Hub-and-Spoke WireGuard VPN System

Complete Architecture and Implementation Guide

Version: 1.0 **Date:** December 2025 **Status:** Design Review - Awaiting Chief Architect Approval

Executive Summary

This document presents a comprehensive design for a hub-and-spoke WireGuard VPN management system with automated spoke provisioning, one-time-use installation tokens, and multi-platform support including Proxmox VE cluster integration.

Key Features

- **One-Time-Use Installation Tokens**: 256-bit cryptographically secure tokens with 24-hour expiration
- **Multi-Platform Support**: Linux, macOS, Windows, and Proxmox VE
- **Zero-Trust Key Generation**: Spoke private keys generated locally, never transmitted
- **Automated Provisioning**: Single-command installation with automatic WireGuard configuration
- **Proxmox Cluster Integration**: Auto-detection and management of multi-datacenter Proxmox clusters
- **Web-Based Dashboard**: React + TypeScript frontend for centralized management
- **RESTful API**: Express + TypeScript backend with SQLite database

Technology Stack

- **Frontend**: React 18, TypeScript, Vite, TailwindCSS
- **Backend**: Node.js, Express, TypeScript, better-sqlite3
- **VPN**: WireGuard (kernel-level implementation)
- **Database**: SQLite (POC), PostgreSQL (production migration path)
- **Deployment**: Ubuntu 22.04+, Nginx reverse proxy, Let’s Encrypt TLS

Table of Contents

1. System Architecture
2. Deployment Architecture
3. Spoke Provisioning Flow
4. Database Design
5. Security Model
6. Proxmox Multi-Cluster Architecture
7. API Specification
8. Installation Script Design
9. Frontend Components
10. Backend Services
11. Deployment Guide
12. Testing Strategy
13. Operations and Monitoring
14. Risk Assessment
15. Implementation Roadmap

1. System Architecture

High-Level Component Diagram

```
%% System Architecture Diagram
%% Mermaid format for rendering

graph TB
    subgraph "Client Layer"
        A1[Admin Browser]
        S1[Spoke: Linux]
        S2[Spoke: macOS]
        S3[Spoke: Windows]
        S4[Spoke: Proxmox Cluster]
    end

    subgraph "Hub Server"
        subgraph "Web Layer"
            RP[Reverse Proxy<br/>nginx/Caddy<br/>:443 HTTPS]
            SPA[React Dashboard<br/>Static Files]
        end

        subgraph "Application Layer"
            API[Express API<br/>Node.js<br/>:3000]
            TC[TokenService]
            WG[WireGuardService]
            IP[IPAddressPool]
            SG[ScriptGenerator]
        end

        subgraph "Data Layer"
            DB[(SQLite Database)]
            WGI[WireGuard Interface<br/>wg0<br/>:51820 UDP]
        end
    end

    subgraph "VPN Network"
```

```
    VPN[10.0.1.0/24<br/>Hub-and-Spoke Mesh]
end

A1 -->|HTTPS| RP
RP -->|Serve Static| SPA
RP -->|Proxy /api/*| API

S1 & S2 & S3 & S4 -->|curl install script| API
S1 & S2 & S3 & S4 -->|POST /spoke/register| API

API --> TC
API --> WG
API --> IP
API --> SG
API --> DB

WG -->|wg set, wg-quick| WGI

S1 & S2 & S3 & S4 -.->|WireGuard VPN| VPN
WGI -.->|WireGuard VPN| VPN

style A1 fill:#e1f5ff
style S1 fill:#fff4e1
style S2 fill:#fff4e1
style S3 fill:#fff4e1
style S4 fill:#e8f5e9
style RP fill:#f3e5f5
style API fill:#fff3e0
style DB fill:#e0f2f1
style WGI fill:#fce4ec
style VPN fill:#f1f8e9
```

Figure: System Architecture

The system consists of five major layers:

1.1 Presentation Layer (React Dashboard)

- **Technology**: React 18 + TypeScript + Vite
- **Purpose**: Web-based management interface
- **Key Components**:
  - HubInitializer: First-run setup wizard
  - SpokeManager: Spoke list and token generation
  - ProxmoxClusterView: Hierarchical cluster visualization
  - InstallationInstructions: Platform-specific installation commands

1.2 API Layer (Express Backend)

- **Technology**: Express.js + TypeScript
- **Port**: 3000 (internal), proxied via Nginx on 443
- **Endpoints**:
  - `/api/hub/*` - Hub management
  - `/api/installation/*` - Token and script serving
  - `/api/spoke/*` - Spoke registration and status
  - `/api/proxmox/*` - Proxmox cluster management

1.3 Service Layer

- **TokenService**: Secure token generation and validation
- **WireGuardService**: Hub configuration and peer management
- **ScriptGenerator**: Template rendering for installation scripts
- **IPAddressPool**: CIDR-based IP allocation

1.4 Data Layer

- **Database**: SQLite (better-sqlite3)
- **Location**: `/var/lib/wireguard-hub/database.sqlite`
- **Tables**: hub_config, installation_tokens, spoke_registrations, proxmox_clusters

1.5 System Layer

- **WireGuard Interface**: `wg0` kernel module
- **Port**: 51820/udp

- **Management**: systemd services (`wg-quick@wg0` , `wireguard-hub-api`)

1.6 Network Topology

Hub-and-Spoke Model:

```
Hub (10.0.1.1) ← → Spoke 1 (10.0.1.10)
                ← → Spoke 2 (10.0.1.11)
                ← → Spoke 3 (10.0.1.12)
                ← → ... (up to 253 spokes on /24 network)
```

Key Characteristics:

- All traffic routes through hub (star topology)
- No direct spoke-to-spoke communication by default
- Hub acts as router for 10.0.1.0/24 network
- Each spoke maintains single persistent connection to hub

---

2. Deployment Architecture

Production Deployment Diagram

```
%% Production Deployment Architecture

graph TB
    subgraph "Internet"
        DNS[DNS: hub.example.com<br/>A Record → 203.0.113.5]
        CLIENT[Client Browsers<br/>Admin Dashboard Access]
        SPOKES[Spoke Devices<br/>Installation Scripts]
    end

    subgraph "DMZ / Firewall"
        FW[Firewall Rules<br/>443/tcp HTTPS<br/>51820/udp WireGuard]
    end

    subgraph "Hub Server: Ubuntu 22.04"
        subgraph "Reverse Proxy Layer"
            NGINX[Nginx<br/>:443 HTTPS<br/>TLS Termination]
        end

        subgraph "Application Layer"
            STATIC[Static Files<br/>dist/<br/>React SPA]
            EXPRESS[Express API<br/>:3000<br/>Node.js Process]
        end

        subgraph "Service Layer"
            TOKEN[TokenService]
            WGSERVICE[WireGuardService]
            IPPOOL[IPAddressPool]
            SCRIPT[ScriptGenerator]
        end

        subgraph "Data Layer"
            SQLITE[(SQLite DB<br/>/var/lib/wireguard-hub/<br/>database.sqlite)]
        end
```

```
    subgraph "System Layer"
        WG0[WireGuard Interface<br/>wg0<br/>:51820 UDP<br/>Kernel Module]
        SYSTEMD[systemd Services<br/>wg-quick@wg0<br/>wireguard-hub-api]
    end
end

subgraph "Monitoring & Backup"
    PROM[Prometheus<br/>Metrics Collection]
    GRAFANA[Grafana<br/>Dashboards]
    BACKUP[Daily Backups<br/>DB + Hub Keys<br/>GPG Encrypted]
end

DNS --> FW
CLIENT --> FW
SPOKES --> FW

FW -->|443/tcp| NGINX
FW -->|51820/udp| WG0

NGINX -->|Serve| STATIC
NGINX -->|Proxy /api/*| EXPRESS

EXPRESS --> TOKEN
EXPRESS --> WGSERVICE
EXPRESS --> IPPOOL
EXPRESS --> SCRIPT

TOKEN --> SQLITE
WGSERVICE --> SQLITE
WGSERVICE -->|wg set, wg-quick| WG0

SYSTEMD -->|Manages| WG0
SYSTEMD -->|Manages| EXPRESS

EXPRESS -.->|Export Metrics| PROM
PROM --> GRAFANA
```

```
    SQLITE -.->|Cron Daily| BACKUP

    style FW fill:#f44336,color:#fff
    style NGINX fill:#009688,color:#fff
    style EXPRESS fill:#ff9800,color:#fff
    style SQLITE fill:#2196f3,color:#fff
    style WG0 fill:#9c27b0,color:#fff
    style BACKUP fill:#4caf50,color:#fff
```

