CLOUDNATIVE SECURITYCON NORTH AMERICA 2023

Container Patching

Making It Less Gross Than The Seattle Gum Wall

Greg Castle

@mrgcastle, @gregcastle@infosec.exchange

Weston Panther

Simple SLO View Of Patching

0 days Scanner detects

15 days Maintainer patches

30 days Production patched

Severity	FedRAMP Targets
CRITICAL/ HIGH	30 days
Medium	90 days
Low	180 days

A Trip Down Empathy Lane

2 weeks 🛡️ …Bi-weekly cluster scan…

2 weeks 🛡️ Hey web team, webfrontend is missing 2 CRITICAL patches

3 weeks 🛡️ Friendly ping?

3 weeks 🕸️ Not our code, maybe django base container?

3 weeks 🛡️ Django container team, can you patch?

4 weeks 🐍 These vulns are in perl, we don’t even use perl, do we need to patch?

A Trip Down Empathy Lane

4 weeks 🛡️ Yes, or better yet, remove perl.

OUT OF FedRAMP/PCI SLO

5 weeks 🐍 Patched 🎉 acme-django:v2.1.1

5 weeks 🛡️ Hey web team, rebuild with acme-django:v2.1.1

6 weeks 🕸️ Done!

7 weeks 🛡️ Still running the old version?

A Trip Down Empathy Lane

8 weeks 🕸 Forgot to update the K8s manifest. Done!

9 weeks 🛡 …Still no? Also there's three new HIGH vulns, but let's get this done first.

10 weeks 🕸 Had to soak in QA first, updated for prod rollout.

11 weeks 🛡 Fixed! Who else runs django apps..?

Why It’s Gross

Humans at every step

Which layer needs patching?

No inventory

Patching unused code

Vulns faster than patches

= Slow, incomplete, unscalable patching

Is the majority of the industry doing better than this today?

88% of respondents:

“Challenging to ensure containerized applications are free from vulnerabilities”

https://www.slim.ai/blog/container-report-2022/

GKE Container Patching Case Study

Enforcement Points

Prevent: minimal containers

Detect: scanning capability/coverage

Fix: ownership, dependencies, release

Monitor: dashboards, alerting, escalations

What Do We Know Anyway?

Patching for 1000s of containers across GKE, Anthos and adjacent products

But...our environment constraints help a lot:

- Mandatory use of compiled language
- Mandatory container repo
- Mandatory base images
- Control over code/config pre-submit
- Control over release

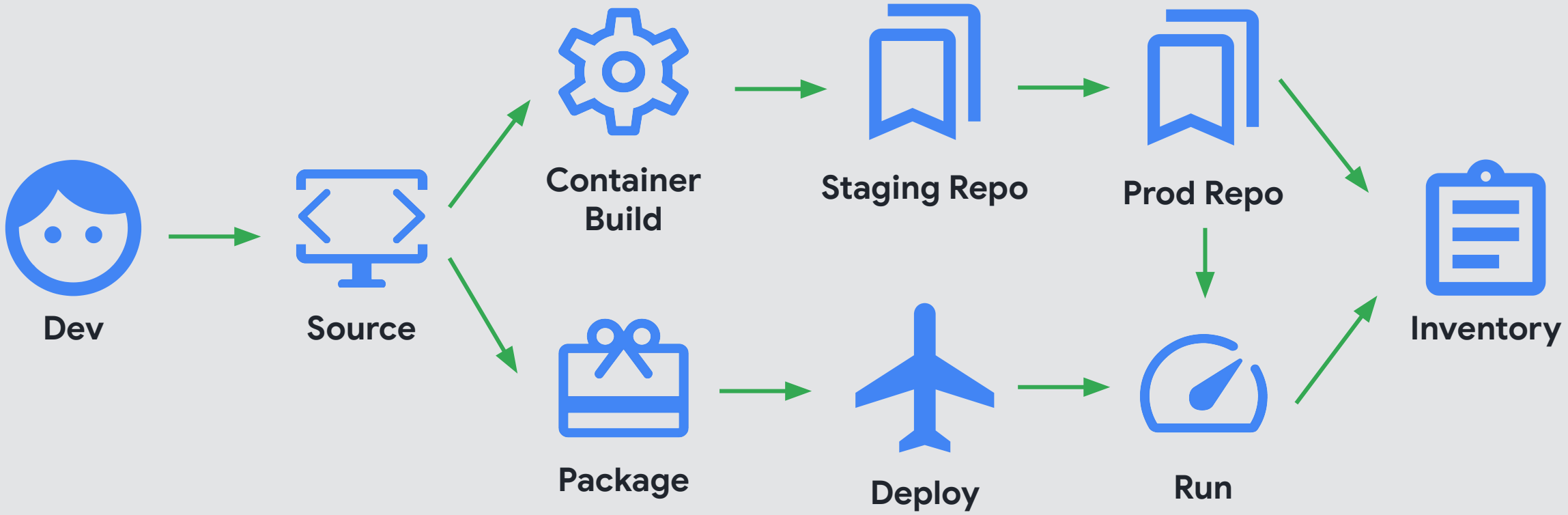
What Containers?

