Ship Faster, Ship Safer

A K8s-native agentic canary pipeline for critical training cluster services



Youmin Han

Staff Software Engineer, AI Platform, LinkedIn



Zhuo Zhi

Senior Software Engineer, AI Platform, LinkedIn

KubeCon + CloudNativeCon North America 2026

LinkedIn

Agenda

KubeCon

- Motivation
- How our rollout process has evolved
- A Kubernetes-native canary
- Results
- What's next: agents and shadow runs

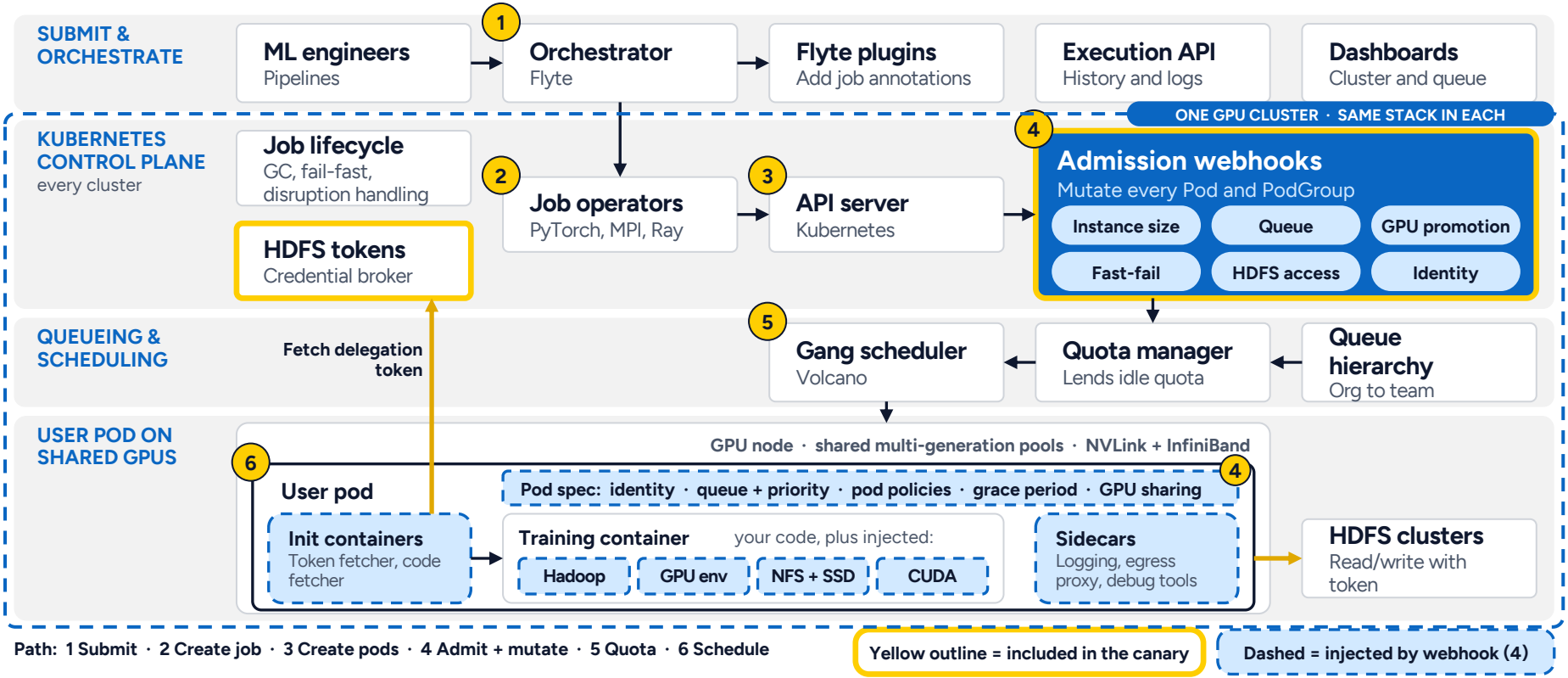
Why our platform needs a canary



1

Millions of ML workflow executions per month on LinkedIn's AI training platform.

