PORTALDOT HACKER HOUSE 2026 · MODULAR CLOUD & MIDDLEWARE

Portal Meter

Sell any API per call on Portaldot V3,
settled across both of its contract VMs.

portal-meter.swiftbotlab.workers.dev/demo · Swift Bot Labs

The problem

- APIs are now bought by **programs**: apps, bots, AI agents, indexers. They want to pay **per call**, cents or less.
- API keys + monthly plans don't fit a client that needs 40 calls today and none tomorrow.
- "Pay first" protocols (HTTP 402 / x402) move money but give the buyer **no recourse** when the call fails.
- And no way to know, before paying, whether a provider actually delivers.

What a machine buyer needs
price up front · pay only for what is served · automatic refund when it isn't · a public track record of the seller

What one request costs on-chain

Every new storage slot locks a deposit. We measured our own first design, one payment per request, and it does not fit a call worth a fraction of a cent.

	Per-request escrow	Prepaid session
Transactions per request	2	**0**
Locked for good per request	7.53 tPOT	**0.076 tPOT**
100 requests	about 753 tPOT	**7.6 tPOT**

Measured on the PortaldotV3 testnet with our own keys: blocks 276657 to 276660 and 282559. Test tokens have no value; the ratio is the point. deployments/proof/

100 times less locked per request
Same chain, same token, same seller. Escrow stays for calls worth a few cents or more: refund path, served once.

Prepaid sessions

1. **Open once:** the payer locks a budget and can name a fresh session key: an agent can spend this budget and nothing else
2. **Every request is a signed voucher, not a transaction.** The seller checks signature, budget, count and chain before the handler runs
3. **Each voucher commits to the hash of the previous response.** A redeemed batch is signed by the session key, not claimed by the provider
4. **One Ethereum transaction settles many sessions** across both VMs: `redeem` in Solidity, `record` in ink!, all or nothing
5. **Failed request: not charged. Lost response: replayed free.**
6. **Anyone can replay the public log** against the chain: ACCEPT or REJECT, with every check listed

Per-request escrow, for calls worth more

1. **GET** → `402` with a quote (price, token, endpoint, nonce)
2. Client pays **one request** into escrow: `MeterSettlement.pay()` on Portaldot's EVM; the request id is bound to the payer on-chain
3. Client retries with a signed proof → served **once**, with a provider-signed receipt
4. Provider **settles** or **refunds**; the buyer can reclaim after a delay if the provider does nothing
5. The outcome is **anchored in an ink! registry** in the same transaction; buyers confirm or dispute

Why Portaldot is essential

Native assets are ERC-20s

Every `pallet-assets` token is reachable from Solidity at a precompile address. Endpoints are priced in a native asset; no wrapped token to deploy or trust.

Two VMs, one identity

Escrow in Solidity (pallet-revive), provider registry in ink! (pallet-contracts). Both sides key accounts by the same EVM address, using the runtime's own mapping rule.