~~Vendor/MSP containers~~

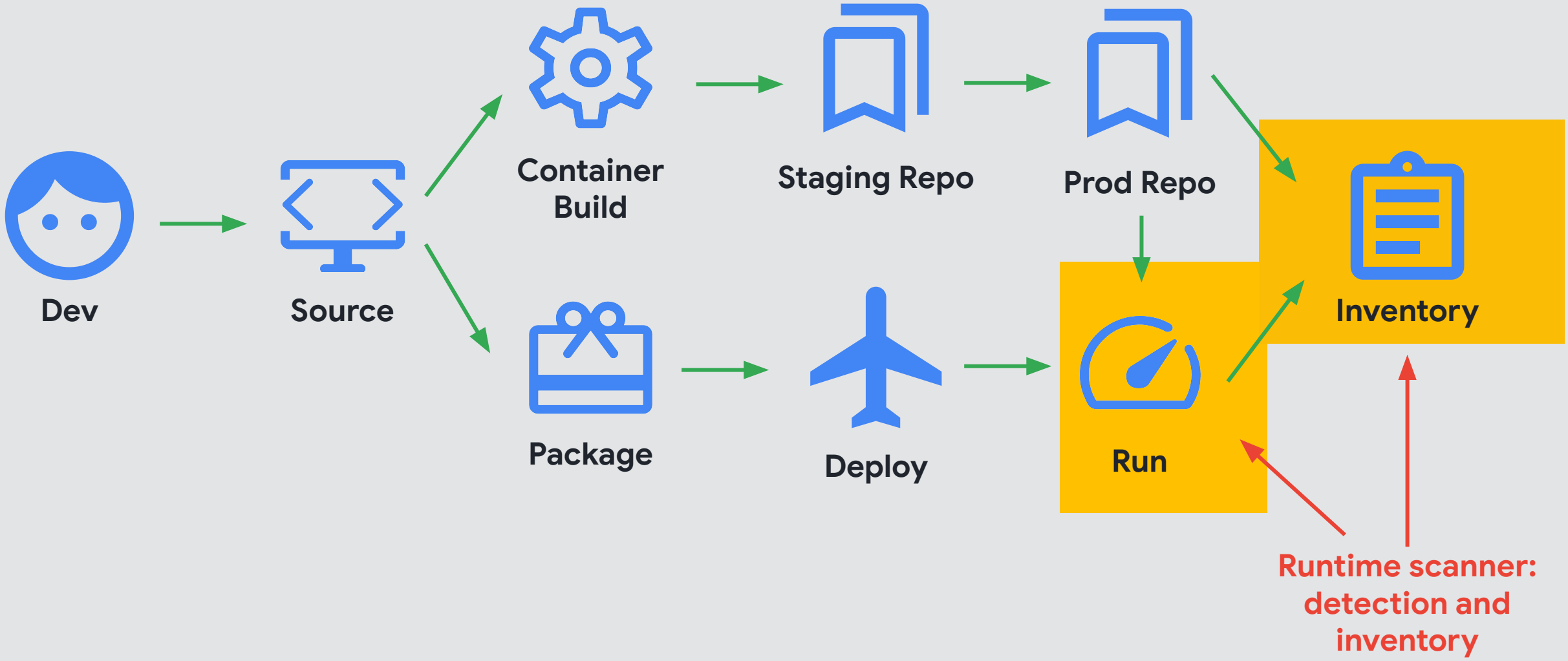
Containers you rebuild

K8s manifests you update

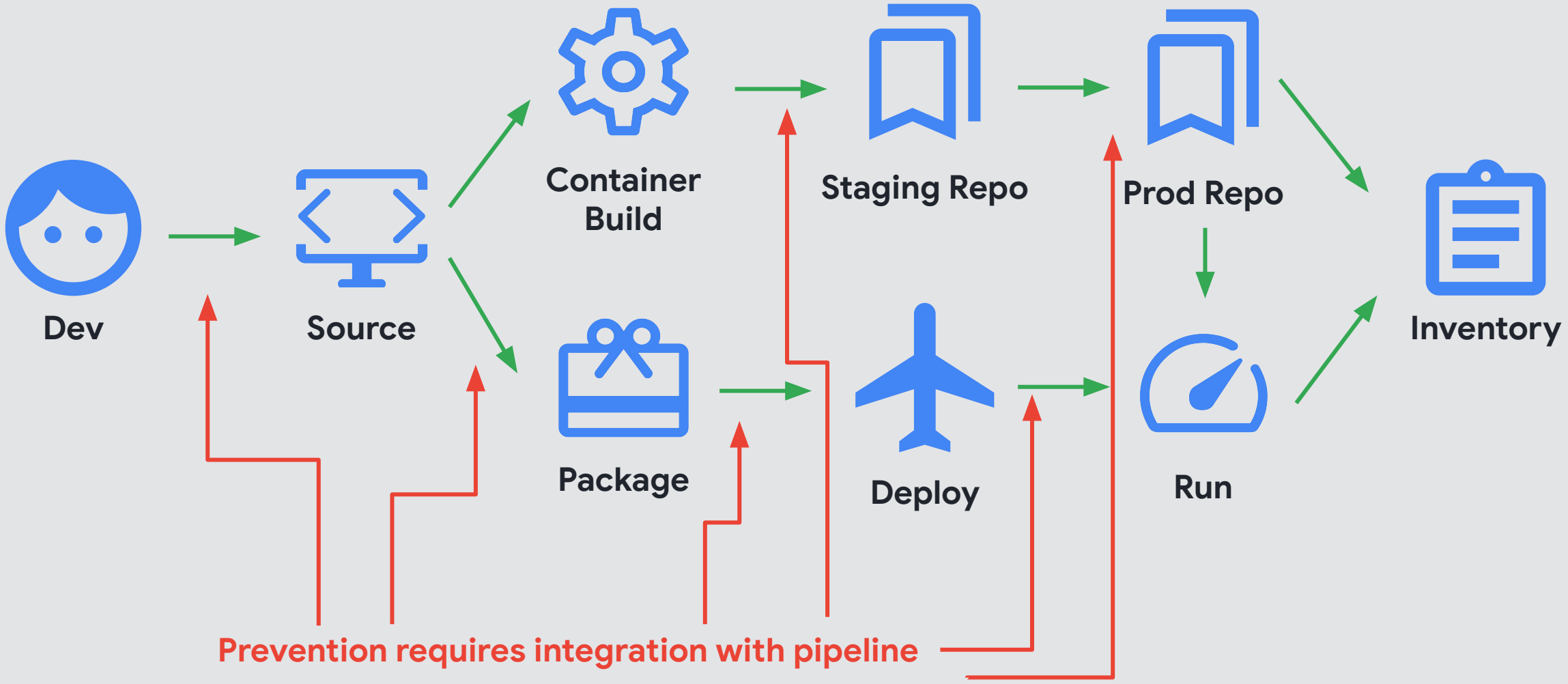
Container/K8s Delivery Pipeline



Good Start: Runtime Detection



Better: Prevention Complementing Detection



Prevent

Prevent: Problems

So many containers

With so many dependencies

Meeting SLO is hard without reducing volume

Prevent: Strategy

- Standardize base containers
- Minimal containers: Less code, less vulns, less patching
- Remove unused code: separate build and runtime images
- Two approaches:
  - Start small: Scratch, Distroless, Wolfi/Chainguard Images
  - Slim down: SlimToolkit
- Challenge: apply consistently everywhere

Our Solution

- Standardize on Distroless
  - Just enough to run golang binaries
- All containers in a single repository
  - Inventory
  - Availability

Our Solution

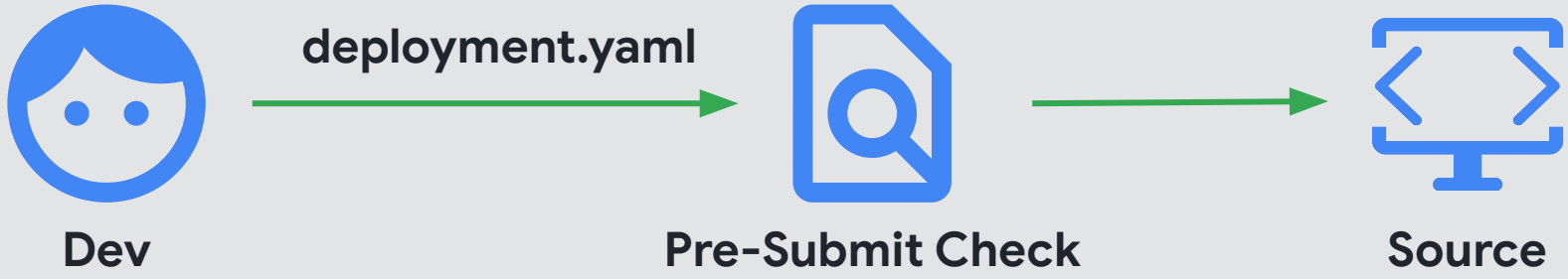
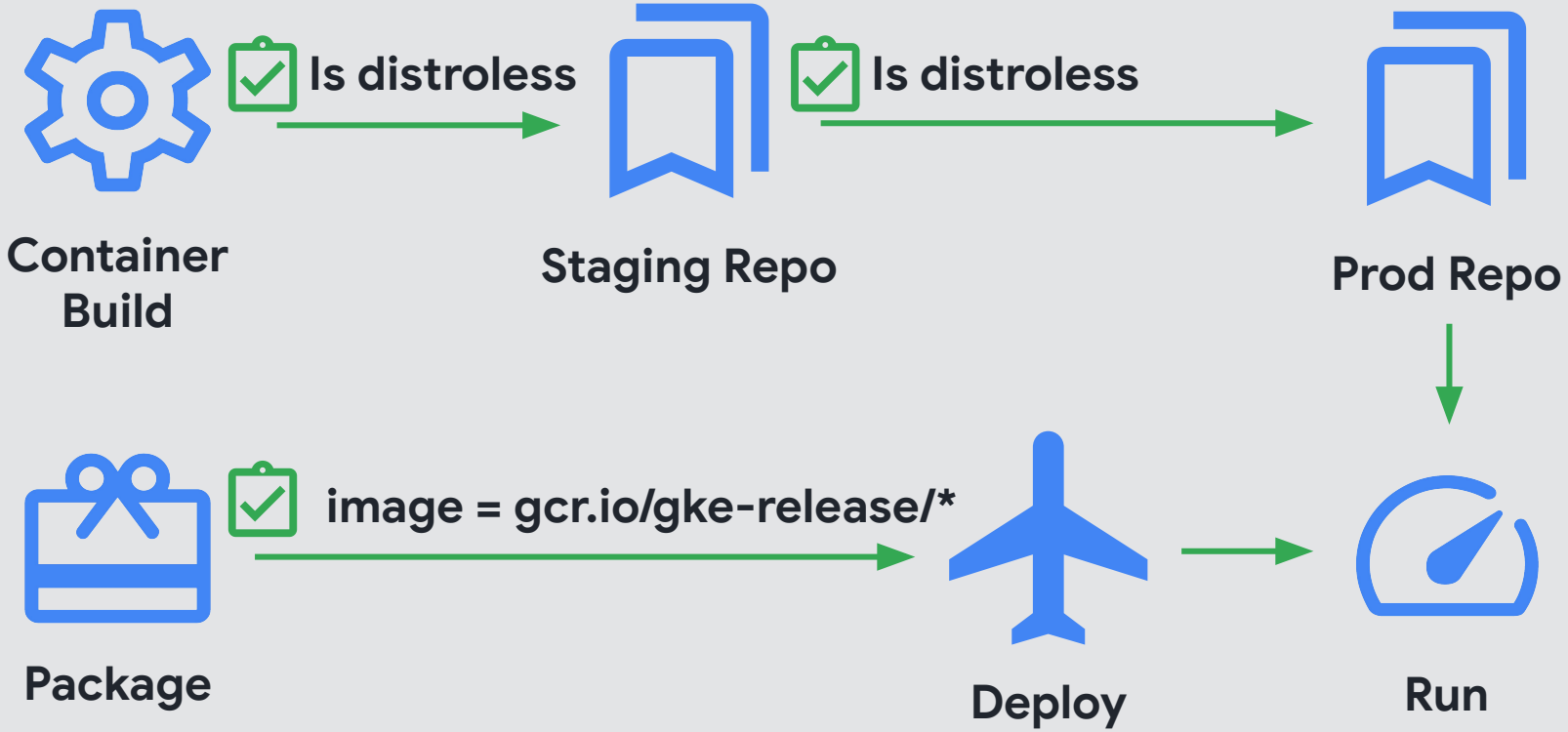


