Raiffeisen Bank International

From GitOps to AIOps

Evolving RBI's Kubernetes Platform with Crossplane and sharded Kargo

Gabor Horvath & Ewald Überall

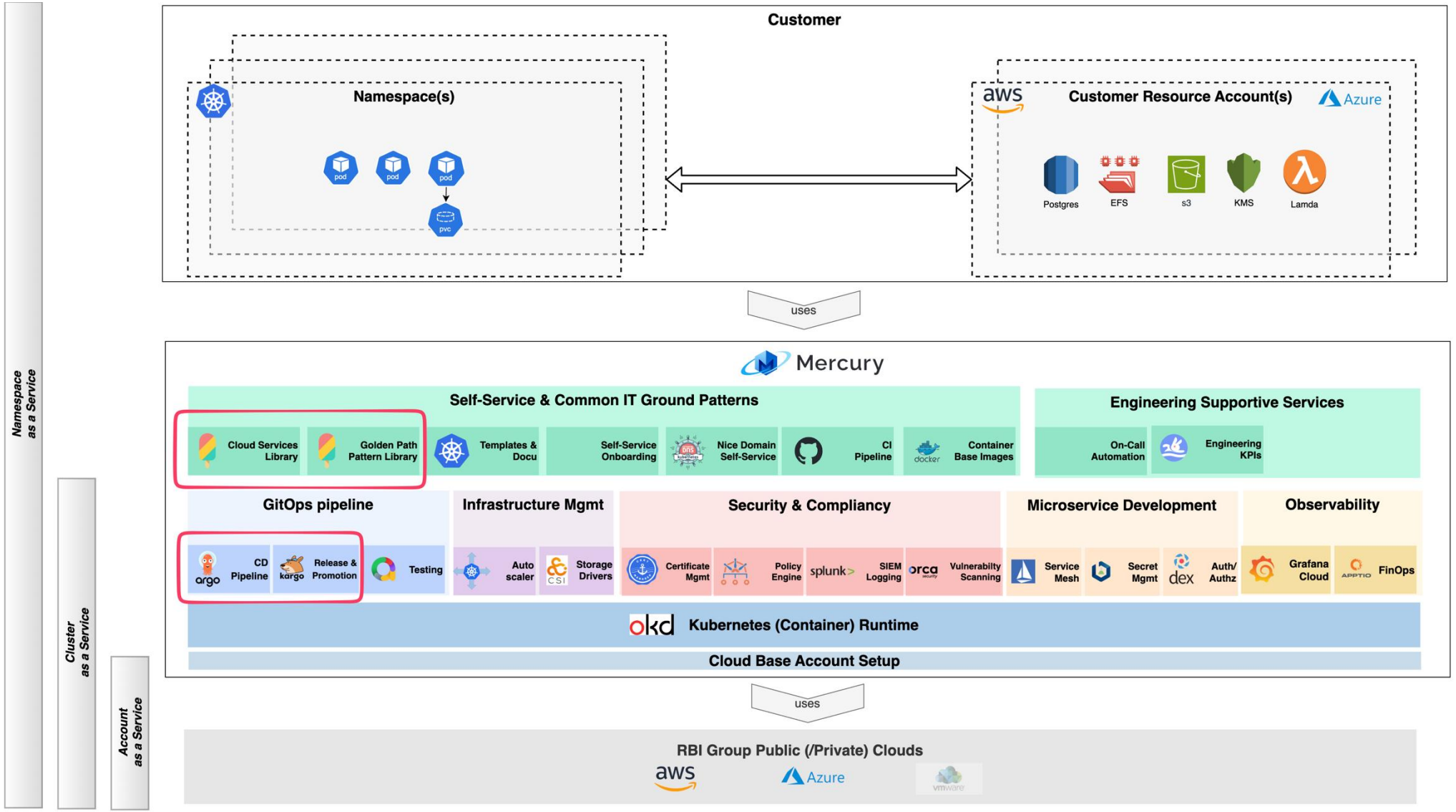
About Raiffeisen Bank International

- **one of Austria‘s leading corporate and investment banks**
- **11 subsidiary banks in CEE**
- around 1300 business outlets
- more than 1000 awards from financial magazines
- **18.6 million customers**
- ATX listed since 2005
- EUR 210 bn total assets (31.12.2025)
- **around 42000 employees**
- in CEE since 1986

Mercury Team & Mercury Platform

- group-wide platform for building and running cloud native applications
- Team of 20 people (15 engineers)
- 100+ engineers uses the platform
- 1000+ namespaces across 13 clusters
- 2 other platform teams in our banking groups are re-using the stack
- Namespace as a Service: fully managed Kubernetes runtime platform, based on OKD, owned and maintained by the Mercury Team
    - customers get dedicated and isolated namespaces and resources to run their containers
- Account as a Service: Customer Resource Account for deploying cloud resources in AWS, connected to container workload running on the Mercury NaaS clusters
- Cluster as a Service: Dedicated, Kubernetes runtime platform, based on OKD, self-hosted by the product teams

