



**ANASTASIA LABS**

## **Lucid Evolution 2.0**

Proof of Achievement - Milestone 1

<b>Project Id</b>	1300128
<b>Project Manager</b>	Jonathan Rodriguez
<b>Proposal Link</b>	Catalyst Proposal

## Contents

<b>1. Executive Summary .....</b>	<b>1</b>
1.1. Purpose of this Document .....	1
<b>2. Problem Analysis and Solution Rationale .....</b>	<b>2</b>
2.1. Identified Problems .....	2
2.2. Design Objectives .....	2
2.3. Key Advantages .....	3
<b>3. Architecture Overview .....</b>	<b>4</b>
3.1. Technology Stack .....	6
3.1.1. Effect-TS .....	6
3.1.2. Dockerode Integration .....	7
3.1.3. Real Cardano Node Approach .....	7
3.2. Configuration System Design .....	8
3.2.1. Comprehensive Era Support .....	8
3.3. Multi-Instance Architecture .....	9
3.3.1. Isolation Strategy .....	9
3.3.2. Resource Management .....	9
<b>4. Core Component Design .....</b>	<b>10</b>
4.1. Container Management Layer .....	10
4.1.1. DevNet Initialization Example .....	11
4.2. Provider Integration Layer .....	12
<b>5. Error Handling Strategy .....</b>	<b>13</b>
5.1. Tagged Error System .....	13
5.2. Operational Resilience .....	13
<b>6. Key Interfaces &amp; Types .....</b>	<b>14</b>
<b>7. User Requirements .....</b>	<b>17</b>
7.1. Functional Requirements .....	17
7.2. Non-Functional Requirements .....	18

<b>8. Feature Prioritization .....</b>	<b>19</b>
<b>9. Conclusion .....</b>	<b>20</b>
<b>10. Next Steps .....</b>	<b>21</b>
<b>11. Project Manager approval .....</b>	<b>22</b>

# Private Testnet SDK & L2 Provider Integration

## Design Specification Document

### 1. Executive Summary

**Lucid Evolution 2.0 - Private Testnet SDK & L2 Provider Integration** introduces a powerful **TypeScript SDK** designed to provide Cardano developers with a **reliable and efficient testing environment** for smart contracts and decentralized applications. It directly tackles the common frustrations of current development workflows, such as reliance on incomplete emulators, cumbersome shell scripts, and manual Docker commands. This SDK transforms the experience by offering developers **programmable control over real Cardano nodes**.

A key benefit is the ability to **spin up on-demand, disposable private testnets** using Dockerode and Testcontainers, all powered by a concise **Effect-TS API**. This fundamentally shifts infrastructure control into the codebase, eliminating external dependencies and enabling seamless integration with popular testing frameworks. Ultimately, Lucid Evolution 2.0 empowers developers to **focus on application logic rather than infrastructure management**, significantly enhancing productivity and reliability.

#### 1.1. Purpose of this Document

This Design Specification Document **outlines the architecture, functionalities, and user requirements** for the Lucid Evolution 2.0 Private Testnet SDK. It serves as a foundational document for the **development team, stakeholders, and community**, ensuring a shared understanding of the SDK's scope, design principles, and expected behavior.

## 2. Problem Analysis and Solution Rationale

This section details the challenges in current Cardano development workflows and explains the strategic reasoning behind the Lucid Evolution 2.0 SDK.

### 2.1. Identified Problems

Current Cardano development workflows face several significant obstacles:

- **External Tool Dependencies:** Developers often rely on bash scripts or raw Docker commands that are separate from their main testing code. This leads to unstable setups and inconsistent test results.
- **Inaccurate Environments:** Many existing testing tools use incomplete emulators that do not accurately simulate real Cardano blockchain behavior. This causes unreliable tests that fail to mirror mainnet conditions.
- **Complex Setup:** Manual configuration of private testnets requires extensive infrastructure knowledge and consumes significant developer time, delaying project initiation.
- **Poor Integration:** The lack of direct TypeScript integration with testing tools creates workflow inefficiencies, requiring developers to manage disparate systems.

