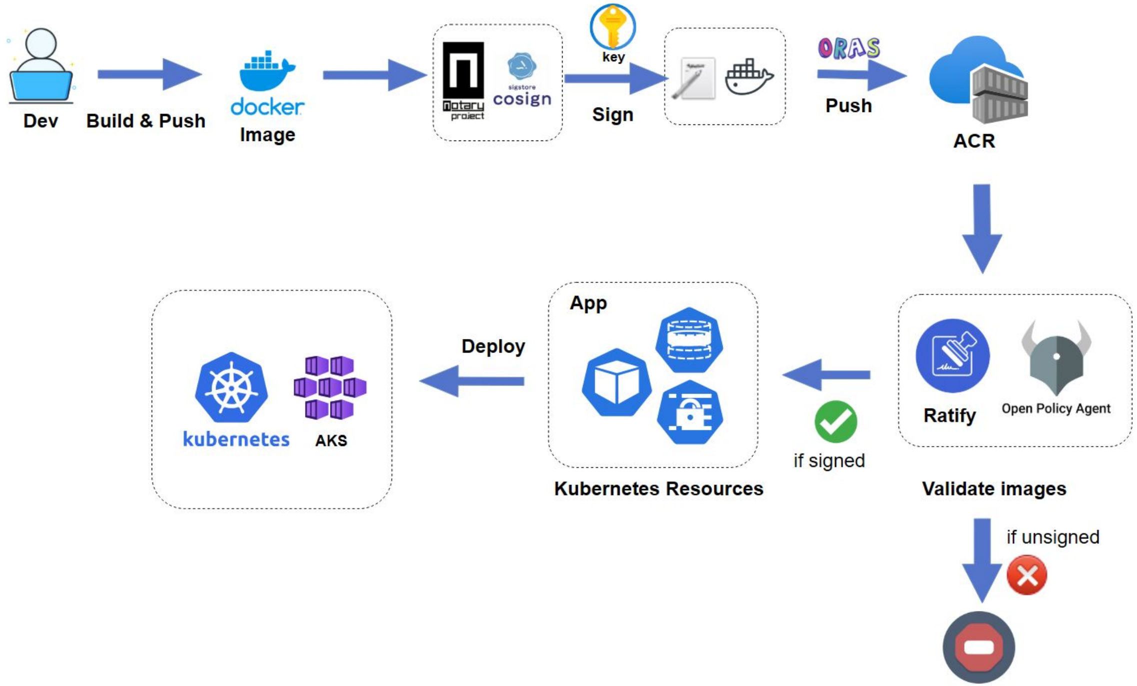
- **Cosign**
- **Docker Content Trust (DCT)**
- **Notation (Notary v2)**

Notary (notation)

The Notary Project is designed to secure software supply chains by enabling digital signatures and verification for container images and other OCI (Open Container Initiative) artifacts.

- Open-source
- CNCF incubating project
- Supports any software artifacts, not only images
- Supports COSE (CBOR Object Signing and Encryption) and JWS signature formats
- Supports OCI-compliant registries
- Github Actions for installation and signing available
- Used by Ratify to verify signatures

Notary Workflow



Sign container images with notation and AKV

Workflow:

- **Certificate requirements for root and intermediate certificates:**
  - The basicConstraints extension must be present and marked as critical. The CA field must be set to true.
  - The keyUsage extension must be present and marked as critical. Bit positions for keyCertSign must be set.
- **Certificate requirements for leaf (signing) certs:**
  - Subject must contain common name (CN), country/region (C), state or province (ST), and organization (O).
  - X.509 key usage flag must be DigitalSignature only.
  - Extended Key Usages (EKUs) must be empty or 1.3.6.1.5.5.7.3.3 (for code signing).
  - Key must NOT be exportable
  - Use supported key type and size:Algorithm Selection
- **Import certificate in Key Vault**
- **Authorize access to Key Vault (use RBAC)**
- **Use notation GitHub actions in image build pipeline to sign artifacts**

Ratify

Ratify is a verification engine as a binary executable and on Kubernetes which enables verification of artifact security metadata and admits for deployment only those that comply with policies you create..

- Open-source
- CNCF sandbox project
- Can verify signatures, validate checksums, and ensure that artifacts are up-to-date
- Flexible verification policies
- (Relatively) easy installation
- Consists of the following parts:
  - Executor
  - Referrer Store
  - Reference verifier
  - Policy Providers

Validate container image signatures in AKS with Ratify

Workflow:

- **Set up identity and access controls for Container Registry:** Configure the identity that Ratify uses to access Container Registry with the necessary roles.
- **Set up identity and access controls for Key Vault:**
  - Configure the identity that Ratify uses to access Key Vault with the necessary roles.
- **Set up Ratify on your AKS cluster:** Set up Ratify by using a Helm charts
- **Set up a custom Azure policy:** Create and assign a custom Azure policy with the desired policy effect: Deny or Audit.

Prerequisites:

- **Install latest Azure CLI, Helm and kubectl**
- **Enable OIDC issuer an AKS cluster**
- **Connect Azure Container Registry to AKS cluster**
- **Enable Azure Policy add-on (enables Gatekeeper and OPA Agents)**

Link Collection

- Software supply chain security | Google Cloud
- Supply-chain Levels for Software Artifacts
- US DoD Securing the Software Supply Chain
- NIST SP 800-204D
- OWASP Supply Chain Security
- Notary
- Ratify
- Copacetic
- Eraser
- Sbom-tool
- ORAS

Questions?

GET IN TOUCH WITH US

Let's work together.

Public
Cloud
Group

Andreas Wimmersberger
Lead Cloud Consultant
andreas.wimmersberger@pcg.io

With a product portfolio designed to accompany organizations of all sizes in their cloud journey and competence that is a synonym for highly qualified staff that customers and partners like to work with, PCG is positioned as a reliable and trustworthy partner for the hyperscalers, relevant and with repeatedly validated competence and credibility.

We have the highest partnership status with the three relevant hyperscalers. As experienced providers, we advise our customers independently.

PUBLIC CLOUD GROUP GMBH
Peter-Behrens-Platz 10
4020 Linz

VISIT OUR WEBSITE

aws