Database Architecture Report for NDFC Automation

Objective: This document outlines a recommended data storage strategy for a custom platform designed to automate Cisco VXLAN EVPN fabric configurations. The goal is to ensure data integrity, flexibility, and long-term maintainability.

1. Database Approach: Relational vs. NoSQL vs. Hybrid

A successful automation platform requires a database that is both robust and flexible.

- **Relational (e.g., pure PostgreSQL):** Offers excellent data integrity through strict schemas and relationships (foreign keys). However, it can be rigid and complex when storing the deeply nested configuration data found in the netascode models.
- **Non-Relational (e.g., MongoDB):** Provides maximum flexibility for evolving, nested data. Its weakness is the lack of built-in relational integrity; the application becomes solely responsible for preventing invalid states (e.g., a port assigned to a non-existent switch), which is a significant risk for a network source of truth.
- **Hybrid (PostgreSQL with JSONB):** This approach combines the strengths of both. It uses standard relational tables and constraints to guarantee the integrity of core network inventory (switches, interfaces) while using flexible JSONB columns to store complex, vendor-specific configuration details.

Recommendation: The **Hybrid approach is the best fit**. It provides the strict integrity required for a source of truth while offering the flexibility needed to handle complex netascode data models.

2. The Role of JSONB in the Hybrid Approach

JSONB is a native binary JSON data type in PostgreSQL and is the key to our recommended architecture.

- **Why it's the right tool:**
  - **Flexibility:** It perfectly stores nested configuration blocks from the netascode models (e.g., spanning_tree, multisite) without requiring a rigid table structure.
  - **Performance:** JSONB is indexed, allowing for efficient deep querying and partial, atomic updates.
  - **Future-Proofing:** If Cisco adds new attributes to a data model, **no database schema change is required**. The application simply writes the new keys into the existing JSONB column, making the system highly adaptable.

3. Data Modeling Strategy: Logical Entity vs. Direct Mapping

This addresses how we translate the netascode models into database tables.

- **Logical Entity Model:** We define tables based on core network concepts (switches, interfaces, networks). This model enforces strong business rules and relationships, leading to high data integrity.
- **Direct Mapping Model:** We create a table for every class in the source YAML structure. This is easier to map initially but can lead to weaker integrity and more complex queries.

Recommendation: A pragmatic combination of both is optimal.

1. Use the **Logical Entity** model for foundational, stable objects like Fabrics, Switches, Interfaces, and Networks to guarantee the integrity of the network inventory.
2. Use **Direct Mapping** (or store in JSONB) for complex, vendor-specific configuration blocks like fabric_links, linking them back to the core logical entities.

4. Proposed Storage Structure & Port Modify Workflow

This section details the proposed schema and a common operational workflow.

Proposed Hybrid Schema:
This schema uses a combination of relational tables for core inventory and JSONB columns for flexible configuration data. Below are examples of how these tables would be populated.
fabrics

id (PK)	name
1	nj-DC

switches

id (PK)	fabric_id (FK)	hostname	role	config (JSONB)
10	1	nj-leaf-1	leaf	{"management_ip": "10.0.0.11", "vtep_loopback_id": "101"}
11	1	nj-leaf-2	leaf	{"management_ip": "10.0.0.12", "vtep_loopback_id": "102"}

interfaces

id (PK)	switch_id (FK)	name	config (JSONB)
101	10	Ethernet1/10	{"mode": "access", "description": "ESXi Host 1"}

102	10	Ethernet1/11	{"mode": "access", "description": "App Server"}
201	11	Ethernet1/10	{"mode": "access", "description": "ESXi Host 2"}

network_attach_groups

id (PK)	name
5	esxi-compute-ports

networks

id (PK)	fabric_id (FK)	attach_group_id (FK)	name	vlan_id	config_details (JSONB)
50	1	5	Web-Servers-Network	101	{"gw_ip_address": "192.168.10.1/24"}

interface_assignments

attach_group_id (FK)	interface_id (FK, UNIQUE)
5	101
5	201

Workflow Example: modify_port(switch='leaf-1', port='Ethernet1/22', vlan=123)
This workflow atomically moves a port to a new network attachment group, ensuring the database state is always valid.

1. **Find State (Read):** The backend queries the database to get the primary keys for the switch, interface, and the target network attach group.
2. **Update Atomically (Write):** Within a single database transaction, the backend performs the move.
   - DELETE FROM interface_assignments WHERE interface_id = ?;
   - INSERT INTO interface_assignments (attach_group_id, interface_id) VALUES (?, ?);

The UNIQUE constraint on the interface_id column is critical. It makes it impossible for the database to contain an invalid state where a port is assigned to two groups at once.

5. Build Custom vs. Adopt Nautobot

The final decision is whether to build this database or adopt a pre-built Network Source of

Truth (NSoT).

- **Build Custom PostgreSQL Database:**
  - **Pros:** Total control over a lightweight schema perfectly tailored to the netascode models and Git-based workflow.
  - **Cons:** Higher upfront development and maintenance effort. Involves building features (IPAM, device management) that are standard in existing NSoTs.
- **Adopt Nautobot:**
  - **Pros:** Production-ready NSoT with a GUI, API, and a large feature set out of the box. Backed by a strong open-source community.
  - **Cons:** Its data model is more generic and opinionated. You would need to adapt the netascode model to fit Nautobot's structure, likely using Custom Fields or developing a Nautobot App.

Final Recommendation:

- If the project's scope is **strictly limited** to this NDFC Git workflow, and development resources are available, **building a custom database** provides a perfectly tailored solution.
- If the organization plans to build a **broader Network Source of Truth**, of which this automation is just one component, **adopting Nautobot** is the more strategic long-term choice. The initial adaptation effort will be offset by the vast number of features gained.