Figure: Deployment Architecture

2.1 Infrastructure Requirements

Hub Server Specifications:

- **OS**: Ubuntu 22.04 LTS or newer
- **CPU**: 2+ cores (4+ recommended for high spoke count)
- **RAM**: 4GB minimum (8GB recommended)
- **Storage**: 20GB+ SSD
- **Network**: Public IPv4 address, 1Gbps+ link
- **Ports**: 443/tcp (HTTPS), 51820/udp (WireGuard)

DNS Configuration:

```
hub.example.com    A    203.0.113.5
```

2.2 Service Architecture

Nginx Reverse Proxy:

- TLS termination (Let's Encrypt)
- Serves static files from `dist/` (React SPA)
- Proxies `/api/*` to Express backend on port 3000
- HTTP/2 enabled for performance

Express API:

- Runs as systemd service (`wireguard-hub-api.service`)
- Listens on `127.0.0.1:3000`
- Auto-restart on failure
- Logs to journalctl

WireGuard Interface:

- Runs as systemd service (`wg-quick@wg0.service`)
- Listens on `0.0.0.0:51820/udp`
- Kernel-level packet processing
- Automatic peer handshake every 25 seconds (PersistentKeepalive)

2.3 Data Flow

Dashboard Access:

```
Browser → [443/tcp HTTPS] → Nginx → Static Files (React SPA)
Browser → [443/tcp HTTPS] → Nginx → [/api/*] → Express (3000)
```

Spoke Installation:

```
Spoke → [443/tcp HTTPS] → Nginx → Express → Script Generator
Spoke → [443/tcp HTTPS] → Nginx → Express → TokenService → SQLite
Spoke → [51820/udp] → WireGuard Interface (wg0)
```

---

3. Spoke Provisioning Flow

Complete Provisioning Sequence

```
%% Spoke Provisioning Sequence Diagram

sequenceDiagram
    participant Admin as Admin<br/>(Browser)
    participant Dashboard as React<br/>Dashboard
    participant API as Express<br/>API
    participant DB as SQLite<br/>Database
    participant WG as WireGuard<br/>Service
    participant Spoke as Spoke<br/>Device

    Admin->>Dashboard: 1. Click "Add Spoke"<br/>Name: "laptop-001"
    Dashboard->>API: 2. POST /api/installation/token<br/>{spokeName: "laptop-001"}

    API->>API: 3. Generate 256-bit token<br/>crypto.randomBytes(32)
    API->>DB: 4. Allocate next IP<br/>10.0.1.5/24
    API->>DB: 5. INSERT token record<br/>(unused, expires 24h)
    DB-->>API: Token created

    API-->>Dashboard: 6. Return token + install cmd
    Dashboard-->>Admin: 7. Show installation command<br/>curl https://hub/.../TOKEN | bash

    Admin->>Spoke: 8. SSH to spoke device<br/>Execute install command

    Spoke->>API: 9. GET /api/installation/script/TOKEN?platform=linux
    API->>DB: 10. Validate token<br/>(exists, unused, not expired)
    API->>DB: 11. Mark token as USED<br/>(atomic UPDATE)
    DB-->>API: Token marked used
    API-->>Spoke: 12. Return rendered script<br/>(with embedded config)

    Spoke->>Spoke: 13. Install WireGuard<br/>apt-get install wireguard
    Spoke->>Spoke: 14. Generate keys locally<br/>wg genkey | wg pubkey
    Spoke->>Spoke: 15. Private key NEVER leaves device
```

```
    Spoke->>API: 16. POST /api/spoke/register<br/>{token, publicKey, os}
    API->>DB: 17. Validate token again<br/>(prevent replay)
    API->>DB: 18. Check public key unique
    API->>DB: 19. INSERT spoke registration
    DB-->>API: Spoke registered

    API->>WG: 20. wg set wg0 peer <publicKey><br/>allowed-ips 10.0.1.5/32
    WG-->>API: Peer added
    API->>WG: 21. wg-quick save wg0
    WG-->>API: Config saved

    API-->>Spoke: 22. Registration successful

    Spoke->>Spoke: 23. Create /etc/wireguard/wg0.conf
    Spoke->>Spoke: 24. systemctl enable wg-quick@wg0
    Spoke->>Spoke: 25. systemctl start wg-quick@wg0

    Spoke->>WG: 26. Establish VPN tunnel
    WG->>Spoke: 27. Handshake complete

    Spoke-->>Admin: 28. ✅ Installation complete<br/>VPN IP: 10.0.1.5

    Note over Admin,Spoke: Total time: ~30-60 seconds
```

Figure: Spoke Provisioning Sequence

3.1 Phase 1: Token Generation (Dashboard)

User Action: Admin clicks “Add Spoke” and enters name “laptop-001”

Backend Processing:

1. Generate 256-bit secure random token: `crypto.randomBytes(32).toString('base64url')`
2. Allocate next available IP from pool (e.g., 10.0.1.5/24)
3. Create database record:

```
{
  "id": "uuid-123",
  "token": "abc123...xyz789",
  "spokeId": "uuid-456",
  "spokeName": "laptop-001",
  "allowedIPs": ["10.0.1.5/24"],
  "createdAt": "2025-12-22T10:00:00Z",
  "expiresAt": "2025-12-23T10:00:00Z",
  "used": false,
  "hubEndpoint": "203.0.113.5:51820",
  "hubPublicKey": "hub-public-key...",
  "networkCIDR": "10.0.1.0/24"
}
```

Frontend Display:

```
# Installation Command (Linux)
curl -sSL https://hub.example.com/api/installation/script/abc123...xyz789?platform=linux | s

# Installation Command (Windows PowerShell - Run as Administrator)
Invoke-WebRequest -Uri "https://hub.example.com/api/installation/script/abc123...xyz789?plat
```

3.2 Phase 2: Script Execution (Spoke Device)

Step 1: Script Download

- Spoke device fetches script via HTTPS
- Backend validates token (exists, not used, not expired)
- Backend atomically marks token as used: `UPDATE installation_tokens SET used = 1, used_at = NOW() WHERE token = ? AND used = 0`
- Backend returns rendered script with embedded configuration

Step 2: WireGuard Installation

- Linux: `apt-get install wireguard` or `yum install wireguard-tools`
- macOS: `brew install wireguard-tools`
- Windows: `choco install wireguard`

- Proxmox: `apt-get install wireguard` (Debian-based)

Step 3: Key Generation (Local)

```
PRIVATE_KEY=$(wg genkey)
PUBLIC_KEY=$(echo "$PRIVATE_KEY" | wg pubkey)
```

Critical: Private key NEVER leaves the spoke device

Step 4: Spoke Registration

```
curl -X POST https://hub.example.com/api/spoke/register \
  -H "Content-Type: application/json" \
  -d '{
    "token": "abc123...xyz789",
    "publicKey": "spoke-public-key...",
    "os": "linux"
  }'
```

3.3 Phase 3: Hub Configuration Update

Backend Processing:

1. Validate token (double-check not reused)
2. Validate public key format (44 characters, base64)
3. Check public key uniqueness (prevent impersonation)
4. Create spoke registration record in database
5. Add peer to WireGuard interface:

   ```
   wg set wg0 peer <spoke-public-key> allowed-ips 10.0.1.5/32 persistent-keepalive 25
   ```

6. Persist configuration: `wg-quick save wg0`

Success Response:

```
{
  "success": true,
  "spokeId": "uuid-456",
  "spokeName": "laptop-001",
  "allowedIPs": ["10.0.1.5/24"],
  "status": "active"
}
```

3.4 Phase 4: Spoke Configuration and Connection

Script Actions:

1. Create `/etc/wireguard/wg0.conf`:

```
[Interface]
Address = 10.0.1.5/24
PrivateKey = <locally-generated-private-key>
DNS = 1.1.1.1, 8.8.8.8

[Peer]
# Hub
PublicKey = <hub-public-key>
Endpoint = 203.0.113.5:51820
AllowedIPs = 10.0.1.0/24
PersistentKeepalive = 25
```

