WSO2CON AFRICA

SEPT 22 – 24, 2026 | NAIROBI, KENYA

Accelerating the SDLC with Harness Engineering

Building the guides and sensors your agents can't bypass.

Tishan Dahanayakage
Director & Head of Engineering
Choreo BU WSO2

Isuru Wimalasundera
Associate Enterprise Architect
Sales Engineering WSO2

The horse, the cart and the tack

A harness doesn't make the horse stronger. It decides where the power goes.

Traces
MCP tools

Blinkers
Agent identity + `filterByAuthz`

Reins
Logs, alerts, validation messages

Fence
CEL rules in the CRD

Agent = Model + Harness

"Everything in an AI agent except the model itself."

Birgitta Böckeler · LangChain · 10 Mar 2026

TRACES
Which tools it can reach

BLINKERS
What it's allowed to see

REINS
How it learns it was wrong

FENCE
What it simply cannot do

You didn't train the model and you can't make it smarter. All four of these are yours.

The discipline

"Anytime you find an agent makes a mistake, you take the time to engineer a solution such that the agent never makes that mistake again ."

Mitchell Hashimoto · 5 Feb 2026

A loop, not a config file. You'll watch it run twice today.

The constraint moved downstream

Capability didn't follow. That downstream is your platform.

WRITING CODE

~96%

SWE-bench Verified
Saturated. Solved.

RUNNING IT

47%

ITBench-AA SRE
Every frontier model under 50

29%

OTelBench
Observability

19.4%

IaC-Eval
Terraform

35.2%

InfraBench
Cleanup

Same models. A third to a fifth of the score, the moment the task is running code rather than writing it.

Sources: SWE-bench Verified leaderboard · ITBench (IBM Research, 2025) · OTelBench · IaC-Eval (NeurIPS 2024) · InfraBench. Figures as published at time of writing — verify before the day.

And it's already showing

FAROS AI · TEAMS ADOPTING CODING AGENTS

+16.2%

PR merge rate
More code written

−11.7%

Deployments per week
Less of it shipping

+242.7%

Incidents per PR
More of it breaking

CIRCLECI

−7%

main-branch throughput

70.8%

build success — a five-year low

More code written. Less of it shipping. More of it breaking.

Sources: Faros AI, *The AI Productivity Paradox* (2025) · CircleCI, State of Software Delivery. Verify figures before the day.

Harness variance now rivals model variance

QWEN2.5-CODER-7B · TERRAFORM

14.0% → **62.9%**

From a **staged verifier loop** alone. Same model.

GPT-4O · WITH A POLICY ORACLE

93.0%

Nobody retrained anything. **The harness did that.**

Both papers found the effect is larger on weaker models. In this room, that's a feature, not a caveat.

Sources: staged-verifier and policy-oracle Terraform results from the IaC-Eval follow-up papers. Verify citations before the day.

Guides and sensors

Two questions about any piece of harness: does it act **before** the agent (a guide) or **after** it (a sensor) — and is the judgement made by **code** or by a **model**?

> "Separately, you get either an agent that keeps repeating the same mistakes, or an agent that encodes rules but never finds out whether they worked."

Böckeler's 2×2. Every strap today gets a square.

	Computational code decides	**Inferential** a model decides
Feedforward guides — before	**Rules that act before the object exists** Code refuses the request or the result outright. **e.g.** CEL validation in the CRD, deny-by-default roles, the tool budget	**Context the model reads and reasons over** Guidance it can follow — or ignore. **e.g.** skills, runbooks, MCP tool descriptions
Feedback sensors — after	**Deterministic checks on what happened** Facts the agent cannot argue with. **e.g.** build logs, validation errors, health probes	**A model judging the outcome** Useful, and never load-bearing. **e.g.** an agent reviewing its own diff

The claim

Controlling what an agent is allowed to **ask for** is not the same as controlling what is allowed to **exist**.

HOW EVERYONE DOES IT TODAY

Check the request

Every shipped agent-governance system gates the ask: may this agent call this tool, with these arguments? Block the ask and you close that one path — a different tool, a new prompt or a human console leaves the same door open.

> The agent asks for 30 replicas. The tool call is denied.

