
# Image Signing and Runtime Verification at Scale: Datadog's Journey

*Ethan Lowman, Datadog*

# Datadog



Observability and security platform

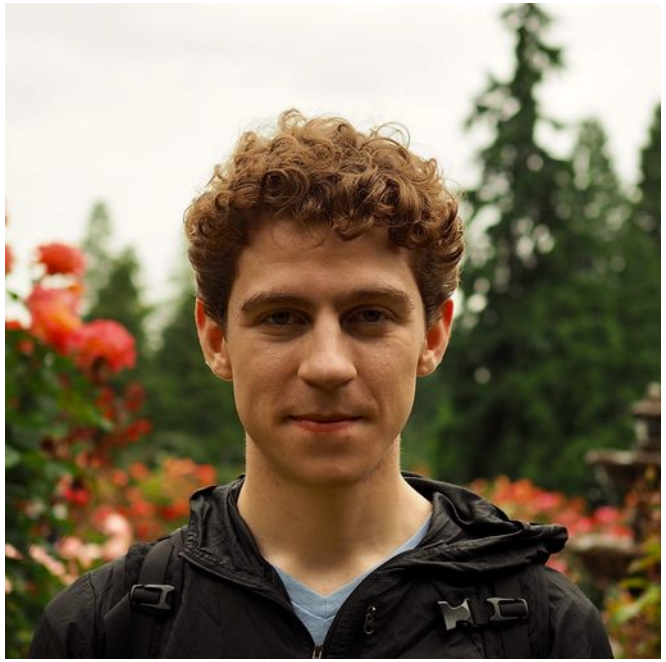
- Tens of trillions of events per day
- Millions of hosts
- Over 600 integrations
- Almost 5,000 employees

Runs on self-hosted Kubernetes

- Dozens of clusters
- Tens of thousands of nodes
- Hundreds of thousands of pods