Mercury Platform – High Level Architecture

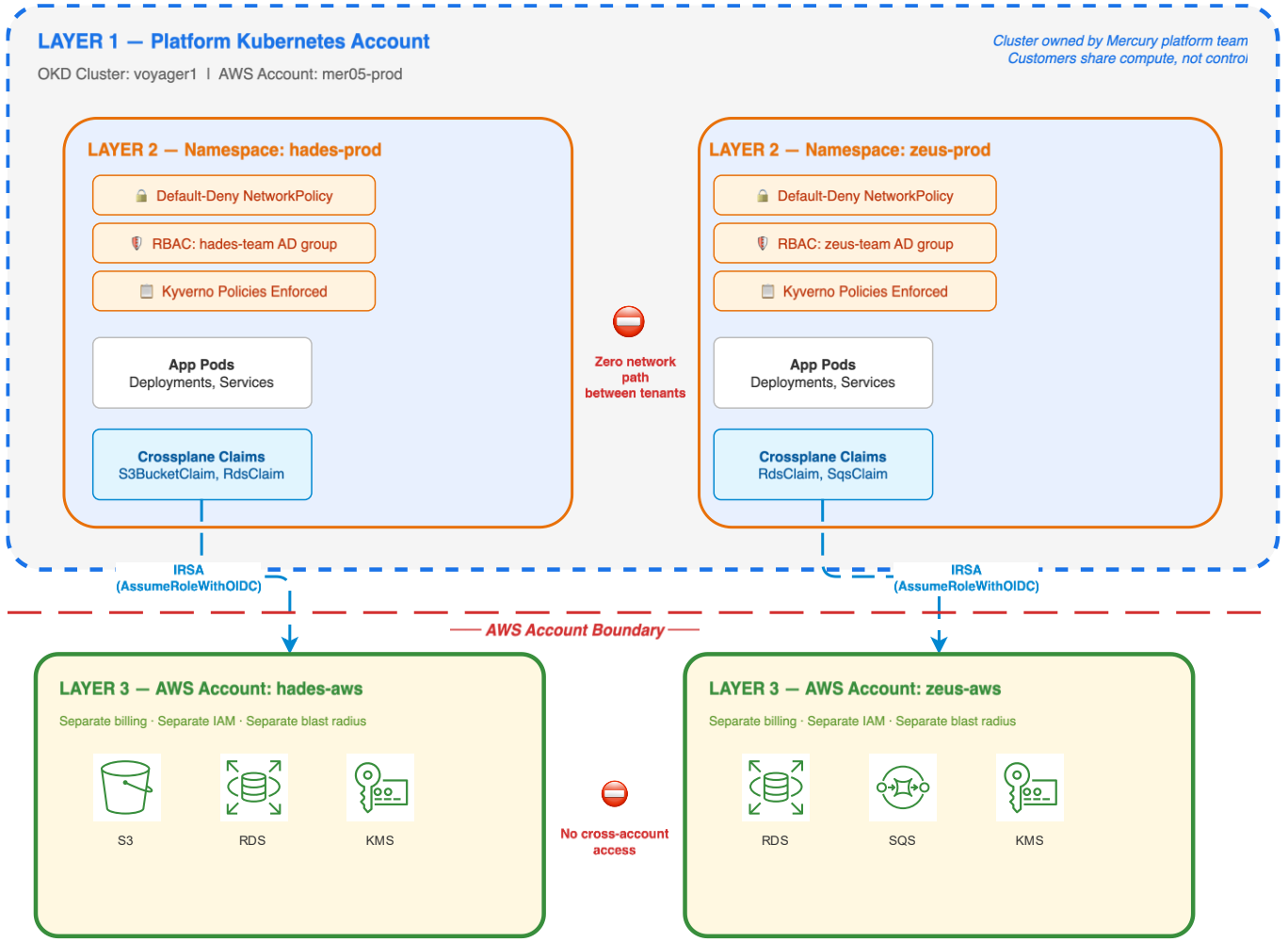


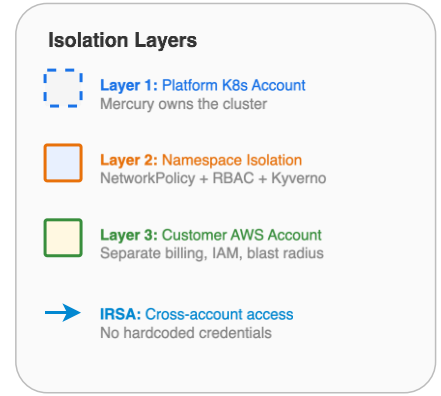
From GitOps to AIOps: Evolving RBI's Kubernetes Platform with Crossplane and sharded Kargo

