Akuity

Shipping Software Safely at AI Scale

Reference patterns for deployment, promotion, and operations

FROM THE CREATORS OF ARGO & KARGO

argo kargo

Introduction

AI platform teams ship across many environments - and the complexity compounds fast.

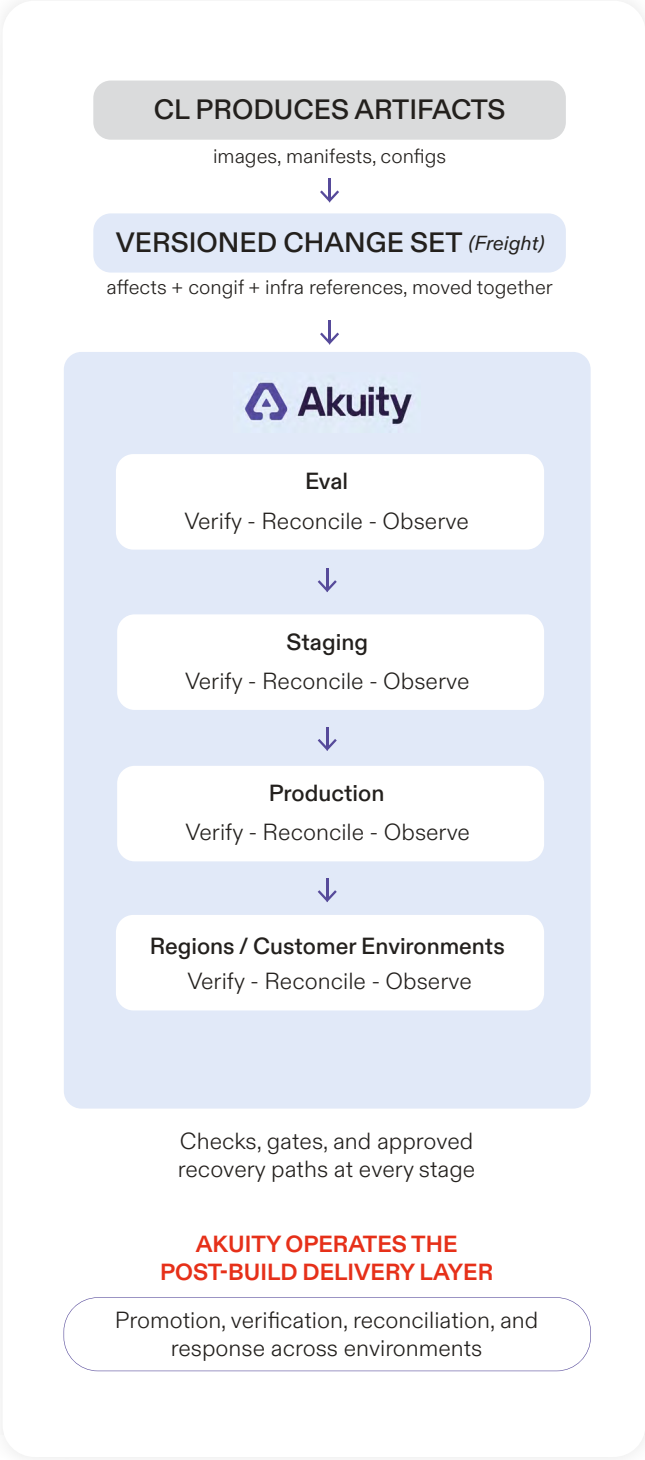
GPU cloud providers coordinate platform services across GPU clusters and customer-owned infrastructure. Inference platforms move serving systems and model-adjacent configuration through eval, staging, regional production, and tenant-isolated environments. AI application teams ship service changes across staging, regional production, and sometimes customer VPCs.

As fleets and environments scale, a release is more than an image tag. It can include Helm values, Kustomize overlays, routing configuration, feature flags, infrastructure references, and workload-specific configuration. The same change set has to land in environments that look similar but behave differently.

The delivery challenge is keeping those changes connected: what moved together, where it was promoted, what verified it, whether the live state matches the intended state, and what context SREs have when something breaks.