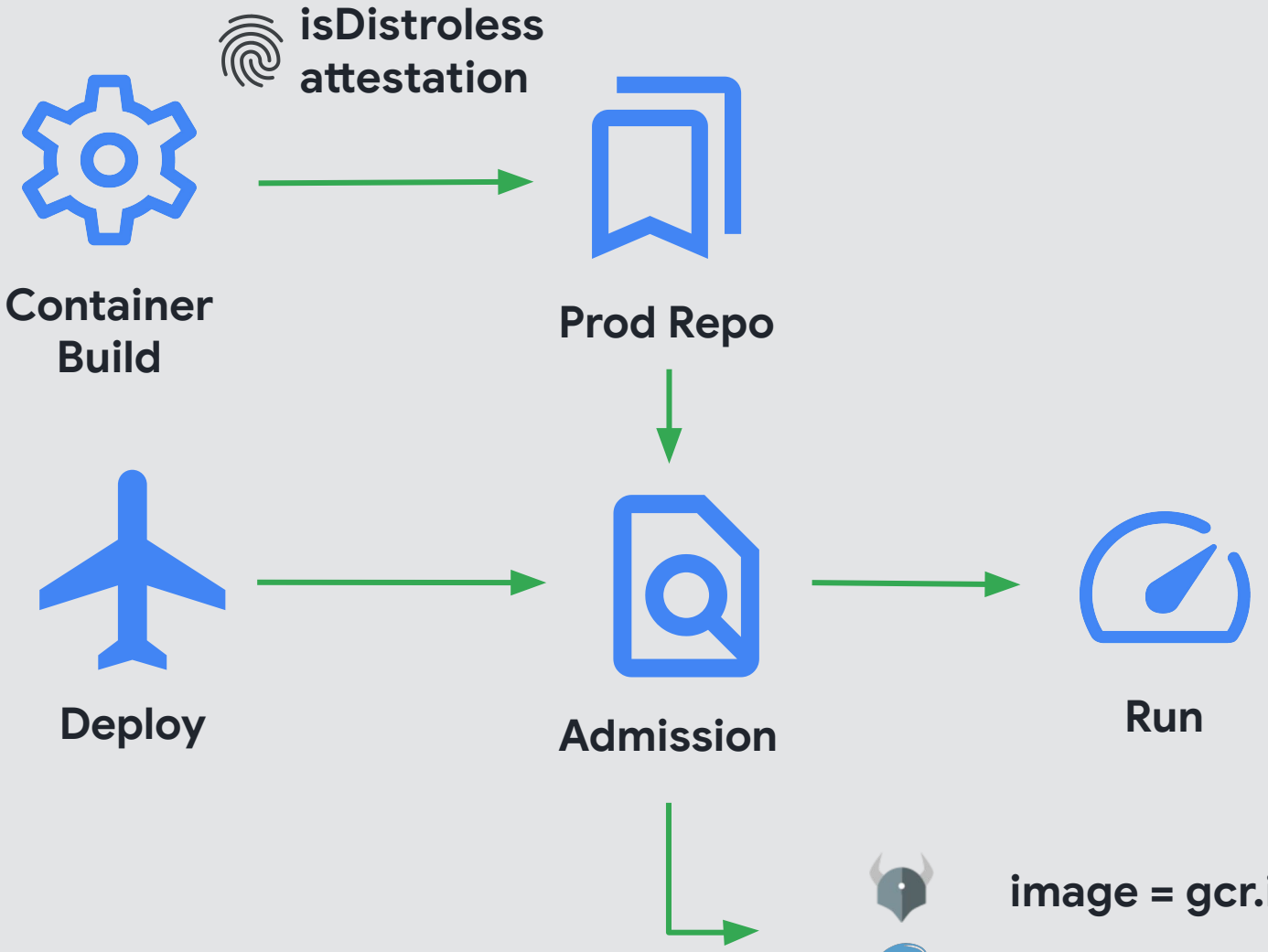
image = gcr.io/gke-release/*

Container /etc/os-release
HOME_URL=https://github.com/
GoogleContainerTools/distroless