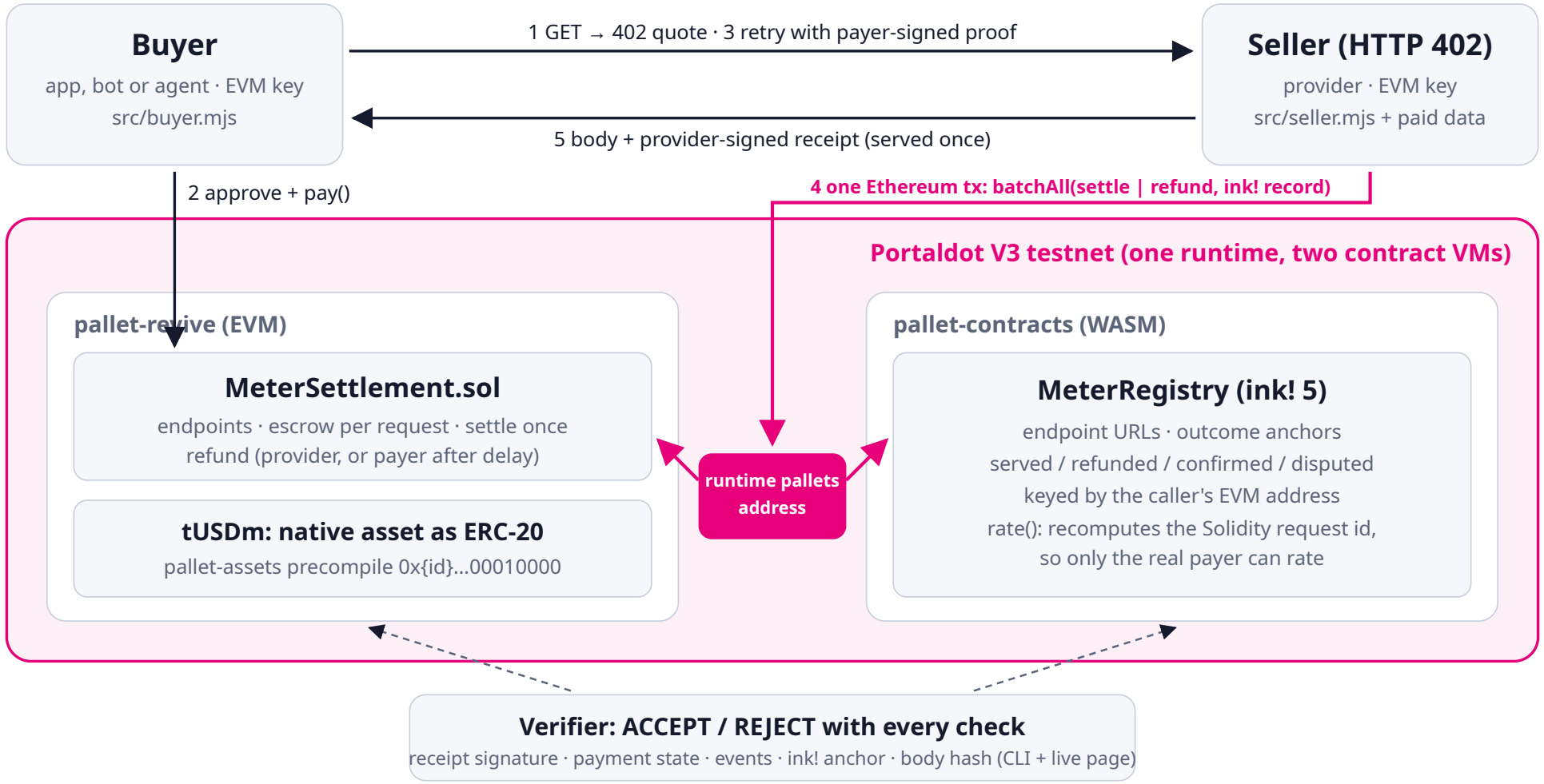
One transaction across both VMs

An Ethereum transaction to the runtime pallets address carries `utility.batchAll([revive.call(settle), contracts.call(record)])`: settlement and anchor succeed or fail together. A single-VM chain cannot do this.

Ink! checks EVM payers natively

The registry recomputes the Solidity request id (keccak + ABI) from the caller's address, so only the real payer can rate a request, with no bridge or oracle.