WHAT WE ARE CLAIMING

Check the result

A rendered-output validation rule inspects the object the platform is about to create, after every template and default has been applied. It does not care who asked — any tool, any agent, any prompt, any human. If the result breaks the rule, the result does not exist.

> Anything that renders 30 replicas is rejected, whoever asked for it.

In the next eighty minutes an agent will build a two-project payment system on an empty cluster, break production, and then fail to break it again.

Demo Usecase

mopay

Merchant collections and settlement, Kenya and Nigeria. Merchants take payments over mobile money and card; the system aggregates, keeps a double-entry ledger, settles on a schedule, and shows merchants what they collected and when they get paid.

WSO2

The map

Developer-owned Platform-owned

project · mopay

- mopay-console
- collections-api
- settlement-worker

ledger · sole DB owner → mopay-db

NAMESPACE

GET
/v1/merchants/{id}/standing

project · merchant-platform

Different team

merchant-api → merchant-db

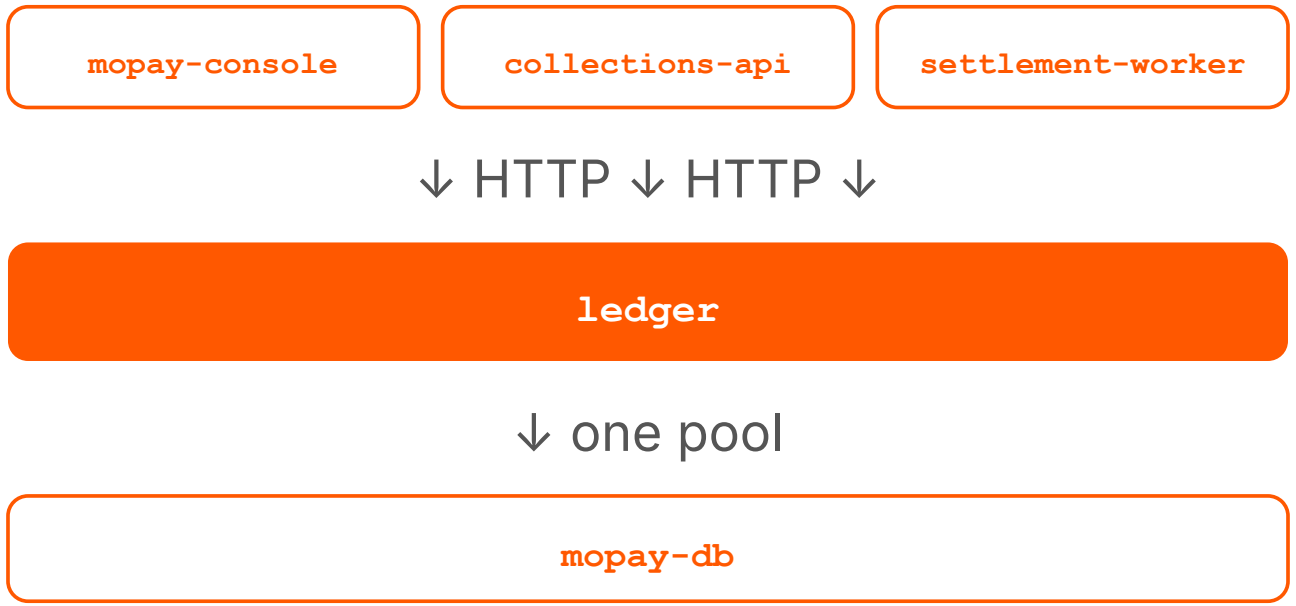
collections-api crosses the boundary by endpoint only. It never touches merchant-db.

environments Platform-owned. Same artifact, four targets. Cross-project resources: unsupported. Cross-project endpoints: first-class.

development → staging → prod-ke prod-ng

One component holds a connection

`ledger` only. Everything else reaches data through it — zero pgx imports anywhere else.



Post-render validation sees one component's rendered resources and nothing else. One pool makes the ceiling expressible.

Remember this number

27

`mopay-db` is configured `max_connections = 30`.
Postgres reserves three for superusers.

You get 27.

- A literal in the `ResourceType` manifest. Platform-owned.
- Not a parameter. Not an output.
- No developer can read it. Nothing in the system will ever tell the agent this number.

