Cluster Catalyst Hooks

Making Evernode Clusters Easier to Summon, More Likely to Persist, and Harder to Fork

Scott Robert Chamberlain · May 2026

1. Summary

1.1 This paper proposes a new primitive called the *Cluster Catalyst Hook*, a Hook attached to a Xahau account that summons autonomous AI Agents, and other instance-owners, to spin up an Evernode cluster. The Hook specifies the contract the cluster will run, how the UNL will be determined, how instance-owners will checkpoint save their state, and the economic model (if any) to incentivise instances. It provides an out-of-the-box solution to the on-chain governance infrastructure that all UNL-based consensus models need to ensure they persist and do not fork. It is tailor-made for a world where autonomous AI Agents dominate but is agnostic about whether the instances are spun up by Agents or by humans.

2. The Problem of Coordinating Evernode Clusters

2.1 HotPocket[1] provides everything a layer-1 chain needs at the consensus layer: round-based BFT consensus across N instances, deterministic state transitions, a transaction queue, finality. What it does not provide is what the host chain (Xahau itself) provides through `UNLModify` and the negative UNL: a protocol-level mechanism by which the network of instances agrees on which instances belong to the network. UNL composition has lived outside HotPocket — in configuration files, deployment scripts, social agreements among operators, and out-of-band trust about who is in.

2.2 This has had three consequences. First, spinning up a permissioned BFT chain on Evernode has been a deployment ceremony rather than a transaction. Second, changing UNL composition once a cluster is running has required coordinated reconfiguration across operators. Third, and most consequentially, the absence of an on-chain UNL primitive has made it hard to think of HotPocket clusters as first-class layer-1 chains in their own right. They have looked like distributed applications running on Evernode, rather than as the chains they actually are.

2.3 The missing primitive is small and well-defined: an on-chain authority that publishes which instances are currently in a cluster's UNL, that participants can read and audit, that can change membership according to rules anyone can examine, and that makes spinning up a new cluster a single transaction rather than a coordination problem. Xahau already has the substrate for this — Hooks, URITokens, Remarks, and an account model that can host an arbitrary number of these primitives. What was missing was the recognition that they compose into the primitive HotPocket has needed.

3. The Catalyst Hook Pattern

3.1 A Cluster Catalyst Hook is a Hook attached to a dedicated Xahau account that knows how to compose a HotPocket cluster. The Xahau account is the constitutional anchor. The Hook is the rule-set the participants agree to honour. The pattern has six invariant components.

(a) A *Xahau account* with the Hook installed. The account address is the cluster's identity on-chain. Anyone can find the cluster by querying its account.

(b) A *registration mechanism* by which an instance declares itself a candidate for the cluster's UNL. In every implementation so far, registration is a Remit transaction sent to the Hook account, carrying a URIToken minted by the registering instance. The URIToken is the durable handle the Hook uses to track the instance thereafter.

(c) A *state-publication channel* by which each registered instance writes rolling operational state to the Remarks of its own URIToken, signed by its own key. The merge-by-name semantics of `SetRemarks` make per-checkpoint writes cheap; the issuer-only write check enforces that no other party can spoof an instance's state. The Hook never participates in these writes.[2]

(d) A *rule set* embedded in the Hook itself, which the Hook applies to compute the cluster's UNL composition from the state it reads across registered URITokens. The rule set may be elaborate (state-hash convergence, liveness, funding) or simple (first N bidders to remit a stake token). It is part of the Hook's WebAssembly.

(e) A *lifecycle arbitration channel* by which the Hook admits, evicts, retires, or pays members. In every implementation so far, this is `HookGrants = ttPAYMENT + ttURITOKEN_BURN`: Payments out to members for earned fees or refunds, `URITokenBurn`s for retirements and exits.

(f) A *contract reference* recorded by the Hook, identifying the HotPocket contract the cluster runs. This is what makes the cluster a *specific* cluster doing a *specific* thing rather than a generic computation pool. Different Catalyst Hooks summon different clusters running different contracts.

3.2 Each variant of the pattern differs in the rules baked into (d) and the lifecycle phases recognised by (e). The other four components are invariant. A new Catalyst Hook is a new rule set in the same architectural slot.

4. The Catalogue of Variants

4.1 Once Catalyst Hooks are named as a class, the design space they open up becomes legible. Each variant is a different rule set in the same architectural slot. The list below is not exhaustive; it is the immediately obvious set of shapes the pattern accommodates.

(a) **Permanent service catalysts.** Fluid membership, convergence-gated, no end. The Hook admits members whose recent state is in agreement with the cluster majority and rotates seats on a moment cadence. The classic shape for a public oracle, a community service, or a long-running infrastructure cluster. The Cluster Persistence Hook (§6 below) is the canonical example.

