Le-Space

SPIN-OFF E · PROTOTYPE

Relay Button on-chain

Relay deployments through a smart contract — authorised, settled and capped

> No account, no invoice, no fronting the money: the contract holds the balance, releases the launch and settles on consumption.

Docs Source npm

The problem

The Relay Button launches infrastructure at the press of a button. Paying for it still works like everywhere else — and everything awkward follows from that.

- **The balance sits with the provider**
  An account, a top-up, a relationship of trust. Exactly the dependency the rest of the stack avoids.
- **Deploying for others means fronting the money**
  An app offering the Relay Button to its users pays first and bills afterwards — manually, and at its own risk.
- **One click costs money — with no cap**
  An embedded launch button with no balance check is an open tab for anyone who finds it.
- **No share in the usage**
  The toolkit is open source and gets used — but there is no path to earning anything from a node that was launched.

The solution: a prepaid vault

A contract holds the balance and releases it step by step. Four calls cover the whole flow.

- **Deposit** TODAY

  `approve` and `deposit` with an ERC-20 token. The balance stays the user's — we never hold it.

- **Reserve** TODAY

  `reserve(intentHash, amount, expiresAt)` — locked for this one deployment, with an expiry.

- **Consume** TODAY

  `consume(intentHash, amount)` after the launch. What settles is what the node actually cost.

- **Refund** TODAY

  If the deployment never comes up, `refund(intentHash)` releases the expired reservation. Nobody has to ask for it.

> These four calls are not a design sketch: the client ships as `@le-space/browser` in the Relay Button monorepo — including balance reads, on Base, Avalanche and Ethereum.

More than paying: authorising

The real gain is not the payment but that the contract decides whether a node may start at all.

- **Money is bound to one intent**

  The `intentHash` ties the reservation to exactly one deployment. The balance cannot be spent on something else.

- **The contract is the referee**

  No reservation, no launch. The go-ahead no longer depends on a backend somebody runs and could switch off.

- **The cap is built in**

  Only what was deposited can be spent. An embedded launch button can no longer surprise anyone.

- **Every launch has a receipt**

  Reservation and consumption are transactions. Who launched what and when is verifiable without anyone keeping books.

RELAY BUTTON ON-CHAIN

How we earn

A fee per launched node — collected in the same step that settles it.

- **A fee on consumption**

  The vault keeps a share on `consume`. No collections, no invoice, no bad debt.

- **Scales with usage, not with sales**

  Every node anyone launches anywhere through the toolkit contributes — without us needing to know about it.

- **Non-custodial**

  Until consumption the balance is the user's. We hold no customer funds — the same line as in spin-offs B and C.

- **Opens the toolkit to third parties**

  Whoever embeds the Relay Button no longer fronts anything. That is what makes embedding it worth doing at all.

RELAY BUTTON ON-CHAIN

Who launches with it

Wherever somebody needs infrastructure but does not want a relationship with a provider.

App vendors

– Embed the Relay Button in their app

– Users pay for themselves, the app fronts nothing

– No billing system needed

Communities & DAOs

– A shared purse for a shared relay

– Visible who funded what, and for how long

– Keeps running as long as somebody tops it up

Agencies & resellers

– Deploy for clients without going into advance

– The client fills their own vault

– Fits the consulting business from spin-off D

Status & next steps

The client side exists and is published. What is missing is the contract behind it — and the fee inside it.

- **Vault client published** TODAY

  Deposit, reserve, consume, refund plus balance and reservation reads — in `@le-space/browser`, for three EVM chains.

- **Publish the contract itself** IN PROGRESS

  Source, deployment addresses and an audit. Without those the client is a key without a lock.

- **Build the fee logic in** IN PROGRESS

  The share per launched node belongs in the contract, not on an invoice. Rate and recipient configurable.

- **Couple it to the lifecycle** PLANNED

  Extend runtime, stop a node, return the remainder — and the same vault for providers beyond Aleph.

Open questions

Four points to settle before the first real balance goes in.

- **Who holds the balance**

  Staying non-custodial is the whole precondition. The contract has to be built so we can never move a user's money — not even by accident.

- **Price and cost drift apart**

  A node costs in euros and is paid in tokens. Between reservation and consumption that can move — stablecoin or buffer?

- **Gas per launch**

  Reserving and consuming are two transactions. On an L2 that is negligible, on Ethereum it is not — the chain is a product decision.

- **Who is liable for the node**

  When a contract rather than a person releases the launch: who operates what runs on it? That needs an answer before somebody asks.

RELAY BUTTON ON-CHAIN

Let's talk

Wanted: an app that wants to embed the Relay Button without fronting the cost — and someone to finish and audit the vault contract with us.

Le Space UG (haftungsbeschränkt) · Nico Krause

contact@le-space.de · local-first.le-space.de

+49 / 87 21 / 5 06 49 96

Docs | Source | npm