Raiffeisen Bank International

Scale & Multi-Tenancy Reality

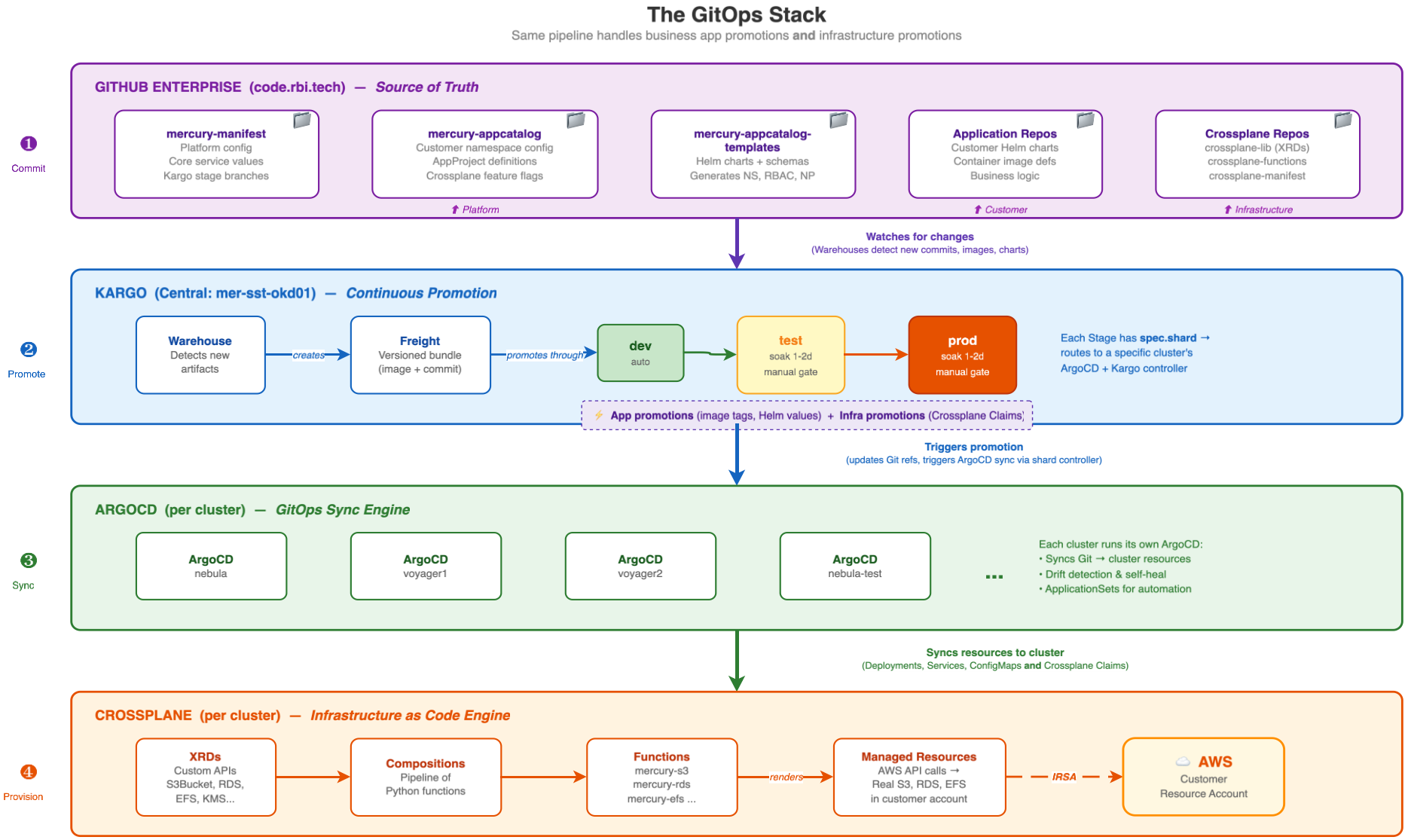
- 70+ customers
- 100+ namespaces
- 120 AWS accounts
- Three distinct isolation layers
- Isolation is real and strong,
  - but it makes promotion complex





Mercury GitOps Stack

- Same pipeline for business apps and infrastructure promotions (Crossplane Claims)
- Promotion gates (soak time + manual gates, PR approvals)
- spec.shard maps each Stage to a specific cluster – this is how Kargo knows where to execute promotion



Not All Promotions Are Equal

App Promotion	Infra Promotion
Bad image tag → pods crash	Bad Crossplane Claim → real AWS resource changed
Rollback: replace Freight, pods recover in seconds	Rollback: AWS API calls – minutes (RDS), days (KMS), or irreversible (data loss)
Blast radius: pods in one namespace	Blast radius: customer's AWS account
Failure is visible immediately	Failures can be silent (wrong account, orphaned resources)