(b) **Project catalysts.** Bounded seats, escrow-paid, defined end. A founder pre-funds a job; agents bid for seats; the cluster forms, executes the contract, and dissolves when funds drain. The shape for a dispute panel, a bounded computation gig, an oracle query with defined scope, a content review job. The Commission Hook (§7 below) is the canonical example.

(c) **Subscription catalysts.** Members pay to remain in the UNL because they earn fees from the cluster's operation. The Hook admits members who maintain a posted bond or pay a periodic seat fee; the cluster runs a contract whose users pay the cluster directly for service. The shape for a DEX matching engine, a relay network, a paid oracle, an inference cluster.

(d) **Governance catalysts.** Members are nominated and revocable by token holders or by an external governance process. The cluster runs the governance contract itself — proposal evaluation, voting, deliberation, treasury control. The shape for DAOs that want their decision-making executed on a multi-instance cluster rather than on a single trusted server.

(e) **Sub-chain validator catalysts.** Members are validators of an L2 protocol that settles back to Xahau. The Catalyst Hook governs the validator set; the cluster runs the L2 protocol. The shape for application-specific chains that want their validator set anchored on a public ledger rather than maintained off-chain.

4.2 Each variant differs from the others only in the rule set the Hook applies to admit, evict, and pay members. The substrate — a Xahau account, a Hook, URIToken-mediated registration, `SetRemarks`-published state, Payment/Burn lifecycle — is invariant. A future Catalyst Hook is a new rule set, not a new substrate.

5. Worked Example I — The Cluster Persistence Hook

5.1 The Cluster Persistence Hook is a canonical example of the permanent-service-catalyst shape and the cleanest available illustration of state-convergence as an admission criterion.

Eligibility by state convergence

5.2 Each member instance writes a checkpoint to its URIToken Remarks every round. The checkpoint contains the member's current `state_hash`, `ledger_seq`, `hp_round`, the peers it observes, and an application-data slot. The Hook is not involved in these writes — agents sign their own `SetRemarks` transactions; the spec is explicit that *"the hook does NOT fire on `SetRemarks`. Zero hook execution cost."*[3]

5.3 When other transactions cause the Hook to fire, it reads across the URITokens of all current members and computes the dominant state hash — the value of `state_hash` that the majority of member URITokens currently report. A member becomes eligible for the dynamic UNL when three conditions are simultaneously true: its last `AGREEMENT_DEPTH` consecutive checkpoints all match the dominant hash (default depth = 3, the fork-prevention gate); it is still live within `LIVENESS_WINDOW`; and its funding meets `MIN_FUNDING`. From the eligible pool the Hook selects up to `UNL_TARGET` (default 30) members and rotates `ROTATION_COUNT` (default 2) of them every `MOMENT_SIZE` ledgers (~4320, roughly four hours). Below the target the rotation is a no-op, giving graceful scaling up and down with cluster size.

What this shape gives you

5.4 State-convergence as an admission criterion produces a self-policing cluster: members who fall out of sync with the majority lose their seat at the next rotation; members who rejoin the convergence path become eligible again after `AGREEMENT_DEPTH` consecutive matching checkpoints. There is no end-state, no founder, no escrow. The cluster persists for as long as enough members remain in convergence and accommodates churn by treating it as part of normal operation.

---

6. Worked Example II — The Commission Hook

6.1 The Commission Hook assembles a bounded cluster of paid agents around a defined job, runs the contract to completion, and dissolves cleanly. It is a single primitive that does both seat allocation and payment scheduling.

The phase machine

6.2 Six phases, stored in `K_PHASE`, govern what the Hook accepts at each point in the lifecycle:

(a) **`GENESIS` → `OPEN`.** The founder sends an Invoke with op=0 carrying seat count, UNL size, payment mode, contract reference, and deadline. The Hook records the founder's address in `K_FOUNDER` and locks the commission into `OPEN`. From this point, the only state changes are admission events triggered by agent bids.

(b) **`OPEN` → `FORMING`.** Agents bid by sending a Remit carrying a `MintURIToken` with the `tfBurnable` flag set, whose URI is the agent's own Xahau address. The Hook validates the URIToken ID by recomputing `SHA512Half([0x00, 0x55] + sender + uri_bytes)` and storing it in the agent's `MB_UTID` slot. When `MEMBER_COUNT` reaches `SEAT_COUNT`, the Hook pre-calculates the per-moment share and transitions to `FORMING`.

(c) **`FORMING` → `EXECUTING`.** Triggered when the first member calls Invoke op=1 (Claim) with all seats filled.

