ARGOCON NORTH AMERICA 2026 · SALT LAKE CITY

When the Controller and the Agent Both Think They're Right

Reconciliation in an agentic cluster

Bert Bullough
Lead Architect, Research Cloud Computing · Idaho National Laboratory

THE PREMISE

The clean story

1 SOMEONE

Hand-mutates the cluster. A hotfix, a kubectl edit, a late night.

2 ARGO

Detects drift. Self-heal reverts the cluster to the approved state in Git.

3 AUDITOR

Nods. Restoring desired state is not a change. Git is the record. Nothing to see.

“Drift correction isn’t a change.” Every GitOps shop has said this sentence to an auditor. It has always been true.

THE ASSUMPTION NOBODY WROTE DOWN

That line only holds if the thing causing drift is a human with fat fingers.

Humans drift by accident. Reverting an accident restores intent. Nobody put that premise in the ATO because it was never false.

THE NEW ACTOR

Enter the agent

A human with fat fingers

- Changes state by accident
- Has no record of why
- Is not watching the cluster
- Does not retry when reverted

An agentic remediation system

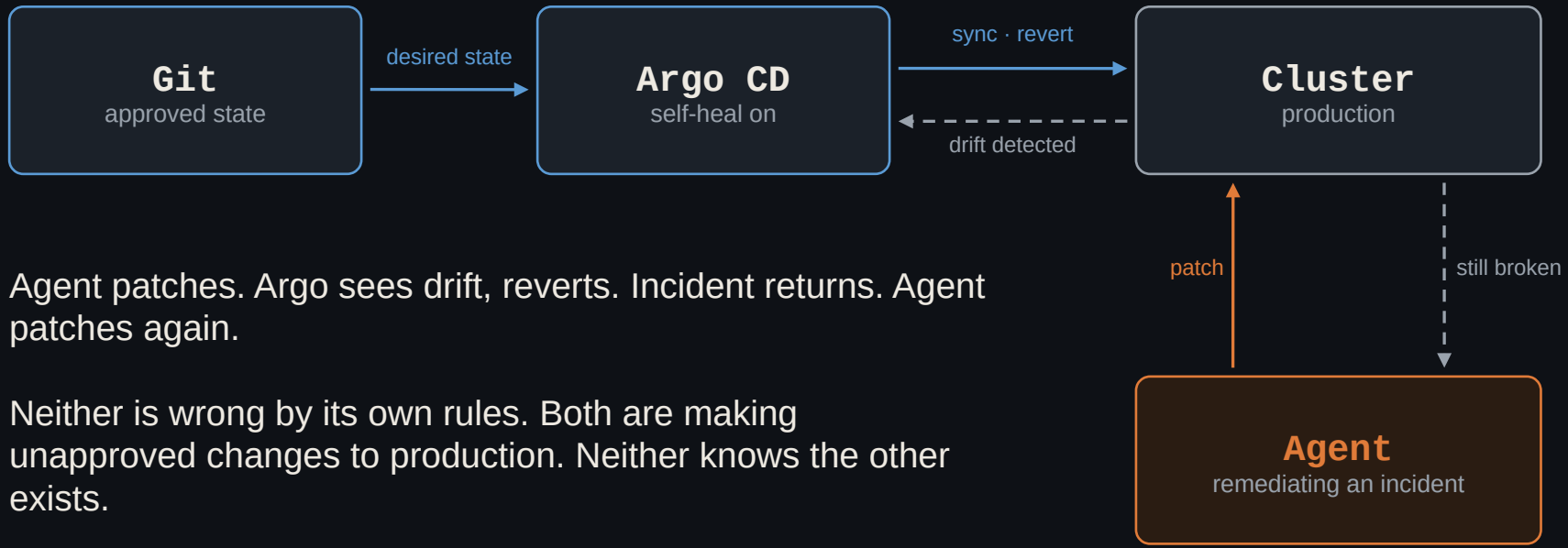
- Changes state on purpose
- Has a reason: an incident it is fixing
- Is watching the cluster continuously
- Retries when reverted, because the incident is still there

That is not drift. That is intent. Argo has one response to both.

“Agent” here means any non-human actor that watches the cluster and writes to it: an LLM-driven SRE bot, an operator-style remediator, a cron job with kubectl. The argument is identical for all three.

TWO RECONCILIATION LOOPS · ONE CLUSTER

The fight



Agent patches. Argo sees drift, reverts. Incident returns. Agent patches again.

Neither is wrong by its own rules. Both are making unapproved changes to production. Neither knows the other exists.

The Flap

Agent patches every 20 seconds. Self-heal reverts every sync. Five clean cycles in 90 seconds.

`Agent:` open → apply → resolved

`Argo:` OutOfSync → Synced → Healthy

Both report success. Repeat until one of them is turned off.

