n8n_nginx v3.0 Implementation Plan

Project Overview

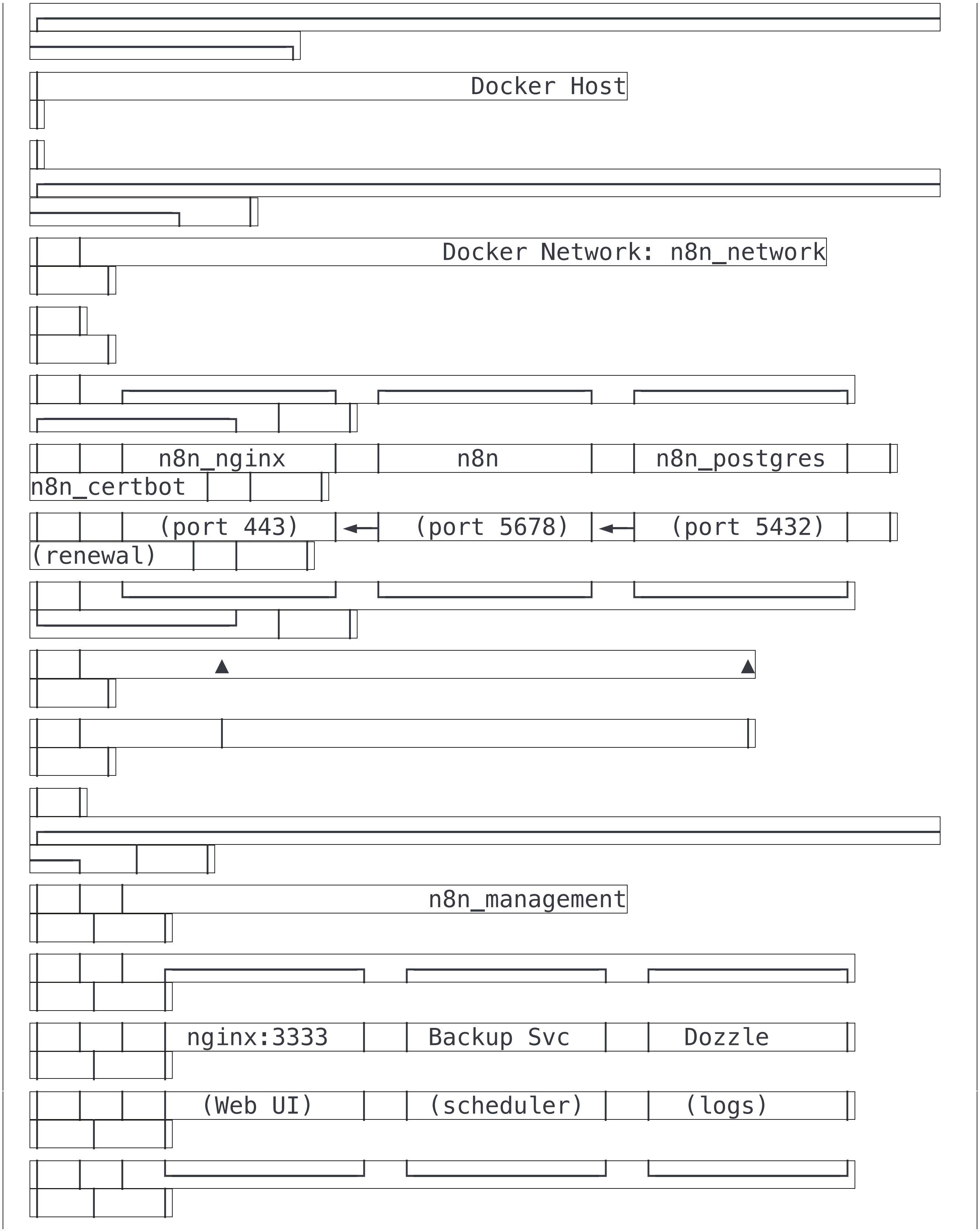
Vision

Transform n8n_nginx from a simple deployment stack into a comprehensive, enterprise-ready n8n management platform with:

- Automated backup and recovery system
- Web-based management console
- Container orchestration and monitoring
- NFS-based centralized storage
- Optional Cloudflare tunnel integration
- Optional Tailscale for secure remote management

Updated Architecture

```
┌──────────────────────────────────────────────────────────────────┐
│                         Internet / External Access               │
└──────────────────────────────────────────────────────────────────┘
           │                                    │
           │ Cloudflare Tunnel                  │ Direct (if no tunnel)
           │ (optional)                         │
           ▼                                    ▼
┌──────────────────────────┐          ┌──────────────────────────┐
│   n8n_cloudflared        │          │      Firewall            │
│   (tunnel container)     │          │   Port 443 forwarding    │
└──────────────────────────┘          └──────────────────────────┘
           │                                    │
           └──────────────┬─────────────────────┘
                          ▼
```


Docker Host
Docker Network: n8n_network
n8n_nginx
(port 443)
n8n
(port 5678)
n8n_postgres
(port 5432)
n8n_certbot
(renewal)
n8n_management
nginx:3333
(Web UI)
Backup Svc
(scheduler)
Dozzle
(logs)

```
│  │  │  ┌─────────────┐  ┌─────────────┐
│  │  │  │   Adminer   │  │ Flow Extract│
│  │  │  │ (optional)  │  │   Service   │
│  │  │  └─────────────┘  └─────────────┘
│  │
│  │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────────────┐
│  │  Portainer  │  │  Tailscale  │  │ Temp PG Container   │
│  │             │  │             │  │ (on-demand)         │
│  │ (optional)  │  │ (optional)  │  │ (for flow           │
│  │             │  │             │  │ extraction)         │
│  └─────────────┘  └─────────────┘  └─────────────────────┘
│  │
│
│
│
│  NFS Mount: /mnt/nvme/loftai (from loft-scale02.loft.aero)
│  │
│  ├── /backups/postgres
│  │
│  ├── /backups/n8n
│  │
│   │   ├── /backups/ssl
│   │   ├── /documents
│   │   └── /temp (for flow extraction)
│
└─────────────────────────────────────────────────────────────
└─────────────────────────────────────────────────────────────
```

Phase Breakdown

Phase 1: Foundation & Infrastructure

Goal: Setup.sh enhancements for NFS, notifications, and state management

Task ID	Task	Priority	Dependencies
1.1	Setup state persistence system	High	None
1.2	NFS package installation & configuration	High	1.1
1.3	NFS directory structure creation	High	1.2
1.4	Notification system setup (email/pushbullet/twilio)	Medium	1.1
1.5	Docker volume mapping for NFS paths	High	1.3