Platform architecture: from workflow submission to GPU



Our webhook provides the platform's core capabilities

ADMISSION WEBHOOK · MUTATES EVERY POD AND PODGROUP · 10+ CAPABILITIES, EACH DRIVEN BY JOB ANNOTATIONS

■ has a canary instance

Instance sizing

Turns an instance type into GPU, CPU and memory, and node pool

If wrong: jobs OOM, stay pending or land on the wrong nodes

Queue and priority

Maps the user to a quota queue and priority class

If wrong: silent. Jobs run in the wrong queue or at the wrong priority

GPU promotion

Moves eligible jobs to a newer compatible GPU (H100 → H200) when there is headroom

If wrong: wasted capacity, or jobs on hardware they can't use

Fast-fail checks

Rejects invalid GPU counts, oversized requests and unpullable images up front

If wrong: bad jobs stay pending forever, or valid jobs are rejected

Data access

Injects HDFS token init containers, NFS and local SSD mounts

If wrong: pods can't read their data

Identity and add-ons

Stamps workload identity; adds pre-stop hook, egress proxy, and debug tools

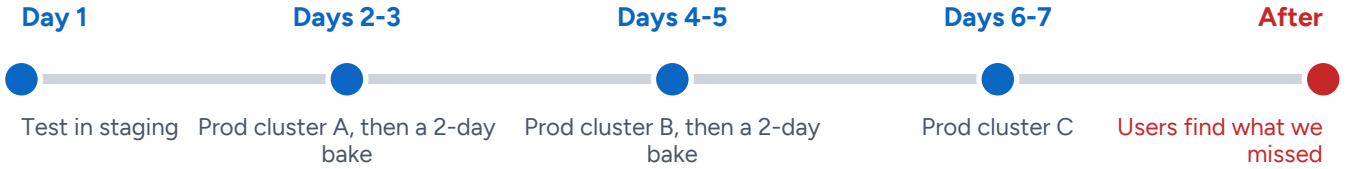
If wrong: broken authentication, lost checkpoints

One webhook, 10+ capabilities. Webhook down = no pod can start

Old releases took a week and still missed prod bugs

~1 week

per release



WHY IT WAS SLOW

Cluster-by-cluster bake

Prod clusters are rolled out one at a time. We check health and user reports, then bake for 2 days before the next, so a bad release can't reach every cluster at once.

→ About a week per release

WHY IT STILL MISSED BUGS

Staging can't reproduce prod

Staging has GPUs, but different GPU types, networking, HDFS setup and storage drivers. Bugs that depend on prod only show up in prod.

→ Alerts/users catch what we missed, after impact

NEW IN THE AGENT ERA

PR volume and development velocity keep climbing.
Staging can't test every change faithfully, and a week-long rollout can't keep pace with the PRs.

We needed prod-fidelity tests on every change, without a week of cluster-by-cluster waiting.

Missed bugs reached prod: what they cost and how long they lasted

23 min
first to last failure

HDFS broke after a webhook release

Impact: 60 workflows and 579 pods on one cluster.
Root cause: Wrong init-container image and Hadoop classpath in the release config.

~30 min
webhook down

Webhook down after a dependency upgrade

Impact: No pod admission.
Root cause: The server silently failed to start after a client upgrade.

Hours
jobs misplaced

Jobs in the wrong queue

Impact: Jobs ran using the limited GPU quotas from wrong queues.
Root cause: A renamed queue and a stale mapping cache. The queue path fails open.

~2 days

GPU jobs lost their GPUs

Impact: 820 executions in 110 workflows.
Root cause: A webhook change altered library setup in GPU pods.

The quieter the failure, the longer it lasted: loud failures took minutes; silent ones took hours to days.

How our rollout approach evolved



2

Our rollout evolved in three phases that build on each other

PHASE 1

Staging-only tests

Test every release in a staging cluster.

Limit: Staging differs from prod, so a pass proves little.

Now: Kept for global tests a namespace can't cover, e.g. CRD changes.

PHASE 2

Percentage-based ramp

Route a percentage of traffic to a new branch with the new logic, the rest to the old branch.

Limit: Risks critical jobs, covers job types unevenly, and breaks for all-or-nothing and multi-component changes.

Now: optional. Kept for quick rollback by setting the percentage to 0.

PHASE 3 · TODAY

Namespace-scoped canary instances

A copy of each prod component, with prod config, serving only a canary namespace.