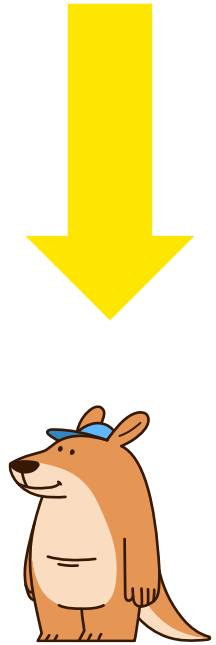
Risks:

- orphaned resources: failed sync + force sync = duplicate RDS instances the customer pays for without knowing
- Test account is the gate to prod: break the test-tier AWS account → integration tests fail → prod promotions blocked
- wrong or incomplete Crossplane v1 → v2 migrations

Raiffeisen Bank International

So what do we need?

- Isolated execution – a failing infra promotion on test must not affect prod
- Different risk policies – infra and prod promotions need stricter gates than app promotions
- Visibility without central control – one UI to see everything, but no single controller touching everything



Kargo sharding aka Kargo Distributed topology configuration aka Kargo Agent-based architecture

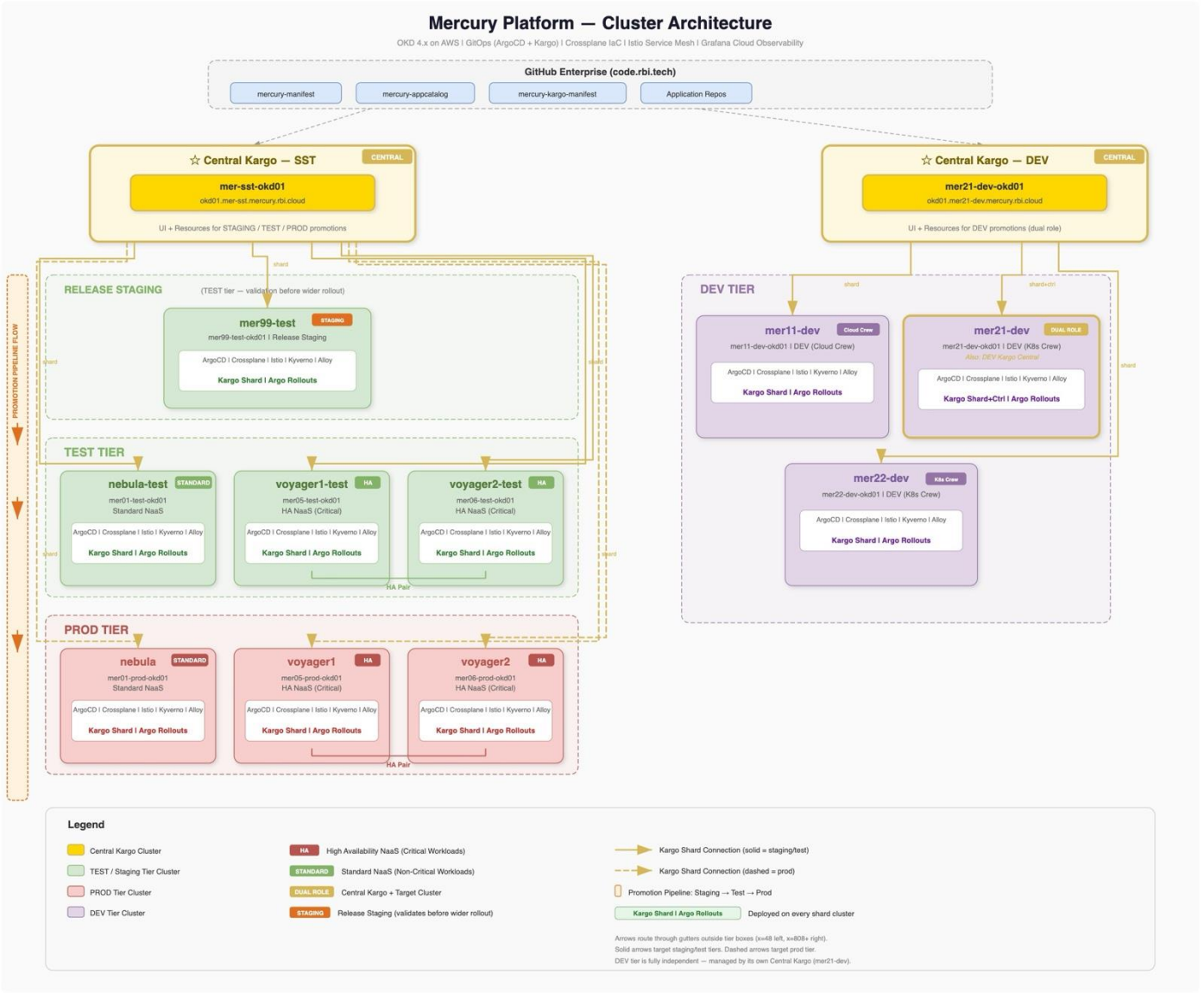
Why shard Kargo?