2. Set permissions: `chmod 600 /etc/wireguard/wg0.conf`
3. Enable service: `systemctl enable wg-quick@wg0`
4. Start VPN: `systemctl start wg-quick@wg0`

Verification:

```
$ wg show wg0
interface: wg0
  public key: spoke-public-key...
  private key: (hidden)
  listening port: 51820

peer: hub-public-key...
  endpoint: 203.0.113.5:51820
  allowed ips: 10.0.1.0/24
  latest handshake: 1 second ago
  transfer: 1.2 KiB received, 856 B sent
  persistent keepalive: every 25 seconds
```

Total Provisioning Time: 30-60 seconds

---

4. Database Design

Entity Relationship Diagram

```
%% Database Entity Relationship Diagram

erDiagram
    HUB_CONFIG ||--o{ INSTALLATION_TOKENS : "provides config for"
    INSTALLATION_TOKENS ||--o| SPOKE_REGISTRATIONS : "consumed by"
    PROXMOX_CLUSTERS ||--o{ SPOKE_REGISTRATIONS : "contains"

    HUB_CONFIG {
        int id PK "Always 1 (singleton)"
        string interface_address "10.0.1.1/24"
        int listen_port "51820"
        string private_key "Base64 encoded"
        string public_key "Base64 encoded"
        string network_cidr "10.0.1.0/24"
        json dns "Optional DNS servers"
        string endpoint "Public IP:port"
        datetime created_at
        datetime updated_at
    }

    INSTALLATION_TOKENS {
        string id PK "UUID"
        string token UK "32-byte base64url"
        string spoke_id UK "Pre-assigned UUID"
        string spoke_name "User-friendly name"
        json allowed_ips "Allocated IP list"
        datetime created_at
        datetime expires_at "24 hours from created"
        boolean used "0 or 1"
        datetime used_at "When consumed"
        string hub_endpoint "From hub_config"
        string hub_public_key "From hub_config"
        string network_cidr "From hub_config"
```

```
        json dns "Optional"
        int persistent_keepalive "Default 25"
    }

    SPOKE_REGISTRATIONS {
        string id PK "Same as spoke_id from token"
        string token_id FK "Links to token"
        string name "Spoke name"
        string public_key UK "Spoke's public key"
        json allowed_ips "IP assignments"
        datetime registered_at
        datetime last_handshake "From wg show"
        string status "pending/active/inactive"
        string os "linux/macos/windows/proxmox"
        boolean is_proxmox "True for Proxmox nodes"
        string proxmox_cluster_id FK "Links to cluster"
        string proxmox_node_name "pve1, pve2, etc"
        string proxmox_version "Proxmox VE version"
        json metadata "Extensible field"
    }

    PROXMOX_CLUSTERS {
        string id PK "UUID"
        string cluster_name UK "From pvecm status"
        string datacenter "User metadata"
        string description "User metadata"
        datetime created_at
        datetime updated_at
    }
```

Figure: Database ERD

4.1 Table: hub_config (Singleton)

Purpose: Store hub WireGuard configuration

```
CREATE TABLE hub_config (
  id INTEGER PRIMARY KEY CHECK (id = 1),  -- Enforce single row
  interface_address TEXT NOT NULL,        -- 10.0.1.1/24
  listen_port INTEGER NOT NULL,           -- 51820
  private_key TEXT NOT NULL,              -- Base64 encoded, AES-256 encrypted at rest
  public_key TEXT NOT NULL,               -- Base64 encoded
  network_cidr TEXT NOT NULL,             -- 10.0.1.0/24
  dns TEXT,                               -- JSON array: ["1.1.1.1", "8.8.8.8"]
  endpoint TEXT NOT NULL,                 -- 203.0.113.5:51820
  created_at TEXT NOT NULL,
  updated_at TEXT NOT NULL
);
```

Sample Data:

```
{
  "id": 1,
  "interfaceAddress": "10.0.1.1/24",
  "listenPort": 51820,
  "privateKey": "oK5RN+kVJVy...",
  "publicKey": "jNQCb8RA9pf...",
  "networkCIDR": "10.0.1.0/24",
  "dns": ["1.1.1.1", "8.8.8.8"],
  "endpoint": "203.0.113.5:51820",
  "createdAt": "2025-12-22T09:00:00Z",
  "updatedAt": "2025-12-22T09:00:00Z"
}
```

4.2 Table: installation_tokens

Purpose: Store one-time-use installation tokens

```
CREATE TABLE installation_tokens (
  id TEXT PRIMARY KEY,                    -- UUID
  token TEXT UNIQUE NOT NULL,             -- 32-byte base64url (256-bit)
  spoke_id TEXT UNIQUE NOT NULL,          -- Pre-assigned UUID
  spoke_name TEXT NOT NULL,               -- User-friendly name
  allowed_ips TEXT NOT NULL,              -- JSON array: ["10.0.1.5/24"]
  created_at TEXT NOT NULL,
  expires_at TEXT NOT NULL,               -- created_at + 24 hours
  used INTEGER DEFAULT 0,                 -- Boolean: 0 or 1
  used_at TEXT,                           -- Timestamp when consumed
  hub_endpoint TEXT NOT NULL,             -- From hub_config
  hub_public_key TEXT NOT NULL,           -- From hub_config
  network_cidr TEXT NOT NULL,             -- From hub_config
  dns TEXT,                               -- JSON array (optional)
  persistent_keepalive INTEGER DEFAULT 25
);

CREATE INDEX idx_token ON installation_tokens(token);
CREATE INDEX idx_expires_at ON installation_tokens(expires_at);
```

Sample Data:

```
{
  "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "token": "Kd8jD_s9fN2mP1qR6tV8wX0yZ5aB3cE7gH4iJ",
  "spokeId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "spokeName": "laptop-001",
  "allowedIPs": ["10.0.1.5/24"],
  "createdAt": "2025-12-22T10:00:00Z",
  "expiresAt": "2025-12-23T10:00:00Z",
  "used": 0,
  "usedAt": null,
  "hubEndpoint": "203.0.113.5:51820",
  "hubPublicKey": "jNQCb8RA9pf...",
  "networkCIDR": "10.0.1.0/24",
  "dns": ["1.1.1.1", "8.8.8.8"],
  "persistentKeepalive": 25
}
```


4.3 Table: spoke_registrations

Purpose: Store registered spokes

```
CREATE TABLE spoke_registrations (
  id TEXT PRIMARY KEY,                    -- Same as spoke_id from token
  token_id TEXT NOT NULL,                 -- Foreign key to installation_tokens
  name TEXT NOT NULL,
  public_key TEXT UNIQUE NOT NULL,        -- 44-char base64
  allowed_ips TEXT NOT NULL,              -- JSON array
  registered_at TEXT NOT NULL,
  last_handshake TEXT,                    -- From 'wg show' output
  status TEXT NOT NULL,                   -- pending/active/inactive
  os TEXT NOT NULL,                       -- linux/macos/windows/proxmox

  -- Proxmox-specific fields
  is_proxmox INTEGER DEFAULT 0,           -- Boolean
  proxmox_cluster_id TEXT,                -- Foreign key (NULL for standalone)
  proxmox_node_name TEXT,                 -- Node name within cluster
  proxmox_version TEXT,                   -- Proxmox VE version

  metadata TEXT,                          -- JSON (extensible)

  FOREIGN KEY (token_id) REFERENCES installation_tokens(id),
  FOREIGN KEY (proxmox_cluster_id) REFERENCES proxmox_clusters(id)
);

CREATE INDEX idx_public_key ON spoke_registrations(public_key);
CREATE INDEX idx_status ON spoke_registrations(status);
CREATE INDEX idx_proxmox_cluster ON spoke_registrations(proxmox_cluster_id);
```

Sample Data (Linux):

```
{
  "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "tokenId": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "name": "laptop-001",
  "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
  "allowedIPs": ["10.0.1.5/24"],
  "registeredAt": "2025-12-22T10:05:00Z",
  "lastHandshake": "2025-12-22T10:10:00Z",
  "status": "active",
  "os": "linux",
  "isProxmox": false,
  "metadata": {
    "wireguardVersion": "1.0.20210914",
    "kernelVersion": "5.15.0-91-generic"
  }
}
```