Architecture



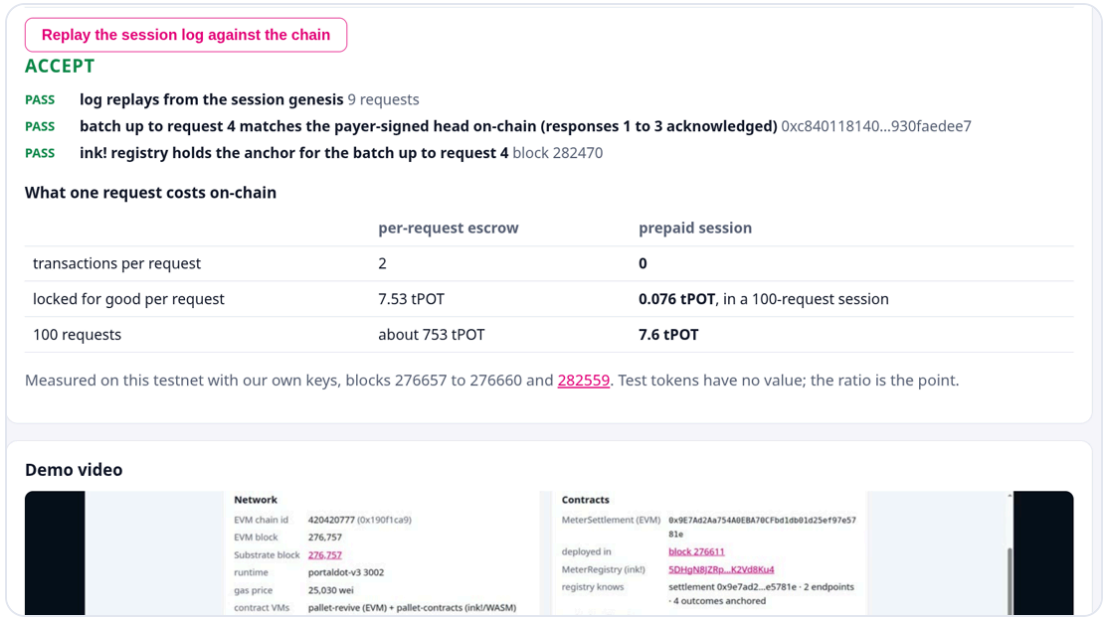
Tested, and checked on the live chain

42
Solidity tests
unit · fuzz · invariant

8
ink! unit tests

73
Node tests incl.
end to end on a chain

- A CI workflow in the repository: Foundry, ink! release build (cargo-contract 5.0.3), Node end to end
- 36 negative checks on the live seller and contracts: replays, strangers and tampered vouchers are refused
- Reused from a folder of its own: examples/paid-indexer installs the package, sells a different API (Portaldot's explorer index) and settled on the testnet, redeem + ink! anchor in one tx (block 323215)
- Independent security review before launch: findings fixed, MeterSession redeployed
- The live page buys a real call from the browser and replays the log against both VMs
- Open source, MIT. Source, history and proofs: portal-meter.swiftbotlab.workers.dev



Live on the PortaldotV3 testnet

MeterSettlement (Solidity, EVM)	`0x9E7Ad2Aa75…ef97e5781e`	deploy 0x1adf1c42…15731227 · block 276611
MeterSession (Solidity, EVM)	`0xCc2aBc8AC1…A0b743B3d5`	deploy 0xfa1c962f…5cbb6d2d · block 282417
MeterRegistry (ink!, WASM)	`5DHgN8JZRpyo…K1K2Vd8Ku4`	instantiate 0xef17ab33…97bd60e0 · block 276655
137 requests from 3 sessions settled: redeem + ink! anchor, one Ethereum tx	`0xe5038746f4…a1aded7f09`	block 282559
Native asset #43907 tUSDm	`0x0000ab8300…0000010000`	ERC-20 precompile address
Served: settle + ink! anchor, one Ethereum tx	`0xf5264278d3…1e7e0bacb6`	block 276658
Refunded: refund + ink! anchor, one Ethereum tx	`0x1a52029d1a…d68a36f314`	block 276660
Buyer confirms in ink! from its Ethereum key	`0x91d0a18a7a…7970cd3297`	block 276665

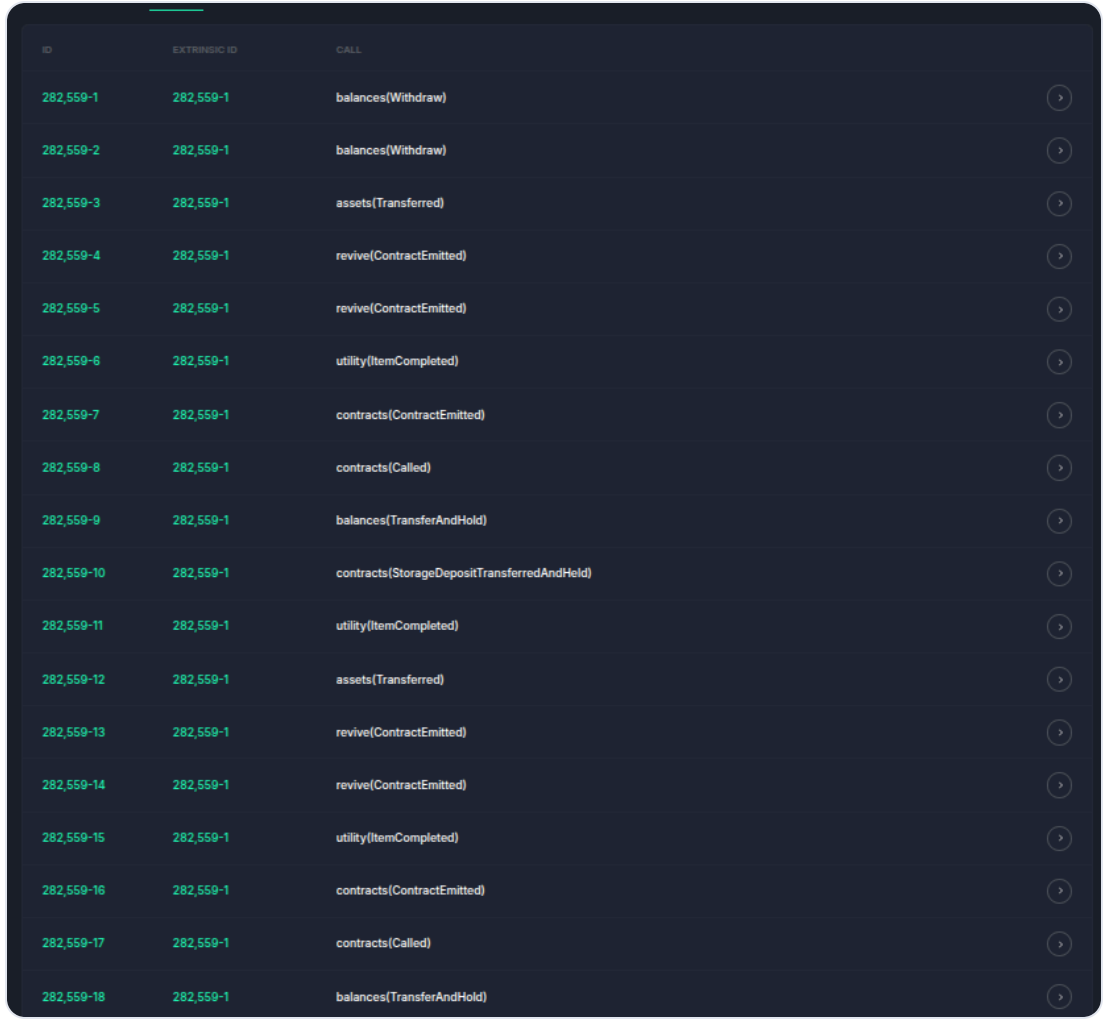
All transactions with blocks: deployments/testnet.json · live page: portal-meter.swiftbotlab.workers.dev/demo · explorer https://scanner-testnet-v3.portaldot.dev