Result: Prod environment, representative real jobs, no user exposure.

Now: the only run required before a release lands in prod clusters.

Each phase builds on the last: it fixes the earlier blind spot and makes the earlier step optional.

Phase 1: it passed in staging, and still broke in prod

Staging cluster PASSED

Shared storage mount succeeded.
Every test in the suite is green.

Prod cluster FAILED

The same mount failed.
The storage mount paths differ from staging.

Why staging can't match prod

Storage mount paths
Paths and mounts differ per cluster

HDFS access
Per-cluster endpoints and credentials

Node hardware
Different hardware generations and drivers

Webhook config
Per-cluster values and namespace labels

Takeaway: a pass in staging checks the code, not the environment it will actually run in.

Phase 2: a percentage ramp limits blast radius, but not the risk

What it gave us: Real workloads exercise the new version

What it gave us: A gradual ramp limits the blast radius

Where it broke down

1. **Critical jobs at risk**
   A random share of users' critical training jobs ran on a version nobody had validated.

2. **Uneven coverage**
   A percentage can't balance job types, so some types missed the ramp window entirely.

3. **All-or-nothing changes**
   The HDFS credential broker's DB schema migration is all or nothing. It can't be applied to just a share of requests.

4. **Multi-component changes**
   We couldn't pin one workload to the new versions of several components together. It could hit one new and one old.

Root cause: percentage ramps assume interchangeable requests, divisible changes and a single component.

Phase 3: namespace-scoped canary instances

1

Replicate

Every prod component gets a canary instance with the same prod config and the same cluster environment.

2

Scope

A canary instance serves only a dedicated canary namespace. User jobs never reach it.

3

Run real jobs

Representative workloads, modeled on real user training jobs, run in that namespace.

Ever-growing coverage: every new incident or failure mode becomes a new canary workload, so the suite keeps expanding and the same failure can't slip through twice.

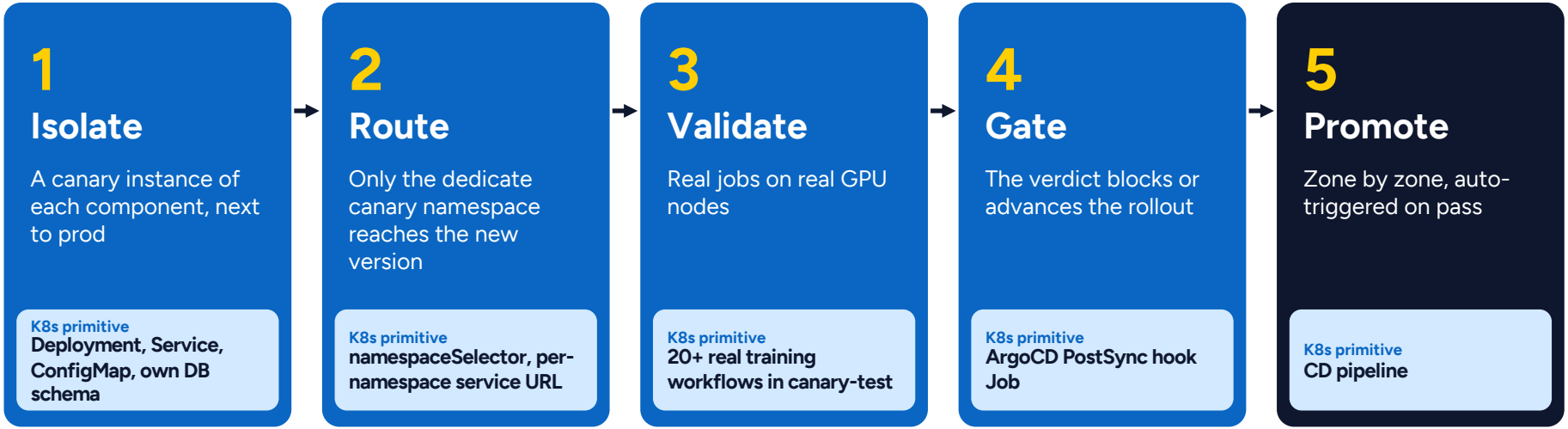
What this fixes: prod fidelity (phase 1), no user exposure and every job type covered (phase 2), and multi-component and all-or-nothing rollouts.