Phase 2: Cloudflare Tunnel & Tailscale Integration

Goal: Add optional tunnel and VPN containers

Task ID	Task	Priority	Dependencies
2.1	Cloudflare tunnel container configuration	High	Phase 1
2.2	Tunnel exposure options (webhooks/UI/management)	High	2.1
2.3	Tailscale container configuration	Medium	Phase 1
2.4	README documentation for Cloudflare setup	High	2.1, 2.2
2.5	README documentation for Tailscale setup	Medium	2.3

Phase 3: Backup System Core

Goal: Build the backup service with scheduling and retention

Task ID	Task	Priority	Dependencies
3.1	PostgreSQL backup service (pg_dump + compression)	High	1.3
3.2	n8n volume backup service	High	1.3
3.3	SSL certificate backup service	Medium	1.3

Task ID	Task	Priority	Dependencies
3.4	Environment/secrets backup (encrypted)	Medium	1.3
3.5	Backup scheduler with intuitive configuration	High	3.1-3.4
3.6	Retention policy engine	High	3.1
3.7	Backup failure detection & notification	High	1.4, 3.1
3.8	Manual backup trigger API	Medium	3.1

Phase 4: Management Container Base

Goal: Create the n8n_management container structure

Task ID	Task	Priority	Dependencies
4.1	Debian-based container Dockerfile	High	None
4.2	Internal nginx configuration (SSL via certbot)	High	4.1
4.3	SSL certificate for loftaimgmt.loft.aero	High	4.2
4.4	Supervisor/process manager setup	High	4.1
4.5	Docker socket access for container management	High	4.1
4.6	Integration with docker-compose.yaml	High	4.1-4.5

Phase 5: Web UI Storyboarding (No Backend)

Goal: Create 4 visual storyboard designs for approval

Task ID	Task	Priority	Dependencies
5.1	Modern Minimal Design #1	High	None
5.2	Modern Minimal Design #2	High	None
5.3	Dashboard Heavy Design #1	High	None
5.4	Dashboard Heavy Design #2	High	None
5.5	Icon asset collection from Flaticon	Medium	5.1-5.4
5.6	User approval and design selection	High	5.1-5.5

Phase 6: Web UI Implementation

Goal: Build the selected UI design with full backend

Task ID	Task	Priority	Dependencies
6.1	Technology stack finalization	High	5.6
6.2	Authentication system (single admin + subnet)	High	6.1
6.3	Dashboard home page	High	6.2
6.4	Backup management pages	High	6.3
6.5	Backup schedule configuration UI	High	6.4
6.6	Container health dashboard	High	6.3
6.7	Host system monitoring (CPU/RAM/Disk/NFS)	High	6.3
6.8	Container controls (stop/restart/compose)	High	6.6
6.9	Host power controls with confirmation	Medium	6.3
6.10	Dozzle integration for logs	High	6.6
6.11	Adminer integration (SSO)	Medium	6.2

Task ID	Task	Priority	Dependencies
6.12	Portainer link integration	Low	6.3

Phase 7: Flow Extraction System

Goal: Build the backup flow extraction and restoration feature

Task ID	Task	Priority	Dependencies
7.1	Backup browser and selector UI	High	6.4
7.2	Temporary PostgreSQL container spawner	High	7.1
7.3	PostgreSQL version detection from backup	High	7.2
7.4	Backup mounting and database restoration	High	7.2
7.5	Flow listing from restored database	High	7.4
7.6	Flow JSON export with complete data	High	7.5
7.7	Flow import to production (with auto-rename)	High	7.6
7.8	Temporary container cleanup service	Medium	7.2
7.9	pg_dump to SQL conversion tool	Low	6.4

Phase 8: Setup.sh Integration

Goal: Integrate all new features into interactive setup

Task ID	Task	Priority	Dependencies
8.1	NFS configuration prompts	High	Phase 1
8.2	Cloudflare tunnel prompts	High	Phase 2
8.3	Tailscale configuration prompts	Medium	Phase 2
8.4	Management container prompts	High	Phase 4
8.5	Adminer configuration prompts	Medium	Phase 6
8.6	Portainer detection and prompts	Medium	Phase 6
8.7	Backup configuration prompts	High	Phase 3
8.8	Notification setup prompts	Medium	1.4
8.9	State resume functionality	High	1.1
8.10	Port availability checking	Medium	8.4

Phase 9: Documentation & Polish

Goal: Complete README updates and final testing

Task ID	Task	Priority	Dependencies
9.1	README: Cloudflare tunnel walkthrough	High	Phase 2
9.2	README: NFS configuration guide	High	Phase 1
9.3	README: Management interface guide	High	Phase 6
9.4	README: Backup and restore guide	High	Phase 3, 7
9.5	README: Flow extraction guide	High	Phase 7
9.6	README: Tailscale setup guide	Medium	Phase 2
9.7	In-app help system	Medium	Phase 6
9.8	End-to-end testing	High	All

Task ID	Task	Priority	Dependencies
9.9	Version bump to 3.0.0	High	9.8

Agent Definitions & Prompts

Update Todos

```
 Create comprehensive v3.0 implementation plan
 Define project phases and milestones
 Creating agent task breakdown with prompts
 Design storyboard specifications
```

Below are the specialized agents I recommend, with their task domains and detailed prompts for each.

Agent 1: Infrastructure Agent

Purpose: Handle NFS setup, notifications, and state persistence in setup.sh

```
# Infrastructure Agent Prompt

## Context
You are working on the n8n_nginx v3.0 project. Read these files first:
- /home/user/n8n_nginx/README.md
- /home/user/n8n_nginx/setup.sh
- /home/user/n8n_nginx/docker-compose.yaml

## Your Tasks

### Task 1.1: State Persistence System
Create a state persistence mechanism for setup.sh that:
- Writes to a `.n8n_setup_state` file after each major step completes
- Stores: current step number, completed steps, all gathered configuration values
- On restart, detects this file and asks user: "Previous installation detected at step X. Resume or start fresh?"
```

- If resume: skip completed steps, restore variables, continue from last incomplete step
- If fresh: backup old state file, start new installation
- Format: JSON for easy parsing

### Task 1.2: NFS Package Installation

Modify setup.sh to add NFS setup section:

- Detect OS and install appropriate NFS packages:
  - Debian/Ubuntu: `nfs-common`
  - CentOS/RHEL/Rocky/Alma: `nfs-utils`
- Prompt for NFS server hostname/IP (e.g., `loft-scale02.loft.aero`)
- Prompt for NFS export path (e.g., `/mnt/nvme/loftai`)
- Prompt for local mount point (default: `/mnt/nfs/n8n`)
- Test mount with temporary mount, verify access, then unmount
- Add permanent mount to /etc/fstab with options: `rw,sync,hard,intr,nfsvers=4`
- Handle failure gracefully: if mount fails, save state, show error, exit with instructions

### Task 1.3: NFS Directory Structure

After successful NFS mount:

- Check if mount point is empty
- If empty, ask: "Create recommended directory structure? [Y/n]"
  - If yes, create:
    - `backups/postgres` - PostgreSQL dumps
    - `backups/n8n` - n8n volume backups
    - `backups/ssl` - SSL certificate backups
    - `documents` - General document storage
    - `temp` - Temporary files (flow extraction)
  - If no, prompt for each directory:
    - "PostgreSQL backup directory path:"
    - "n8n backup directory path:"

```markdown
    - "Documents directory path:"
    - Allow "skip" for optional directories
- If not empty, show contents and ask if this is the correct directory
- Ask: "Do you need additional directories? [y/N]"
  - If yes, prompt for: directory name, which container needs access, read-only or read-write

### Task 1.4: Notification System
Add notification configuration section:
- Ask: "Configure notifications for backup failures and alerts? [Y/n]"
- If yes, present options:
  1) Email (SMTP)
  2) Pushbullet
  3) Twilio SMS
  4) Skip notifications
- For Email:
  - SMTP server, port, username, password (hidden input)
  - From address, To address
  - Test send functionality
- For Pushbullet:
  - API token
  - Test push functionality
- For Twilio:
  - Account SID, Auth Token, From number, To number
  - Test SMS functionality
- Store notification config in `.n8n_notifications_config`
- If they skip: remind them to check management interface regularly

### Task 1.5: Docker Volume Mapping
Update docker-compose.yaml generation to:
```

~~~
- Add NFS-based volumes for containers that need them:
  - n8n: documents directory (read-write)
  - n8n_postgres: no direct NFS (backups go TO NFS, not FROM)
  - n8n_management: full NFS access for backup operations
- Use bind mounts from host NFS mount point
- Example volume syntax:
  ```yaml
  volumes:
    - /mnt/nfs/n8n/documents:/home/node/documents:rw
    - /mnt/nfs/n8n/backups:/backups:rw
~~~

Output Requirements

- Modify setup.sh in place
- Follow existing code style (use existing print_* functions, confirm_prompt, etc.)
- Add new sections after existing configuration but before deployment
- Update CONFIG_FILE saves to include new NFS and notification settings
- Test all error paths and edge cases

~~~

Agent 2: Tunnel & VPN Agent
Purpose: Cloudflare tunnel and Tailscale container integration


```markdown
# Tunnel & VPN Agent Prompt


## Context
You are working on the n8n_nginx v3.0 project. Read these files first:
- /home/user/n8n_nginx/README.md
~~~

- /home/user/n8n_nginx/setup.sh
- /home/user/n8n_nginx/docker-compose.yaml

## Your Tasks

### Task 2.1: Cloudflare Tunnel Container

Add n8n_cloudflared container configuration:

- Container: `cloudflare/cloudflared:latest`
- Container name: `n8n_cloudflared`
- Command: `tunnel --no-autoupdate run --token <TOKEN>`
- No exposed ports (tunnel handles routing)
- Restart: always
- Network: n8n_network
- Make it optional (commented out if user says no)

### Task 2.2: Tunnel Exposure Options

Modify setup.sh to ask about tunnel exposure:

1. "Expose n8n webhooks externally via tunnel? [Y/n]" (default: yes)
   - Required for external workflow triggers
2. "Expose n8n UI externally via tunnel? [y/N]" (default: no)
   - Warn: "The n8n UI will be accessible from the internet. Ensure strong authentication."
3. "Expose management interface via tunnel? [y/N]" (default: no)
   - Warn: "NOT RECOMMENDED. Management should remain internal-only."

Store responses and use them when configuring the tunnel ingress rules in the README instructions.

### Task 2.3: Tailscale Container

Add n8n_tailscale container configuration:

- Container: `tailscale/tailscale:latest`

~~~text
- Container name: `n8n_tailscale`
- Environment:
  - TS_AUTHKEY: (user provides during setup)
  - TS_STATE_DIR: /var/lib/tailscale
  - TS_EXTRA_ARGS: --advertise-tags=tag:n8n
- Volumes:
  - tailscale_state:/var/lib/tailscale
  - /dev/net/tun:/dev/net/tun
- Capabilities: NET_ADMIN, NET_RAW
- Network mode: host (for VPN functionality)
- Make it optional

Setup.sh prompts:
- "Set up Tailscale for secure remote management? [y/N]"
- If yes: "Enter your Tailscale auth key (from
admin.tailscale.com):"
- Explain: "Tailscale provides secure remote access to internal
services without exposing ports."

### Task 2.4: Cloudflare Tunnel README Section
Create comprehensive README documentation:

```markdown
## Cloudflare Tunnel Setup

### Prerequisites
1. A Cloudflare account with your domain added
2. Domain must use Cloudflare nameservers

### Step 1: Create a Tunnel
1. Go to [Cloudflare Zero Trust
Dashboard](https://one.dash.cloudflare.com/)
2. Navigate to **Access** → **Tunnels**
~~~

```
3. Click **Create a tunnel**
4. Name your tunnel (e.g., `n8n-tunnel`)
5. Copy the tunnel token (starts with `ey...`)

### Step 2: Configure Tunnel Ingress
In the Cloudflare dashboard, add these public hostnames:

| Subdomain | Service | Path |
|-----------|---------|------|
| n8n.yourdomain.com | http://n8n_nginx:443 | /webhook/* |
[Add more rows based on user selections]

### Step 3: Enter Token During Setup
When running setup.sh, paste your tunnel token when prompted.

### Why Use a Tunnel?
- No need to open port 443 on your firewall
- Cloudflare handles DDoS protection
- Easy to disable/enable without firewall changes
- Works behind NAT/CGNAT
```

Task 2.5: Tailscale README Section

Create Tailscale documentation section explaining:

- What Tailscale is and why use it
- How to create an auth key
- How to access services after connection
- Recommended ACL policies for n8n

Output Requirements

- Update docker-compose.yaml generation in setup.sh
- Add new configuration sections to setup.sh
- Create README sections as specified

- Ensure containers are properly commented out when not selected

