

This is a comprehensive plan for the major upgrade of your n8n_nginx setup to Version 3. As your program architect, I have analyzed your requirements and the subsequent clarifications to design a robust architecture and a phased implementation roadmap.

1. System Architecture Overview

The Version 3 architecture transforms the setup into a self-managed application suite with enhanced capabilities, centralized control, and robust data management.

1.1 Container Structure (Docker Compose Services)

We will maintain the existing services and introduce several new ones:

- **Core:** n8n, n8n_postgres, n8n_nginx, n8n_certbot.
- **[NEW] n8n_management:** The centerpiece of the upgrade. It will host the management web application, run a dedicated Nginx instance, manage backup scheduling/execution, and interact with the Docker API for container management and flow restoration.
- **[NEW/OPTIONAL]:** n8n_cloudflare (Cloudflare Tunnel), n8n_adminer (DB Management), n8n_portainer (or Agent), n8n_dozzle (Real-time logs and Shell Access), n8n_tailscale (Recommended for secure remote management VPN).

1.2 Management Container Technology Stack

To meet the requirements for a fast, modern, and highly functional interface capable of complex backend tasks, I recommend the following stack:

- **Base Image:** python:3.11-slim-bullseye (Lightweight, Debian-based).
- **Backend:** **FastAPI (Python)**. High-performance and ideal for building the APIs needed for Docker interaction, backup scripts, and authentication.¹
- **Frontend:** **Vue.js (Vite) + TailwindCSS**. Provides a reactive, component-based structure with rapid styling capabilities to achieve the desired modern aesthetics.
- **Web Server:** Nginx (inside the container) to serve the frontend and proxy to the FastAPI backend.²

1.3 Networking and Security

- **Access:** The management interface will be accessible via a dedicated subdomain (e.g., <https://loftaimgmt.loft.aero>) on the standard HTTPS port (443).
- **SSL:** n8n_certbot will be configured to manage certificates for both the primary n8n domain and the management subdomain.
- **Authentication:** Single admin username/password configured during setup.
- **Authorization:** Access restricted by IP/Subnet rules (e.g., 10.200.40.0/24) configured during setup.
- **Integrations (SSO):** Adminer and Dozzle will be proxied through the n8n_management Nginx. Access will be gated by the management authentication, providing a single-sign-on experience.

1.4 Data Persistence and NFS

- **Host Responsibility:** The Docker host is the sole NFS client. `setup.sh` will handle client installation, connection validation, and persistent mounting (using options `rw, sync, hard, intr`). The installation will halt if the NFS connection fails.
- **Container Access:** Containers access the NFS share via Docker volumes mapped to the host mount point (e.g., `/mnt/nvme/loftai`).

- **Structure:** Directories (backups/postgres, backups/n8n_config, documents, temp_restore, postgres_data) will be managed during setup.

2. Phased Implementation Roadmap

The project is divided into five phases with an estimated total duration of 8-10 weeks.

Phase 1: Core Infrastructure and Setup Script Overhaul (1-2 Weeks)

- **Goal:** Modify the setup process for the new architecture, handle NFS, and introduce optional components.
- **Tasks:**
 - Restructure the repository.
 - Overhaul setup.sh: Implement state management (resume on failure), NFS configuration (installation, testing, persistent mounting, directory setup), prompts for optional containers (Cloudflare, Portainer, Tailscale), and management container basics (subdomain, credentials, IP restrictions).
 - Configure Notification preferences (Email, Pushbullet, Twilio).
 - Create the skeleton docker-compose.yaml (v3) incorporating all potential services and dynamic generation logic.
 - Update README.md with NFS prerequisites and Cloudflare Tunnel setup instructions.

Phase 2: Management Interface - Design and Storyboarding (1 Week)

- **Goal:** Design the UI/UX and create interactive mockups for approval.
- **Tasks:**
 - Define Information Architecture (Dashboard, Backups, Flow Restore, Server Health, Settings, Tools).
 - Create Wireframes.
 - Develop 4 Design Mockups (2 Modern Minimal, 2 Dashboard Heavy) using colored icons.
 - Deliver an interactive, clickable HTML/CSS/JS storyboard of the chosen design using mock data.

Phase 3: Management Interface - Backend Development (2-3 Weeks)

- **Goal:** Implement the backend logic, API endpoints, and core background services.
- **Tasks:**
 - Initialize the FastAPI project and implement authentication/security.
 - Develop the Backup Engine (PostgreSQL pg_dump - custom format, compressed, and n8n configuration backups).
 - Implement the Scheduling Service (intuitive scheduling, no cron syntax) and Retention engine (e.g., using APScheduler).
 - Implement the Notifications System for backup status and system alerts.
 - Integrate Adminer and Dozzle via reverse proxy with SSO.