# About Me

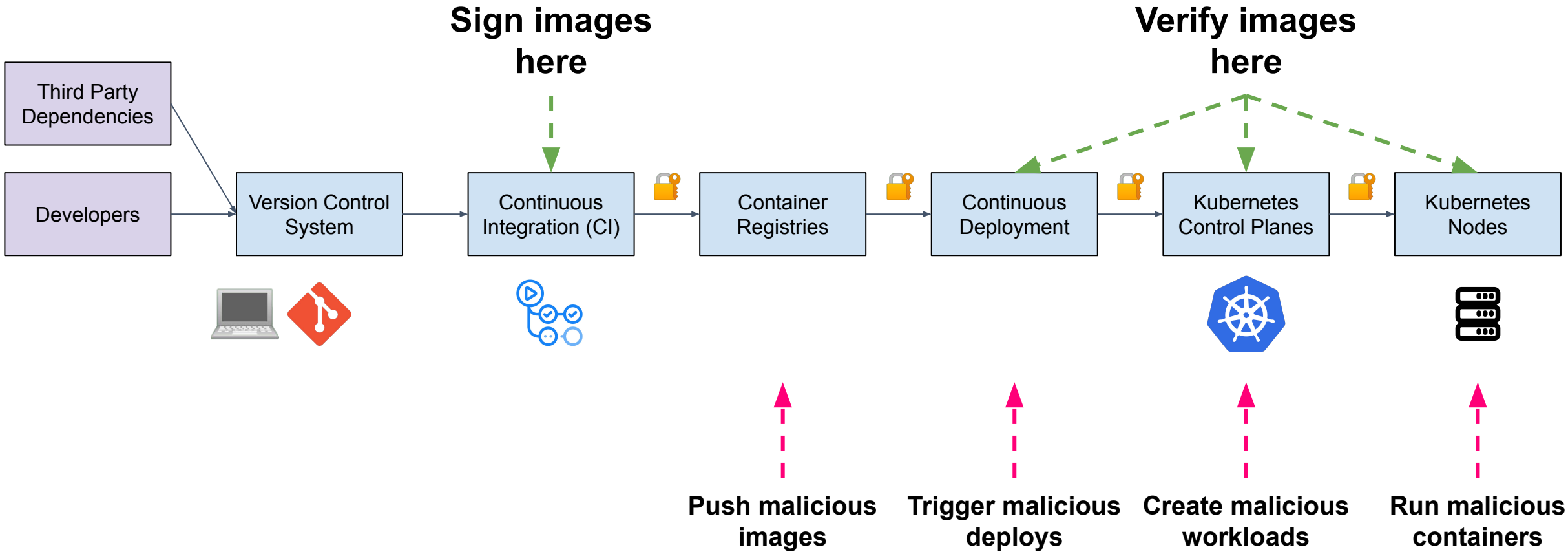




**Ethan Lowman**
Senior Software Engineer
*Datadog*

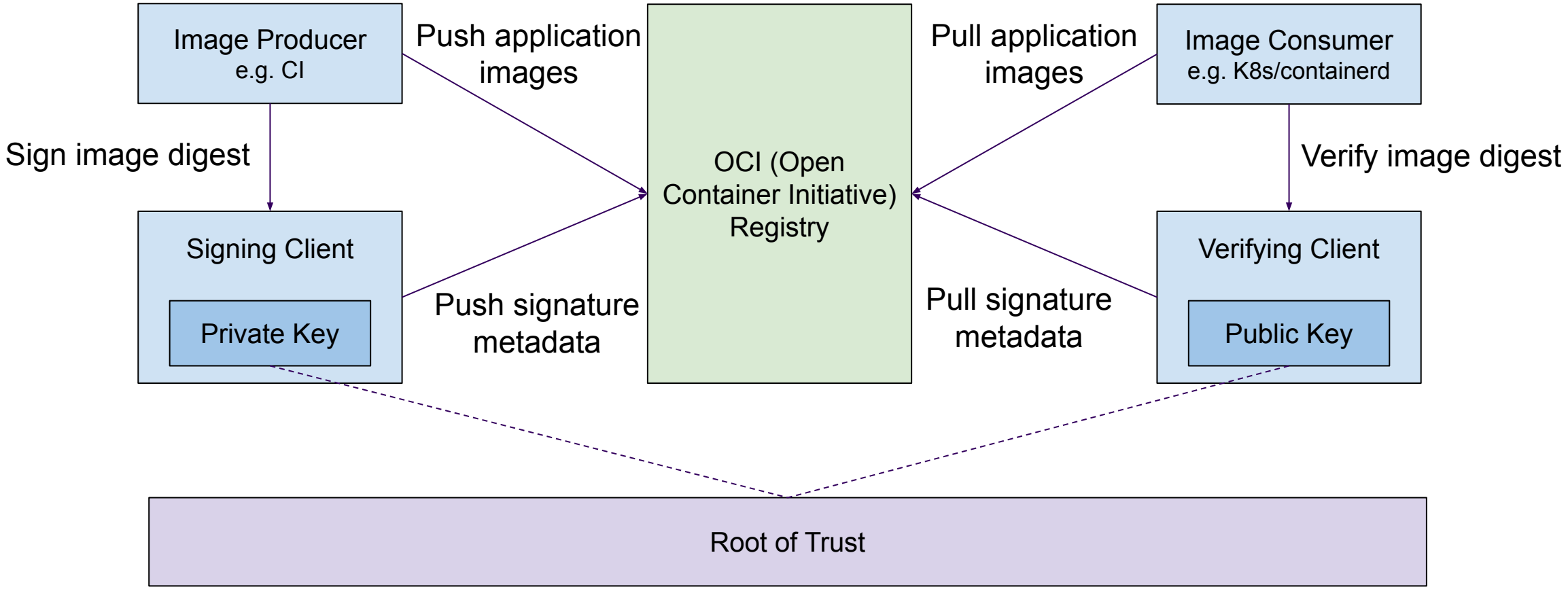
ethan.lowman@datadoghq.com

# Why sign & verify images?



# Modern Consensus on Image Signing



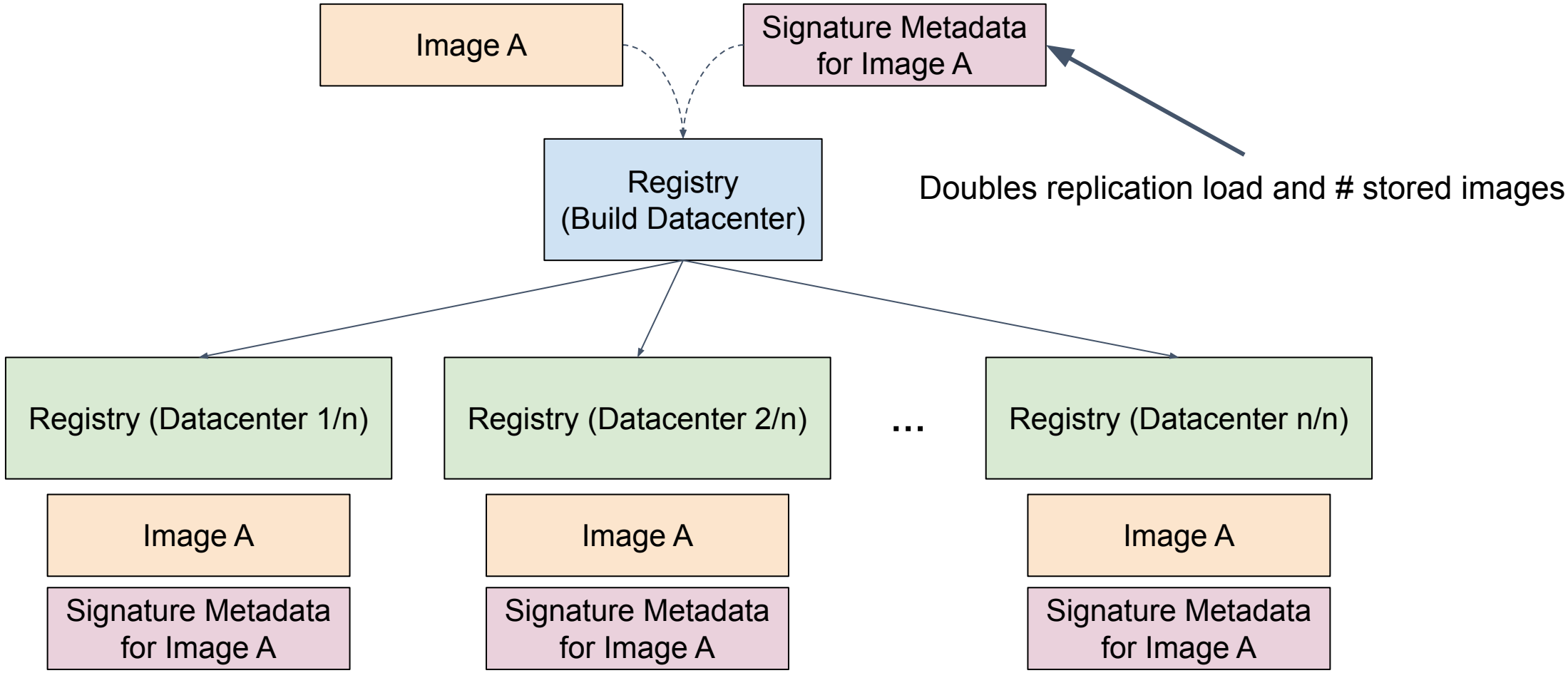



**Examples:** Sigstore/cosign, Notary v2

# Signature Metadata in a Registry



- The registry is not a new runtime dependency
- Signature replication is a solved problem if you're already running images in multiple datacenters



# Signature Format



Our design is loosely based on Sigstore's cosign signatures.

**Caveat**

Strongly consider using an open standard if one is available, especially if you need interoperability. Cosign might meet our requirements if we started over today.

# Signature Format: Payload

KubeCon CloudNativeCon Europe 2023

OCI Descriptor with custom annotations for signature metadata

```
{
  "annotations": {
    "com.datadoghq.image-integrity.signer.claims.timestamp": "2023-02-03T21:48:33-04:00",
    "com.datadoghq.image-integrity.signer.claims.client_subject": "0638c9b7-43b3-a2fe-...",
    "com.datadoghq.image-integrity.signer.claims.client_email": "example@internal.service.identity",
    // etc.
  },
  "digest": "sha256:adab3844f497ab9171f070d4cae4114b5aec565ac772e2f2579405b78be67c96",
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "size": 1638
}
```

Digest, media type, and size of the protected artifact

Custom signature metadata fields (e.g. signer identity & environment)

# Signature Format: Envelope



**Signature Algorithm:** Ed25519

**Envelope Format:** Dead Simple Signing Envelope (DSSE)

```
{
  "payload": "<base64-encoded OCI Descriptor>",
  "payloadType": "application/vnd.oci.descriptor.v1+json",
  "signatures": [
    {
      "keyid": "SHA256:TfImnpH/n1t3tBnj0Nv8XGcejhvxFye4vdrk+TwyKnE",
      "sig": "<base64-encoded Ed25519 signature>"
    }
  ]
}
```

# Signature Format: OCI Layers



Each OCI layer stores one DSSE blob, annotated by key ID for direct lookup


```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 233,
    "digest": "sha256:c4a08471155229261aa4944de79b3b7c3206105f64299338cd055f9f4c59a2bf"
  },
  "layers": [
    {
      "mediaType": "application/vnd.datadog.image-integrity.dsse.v1+json",
      "size": 1360,
      "digest": "sha256:94047bef27d03f89ff17184a9d00a5e38463a0d869dcf4b676ba8c8b8352c083",
      "annotations": {
        "com.datadoghq.image-integrity.v1/key-id": "SHA256:TfImnpH/n1t3tBnj0Nv8XGcejhvxFye4v..."
      }
    }
  ]
}
```


