KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

The Seams Are the Trap: Agent Fleets from CNCF Parts You Already Have

Salt Lake City, Utah, United States

Aleksander Slominski





Mariusz Sabath

KubeCon CloudNativeCon
North America 2026

Every vendor is shipping an agent platform. You probably don't need one.

What you need is to know how the parts you already run fit together — and where they do not.



Can I run AI agent fleets using Kubernetes parts?

You already run the parts.

The hard bit is not scaling agents but it is the four places the composition leaks and knowing which of your own control claims are not true yet.

- A platform is the right answer for some teams. This talk is not anti-vendor.
- The claim is that composition is sufficient and not that products are bad.
- The four seams are the take-home artifact. Two of ours are still open.

The problem

AI agent fleet, not a collaboration. Many non-deterministic agents, running safely, at a scale where you cannot babysit them.

- These agents do not talk to each other: there is no agent-to-agent protocol here
- They are consumers of event logs, which is what makes state survive the process
- Non-deterministic: constrain what an agent can do and do not predict what it will do
- Bursty: idle capacity is waste, and a cold start is latency
- Untrusted by default: an agent is a program someone else's model wrote

> **100 agents, fanned out and tracked to completion in one minute.**

KubeCon CloudNativeCon
North America 2026

Guiding principles



Seven principles

Composition over framework.

PRINCIPLE	WHAT IT RULES OUT	EVIDENCE IN THE DEMO
Scalable to zero	always-on agent pools	idle at 0 replicas; wake 2–4 s
Existing technologies	a new runtime to learn	KEDA, Strimzi, CloudEvents — all stock
No new platform	a framework with our name on it	two services + a wire contract
Trusted agents only	“the broker is inside the perimeter”	forged response refused and inert
Authorized users only	an open submit endpoint	401 → sign in → 202, submitter on the event
Auditable after the fact	trusting a dashboard	completions chain to signed wakes
Fails closed	a half-configured control	when no keyset configured then refuse to start

The building blocks

Two CNCF projects and one spec — all of it stock, none of it ours.

Strimzi / Kafka — the durable event log, run on Kubernetes by an operator

One partition to one consumer, so per-conversation ordering holds across pods

cncf.io/projects/strimzi

KEDA — the autoscaler; replicas follow consumer lag, 0..N and back to zero

Idle costs nothing: 0 replicas at rest, and a wake measured at 2.1–4.2 s

cncf.io/projects/keda

CloudEvents — the envelope; one event shape for wakes, responses and completions

CloudEvents 1.0 in binary mode over Kafka headers

cloudevents.io

Events scaling lag means “work not finished”, not “work not read”.

The Kafka offset is committed only after the terminal event.



Scaling and long-running agents

One decision to commit the offset **after** the terminal event settles both problems at once.

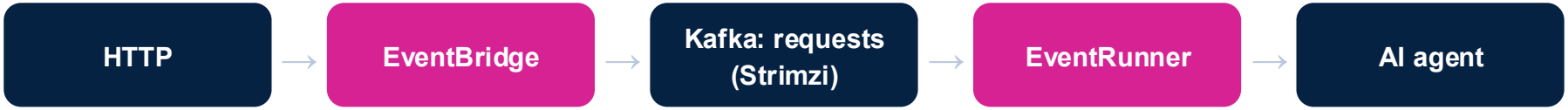
- Consumer lag becomes a correct queue gauge, so KEDA can scale on it
- It is structurally impossible for KEDA to scale away a pod mid-stream

> **The fix for “never miss an event” is also the fix for “don't kill a running agent.”**

MEASURED	VALUE	NOTE
Wake, 0 → 1	2–4 s	most time of it is the broker's rebalance delay
Back to zero	30–31 s	cooldownPeriod: 30

Architecture — two services, four stock projects

WAKE PATH: from input to running agent



RESPONSE PATH: from agent output to audit trail



KEDA watches consumer lag on Kafka: requests and scales EventRunner 0..N

ours — two services

stock — Strimzi, CloudEvents, KEDA, the agent itself

Both paths carry CloudEvents 1.0 and terminal events are signed with a detached JWS.

What is stock, what is ours

LAYER	WHAT	WHOSE
Scaling	KEDA ScaledObject, 0..N on consumer lag	stock
Transport	Strimzi-backed Kafka, binary CloudEvents	stock
Envelope	CloudEvents 1.0 + detached JWS (Ed25519)	stock spec
Front door	EventBridge: HTTP, store, views, auth	OURS
Consumer	EventRunner: spawns the agent, emits events	OURS

A Deployment scaled 0..N, not a Job per request: Kafka already assigns one partition to one consumer, so per-conversation ordering holds across pods for free.

KubeCon CloudNativeCon
North America 2026

The four seams



Four named integration points where the composition leaks

SEAM	WHERE IT LEAKS	OUR STATUS
Cold-start identity gap	the agent egresses before SPIRE issues an SVID	OPEN — no SPIRE; the upgrade path
Trust-scope drift	egress allowlist decided at deploy, stale by wake	PARTIAL — enforced, no live check
Unverified delivery	the broker hands off events before signature check	DONE — both directions, on a cluster
Unbound audit	completion not chained back to the signed wake	DONE — ce_causationid, verified

Scale-to-zero is a trust argument, not just a cost argument

- The event runner reads its egress allowlist at startup
- Scale-to-zero means no long-lived process is holding a stale decision
- Edit the list while replicas are 0 and the next wake enforces the new one

How it works — end to end

Seven steps. Step 4's ordering is the one that matters.