LIVE OR RECORDED 0:14 · Agentic Engineer · 5 min

That scenario, built by a harness we already run

Agentic Engineer produced mopay — the system on the last four slides. It does not stop at design: **requirements through to a deployed, running system**, the whole cycle.

Experimental

Unsupported — not a prototype that stops at diagrams. Those are different claims.

Internal

A harness our own team built around OpenChoreo, for our own use.

Not a product

Not installable today. Shown as an aspiration, not an offer.

> "It is where we think this ends up — the shape we would like every OpenChoreo user to be able to reach."

How it got there — and where we stop watching

1
Requirements chat
Collections, ledger, settlement, two countries.

→

2
Discovers `merchant-api`
In another project. Reads its contract from the platform catalogue.

→

3
Designs the call
`GET /v1/merchants/{id}/standing` — not a duplicate merchants table.

→

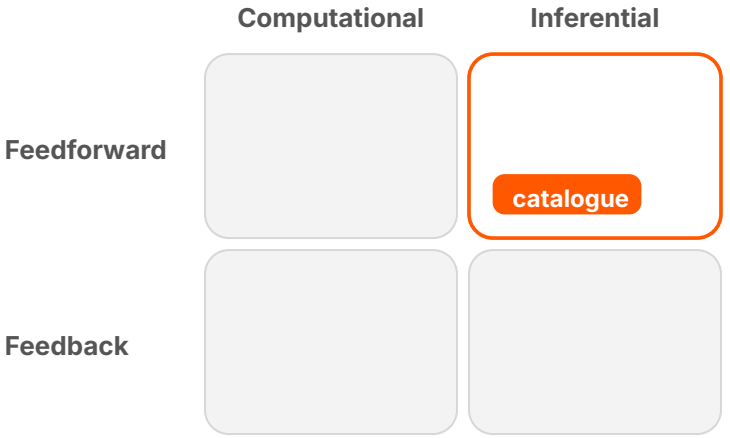
4
Commits the design
Domain model, wireframes, specs. **We stop watching here.**

It is still going. It does not need me — which is rather the point.

The run carries on by itself — builds, deploys, runs — while the rest of the session happens. We stop *watching* because the next hour is about how a harness like that is built, not about watching a finished one work.

The catalogue steered the design before a line of code existed. Without it you would have two sources of truth for who a merchant is — and find out in six months.

On the 2×2



Inferential feedforward — the only guide in the lab that acts at design time.

FROM HERE ON

Ours is internal, and you cannot have it. What you can have is the shape.

Agentic Engineer → the same shape, on Claude Code against the same system

Starting with the one thing that makes any of it possible: giving the agent the platform's own tools.

If you want to follow along

Two repositories, two colours. Everything we build today lands in one of them.

APP REPO · DEVELOPER-OWNED

`wso2con / 2026-NBO-Internal-Developer-Platform-tutorial-2`

github.com/wso2con/2026-NBO-Internal-Developer-Platform-tutorial-2

OPS REPO · PLATFORM-OWNED

`wso2con / 2026-NBO-Internal-Developer-Platform-tutorial-2-Ops`

github.com/wso2con/2026-NBO-Internal-Developer-Platform-tutorial-2-Ops

SETUP Installing the MCP servers on Claude Code · 0:19

Two commands, and the agent can reach the platform

Claude Code, no MCP server, zero tools. Two servers go on: the control plane, and observability.

1 · Control plane — 130 tools

```
$ claude mcp add --transport http \
    --client-id user_mcp_client --callback-port 8075 \
    openchoreo-cp <control-plane>/mcp
```

Projects, components, builds, deployments, promotion. Everything the agent can **ask the platform to do**.

✓ openchoreo-cp · connected · 130 tools

2 · Observability — 11 tools

```
$ claude mcp add --transport http \
    --client-id user_mcp_client --callback-port 8076 \
    openchoreo-obs <observability>/mcp
```

Runtime logs, pod state, metrics. Everything the agent can **see for itself** — the sensors.

✓ openchoreo-obs · connected · 11 tools

PER SERVER

BEFORE 0 tools → **AFTER** 141 tools

browser → OAuth consent → token back → `/mcp` connected

callbacks on `:8075` / `:8076`

Two commands, and the agent went from no reach at all to more surface than it can hold in context. That is the next problem.