One layer = one signing envelope for one key

# Signature Format: Registry Layout



Resolve signature reference from signed digest

**Signed Artifact Location:**

`registry.example.com/my/image@sha256:abc123...`

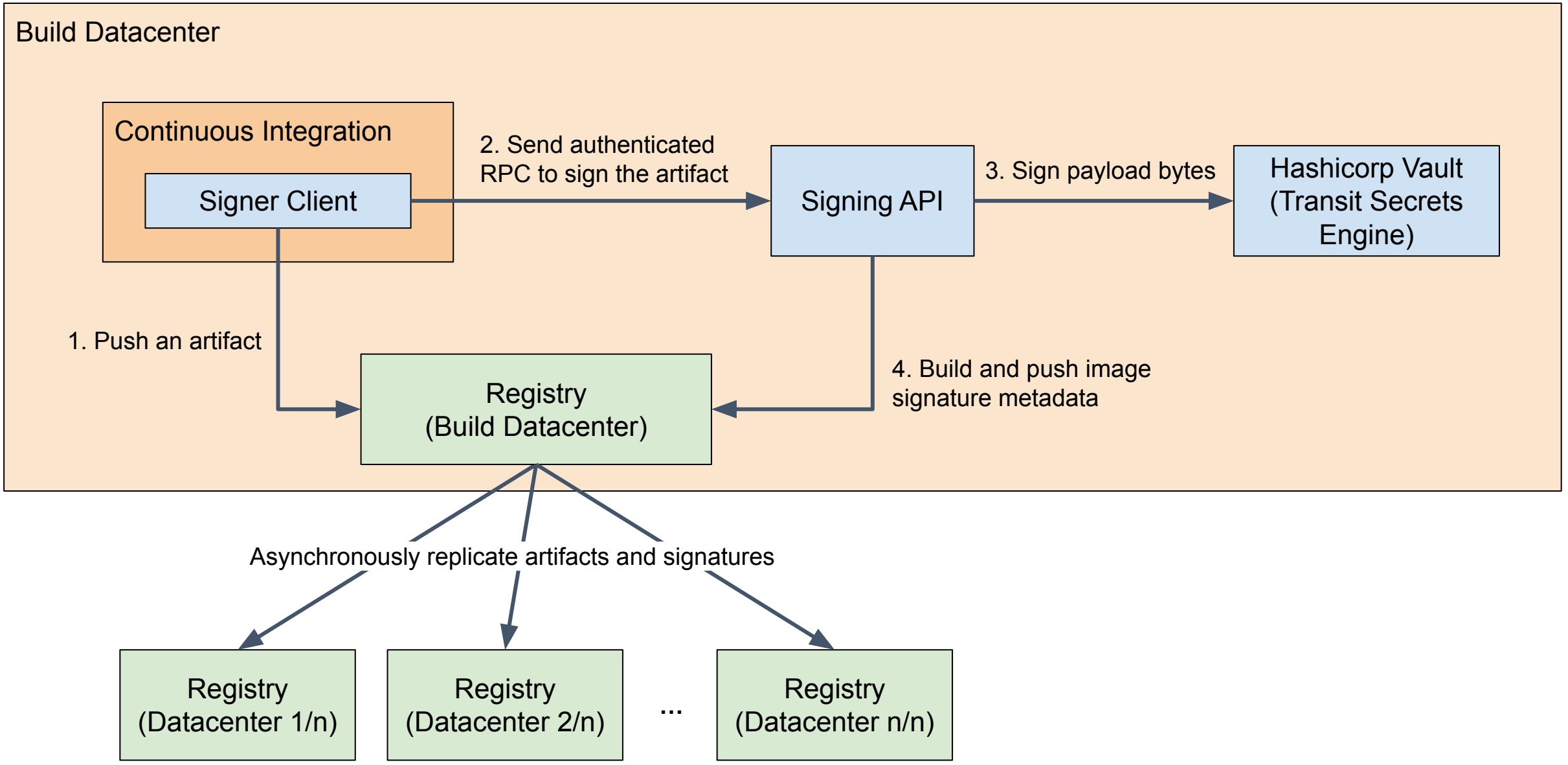
**Signature Location:**

`registry.example.com/signatures/my/image:sha256-abc123...`

*Property:*

Signatures have dedicated cloud registry storage quotas, independent from the signed images.

# Signing as a Service



# Signing Thin Client



**General Purpose (Shell)**

```
ddsign sign registry.internal/my-image:v3@sha256:abc123...
```

**Docker-Built Images (Shell)**

```
ddsign sign registry.internal/my-image:v3 --digest-from-docker
```

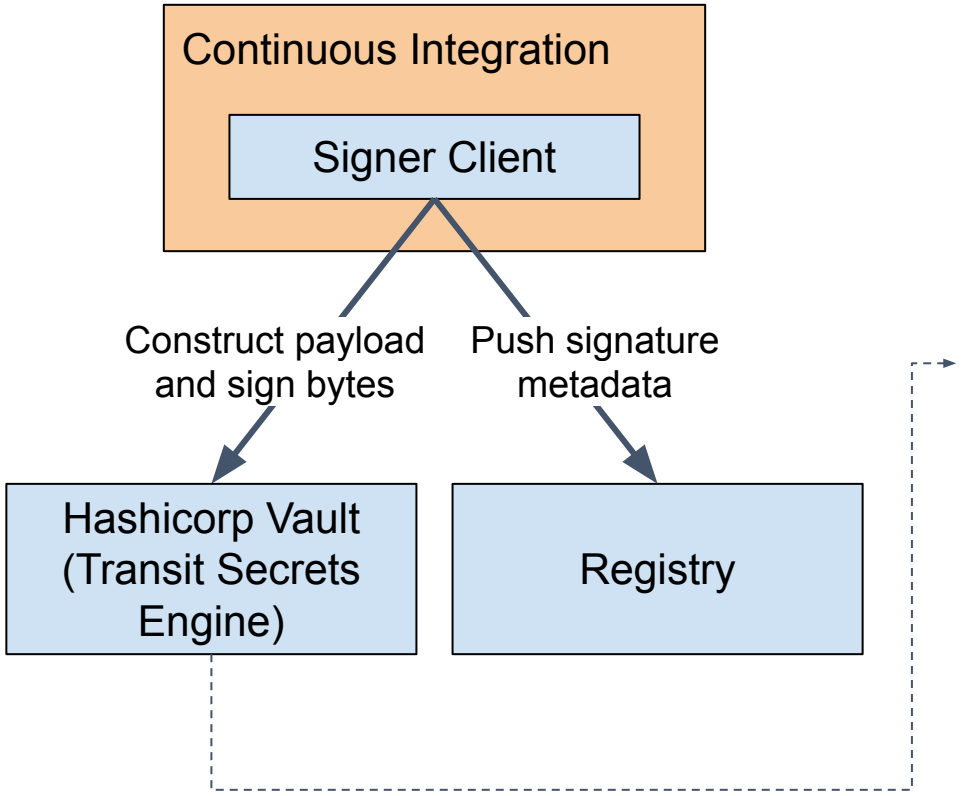
**Bazel**

```
dd_container_sign(
    name = "sign-my-image",
    pushed_image = ":pushed-my-image",
)
```

# Signing Service: Least Privilege & Auditability



Suppose CI signed with Hashicorp Vault directly:




## Hashicorp Vault audit logs

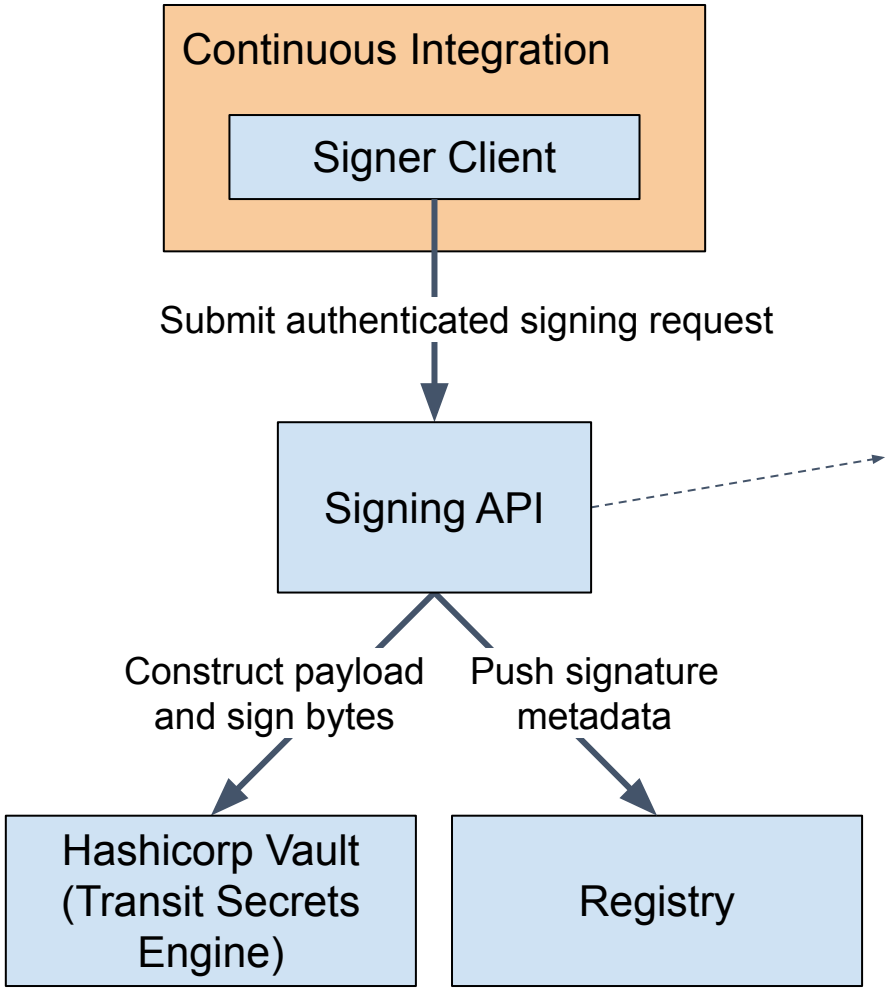

```
data {
    input          hmac-sha256:18c77d3538c17171ff34a4d6e5f63e93ed5874307fd302922fc3dd3d5d8dbe2b
    key_version    hmac-sha256:5b3469b7db63fb80eff7656a94609f5af41fe5a24290081a1326e0e30e4078ac
}
id                 bbe9faa7-0196-e989-940b-8636d9755b0e
mount_accessor     transit_b71e90b6
mount_type         transit
namespace {
    id  root
}
operation          update
path               crypto/k8s/sign/k8s_image-integrity_image-integrity-
                   signer_artifact_signing_1
```


Audit logs are opaque and hard to interpret.

# Signing Service: Least Privilege & Auditability



Mediating Vault & Registry access with the API:




## Signing API audit logs

```
ci_commit_branch   main
ci_commit_sha      f108225984de153086c77b2f95ff7cd5e71bc3a6
ci_job_id          246064299
ci_pipeline_id     14429574
ci_project_path    DataDog/image-integrity-testing

parsed_artifact {
    digest         sha256:01c1e3796cc29194f84f0ebf3b54a0e0d36f1178ec20a537b6031c46aa9b9bdf
    is_digested    true
    is_tagged      false
    registry       registry.ddbuild.io
    repository     image-integrity-test/hello-world
    valid          true
}
```

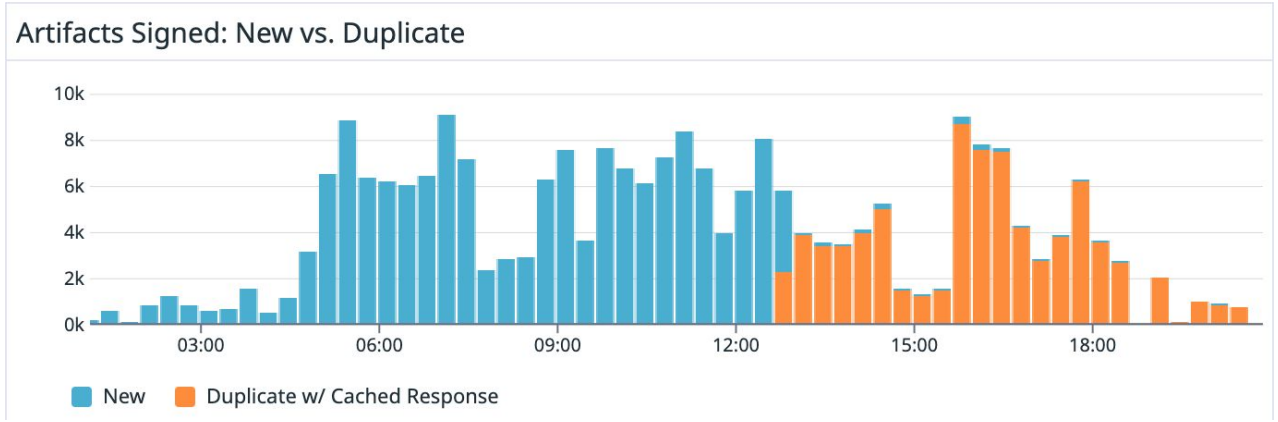
Audit logs have full request context.

Minimal trusted compute base for direct Vault access.

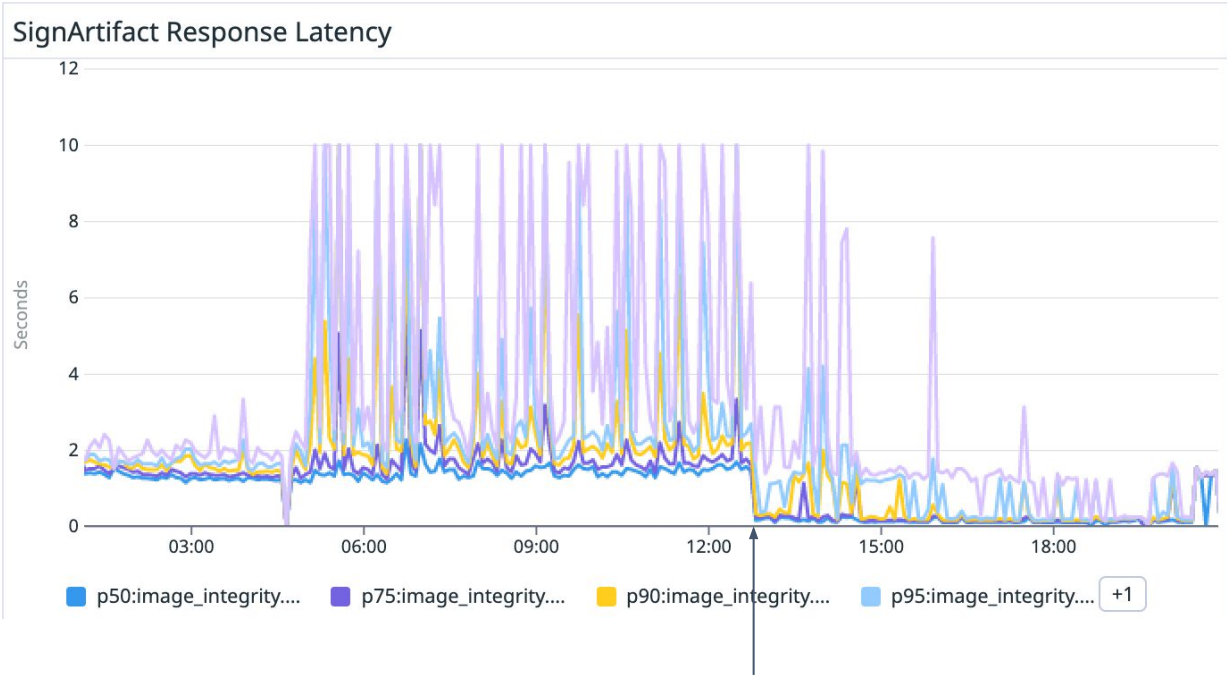
# Signing Service: Encapsulation



- Rarely need to update signing clients in all CI jobs
- Central control of load against registry & key management provider (Vault)
- Central key management transparent to clients