1 **Submit**
POST /v0/agents — GitHub sign-in verified, ce_submitter stamped

2 **Sign**
detached JWS; the submitter is a signed attribute

3 **Wake**
lag ≥ 1 on requests; KEDA scales 0 → 1

4 **Verify**
signature first, then the submitter allowlist

5 **Run**
every stdout frame becomes a response CloudEvent

6 **Bind**
terminal event signed, carries ce_causationid back to the wake

7 **Zero**
offsets committed, lag 0, KEDA removes the replica

Signature before allowlist, verify first, then trust the attribute.

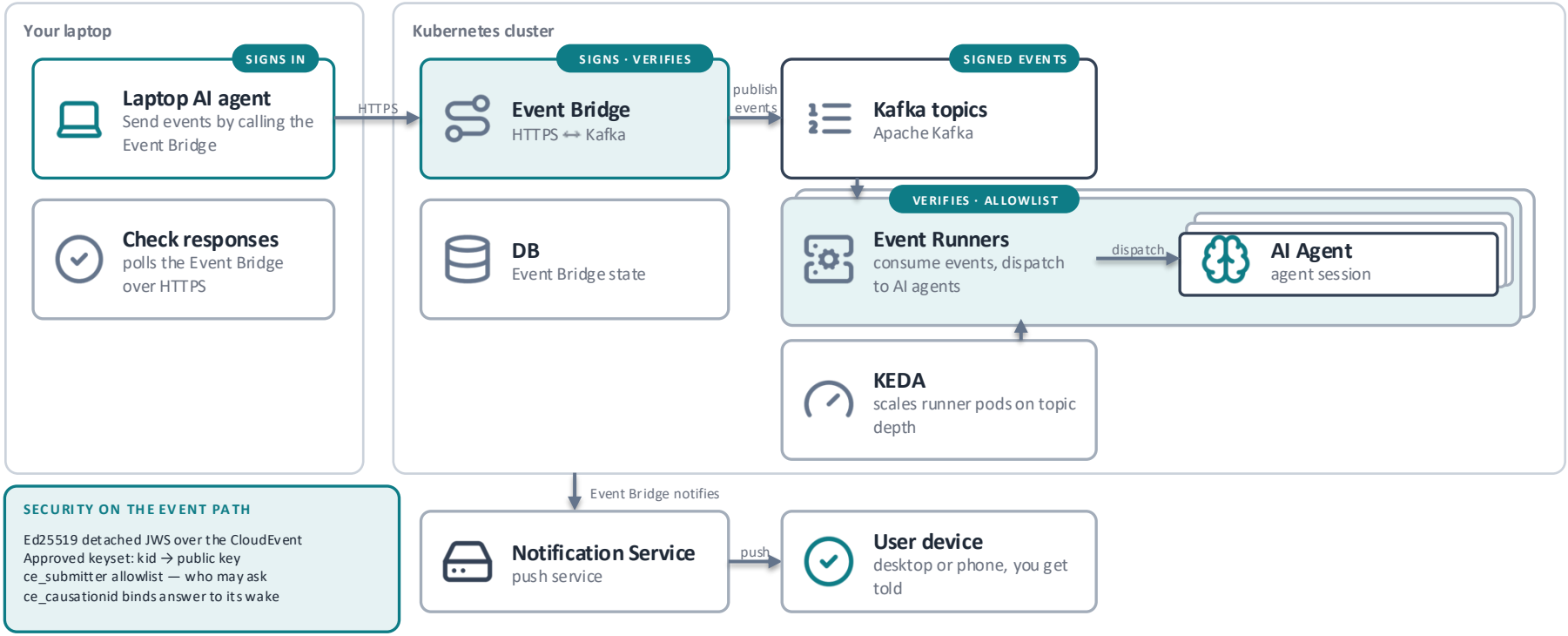
What we show in demo

- Part One: connect to Event Bridge and run one or one hundred agents
  - Request to run one agent with one event
  - Request to run 100 agents with 100 events

- Part Two: check audit log. Verify that events are signed and sequenced

- Part Three: check that processing unsigned events is rejected

Running AI Agent fleet with KEDA, Apache Kafka, Strimzi, Cloud Events, Event Bridge and Event Runners



Every event on the topic is signed at the edge and verified before an agent runs — the dashed panel is the SPIFFE/SPIRE target, not today's build.

KubeCon CloudNativeCon North America 2026

Demo

Recorded and narrated — hard stop at 4 minutes



What to take away

Four seams, two closed, two named honestly — and the method that found them.

TAKEAWAY	WHAT IT MEANS FOR YOU
Composition is sufficient	KEDA + Strimzi + CloudEvents + JWS, two services of our own
Lag means “not finished”	Commit offsets after the terminal event and the autoscaler stops lying
Scale-to-zero is a control	No idle process holds a stale policy and edit it at 0 replicas
Verify, then trust the attribute	Signature before allowlist, or you filter on attacker-chosen input
Audit your own claims	Keep track what your agents are doing, what were inputs and outputs etc.

Learn more

- **Everything shown is open source — github.com/rossoctl/eventing-lab**
- This work is ongoing and it is part of https://www.rossoctl.dev/
  - Moca - A dense, warm process-manager runtime for stateful AI coding agents — sessions resume exactly where they left off. Agent state is durable, tool execution runs in isolated sandboxes, and one supervisor serves many sessions per warm worker.
  - Cortex - See what your coding agent actually sends — and pay less for it. A proxy in the agent's request path that shows every model call, tool call, and cost as it happens, and strips the tool definitions the agent never calls. Cortex Local runs on individual, end-user machines; Cortex Remote, deployed along the harness, is being explored.
- And many other projects - check our website and github for the latest developments
- And check another talk at 3:30pm: “Simulate, Evolve, Validate, Ship: AI-Discovered Algorithms for LLM Serving Stacks”

KubeCon CloudNativeCon North America 2026

Questions

https://github.com/rossoctl/eventing-lab