### 2.2. Design Objectives

Our solution architecture is designed to overcome these problems by achieving the following core objectives:

- **Direct Control:** Provide intuitive TypeScript APIs for complete testnet lifecycle management.
- **Real Node Integration:** Utilize actual Cardano nodes instead of emulators to ensure authentic blockchain behavior and accurate testing.
- **Enhanced Developer Experience:** Offer simple default configurations for quick starts, alongside comprehensive customization options for advanced use cases.
- **Production Parity:** Ensure that private testnets behave identically to the Cardano mainnet, enabling highly reliable and confident contract testing.

## 2.3. Key Advantages

This SDK offers substantial advantages throughout the development lifecycle:

- **Significant Reduction** in developer setup time for testing environments.
- **Elimination of external script dependencies**, simplifying workflows and reducing maintenance.
- **Enhanced Reliability** through real Cardano node integration, leading to more accurate test results.
- **Improved Test Coverage** via consistent, reproducible environments, fostering higher quality dApps.

### 3. Architecture Overview

This section presents a high-level overview of the SDK's architecture, illustrating how its core components interact to provide private testnet functionality.

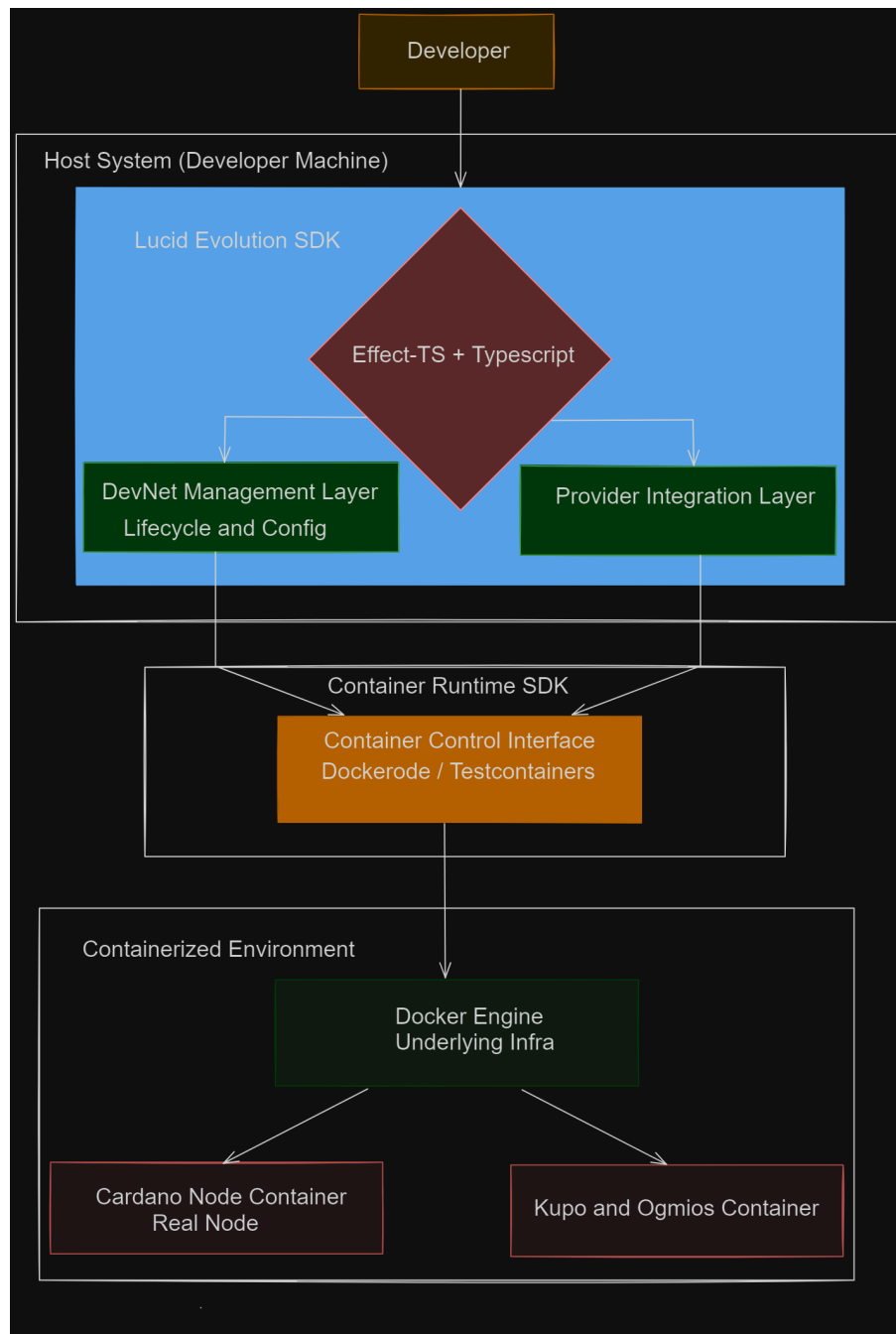


Figure 1: Private Testnet SDK

The diagram above illustrates the layered architecture of the Lucid Evolution 2.0 SDK. Here's a breakdown of each key component:

- **DevNet Management Layer:** This part of the SDK handles the full lifecycle of your testnets. It manages container orchestration (starting, stopping, and removing them), generates necessary configurations (like genesis files and cryptographic keys), and ensures proper resource and port isolation for each testnet.
- **Provider Integration Layer:** This layer connects your application to the testnet. It implements Kupo and Ogmios as a Provider interface, letting you submit transactions, query UTxOs, retrieve datums, and interact with the blockchain. It also extends this functionality to manage off-chain transactions, and on-chain settlement processes.
- **Container Runtime Libraries (Dockerode / Testcontainers):** These are Node.js libraries that provide the programming interface to interact directly with Docker. They allow the SDK to programmatically control Docker containers.