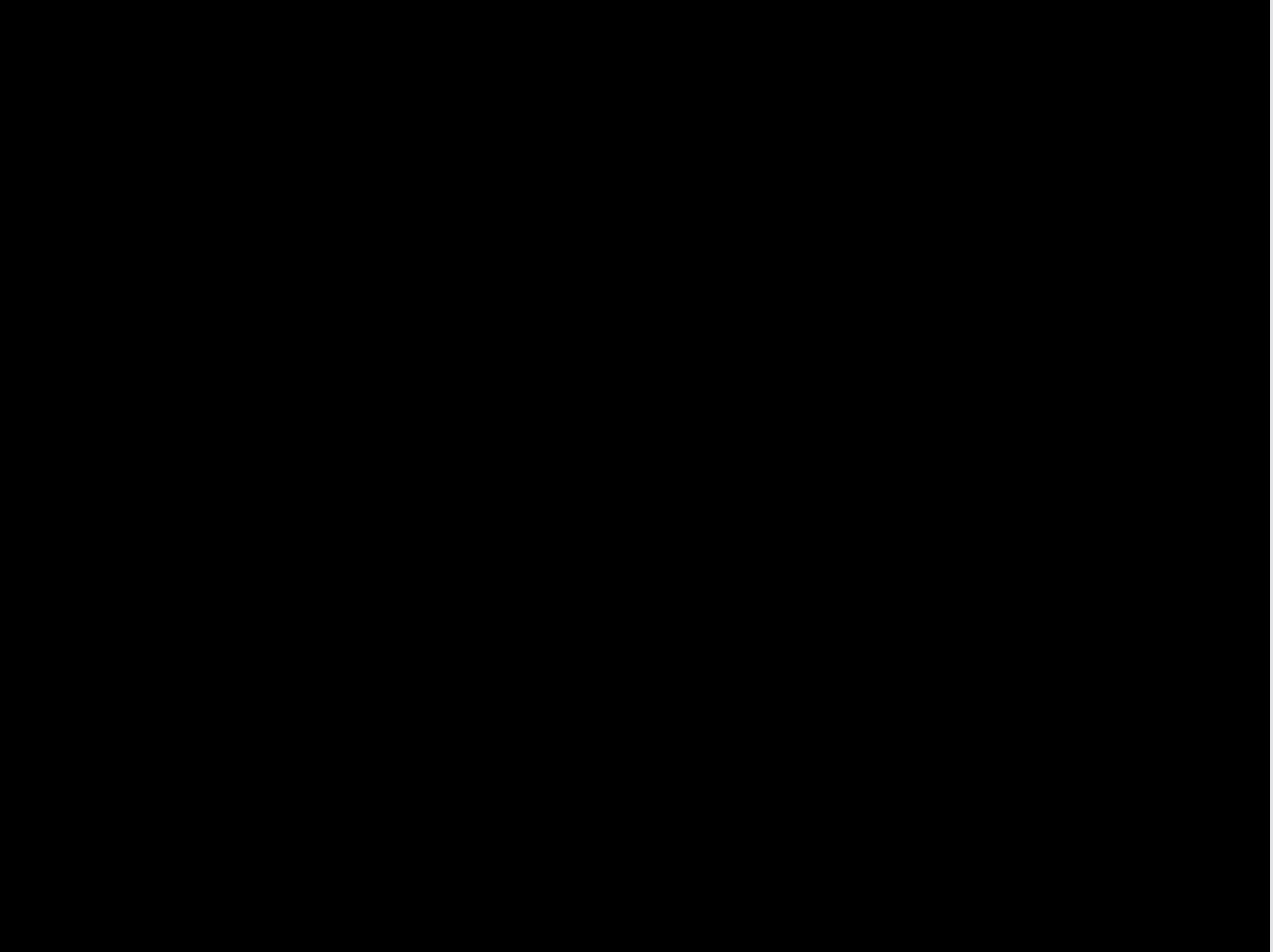
Alternatives



Alternatives: Admission



On GKE: Use Binary Authorization



CLOUDNATIVE SECURITYCON NORTH AMERICA 2023

Demo: Admission

Prevent: Summary

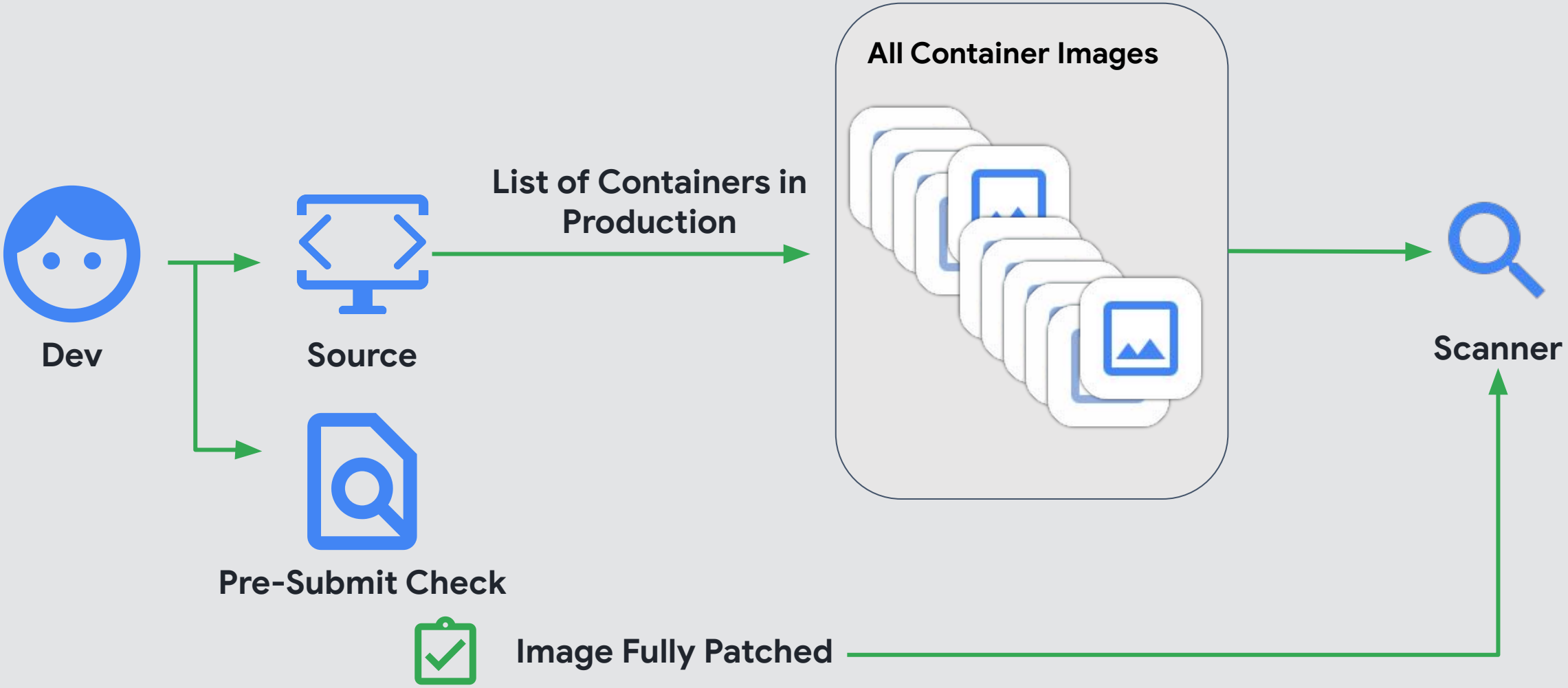
- Identify and use enforcement points
- Standardize on patchable base containers
- Standardize on container registries for inventory

Detect

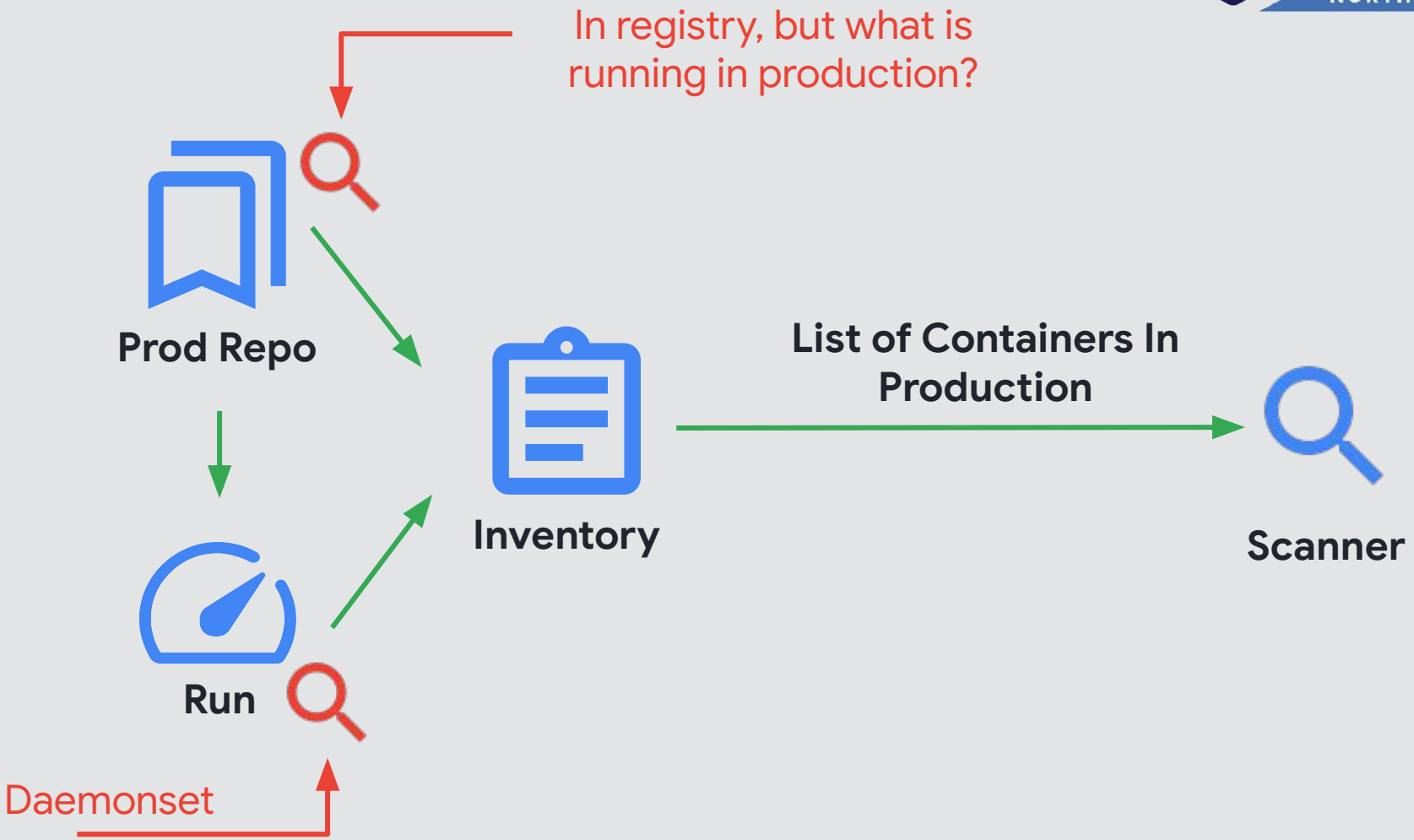
Detect: Problems

- Which containers to scan?
- Which scanner?
  - Different coverage
  - Different vuln sources
  - Duplicate handling
  - Filtering noise
- Which layer has the vuln?