Sample Data (Proxmox):

```
{
  "id": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
  "tokenId": "e47ac10b-58cc-4372-a567-0e02b2c3d480",
  "name": "pve1.dc1",
  "publicKey": "yU1n0bW4cS6fE9gH2iK5lM7oQ0rT3vY8zA1dF5jP=",
  "allowedIPs": ["10.0.1.10/24"],
  "registeredAt": "2025-12-22T11:00:00Z",
  "lastHandshake": "2025-12-22T11:05:00Z",
  "status": "active",
  "os": "proxmox",
  "isProxmox": true,
  "proxmoxClusterId": "c3d4e5f6-a7b8-9012-cdef-123456789012",
  "proxmoxNodeName": "pve1",
  "proxmoxVersion": "8.1.3",
  "metadata": {
    "clusterName": "datacenter1-prod",
    "clusterNodes": "pve1,pve2,pve3",
    "isClustered": true
  }
}
```

4.4 Table: proxmox_clusters

Purpose: Organize Proxmox nodes by cluster

```
CREATE TABLE proxmox_clusters (
  id TEXT PRIMARY KEY,                    -- UUID
  cluster_name TEXT UNIQUE NOT NULL,      -- From 'pvecm status'
  datacenter TEXT,                        -- User-defined metadata
  description TEXT,                       -- User-defined metadata
  created_at TEXT NOT NULL,
  updated_at TEXT NOT NULL
);

CREATE INDEX idx_cluster_name ON proxmox_clusters(cluster_name);
```

Sample Data:

```
{
  "id": "c3d4e5f6-a7b8-9012-cdef-123456789012",
  "clusterName": "datacenter1-prod",
  "datacenter": "US-East-1",
  "description": "Production cluster for primary workloads",
  "createdAt": "2025-12-22T11:00:00Z",
  "updatedAt": "2025-12-22T11:00:00Z"
}
```


4.5 Database Relationships

1:1 Relationship:

- `installation_tokens.spoke_id` → `spoke_registrations.id` (one token creates one spoke)

1:N Relationships:

- `proxmox_clusters.id` → `spoke_registrations.proxmox_cluster_id` (one cluster has many nodes)

Cascade Behavior:

- Deleting a spoke does NOT delete the token (audit trail preserved)
- Deleting a cluster requires all nodes to be removed first (or nullifies foreign key)

---

5. Security Model

Security Architecture Diagram

```
%% Security Model & Threat Mitigation

graph TB
    subgraph "Threat Actors"
        T1[External Attacker<br/>Internet-based]
        T2[Malicious Insider<br/>Stolen Token]
        T3[MITM Attacker<br/>Network Intercept]
    end

    subgraph "Attack Vectors"
        A1[Token Theft]
        A2[Token Brute Force]
        A3[Token Replay]
        A4[Hub Key Compromise]
        A5[Spoke Impersonation]
        A6[Script Tampering]
    end

    subgraph "Security Controls"
        C1[HTTPS/TLS 1.3<br/>Transport Encryption]
        C2[256-bit Token Entropy<br/>2^256 combinations]
        C3[24-hour Expiration<br/>Limited Time Window]
        C4[Atomic DB Update<br/>One-time Use Enforcement]
        C5[Hub Key Encryption<br/>AES-256 at Rest]
        C6[Public Key Uniqueness<br/>DB Constraint]
        C7[Code Signing<br/>Script Integrity]
        C8[Rate Limiting<br/>10 tokens/hour/IP]
        C9[CORS Policy<br/>Origin Restriction]
        C10[Zero-Trust Keys<br/>Generated on Spoke]
    end

    subgraph "Assets Protected"
        AS1[Hub Private Key<br/>CRITICAL]
```

```
    AS2[Installation Tokens<br/>LIMITED SCOPE]
    AS3[Spoke Private Keys<br/>LOCAL ONLY]
    AS4[Network Topology<br/>DATABASE]
end

T1 --> A1
T1 --> A2
T1 --> A6
T2 --> A1
T2 --> A3
T3 --> A1
T3 --> A6

A1 -->|Mitigated by| C1
A1 -->|Mitigated by| C3
A2 -->|Mitigated by| C2
A2 -->|Mitigated by| C8
A3 -->|Mitigated by| C4
A4 -->|Mitigated by| C5
A5 -->|Mitigated by| C6
A6 -->|Mitigated by| C7

C1 --> AS2
C2 --> AS2
C3 --> AS2
C4 --> AS2
C5 --> AS1
C6 --> AS3
C7 --> AS2
C8 --> AS2
C9 --> AS4
C10 --> AS3

style T1 fill:#f44336,color:#fff
style T2 fill:#ff9800,color:#fff
style T3 fill:#ff5722,color:#fff
style C1 fill:#4caf50,color:#fff
style C2 fill:#4caf50,color:#fff
```

```
    style C3 fill:#4caf50,color:#fff
    style C4 fill:#4caf50,color:#fff
    style C5 fill:#2196f3,color:#fff
    style C6 fill:#4caf50,color:#fff
    style C10 fill:#2196f3,color:#fff
    style AS1 fill:#d32f2f,color:#fff
    style AS2 fill:#ffa726,color:#fff
    style AS3 fill:#66bb6a,color:#fff
```

Figure: Security Model

5.1 Threat Model

Threat Actors:

1. **External Attacker**: Internet-based adversary attempting unauthorized access
2. **Malicious Insider**: User with stolen installation token
3. **MITM Attacker**: Network-level adversary intercepting traffic

Attack Vectors:

1. **Token Theft**: Stolen installation token used for unauthorized spoke registration
2. **Token Brute Force**: Attempting to guess valid tokens
3. **Token Replay**: Reusing previously used token
4. **Hub Key Compromise**: Theft of hub private key
5. **Spoke Impersonation**: Registering with duplicate public key
6. **Script Tampering**: Modified installation script with malicious payload

5.2 Security Controls

C1: HTTPS/TLS 1.3 Transport Encryption

- **Mitigation**: Token Theft, Script Tampering
- **Implementation**: Let's Encrypt certificates, Nginx TLS termination
- **Cipher Suites**: TLS 1.3 only, forward secrecy enabled
- **HSTS**: `Strict-Transport-Security: max-age=31536000; includeSubDomains`

C2: 256-bit Token Entropy

- **Mitigation**: Token Brute Force
- **Implementation**: `crypto.randomBytes(32)` → 2^256 combinations
- **Guessing Difficulty**: ~10^77 attempts needed (impossible with current technology)

C3: 24-hour Token Expiration

- **Mitigation**: Token Theft (limited time window)
- **Implementation**: Database check on `expires_at` field
- **Cleanup**: Cron job removes expired tokens daily

C4: Atomic Database Update (One-Time Use)

- **Mitigation**: Token Replay
- **Implementation**:

```
UPDATE installation_tokens
SET used = 1, used_at = NOW()
WHERE token = ? AND used = 0
```

- **Race Condition Protection**: If `result.changes = 0`, token already used

C5: Hub Key Encryption at Rest

- **Mitigation**: Hub Key Compromise
- **Implementation**: AES-256 encryption of private key in database
- **Key Management**: Encryption key stored in environment variable (not in database)
- **Backup**: GPG-encrypted backups of hub keys

C6: Public Key Uniqueness Constraint

- **Mitigation**: Spoke Impersonation
- **Implementation**: `UNIQUE` constraint on `spoke_registrations.public_key`
- **Database Enforcement**: Prevents duplicate public keys

C7: Code Signing (Future Enhancement)

- **Mitigation**: Script Tampering

- **Implementation**: Sign installation scripts with GPG key
- **Verification**: Scripts validate signature before execution

C8: Rate Limiting

- **Mitigation**: Token Brute Force, DoS
- **Implementation**: Max 10 token generations per hour per IP
- **Library**: express-rate-limit middleware

C9: CORS Policy

- **Mitigation**: Unauthorized API access from malicious websites
- **Implementation**: Restrict origins to `https://hub.example.com`

C10: Zero-Trust Key Generation

- **Mitigation**: Hub Key Compromise impact
- **Implementation**: Spoke private keys generated locally, never transmitted
- **Benefit**: Hub compromise does NOT expose spoke private keys