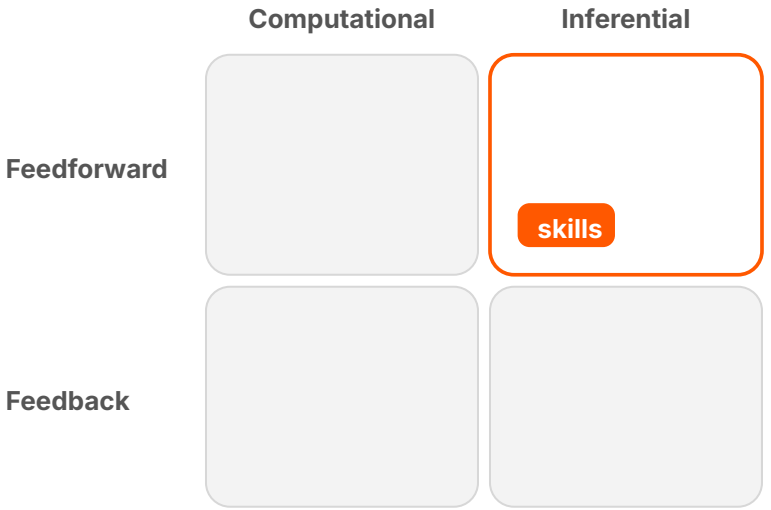
LIVE Wire the agent up · 0:21

The skills

This is your runbook. Not a document about your runbook — the runbook, as data the agent reads.

```
$  ls ~/.claude/skills/
openchoreo-deploy/      openchoreo-promote/
openchoreo-diagnose/    openchoreo-conventions/
```

On the 2×2



A guide the model reads and reasons over, before it acts.

Straps 1 and 3 light 1 tools 3 skills

What the platform team already built

Two views in the console — and only these two.

TWO PROJECTS

- mopay
- merchant-platform

Two teams, two bounded contexts.
Developer-owned.

THE PE CATALOGUE

ComponentTypes	managed-service, scheduled-task
ResourceType	managed-postgres
Traits	observability-alert-rule, api-configuration
Environments	development, staging, prod-ke, prod-ng
Pipeline	mopay-regional

All of this already exists. A platform team created the projects and authored the types before anyone wrote a line of mopay. That work is real and it is someone's job — it is just not the delivery cycle we are here to watch.

LIVE 0:24 · derive the descriptors · ~5 min

The repo, and what is not in it

lab-start · FILE TREE

```
demo-usecase/
├── mopay-console/       Dockerfile  src/
├── collections-api/     go.mod  cmd/  internal/
├── ledger/              go.mod  cmd/  internal/
├── settlement-worker/   go.mod  cmd/  internal/
└── merchant-api/        go.mod  cmd/  internal/
    └──
workload.yaml          # the only one
```

THE PROMPT

```
This is a brownfield onboarding. Five services live in
this repo under demo-usecase/.

merchant-api already runs on OpenChoreo and has a
workload descriptor checked in. The other four —
mopay-console, collections-api, ledger and
settlement-worker — have none.

Produce an OpenChoreo workload descriptor for each of
the four.
Derive everything from the application source.  Do NOT call
list_components, get_component or
get_workload.

For each descriptor tell me what evidence you used,
and say plainly where you guessed.
```

This is the normal case. The code exists, it works, and it describes itself to a compiler and to nobody else.

4m 53s four descriptors, no retries · measured 21 Sept

What the descriptor says

A container image says how to build and what to run. It does not say what port the thing listens on *as an endpoint* , who may reach it, what it depends on, or which values differ per environment. That gap is the descriptor.

Field	Comes from	What it settles
`endpoints[]`	the source's own routes and ports	who may call it
`dependencies.endpoints[]`	how the code reaches other services	resolved by the platform, never hardcoded
`dependencies.resources[]`	the database the code opens	credentials injected, never written down
`configurations.env[]`	`internal/config/config.go`	the one place each service reads its environment

Two colours, two repositories

Developer-owned

Source code · Project · Component · Workload · Resource · WorkflowRun · ComponentRelease · ReleaseBinding

Outlined: created via API at build and deploy time, not checked in.

`component.yaml` and `workload.yaml` live beside the source, in the **app repo**.