Hub-and-Spoke	Kargo Control plane per Cluster	Kargo Agent Model ✅
One controller, many clusters	One controller per cluster	Central data, distributed controllers
Single pane of glass	No unified view	Single pane of glass
High-value target if compromised	No central risk	Control plane has no cluster access
Performance degrades at scale	Scales well	Scales well

Mercury has 6+ clusters, each with its own Argo CD. A single Kargo controller can’t talk to all of them, and shouldn’t. The distributed controllers ”phone home” to central Kargo Control Plane.

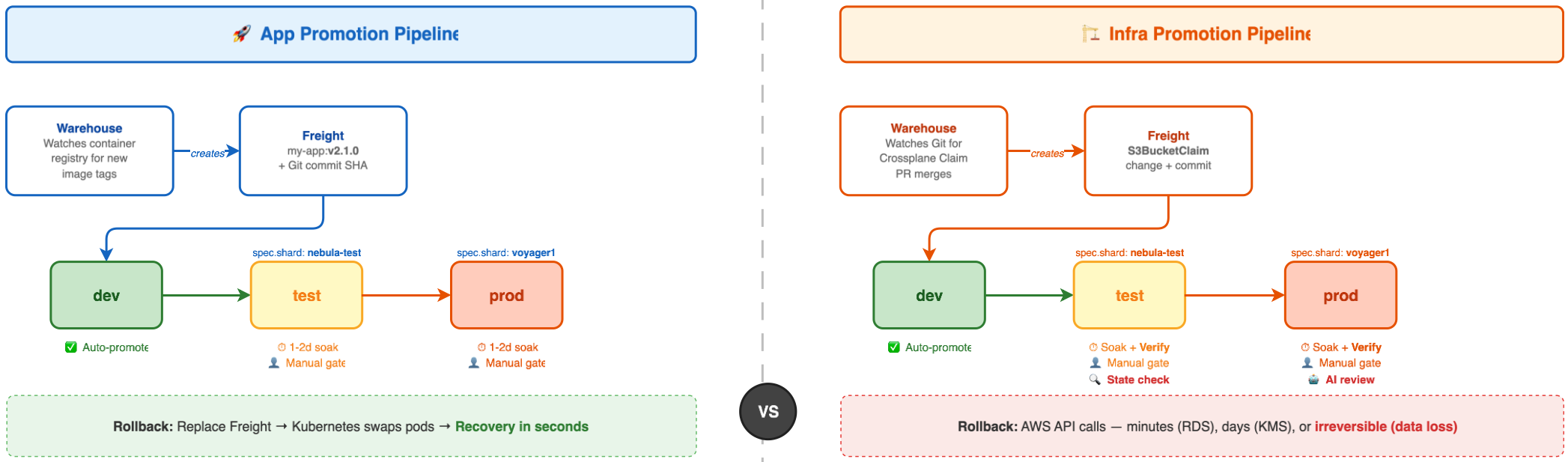
Mercury's Kargo Sharded Topology

- SST Kargo: all TEST/PROD promotions, single customer facing UI
- DEV Kargo: all DEV promotions (dual role = control plane + target)
- Each shard: local Kargo controller + local Argo CD + local Argo Rollouts
- Control plane store all Kargo resources – shards only execute
- In practice:
  - Customer defines spec.shard on their Stage
  - Central Kargo (SST) stores the Stage, Freight, Promotion resources
  - Shard controller picks up the Promotion, executes it with local Argo CD



Infra Promotions

Infra Promotions — Same Pipeline, Stricter Controls



What Sharding Enables: Different Risk Policies Per Pipeline

Aspect	App Pipeline	Infra Pipeline	Why It Matters
Freight Source	Container image tag	Crossplane Claim PR	Infra changes are Git commits, not binary artifacts
Verification	Health check + smoke test	**State check + AI analysis**	Must verify real AWS resource state, not just K8s health
Rollback Time	**Seconds**	**Minutes → days → irreversible**	RDS: 20min. KMS delete: 7-30d wait. Data loss: permanent
Failure Mode	Pods crash — visible immediately	**Silent: wrong account, orphans**	Orphan resources = customer pays for infra they don't know exists

With sharding, infra promotions can have different verification steps, longer soak times, and additional gates — per shard, per cluster, per risk profile

Raiffeisen Bank International

What Sharding Gives Us?