~~~
---

### Agent 3: Backup System Agent
**Purpose**: Build the complete backup service

```markdown
# Backup System Agent Prompt

## Context
You are working on the n8n_nginx v3.0 project. The backup system
runs inside the n8n_management container.

## Your Tasks

### Task 3.1: PostgreSQL Backup Service
Create a backup script `backup_postgres.sh`:
- Connect to n8n_postgres container
- Use pg_dump with custom format (-Fc) for pg_restore
compatibility
- Compress with gzip (configurable compression level)
- Filename format: `postgres_backup_YYYY-MM-DD_HH-MM-SS.dump.gz`
- Store in NFS backup directory
- Log success/failure with timestamp
- Return exit code for scheduler

### Task 3.2: n8n Volume Backup Service
Create `backup_n8n_volume.sh`:
- Backup /home/node/.n8n from n8n container
- Use tar with gzip compression
- Filename: `n8n_volume_YYYY-MM-DD_HH-MM-SS.tar.gz`
~~~

- Exclude temporary files, caches
- Store in NFS backup directory

### Task 3.3: SSL Certificate Backup

Create `backup_ssl.sh`:
- Backup from letsencrypt Docker volume
- Include all certificate files and account data
- Filename: `ssl_backup_YYYY-MM-DD_HH-MM-SS.tar.gz`

### Task 3.4: Environment Backup (Encrypted)

Create `backup_env.sh`:
- Collect: docker-compose.yaml, .env files, cloudflare.ini (sensitive!)
- Encrypt with user-provided passphrase (set during setup)
- Use openssl enc with AES-256-CBC
- Filename: `env_backup_YYYY-MM-DD_HH-MM-SS.tar.gz.enc`

### Task 3.5: Backup Scheduler

Create backup scheduler service:
- Config file format (YAML or JSON):

```yaml
schedules:
  postgres:
    enabled: true
    frequency: daily  # hourly, daily, weekly, monthly
    time: "02:00"     # for daily/weekly/monthly
    day: 1            # for weekly (1=Monday) or monthly (1-28)
  n8n_volume:
    enabled: true
    frequency: daily
    time: "02:30"
  ssl:
    enabled: true
    frequency: weekly
    time: "03:00"
    day: 7
```

- Use Python schedule library or cron-like implementation
- Support for multiple schedules per backup type
- Next run time calculation and display
- Manual trigger capability via Unix socket or HTTP API

Task 3.6: Retention Policy Engine

Create retention manager:

- Config format:

```
retention:
  hourly: 24    # keep last 24 hourly backups
  daily: 7      # keep last 7 daily backups
  weekly: 4     # keep last 4 weekly backups
  monthly: 12   # keep last 12 monthly backups
```

- Run after each backup completes
- Identify backup age and category
- Delete expired backups
- Log deletions
- Never delete last remaining backup of each type

Task 3.7: Backup Failure Handling

Implement failure detection:

- Capture exit codes from backup scripts
- Retry failed backup after 5 minutes
- If retry fails, trigger notification
- Log failure details for debugging
- Set status flag readable by web UI

- Create `backup_status.json`:

```json
{
  "postgres": {
    "last_success": "2025-12-04T02:00:00Z",
    "last_attempt": "2025-12-04T02:00:00Z",
    "status": "success",
    "error": null
  }
}
```

Task 3.8: Manual Backup API

Create HTTP endpoint or Unix socket API:

- POST /api/backup/postgres - trigger immediate postgres backup
- POST /api/backup/n8n - trigger immediate n8n volume backup
- POST /api/backup/all - trigger all backups
- GET /api/backup/status - return backup_status.json
- Protect with same auth as management UI

Task 3.9: pg_dump to SQL Conversion

Create conversion utility:

- Input: .dump.gz file path
- Output: .sql file in same directory
- Use pg_restore with -f flag to output SQL
- Handle large files efficiently

Output Requirements

- All scripts go in management container at /opt/backup/
- Python preferred for scheduler (easier scheduling logic)
- Shell scripts for actual backup operations (simpler Docker interaction)
- Include Dockerfile additions for required packages

### Agent 4: Management Container Agent
Purpose: Build the n8n_management container infrastructure

~~~markdown
# Management Container Agent Prompt

## Context
You are building the n8n_management container for the n8n_nginx v3.0 project.

## Your Tasks

### Task 4.1: Container Dockerfile
Create `management/Dockerfile`:
```dockerfile
FROM debian:bookworm-slim

# System packages
RUN apt-get update && apt-get install -y \
    nginx \
    python3 \
    python3-pip \
    python3-venv \
    postgresql-client \
    docker.io \
    curl \
    jq \
    supervisor \
    && rm -rf /var/lib/apt/lists/*
~~~

```
# Python virtual environment
RUN python3 -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Python dependencies
COPY requirements.txt /tmp/
RUN pip install -r /tmp/requirements.txt

# Directory structure
RUN mkdir -p /opt/backup /opt/webapp /var/log/management

# Copy application files
COPY backup/ /opt/backup/
COPY webapp/ /opt/webapp/
COPY nginx/ /etc/nginx/
COPY supervisor/ /etc/supervisor/conf.d/

EXPOSE 443

CMD ["supervisord", "-n", "-c",
"/etc/supervisor/supervisord.conf"]
```

Task 4.2: Internal Nginx Configuration

Create `management/nginx/management.conf`:

- Listen on 443 (SSL)
- SSL certificate paths for loftaimgmt.loft.aero
- Proxy pass to webapp (Flask/FastAPI on port 8000)
- Location for Adminer (/adminer)
- Location for Dozzle proxy (/logs)
- Subnet restriction support (optional, configurable)

Task 4.3: SSL Certificate for Management

Modify setup.sh to:

- Ask for management domain (default: loftaimgmt.loft.aero)
- Request SSL certificate using same DNS-01 challenge
- Store certificate in letsencrypt volume
- Configure certbot renewal for both domains

Task 4.4: Supervisor Configuration

Create `management/supervisor/supervisord.conf`:

- nginx (web server)
- backup-scheduler (Python scheduler)
- webapp (Flask/FastAPI application)
- dozzle (optional, if embedded)

Task 4.5: Docker Socket Access

Configure container for Docker management:

- Mount docker.sock: `/var/run/docker.sock:/var/run/docker.sock`
- Add capability if needed
- Ensure webapp can execute docker commands
- Security consideration: limit to specific operations

Task 4.6: Docker Compose Integration

Add to docker-compose.yaml:

```yaml
management:
  build: ./management
  container_name: n8n_management
  restart: always
  ports:
    - "443:443"  # This conflicts! Need different port or routing
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock:ro
    - /mnt/nfs/n8n:/nfs:rw
    - letsencrypt:/etc/letsencrypt:ro
    - ./management/config:/config:ro
  environment:
    - ADMIN_USERNAME=${MGMT_ADMIN_USER}
    - ADMIN_PASSWORD_HASH=${MGMT_ADMIN_HASH}
    - ALLOWED_SUBNETS=${MGMT_ALLOWED_SUBNETS}
  depends_on:
    - postgres
    - n8n
  networks:
    - n8n_network
```

Note: Need to resolve port 443 conflict with n8n_nginx. Options:

1. Management on different external port (e.g., 8443)
2. Route through n8n_nginx with SNI-based routing
3. Use separate IP if available

Recommend option 1: Management on port 8443, so:

- https://loftai.loft.aero:443 → n8n
- https://loftaimgmt.loft.aero:8443 → management

Or, since internal-only, could use port 443 internally but map to 8443 externally.

Output Requirements

- Create management/ directory structure
- All configuration files ready for build
- Update setup.sh to prompt for management configuration
- Handle port selection and conflict detection

~~~

Agent 5: Storyboard Designer Agent
Purpose: Create 4 visual UI storyboard designs

```markdown
~~~

```
# Storyboard Designer Agent Prompt

## Context
You are creating UI storyboard designs for the n8n management
interface. These are functional HTML/CSS mockups that work
without backend - all data is hardcoded for visualization.

## Design Requirements
- Internal-only management interface
- Single admin user
- Modern, professional appearance
- Colored icons from Flaticon
- Fast and responsive feel
- Built-in help tooltips

## Pages to Design

### 1. Login Page
- Username and password fields
- "Remember me" checkbox
- Clean, centered card design

### 2. Dashboard Home
- System health overview (all containers status)
- Quick stats: backup status, disk usage, last backup time
- Quick action buttons
- Alerts/warnings section

### 3. Backup Management
- List of all backups (sortable, filterable)
- Backup type tabs (Postgres, n8n, SSL, Env)
- Download button per backup
- Delete button (with confirmation)
- "Backup Now" button

### 4. Backup Schedule
- Visual schedule editor (NOT cron syntax)
- Enable/disable toggles per backup type
- Frequency selector (hourly/daily/weekly/monthly)
- Time picker
- Retention settings per type

### 5. Container Dashboard
- Card per container showing:
  - Name, image, status (running/stopped)
  - Uptime, memory usage
  - Action buttons (restart, stop, logs)
- Overall compose controls (down/up)
- Expandable container details (config, ports, volumes)

### 6. Host System Monitor
- Real-time CPU usage graph
- Memory usage (used/total/percent)
- Disk usage per mount (including NFS)
- Host info (hostname, OS, uptime)
- Power controls (shutdown/reboot with confirmation modal)

### 7. Flow Extraction
- Backup selector dropdown
- "Load Backup" button
- Progress indicator during temp container spin-up
- Flow list from backup
- Per-flow actions: Download JSON, Import to Production
- Cleanup status
```

### 8. Settings Page
- Notification configuration
- Subnet restrictions
- Password change
- Log level settings

### 9. Log Viewer (Dozzle Integration)
- Container selector
- Live log stream
- Search/filter box
- Clear/pause controls

## Create 4 Variations

### Design A: Modern Minimal #1
- Color scheme: White background, blue (#2563eb) accents, gray text
- Typography: Inter or system-ui, generous line-height
- Layout: Sidebar navigation, lots of whitespace
- Cards: Subtle shadows, rounded corners (8px)
- Animations: Subtle fade-ins

### Design B: Modern Minimal #2
- Color scheme: Light gray (#f8fafc) background, teal (#0d9488) accents
- Typography: Plus Jakarta Sans
- Layout: Top navigation bar, content below
- Cards: Flat design, thin borders
- Animations: None (snappy feel)

### Design C: Dashboard Heavy #1

- Color scheme: Dark sidebar (#1e293b), white content, orange (#f97316) accents
- Typography: Roboto
- Layout: Collapsible sidebar, dense information
- Charts: Line charts for CPU/memory, pie for disk
- Cards: Colored headers, metrics prominent
- Animations: Chart animations on load

### Design D: Dashboard Heavy #2

- Color scheme: Full dark mode, purple (#8b5cf6) and cyan (#06b6d4) accents
- Typography: JetBrains Mono for metrics, Inter for text
- Layout: Grid-based dashboard, draggable widgets (visual only)
- Charts: Area charts, radial progress for disk
- Cards: Glassmorphism effect
- Animations: Smooth transitions

## Technical Requirements

- Each design is a standalone HTML file with embedded CSS
- Use hardcoded data (no JavaScript fetch)
- Use placeholder images for icons (note which Flaticon icons to find)
- Responsive (works on 1024px+ screens)
- Include all pages in a single HTML using tabs/sections
- Add comments noting where dynamic data would come from

## Deliverables

- `storyboard_modern_1.html`
- `storyboard_modern_2.html`
- `storyboard_dashboard_1.html`
- `storyboard_dashboard_2.html`
- `icon_requirements.md` - list of icons needed with descriptions

```
## Icon Categories Needed
- Container status (running, stopped, error)
- Backup (database, archive, download, delete)
- System (cpu, memory, disk, network)
- Actions (refresh, power, settings, logout)
- Navigation (dashboard, backups, containers, settings)
- Alerts (success, warning, error, info)
```

Agent 6: Web Application Agent

Purpose: Build the actual web application after design approval

```
# Web Application Agent Prompt

## Context
You are building the management web application for n8n_nginx
v3.0. Wait for design approval before starting.

## Prerequisites
- Selected storyboard design file
- Technology stack decision (recommend based on selected design)

## Technology Recommendations

### If Modern Minimal Selected:
- **Backend**: Python + Flask
- **Frontend**: HTMX + Alpine.js + TailwindCSS
- **Why**: Fast development, minimal JavaScript, clean
architecture

### If Dashboard Heavy Selected:
- **Backend**: Python + FastAPI
- **Frontend**: Vue 3 + TailwindCSS or React + Chakra UI
- **Why**: More interactivity needed for charts, real-time
updates

## Your Tasks

### Task 6.1: Technology Finalization
Based on selected design:
1. Confirm tech stack with user
2. Create project structure
3. Set up development environment
4. Create requirements.txt / package.json

### Task 6.2: Authentication System
Implement:
- Single admin user (username + bcrypt password hash)
- Session-based auth (secure cookies)
- Optional subnet restriction check
- Login/logout endpoints
- Password change functionality
- Session timeout (configurable, default 24h)

### Task 6.3-6.12: Page Implementation
Implement each page according to selected design:
- Use real data from:
  - Docker API for container info
  - /nfs/backups for backup listings
  - /proc for system stats
  - backup_status.json for backup health
- Implement all interactions (buttons, forms, modals)
- Add loading states
- Add error handling
- Add help tooltips

## API Endpoints Needed
```