A Kubernetes-native canary



3

The pattern: five Kubernetes-native moves



Same canary, same nodes, same data. The only thing that differs is which namespace a pod comes from.

We'll walk the five moves in order: the strip at the bottom of each slide shows where we are.

Two worked examples of the pattern

EXAMPLE 1

Admission webhook

The service itself

Its own Deployment and webhook configuration, selected by namespace label.

- → Isolate: a second webhook stack next to prod
- → Route: only canary-test meets the new version
- → Gate: real jobs, judged by an ArgoCD PostSync hook

EXAMPLE 2

HDFS credential service

A backing service

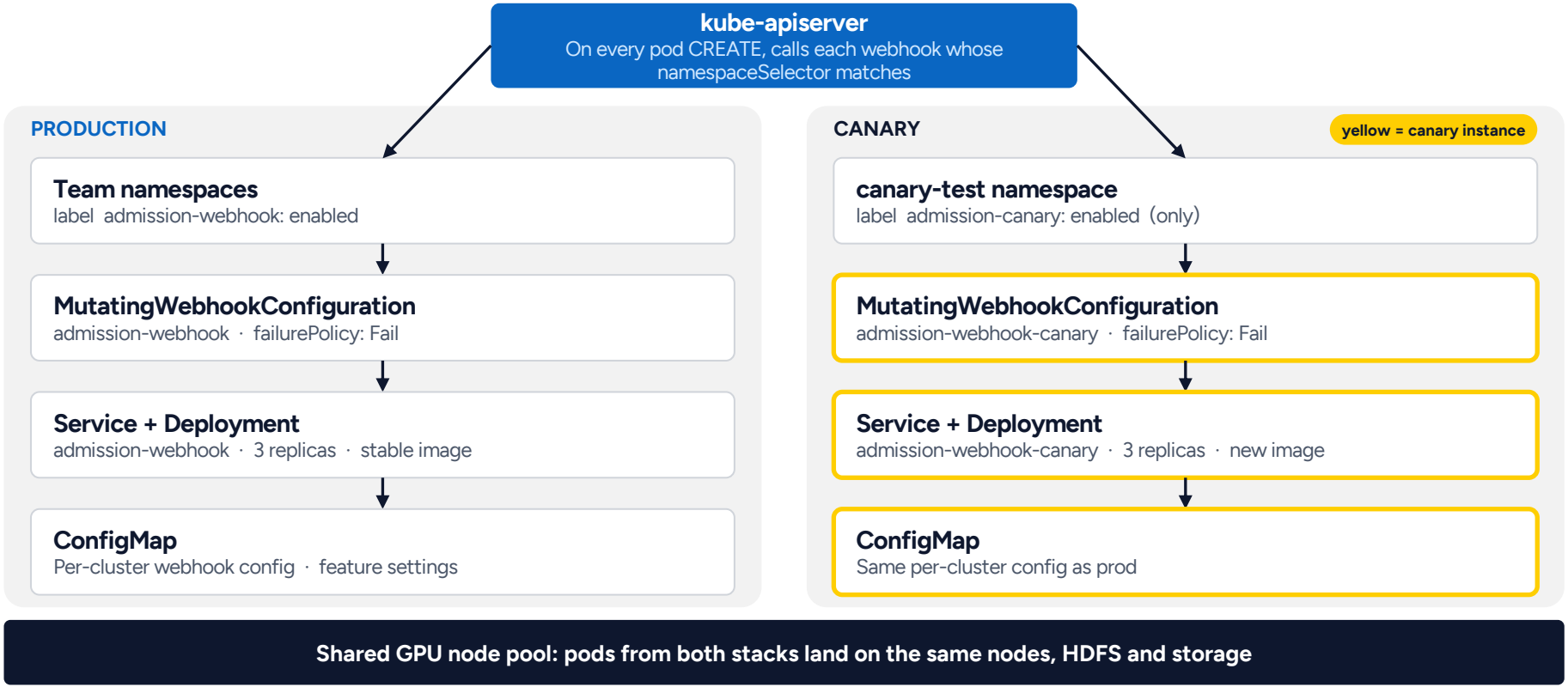
The canary webhook points the injected init container at the canary URL.