Phase 4: Management Interface - Frontend Development & Integration (2-3 Weeks)

- **Goal:** Connect the approved frontend design to the backend APIs.
- **Tasks:**
 - Initialize the Vue.js/Vite project based on the storyboard.
 - Implement the Dashboard, Backup Management UI (listing, downloads, manual trigger, .dump to .sql conversion tool), and Settings UI.
 - Integrate links/iframe for Tools (Adminer, Portainer, Dozzle).

Phase 5: Advanced Features and Final Polish (2 Weeks)

- **Goal:** Implement complex features (Flow Restore and Server Health) and finalize the project.
- **Tasks:**
 - Implement Server Health Backend: Docker API integration (container management) and Host metrics/control (Reboot/Shutdown).
 - Implement Server Health UI: Visual dashboard with secure confirmation dialogs and countdown delays.
 - Implement Flow Restore Engine: Logic to spin up temporary PostgreSQL containers (matching version), manage temporary storage, restore backup, extract flows, and re-import into production with auto-renaming (e.g., _restored_YYYYMMDD).
 - Implement Flow Restore UI: Step-by-step restoration wizard, including cleanup.
 - Comprehensive Testing, QA, and final documentation updates.

3. Agent Team and Prompts

We will utilize a team of specialized agents for execution.

1. DevOps Engineer (Shell & Docker Specialist)

- **Role:** setup.sh, Docker Compose, NFS configuration, host-level interactions.
- **Example Prompt (Phase 1):**
 "You are a Senior DevOps Engineer. Overhaul the existing setup.sh script. Key requirements: 1) Implement a status file mechanism to allow resuming installation. 2) Integrate NFS configuration: install nfs-common, prompt for server/export, validate the mount (exit on failure), configure a persistent mount via /etc/fstab (options: rw, sync, hard, intr), and interactively set up the required directory structure. 3) Add prompts for optional services: Cloudflare Tunnel (Token required), Portainer (CE or Agent), and Tailscale."

2. Backend Developer (Python/FastAPI Specialist)

- **Role:** n8n_management backend logic, API development, Docker SDK interaction, backup engine.
- **Example Prompt (Phase 3):**
 "You are a Backend Architect. Design the core backup engine using Python/FastAPI. Implement the scheduler (e.g., APScheduler) and the execution logic. The execution must use pg_dump (custom format, compressed) against the n8n_postgres container. Include logic for managing customizable retention policies (e.g., 24 hourly, 7 daily) and triggering notifications (Email/Twilio) on failure."

3. UI/UX Designer (Storyboarding & Aesthetics)

- **Role:** Storyboarding, prototyping.
- **Example Prompt (Phase 2):**
"You are a UI/UX specialist. Create four distinct storyboard designs (2 Modern Minimal, 2 Dashboard Heavy) for the n8n management interface. Use colored icons. Focus on the 'Health Dashboard' and an intuitive 'Backup Scheduler' (no cron syntax). Deliver interactive static HTML/TailwindCSS prototypes for review."

4. Frontend Developer (Vue.js/TailwindCSS Specialist)

- **Role:** Implementing the frontend and integrating it with the backend.
- **Example Prompt (Phase 4):**
"Implement a Vue.js 'Backup Management' interface based on the approved storyboard. It must connect to the FastAPI backend to: 1) List available backups with date, size, and type. 2) Provide a download button for each backup. 3) Include an intuitive scheduler configuration UI. 4) Implement the .dump to .sql conversion tool."

5. Integration and Database Specialist

- **Role:** Complex PostgreSQL interactions, flow extraction orchestration, temporary container management.
- **Example Prompt (Phase 5):**
"You are an Integration Specialist. Implement the backend logic for the Flow Extraction feature. This requires: 1) Using the Docker SDK to spin up a temporary PostgreSQL container matching a specific version. 2) Restoring a selected pg_dump file into it. 3) Executing SQL to list and extract a specific workflow as JSON. 4) Implementing logic to inject the JSON into the production database, automatically renaming the flow if a conflict exists. 5) Ensuring complete cleanup of the temporary container and files."

6. Technical Writer (Documentation)

- **Role:** Documentation overhaul, clarity, and user guides.
- **Example Prompt (Phase 1):**
"You are a Technical Writer. Update the README.md to include a comprehensive guide on the new Cloudflare Tunnel integration. Explain the security benefits. Provide step-by-step instructions on how to create the tunnel in the Cloudflare Zero Trust dashboard, obtain the Tunnel Token, and input it during the setup.sh process. Also, clearly explain how backups work, emphasizing that PostgreSQL backups contain the n8n flows."