This paper describes a reference architecture for that post-build workflow: Enterprise Argo CD for deployment and reconciliation, Kargo for stage-to-stage promotion, and Akuity Intelligence for visibility, review, and response. Built by the creators of Argo CD and Kargo, Akuity brings these capabilities together in one managed delivery platform. As teams add more applications, clusters, environments, and release paths, the operating model around deployment determines whether releases stay safe.

Before the architecture, here are the failure modes that show up first as fleets grow.



Common failure patterns at scale

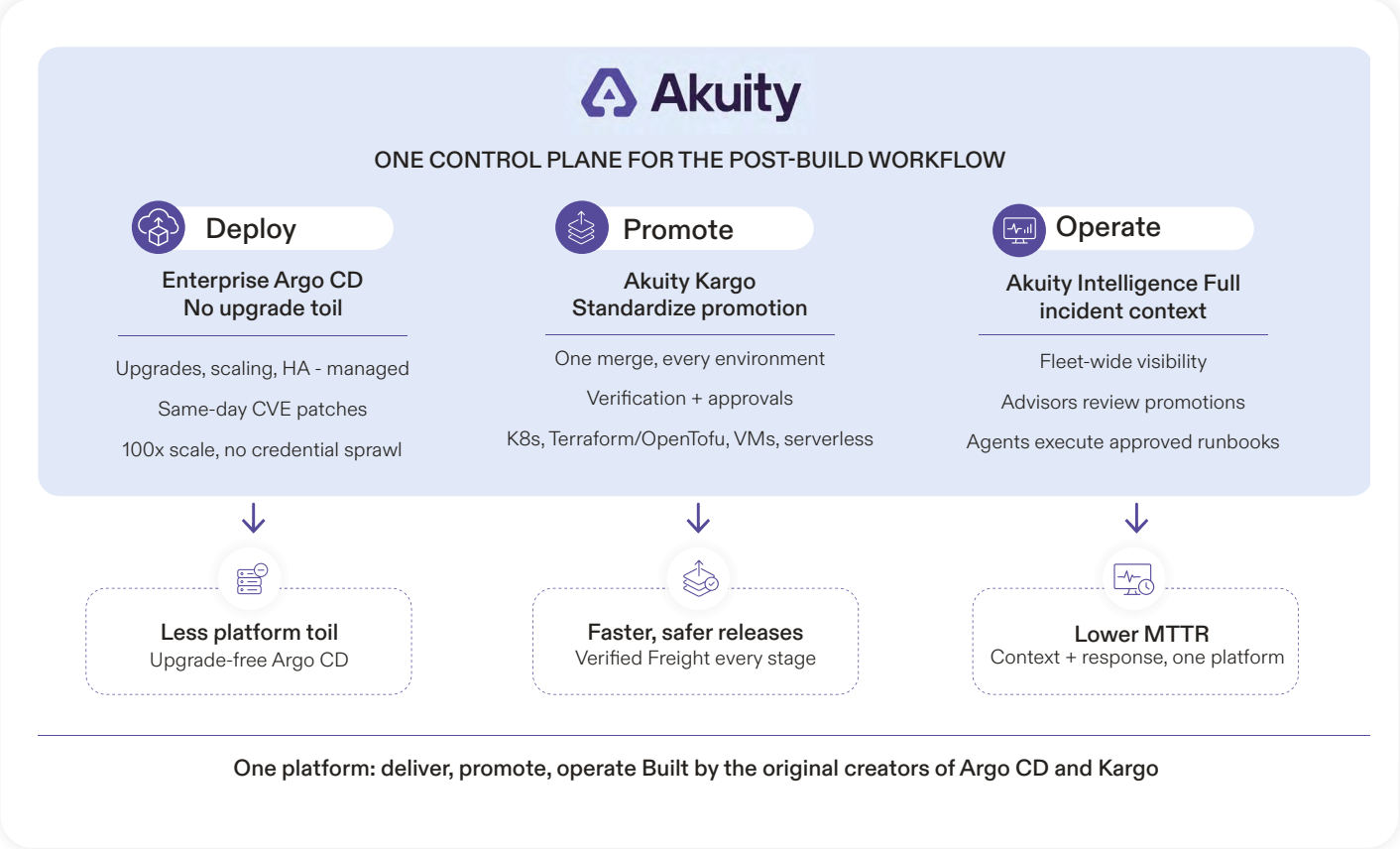
These patterns show up across stacks: Argo CD, Flux, homegrown CD for Kubernetes, separate Terraform pipelines for cloud infrastructure, and different tools for VMs or serverless. As fleets grow, the issue is usually not one broken controller. It is the operating model that connects deployment, promotion, and operations.