- → Isolate: its own Deployment, chart and MySQL schema
- → Route: one URL in the canary webhook's ConfigMap
- → Gate: its own PostSync health check

One pod in canary-test exercises both at once

So the suite validates the webhook and its backing service together, on the same nodes and data as production.

Isolate: two webhook stacks behind one API server



Isolate: a canary credential service with its own database

PRODUCTION

Deployment + Service
credential-service · 3 replicas · stable image

↓

Helm chart + ArgoCD app
Sync only

↓

MySQL
Prod schema · prod DB user

CANARY

yellow = canary instance

Deployment + Service
credential-service-canary · 3 replicas · new image

↓

Helm chart + ArgoCD app
Sync + PostSync health check (HDFS workflows)

↓

MySQL
Canary schema · canary DB user

Migrations land on canary first
Schema changes meet real data shapes before prod.

Isolation by schema and DB user
Even on a shared DB host, the canary can't touch prod data.

All-or-nothing changes become testable
The whole service, schema included, flips as a unit.

Shared data path: both instances issue delegation tokens for the same HDFS clusters

Route: one label sends canary pods to the canary webhook

CANARY PATH

Pod
in canary-test

→

Namespace label
admission-canary: enabled (only)

→

Canary webhook config
namespaceSelector matches the canary label

→

Canary webhook
New image mutates the pod

PRODUCTION PATH

Pod
in a team namespace

→

Namespace label
admission-webhook: enabled

→

Prod webhook config
namespaceSelector matches the prod label

→

Prod webhook
Stable image mutates the pod

```yaml
# canary: same kind, new name, new label
kind: MutatingWebhookConfiguration
metadata:
  name: admission-webhook-canary
webhooks:
- clientConfig:
    service: {name: admission-webhook-canary}
  namespaceSelector:
    matchLabels: {admission-canary: enabled}
  failurePolicy: Fail
```

One label is the switch
Each namespace carries exactly one label, so one webhook mutates each pod.

No proxy, no mesh
The API server does the routing; prod pays zero extra hops.

Prod stays on the prod webhook
Team namespaces never carry the canary label.

Route: one URL sends canary pods to the canary credential service

CANARY PATH

Pod
in canary-test

→

Token URL
credential-service-canary

→

Credential service canary
New version, own Deployment and ArgoCD app

→

Own DB schema
Separate MySQL schema + DB user

PRODUCTION PATH

Pod
in a team namespace

→

Token URL
credential-service (prod)

→

Credential service
Stable version

→

Prod DB schema
Prod schema + DB user

```
# canary webhook ConfigMap: the only difference
kind: ConfigMap
metadata:
  name: admission-webhook-canary
data:
  credentialServiceUrl: https://credential-service-canary/token

# prod ConfigMap
  credentialServiceUrl: https://credential-service/token
```

One URL is the switch
Canary pods get the canary URL; team pods get the prod URL.

Clients don't change
The token fetcher calls whatever URL it was given.

Prod stays on the prod URL
Team namespaces never get the canary URL.

Automatic Validation: 20+ real training workflows on all prod clusters

1

CPU workloads:

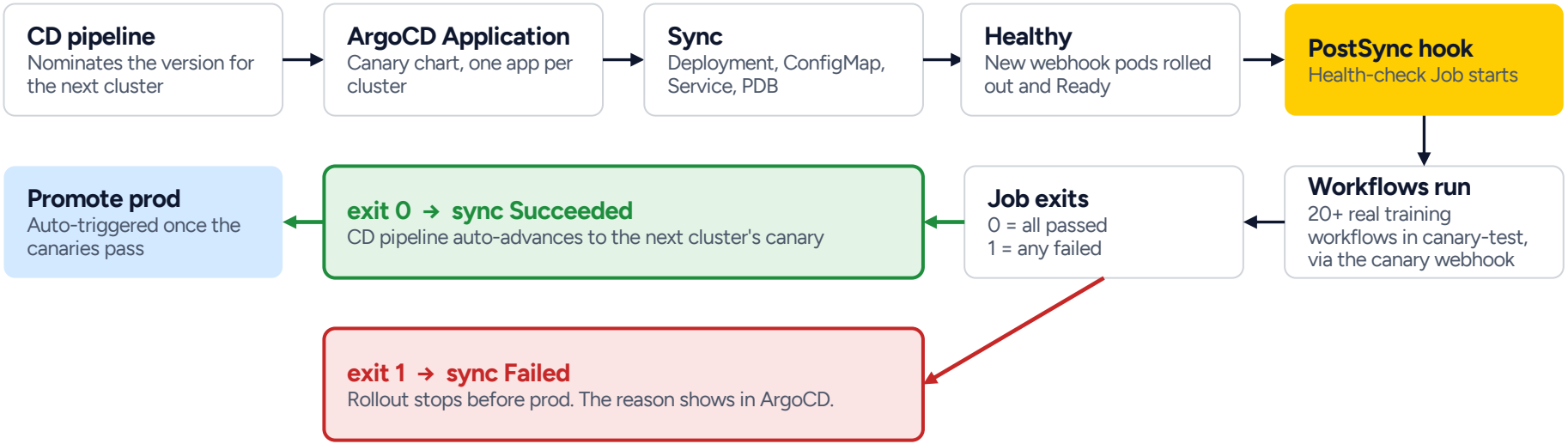
Data access, identity, queue mapping, mlflow logging, agent sandbox

2

GPU workloads:

PyTorch, TensorFlow, MPI, Ray, GNN, GPU promotion

Gate: ArgoCD syncs the canary, then a PostSync hook decides

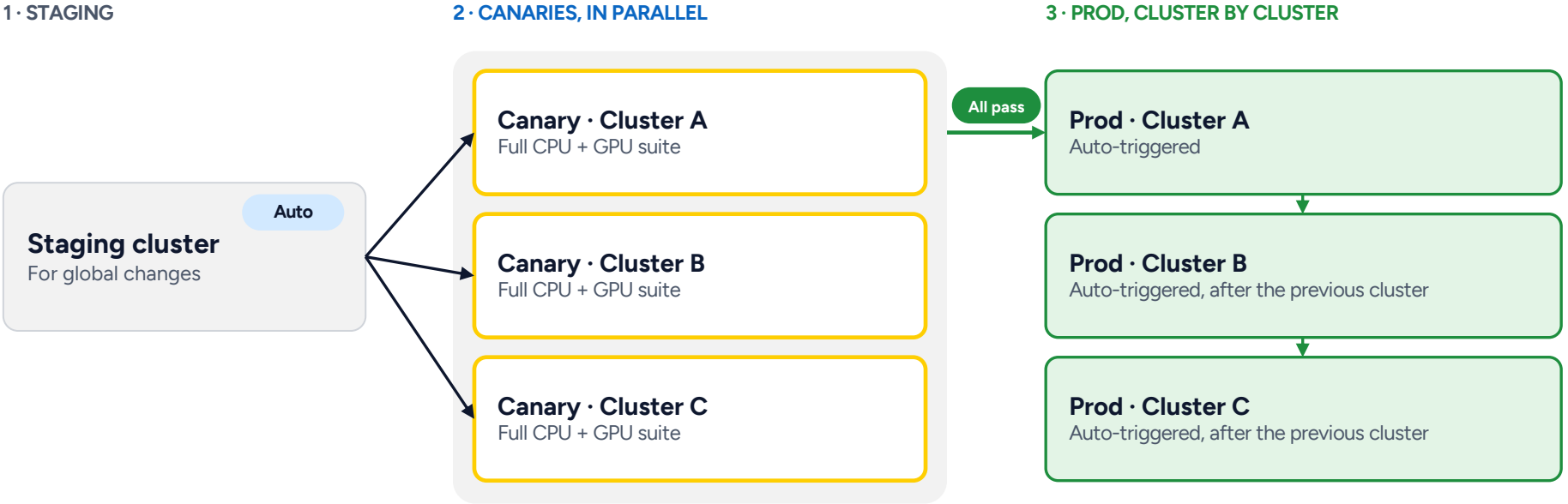


Ordering for free
PostSync runs only after every resource is Synced and Healthy.

Evidence survives
The failed Job and its logs stay until the next sync.

Fix, then replay
Fix forward or roll back the canary; replay the job with zero user impact.

Rollout order: staging first, canaries in parallel, then prod



No human in the loop: the canary is the approval. A red canary stops everything before prod.