Which Container? Our Solution



Which Container? Alternatives



Which Scanner?

Language Pack Scanning

Scan programs in your container:

- Rust Cargo.lock
- Python egg files
- Go binaries / go.mod
- etc.

SBOM Consumption VEX Support

Scanners are starting to support SBOMs
Filter out remediated vulnerabilities based on VEX

Supplemental CVE Sources

More vulns from more places

- OS vendor feeds
- Github Advisories Database
- Language-specific DBs (vuln.go.dev)

Which Scanner?

Base Image Detection

Try to determine the base images:

- From metadata in image manifest
- From the Dockerfile

Reachability Analysis

Try to figure out if the code is actually in use:

- Typically uses source
- Can use symbol table in a binary

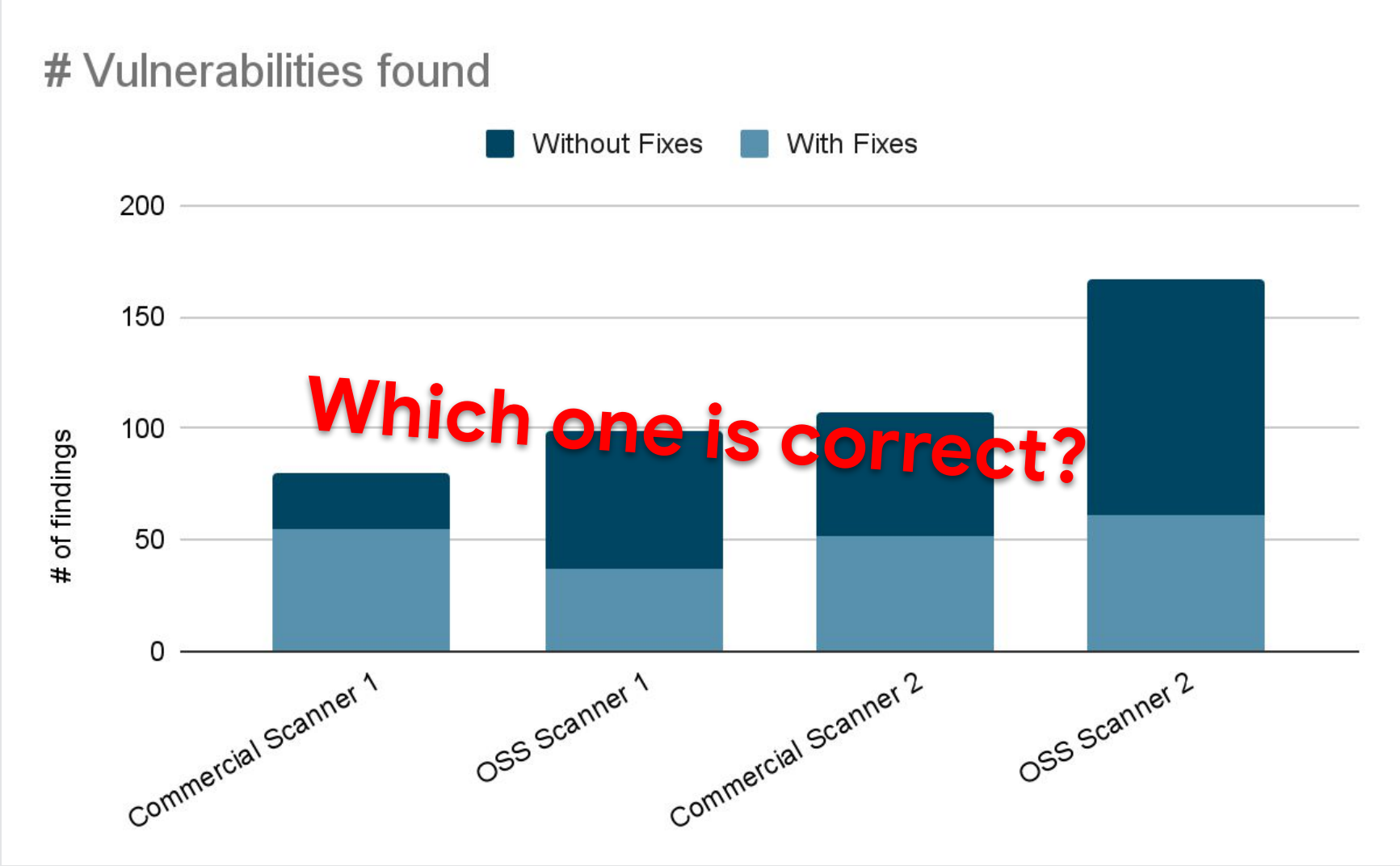
Additional Scans

Scan all the things:

- CIS benchmarks
- Hardcoded keys
- Misconfigurations (root user, host volume mounts, etc.)

False Positives vs. Coverage

- Vulnerable module (golang.org/x/crypto/ssh)
- Built with old golang version (1.18.1)
- On old debian base (buster-20210208)



Which Scanner? Our Solution

Public containers: probably “more than one”