POST /api/auth/login POST /api/auth/logout POST /api/auth/change-password

GET /api/dashboard/stats GET /api/containers POST /api/containers/{id}/restart POST /api/containers/{id}/stop GET /api/containers/{id}/logs

GET /api/backups GET /api/backups/{type} POST /api/backups/trigger/{type} DELETE /api/backups/{id} GET /api/backups/{id}/download

GET /api/schedule PUT /api/schedule GET /api/retention PUT /api/retention

GET /api/system/stats POST /api/system/shutdown POST /api/system/reboot

GET /api/flows/backups POST /api/flows/load/{backup_id} GET /api/flows/list GET /api/flows/{id}/download POST /api/flows/{id}/import POST /api/flows/cleanup

```


## Output Requirements
- All code in management/webapp/
- Include unit tests
- Include API documentation
- Follow security best practices
```

Agent 7: Flow Extraction Agent

Purpose: Build the backup flow extraction system

```
# Flow Extraction Agent Prompt

## Context
Building the flow extraction feature that allows users to
recover individual workflows from PostgreSQL backups.

## Your Tasks
```

~~~
### Task 7.1: Backup Browser UI
Create UI component:
- List all postgres backups from NFS
- Show: filename, date, size
- Sort by date (newest first)
- Filter by date range
- Select backup button

### Task 7.2: Temporary Container Spawner
Create Python service to:
- Accept backup file path
- Detect PostgreSQL version from backup metadata
- Pull matching pgvector/pgvector:pgXX image
- Create temporary container:
  - Random name: `pg_restore_temp_{uuid}`
  - No exposed ports (internal network only)
  - Mount: backup file, temp data directory
  - Environment: POSTGRES_USER, PASSWORD from main config
- Wait for container to be healthy
- Return container ID and connection info

### Task 7.3: Version Detection
Implement backup version detection:
```python
def detect_pg_version(backup_path):
    # pg_restore -l outputs header with version info
    # Parse: "; Archive created at ... by pg_dump (PostgreSQL) 16.x"
    # Return major version number (e.g., 16)
~~~

Task 7.4: Database Restoration

After temp container is running:

1. Create temporary database
2. pg_restore backup into temp database
3. Verify restoration success
4. Return success/failure status

Task 7.5: Flow Listing

Query restored database for workflows:

```sql
SELECT
    id,
    name,
    active,
    created_at,
    updated_at,
    (SELECT COUNT(*) FROM workflow_entity) as node_count
FROM workflow_entity
ORDER BY name;
```

Return as JSON list for UI display.

Task 7.6: Flow Export

Export single workflow as importable JSON:

```sql
SELECT json_build_object(
    'name', name,
    'nodes', nodes,
    'connections', connections,
    'settings', settings,
    'staticData', "staticData",
    'pinData', "pinData"
) FROM workflow_entity WHERE id = :id;
```

- Wrap in n8n import format
- Include version info

- Save to temp file and return download path

Task 7.7: Flow Import to Production

Import flow into live n8n_postgres:

1. Parse exported JSON
2. Check for name conflicts:

```
3. SELECT id FROM workflow_entity WHERE name = :name;
```

4. If conflict, auto-rename: `{name}_restored_{timestamp}`
5. Insert into production database
6. Return success and final workflow name

Task 7.8: Cleanup Service

Implement cleanup:

- Track spawned temp containers
- Set timeout (default: 30 minutes of inactivity)
- Auto-cleanup on session end
- Manual cleanup button
- Remove temp container
- Remove copied backup file (if on local temp)
- Remove temp directory

Task 7.9: SQL Export Tool

Create backup format converter:

- Input: .dump.gz file
- Decompress
- Use pg_restore -f to output SQL
- Compress output as .sql.gz
- Return download path

Output Requirements

- All code in management/webapp/services/flow_extraction/
- Include error handling for all Docker operations

- Log all operations for debugging
- Include timeout handling

~~~
---

### Agent 8: Setup.sh Integration Agent
**Purpose**: Integrate all new features into setup.sh

```markdown
# Setup.sh Integration Agent Prompt

## Context
Integrate all v3.0 features into the existing setup.sh script.

## Current Setup.sh Flow
1. Welcome screen
2. Root check
3. LXC container check
4. Docker installation
5. System checks
6. DNS provider configuration
7. Domain configuration
8. Database configuration
9. Container names
10. Email & timezone
11. Encryption key
12. Portainer agent
13. Summary & confirmation
14. Generate files
15. Deploy
~~~

## New Sections to Add (in order)

### After Step 6 (DNS Provider):
- Cloudflare tunnel configuration (Task 8.2)
- Tailscale configuration (Task 8.3)

### After Step 12 (Portainer):
- NFS configuration (Task 8.1)
- Management container setup (Task 8.4)
- Management domain & port (Task 8.10)
- Backup configuration (Task 8.7)
- Notification setup (Task 8.8)
- Adminer configuration (Task 8.5)
- Full Portainer detection & options (Task 8.6)

### New: State Resume (Task 8.9)
At the very beginning, after welcome:
- Check for .n8n_setup_state file
- If exists, offer resume or fresh start

## Task Details

### Task 8.1: NFS Prompts
```bash
configure_nfs() {
    print_section "NFS Storage Configuration"

    if ! confirm_prompt "Configure NFS storage for backups and documents?"; then
        NFS_ENABLED=false
        return
    fi
    # Install NFS packages
    # Prompt for server
    # Prompt for export path
    # Prompt for local mount point
    # Test mount
    # Configure directories
    # Add to fstab
    # Save to state
}
```

Task 8.2: Cloudflare Tunnel Prompts

```
configure_cloudflare_tunnel() {
    print_section "Cloudflare Tunnel Configuration"

    echo "Cloudflare Tunnels provide secure external access without opening firewall ports."
    echo "Prerequisites: Domain on Cloudflare, tunnel created in Zero Trust dashboard"

    if ! confirm_prompt "Set up Cloudflare Tunnel?"; then
        TUNNEL_ENABLED=false
        return
    fi

    # Prompt for tunnel token
    # Ask about webhook exposure
    # Ask about UI exposure (with warning)
    # Ask about management exposure (with strong warning)
    # Save to state
}
```

