

WHITEPAPER

A Disciplined Approach to Container Supply Chain Security: From Image Intake to Runtime Enforcement

RUNTIME

SUPPLY CHAIN

KYVERNO



Introduction

Organizations often rely on public container registries for base images and third-party components, leading to inconsistent security posture, unclear provenance, and fragmented operational control. RapidFort addresses this by providing a curated library of container base images, including runtimes such as Python, Node.js, and Java, in addition to Linux distributions such as Ubuntu, Red Hat UBI, and Alpine. RapidFort also provides commonly used third-party application images, including load balancers, databases, and other infrastructure components, giving organizations a consistent and trusted starting point for containerized workloads.

Rather than allowing individual development teams to independently source and manage these images, organizations adopt a centralized secure model that standardizes how images are ingested, validated, and distributed. This paper outlines a disciplined approach to enforcing that model, from upstream image sourcing to build-time controls and runtime enforcement within Kubernetes.



Key Insight: A disciplined model starts with curated upstream images, extends through standardized CI pipelines, and is enforced at runtime within Kubernetes, moving organizations from fragmented practices to a controlled and verifiable software supply chain.

Stage 1: Upstream Sourcing from RapidFort

Stage 1 establishes control at the organization's image-ingress point by centralizing, controlling, and securing how approved container images are selected, verified, customized, and published for internal use. This function is typically owned by a Platform Operations or Platform Security team and implemented through a certified, secure image pipeline.

In this model, approved images with zero or near-zero CVEs that are pre-scanned by RapidFort for malware and tampering are sourced from the RapidFort curated image repository rather than directly from arbitrary public upstream repositories. These curated images are inherently "cooled down" to avoid introducing zero-day vulnerabilities and protected with malware removal threat intelligence capabilities. During ingestion, each image should be verified using a tool such as Cosign.

For base images that development teams will eventually consume and build on top of, customization happens during this stage. This may include adding internal CA certificates, applying organization-specific labels, etc. In this workflow, the certified secure image pipeline builds the derived image, signs it on behalf of the organization, and generates required attestations for the resulting artifact.

The resulting certified secure images are published to an internal registry and become the only approved base images available to development teams. This approach gives Platform Operations centralized control over the container image supply chain and ensures that every base image has been validated before use.

The following example pipeline illustrates how this image workflow can be implemented.

