Contents

EverArcade Whitepaper v2.2

Decentralized Game Hosting Platform on Evernode
Rust-Powered Sandbox for Web3 Game Economies
Version 2.2 – February 2026 Authors: @Lj26ft / EverArcade Team v2.2 Update: Formalized Execution Sovereignty, XRPL anchoring clarification, and elastic scaling model.

Abstract

EverArcade is a decentralized game hosting platform built natively on **Evernode** (Layer 2 on Xahau/XRPL), enabling game developers to deploy persistent, deterministic multiplayer games as sandboxed “appchains.” Using modular **RustRigs** — event-sourced Rust crates that act as instruction generators — devs build chain-neutral game economies with self-custodial assets, payments, and logic, without rewriting engines or mastering blockchain.

Evernode's HotPocket consensus engine turns POSIX apps into trust-minimized clusters. EverArcade abstracts this into a sandboxed Rust SDK, allowing indie devs to push GPU-aware workloads, prototype offline, bind to Unity/Godot/Bevy, and host on decentralized Evernode nodes. Result: provably fair games, no central servers, low-cost scaling, and true player ownership.

Launching 2026.

1. The Problem: Web3 Gaming Still Sucks for Real Devs

Traditional game chains (e.g., Immutable X, Ronin, Polygon zkEVM gaming L2s) force devs into rigid smart-contract VMs (Solidity/EVM), high gas for state, centralized sequencers, or off-chain servers for real-time logic. Result:

- Netcode/sync headaches
- Cheat vectors
- Wallet friction
- Centralized IP control
- Poor GPU/compute integration

General L1/L2s aren't built for persistent game hosting — they're transaction processors, not appchain hosts.

2. EverArcade: Decentralized Hosting + Sandboxed Dev Platform

EverArcade treats Evernode as the foundation for **per-game appchains**:

- Each game leases a HotPocket cluster → dedicated consensus group (UNL validators) for deterministic execution.
- Sandboxed isolation via Sashimono jails → no cross-game interference.
- Rust-based SDK → `cargo add everarcade_rustrigs` → drop-in crates for commands/events/projections.
- **Rust instruction generators (RustRigs)** → modular crates that generate deterministic Rust instructions from game domain models (e.g., `game_rig`, `asset_rig`, `nft_rig`, `vault_rig` for rewards).

Core workflow (5 steps, ~10-20% backend code vs raw Evernode):

1. Design domain: actions → events → views (standard game loop style).
2. Add crates: `cargo add everarcade_core everarcade_game_rig`.
3. Wrap loop in orchestrator: `tick()`, command dispatch → event sourcing → state projection.
4. Bind engine: Unity/Godot/Bevy (WASM/native) via Rust → JS/WASM bridge.
5. Deploy: `evrs deploy` to leased Evernode hosts → decentralized, cheap, persistent.

Xahau Hooks enforce atomic on-chain elements (NFT mints on win, payments) trust-minimized.

3. Technical Specification: RustRigs

RustRigs are the core abstraction layer — a collection of composable, event-sourced Rust crates that generate deterministic instructions for game logic, state transitions, and economy rules.

Key Design Principles

- **Determinism by construction** — All logic executes identically across HotPocket cluster nodes via replay of ordered events.
- **Event sourcing** — State is derived from immutable event log (stored on-host disk + consensus-protected).
- **Modular rigs** — Each rig handles a domain (core, gameplay, assets, NFTs, rewards).
- **Instruction generators** — Rigs produce verifiable, serializable Rust instructions (e.g., `Command → Event → Projection` pipeline).
- **Sandbox isolation** — Runs in Sashimono jail; no direct host FS/network access outside allowed channels.

Core Flow

1. **Command** (player input or timer tick) → validated.
2. **Orchestrator** dispatches to relevant rig.
3. **Rig** generates **Event** (immutable fact, e.g. `PlayerMoved`, `ItemMinted`).
4. **Events** appended to log via HotPocket consensus.
5. **Projections** reduce events to current state views (queryable).
6. **Hooks** (Xahau) atomically settle on-chain elements (e.g. NFT trust-line set, payment path).