DIY GitOps is the right starting point for most teams. The question is when the operating model around it stops scaling.

→ Pattern	→ Symptom
Controller resource pressure	Shared controllers accumulate cache, queue, and memory pressure as app and cluster counts grow. Teams see OOMKilled controller pods, delayed syncs, and slower drift detection.
Promotion logic in CI	Each team writes its own scripts, PR conventions, and approval gates. There is no single object that shows what moved, where it moved, who approved it, and which checks passed.
Related changes desync	Application and infrastructure changes move through separate pipelines. CI passes, but production fails on a combination that was never tested together.
Credential sprawl	Hub-and-spoke setups require persistent kubeconfigs or service-account tokens for each managed cluster. Credential rotation, audit, and private-cluster access become operational work.
Incident context fragmentation	The on-call searches CI logs, Git commits, controller events, dashboards, and Slack threads to reconstruct what changed before choosing a rollback, retry, or mitigation path.

Reference architecture: deploy, promote, operate

Platform teams need one place to deploy, promote, and operate services across environments. Akuity delivers this as a single managed platform built on Argo CD and Kargo.



Deploy with Enterprise Argo CD

Akuity reduces the operational work of running Argo CD: upgrades, controller scaling, HA tuning, and security patching. It runs a re-architected, agent-based version of Argo CD — same reconciliation model, API, and application workflow your team already uses, with the control plane managed and the Application Controller running inside each workload cluster through the Akuity Agent. When a critical Argo CD vulnerability was disclosed, Akuity customers were already patched or had coordinated upgrades ahead of public disclosure.

Promote with Akuity Kargo

One merged change can move through environments using defined promotion rules, approvals, and verification — replacing the dozens of pull requests teams open to roll out across environments manually. Kargo handles this with a single object model: Warehouses watch artifact sources, Freight represents the versioned set of artifacts being promoted, and Stages define where each Freight moves next. Approvals and verification gate downstream movement. The same promotion model can apply across Kubernetes and infrastructure workflows, including Terraform/OpenTofu-managed resources, VMs, and serverless. For example, Cisco ThousandEyes uses this to ship across 25 environments and 2,500+ applications.

Operate delivery with full context and agent-based platform services

Identical recurring incidents get resolved autonomously inside your team's existing RBAC, against runbooks your engineers already wrote. Akuity Intelligence connects delivery state, promotion history, and incident response on the same platform — when an alert fires, the agent already has the context an engineer would: what changed, where it was promoted, which clusters are affected, and which response paths are approved. One customer's On-Call Agent resolved 25 identical incidents overnight from a two-line runbook the SRE wrote during the first one.

Together, these capabilities turn delivery into a connected platform workflow instead of a set of disconnected deployment scripts, approval handoffs, dashboards, and incident threads.

Three patterns that hold up at scale

Pattern 1: ApplicationSet for multi-cluster deployment

ApplicationSet is a practical abstraction for fleets where the same workload runs across many clusters with per-environment overlays. Generators (clusters, Git directories, lists, matrix) produce Argo CD Applications from a single template.

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
spec:
  generators:
    - clusters: { selector: { matchLabels: { workload: inference } } }
  template:
    spec:
      source: { path: apps/inference/overlays/{{.metadata.labels.env}} }
      destination: { server: '{{.server}}' }
```

New clusters opt in by adding the workload label. The same pattern fans out across GPU pools (label by hardware class), regions, tenant-isolated environments, and customer-owned clusters. Eval, staging, inference production, regions, and per-customer environments share a template while keeping their own overlays. The platform team owns the template; service teams own their overlays.

Three patterns that hold up at scale

Pattern 2: Kargo Stages with verification

Kargo separates promotion from CI. A Stage declares which Freight it requests, how to apply it (the promotion template), and what verification gates downstream movement.