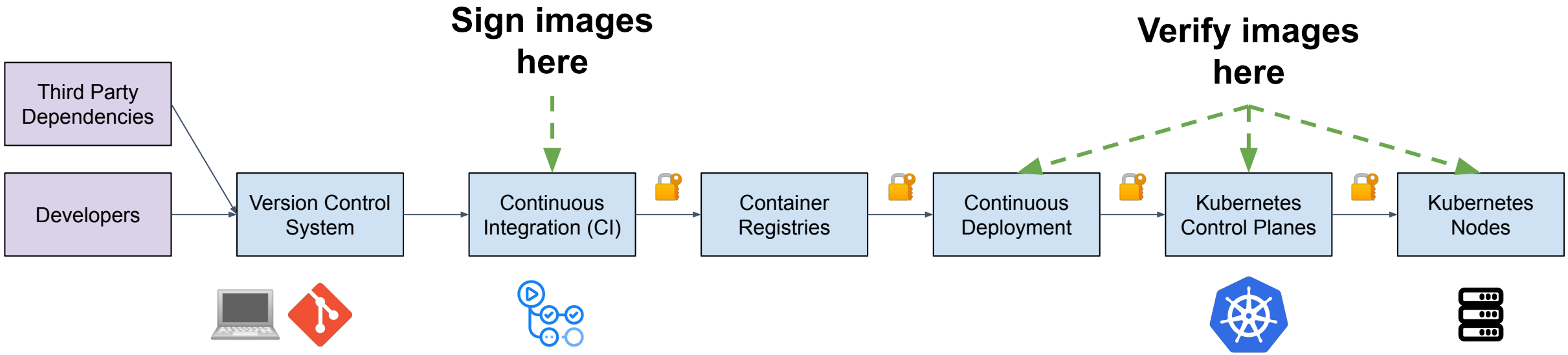



Reproducible builds → only ~3% of image signing requests are for new images.