Sample Pseudocode (Minimal Rig)

```rust
// everarcade_core/src/lib.rs
use everarcade_traits::{Command, Event, Projection, Rig};

pub struct CoreRig;

impl Rig for CoreRig {
    fn handle(&self, cmd: Command) -> Result<Vec<Event>, Error> {
        match cmd {
            Command::Tick { timestamp } => Ok(vec![Event::Tick { timestamp }]),
            Command::PlayerJoin { player_id } => {
                // Validate, emit event
                Ok(vec![Event::PlayerJoined { player_id, timestamp: now() }])
            }
            _ => Err(Error::UnknownCommand),
        }
    }

    fn project(&self, state: &mut GameState, event: &Event) {
        match event {
            Event::PlayerJoined { player_id, .. } => {
                state.players.insert(*player_id, Player::default());
```

```
            }
            // ... more reducers
        }
    }
}

// Usage in orchestrator (HotPocket entry)
fn on_command(cmd_bytes: &[u8]) -> Vec<u8> {
    let cmd: Command = deserialize(cmd_bytes);
    let events = core_rig.handle(cmd).unwrap_or_default();
    // Append to log via HotPocket API, emit projections
    serialize(&events)
}
```

Determinism Guarantees

- All rigs are pure functions (no floats unless fixed-point, no rand without seeded).
- Timestamp/order enforced by HotPocket round consensus.
- Replays from genesis log yield identical state.

4. GPU Wrapper Theory & Implementation Plan

Evernode/HotPocket provides deterministic CPU consensus but lacks native GPU support. EverArcade introduces a **hybrid acceleration layer** for compute-heavy games (physics sims, procedural gen, AI, rendering previews).

Theory

- **Split execution**: Consensus-critical logic (state transitions, economy rules) runs deterministically in HotPocket CPU rounds.
- **Non-deterministic accel** runs locally on host GPU (via CUDA/Vulkan) or offline during dev.
- **Results** fed back as verifiable inputs (e.g. Merkle-ized seed + output hash checked in rig).
- **Wrapper crate** (`everarcade_gpu`) bridges: local dev uses real GPU; deployed host uses CPU fallback or accelerated path (future host GPU leasing).

Phased Approach

1. **Phase 1 (2026 Q1)**: Local/offline GPU accel for prototyping (wgpu crate, compute shaders).
2. **Phase 2**: Deterministic seed → GPU → hash verification rig.
3. **Phase 3**: Host GPU awareness via Sashimono extensions (advertise capability in lease offers).
4. **Phase 4**: Full on-host GPU compute for non-state-affecting workloads (e.g. batch sims).

This makes EverArcade unique: first Evernode platform bridging traditional GPU game dev to decentralized hosting without sacrificing determinism.

5. Sample Crate Dependency Diagram

EverArcade uses a clean, layered crate structure for separation of concerns.

ASCII Tree View (cargo tree style)

```
everarcade-game (binary / workspace root)
├── everarcade_core              # Determinism primitives, orchestrator
│   └── everarcade_traits         # Shared traits: Command, Event, Rig
├── everarcade_game_rig       # Genre-specific logic (e.g. arcade shooter)
│   └── everarcade_core
├── everarcade_asset_rig          # Items, inventory, ownership
│   └── everarcade_core
├── everarcade_nft_rig            # Mint/transfer via Xahau Hooks
│   ├── everarcade_core
│   └── xrpl-rs                   # XRPL client for hooks settlement
├── everarcade_gpu (dev-only)     # GPU wrapper + wgpu integration
│   └── wgpu                      # Cross-platform GPU API
└── everarcade_sdk            # User-facing: macros, bindings for engines
    └── everarcade_core
```

This structure enforces modularity: core never depends on game-specific rigs; rigs are swappable.

6. Unique Differentiation from Regular Game Chains

Aspect	Regular Game Chains (EVM L2s, Ronin, etc.)	EverArcade on Evernode
Execution Model	EVM / WASM VMs, gas-per-op	HotPocket POSIX consensus — full app runtime determinism
Hosting	Centralized nodes or rollup sequencers	Decentralized marketplace (hosts lease via EVR)
Persistence	Off-chain servers for real-time	On-host disk + consensus — 24/7 stateful clusters
Dev Language	Solidity / Move / Rust subset	Full Rust + any POSIX (sandboxed SDK focus)
GPU / Compute	Rare / off-chain	GPU-aware wrapper in dev (local accel → host push)
Consensus Granularity	Chain-wide	Per-game appchain (custom UNL per cluster)
Onboarding Barrier	Blockchain expertise required	Prototype offline, no chain knowledge needed
Economies	Token-centric, high friction	Chain-neutral, self-sovereign (cross-chain potential)
Cost / Scaling	Gas spikes, sequencer fees	Lease-based (EVR), host competition → low & predictable

EverArcade is unique in bringing **GPU-aware hosting** to Evernode: developing a wrap-

per that accelerates non-consensus workloads (rendering sims, AI, physics) locally/on-host without breaking deterministic consensus rounds. Indie devs retain self-custody of IP/assets — no platform takes a cut or controls distribution.