Platform-owned

ComponentType · ResourceType · Workflow · Environment · DeploymentPipeline · AuthzRole · validation rules

The types, the pipelines and the rules live in an **ops repo the agent cannot push to** .

The fence isn't a setting. It's a different repository with different access.

What it was handed, and what it had to work out

Handed

- The project names — `mopay` and `merchant-platform`
- The endpoint names already in use on the platform

A dependency that names an endpoint wrongly cannot resolve — and a live failure there teaches nothing.

Its own work

- Every visibility
- Every dependency — including the one across the project boundary
- Every environment variable, and which one is per-environment
- The component shape each descriptor implies

Three judgements worth naming

Not the syntax — the decisions. Everything below came out of reading the application source, never the running platform.

`ledger`

`visibility: [project]` and nothing wider. It cites the source's own comment, and corroborates with `docker-compose` using `expose:` rather than a published port.

A security boundary, recovered from two comments in a repo.

`settlement-worker`

No endpoints, so it becomes a `cronjob`. The descriptor picks the component type.

A service type would have been rejected — it demands an endpoint.

`collections-api`

`namespace` visibility and `project: merchant-platform` — the only shape that works across a project boundary.

No hostname written anywhere — the address is injected at deploy.

The constraint *is* derivable — because the codebase wrote it down. That is the argument for writing intent next to the code.

One descriptor, on screen

`settlement-worker/workload.yaml` — the next ten minutes hinge on it.

```
kind: Workload
metadata:
  name: settlement-worker
  project: mopay
spec:

# no endpoints: block — nothing listens   dependencies:
    endpoints:
      - component:
ledger          name: api
        visibility: project
        env:
          - name: LEDGER_BASE_URL
            from: address
  configurations:
    env:
      - name: DATA_REGION
        value:
""
```

No `endpoints:` block

That absence decides the component type. `managed-service` refuses a workload with no endpoint, so the descriptor chooses `scheduled-task` — not the reverse.

One dependency

`ledger / api` at project visibility, bound to `LEDGER_BASE_URL` . No hostname anywhere. **Say the name out loud.**

`DATA_REGION` declared and empty

Every other value set. The empty one is the per-environment slot.

LIVE 0:35 · deploy, fail, self-correct · 10 min

“Three of these are already deployed.”

Now say it — not earlier. In the interest of time, `ledger`, `collections-api` and `mopay-console` were built and deployed before the session.

`merchant-api`	other team
`ledger`	already live
`collections-api`	already live
`mopay-console`	already live
`settlement-worker`	built live

The console is the result of work they watched
The room has already read the descriptors those components came from — not a setup they were asked to accept.

Three identical builds teach nothing the fourth will not
Each build is ~3 minutes of a progress bar. Three of them live costs ten minutes.

> Then hand the fourth to the agent: create the component, build it, deploy it to `development`.

Build watch — how a build happens here

1
Agent creates a `WorkflowRun`
Picks a Workflow the ComponentType allows; supplies repo, branch, path.

→

2
Argo runs it in the workflow plane
A multi-stage template the platform team authored — not the developer.

→

3
Image pushed to the registry
Tagged and recorded against the component.

→

4
Workload generated from the descriptor
`generate-workload-cr` — `workload.yaml` becomes the Workload CR.

→

5
A `ComponentRelease` is cut
Frozen snapshot. The binding deploys it.

WORKFLOWS ALLOWED BY `managed-service`

`gcp-buildpacks-builder` AGENT PICKS
No Dockerfile needed — detects the language and builds the image.

Go services have no Dockerfile in the repo, so this is the only one that fits.

`docker-builder`
Builds from a Dockerfile you wrote.

What `mopay-console` uses — ~1m 40s, half the time.

The agent doesn't choose freely. The ComponentType lists the workflows; the repo decides which one applies.

THE TRAP

```
paketo Go buildpack   caps at  go 1.25.6 every go.mod
says     go 1.26 BP_GO_VERSION          cannot rescue it
→  gcp, not paketo
```

No agent can discover that. It's on a slide — and a slide is a harness artifact too, one with a human in the loop.

3m 30s measured for settlement-worker (21 Sept: 3m 35s · 3m 35s · 2m 57s).