	Without Sharding	With Sharding
Controller scope	Single central controller touches all clusters – high-value target, single point of failure	Shard controllers are local to each cluster - no cross-cluster privileged access
Argo CD integration	One Kargo <-> Argo CD integration point - bottleneck, can't scale past one Argo CD	1:1 Kargo shard per Argo CD - each shard works with its local Argo CD instance
Failure blast radius	Promotion failure in one cluster can cascade - stuck controller blocks all clusters	Contained per shard - stuck nebula-test does not affect voyager1-prod
Risk policies	Same soak times, same gates, same verification for all promotions regardless of type	Per-shard, per-workload-type policies - infra promotions get stricter gates than app promotions
Reconciliation	Single queue - all Stages across all clusters compete for the same controller loop	Parallel, scoped queues - each shard reconciles independently
Observability	Full visibility from one place	Single UI still - SST Kargo stores all resources, shards only execute

Crossplane: The Promise

So Kargo handles the promotion. But what about the infrastructure it’s promoting?

Raiffeisen Bank International

Why We Needed V2

The problem

Status of managed resources belonging to a claim was barely visible in v1

Customers had no easy way to see what resources exist and what state they're in

What we did to help

Heavy use of the status transformer function in our compositions

Surfaced managed resource names and their current state into the claim status

But it wasn't enough

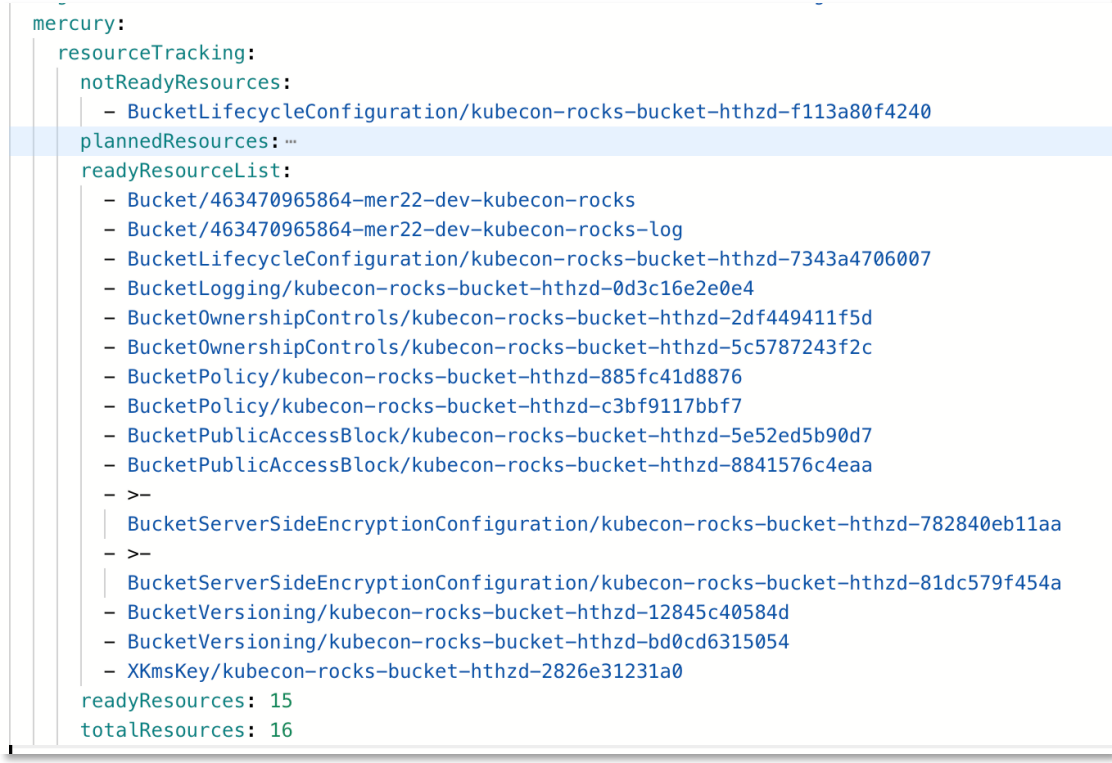
Still too cumbersome for customers to find the right information

Hard to explain where to look - resulted in a high volume of support requests

The conclusion

Adopt Crossplane v2 as fast as possible

V2 brings native visibility with namespaced resources



The Migration Process in 3 Steps

No resource recreation. No downtime. One claim at a time.