5.3 Assets and Protection Levels

Asset	Criticality	Protection
Hub Private Key	**CRITICAL**	AES-256 encryption, GPG backups, restricted file permissions (600)
Installation Tokens	**LIMITED SCOPE**	HTTPS transport, 24h expiration, one-time use
Spoke Private Keys	**LOCAL ONLY**	Never leave spoke device, stored in /etc/wireguard/ (600 permissions)
Network Topology	**DATABASE**	Database-level access control, encrypted backups

5.4 Security Assumptions

Trusted Components:

- Hub server is physically secure
- Hub server OS is hardened and patched

- Database file permissions are properly configured
- Admin users accessing dashboard are authenticated (future: add auth)

Attack Surface Reduction:

- No SSH access required from spokes to hub
- No database ports exposed to internet
- Only ports 443/tcp and 51820/udp open
- WireGuard protocol cryptographically secure (ChaCha20, Poly1305, Curve25519)

5.5 Audit Trail

Token Lifecycle Logging:

```
Event: Token Created
  - Token ID: uuid-123
  - Spoke Name: laptop-001
  - IP Allocated: 10.0.1.5
  - Timestamp: 2025-12-22T10:00:00Z

Event: Token Used
  - Token ID: uuid-123
  - Source IP: 192.0.2.50
  - User Agent: curl/7.68.0
  - Timestamp: 2025-12-22T10:05:00Z

Event: Spoke Registered
  - Spoke ID: uuid-456
  - Public Key: xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=
  - OS: linux
  - Timestamp: 2025-12-22T10:05:30Z
```

Database Audit:

- All token and spoke records are immutable (no UPDATE/DELETE, only INSERT)
- Token `used` flag and `used_at` timestamp provide audit trail
- Spoke registration records preserve full token lifecycle

---

6. Proxmox Multi-Cluster Architecture

Proxmox Cluster Topology Diagram

```
%% Proxmox Multi-Cluster Architecture

graph TB
    subgraph "Hub (Central)"
        HUB[WireGuard Hub<br/>10.0.1.1<br/>203.0.113.5:51820]
        DB[(Database<br/>Cluster Registry)]
    end

    subgraph "Datacenter 1: Production"
        subgraph "Cluster: dc1-prod"
            PVE1[pve1.dc1<br/>10.0.1.10<br/>Proxmox Node]
            PVE2[pve2.dc1<br/>10.0.1.11<br/>Proxmox Node]
            PVE3[pve3.dc1<br/>10.0.1.12<br/>Proxmox Node]
            CEPH1[Ceph Storage<br/>Shared across cluster]
        end
        PVE1 & PVE2 & PVE3 -.->|Cluster Traffic| CEPH1
        PVE1 <-->|Corosync| PVE2
        PVE2 <-->|Corosync| PVE3
        PVE3 <-->|Corosync| PVE1
    end

    subgraph "Datacenter 2: Backup"
        subgraph "Cluster: dc2-backup"
            PVE4[pve1.dc2<br/>10.0.1.20<br/>Proxmox Node]
            PVE5[pve2.dc2<br/>10.0.1.21<br/>Proxmox Node]
            NFS[NFS Storage]
        end
        PVE4 & PVE5 -.->|Storage| NFS
        PVE4 <-->|Corosync| PVE5
    end

    subgraph "Edge Site"
        subgraph "Cluster: edge-site1"
```

```
        PVE6[pve1.edge<br/>10.0.1.30<br/>Proxmox Node]
        PVE7[pve2.edge<br/>10.0.1.31<br/>Proxmox Node]
    end
    PVE6 <-->|Corosync| PVE7
end

subgraph "Standalone Nodes"
    PVE8[pve-standalone<br/>10.0.1.50<br/>No Cluster]
end

HUB -.->|WireGuard VPN| PVE1
HUB -.->|WireGuard VPN| PVE2
HUB -.->|WireGuard VPN| PVE3
HUB -.->|WireGuard VPN| PVE4
HUB -.->|WireGuard VPN| PVE5
HUB -.->|WireGuard VPN| PVE6
HUB -.->|WireGuard VPN| PVE7
HUB -.->|WireGuard VPN| PVE8

HUB --> DB

Note1[Each node detects cluster<br/>membership via pvecm status]
Note2[All nodes register individually<br/>with unique IP & keys]
Note3[Dashboard groups nodes<br/>by cluster for visibility]

style HUB fill:#f44336,color:#fff
style PVE1 fill:#4caf50,color:#fff
style PVE2 fill:#4caf50,color:#fff
style PVE3 fill:#4caf50,color:#fff
style PVE4 fill:#2196f3,color:#fff
style PVE5 fill:#2196f3,color:#fff
style PVE6 fill:#ff9800,color:#fff
style PVE7 fill:#ff9800,color:#fff
style PVE8 fill:#9e9e9e,color:#fff
style DB fill:#e0e0e0
```

Figure: Proxmox Cluster Architecture

6.1 Design Philosophy

Key Principle: Each Proxmox node is an individual spoke

Rationale:

- Independent connectivity (cluster remains functional if one node's VPN fails)
- Unique IP addressing per node (enables node-specific routing)
- Simplified troubleshooting (identify issues per node)
- Flexible cluster membership (nodes can join/leave without affecting others)

6.2 Deployment Scenarios

Scenario 1: Three-Node Production Cluster

Setup:

```
Cluster: datacenter1-prod
├─ pve1.dc1 (10.0.1.10) - Node 1
├─ pve2.dc1 (10.0.1.11) - Node 2
└─ pve3.dc1 (10.0.1.12) - Node 3

Corosync: Internal network (192.168.1.0/24)
Storage: Ceph shared across all nodes
VPN: Each node connects to hub individually
```

Installation Process:

1. Generate token for "pve1.dc1" → Run installation script on pve1
2. Generate token for "pve2.dc1" → Run installation script on pve2
3. Generate token for "pve3.dc1" → Run installation script on pve3
4. Dashboard automatically groups nodes by cluster name "datacenter1-prod"

Network Configuration:

- Corosync traffic: Internal network (NOT through VPN)
- VM migration: Can use VPN (requires testing for performance)
- Ceph traffic: Internal network (NOT through VPN)
- Management access: Through VPN (ssh, web UI)

Scenario 2: Multi-Datacenter Deployment

Setup:

```
Datacenter 1 (US-East):
  Cluster: dc1-prod
  ├─ pve1.dc1 (10.0.1.10)
  ├─ pve2.dc1 (10.0.1.11)
  └─ pve3.dc1 (10.0.1.12)

Datacenter 2 (US-West):
  Cluster: dc2-backup
  ├─ pve1.dc2 (10.0.1.20)
  └─ pve2.dc2 (10.0.1.21)

Edge Site:
  Cluster: edge-site1
  ├─ pve1.edge (10.0.1.30)
  └─ pve2.edge (10.0.1.31)

Standalone:
  pve-standalone (10.0.1.50)
```

Use Cases:

- Centralized management of all Proxmox nodes via VPN
- Disaster recovery (backup VMs to remote datacenter)
- Edge computing (manage remote sites from central hub)
- Hybrid deployments (mix clustered and standalone nodes)

6.3 Proxmox Installation Script

Special Detections:

1. **Environment Check**: `pveversion` command exists
2. **Cluster Detection**: Parse `pvecm status` output
3. **Node Name**: `hostname` command
4. **Cluster Members**: `pvecm nodes` output

Script Logic:

```
# Detect Proxmox environment
if ! command -v pveversion &> /dev/null; then
  echo "Error: Not a Proxmox VE system"
  exit 1
fi

PVE_VERSION=$(pveversion | head -n1 | awk '{print $2}')

# Detect cluster membership
CLUSTER_STATUS=$(pvecm status 2>/dev/null || echo "not_clustered")

if echo "$CLUSTER_STATUS" | grep -q "Cluster information"; then
  IS_CLUSTERED=true
  CLUSTER_NAME=$(echo "$CLUSTER_STATUS" | grep "Name:" | awk '{print $2}')
  NODE_NAME=$(hostname)
  CLUSTER_NODES=$(pvecm nodes | grep -v "^Node" | awk '{print $3}' | tr '\n' ',' | sed 's/,$
else
  IS_CLUSTERED=false
  NODE_NAME=$(hostname)
fi

# Register with hub, including Proxmox metadata
curl -X POST "$CALLBACK_URL" \
  -H "Content-Type: application/json" \
  -d "{
    \"token\": \"$TOKEN\",
    \"publicKey\": \"$PUBLIC_KEY\",
    \"os\": \"proxmox\",
    \"isProxmox\": true,
    \"proxmoxNodeName\": \"$NODE_NAME\",
    \"proxmoxClusterName\": \"$CLUSTER_NAME\",
    \"proxmoxVersion\": \"$PVE_VERSION\",
    \"metadata\": {
      \"isClustered\": $IS_CLUSTERED,
      \"clusterNodes\": \"$CLUSTER_NODES\"
    }
  }"
```

