MARTORI.STUDIO

Ecosystem Architecture Blueprint: Netlify, Supabase, & Render

Building a digital landscape focused on Sovereignty and Slow Tech requires an infrastructure that feels lightweight, instantaneous, and structurally unshakeable. To support the five studio pillars—**Sanctuary, Vigil, Marrow, Ledger, and Drift**—the technical core leverages a modern decoupled stack where presentation, secure state handling, and custom execution compute are cleanly partitioned.

The Architectural Topology

Instead of a monolithic layout where a single server handles every requirement, responsibilities are split across three environments. This division ensures optimized resource delivery, granular security controls, and a unified context.

1. PRESENTATION

`Netlify`

Hosts the customer-facing interface. Serves static React/Vite assets instantly via worldwide edge caching. Acts as the visual portal for all five studio applications.

2. STATE & IDENTITY

`Supabase`

The data vault. Handles transactional integrity, user authentication, and Row-Level Security (RLS) policies directly on top of PostgreSQL.

3. CUSTOM ENGINE

`Render`

The background processing layer. Runs custom Python/Django environments for persistent scripts, data transformations, and custom APIs.

Functional Tier Responsibilities

Netlify — The Static & Interactive Edge

RESPONSIBILITY: INTERFACE DELIVERY & USER EXPERIENCE FLUIDITY

- **Client-Side Compilation:** Compiles user interfaces using React and Vite, pushing built chunks to global edge nodes so interactions remain fast and uninterrupted.
- **Route Delegation:** Employs structured pushstate redirects (***/* → /index.html***) ensuring client-side navigation between Sanctuary and Drift occurs instantaneously without requesting new pages from a server.
- **Variable Baking:** Environment flags prefixed with ***VITE_*** are structural constraints injected directly into compiled code at build time, facilitating fast browser-to-database requests.

Supabase — The Sovereign Data Vault

RESPONSIBILITY: IDENTITY ASSURANCE, PERSISTENCE, & ACCESS CONTROLS

- **Data Integrity:** Runs an enterprise-grade PostgreSQL foundation to map out relational definitions across product domains (e.g., matching security tables with active ledgers).
- **Row-Level Isolation (RLS):** Enforces strict mathematical ownership filters directly at the database level. Frontend consumers only read data matching their exact credentials: ***auth.uid() = user_id***.
- **State Synchronization:** Powers live listeners using realtime web sockets, allowing applications to react natively to back-end changes without manual reloading.

Render — The Compute Core

RESPONSIBILITY: HEAVY PROCESSING & PERSISTENT BACKGROUND ORCHESTRATION

- **Custom Execution:** Hosts long-running Python/Django applications that cannot live within browser environments due to complexity or secret exposure rules.
- **Ecosystem Workflows:** Manages heavy background tasks, long-poll event listeners, or custom metric logic for automation sequences.
- **Secure Internal Pipelines:** Interacts with Supabase using private, high-privilege connection keys (***SUPABASE_DATABASE_URL***), processing data safely outside public browser scrutiny.

Ecosystem Flow Example: When a user experiences a micro-ritual within *Drift* or updates parameters inside *Sanctuary*, Netlify displays the interactive interface instantly. The state change is saved straight to Supabase via authenticated browser APIs. Concurrently, if background tracking is needed, Render reads the structural shift in Supabase, evaluates data rules, and triggers background automations silently behind the scenes.