Step 1) Enable Migration Mode

Add migration-parameter to the existing V1 claim

Composition function reads all managed resources from the pipeline context

Writes the full resource list into the claim's status field

Step 2) Create V2 Claim

Create a new V2 claim with the exported resource list

V2 claim imports all existing managed resources

Ownership remains with the V1 claim - no disruption yet

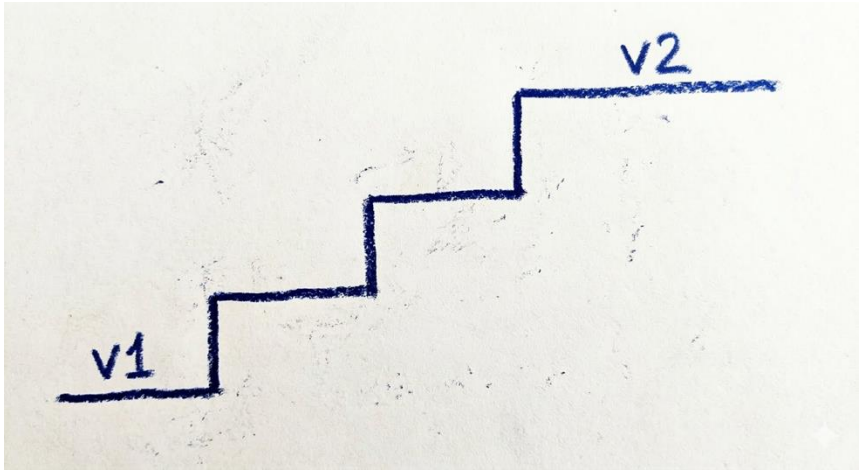
Step 3) Cut Over

Delete the V1 claim

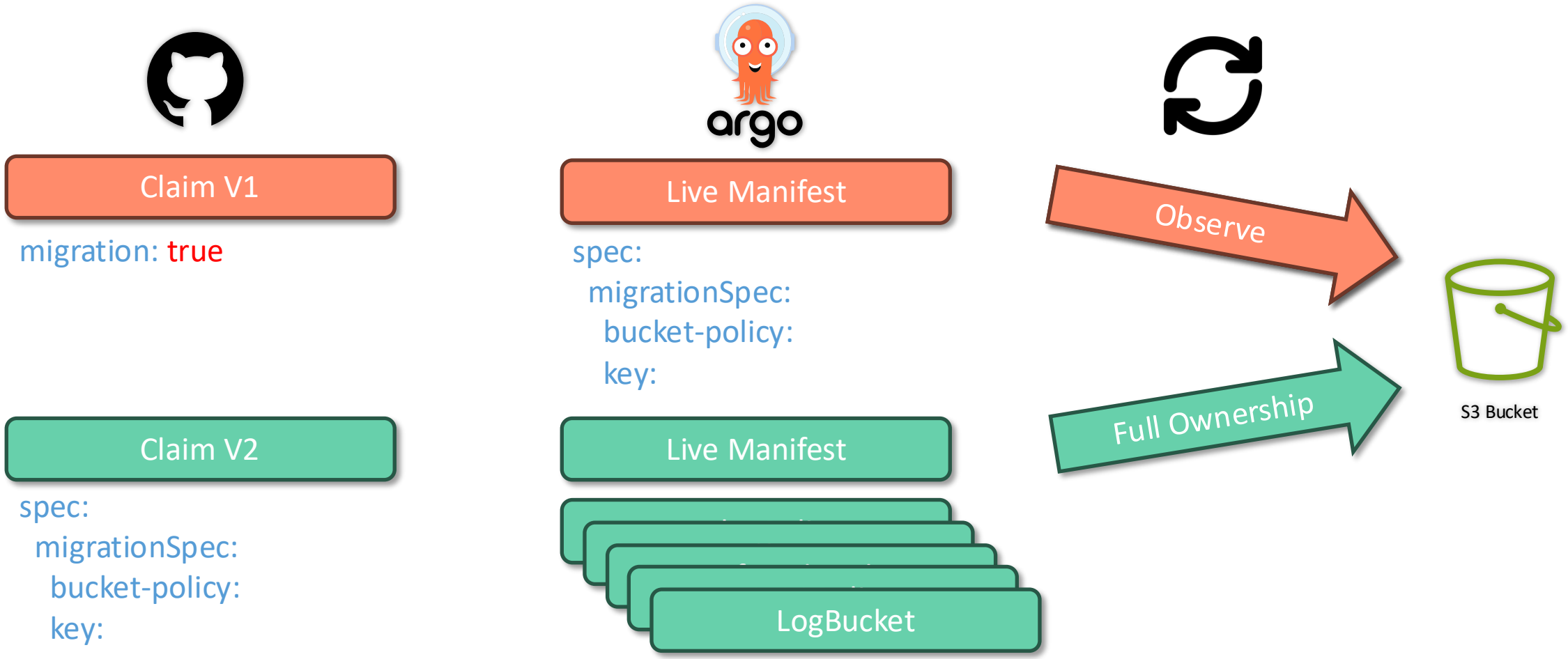
Remove the managed-resources parameter from the V2 claim

V2 claim takes full ownership - migration complete

Crossplane



Same Resources. New Owner.