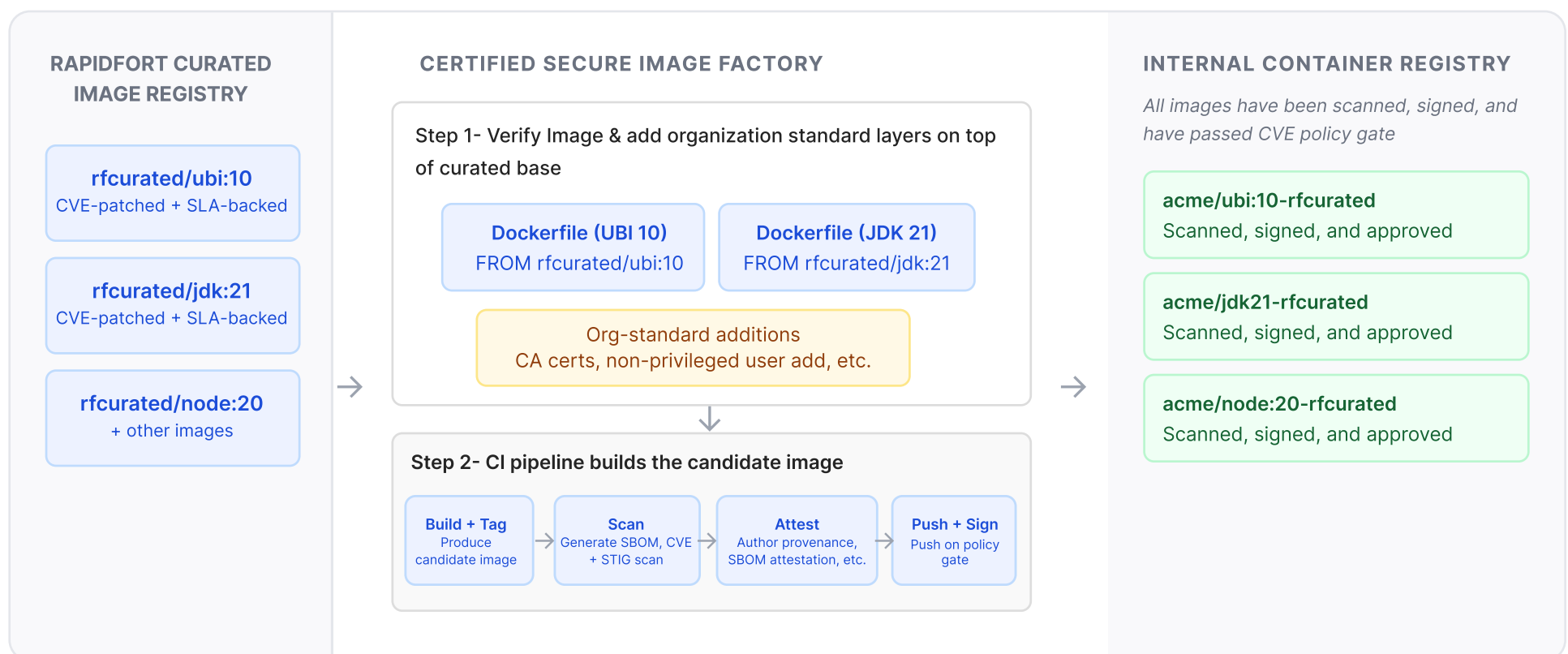
Zero or near-zero CVE images

Cosign verification at ingestion

Provenance attestations generated

Centralized registry control

CERTIFIED SECURE IMAGE FACTORY — PIPELINE OVERVIEW



For third-party images that will be used directly and require no modification, such as database, load balancer, or vendor images, the organization can validate the image, approve it, and make it available for downstream consumption without rebuilding it.

Once approved base and third-party images are available through the internal registry, Platform Operations can extend this control into the developer workflow through a standardized application image pipeline. This creates the foundation for Stage 2, where build-time enforcement ensures that development teams consistently build application images from approved sources with the required security checks in place.

Stage 2: Build-Time Enforcement for Development Teams

Stage 2 focuses on enforcing standardized security controls within development teams' CI pipelines. Applying these controls at build time is critical, as it prevents non-compliant container images from being created in the first place. This enforcement typically includes validating base image sources and ensuring adherence to established best practices. The example below illustrates how these checks can be implemented using Hadolint, an open source Dockerfile linter. Additional tools, such as Conftest, can be introduced when more advanced or customizable policy enforcement is needed.

Dockerfile Linting with Hadolint

Hadolint provides a practical way to enforce Dockerfile standards directly within CI pipelines. It validates Dockerfiles against established best practices and, importantly, supports a trusted registry configuration that allows teams to explicitly whitelist approved image sources.

The example below shows a simple rule configuration that rejects any **FROM** instruction that references an unapproved registry:

```
JSON
# .hadolint.yaml
trustedRegistries:
- registry.internal.acme.com
```

Example: Compliant vs. Non-Compliant Dockerfile

The following examples illustrate how Hadolint evaluates Dockerfiles when a trusted registry policy is enforced.