- **Underlying Infrastructure (Docker Runtime):** This is the foundation. It's the Docker engine running real Cardano nodes (supporting any era), Kupo indexer, and Ogmios server, each isolated within its own container. This ensures your tests run against environments that truly mimic the production network.



## 3.1. Technology Stack

The technology stack for Lucid Evolution 2.0 was selected to ensure robustness, type safety, and direct control over the testing environment.

### 3.1.1. Effect-TS

- **Decision:** We adopted Effect-TS for functional programming and robust error handling.
- **Rationale:**
  1. **Composability:** Complex container operations are composed functionally, resulting in modular and reusable code.
  2. **Error Safety:** Tagged Errors provide precise failure context and structured recovery strategies, improving debugging and resilience.
  3. **Type Safety:** Comprehensive static type checking eliminates common runtime errors, leading to more stable code.
  4. **Concurrency** Offers superior handling of concurrent container operations, crucial for efficient testnet management.

### Implementation Example:

```
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;
}> {}
```

Disclaimer: This code example illustrates the intended functionality and is subject to change as development progresses.

### 3.1.2. Dockerode Integration

- **Decision:** We utilize **Dockerode** for direct programmatic interaction with the Docker API.
- **Rationale:**
  1. **Programmatic Control:** Provides direct API access, eliminating reliance on brittle shell commands.
  2. **Structured Error Handling:** Enables structured error responses from Docker, allowing for precise failure diagnosis and recovery.
  3. **Resource Management:** Offers fine-grained control over container lifecycle and resources, optimizing utilization.
  4. **Cross-Platform Consistency:** Ensures uniform behavior across macOS, Linux, and Windows environments (via WSL2).

### 3.1.3. Real Cardano Node Approach

- **Decision:** We deploy actual Cardano nodes within Docker containers instead of emulators.
- **Rationale:**
  - **Authentic Behavior:** Guarantees identical transaction processing and protocol semantics to production networks.
  - **Full Protocol Compliance:** Provides complete support for all Cardano eras and features, ensuring comprehensive test coverage.
  - **Future Compatibility:** Automatically adapts to protocol upgrades, reducing maintenance overhead.
  - **Accurate Debugging:** Real node logs and metrics offer authentic insights for troubleshooting.