(d) **`EXECUTING`.** Agents call Claim repeatedly to draw their accrued share. Funds drain from `K_FUNDS`. The Hook supports three payment modes selected at `GENESIS`: `SPLIT` (per-moment dynamic over remaining duration), `PER_MOMENT` (fixed per moment interval), `TIME_CAP` (fixed per total duration). The mode is locked at formation and cannot be changed.

(e) **COMPLETE / CANCELLED.** When `K_FUNDS` hits zero or the founder calls Invoke op=4, no new bids are accepted. Members claim final amounts; their URITokens burn; the v0.4 emission-tracking pattern in `cbak()` defers cleanup of global counters until the burn confirms, so a failed emission does not desync state.

Per-member state

6.3 Sixteen 32-byte global state keys (founder, phase, member count, funds, formation funds, share, deadline, and so on) plus a 55-byte per-member record keyed by agent address. The per-member record holds the URIToken ID at `MB_UTID`, cumulative claimed amount at `MB_TOTCLM`, last claim ledger at `MB_LASTCLM`, a single byte at `MB_INUNL` indicating UNL-vs-bench, and (v0.4) a `MB_STATUS` byte recording `pending_delete` for the emission-tracking pattern. The trigger surface is `HookOn = ttPAYMENT + ttREMIT + ttINVOKE + ttSETHOOK`; the emission grants are `HookGrants = ttPAYMENT + ttURITOKEN_BURN`.

What this shape gives you

6.4 Project-catalyst semantics produce a temporary firm. A founder, a bounded job, a fixed cluster, a defined end, transparent payouts. This is the shape that maps onto how most real-world agent commerce will arrive: dispute panels, computation gigs, oracle queries, content reviews — units of work with a beginning, an end, and a price. The Cluster Persistence shape composes a standing body; the Commission shape composes a contracted-out delivery team. Both are needed, and a long-running organisation can be built as a sequence of overlapping commissions if it wants the project shape's accountability with the persistence shape's continuity.

7. Future Directions

7.1 Working versions of both the Cluster Persistence Hook and Commissioning Hook have been deployed to the Xahau testnet and live Evernode instances. The Commissioning Hook has been used to summon 3 autonomous AI Agents to orchestrate numerous iterations of a Clockwork Court to test the efficacy of that concept.

7.2 Several directions follow naturally from naming the Catalyst Hook as a class.

(a) **Catalyst templates for the rest of the catalogue.** Subscription, governance, and sub-chain-validator catalysts have not yet been implemented. Each is a different rule set in the same architectural slot. Producing reference implementations of each — together with a small library of governance idioms (token-weighted voting, multisig founder, time-locked rule changes) — would make the pattern broadly usable rather than an EverAgents-only primitive.

(b) **A Catalyst Foundry.** Most prospective Catalyst Hook deployers will not be Hook developers. A factory pattern that accepts cluster-specific parameters and emits a correctly structured Hook from an audited template would lower the barrier to deployment dramatically. The Hook Foundry direction noted in the Durable State paper applies here as well; a Catalyst Foundry is a specialisation of the same idea.

(c) **Composition of catalysts.** A long-running organisation can be modelled as a sequence of overlapping project commissions, each finite but arranged so that the next opens before the last closes. This is one way to get persistence-catalyst durability with project-catalyst accountability. Other compositions are possible: an instance might participate in multiple Catalyst Hooks simultaneously, taking on different roles in different clusters. The composition primitives have not been worked out; the substrate supports them.

(d) **Cross-Catalyst protocols.** Once enough Catalyst Hooks are running, instances will want to move between them, reuse identities, port reputations. A standardised cross-Catalyst identity primitive — likely an extension of the URIToken pattern that registries can read — would let agents accumulate operational history that survives any specific cluster's lifetime.

The deeper question

7.3 Once Catalyst Hooks are named, what changes in how people build on Evernode? The hypothesis worth testing is that a primitive like this shifts the unit of construction from the application to the cluster: developers stop thinking about which server to deploy to and start thinking about which Catalyst Hook to nail to Xahau. If that shift is real, the next generation of decentralised applications will look qualitatively different from the current one. Catalyst Hooks would be the quiet primitive making the difference.

---

[1]HotPocket: the BFT consensus engine that coordinates instances of an Evernode contract cluster. Source at https://github.com/HotPocketDev/hpcore.

[2]Durable State on Ephemeral Infrastructure (2026): describes the `SetRemarks` merge-by-name semantics, the issuer-only write check, the 32 × ~228 byte ≈ 7KB encrypted capacity per URIToken, and the `_digest` Remark integrity seal.

[3]Cluster Persistence Hook v0.3 specification, §4.2.