LIVE 0:39 · deploy to development

The build was green. The deployment isn't.

CONTROL PLANE

```
status: ReleaseSynced
pendingConnections:
  - component:
ledger-api , endpoint: api
    reason: "ReleaseBinding not found
      for component mopay/ledger-api"
```

DATA PLANE

```
settlement-worker failed:
  invalid configuration:
  - LEDGER_BASE_URL: required but not set
# the CronJob still renders, and still fires
```

A presenter typed `ledger-api` before the beat — there is no component by that name. Nothing validates a dependency's `component:` at build time, so the bad reference goes into the Workload untouched. **The platform and the application agree, independently** — and unlike a bad env value, this binding never says `Ready=True`.

reads `get_release_binding` + logs →

fixes the **descriptor** back to `ledger` → redeploys · ~15 s →

rows_processed=287

Live-first: `update_workload` , then commit the descriptor to match. Source-first needs another build — prompt for live-first if behind.

Two witnesses, and why it self-corrects

The platform names the object it could not find; the application names the variable it needed. Either alone is actionable — together they tell a typo from a target not deployed yet. The second message exists because a human wrote it. `internal/platform/config.go` accumulates every problem rather than failing on the first —

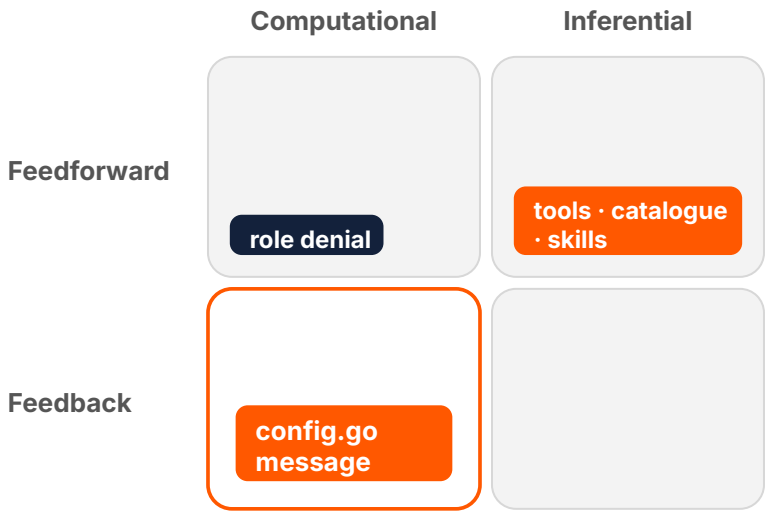
> *"so a misconfigured deployment reports every missing variable at once instead of one per restart."*

One function. An agent corrected its own mistake without asking anyone.

Control group. Same agent, same session — in twenty minutes something else breaks and it will have no idea.

Strap 4 lights 4 **sensors**

On the 2×2



Ready does not mean working

A crash-looping container **flaps** Ready.

A stable one can be Ready and **bound to the wrong variables** .

The proof comes from the process, not the tool

```
settlement run completed  rows_processed=287  merchants_settled=7  duration_ms=165
```

Green in the tool is a **claim**. The log line is **evidence**.

Development is whole: four mopay components deployed, the fifth settling on a schedule, every descriptor in the repo — and the only human edit in the last ten minutes was the mistake.

LIVE 0:45 · promote

Promote

Same `ComponentRelease`, same image, four environments. Only configuration differs.

	development	staging	prod-ke	prod-ng
`DATA_REGION`	KE	KE	KE	**NG**
`ledger · dbMaxConns`	10	10	10	10

There is **no approval gate** . The agent can only promote along edges a platform engineer declared.

Unattended, in seconds. Nothing in the platform slows it down — and for the next ten minutes that will look like the best thing you have seen today.

LIVE 0:47 · the incident

Month-end is slow

The agent raises **ledger's** `dbMaxConns` from **10** to **30** — a component it wasn't deploying. Individually reasonable: raising a pool is the obvious response to queueing at peak.

10 → 30

against a database that has 27

ledger's own config comment, verbatim

*"This component supplies the MECHANISM — a default and an environment override. It deliberately enforces **no ceiling of its own** … **A component that polices its own limits leaves the platform nothing to enforce.** "*

The agent did everything right and still broke it.

