ANASTASIA LABS

Lucid Evolution 2.0

Proof of Achievement - Milestone 1

Project Id	1300128
Project Manager	Jonathan Rodriguez
Proposal Link	Catalyst Proposal

Contents

Private Testnet SDK & L2 Provider Integration Design Specification Document

1. Executive Summary

Lucid Evolution 2.0 Private Testnet SDK is a TypeScript package designed to provide developers with a reliable and efficient testing environment for Cardano smart contracts. It addresses the current challenges faced by developers in simulating Cardano blockchain behavior, such as incomplete emulators, shell scripts and manual docker commands, providing developers with programmable control over real Cardano nodes.

A key benefit: is that using Dockerode and Testcontainers enables developers to spin up on-demand, disposable prvate testnets in under 30 seconds, eliminating external dependencies and enabling seamless testing framework integration.

Core Achievement: The SDK fundamentally transforms the developer experience by shifting infrastructure control into the codebase, freeing developers to focus on application logic rather than environment orchestration.

2. Introduction & Objectives

2.1. Project Context

Cardano DApp and smart contract developers lack reliable, reproducible test environments that mirror mainnet behavior. Current TypeScript transaction builders depend on external emulators or ad-hoc Docker scripts, leading to inconsistent results, fragile CI pipelines, and increased development overhead.

2.2. Project Objectives

1. **Direct Control:** Expose a TypeScript API (Effect-TS) to manage the full lifecycle of Cardano node containers—create, start, stop, remove, status—without leaving the JavaScript/TypeScript ecosystem.
2. **Real-Node Integration:** Run actual Cardano nodes (Byron → Conway) inside Docker containers to guarantee protocol-accurate behavior.
3. **Developer Experience:** Provide zero-configuration sensible defaults while allowing deep customization (genesis parameters, network magic, ports).
4. **Production Parity:** Ensure the SDK replicates mainnet consensus, mempool, transaction validation, and L2 state channels (Hydra).
5. **Modularity & Extensibility:** Design a clear separation between DevNet management, Lucid provider integration, and L2 modules—enabling future expansion.
6. **Security & Resilience:** Implement secure key handling, ephemeral directories, resource isolation, and robust error handling (Tagged Errors).
7. **Seamless Test Framework Integration:** Offer utilities (custom Jest/Vitest matchers, test network manager, mock data) to simplify writing and running automated tests.

3. Problem Statement

3.1. Current Challenges

Developers face four primary challenges when building and testing Cardano applications:

External Dependencies: Relying on bash/CLI scripts and raw Docker commands breaks TypeScript-first workflows and complicates CI/CD.

Inconsistent Emulation: Incomplete emulators (e.g., mock ledger states) fail to emulate real block production, consensus rules, or L2 dynamics—leading to false positives/negatives.

Complex Setup: Manual configuration of private testnets, genesis files, and key material demands deep infrastructure knowledge, wasting developer time.

Integration Gaps: No native TypeScript API to spin up, configure, or query a real Cardano node within unit/integration tests.

Without a reliable, repeatable, and programmable test environment, developer productivity, test coverage, and smart contract security suffer.

3.2. Proposed Solution

Comprehensive TypeScript package enabling private testnets with real Cardano nodes, controllable through Effect-TS APIs, with integrated L2 provider capabilities including Hydra support.

4. Solution Overview

4.1. Project Architecture

- DevNet Manager: Orchestrates container lifecycle, configuration generation (genesis files, keys), and resource isolation.
- Provider Module: Implements Lucid's Provider interface backed by DevNet containers, exposing methods for submitting transactions, querying UTxOs, retrieving datums, etc.
- L2 Integration: Extends the Provider to manage Hydra heads, off-chain transactions, and on-chain settlement.
- Effect-TS Core: Functional programming primitives for concurrency, error handling (Tagged Errors), logging, and resource cleanup.
- Dockerode/Testcontainers: Library layers for programmatically interacting with Docker via Node.js.
- Docker Runtime: Runs real Cardano node(s) (any era) and Hydra node(s), each in isolated containers.