Compliant Dockerfile

```
None
FROM registry.internal.acme.com/rapidfort/python:3.14
...
```

Non-Compliant Dockerfile

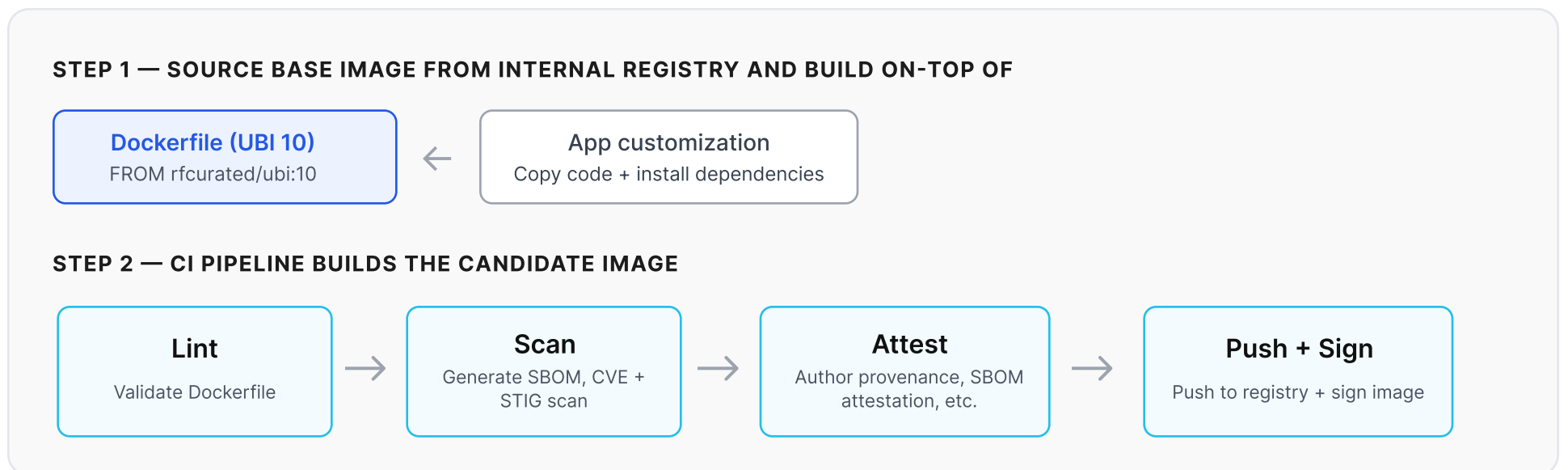
```
None
FROM python:3.14
...
```

Hadolint Output

```
None
Dockerfile:1 DL3026 error: Use only an allowed registry in the FROM image
```

This output shows how images sourced from unapproved registries are immediately flagged during the build process, preventing them from progressing further in the pipeline.

Sample Developer Pipeline



The above example illustrates how these controls come together within a typical developer CI pipeline. The process starts by sourcing a base image from an approved internal registry, ensuring all builds inherit from a trusted foundation. Developers then layer application code and dependencies on top of this base image as part of standard development workflows.

Once the build is triggered, the pipeline enforces a sequence of automated checks. Dockerfiles are first validated against policy using linting tools to confirm compliance with approved standards. The resulting image is then scanned to generate an SBOM and identify known vulnerabilities or configuration issues. After validation, the image is attested with provenance and metadata, establishing a verifiable record of how it was built. Finally, only compliant images are pushed to the registry and cryptographically signed, making them eligible for downstream deployment.

This approach ensures that security and compliance are embedded directly into the developer workflow, rather than applied as a separate or downstream control point.

Stage 3: Runtime Enforcement in Kubernetes

Build-time controls are effective at catching issues early, but they are not sufficient on their own. The runtime stage serves as the final enforcement point, ensuring that only approved and compliant images are allowed to run within the cluster.

Kubernetes policy engines such as [Kyverno](#) or [Gatekeeper](#) enable administrators to define and enforce these controls at deployment time. While commonly used to validate image sources and signatures, these tools also support a broader set of Kubernetes-native policies, including enforcing security contexts, resource limits, network controls, and other configuration standards across workloads.

Kyverno Overview

Kyverno is a Kubernetes-native policy engine that runs as an admission controller. It evaluates API requests, such as Pod creation or Deployment updates, before they are persisted to the cluster. Based on declarative policies, Kyverno can validate requests, block non-compliant resources, apply mutations, or generate audit findings for visibility.