LIVE 0:49 · load — start it before you explain it

```
$ cd demo-usecase && ./scripts/load-platform.sh development
```

130

concurrent

40

seconds

Against the external URL, through the gateway — `collections-api` is the only external endpoint. The script prints the UTC window either side: **write it down.**

130, not 80. Eighty was a Compose figure — over the gateway it only touches the wall without sustaining it.

What is already watching: three rules, twelve instances

Traits on the **Component**; per-environment enablement and channels on the binding. Created before the session, with the components.

Instance	Watches	Condition
`collections-refused-upstream`	`collections-api`	≥5 / 5m
`ledger-db-capacity`	`ledger`	≥1 / 5m
`settlement-lag-critical`	`collections-api`	≥1 / 5m

The shape of the failure

One fault, three witnesses. Each component sees a different piece of it — and only one of them is where it broke.

mopay-db

WHERE IT BROKE

Postgres has 30 slots and keeps 3 for superusers. The 28th connection is refused — the database is doing exactly what it was configured to do.

```
FATAL: remaining connection slots are
reserved for roles with the SUPERUSER
attribute (SQLSTATE 53300)
```

ledger

THE CAUSE

Its pool was told it may open 30. It reaches 29 and can never fill. A pool that cannot fill, sitting next to a database that is refusing.

```
pool: max=30 total=29 acquire:
connection refused by server
ledger-db-capacity → firing
```

collections-api

THE SYMPTOM

Never touches the database. It calls ledger over HTTP, gets refused, and returns errors to merchants. It is the loudest — and it is not where anything broke.

```
POST /v1/collections → 503
upstream=ledger status=refused
collections-refused-upstream →
firing
```

You asked for 30. You got **27** . The number on the tin is not the number you get — and there was no way for the agent to find that out.

The component that screamed is not the component that broke.

LIVE 0:58 · the SRE agent

Found in ninety seconds. Cannot fix it.

1

The alert triggers it

It carries `triggerAiRca` . RCA starts with no human in the loop.

2

Clears the obvious suspect

`merchant-api` — the cross-project dependency — is eliminated first.

3

Finds the cause

`query_component_logs` surfaces the ledger capacity line. Pool exhaustion, named.

4

Recommends. Cannot act.

It proposes a rollback. You roll back by hand, on screen.

ITS ROLE

`15 × *:view` `incidents:update` and nothing else

Say *it recommends* — never *it applies fixes*.

CONTROL GROUP, REVISITED

Twenty minutes ago the same agent fixed itself in one turn off a message a human wrote. This time nothing wrote it anything. **Same agent, same session. The difference is entirely what we built around it.**

The gap between detection and action is what we're closing — with a fence, not by handing it the keys.

4 sensors · fully lit

So take production away — and why we are not going to

> "The obvious enterprise response is to take production off the agent. Scope its role, drop the deploy tool, put a human in front of every change. And that works — the agent stops breaking production, because it stops touching production. It also stops being useful: it diagnoses in seconds and then waits hours for someone to paste its fix. That is where the efficiency dies, and most enterprises will land there, because it is the only control they have."

DORA'S VERIFICATION TAX

Described rather than demonstrated. RBAC alone doesn't solve anything — it makes the agent a slower human.

STRAP 2 · NAMED, NOT STAGED 2 blinkers

`filterByAuthz` removes a tool from `tools/list` rather than failing the call. Real, and on this platform — but a live revoke costs eight minutes to prove something the room already believes.

The fence is what lets you keep the power switched on. Everything up to here was setup.

LIVE 1:05 · the fence · (a)

The naive rule

ClusterComponentType **managed-service** · **platform-owned** · `spec.preRenderValidations[]` · **ops repo** `component-types/managed-service.yaml`

```
${environmentConfigs.dbMaxConns <= 20}
```

It looks right.

It is also the rule the repo's own `FAILURE-SCENARIOS.md` reaches for first.

The agent beats it without trying

1. It **complies** — sets `dbMaxConns: 20`.
2. It **scales to 2 replicas**.
3. The rule passes. It deploys.
4. **It breaks again.**

$$2 \times 20 = 40$$

against 27

Your first guardrail was wrong, and the agent found the gap without trying to. It wasn't being adversarial — it was solving the problem you gave it, inside the constraint you gave it.