5. Detailed Component Specifications

5.1. DevNet Container Management

5.1.1. Core Types & Errors

```
export class CardanoDevNetError extends Data.TaggedError("CardanoDevNetError")<{
  reason:
    | "container_not_found"
    | "container_creation_failed"
    | "container_start_failed"
    | "container_stop_failed"
    | "container_removal_failed"
    | "container_inspection_failed"
    | "temp_directory_creation_failed"
    | "config_file_write_failed"
    | "file_permissions_failed";
  message: string;
  cause?: unknown;
}> {}

export interface DevNetContainer {
  readonly id: string;
  readonly name: string;
  readonly socketPath: string; // e.g. /opt/cardano/ipc/node.socket
}
```

Manages Docker container lifecycle and configuration.

CardanoDevNetError: Tagged error types for container operations. Gives precise failure context (e.g. `container_inspection_failed`) with actionable messages and underlying causes.

DevNetContainer: Container identifier interface. Holds Docker container ID, name, and socket path for RPC.

5.1.2. DevNetConfig Schema

```
interface DevNetConfig {
  readonly containerName?: string;
  readonly image?: string; // Docker image (e.g., "ghcr.io/cardano-node:10.4.1")
  readonly ports?: { readonly node: number; readonly submit: number };
  readonly networkMagic?: number;
  readonly nodeConfig?: Partial<NodeConfig>;
  readonly byronGenesis?: Partial<ByronGenesis>;
  readonly shelleyGenesis?: Partial<ShelleyGenesis>;
  readonly alonzoGenesis?: Partial<AlonzoGenesis>;
  readonly conwayGenesis?: Partial<ConwayGenesis>;
  readonly kesKey?: Partial<KesKey>;
  readonly opCert?: Partial<OpCert>;
  readonly vrfSkey?: Partial<VrfSkey>;
}
```

- All fields have sensible defaults (DEFAULT_DEVNET_CONFIG) to enable zero-config setups.
- Each genesis interface (Byron, Shelley, Alonzo, Conway) is a partial of a comprehensive era-specific type with 40+ parameters (epoch lengths, protocol params, initial funds, delegation, etc.).
- Cryptographic key types (KES, VRF, OCERT) include default ephemeral key pairs; permissions (0o600) are enforced.

5.1.3. Container Lifecycle APIs

```
export const make: (
  config?: DevNetConfig
) => Effect<DevNetContainer, CardanoDevNetError>

export const startContainer: (
  container: DevNetContainer
) => Effect<void, CardanoDevNetError>

export const stopContainer: (
  container: DevNetContainer
) => Effect<void, CardanoDevNetError>

export const removeContainer: (
  container: DevNetContainer
) => Effect<void, CardanoDevNetError>

export const getContainerStatus: (
  container: DevNetContainer
) => Effect<Docker.ContainerInspectInfo, CardanoDevNetError>
```

- **make(config?):**
  1. Constructs a `Required<DevNetConfig>` by merging defaults with overrides.
  2. Calls `findContainer` by name; if exists, inspects & stops/removes.
  3. Invokes `createDockerContainer`:
     - Creates a secure temporary directory (`fs.promises.mkdtemp`) for config + keys.
     - Writes `config.json`, `topology.json`, all `genesis-*.json`, `kes.skey`, `pool.cert`, `vrf.skey` with proper permissions (0o600).
     - Generates a Docker container spec with `Image`, `Env`, `HostConfig` (port bindings, volume mounts), and `Cmd` to run `cardano-node run --config /opt/cardano/config/config.json ...`.
  4. Returns DevNetContainer (id, name, socketPath).
- **startContainer / stopContainer / removeContainer:**

  Use `docker.getContainer(container.id).start()/stop()/remove()`, each wrapped in Effect.tryPromise with CardanoDevNetError on failure.
- **getContainerStatus:** Inspects container via Docker API, returning `ContainerInspectInfo` for status fields (`State.Status`, `State.Running`, resource usage).

5.1.4. Multi-Instance Support & Resource Isolation

- **Isolation:** Unique container names (e.g., devnet-), distinct port ranges (node: 4001+, submit: 8090+), separate temp dirs.
- **Resource Limits:** Default Docker HostConfig can include CpuShares: 1024, Memory: 2GB (configurable in DevNetConfig).
- **Health Checks:** Optional polling loop to wait for node readiness (e.g., check socket presence before returning from startContainer).