Results



4

From a week of hoping to an hour of proof

Same clusters, same GPUs. Here's what changed once every release had to pass the canary first.

	BEFORE		AFTER
SPEED	~1 week per release, cluster by cluster	→	**~1 hour, auto-validated**
COVERAGE	Staging only	→	**Every prod cluster, on real GPUs and storage**
GPU COST	A dedicated GPU test cluster	→	**Shared GPU nodes, zero idle GPUs**
RELIABILITY	Several SEVs in prod	→	**10+ potential incidents caught before prod**
DEBUGGING	Roll back, then dig through logs	→	**Replay the job on the canary**
RELEASES	Stacked changes, hard to isolate	→	**Each version verified on its own**
AGENT-READY	Manual checks can't keep up	→	**Up to 10+ commits a day, each verified**

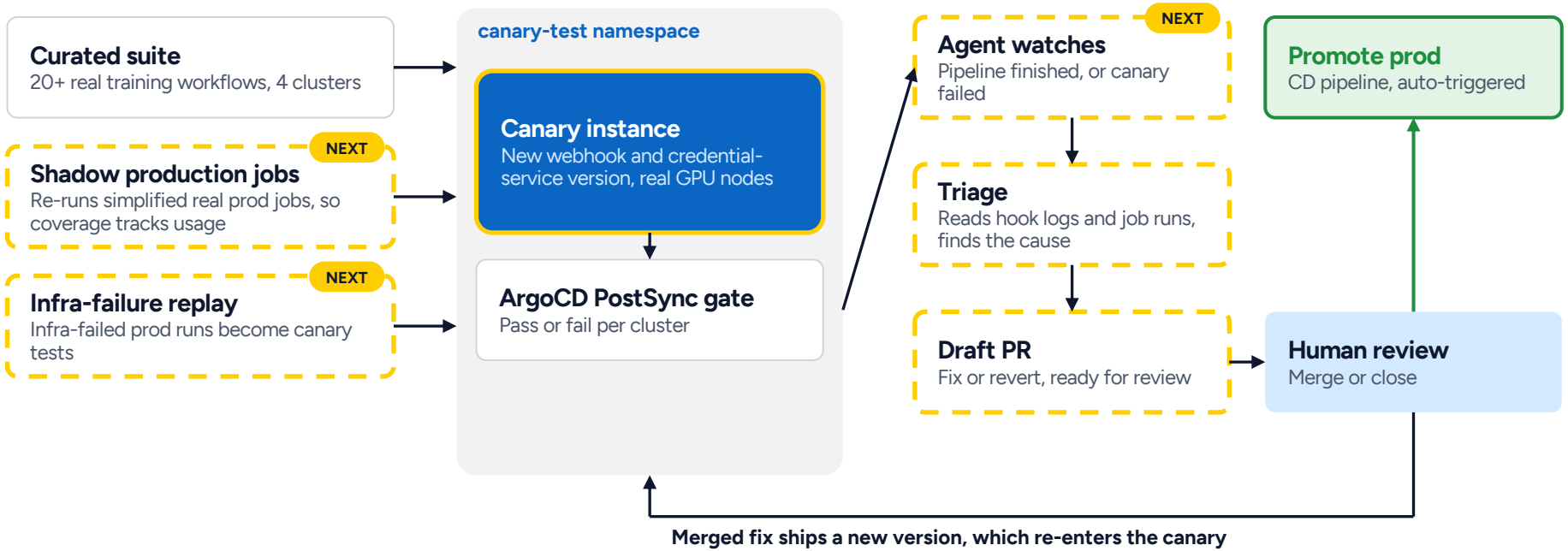
Caught before prod: classpath mismatch · Hadoop binary rollout · missing NFS/CSI driver + GPU sharing config · NFS path casing bug

What's next



5

What's next: production feeds the canary, agents feed the code



Today: curated suite + PostSync gate + automatic promotion. **Next (dashed):** shadow runs, infra-failure replay, agent-drafted PRs.

Thank you.
Questions?



Youmin Han

Staff Software Engineer, AI Platform, LinkedIn
linkedin.com/in/youminhan



Zhuo Zhi

Senior Software Engineer, AI Platform, LinkedIn
linkedin.com/in/zhuo-zhi-4599741b4