Identify gaps and false positives

See what our customers see

Detect: Noise

- Codepath is unused
- Recent CVEs with no patch

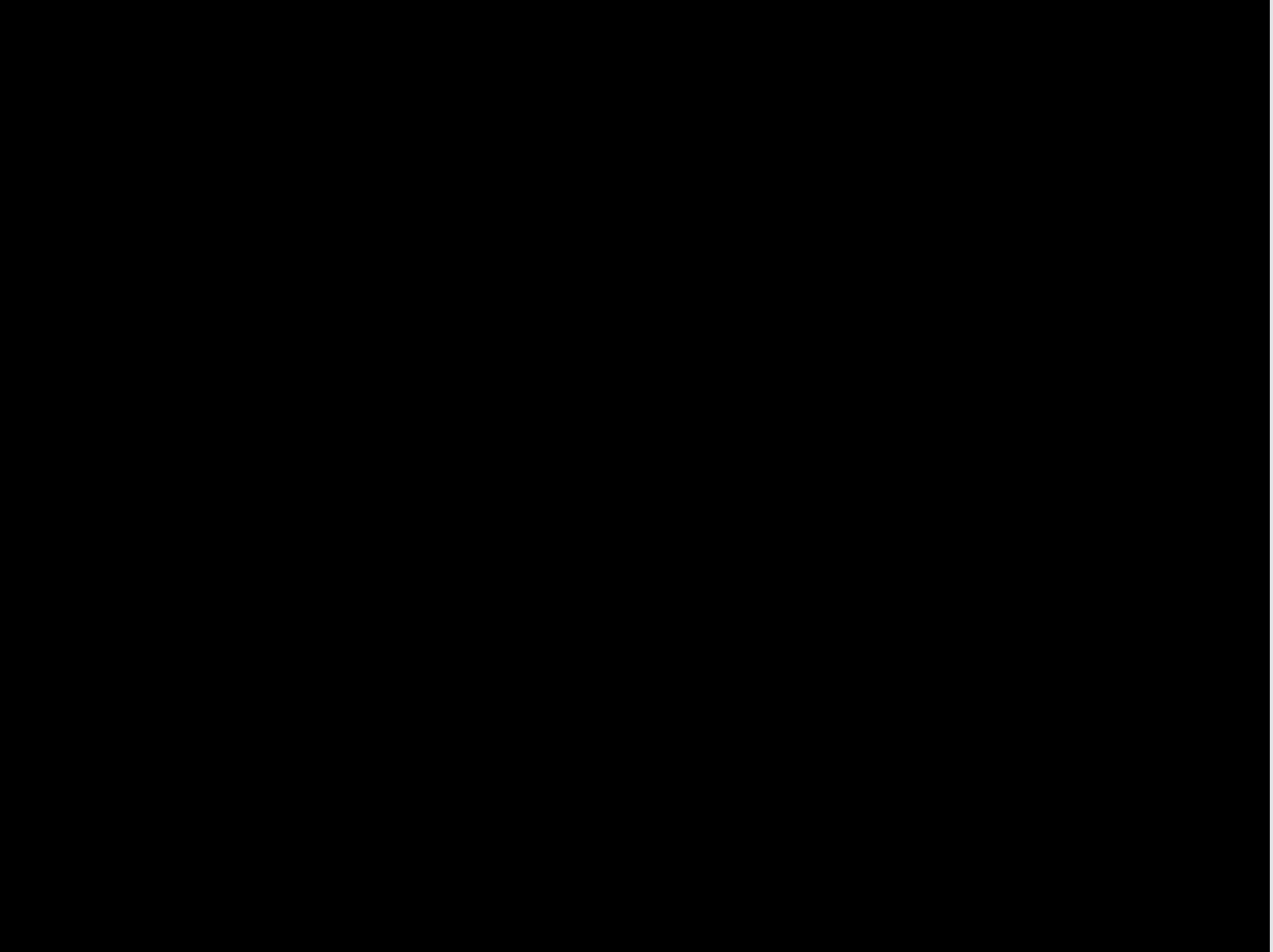
User Control

- CVEs that will never be patched (debian CVE-2004-0971, CVE-2005-2541, CVE-2010-4756)
- Ancient low priority vulns without patches (debian CVE-2011-4116, CVE-2016-2781)
- OS vendor has a lower rating than NVD (debian CVE-2022-37434)
- CVE is for a different architecture (golang CVE-2021-38297)
- CVEs that are clearly overrated (CVE-2020-29363: 9.8 down to 7.5)

Scanner Control

Noise: Golang Specific

- Problem: *all vulns in whole minor go version or module* attributed to a container if it was built with that version
- Solution: govulncheck can be integrated with scanners to report only reachable vulnerabilities
- https://go.dev/blog/vuln



CLOUDNATIVE SECURITYCON NORTH AMERICA 2023

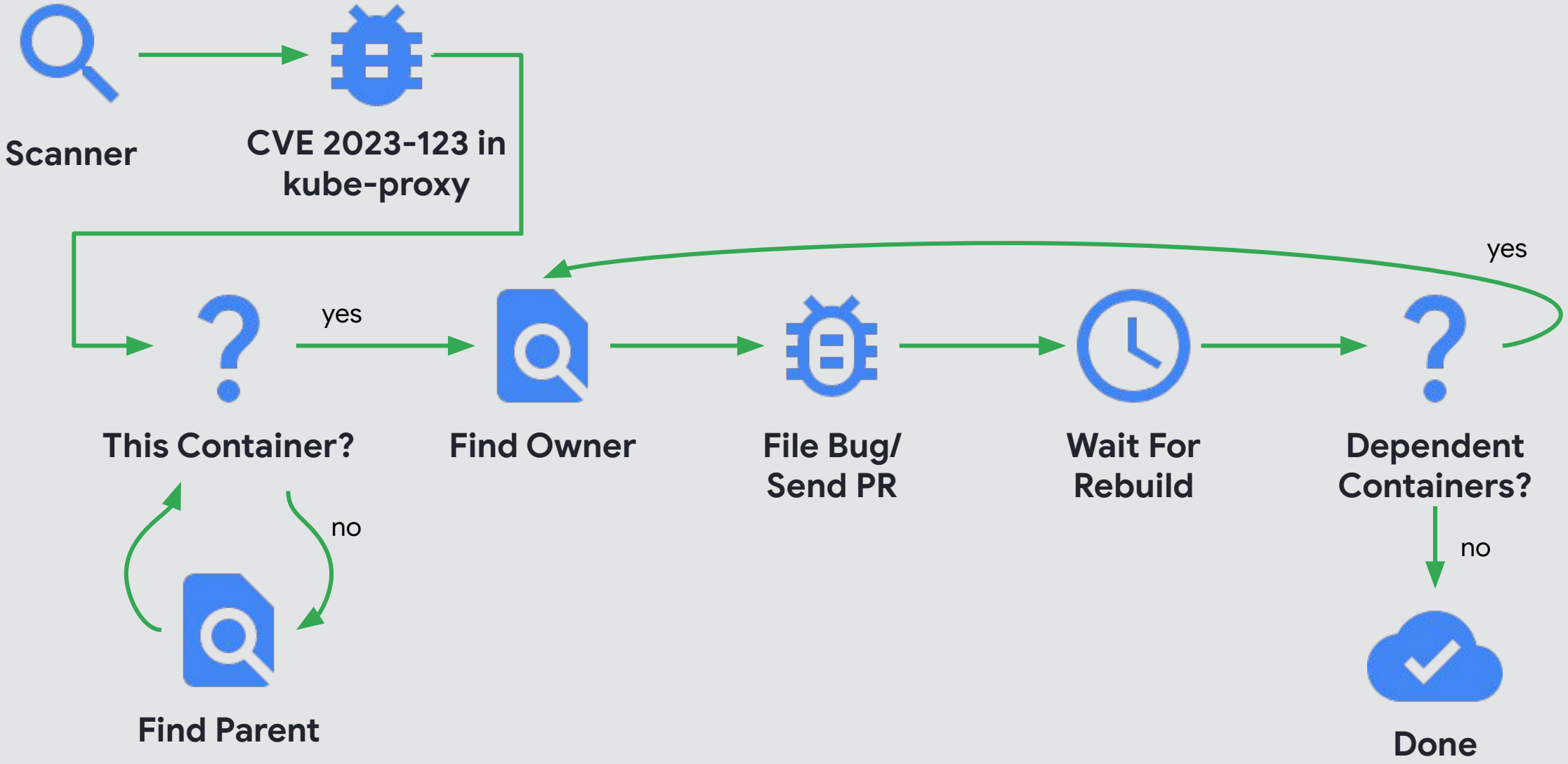
Demo: govulncheck

Detect: Summary

- Take advantage of new advances in coverage
- Look to your scanner vendor to help with noise
- Use silence/ignore where it fits threat model