Final state: Git wins. The incident the agent was fixing is exactly where it started.

```
[AGENT]  20:14:19 Patched annotation marker
[ARGOCD] 20:14:22 Synced deploy/demo-app
[AGENT]  20:14:25 Reverted to 'none'

[AGENT]  20:14:39 Patched annotation marker
[ARGOCD] 20:14:42 Synced deploy/demo-app
[AGENT]  20:14:45 Reverted to 'none'

         ··· cycles 3, 4, 5 ···

[AGENT]  20:15:49 5 cycles executed.
```

The loud one. You will catch this in staging. It is the warning shot.

FAILURE MODE 2 OF 3 · SCENARIO 02

The Window Ambush

1. A deny syncWindow is open: the only tool we have for “do not change production right now”
2. Agent patches DEMO_ENV during the window. Argo sees OutOfSync and does nothing
3. Drift sits for two minutes, visible, unresolved
4. Window closes. Argo reverts, at a time no human did anything

Sync windows were designed around humans and maintenance calendars. Point an agent at them and the window becomes a fuse.

```
syncWindow: deny  duration: 2m

[AGENT]  20:20:08 Set DEMO_ENV=production
         20:20:09 App status: OutOfSync
         20:20:15 Sync blocked (deny window)
         ··· drift persists 2 min ···
         20:22:01 Window closed
[ARGOCD] 20:22:33 Synced deploy/demo-app
         20:22:33 DEMO_ENV → baseline

No human acted. No ticket. No trigger.
Timestamp correlates with nothing.
```

A production change with no actor, no ticket, and a timestamp that correlates with nothing.

The Masked Outage

1. Bad config lands in Git: memory limit 32Mi
2. Argo syncs it. Pods OOMKill
3. Agent diagnoses correctly, patches to 256Mi. Outage stops
4. Self-heal sees drift, reverts to 32Mi. Outage resumes

Argo's history says: synced, synced, synced. It does not say a correct fix was applied and undone. The operator investigating the outage cannot see that from the audit trail.

```
application-history.json
{ id: 85, automated, ebb96dc… }
{ id: 86, automated, ebb96dc… }
{ id: 87, automated, ebb96dc… }
{ id: 88, automated, ebb96dc… }
{ id: 89, automated, ebb96dc… } ← revert

status.sync:   "Synced"
status.health: "Healthy"

Agent patched memory to 256Mi.
Self-heal reverted to 32Mi.
Neither action appears in the history.
```

The agent was right. Argo was right. Production was down. The record says nothing happened.

FEDRAMP HIGH · NIST 800-53

In a regulated boundary, that is a finding

Every word Argo has, `drift`, `OutOfSync`, `self-heal`, `prune`, `sync window`, assumes one authoritative controller. There is no name for “two non-human actors disagree about what production should be.” Configuration management controls do not share that blind spot.

Control	What it expects	What the fight produces
CM-3	Changes to the system are approved before they happen	Two actors changing production, neither approved for that change
CM-5	Only authorized actors can make changes	An agent with cluster write, or worse, ungated repo write
AU-10	Actions are attributable to a specific actor	Two logs, two actors, no joined record of one event
CM-6	Configuration settings match an approved baseline	The baseline is whatever won the last sync interval

“Drift correction isn’t a change” was never written into the ATO. The assessor will not accept it once the drift is intentional.

ARBITRATION · THE OBVIOUS FIX

“Just give the agent a commit bit”

Stop the fight by making the agent write to Git instead of the cluster. Argo syncs the commit. Everything goes quiet.

What Argo sees

A commit on main. Synced. Healthy. Textbook GitOps.

What the auditor sees

Approved state in Git. Reconciled. Clean.

What actually happened

A non-human actor changed the definition of approved. Nobody approved it.

Git being the source of truth was never the control. Humans gating what enters Git was the control. Drop the gate and you have laundered a change into approved state.

ARBITRATION · WHAT TEAMS WILL TRY FIRST

Three ways to make the fight go quiet

Each one stops the symptom by removing a control. None of them is a fix.

ignoreDifferences on the whole Deployment

Argo stops reverting because Argo stopped looking. Git no longer governs the thing you actually deploy.

Turn off self-heal in production

The fight ends because one side left the ring. You kept Argo and gave up GitOps where it mattered most.

Give the agent its own Application

Same resources, second Application, second repo. You now have two controllers fighting with two audit trails instead of one.

The auditor will notice which control went missing. The agent will not notice anything; it got what it wanted.