```yaml
apiVersion: kargo.akuity.io/v1alpha1
kind: Stage
spec:
  requestedFreight:
    - origin:
        kind: Warehouse
        name: inference-service
  promotionTemplate:
    spec:
      steps:
        - uses: git-clone
        - uses: kustomize-set-image
        - uses: git-commit
        - uses: git-push
        - uses: argocd-wait
  verification:
    analysisTemplates:
      - name: production-slo-check
```

AnalysisTemplates can run integration tests, Datadog or Prometheus queries, HTTP checks, or load tests after a promotion. If upstream verification fails, Freight is not eligible for downstream Stages. A failed verification can also trigger automatic rollback to the previous stable Freight without human intervention.

For example, a Stage's AnalysisTemplate can query Datadog APM during a promotion and trigger automatic rollback if HTTP error rate exceeds a threshold — catching a bad image before it propagates to downstream environments.

Promotion steps are not limited to Kubernetes. A Stage can run tf-plan and tf-apply as part of its promotion template, then pass Terraform outputs to subsequent steps — useful when an IAM role, queue, or networking change has to land before the application that depends on it. The same Stage model applies whether the target is Kubernetes, Terraform-managed cloud resources, VMs, or serverless functions.

The promotion model becomes: build once, package related changes as Freight, promote through Stages, verify at each step, reconcile.

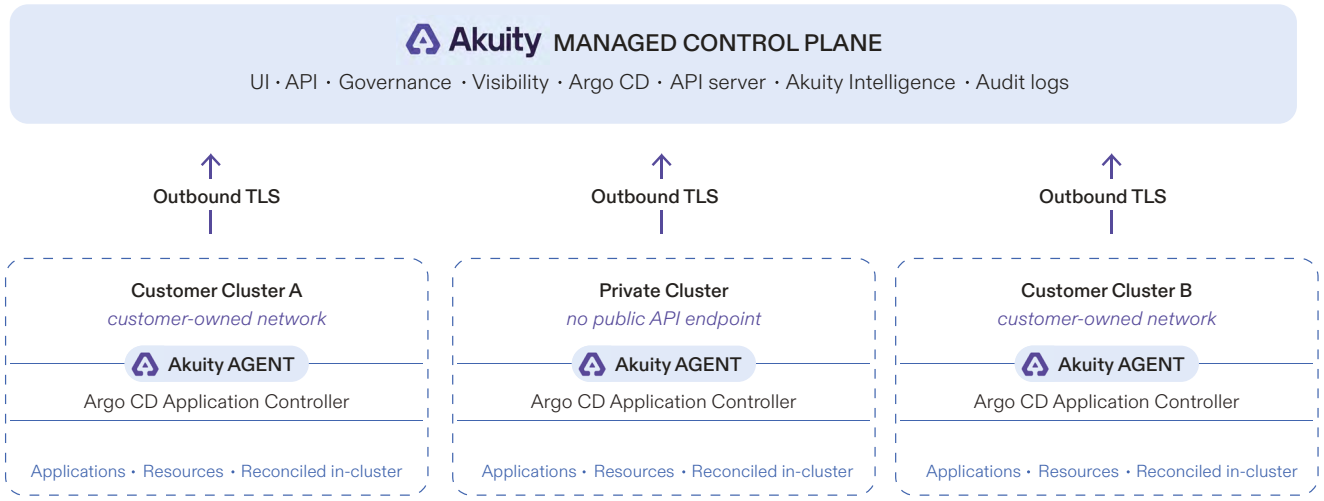
Three patterns that hold up at scale

Pattern 3: Managed control plane with in-cluster reconciliation

Akuity splits the Argo CD topology. The Argo CD API server runs in the managed control plane. The Application Controller runs inside each workload cluster and connects back through the Akuity Agent over an outbound TLS connection.

The implications:

- No public Kubernetes API endpoint required. The agent dials out; the control plane does not dial in.
- No persistent kubeconfigs or service-account tokens in the control plane. Cluster credentials stay in the cluster.
- Reconciliation happens close to the workloads. Akuity's documented testing shows up to 80% less traffic between control plane and clusters versus a centralized hub-and-spoke topology.
- Private and customer-owned clusters connect without VPN or peering arrangements.



The agent dials out. The control plane does not require inbound access to private or customer-owned clusters.

What this looks like in practice

Operational context on one platform

When delivery state, promotion history, and runtime signals live in the same platform, the operations layer can do more than display dashboards. Insights show fleet health across environments. Advisors review proposed promotions and suggest remediation when a deployment degrades. Agents can execute pre-approved runbooks inside the team's existing RBAC without paging an engineer for incidents the team has already documented.

The common thread: every workflow operates against the same delivery objects platform teams manage directly — Applications, Stages, Freight, and the runtime signals around them. An SRE doesn't have to reconstruct a delivery timeline from CI logs, Git commits, controller events, and Slack threads to understand what changed before an incident. The context is already on the platform that deployed and promoted the change.