Task 8.3: Tailscale Prompts

```
configure_tailscale() {
    print_section "Tailscale VPN Configuration"

    echo "Tailscale provides secure remote access to your management interface."

    if ! confirm_prompt "Set up Tailscale for remote management?"; then
        TAILSCALE_ENABLED=false
        return
    fi

    # Prompt for auth key
    # Save to state
}
```

Task 8.4: Management Container Prompts

```
configure_management() {
    print_section "Management Interface Configuration"

    if ! confirm_prompt "Set up web-based management interface?"; then
        MANAGEMENT_ENABLED=false
        return
    fi

    # Domain for management
    prompt_with_default "Management interface domain" "loftaimgmt.loft.aero" "MGMT_DOMAIN"

    # Admin credentials
    prompt_with_default "Admin username" "admin" "MGMT_ADMIN_USER"
    prompt_with_default "Admin password" "" "MGMT_ADMIN_PASS"
    # Generate hash

    # Subnet restriction
    if confirm_prompt "Restrict management access to specific subnets?"; then
        prompt_with_default "Allowed subnets (comma-separated)" "10.0.0.0/8,172.16.0.0/12,192.168.0.0/16" "MGMT_ALLOWED_SUBNETS"
    fi

    # Save to state
}
```

Task 8.5: Adminer Prompts

```
configure_adminer() {
    print_section "Database Management (Adminer)"

    if ! confirm_prompt "Include Adminer for database management?"; then
        ADMINER_ENABLED=false
        return
    fi

    ADMINER_ENABLED=true
    echo "Adminer will be accessible through the management interface."

    # Note: No separate port needed - accessed via management UI
}
```

Task 8.6: Portainer Detection

```
configure_portainer_enhanced() {
    print_section "Portainer Configuration"

    # Check if Portainer or agent is already running
    if docker ps --format '{{.Names}}' | grep -qE
'(portainer|portainer_agent)'; then
        print_info "Portainer detected already running"
        PORTAINER_SKIP=true
        return
    fi

    echo "Portainer provides container management UI."
    echo ""
    echo "Options:"
    echo "  1) Install Portainer Agent (you have Portainer
elsewhere)"
    echo "  2) Install full Portainer CE (first Portainer
installation)"
    echo "  3) Skip Portainer"

    # Handle selection
}
```

Task 8.7: Backup Prompts

```
configure_backups() {
    print_section "Backup Configuration"

    # Already covered if NFS was configured
    if [ "$NFS_ENABLED" != true ]; then
        print_warning "NFS not configured. Backups will use
local storage."
        # Prompt for local backup directory
    fi

    # Schedule configuration
    echo "Configure backup schedule:"
    echo "  1) 24 hourly, 7 daily, 4 weekly, 12 monthly
(recommended)"
    echo "  2) 7 daily, 4 weekly, 12 monthly"
    echo "  3) Custom (configure in management interface)"

    # Encryption passphrase for env backups
    prompt_with_default "Passphrase for encrypted backups" ""
"BACKUP_PASSPHRASE"
}
```

Task 8.8: Notification Prompts

(As defined in Infrastructure Agent)

Task 8.9: State Resume

```
check_resume_state() {
    if [ -f "$STATE_FILE" ]; then
        print_section "Previous Installation Detected"

        source "$STATE_FILE"
        echo "Found installation at step: $CURRENT_STEP"
        echo "Completed: $COMPLETED_STEPS"

        if confirm_prompt "Resume previous installation?"; then
            RESUME_MODE=true
            return 0
        else
            mv "$STATE_FILE" "${STATE_FILE}.backup.$(date +%s)"
            return 1
        fi
    fi
    return 1
}
```

Task 8.10: Port Checking

```
check_port_available() {
    local port=$1
    if ss -tulpn 2>/dev/null | grep -q ":$port " || \
       netstat -tulpn 2>/dev/null | grep -q ":$port "; then
        return 1
    fi
    return 0
}

get_management_port() {
    local default_port=8443

    while true; do
        prompt_with_default "Management interface port" "$default_port" "MGMT_PORT"

        if check_port_available "$MGMT_PORT"; then
            print_success "Port $MGMT_PORT is available"
            break
        else
            print_error "Port $MGMT_PORT is already in use"
            ((default_port++))
        fi
    done
}
```

Output Requirements

- Modify setup.sh in place
- Maintain existing style and helper functions
- Update all file generation functions for new features
- Update configuration summary to show new options.
- Test all code paths

~~~

Agent 9: Documentation Agent
Purpose: Write comprehensive README documentation

```markdown
# Documentation Agent Prompt

## Context
Update README.md for n8n_nginx v3.0 with all new features.

## Documentation Structure

### Update Version & What's New
```markdown
**Version:** 3.0.0
**Release Date:** [DATE]

## What's New in v3.0.0

### Management Interface
- Web-based dashboard for managing your n8n stack
- Real-time container monitoring and logs
~~~

```
- One-click backup and restore
- Flow extraction from backups

### Backup System
- Automated PostgreSQL, n8n, and SSL certificate backups
- Flexible scheduling (hourly, daily, weekly, monthly)
- Configurable retention policies
- NFS storage support for enterprise deployments

### Cloudflare Tunnel Integration
- Secure external webhook access without port forwarding
- Optional UI exposure with authentication
- Step-by-step setup guide

### Tailscale VPN
- Secure remote management access
- Zero-config VPN setup

### Enhanced Portainer Support
- Full Portainer CE or Agent installation
- Automatic detection of existing installations
```

New Sections to Add

Cloudflare Tunnel Setup (Task 9.1)

- Prerequisites checklist
- Creating a tunnel in Cloudflare dashboard
- Configuring tunnel ingress rules
- Token integration during setup
- Troubleshooting common issues

NFS Configuration (Task 9.2)

- When to use NFS vs local storage

- NFS server requirements
- Export configuration example
- Mount options explanation
- Directory structure guide

Management Interface (Task 9.3)

- Accessing the interface
- Dashboard overview
- Backup management
- Container controls
- System monitoring
- Settings configuration

Backup & Restore (Task 9.4)

- Backup types explained
- Schedule configuration
- Retention policies
- Manual backup triggering
- Restoring from backup
- Disaster recovery scenarios

Flow Extraction (Task 9.5)

- When to use flow extraction
- Step-by-step extraction process
- Importing extracted flows
- Cleanup procedures

Tailscale Setup (Task 9.6)

- Creating Tailscale account
- Generating auth keys
- Accessing services via Tailscale
- ACL recommendations

In-App Help (Task 9.7)

Create help content for each management UI page:

- Contextual help tooltips
- Page-specific guidance
- Troubleshooting tips
- Link to full documentation