6.4 Backend Handling

SpokeController.registerSpoke():

```
if (isProxmox && proxmoxClusterName) {
  // Check if cluster exists
  let cluster = db.prepare('SELECT * FROM proxmox_clusters WHERE cluster_name = ?')
    .get(proxmoxClusterName)

  // Create cluster if not exists
  if (!cluster) {
    const clusterId = uuidv4()
    db.prepare(`
      INSERT INTO proxmox_clusters (id, cluster_name, datacenter, created_at, updated_at)
      VALUES (?, ?, ?, ?, ?)
    `).run(clusterId, proxmoxClusterName, null, new Date().toISOString(), new Date().toISOS

    cluster = { id: clusterId, cluster_name: proxmoxClusterName }
  }

  // Associate spoke with cluster
  registration.proxmoxClusterId = cluster.id
  registration.proxmoxNodeName = proxmoxNodeName
}
```

6.5 Dashboard Visualization

Hierarchical View:

```
🏢 Cluster: datacenter1-prod (3 nodes) [All Active ✓]
  ├─ 🖥️ pve1.dc1 | 10.0.1.10 | ✅ Active (2 min ago)
  ├─ 🖥️ pve2.dc1 | 10.0.1.11 | ✅ Active (1 min ago)
  └─ 🖥️ pve3.dc1 | 10.0.1.12 | ✅ Active (3 min ago)

🏢 Cluster: datacenter2-backup (2 nodes) [Degraded ⚠️]
  ├─ 🖥️ pve1.dc2 | 10.0.1.20 | ✅ Active (1 min ago)
  └─ 🖥️ pve2.dc2 | 10.0.1.21 | ❌ Inactive (15 min ago)

💻 Standalone Proxmox Hosts
  └─ 🖥️ pve-standalone | 10.0.1.50 | ✅ Active (1 min ago)
```

Cluster Status Indicators:

- **All Active**: All nodes have handshake within last 3 minutes
- **Degraded**: Some nodes inactive (handshake > 5 minutes)
- **Critical**: Majority of nodes offline

---

7. API Specification

7.1 Hub Management Endpoints

POST /api/hub/initialize

Purpose: First-run hub setup (generates keys, creates WireGuard interface)

Request Body:

```
{
  "networkCIDR": "10.0.1.0/24",
  "listenPort": 51820,
  "endpoint": "203.0.113.5:51820",
  "dns": ["1.1.1.1", "8.8.8.8"]
}
```

Response (201 Created):

```
{
  "success": true,
  "hubConfig": {
    "id": 1,
    "interfaceAddress": "10.0.1.1/24",
    "listenPort": 51820,
    "publicKey": "jNQCb8RA9pf...",
    "networkCIDR": "10.0.1.0/24",
    "dns": ["1.1.1.1", "8.8.8.8"],
    "endpoint": "203.0.113.5:51820",
    "createdAt": "2025-12-22T09:00:00Z"
  }
}
```

Errors:

- 409 Conflict: Hub already initialized

GET /api/hub/config

Purpose: Retrieve current hub configuration

Response (200 OK):

```
{
  "id": 1,
  "interfaceAddress": "10.0.1.1/24",
  "listenPort": 51820,
  "publicKey": "jNQCb8RA9pf...",
  "networkCIDR": "10.0.1.0/24",
  "dns": ["1.1.1.1", "8.8.8.8"],
  "endpoint": "203.0.113.5:51820",
  "createdAt": "2025-12-22T09:00:00Z",
  "updatedAt": "2025-12-22T09:00:00Z"
}
```

Note: Private key is NEVER returned in API responses

PUT /api/hub/config

Purpose: Update hub settings (endpoint, DNS, etc.)

Request Body:

```
{
  "endpoint": "new-hub.example.com:51820",
  "dns": ["9.9.9.9"]
}
```

Response (200 OK):

```
{
  "success": true,
  "hubConfig": { /* updated config */ }
}
```

GET /api/hub/status

Purpose: Get WireGuard interface status

Response (200 OK):

```
{
  "interface": "wg0",
  "publicKey": "jNQCb8RA9pf...",
  "listenPort": 51820,
  "peerCount": 15,
  "peers": [
    {
      "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
      "endpoint": "192.0.2.50:51820",
      "allowedIPs": ["10.0.1.5/32"],
      "latestHandshake": "2025-12-22T10:10:00Z",
      "transferRx": 1234567,
      "transferTx": 987654,
      "persistentKeepalive": 25
    }
  ]
}
```

7.2 Installation Token Endpoints

POST /api/installation/token

Purpose: Generate new installation token

Request Body:

```
{
  "spokeName": "laptop-001",
  "customIP": "10.0.1.100/24"  // Optional
}
```

Response (201 Created):

```
{
  "success": true,
  "token": {
    "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "token": "Kd8jD_s9fN2mP1qR6tV8wX0yZ5aB3cE7gH4iJ",
    "spokeId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "spokeName": "laptop-001",
    "allowedIPs": ["10.0.1.5/24"],
    "createdAt": "2025-12-22T10:00:00Z",
    "expiresAt": "2025-12-23T10:00:00Z",
    "used": false
  },
  "installCommands": {
    "linux": "curl -sSL https://hub.example.com/api/installation/script/Kd8jD_s9fN2mP1qR6tV8
    "macos": "curl -sSL https://hub.example.com/api/installation/script/Kd8jD_s9fN2mP1qR6tV8
    "windows": "Invoke-WebRequest -Uri 'https://hub.example.com/api/installation/script/Kd8j
    "proxmox": "curl -sSL https://hub.example.com/api/installation/script/Kd8jD_s9fN2mP1qR6t
  }
}
```


Errors:

- 409 Conflict: Custom IP already allocated
- 400 Bad Request: Invalid IP address format
- 503 Service Unavailable: No available IPs in pool

GET /api/installation/tokens

Purpose: List all tokens (used and unused)

Query Parameters:

- `status` (optional): “unused” | “used” | “expired”

Response (200 OK):

```
{
  "tokens": [
    {
      "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
      "spokeName": "laptop-001",
      "allowedIPs": ["10.0.1.5/24"],
      "createdAt": "2025-12-22T10:00:00Z",
      "expiresAt": "2025-12-23T10:00:00Z",
      "used": true,
      "usedAt": "2025-12-22T10:05:00Z"
    }
  ],
  "totalCount": 25,
  "unusedCount": 5,
  "usedCount": 18,
  "expiredCount": 2
}
```

DELETE /api/installation/token/:id

Purpose: Revoke unused token

Response (200 OK):

```
{
  "success": true,
  "message": "Token revoked successfully"
}
```

Errors:

- 400 Bad Request: Token already used (cannot revoke)
- 404 Not Found: Token does not exist

7.3 Script Serving Endpoint

GET /api/installation/script/:token

Purpose: Serve platform-specific installation script

Query Parameters:

- `platform` (required): “linux” | “macos” | “windows” | “proxmox”

Response (200 OK):

```
#!/bin/bash
set -e

# Embedded configuration (replaced by backend)
TOKEN="Kd8jD_s9fN2mP1qR6tV8wX0yZ5aB3cE7gH4iJ"
HUB_ENDPOINT="203.0.113.5:51820"
HUB_PUBLIC_KEY="jNQCb8RA9pf..."
SPOKE_IP="10.0.1.5/24"
NETWORK_CIDR="10.0.1.0/24"
DNS_SERVERS="1.1.1.1, 8.8.8.8"
CALLBACK_URL="https://hub.example.com/api/spoke/register"

# ... (full script)
```