Takeaway: When operations and delivery share a platform, SREs respond faster, advisors review with context, and recurring incidents can resolve against runbooks the team has already approved.

Customers operating at scale

CoreWeave: scaling GitOps across customer environments

CoreWeave

CoreWeave — Cloud provider purpose-built for generative AI and HPC workloads. As CoreWeave scaled co-managed GitOps environments for customers, manual onboarding became too slow and inconsistent.

With Akuity, **CoreWeave reduced customer onboarding from 2–3 days to under 1 hour and now supports thousands of monthly deployments with near-zero deployment failures.**

PolyAI

PolyAI — Conversational AI for enterprise voice applications. The platform team supports two distinct deployment patterns across engineering teams: services that ship together in coordinated releases, and services that deploy independently.

Using Argo CD and Kargo, **PolyAI runs 1,000+ applications from a single delivery architecture without forcing every team into the same release path.**

Cisco ThousandEyes

Cisco ThousandEyes — Network and application performance monitoring at Cisco scale. The team used self-hosted Argo CD for deployment, but promotion still depended on manual pull requests and environment-by-environment coordination.

With Kargo Enterprise, **ThousandEyes manages 2,500+ applications across 25 environments with structured promotion pipelines and automated verification between stages.**

"Before, rolling out a change across all environments could mean opening dozens of pull requests and managing them manually. Now we merge the change and watch the pipeline progress." — *Nikolay Denev, Technical Lead, Engineering Effectiveness, Cisco, ThousandEyes*

ALEPH ALPHA

Aleph Alpha — Sovereign AI for enterprises and governments. As PhariaAI scaled, the team moved from self-hosted Argo CD to Akuity's managed platform.

Aleph Alpha manages 400+ applications across 11 clusters from one control plane and saves tens of hours of maintenance work per month.

Summary

Most AI platform teams already have a deployment mechanism, whether they run a GPU cloud, a frontier lab, an inference platform, or an AI application stack. Promotion and operations remain open problems. As environment counts grow — across eval, staging, GPU pools, regional production, and co-managed customer environments — the failure modes that show up first sit in the operating model around the deployment controllers rather than in the controllers themselves: promotion logic scattered across CI, application and infrastructure changes moving on different timelines, controller resource pressure, credential sprawl across customer clusters, and incident context fragmented across tools.

Akuity delivers the reference architecture as a single managed platform: managed Argo CD for deployment, Akuity Kargo for promotion across Kubernetes, Terraform, VMs, and serverless, and Akuity Intelligence for the operations layer.

Worth a conversation if any of this sounds familiar:

- Promotion logic is spread across CI scripts, manual pull requests, and Slack approvals
- Application controllers are falling behind as clusters and applications grow
- Releases span Kubernetes, infrastructure, and serverless workflows, but there is no end-to-end audit trail
- Customer or tenant clusters need to stay private, with no inbound access from the control plane
- Teams lack one reliable view of what is running, where it was promoted, and which checks passed

BOOK A DEMO

Brought to you by Akuity
akuity.io