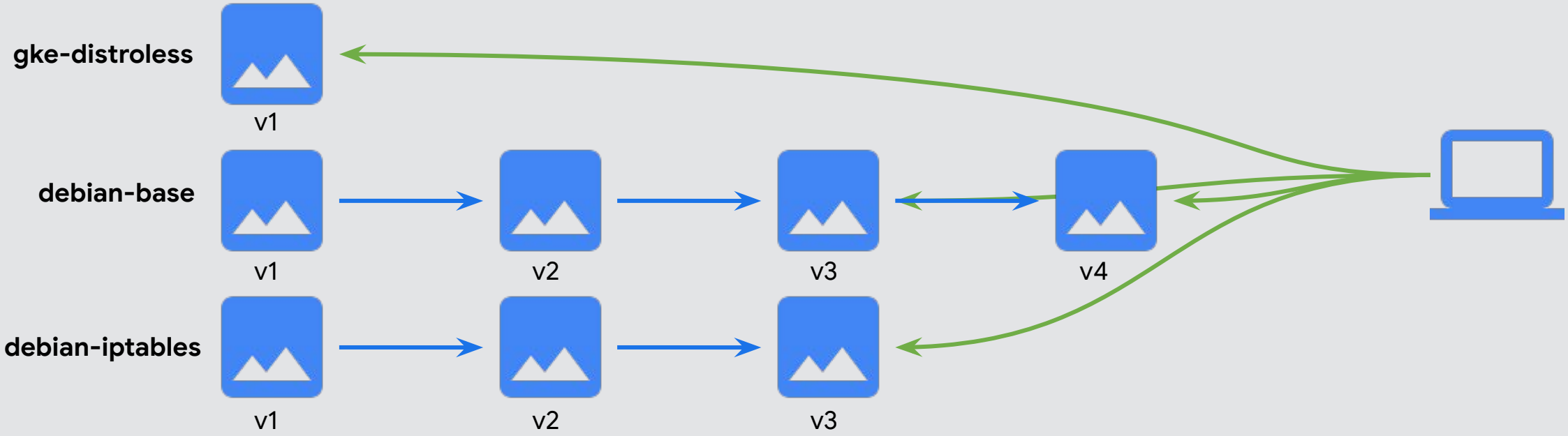
Fix

Problems: Multi-layer Complex Process

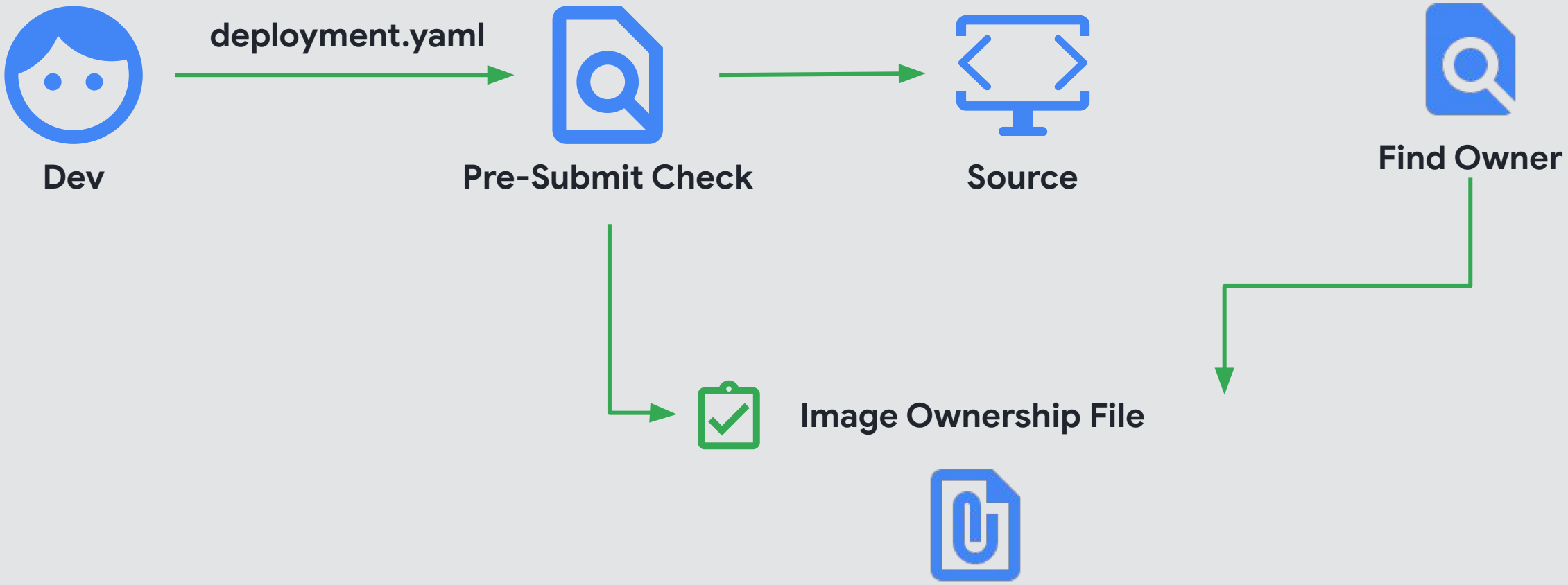


Our Solution: Base Images

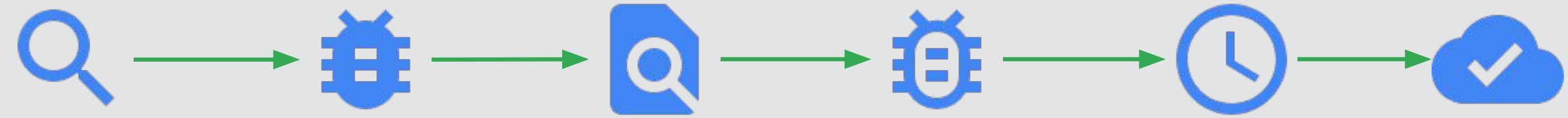
1. Scan the latest base images
2. If fixable vulns, rebuild
3. Repeat for eternity



Our Solution: Ownership



Our Solution: Simplified Process



Summary: Fix

- Track container parent-child relationships
- Automate patching base images
- Comprehensive inventory and ownership
- Use existing ticket systems to track

Monitor

Monitor: Problems

What gets measured...

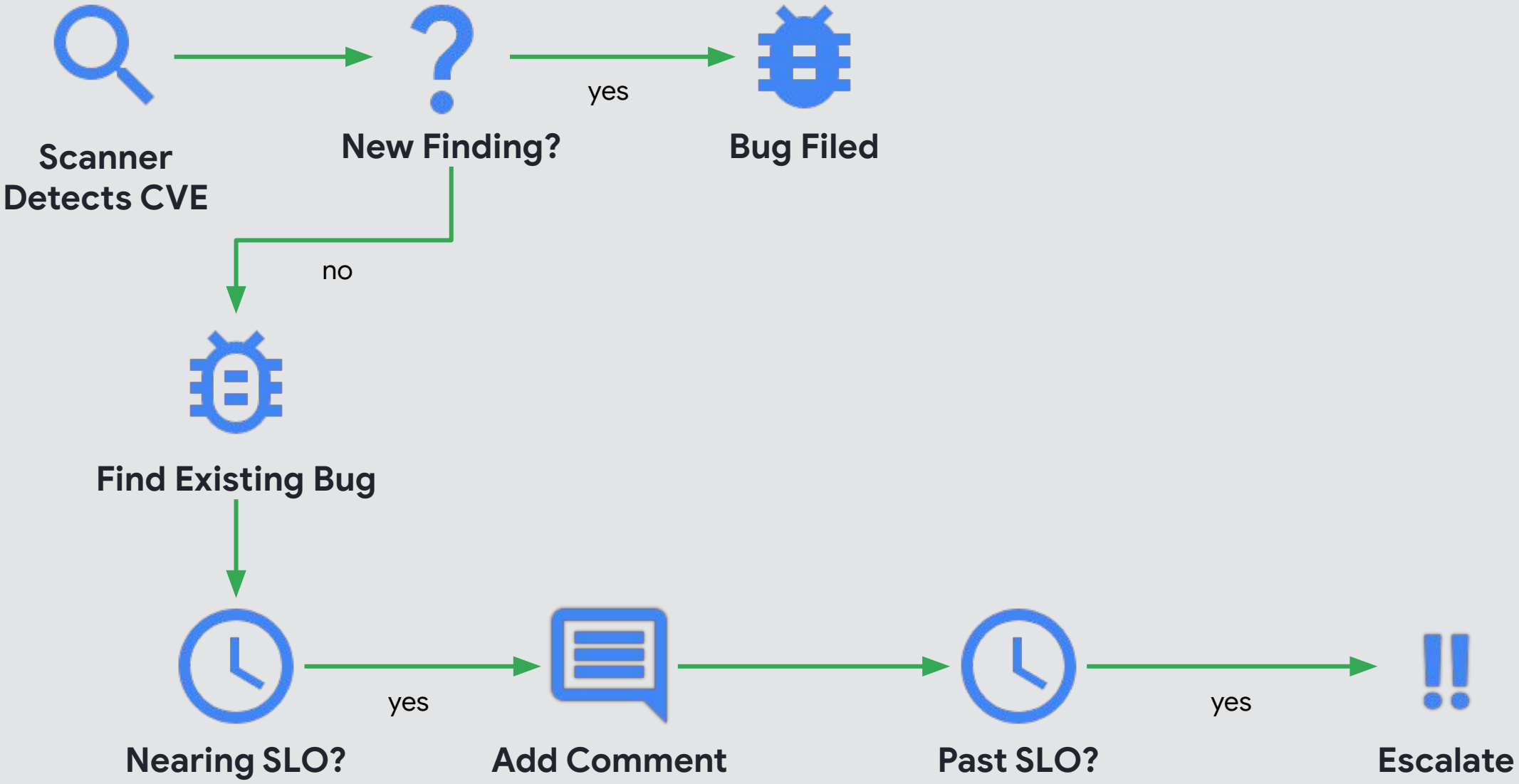
Is CVE-123 patched? Has it rolled out everywhere?

Which containers have the CVE? Which applications use this container?