## 3.2. Configuration System Design

A flexible and secure configuration system is essential for adapting testnets to diverse development needs.

### 3.2.1. Comprehensive Era Support

- **Decision:** We support configuration for all Cardano eras (Byron, Shelley, Alonzo, Conway).
- **Rationale:**
  - **Historical Compatibility:** Allows testing of legacy transaction formats and behaviors.
  - **Era Transition Testing:** Enables validation of applications across hard fork boundaries.
  - **Complete Protocol Coverage:** Ensures support for all Cardano features and capabilities.
  - **Future Proofing:** The design is extensible for upcoming eras.
- **Structure Example:**

```
interface DevNetConfig {
    readonly containerName?: string;
    readonly image?: string;
    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>;
}
```

Disclaimer: This code example illustrates the intended functionality and is subject to change as development progresses.

### 3.3. Multi-Instance Architecture

The SDK is designed to support multiple, independently operating testnet instances, enabling parallel testing and complex simulation scenarios.

#### 3.3.1. Isolation Strategy

- **Decision:** We ensure complete isolation between testnet instances.
- **Rationale:**
  - **Parallel Testing:** Enables multiple test suites to run simultaneously without interference.
  - **Resource Safety:** Prevents resource conflicts and port collisions between instances.
  - **State Isolation:** Maintains independent blockchain states for different test scenarios.

#### 3.3.2. Resource Management

- **Decision:** We provide configurable resource limits with intelligent defaults.
- **Rationale:**
  - **System Protection:** Prevents resource exhaustion on developer machines.
  - **Predictable Performance:** Ensures consistent behavior across different hardware configurations.
  - **Cost Efficiency:** Optimizes resource utilization for development workflows.
  - **Scalability:** Supports both single and multi-instance scenarios efficiently.

## 4. Core Component Design

This section details the primary internal components of the SDK, describing their functions and design benefits.

### 4.1. Container Management Layer

This foundational layer handles all interactions with Docker, overseeing the entire lifecycle of Cardano node containers. It's responsible for the direct programmatic control over your DevNet instances.

- **Core Functions:** This layer exposes the following key functions for managing DevNet containers, all returning an **Effect** for robust error handling:
  - **make():** Creates and provisions a new DevNet container, making it ready for use.
  - **startContainer():** Starts a previously created or stopped DevNet container.
  - **stopContainer():** Stops a running DevNet container without removing its data.
  - **removeContainer():** Removes a DevNet container and its associated data, cleaning up resources.
  - **getContainerStatus():** Retrieves the current status and detailed information about a specific DevNet container.
- **Primary Operations:**
  - **Lifecycle Operations:** Manages container creation, starting, stopping, and removal with comprehensive error handling.
  - **Configuration Generation:** Dynamically generates genesis files and node configurations within secure temporary directories.
  - **Port Management:** Automatically allocates and manages ports for node communication, including collision detection.

#### 4.1.1. DevNet Initialization Example

To quickly spin up a private testnet and configure it with initial funds for testing, developers can use the following pattern:

```
const generateAccountForDevnet = () => {
  const privateKey = CML.PrivateKey.generate_ed25519();
  const publicKey = privateKey.to_public();
  const pubKeyHash = publicKey.hash();
  const keyhash = KeyHash.Decode.hex(pubKeyHash.to_hex());
  const addr = new EnterpriseAddress.EnterpriseAddress({
    networkId: NetworkId.NetworkId.make(0),
    paymentCredential: keyhash,
  });
  const hex = EnterpriseAddress.Encode.hex(addr);
  const address = EnterpriseAddress.Decode.hex(hex);
  const addressBech32 = Address.Encode.bech32(address);

  return {
    hex,
    keyhash,
    address,
    addressBech32,
    privateKey: privateKey.to_bech32(),
  } as const;
};

const account = generateAccountForDevnet();
console.log(`Generated address: ${JSON.stringify(account)}`);

const cluster = await Devnet.Cluster.makeOrThrow({
  clusterName: "devnet-1",
  shellyGenesis: {
    epochLength: 1000,
    initialFunds: {
      [account.hex]: 123456789, // use full address hex instead of keyhash
    },
  },
});

await Devnet.Cluster.startOrThrow(cluster);
```

Disclaimer: This code illustrates the intended functionality and is subject to change as development progresses.