# Where to Verify Signatures?



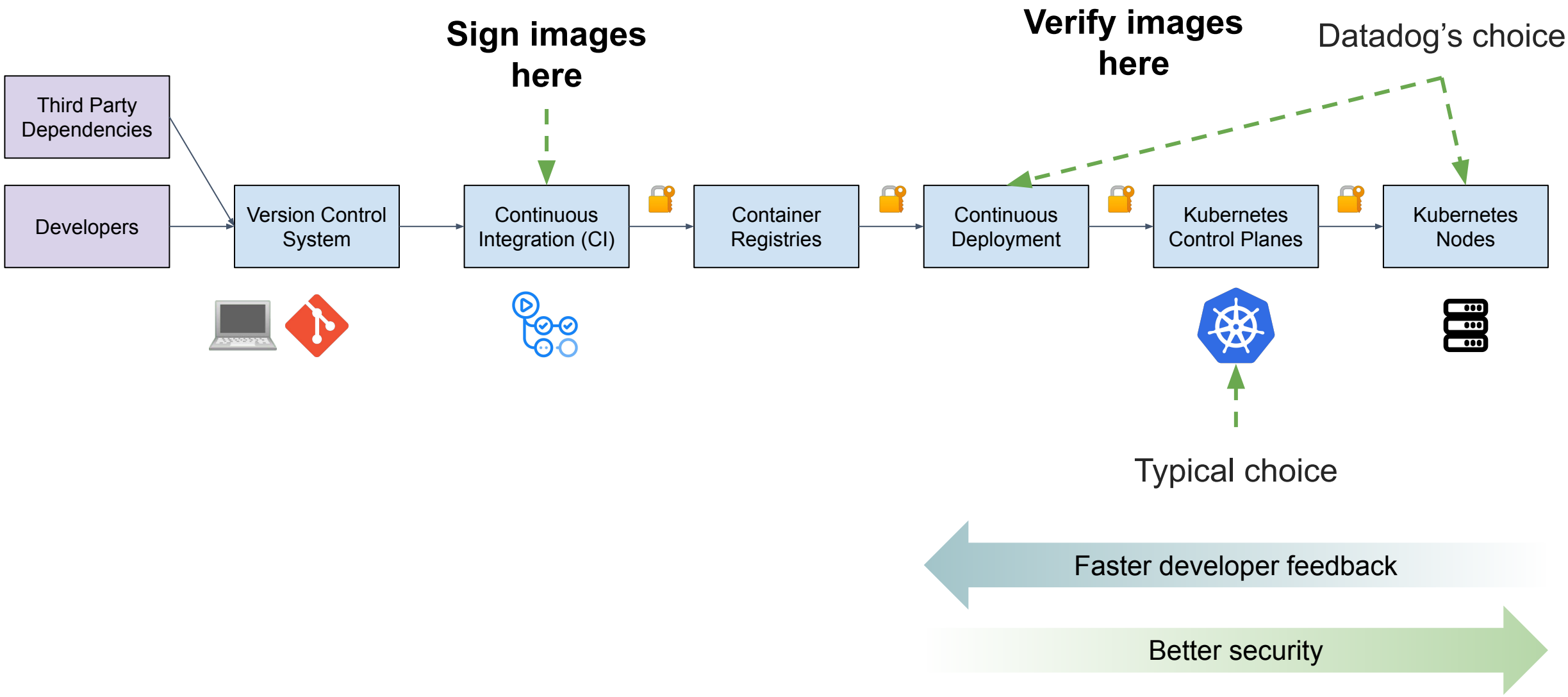



Faster developer feedback

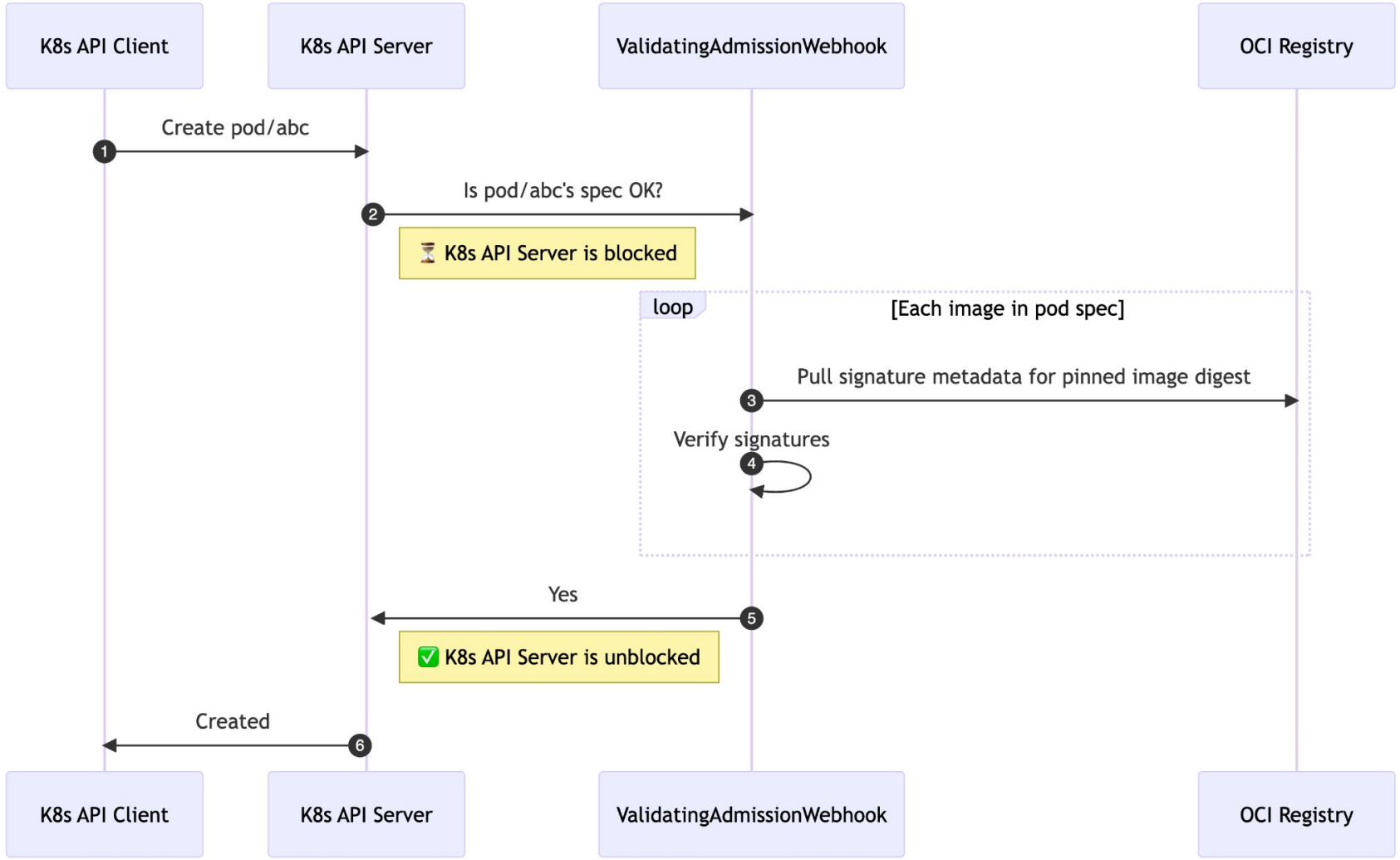
Better security

# Where to Verify Signatures?





# Validating Admission Webhooks




**Disadvantages**

Fetching registry metadata doesn't fit in webhook latency budget:

Goal: **p99 < 10ms**

Actual: **p50 > 200ms**

Registry would be in the control plane hot path (new dependency).

**Workaround**

Could use an `ImagePolicyWebhook` instead (for caching and retries), but relying too heavily on a central cache can lead to metastable failures[1].

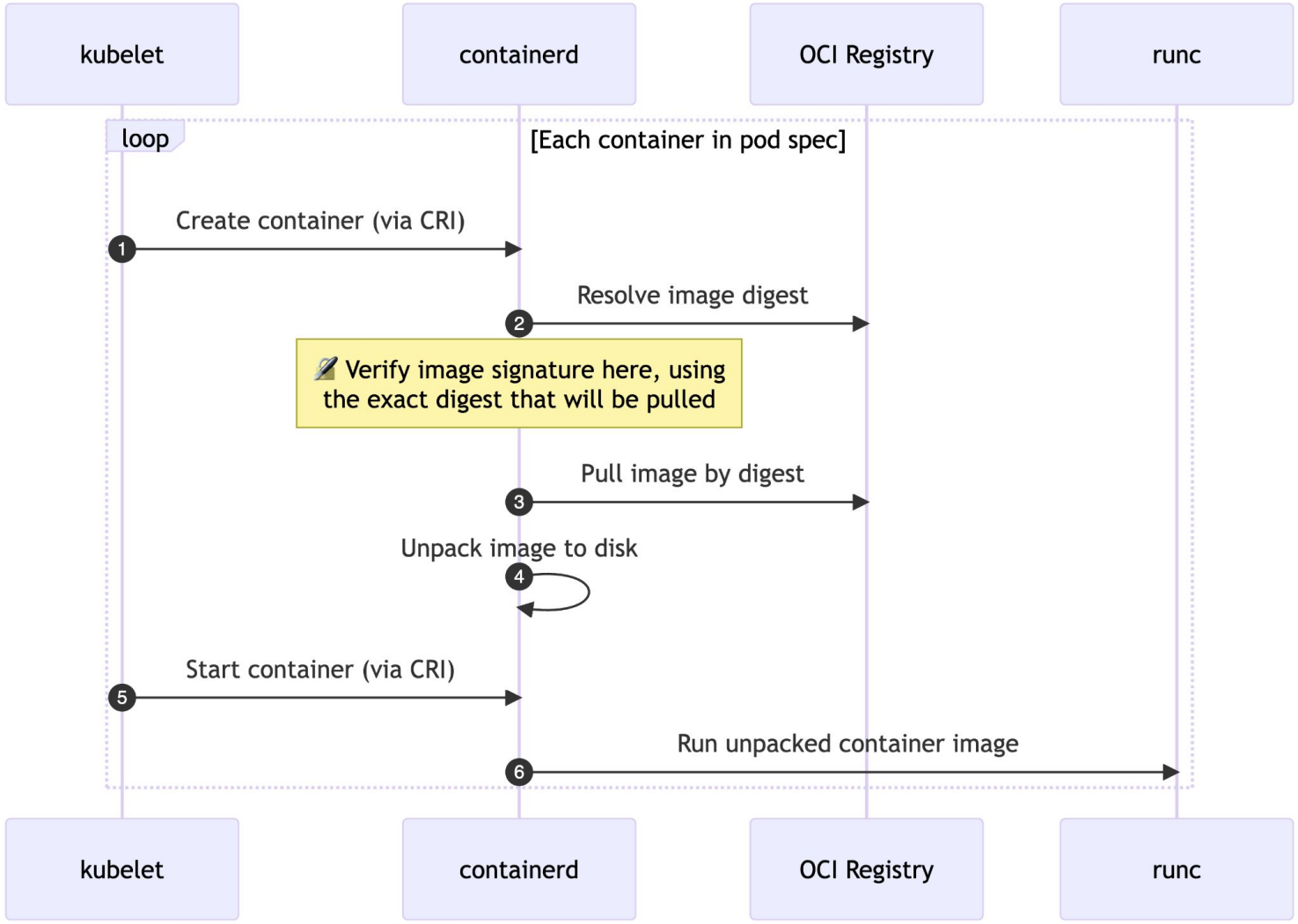
[1] https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s11-bronson.pdf

# Image Verification in containerd



Why verify in containerd?