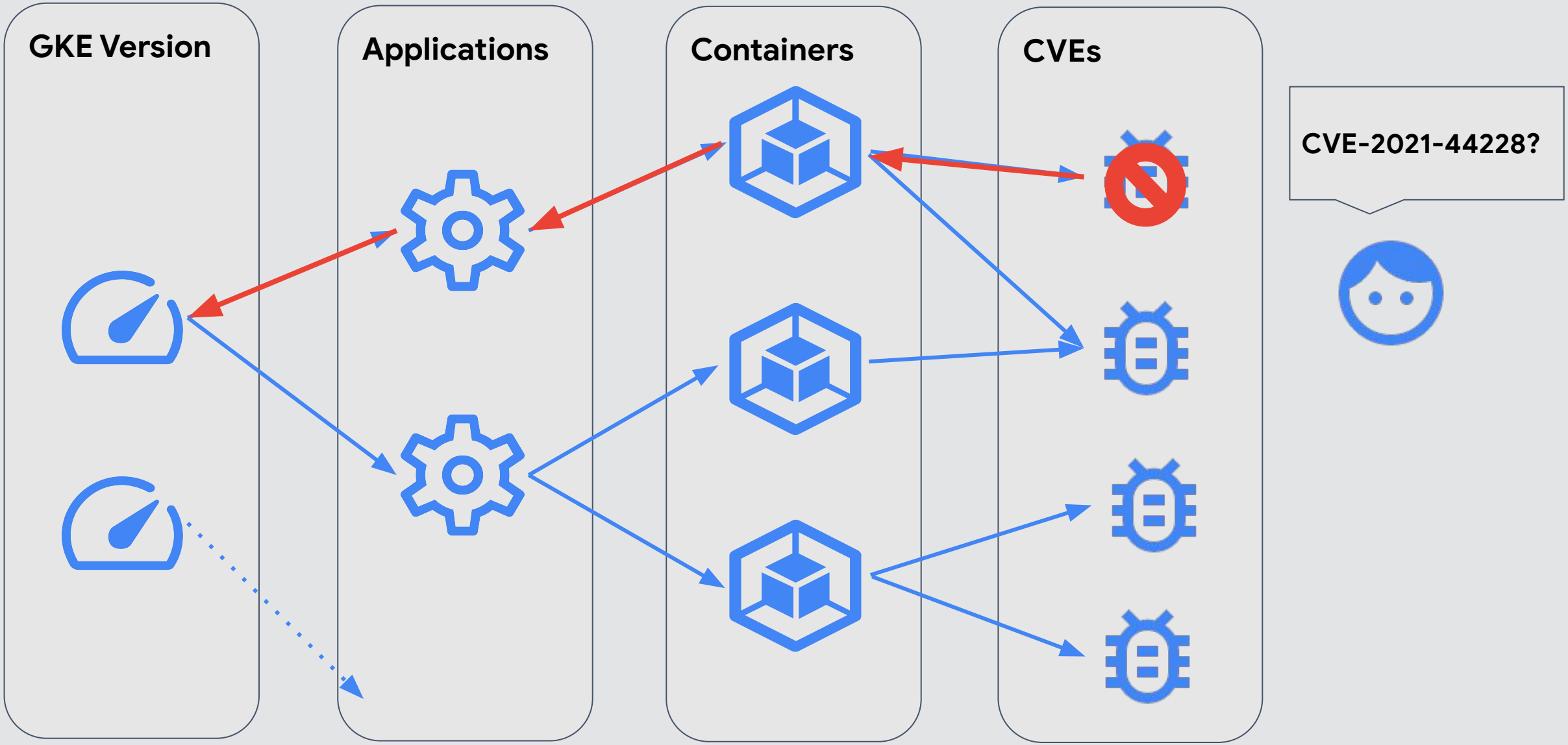
Container isn't patched - who is watching? Who do we escalate to?

Are we meeting our SLOs? What are the gaps and pain points?

Our Solution



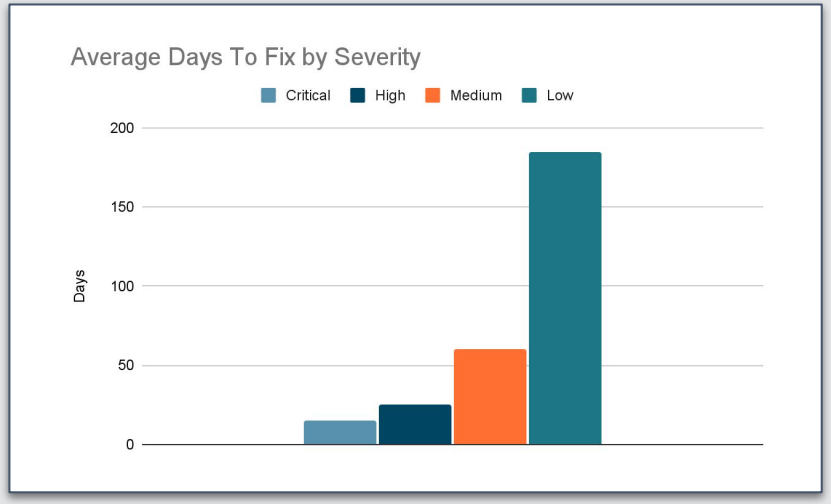
Monitor: Composition



Monitor: Visibility

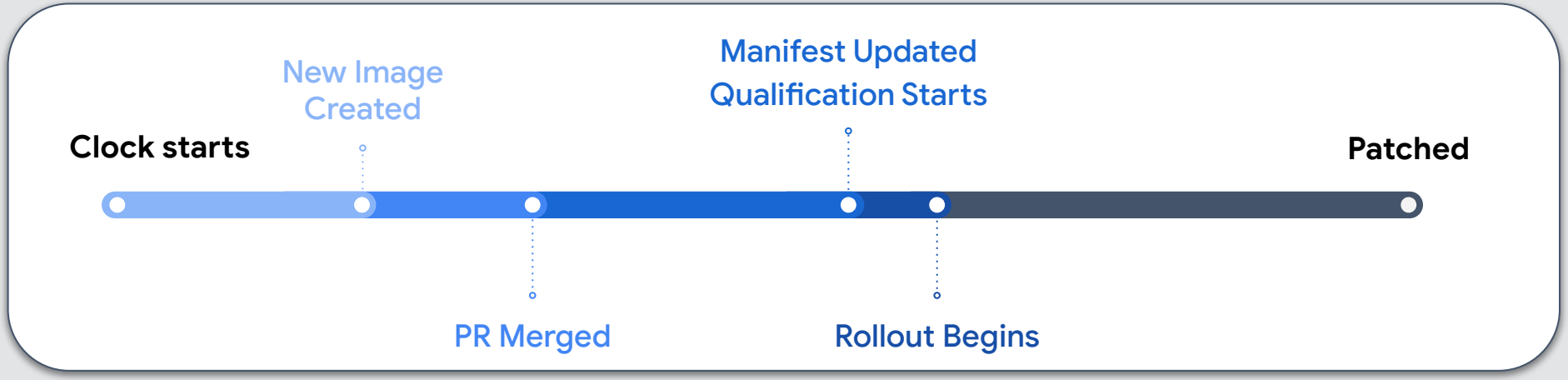
Active CVE Count By Image		
Image Name	**Image Tag**	**# Fixable CVEs**
fake-image	v1.0.1	55
fake-image	v1.0.3	45
demo-image	v3.5	20
nginx	1.22.1	10

Dashboards provide status at-a-glance



Track progress with metrics over time

Measure each step to find pain points



Monitor: Alternatives

- Inventory: Scanners
- Composition:
  - Lyft: Cartography graph database
  - SBOMs / GUAC
  - Ignore layers, just patch: copacetic, crane rebase
- SLO:
  - Bug management software
  - Track commits and rollouts

Summary: Monitor

- Track SLOs over time
- Track patch/release stages to identify bottlenecks
- Use existing systems for escalation/dashboarding

Summary

- Standardize on registries and minimal containers
- Enforce as far left as possible
- Scanners for inventory + visibility
- Record ownership of containers
- Auto-patch if possible
- Tickets to track/escalate

Prefer automation (doing) over telling

Links

- Demo code
- Slim.ai container report
- Lyft patching blogpost
- Separate build and runtime images
- Small images: Scratch, Distroless, Wolfi/Chainguard Images
- SlimToolkit
- AllowedRepos Gatekeeper policy
- Sigstore: signing, policy controller
- GKE Binary Authorization: attestations, image policy
- Opensource scanners: trivy, clair
- Google Container Analysis
- GUAC
- The Seattle Gum Wall

Appendix: Feature Request Wishlist Idea

If enough users report a critical vuln as inaccurate, the scanner manually evaluates, updates the severity for all their users, and works with NIST to correct NVD

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NVD **NIST:** NVD **Base Score:** 9.1 CRITICAL **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H

Prisma Cloud has received 150 reports that dispute this severity
Aqua Security has received 231 reports that dispute this severity
Google Container Analysis has received 109 reports that dispute this severity