ARBITRATION · WHO WINS

Three ways to settle it

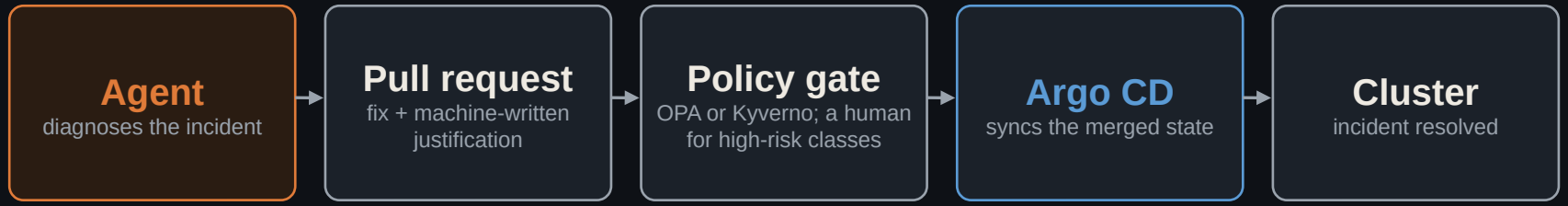
Option	Audit story	What you keep	What you lose
A · Argo always wins Agent is advisory only	Clean. Nothing changes that Git did not approve.	Every existing control	The agent. It is now an alerting system with opinions.
B · Agent wins in scope ignoreDifferences carve-outs	Full of holes. Each carve-out is a field Git no longer governs.	A working agent, fast	Git as source of truth, one field at a time. Scope creeps.
C · Agent commits, Argo enforces **Agent opens PRs; policy gates merge**	Singular. Git is the only record; the agent is a committer, not a controller.	One trail, one authority, a working agent	Latency. A policy-authoring burden most teams underinvest in. The agent cannot act on anything the policy engine cannot evaluate.

B is what most teams will do. It is the wrong answer dressed as pragmatism. C costs more than it looks, and it is still the only one with one trail.

ARBITRATION · WHO LOGS

The agent is not a second controller

It is a committer. Git stays the one record. Argo stays the one enforcer.



Who wins

Policy. Neither controller decides.

Who logs

Git. Agent identity and intent live in the commit.

How the trail survives

There is only one trail. Two controllers collapse back into one.

Some incidents cannot wait for a merge

So give the agent a bounded emergency path. Bounded means: declared in Git first, time-limited, attributed, gated by policy, and reconciled either way.

`t = 0`

Agent commits a self-heal suspension for that Application, with an expiry. Then patches live. Then opens the PR.

`0 < t < 15 min`

Argo does not revert. The deviation is a logged event with an owner, an intent, and an expiry. The PR is under review.

`t = 15 min`

Merged: Argo syncs, deviation closed. Not merged: suspension expires, Argo reverts, deviation stands as a bounded exception.

The gate still applies. The agent may self-suspend only for incident classes pre-approved in policy. The suspension commit is attributable and alerts on open. Without that, this is Option B with a timer on it.

WHAT GOOD LOOKS LIKE

The same outage, with one trail

Today

```
application-history.json
{ id: 88, automated, ebb96dc… }
{ id: 89, automated, ebb96dc… }

status.sync:   "Synced"
status.health: "Healthy"

# agent patch: not here
# self-heal revert: not here
# who, why, when: not here
```

Proposed

```
application-history.json
{ id: 88, automated, ebb96dc… }
{ id: 89, automated, 3f1a9c2…,
  initiatedBy: { kind: Agent,
    name: remediator-01, pr: 4821 } }

events:
DeviationOpened  20:31:04 remediator-01
  reason: OOMKill 32Mi→256Mi  expires: 20:46
DeviationClosed  20:39:12 PR #4821 merged
```

Same agent, same fix, same Argo. The difference is that the record can answer “who changed production and why” without joining two logs.

THE ASK

What Argo needs is not a second controller

It needs a first-class concept of a non-human committer with a bounded emergency path. Today you assemble that from pieces built for something else.

```yaml
# Application spec, proposed
spec:
  syncPolicy:
    automated:
      selfHeal: true
      selfHealSuspension:
        requestedBy: agent/remediator-01
        reason: "OOMKill; PR #4821"
        expiresAt: 2026-11-18T20:46:00Z
        policyRef: emergency-memory-limits
```

Actor identity on sync

A sync caused by an agent commit says so in Argo's own history (initiatedBy), not only in the Git author field.

Time-boxed self-heal suspension

A declarative, expiring exception on an Application, with the requesting actor, the reason, and the authorizing policy.

Deviation as an event

DeviationOpened, Closed, and Expired as distinct audit events, exportable, so one trail can be reconstructed without two logs.

None of this is exotic. All of it is missing. The field names are a proposal; the shape is the ask.

18 MONTHS

“Drift correction isn’t a change” was always a statement about who was drifting.

Agentic remediation is in evaluation at most Argo shops today. Within 18 months it will be in production inside a regulated boundary, and the first assessor to ask “who approved that sync?” turns this talk into a finding. If you run Argo, this is already on your roadmap. You just have not found it yet.

Bert Bullough
bert@bertbullough.com · github.com/dirtmerchant

Questions