Output Requirements

- Update README.md
- Create additional docs if needed (e.g., docs/backup-guide.md)
- Include screenshots (create placeholder sections)
- Maintain existing README style

```
---

## Storyboard Specifications
```

Update Todos

```
Create comprehensive v3.0 implementation plan
Define project phases and milestones
Create agent task breakdown with prompts
Designing storyboard specifications
```

Storyboard Design Specifications

Each storyboard will be a self-contained HTML file with all CSS embedded. Here are the detailed specifications:

Pages Included in Each Storyboard

Page	Key Elements	Mock Data Needed
Login	Username/password, logo, remember me	None
Dashboard	5 container cards, backup status, system stats	Container names, status, mock metrics
Backups	Table of 15 backups, filters, actions	Backup files with dates/sizes
Schedule	Visual scheduler, retention settings	Current schedule config
Containers	5 containers with details, logs preview	Docker ps output
System	CPU/RAM/Disk charts, NFS status, power controls	System metrics
Flows	Backup selector, flow list, actions	5 mock workflows
Settings	Form fields for all settings	Current config
Logs	Live log area, container selector	Mock log lines

Color Palettes

Design A (Modern Minimal #1):

- Background: `#ffffff`
- Surface: `#f8fafc`
- Primary: `#2563eb` (blue-600)
- Secondary: `#64748b` (slate-500)
- Success: `#22c55e` (green-500)
- Warning: `#f59e0b` (amber-500)
- Error: `#ef4444` (red-500)
- Text: `#1e293b` (slate-800)

Design B (Modern Minimal #2):

- Background: `#f8fafc`
- Surface: `#ffffff`
- Primary: `#0d9488` (teal-600)
- Secondary: `#6b7280` (gray-500)
- Success: `#10b981` (emerald-500)
- Warning: `#f97316` (orange-500)
- Error: `#dc2626` (red-600)
- Text: `#111827` (gray-900)

Design C (Dashboard Heavy #1):

- Sidebar: `#1e293b` (slate-800)
- Background: `#f1f5f9` (slate-100)
- Surface: `#ffffff`
- Primary: `#f97316` (orange-500)
- Secondary: `#3b82f6` (blue-500)
- Success: `#22c55e`
- Warning: `#eab308`
- Error: `#ef4444`
- Text: `#0f172a`

Design D (Dashboard Heavy #2 - Dark):

- Background: `#0f172a` (slate-900)
- Surface: `#1e293b` (slate-800)
- Glass: `rgba(255,255,255,0.1)`
- Primary: `#8b5cf6` (violet-500)
- Secondary: `#06b6d4` (cyan-500)

- Success: `#22c55e`
- Warning: `#f59e0b`
- Error: `#ef4444`
- Text: `#f8fafc`

Icon Requirements (for icon_requirements.md)

Category	Icons Needed	Style
Navigation	Dashboard, Database, Container, Settings, Logout, Menu	Colored, 32x32
Status	Running (green circle), Stopped (red circle), Warning (yellow triangle), Error (red X)	Colored, 24x24
Containers	Docker whale, PostgreSQL elephant, Nginx, n8n logo, Server	Colored, 48x48
Actions	Refresh, Download, Delete, Play, Stop, Restart, Edit, Save	Colored, 24x24
Backup	Database backup, Archive, Calendar, Clock, Folder	Colored, 32x32
System	CPU chip, RAM stick, Hard drive, Network, Power, Reboot	Colored, 32x32
Alerts	Info (blue i), Success (green check), Warning (yellow !), Error (red x)	Colored, 24x24
Misc	Help (?), External link, Copy, Eye, Eye-off, Lock	Colored, 24x24

Recommended Execution Order

Stage 1: Parallel Foundation Work

Run these agents simultaneously:

- **Agent 1 (Infrastructure)** - NFS, notifications, state
- **Agent 2 (Tunnel & VPN)** - Cloudflare, Tailscale
- **Agent 5 (Storyboard)** - UI designs

Stage 2: After Stage 1 Completes

- **User Review**: Select storyboard design
- **Agent 3 (Backup System)** - Depends on NFS being ready
- **Agent 4 (Management Container)** - Depends on storyboard selection

Stage 3: After Stage 2 Completes

- **Agent 6 (Web Application)** - Depends on design + container base
- **Agent 7 (Flow Extraction)** - Depends on backup system

Stage 4: Integration

- **Agent 8 (Setup.sh Integration)** - Requires all other agents complete

Stage 5: Final

- **Agent 9 (Documentation)** - After everything is integrated

---

Questions Answered - Dry Run Preview

You asked about "dry-run preview" for flow imports. What I meant was:

Before actually importing a flow into production, show a preview of:

- Flow name (and proposed renamed name if conflict)
- Number of nodes
- Node types used
- Any credentials referenced (without values)
- Confirmation: "This will create workflow 'MyFlow_restored_2025-12-04'. Proceed?"

Based on your answer, I'll skip this feature - we'll just auto-rename and import directly since users can delete unwanted flows afterward.

Next Steps

Update Todos

```
Create comprehensive v3.0 implementation plan
Define project phases and milestones
Create agent task breakdown with prompts
Design storyboard specifications
```

Summary

I've created a comprehensive implementation plan that includes:

1. **9 Phases** with 60+ individual tasks

2. **9 Specialized Agents** with detailed prompts
3. **4 Storyboard Designs** with complete specifications
4. **Execution Timeline** showing parallel work opportunities

To Begin Implementation

When you're ready to start, tell me which of these approaches you prefer:

Option A: Start with Storyboards "Spin up Agent 5 to create the 4 UI storyboards so I can pick a design first."

Option B: Start with Infrastructure "Spin up Agents 1 and 2 in parallel to build the NFS/notification system and tunnel integration while we work on storyboards."

Option C: Full Parallel Start "Spin up Agents 1, 2, and 5 simultaneously to maximize parallel work."

Final Clarifications Needed

Before we begin, please confirm:

1. **Management Interface Port**: Since n8n uses 443, the management interface will need a different port. I recommend **8443** so the URL becomes `https://loftaimgmt.loft.aero:8443`. Alternatively, if you have multiple IPs on the host, we could use 443 on a different IP. **Which do you prefer?**
2. **Dozzle Integration**: Should Dozzle run as:
    - **Embedded** in the management container (one less container, integrated auth)
    - **Separate container** with proxy through management UI (easier updates, official image)
3. **NFS Mount Options**: For your specific NFS server (loft-scale02.loft.aero), are there any specific mount options required? Standard options would be:

```
rw,sync,hard,intr,nfsvers=4.2
```