Content-Type: `text/x-shellscript` or `text/plain`

Errors:

- 400 Bad Request: Invalid token format
- 404 Not Found: Token does not exist
- 410 Gone: Token already used
- 410 Gone: Token expired
- 400 Bad Request: Invalid platform

Error Response Example:

```
{
  "error": "Installation token has expired",
  "code": "TOKEN_EXPIRED",
  "expiresAt": "2025-12-21T10:00:00Z",
  "recoveryHint": "Tokens are valid for 24 hours. Please generate a new token."
}
```

7.4 Spoke Registration Endpoints

POST /api/spoke/register

Purpose: Spoke reports public key after installation

Request Body:

```
{
  "token": "Kd8jD_s9fN2mP1qR6tV8wX0yZ5aB3cE7gH4iJ",
  "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
  "os": "linux",

  // Proxmox-specific (optional)
  "isProxmox": false,
  "proxmoxNodeName": null,
  "proxmoxClusterName": null,
  "proxmoxVersion": null,
  "metadata": {}
}
```

Response (201 Created):

```
{
  "success": true,
  "spoke": {
    "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "name": "laptop-001",
    "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
    "allowedIPs": ["10.0.1.5/24"],
    "registeredAt": "2025-12-22T10:05:30Z",
    "status": "pending"
  }
}
```


Errors:

- 400 Bad Request: Invalid public key format
- 409 Conflict: Public key already registered (impersonation attempt)
- 410 Gone: Token already used
- 410 Gone: Token expired

GET /api/spoke/list

Purpose: List all registered spokes

Query Parameters:

- `status` (optional): “pending” | “active” | “inactive”
- `os` (optional): “linux” | “macos” | “windows” | “proxmox”

Response (200 OK):

```
{
  "spokes": [
    {
      "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
      "name": "laptop-001",
      "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
      "allowedIPs": ["10.0.1.5/24"],
      "registeredAt": "2025-12-22T10:05:30Z",
      "lastHandshake": "2025-12-22T10:10:00Z",
      "status": "active",
      "os": "linux"
    }
  ],
  "totalCount": 25,
  "activeCount": 22,
  "inactiveCount": 2,
  "pendingCount": 1
}
```


GET /api/spoke/:id/status

Purpose: Get detailed spoke status

Response (200 OK):

```
{
  "spoke": {
    "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "name": "laptop-001",
    "publicKey": "xT0m9aV3bR5eD8fG1hJ4kL6nP9qS2uX7yZ0cE4iO=",
    "allowedIPs": ["10.0.1.5/24"],
    "registeredAt": "2025-12-22T10:05:30Z",
    "lastHandshake": "2025-12-22T10:10:00Z",
    "status": "active",
    "os": "linux",
    "metadata": {
      "wireguardVersion": "1.0.20210914",
      "kernelVersion": "5.15.0-91-generic"
    }
  },
  "wireguardPeer": {
    "endpoint": "192.0.2.50:51820",
    "latestHandshake": "2025-12-22T10:10:00Z",
    "transferRx": 1234567,
    "transferTx": 987654,
    "persistentKeepalive": 25
  }
}
```


DELETE /api/spoke/:id

Purpose: Remove spoke (revoke access)

Response (200 OK):


```
{
  "success": true,
  "message": "Spoke removed successfully"
}
```


Side Effects:

- Removes peer from WireGuard interface (`wg set wg0 peer <key> remove`)

- Marks spoke as deleted in database (soft delete)
- Spoke's IP becomes available for reuse

7.5 Proxmox Cluster Endpoints

GET /api/proxmox/clusters

Purpose: List all Proxmox clusters

Response (200 OK):

```
{
  "clusters": [
    {
      "id": "c3d4e5f6-a7b8-9012-cdef-123456789012",
      "clusterName": "datacenter1-prod",
      "datacenter": "US-East-1",
      "description": "Production cluster",
      "nodeCount": 3,
      "activeNodes": 3,
      "inactiveNodes": 0,
      "createdAt": "2025-12-22T11:00:00Z",
      "updatedAt": "2025-12-22T11:00:00Z"
    }
  ],
  "totalClusters": 3,
  "totalNodes": 8
}
```

GET /api/proxmox/clusters/:id

Purpose: Get cluster details with all nodes

Response (200 OK):

```
{
  "cluster": {
    "id": "c3d4e5f6-a7b8-9012-cdef-123456789012",
    "clusterName": "datacenter1-prod",
    "datacenter": "US-East-1",
    "description": "Production cluster",
    "createdAt": "2025-12-22T11:00:00Z",
    "updatedAt": "2025-12-22T11:00:00Z"
  },
  "nodes": [
    {
      "id": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
      "name": "pve1.dc1",
      "publicKey": "yU1n0bW4cS6fE9gH2iK5lM7oQ0rT3vY8zA1dF5jP=",
      "allowedIPs": ["10.0.1.10/24"],
      "proxmoxNodeName": "pve1",
      "proxmoxVersion": "8.1.3",
      "status": "active",
      "lastHandshake": "2025-12-22T11:05:00Z"
    },
    {
      "id": "c3d4e5f6-a7b8-9012-cdef-123456789013",
      "name": "pve2.dc1",
      "publicKey": "zA2o1cX5dT7gF0hI3jL6mN8pR1sU4wZ9aB2eG6kQ=",
      "allowedIPs": ["10.0.1.11/24"],
      "proxmoxNodeName": "pve2",
      "proxmoxVersion": "8.1.3",
      "status": "active",
      "lastHandshake": "2025-12-22T11:04:00Z"
    }
  ]
}
```


PUT /api/proxmox/clusters/:id

Purpose: Update cluster metadata

Request Body:

```
{
  "datacenter": "US-East-2",
  "description": "Updated description"
}
```

Response (200 OK):

```
{
  "success": true,
  "cluster": { /* updated cluster */ }
}
```

DELETE /api/proxmox/clusters/:id

Purpose: Delete empty cluster

Response (200 OK):

```
{
  "success": true,
  "message": "Cluster deleted successfully"
}
```

Errors:

- 400 Bad Request: Cluster still has nodes (must remove all nodes first)

---

8. Installation Script Design

8.1 Linux Installation Script Template

File: `src/backend/scripts/install-spoke-linux.sh.template`

Full Script (see plan file for complete implementation)

Key Features:

- Auto-detects distribution (Ubuntu/Debian/RHEL/CentOS/Fedora)

- Installs WireGuard via native package manager
- Generates keys locally (private key never transmitted)
- Registers with hub and receives configuration
- Creates systemd service for auto-start
- Comprehensive error handling with rollback

Placeholders Replaced:

```
TOKEN="{{TOKEN}}"
HUB_ENDPOINT="{{HUB_ENDPOINT}}"
HUB_PUBLIC_KEY="{{HUB_PUBLIC_KEY}}"
SPOKE_IP="{{SPOKE_IP}}"
NETWORK_CIDR="{{NETWORK_CIDR}}"
DNS_SERVERS="{{DNS_SERVERS}}"
CALLBACK_URL="{{CALLBACK_URL}}"
```

8.2 Proxmox Installation Script Template

File: `src/backend/scripts/install-spoke-proxmox.sh.template`

Proxmox-Specific Additions:

1. **Environment Detection**:

```
if ! command -v pveversion &> /dev/null; then
  echo "Error: This script is designed for Proxmox VE"
  exit 1
fi
```

2. **Cluster Detection**:

```
CLUSTER_STATUS=$(pvecm status 2>/dev/null || echo "not_clustered")

if echo "$CLUSTER_STATUS" | grep -q "Cluster information"; then
  IS_CLUSTERED=true
  CLUSTER_NAME=$(echo "$CLUSTER_STATUS" | grep "Name:" | awk '{print $2}')
  NODE_NAME=$(hostname)
  CLUSTER_NODES=$(pvecm nodes | grep -v "^Node" | awk '{print $3}' | tr '\n' ',' | sed
else
  IS_CLUSTERED=false
  NODE_NAME=$(hostname)
fi
```