7. Bringing Real Value to XRPL / Evernode Ecosystem

- Onboards traditional game devs to Web3 via familiar Rust + engine binds.
- Demonstrates Evernode's power for compute-heavy, persistent dApps beyond simple contracts.
- Enables chain-neutral economies: settle in XAH/XRP/EVR or cross-chain via XRPL DEX/pathfinding.
- Lowers barrier: deterministic execution rigs make provable fairness accessible.
- Scales via Evernode's decentralized host marketplace — no single point of failure.

8. Institutional Extension: White-Label Vault Option (Secondary)

For qualified institutional partners, EverArcade supports optional white-label vault primitives via Xahau Hooks + XRPL trust lines/escrows. These enable compliant, tax-advantaged structures (e.g., Qualified Opportunity Zones) for game-economy funds or asset-backed rewards — fully optional, backend-only, and compliant-focused. Core platform remains gaming/dev-first.

9. Separation of Execution and Settlement

EverArcade enforces a strict architectural separation between deterministic execution and settlement infrastructure.

Execution generates state transitions through RustRigs operating inside HotPocket consensus clusters. Settlement provides cryptographic anchoring and economic finality without influencing execution semantics.

The system is composed of three orthogonal layers:

• Execution Layer — deterministic RustRigs producing ordered events • Commitment Layer — content-addressed execution log (on-host + IPFS compatible) • Settlement Layer — XRPL/Xahau anchoring of cryptographic commitments

Execution produces state. Settlement finalizes commitments of that state.

These responsibilities are intentionally separated to preserve determinism, replayability, and infrastructure neutrality.

##9.2 Deterministic Execution Model

All EverArcade rigs (core, gameplay, assets, NFTs, vaults) satisfy the deterministic transition model:

f:(S,A)→E f:(S,A)→E

Where:

• S = prior state derived from ordered event log • A = validated command • E = immutable event(s) appended to log

Key invariants:

• No global mutable state • No unseeded randomness • No implicit wall-clock time dependency • No hidden side effects • Identical inputs produce identical outputs

HotPocket consensus enforces ordered event agreement across cluster nodes. Replaying the log from genesis must yield identical state on every node.

Determinism is guaranteed by construction.

##9.3 What “Anchoring to XRPL” Means

EverArcade does not store raw game state on XRPL.

Instead:

Events are appended to the on-host consensus log.

The log (or its Merkle root / cryptographic hash) is generated.

That commitment hash is anchored via XRPL/Xahau transaction metadata.

XRPL acts as:

• A timestamping layer • A public finality layer • A cryptographic commitment registry

XRPL does not execute the game. XRPL attests to the integrity of execution history.

Anyone can:

• Re-execute deterministic logic • Reconstruct the event log • Verify anchored commitments match reproduced state

Settlement verifies integrity. Execution defines semantics.

##9.4 Dispatch Layer & Orchestrator Constraints

The orchestrator (dispatch layer) coordinates execution inside the HotPocket app runtime.

It is responsible for:

• Validating commands • Routing to appropriate rigs • Aggregating emitted events • Maintaining canonical ordering

It is not permitted to:

• Modify rig logic • Inject hidden execution paths • Mutate deterministic outputs • Execute unverified custom logic

All execution surfaces are inspectable Rust code. All events are consensus-ordered. All commitments are reproducible.

If an orchestrator instance attempted to alter logic, replay verification against anchored commitments would fail.

Thus, the orchestrator is a coordination component — not a semantic authority.

Execution rules remain sovereign.

##9.5 Elastic Execution Scaling

A game may initially deploy to a single high-performance Evernode host cluster.

As usage increases, additional execution capacity may be provisioned via Evernode's decentralized host marketplace.