- **Design Benefits:**

- **Reliability:** Ensures robust operation with comprehensive error handling and clear error messages.
- **Security:** Implements proper file permissions and secure key management practices.
- **Flexibility:** Allows for customizable configuration with sensible defaults.
- **Maintainability:** Promotes clear separation of concerns and a modular design.

## 4.2. Provider Integration Layer

This component is designed to manage and expose the necessary services for accessing Cardano blockchain data and submitting transactions.

- **Kupo Integration:** The SDK will configure and manage a Kupo indexer instance alongside the DevNet. Kupo provides a robust and efficient way to query UTxOs and historical blockchain data. The SDK will expose endpoints to allow dApps to connect to this Kupo instance for data retrieval.
- **Ogmios Integration:** The SDK will manage an Ogmios server instance, providing a WebSocket interface to interact directly with the Cardano node. Ogmios will be used for real-time node communication, including transaction submission and chain synchronization.
- **Transparent Failover:** If a DevNet container stops unexpectedly, DevNetProvider will automatically attempt to restart it (configurable in **DevNetConfig**).

## 5. Error Handling Strategy

A robust error handling system is critical for a reliable SDK. Error types in the SDK will come with documentation showing the errors and how to fix them with debug friendly error messages, ensuring stability and a positive developer experience.

### 5.1. Tagged Error System

- **Design Philosophy:** Precise error categorization using Effect-TS's Tagged Errors enables targeted recovery strategies.
- **Implementation Benefits:**
  - **Debugging Efficiency:** Provides clear error types with actionable remediation steps for developers.
  - **User Experience:** Generates helpful error messages with troubleshooting guidance.

### 5.2. Operational Resilience

The SDK is designed with built-in mechanisms to recover from unexpected issues and maintain system stability.

- **Recovery Mechanisms:**
  - **Container Cleanup:** Automatic removal of failed or terminated containers.
  - **Resource Reclamation:** Cleanup of temporary files and directories.
  - **Graceful Degradation:** Ensures partial functionality during component failures.



## 6. Key Interfaces & Types

This section details the essential TypeScript interfaces and types that developers will interact with when using the SDK, providing a clear contract for its functionality.

### Error Handling Definition

```
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;
}> {}
```

### DevNet Container Interface

```
interface DevNetContainer { id: string; name: string; }
```

### DevNet Cluster Interface

```
export interface DevNetCluster {
  readonly cardanoNode: DevNetContainer;
  readonly kupo?: DevNetContainer;
  readonly ogmios?: DevNetContainer;
  readonly networkName: string;
}
```

Disclaimer: These code blocks illustrates the intended functionality and is subject to change as development progresses.

## Individual Container Lifecycle Functions

These core functions manage the DevNet's state and return an **Effect** type for robust error handling.

```
// Starts a specific individual DevNet container.
start(container: DevNetContainer): Effect<void, CardanoDevNetError>
startOrThrow(container: DevNetContainer): Promise<void>

// Stops a specific individual DevNet container.
stop(container: DevNetContainer): Effect<void, CardanoDevNetError>
stopOrThrow(container: DevNetContainer): Promise<void>

// Removes a specific individual DevNet container.
remove(container: DevNetContainer): Effect<void, CardanoDevNetError>
removeOrThrow(container: DevNetContainer): Promise<void>

// Gets status information for a specific individual DevNet container.
getStatus(container: DevNetContainer): Effect<Docker.ContainerInspectInfo | undefined,
CardanoDevNetError>
getStatusOrThrow(container: DevNetContainer): Promise<Docker.ContainerInspectInfo |
undefined>
```