Guardrails have bugs too.

LIVE 1:08 · the fence · (b)

The rule that holds

```
${size(dependencies.resources) == 0 ||
  environmentConfigs.
replicas * environmentConfigs.dbMaxConns <= 27}
```

< 5 s

Rejected at render, quoting the rule in full alongside the message.

`ResourcesReady=True`

The previous render keeps serving. A rejected change refuses to advance rather than taking the service down.

Commit it to Ops

A rule that lives only in a cluster is one namespace wipe from gone. It goes in `component-types/managed-service.yaml` — the repo the delivery agent cannot write to.

A release freezes the type

Writing the rule changes **nothing already deployed**. The order is:

1
Write the rule
in the ComponentType

→

2
Cut a new ComponentRelease
the snapshot now includes the rule

→

3
Repoint the binding
per environment

Third time this has mattered today.

A release is a photograph, not a pointer.

The message is the only channel

The capacity is **deliberately invisible** — a literal in a `pe` -only template. Not a parameter. Not an output. The agent cannot look up why the number is 27. Neither can the human at 2am.

~~look it up~~ ~~read the type~~ ~~ask the binding~~ **read the message**

COMPILER-FEEDBACK STUDY

+5.3 to +79.4 pp

compilation success, from a better error message alone

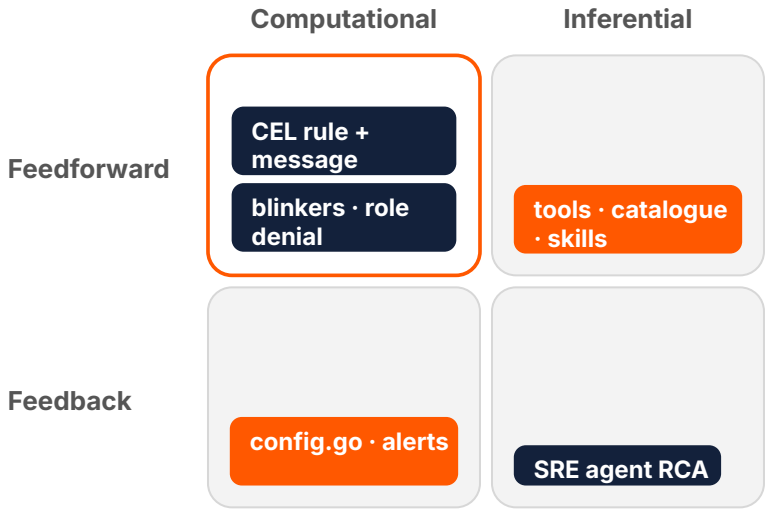
QWEN3-4B

18.0% → 97.4%

Same model. Same rule. **Different sentence.**

Strap 5 lights 5 **fence**

On the 2×2



It acts before the object exists.

1:25 · close

The repo, complete.

```
demo-usecase/                          # app repo · developer-owned
├── mopay-console/
│   component.yaml   workload.yaml
├── collections-api/
│   component.yaml   workload.yaml
├── ledger/
│   component.yaml   workload.yaml
├── settlement-worker/
│   component.yaml   workload.yaml   # typo + fix
├── merchant-api/
│   component.yaml   workload.yaml
└── resources/
    mopay-db.yaml   merchant-db.yaml
ops/                                   # the agent cannot push here
└── component-types/
    └── managed-service.yaml
        preRenderValidations[1]   replicas × dbMaxConns ≤ 27
        message:
the sentence the room wrote
```

SIX STRAPS

1 · tools	traces
2 · blinkers named — identity + `filterByAuthz`	
3 · skills	the runbook as data
4 · sensors	logs, alerts, messages
5 · fence	a CEL rule in the CRD
6 · autonomous	promotion, unattended

> The honest counter is `git log` on the ops repo — a repository the agent cannot push to.

DORA

"When platform quality is high, the effect of AI adoption on organizational performance becomes strong and positive . Conversely, when platform quality is low, the effect of AI adoption on organizational performance is negligible."

AI doesn't fix a team; it amplifies what's already there.

Agents you demo, versus agents you trust in production.

WSO2CON AFRICA

SEPT 22 – 24, 2026 | NAIROBI, KENYA

Thank You!