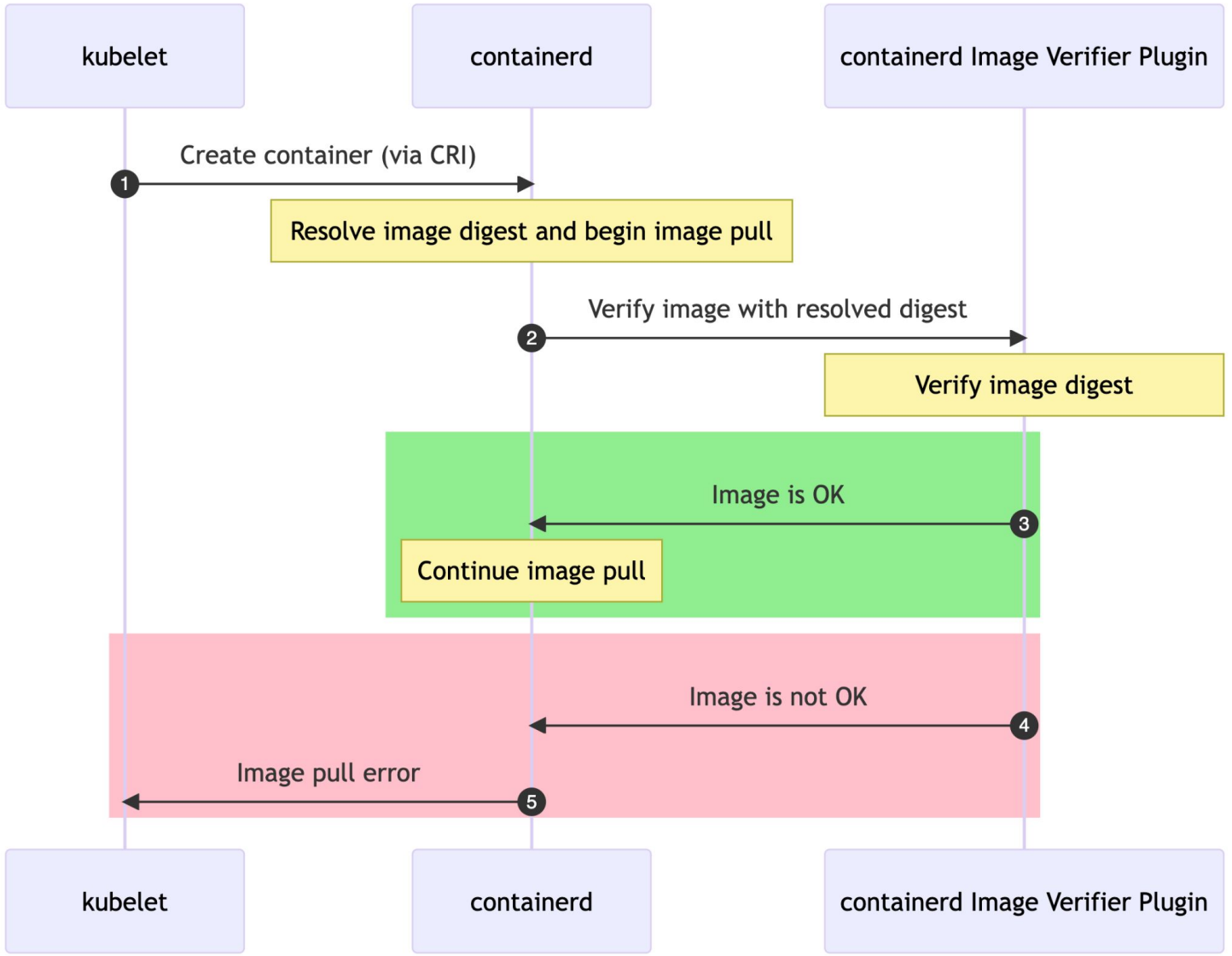
- Ultimately responsible for resolving an image digest
- Closest to runtime we can verify image signatures
- Minimal trusted compute base after signature verification



# Image Verification in containerd



- Adding a plugin system for image verification
- Benefit from kubelet's reliability features for pulling images
- Running a temporary fork & working with maintainers to get the image verification plugin system into containerd 2.0. See [containerd#6691](containerd#6691).



# Developer Perspective

KubeCon CloudNativeCon Europe 2023

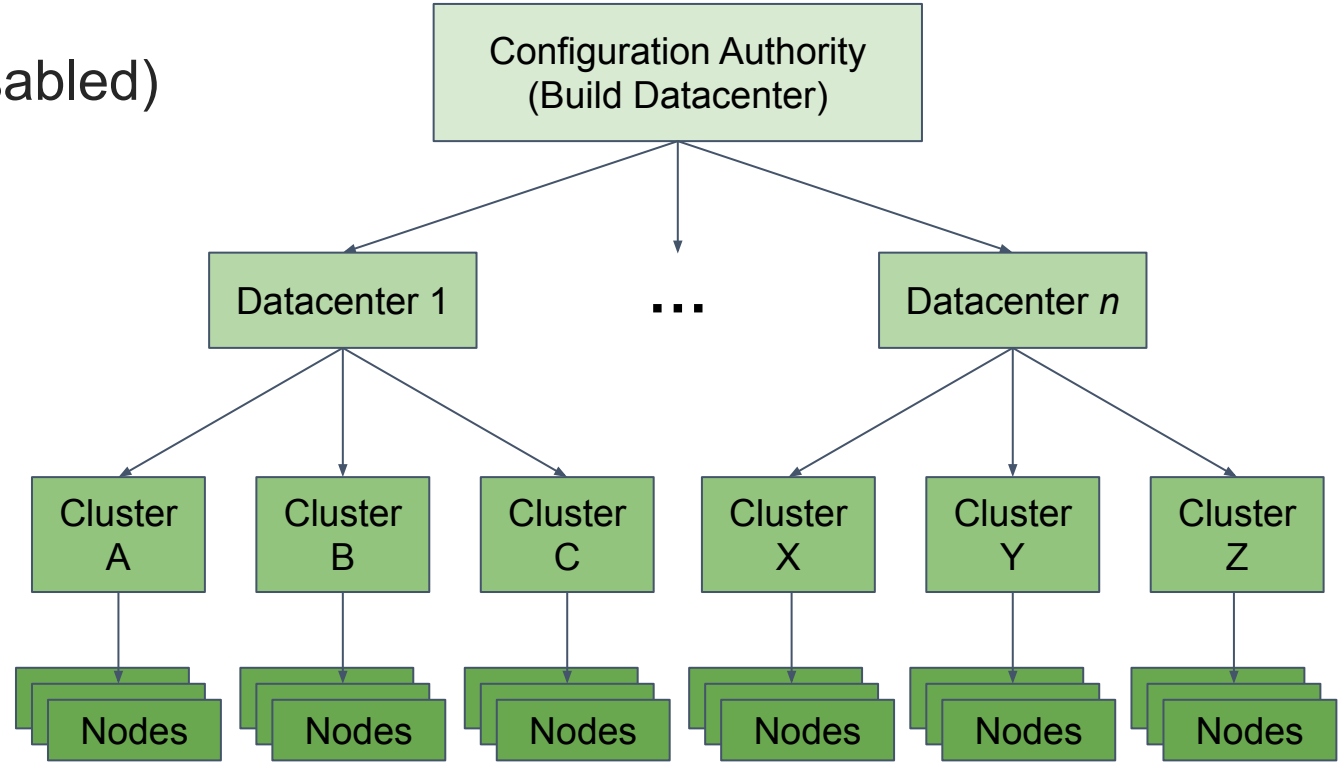
```
$ kubectl get pods
NAME                          READY   STATUS             RESTARTS   AGE
example-pod-images-signed     1/1     Running            0          50s
example-pod-images-unsigned   0/1     ImagePullBackOff   0          48s


$ kubectl get events
...
2m   Warning   Failed    pod/example-pod-images-unsigned    Failed to pull image ... image verifier blocked
pull of registry.example.com/example-signed with digest
sha256:43292eabb01bf6c9dacf41b249248bc445e16698a91358f4dda6ef67629818f2 for reason: ...
(Need help? https://datadoghq.atlassian.net/l/cp/Fn2mj3vS)
...
```