## DevNet Cluster Lifecycle Functions

These functions operate on the entire **DevNetCluster** to manage its lifecycle.

```
// Creates a new Cardano DevNet cluster, including optional Kupo and Ogmios containers.
make(config?: DevNetConfig): Effect<DevNetCluster, CardanoDevNetError>
makeOrThrow(config?: DevNetConfig): Promise<DevNetCluster> // Convenience method to throw
on error

// Starts all containers within a DevNet cluster (Cardano node, then Kupo/Ogmios).
start(cluster: DevNetCluster): Effect<void, CardanoDevNetError>
startOrThrow(cluster: DevNetCluster): Promise<void> // Convenience method to throw on
error

// Stops all containers within a DevNet cluster (Kupo/Ogmios first, then Cardano node).
stop(cluster: DevNetCluster): Effect<void, CardanoDevNetError>
stopOrThrow(cluster: DevNetCluster): Promise<void> // Convenience method to throw on error

// Removes all containers and associated network for a DevNet cluster.
remove(cluster: DevNetCluster): Effect<void, CardanoDevNetError>
removeOrThrow(cluster: DevNetCluster): Promise<void> // Convenience method to throw on
error
```

## DevNet Configuration Interface

This comprehensive interface allows for deep customization of the DevNet.

```
export interface DevNetConfig {
  readonly containerName?: string;
  readonly image?: string;
  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>;
}
```

Disclaimer: This code illustrates the intended functionality and is subject to change as development progresses.

## Genesis Configuration Types

Detailed types for each Cardano era's genesis file (**NodeConfig**, **ByronGenesis**, **ShelleyGenesis**, **AlonzoGenesis**, **ConwayGenesis**) and cryptographic key components (**KesKey**, **OpCert**, **VrfSkey**) enable granular control over network parameters.

## Default Configuration Constants

Pre-defined default values for each configuration type (e.g., **DEFAULT\_NODE\_CONFIG**, **DEFAULT\_BYRON\_GENESIS**, **DEFAULT\_DEVNET\_CONFIG**) ensure zero-config usability.

## 7. User Requirements

This section details the perspectives of developers and the specific capabilities (Functional Requirements) and performance, resource, and compatibility characteristics (Non-Functional Requirements) that the SDK must meet.

### 7.1. Functional Requirements

- **FR-001: DevNet Creation:** Users can call `make(config?)` to provision a new DevNet container with default or overridden settings.
- **FR-002: DevNet Startup:** Users can call `startContainer(container)` and await readiness (node sync).
- **FR-003: DevNet Teardown:** Users can stop and remove a container via `stopContainer(container)` and `removeContainer(container)`.
- **FR-004: Transaction Submission:** Via `DevNetProvider.submitTx(tx)`, users can submit serialized transactions to the testnet.
- **FR-005: UTxO and Datum Queries** `getUtxos`, `getDatum`, and related methods return on-chain data.
- **FR-006: Protocol Parameter Retrieval:** Users can fetch current protocol parameters via `getProtocolParameters()`.
- **FR-007: Custom Genesis Configuration:** Users can supply custom partial genesis objects for each era.
- **FR-008: Multi-Instance Support:** The SDK must allow parallel DevNets with isolated resources and no port collisions.
- **FR-009: Testing Framework:** Integration Provide Jest/Vitest matchers and utilities for automated assertions.
- **FR-010: Error Reporting All:** container and provider errors propagate as `CardanoDevNetError` with reason and message.