Raiffeisen Bank International

The Human Problem

- Customer opens a PR to enable migration
- PR is YAML: namespace labels, feature flags, resource references
- Common mistakes:
  - Missing linkedResourceAccount (#1 Crossplane failure cause)
  - Wrong namespace annotation
  - Copy paste errors
  - YAML indentation errors
- Platform team reviews PRs -> doesn’t scale -> error-prone under time pressure
- Can AI help?

Raiffeisen Bank International

What if the Reviewer Had a Second Pair of Eyes?

1. An AI agent that is aware of Crossplane resources, Argo CD app states
2. It reviews the migration PR against the actual cluster state – not just the YAML diff
3. It flags risk before the promotion enters the pipeline
4. Gives Argo CD application failure analysis in case issue slips through the PR approval gate

AI does not automate the migration or remove human from the loop, it just surfaces risk that would otherwise got unnoticed. Making the invisible visible.

Live Demo

What you'll see:

1. Customer opens Crossplane migration PR
2. Kargo shard picks up the infra Freight
3. Intentional mistake: wrong YAML section for migrationSpec
4. AI PR review grades the change risk
5. AI analysis via Argo CD MCP surfaces the issue and gives remediation hints
6. Fix the PR -> re-promote -> success

Raiffeisen Bank International

Demo

What AI actually did?

- Kargo used ’http’ promotion step to POST to **/analyze/risk endpoint**

- Accepts a GitHub PR identifier (pr_url or owner + repo + pr_number)
- Makes 3 parallel calls to GitHub MCP (via sidecar container) to fetch:
- PR metadata (title, description, author, base/head branches)
- Raw unified diff
- List of changed files
- Filters the diff - drops noise files (lock files, binaries, minified/generated code) before sending to the LLM
- Truncates the diff to ~8k tokens if it exceeds the budget; adds a note to the LLM if truncation occurred
- Assembles a single structured prompt with PR info + file list + cleaned diff
- Calls the LLM (claude-sonnet-4-6) with a system prompt defining deployment risk criteria (blast radius, reversibility, infra impact, data safety)
- LLM responds with structured JSON: risk_grade (small / medium / high) + reasoning (2–4 operator-readable sentences)
- Validates the JSON response (grade must be one of the three values) - returns 502 on invalid/empty LLM output
- Returns { "risk_grade": "...", "reasoning": "..." } to the caller (e.g. a Kargo http promotion step)

What AI actually did? (continued)

- Kargo used ’http’ promotion step to POST to **/analyze/diagnosis**
- Accepts an Argo CD application identifier (app_name, namespace, application_namespace)
- Step 1 - Fetches application status via Argo CD MCP (get_application): sync status, health status, conditions
- Step 2 - Short-circuits if healthy: if the app is both Synced and Healthy, returns immediately without calling the LLM - no unnecessary cost
- Step 3 - If unhealthy, makes 2 parallel calls to Argo CD MCP to gather diagnostic context:
  - Resource tree (all managed Kubernetes resources and their health)
  - Application events (Kubernetes events for the app)
- Truncates data with priority budgeting before sending to the LLM (~8k token limit total):
  - App status gets up to 60% of the budget
  - Resource tree gets half of what remains
  - Events fill the rest
- Calls the LLM (claude-sonnet-4-6) with a system prompt instructing it to act as an operator-focused Argo CD diagnosis agent - identifying root cause, naming the specific failing resource, and ending with a remediation hint
- LLM responds with structured JSON: healthy (bool) + diagnosis (1–3 terse operator-style sentences)
- Validates the JSON response - returns 502 on invalid/empty LLM output, 404 if the app doesn't exist
- Returns { "healthy": false, "diagnosis": "Pod crashlooping due to OOM. Increase memory limit." } to the caller (e.g. a Kargo post-promotion health check)

Raiffeisen Bank International

What we learned? / Key Takeaways

1. Sharded Kargo is how you scale GitOps without losing control in multi-tenant environments
   - treat infra promotions with different risk tolerance than app promotions

2. Crossplane v2 fixes the tenant visibility problem – but migration is manual, risky, and at-scale requires custom tooling

3. AI's role isn't magic – it's a second pair of eyes in the moments where humans fail under complexity, not from incompetence

Raiffeisen Bank International

What's Next?

1. Implementing Kargo's custom promotion steps to better integrate with LLM / MCPs
2. AI assisted PR review as a first-class step in the Kargo pipeline
3. Expanding Crossplane migration functions to all v1 -> v2 resource types
4. Optimizing migration function patterns
5. If you are running Crossplane at scale – we want to hear your migration story

Raiffeisen Bank International

Thank you for listening
Questions?

Got Interested? → jobs.rbinternational.com