One transaction, both VMs

The provider's Ethereum key signs **one** transaction to the runtime pallets address. Here it was signed inside a Cloudflare Worker:

```
utility.batchAll([revive.call(redeem),
contracts.call(record), ...])
```

- **137 requests, 3 sessions, one transaction**
- Budgets move to the payee (`assets.Transferred`, `revive.ContractEmitted`)
- The ink! registry anchors each batch (`contracts.Called`)
- All or nothing: `utility.BatchCompleted`
- Block 282,559 on the testnet explorer, events tab



ID	EXTRINSIC ID	CALL
282,559-1	282,559-1	balances(Withdraw)
282,559-2	282,559-1	balances(Withdraw)
282,559-3	282,559-1	assets(Transferred)
282,559-4	282,559-1	revive(ContractEmitted)
282,559-5	282,559-1	revive(ContractEmitted)
282,559-6	282,559-1	utility(ItemCompleted)
282,559-7	282,559-1	contracts(ContractEmitted)
282,559-8	282,559-1	contracts(Called)
282,559-9	282,559-1	balances(TransferAndHold)
282,559-10	282,559-1	contracts(StorageDepositTransferredAndHeld)
282,559-11	282,559-1	utility(ItemCompleted)
282,559-12	282,559-1	assets(Transferred)
282,559-13	282,559-1	revive(ContractEmitted)
282,559-14	282,559-1	revive(ContractEmitted)
282,559-15	282,559-1	utility(ItemCompleted)
282,559-16	282,559-1	contracts(ContractEmitted)
282,559-17	282,559-1	contracts(Called)
282,559-18	282,559-1	balances(TransferAndHold)

Who pays, and how it grows

- **Users:** data and API providers on Portaldot (indexers, RPC and analytics services, AI tool endpoints) and the apps and agents that call them.
- **First product, live:** the paid Portaldot data API, 0.01 pUSD per call, priced in Portaldot's own test dollar, so any builder can buy with real testnet funds.
- **Revenue (planned):** a small protocol fee on settlement (basis points, not in the testnet contracts yet) and a hosted seller for providers who do not want to run one. The Worker already shows what that service is.
- **Growth:** a few lines to add to any API (Express, Workers, Bun, Deno), then one command to register. Listing in the Portaldot ecosystem directory, wallet support for 402 quotes.

Honest status
Traffic so far is our own measurement key and visitors of the live page. No user demand is claimed. The next milestone is a second provider.

Roadmap, and why keep supporting it

Next 4 weeks

npm package and a provider CLI wizard · paid endpoints from other Portaldot builders · session keys with spending rules · mainnet migration review

Next quarter

provider bond held in the ink! registry · wallet and explorer support for quotes and receipts · a hosted seller for providers

What Portaldot gets

open-source middleware every API builder can reuse · fee-paying transactions on both VMs · measured notes on deposits and cross-VM calls for other teams

Team

Swift Bot Labs, a small studio building payment and API infrastructure. Builds on our MIT-licensed HSK Meter (Oct 2026); this repository was written for Portaldot during the Hacker House.