# Distributing Verifier Config



- The verifier plugin on each node needs:
  - Public keys
  - Verification mode (audit, block, or disabled)
  - Image digest revocation list

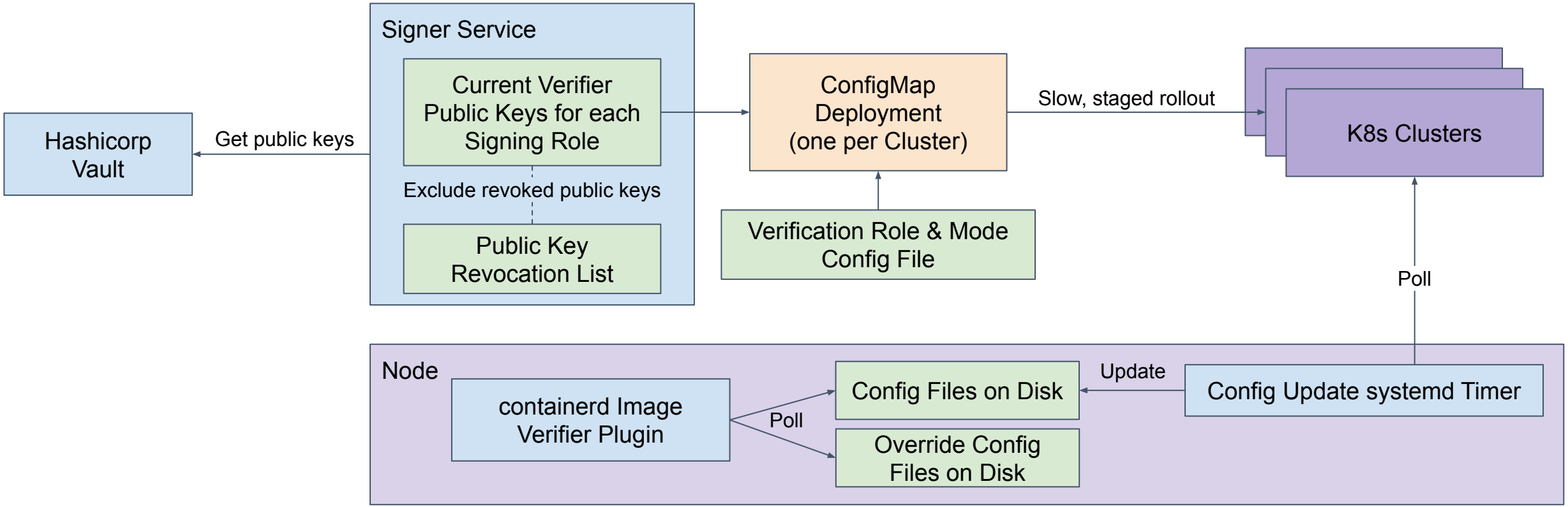
- Requirements:
  - No new node dependencies
  - Slow, staged rollout of config
  - Multiple fallback mechanisms



# Distributing Public Keys & Mode



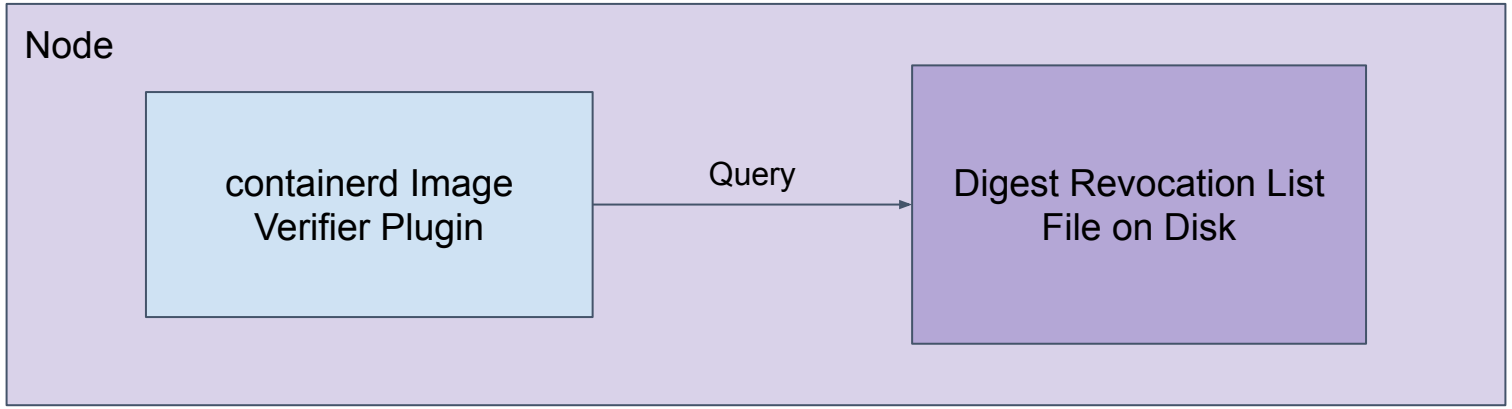
- Layered approach:
  - Defaults baked into node image
  - Periodic dynamic updates pulled from a `ConfigMap` in each cluster
  - Override config layer on disk



# Distributing Image Revocation List



- Baked into the node image
- Updated via node image rollout automation
- Dynamic updates wouldn’t be useful since containerd caches images
  - Prioritize draining nodes that have run a revoked image



# Challenges & Recommendations



- Can’t easily configure verification mode by Kubernetes namespaces
  - Harder to turn on blocking mode in large multi-tenant clusters
  - Dedicated clusters for sensitive workloads → easier to sign all images and enable blocking

# Challenges & Recommendations



- Integrating image signing into many CI configurations is a lot of work, even with a thin client
  - Monorepos or consistent build tooling (e.g. Bazel) make it easier
  - Roll out audit mode verification before you're done rolling out signing

# Challenges & Recommendations



- Node-level image verification is uncharted territory
  - We believe the reliability properties are worth the extra effort

# Takeaways



1. Evaluate image signing as a service for security and scalability.

2. Think about the reliability risks of Kubernetes admission webhooks and the advantages of node-level image verification.

3. Join the conversation and give feedback on the design at https://github.com/containerd/containerd/issues/6691


KubeCon
CloudNativeCon
Europe 2023
ethan.lowman@datadoghq.com