Scaling affects:

• Throughput capacity • Geographic redundancy • Resource availability

Scaling does not affect:

• Deterministic outcomes • Game logic • State validity • Settlement guarantees

Horizontal expansion replicates the same deterministic execution surface. Infrastructure may scale elastically; semantics remain invariant.

Scaling changes who executes. It does not change what execution means.

##9.6 Centralization & Misuse Resistance

A common concern is whether the orchestrator introduces centralization risk.

EverArcade mitigates this by:

• Deterministic replay requirements • Consensus-enforced event ordering • Cryptographic log commitments • Public anchoring of execution history

An orchestrator cannot secretly execute alternative logic without producing divergent state hashes.

Divergence is publicly detectable.

Infrastructure operators control liveness. They do not control semantics.

9.7 Infrastructure Neutrality

Execution is portable across:

• Single-node deployments • Multi-node HotPocket clusters • Sovereign private deployments • Future GPU-enabled hosts • Expanded Evernode accelerator pools

Because logic is deterministic and inspectable, infrastructure evolution does not require protocol redesign.

Infrastructure is elastic. Execution is sovereign. Settlement is subordinate.

##Appendix A — RustRigs as a Public Deterministic Execution Library A.1 Overview

RustRigs is not merely an internal EverArcade SDK component. It is a public, composable, event-sourced Rust library for building deterministic state machines suitable for decentralized execution environments.

While EverArcade deploys RustRigs within Evernode HotPocket clusters, the library itself is chain-neutral and infrastructure-agnostic.RustRigs is designed as a public deterministic execution standard, not a proprietary platform dependency.

RustRigs can be used to build:

• Deterministic multiplayer game logic • Asset and inventory systems • NFT issuance pipelines • Vault and reward primitives • Event-sourced financial models • Simulation engines • Deterministic AI interaction layers

RustRigs is an execution abstraction, not a chain abstraction.

##A.2 Architectural Positioning

RustRigs sits between:

• Application domain logic • Deterministic runtime execution

It formalizes the Command → Event → Projection pattern in a way that enforces determinism by construction.

Each rig implements:

• Command validation • Event emission • State projection

This produces a portable, replayable state machine independent of consensus layer.

RustRigs does not:

• Depend on XRPL • Require Evernode • Assume blockchain settlement • Embed token economics

It is a deterministic instruction generator library.

##A.3 Determinism by Construction

RustRigs enforces deterministic execution constraints:

• No implicit global state • No nondeterministic syscalls • No unseeded randomness • Fixed-point math only (no floating drift) • Explicit timestamp injection (never system clock) • Ordered event sourcing

This allows identical state reconstruction across:

• Local development • Single-node deployments • HotPocket clusters • Sovereign execution nodes

Replaying the event log from genesis must always yield identical state.

Determinism is a library-level invariant.

##A.4 Library Structure

RustRigs follows a layered crate model:

everarcade_traits ├── Command ├── Event ├── Projection └── Rig

everarcade_core ├── Orchestration primitives ├── Determinism guards └── Log interfaces

domain_rig (e.g. game_rig, asset_rig, nft_rig) └── Implements Rig trait

This structure ensures:

• Core primitives remain domain-agnostic • Domain logic remains modular • Execution surfaces remain inspectable • Rigs remain swappable

RustRigs can be published and versioned independently of EverArcade hosting infrastructure.

##A.5 Public Use Cases Beyond EverArcade

RustRigs may be used in:

• Traditional server-authoritative games (without blockchain) • Deterministic simulation research • Financial modeling systems • Decentralized compute marketplaces • On-device event-sourced applications • Educational deterministic programming frameworks

EverArcade demonstrates one deployment model of RustRigs. It does not constrain its applicability.

##A.6 Relationship to HotPocket & Evernode

Within EverArcade:

• RustRigs define deterministic execution semantics • HotPocket provides ordered consensus • Evernode provides decentralized host marketplace • XRPL provides anchoring and settlement

RustRigs remain independent of all three.

If HotPocket were replaced, RustRigs would remain valid.

If XRPL anchoring were removed, RustRigs would remain valid.

If Evernode compute markets evolved, RustRigs would remain valid.

RustRigs define execution semantics. Infrastructure provides coordination and finality.

##A.7 Security Model

RustRigs' security properties derive from:

• Deterministic replay • Immutable event logs • Explicit execution surfaces • Trait-bound contract interfaces

Any deviation from expected behavior produces detectable divergence during replay.