## 7.2. Non-Functional Requirements

- **Performance:**
  - **Startup Time:** Significantly fast initialization from `make()` to node readiness (socket available).
  - **Concurrent Instancing:** Support for multiple simultaneous DevNets with optimized resource usage.
- **Reliability & Resilience:**
  - **Uptime:** High uptime for container operations in local development contexts.
  - **Error Handling:** Graceful degradation—non-blocking errors do not crash the SDK.
- **Usability:**
  - **Zero-Config Defaults:** Reasonable defaults for DevNetConfig (networkMagic, ports, image).
  - **Type Safety:** Complete TypeScript definitions with IntelliSense.
  - **Documentation:** Comprehensive code comments, usage examples, and a Getting Started guide.
- **Compatibility & Portability:**
  - **Operating Systems:** macOS (with Docker Desktop), Linux (x86\_64), Windows 10/11 (with WSL2).
  - **Node.js:** Version 16+ (LTS), TypeScript 4.5+.
  - **Cardano Node Versions:** Compatible with Cardano Node 10.4.1; plan for backward compatibility to 10.3 and forward to next minor release.

## 8. Feature Prioritization

The table below describes the planned development order based on community feedback and project needs.

Priority	Feature	GitHub Issue
<b>P1</b>	<b>Core Testnet Functionality &amp; Basic Integration</b>	
	Private Testnet Lifecycle Management (Spin up/down, start/stop/restart, remove)	<a href="#">#510</a>
	Customizable Parameters (Protocol parameters, genesis file templates)	<a href="#">#514</a>
	Initial UTxO Distribution/Genesis Block Configuration	<a href="#">#511</a>
	Basic Lucid Provider Integration (Transaction submission, basic data queries)	<a href="#">#510</a>
	Transaction Simulation & Testing	<a href="#">#516</a>
	Quickstart Guides & Usage Examples Documentation	<a href="#">#525</a>

Priority	Feature	GitHub Issue
<b>P2</b>	<b>Advanced Management &amp; Wallet/Asset Features</b>	
	Multi-Instance Support (Parallel DevNets with isolation; simulate multi-node clusters)	<a href="#">#512</a>
	Time/Slot Simulation ( awaitSlot/ awaitBlock)	<a href="#">#513</a>
	Blockchain State Management (Save/load snapshots)	<a href="#">#513</a>
	Wallet & Asset Management (Wallet creation/restoration, Multi-Sig support, Token/NFT minting/burning, Asset distribution monitoring)	<a href="#">#523</a>
	Debugging Tools (Visual State Logging, Transaction Inspection, Simulated Network Conditions)	<a href="#">#518</a>

Priority	Feature	GitHub Issue
P3	Layer 2 Integration & Governance)	
	Stake Pool & Governance Simulation (Reward distribution, Delegation, Leader election, CIP-1694 workflows)	#524
	Advanced Tutorials Documentation	#525

## 9. Conclusion

This design specification documents the complete architecture, functional requirements, and prioritized feature set for Milestone 1 of Lucid Evolution 2.0. The SDK's modular, TypeScript-first approach powered by Effect-TS and Dockerode/Testcontainers ensures developers can quickly provision authentic Cardano testnets, with minimal setup.

## 10. Next Steps

### 1. **Milestone 2** (Core SDK Development):

SDK Implementation:

- Initial SDK to create and manage private testnets with features like configurable parameters, automated setup/teardown, and APIs for contract deployment.
- Dockerode/Testcontainers Integration:

On-demand provisioning of isolated test environments with pre-configured container images.

### 2. **Milestone 3** (Documentation & Onboarding):

- Creation of initial user guides and documentation to support developers in using the SDK.
- Development of onboarding materials, including quick start guides and example projects.

### 3. **Milestone 4** (Feature Development and Optimization):

- Implementation of additional features and optimizations based on developer feedback.
- Final adjustments to ensure robust performance and usability of the SDK.

### 4. **Final Milestone** (Project Closeout):

- Project Closeout Report: Summarizing project achievements, challenges, lessons learned.
- Demonstration Video: Showcasing key features and improvements of Lucid Evolution.

Upon approval, the team will begin Milestone 2, adhering to the roadmap and using the performance/risk metrics defined herein. Successful delivery of this SDK will significantly improve Cardano developer productivity, test coverage, and ecosystem reliability.



## 11. Project Manager approval

This section formally records the Project Manager's approval of this Design Specification Document for Milestone 1.

**Approved By:**

---

**(Signature)**

---

**(Date)**

---

**Comments (Optional):**

---

---