Validate

Inspect API requests against policy rules

Block

Reject non-compliant resources at admission

Mutate

Patch or transform resources on the fly

Audit

Generate findings for visibility without blocking

Restricting Image Registries with Kyverno

The following policy enforces that only container images sourced from approved internal registries are allowed to run within the cluster. This prevents workloads from pulling images directly from public or untrusted sources, reducing the risk of introducing vulnerable or tampered artifacts.

In Kyverno, this is implemented as a validation policy that inspects all container image references within a Pod specification, including initContainers and ephemeral containers. Any request that includes an image outside the approved registry list is denied at admission time.

The example below shows how this policy is defined in Kyverno. Annotations have been omitted for clarity.

JSON

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-image-registries
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: require-approved-registry
      match:
        any:
          - resources:
              kinds:
                - Pod
      validate:
        message: "Container images must come from the approved internal registry."
        pattern:
          spec:
            containers:
              - image: "registry.internal.acme.com/*"
            =(initContainers):
              - image: "registry.internal.acme.com/*"
            =(ephemeralContainers):
              - image: "registry.internal.acme.com/*"
```

Extending Runtime Enforcement with Kyverno

Registry enforcement is only one aspect of a broader runtime enforcement strategy. Kyverno policies can be extended to verify image signatures at admission time, ensuring that only trusted, signed artifacts are allowed to run and blocking any unsigned or tampered images before they reach the cluster.

Beyond signatures, Kyverno can enforce the presence and validity of attestations generated during the build process, including vulnerability scan results, SBOMs, and other metadata tied to the image. These attestations can be cryptographically verified against trusted authorities or CI identities, allowing policies to confirm that images were produced through approved pipelines and meet defined security requirements. These controls enable organizations to move beyond basic allowlisting and implement a comprehensive runtime enforcement model.

While Kyverno is used in this example, the underlying approach is not tool-specific. As mentioned earlier, other policy engines, such as Gatekeeper, can be used to enforce similar controls. The key requirement is the presence of an admission controller capable of evaluating and enforcing policy at deployment time.

Closing Remarks

Establishing a secure and scalable container strategy requires more than point-in-time controls. It demands a consistent, end-to-end approach that governs how images are sourced, built, and ultimately deployed. By introducing a disciplined model that starts with curated upstream images, extends through standardized CI pipelines, and is enforced at runtime within Kubernetes, organizations can move from fragmented practices to a controlled and verifiable software supply chain.

RapidFort enables this model by delivering curated and trusted base and third-party images. When combined with a certified secure image pipeline, security and development teams gain control over the origin and composition of every image. Build-time enforcement within developer CI pipelines ensures that this trust is not bypassed, while runtime controls act as the final gate to guarantee that only compliant workloads are allowed to run.

This approach shifts security from a reactive process to an embedded capability within the development lifecycle and enables security teams to continuously meet vulnerability objectives without inhibiting development teams. Each stage reinforces the next, creating a system where trust is continuously validated rather than assumed. The result is not only improved security posture, but also greater consistency, auditability, and operational efficiency across teams.

By adopting this model, organizations can confidently scale their use of containers while maintaining strict control over software supply chain integrity from source to runtime.



ABOUT RAPIDFORT

RapidFort is the leader in Software Supply Chain Security, enabling organizations to eliminate risk across their software stack at scale. Its platform combines curated near-zero CVE container images, runtime profiling, and attack surface management to remove up to 99.9% of vulnerabilities within hours and reduce the attack surface by up to 90%, without code changes. RapidFort was identified as a Gartner Cool Vendor in 2025 and a Nutanix .Next Partner of the Year in 2026.

© 2026 RapidFort, Inc.

Eliminate attack vectors at the source

[Schedule a Call ↗](#)

www.rapidfort.com | sales@rapidfort.com