This makes RustRigs compatible with:

• Verifiable compute • Multi-node execution replication • Cryptographic commitment systems • Third-party audit frameworks

##A.8 Long-Term Vision

RustRigs may evolve into:

• A general-purpose deterministic execution framework • A standard for event-sourced decentralized apps • A portable execution substrate for compute marketplaces • A foundation for sovereign execution environments

EverArcade represents the first production-grade deployment context of RustRigs.

RustRigs is the library. EverArcade is the platform.

##Appendix B — Illustrative GPU Host Capability Advertisement Schema (Non-Final)
##B.1 Overview

EverArcade anticipates GPU-aware host environments within the Evernode marketplace.

Hosts may advertise deterministic execution and acceleration capabilities via a structured metadata schema. The schema below is illustrative and non-final. It represents the type of capability disclosure required for compatibility with EverArcade workloads.

This schema does not define settlement behavior. It defines execution capacity advertisement only.

##B.2 Design Goals

Host capability advertisement must:

• Be machine-readable • Separate deterministic-critical compute from acceleration layers • Expose hardware class without leaking sensitive host data • Allow workload matching • Preserve execution invariants

Acceleration must never compromise deterministic consensus execution.

##B.3 Illustrative JSON Schema (Host GPU Capability Advertisement)

The following schema is illustrative and non-binding. Production versions may evolve prior to mainnet release.

It defines the structure for Evernode hosts to advertise their capabilities in the EverArcade lease marketplace — particularly deterministic runtime support and optional GPU acceleration eligibility.

Important: This is a JSON Schema (draft 2020-12) that validates host advertisement payloads. Example of a compact host advertisement for EverArcade lease offers.

```
{
  "title": "EverArcade Host Capability (Minimal)",
  "type": "object",
  "required": ["host_id", "execution_environment"],
  "properties": {
    "host_id": { "type": "string" },
    "execution_environment": {
      "type": "object",
      "properties": {
        "deterministic_runtime": { "const": "hotpocket-posix" },
        "sandbox": { "const": "sashimono-jail" }
      }
    },
    "gpu_capabilities": {
      "type": "object",
```

```
      "properties": {
        "gpu_enabled": { "type": "boolean" }
      }
    },
    "lease_terms": {
      "properties": {
        "base_fee_evr": { "type": "number" }
      }
    }
  }
}
```

{=latex} ##B.4 Key Architectural Constraints

Deterministic execution must always run on consensus CPU path.

GPU acceleration may only be used for:

Non-consensus-critical workloads

Pre-processing

Simulation

Hash-verifiable outputs

GPU outputs must be verifiable via:

Seeded input determinism

Hash commitments

Merkle proofs (future phase)

GPU capability advertisement does not alter execution semantics.

##B.5 Deterministic Safety Model

Even when gpu_enabled = true:

• Consensus-critical state transitions must execute in deterministic CPU path. • GPU-derived outputs must be validated before inclusion. • Divergent outputs are rejected by consensus rules.

GPU capability increases performance potential. It does not increase semantic authority.

##B.6 Future Extensions (Non-Binding)

Future production schemas may include:

• Remote attestation proofs • ZK validity proofs of GPU computation • Performance benchmark commitments • Reputation scoring • Deterministic GPU emulation modes

Addendum v2.2 – Economics & Phase 2 GPU/Execution Spine Plan

1. Purpose & Scope

This addendum expands EverArcade's economic model to include:

- GPU-enabled execution as an optional overlay
- ZK batching and provably fair game logic
- AI asset integration
- Attachment-based GPU market mechanics
- Phase 2 roadmap for scaling Evernode with specialized compute and game appchain support

It clarifies how Evernode's deterministic lease market interacts with GPU compute, while preserving economic stability and developer-facing simplicity.

2. Evernode Market Principles

2.1 Base Lease Integrity

- CPU-bound execution remains the foundation of Evernode.
- Base leases are priced independently to maintain:
  - Deterministic execution
  - Predictable compute cost
  - Fair access for all node operators

2.2 GPU Attachment Market

- GPU compute is an optional overlay market:
  - Allows specialized hardware to participate without destabilizing the base CPU market.
  - Supports rendering, AI inference, and zk-batching workloads.
  - Nodes earn two revenue streams:
    - Base Evernode lease
    - GPU attachment lease
- GPU compute pricing:
  - Determined via spot auction or tokenized compute unit model
  - Each GPU unit is represented as a declarative execution asset (e.g., GPU-minutes at a given utilization)

3. Execution Spine & ZK Rollups

3.1 Purpose-built Game Appchains Evernode hosts self-sovereign game appchains, each:

- Anchored to XRPL/Xahau
- Capable of running complex, deterministic logic
- Leveraging zk-rollups for provably fair outcomes and privacy-preserving transactions

3.2 Execution Spine Architecture

- Headless Evernode hosts coordinate:
  - Game asset runtime
  - GPU computation
  - ZK batching
- Rollups transmitted to IPFS logs for auditability and traceability
- Ensures provable fairness and compliance with potential regulatory requirements

4. Developer Experience & Asset Management

4.1 Low-Attack-Surface Crates Developers interact only with:

- game-core, asset-core, nft-core, everarcade-core crates

No direct node or market access required — reduces attack surface and operational complexity.

4.2 Drag-and-Drop Asset Integration Frontend allows:

- Bulk import of game assets (images, audio, shaders, AI models)
- Automatic logging into marketplace
- Runtime validation and execution binding

Supports chain-agnostic asset deployment, including XRPL/Xahau ecosystem assets.

5. Economic Incentives & Market Design

5.1 Node Operator Incentives

- Base CPU lease revenue for deterministic compute
- GPU attachment revenue for specialized tasks
- Elastic scaling allows nodes to choose between CPU-only, GPU-only, or hybrid deployment

5.2 Developer Incentives

- Pay-per-use model for hosting game appchains
- Start small and scale with usage:
  - Low initial cost encourages indie dev participation
  - Growth aligns with market-driven GPU and CPU lease prices
- Marketplace ensures visibility and monetization potential for game assets

5.3 Market Stability Separation of CPU and GPU markets prevents:

- Lease price volatility
- Centralization risk
- Distortion of base execution economics

6. Phase 2 Roadmap – GPU & Appchain Scaling

Phase 2 Focus	Description	Economic Impact
GPU Marketplace	Establish attachment-based GPU compute market	Optional revenue stream, preserves CPU lease integrity
ZK Rollup Integration	Enable provably fair execution for games	Compliance, trust, privacy
AI Asset Support	Allow AI-driven game features and content	Expands developer capability, attracts advanced workloads
Self-Sovereign Game Appchains	Independent appchains per game	Horizontal scaling, deterministic execution
Elastic Infrastructure	GPU-aware Evernode orchestration	Maintains semantics, supports high-demand workloads
XRPL/Xahau Anchoring	Transactional and asset interoperability	Chain-agnostic but XRPL-compatible

7. Strategic Advantages

- Decentralized Steam Alternative: Indie devs can deploy, scale, and monetize games without centralized platform restrictions.
- Trust & Compliance: ZK rollups enable provably fair, privacy-preserving gameplay.
- Scalable Market Model: GPU and CPU markets coexist, allowing economic growth without destabilization.
- Institutional-Grade Execution Spine: Deterministic, audited, and extensible for complex game logic.
- Chain Agnostic, XRPL Integrated: Assets and transactions can span multiple ecosystems while anchoring securely to XRPL/Xahau.

8. Conclusion (Addendum)

EverArcade's Phase 2 plan positions the platform to:

- Integrate GPU compute as a specialized, optional attachment market
- Maintain Evernode market stability
- Enable provably fair, privacy-preserving game appchains
- Support AI assets and high-complexity workloads
- Offer scalable, chain-agnostic infrastructure for developers and node operators

By threading GPU, ZK, and appchain mechanics carefully, EverArcade ensures market stability, developer empowerment, and economic alignment while building toward a decentralized gaming ecosystem of institutional-grade reliability. This appendix does not define final marketplace rules.

Roadmap (2026 Launch Focus)

- Q1: RustRigs core + SDK beta, GPU wrapper PoC
- Q2: Testnet clusters, engine bindings (Unity/Godot/Bevy)
- Q3: Mainnet deploy tooling, deterministic demo games
- Q4: Public arcade launch, indie dev onboarding

EverArcade turns Evernode's potential into the premier decentralized game hosting platform — sandboxed, Rust-first, GPU-aware, and truly dev-sovereign. Follow @4EvrArcade, star the repo (link in landing), join discussions.

Built by @Lj26ft on Evernode/XRPL. "'