3. **Registration with Metadata**:

```
curl -X POST "$CALLBACK_URL" \
  -H "Content-Type: application/json" \
  -d "{
    \"token\": \"$TOKEN\",
    \"publicKey\": \"$PUBLIC_KEY\",
    \"os\": \"proxmox\",
    \"isProxmox\": true,
    \"proxmoxNodeName\": \"$NODE_NAME\",
    \"proxmoxClusterName\": \"$CLUSTER_NAME\",
    \"proxmoxVersion\": \"$PVE_VERSION\",
    \"metadata\": {
      \"isClustered\": $IS_CLUSTERED,
      \"clusterNodes\": \"$CLUSTER_NODES\"
    }
  }"
```

8.3 macOS Installation Script

Differences from Linux:

- Install via Homebrew: `brew install wireguard-tools`
- No systemd - use `launchd` plist for auto-start:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/Propert
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.wireguard.wg0</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/wg-quick</string>
    <string>up</string>
    <string>wg0</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

8.4 Windows PowerShell Script

File: `src/backend/scripts/install-spoke-windows.ps1.template`

Key Differences:

- Requires admin privileges: `#Requires -RunAsAdministrator`
- Install via Chocolatey: `choco install wireguard -y`
- Config location: `%ProgramFiles%\WireGuard\wg0.conf`
- Service management: `wireguard /installtunnelservice wg0.conf`

PowerShell Syntax:

```
# Generate keys
$PrivateKey = & wg genkey
$PublicKey = $PrivateKey | wg pubkey

# Register with hub
$Body = @{
  token = $TOKEN
  publicKey = $PublicKey
  os = "windows"
} | ConvertTo-Json

$Response = Invoke-RestMethod -Uri $CALLBACK_URL -Method POST -Body $Body -ContentType "appl

# Create config
$ConfigContent = @"
[Interface]
Address = $SPOKE_IP
PrivateKey = $PrivateKey
DNS = $DNS_SERVERS

[Peer]
PublicKey = $HUB_PUBLIC_KEY
Endpoint = $HUB_ENDPOINT
AllowedIPs = $NETWORK_CIDR
PersistentKeepalive = 25
"@

Set-Content -Path "$env:ProgramFiles\WireGuard\wg0.conf" -Value $ConfigContent

# Install and start service
& wireguard /installtunnelservice "$env:ProgramFiles\WireGuard\wg0.conf"
```

---

9. Frontend Components

9.1 Component Hierarchy

```
App
├─ HubContext (State Management)
│
├─ HubInitializer (First-Run Setup)
│   ├─ Network Configuration Form
│   ├─ Endpoint Input
│   ├─ DNS Settings
│   └─ Initialize Button
│
├─ Dashboard
│   ├─ HubStatusPanel
│   │   ├─ Hub Public Key Display
│   │   ├─ Network CIDR Info
│   │   ├─ Active Spokes Count
│   │   └─ Edit Config Button
│   │
│   ├─ SpokeManager
│   │   ├─ View Mode Toggle (All | By Type | Proxmox Clusters)
│   │   ├─ Add Spoke Button → InstallationTokenGenerator
│   │   ├─ Spoke Table/List
│   │   │   ├─ Regular Spokes (Linux/Mac/Windows)
│   │   │   └─ ProxmoxClusterView
│   │   └─ Status Filters
│   │
│   └─ DashboardSummary
│       ├─ Total Spokes
│       ├─ Proxmox Stats (Clusters, Nodes)
│       └─ Network Utilization
│
├─ InstallationTokenGenerator (Modal)
│   ├─ Spoke Name Input
│   ├─ Custom IP Input (Optional)
│   ├─ Generate Token Button
```

```
│   └─ Success → InstallationInstructions
│
└─ InstallationInstructions (Modal)
    ├─ Tabbed Interface (Linux | macOS | Windows | Proxmox)
    ├─ Copy-to-Clipboard Buttons
    ├─ Token Expiration Countdown
    └─ Installation Status Polling
```

9.2 HubContext (State Management)

File: `src/contexts/HubContext.tsx`

```
interface HubContextValue {
  // Hub state
  hubConfig: HubConfig | null
  hubInitialized: boolean
  hubStatus: HubStatus | null

  // Spoke state
  spokes: SpokeRegistration[]
  pendingTokens: InstallationToken[]
  proxmoxClusters: ProxmoxCluster[]

  // Loading states
  isLoading: boolean
  error: string | null

  // Hub actions
  initializeHub: (config: HubInitConfig) => Promise<void>
  updateHubConfig: (updates: Partial<HubConfig>) => Promise<void>
  refreshHubStatus: () => Promise<void>

  // Token actions
  generateToken: (spokeName: string, customIP?: string) => Promise<InstallationToken>
  revokeToken: (tokenId: string) => Promise<void>
  refreshTokens: () => Promise<void>

  // Spoke actions
  refreshSpokes: () => Promise<void>
  removeSpoke: (spokeId: string) => Promise<void>

  // Proxmox actions
  refreshProxmoxClusters: () => Promise<void>
  updateClusterMetadata: (clusterId: string, updates: { datacenter?: string, description?: s
}

export const HubProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [hubConfig, setHubConfig] = useState<HubConfig | null>(null)
  const [spokes, setSpokes] = useState<SpokeRegistration[]>([])
  // ... other state
```

```
  // Fetch hub config on mount
  useEffect(() => {
    fetch('/api/hub/config')
      .then(res => res.json())
      .then(data => setHubConfig(data))
      .catch(err => setError(err.message))
  }, [])

  // Auto-refresh spokes every 30 seconds
  useEffect(() => {
    const interval = setInterval(() => {
      refreshSpokes()
    }, 30000)
    return () => clearInterval(interval)
  }, [])

  // ... implementation of all methods

  return (
    <HubContext.Provider value={{ /* all values */ }}>
      {children}
    </HubContext.Provider>
  )
}
```

9.3 HubInitializer Component

File: `src/components/HubInitializer.tsx`

Purpose: First-run setup wizard

```
export const HubInitializer: React.FC = () => {
  const { initializeHub } = useHubContext()
  const [formData, setFormData] = useState({
    networkCIDR: '10.0.1.0/24',
    listenPort: 51820,
    endpoint: '',
    dns: '1.1.1.1, 8.8.8.8'
  })

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()

    const config: HubInitConfig = {
      networkCIDR: formData.networkCIDR,
      listenPort: formData.listenPort,
      endpoint: formData.endpoint,
      dns: formData.dns.split(',').map(s => s.trim())
    }

    await initializeHub(config)
  }

  return (
    <div className="hub-initializer">
      <h1>Hub Setup Wizard</h1>
      <form onSubmit={handleSubmit}>
        <label>
          Network CIDR:
          <input
            type="text"
            value={formData.networkCIDR}
            onChange={e => setFormData({ ...formData, networkCIDR: e.target.value })}
            placeholder="10.0.1.0/24"
          />
        </label>

        <label>
          Public Endpoint:
```

```
          <input
            type="text"
            value={formData.endpoint}
            onChange={e => setFormData({ ...formData, endpoint: e.target.value })}
            placeholder="203.0.113.5:51820"
            required
          />
        </label>

        <label>
          DNS Servers (comma-separated):
          <input
            type="text"
            value={formData.dns}
            onChange={e => setFormData({ ...formData, dns: e.target.value })}
          />
        </label>

        <button type="submit">Initialize Hub</button>
      </form>
    </div>
  )
}
```

9.4 SpokeManager Component

File: `src/components/SpokeManager.tsx`

```
export const SpokeManager: React.FC = () => {
  const { spokes, removeSpoke, refreshSpokes } = useHubContext()
  const [viewMode, setViewMode] = useState<'all' | 'by-type' | 'proxmox-clusters'>('all')
  const [showTokenGenerator, setShowTokenGenerator] = useState(false)

  const getStatusIcon = (spoke: SpokeRegistration) => {
    if (spoke.status === 'active') return '✅'
    if (spoke.status === 'pending') return '🟡'
    return '❌'
  }

  const getLastHandshakeDisplay = (lastHandshake?: Date) => {
    if (!lastHandshake) return 'Never'
    const seconds = Math.floor((Date.now() - new Date(lastHandshake).getTime()) / 1000)
    if (seconds < 60) return `${seconds} sec ago`
    if (seconds < 3600) return `${Math.floor(seconds / 60)} min ago`
    return `${Math.floor(seconds / 3600)} hours ago`
  }

  return (
    <div className="spoke-manager">
      <div className="header">
        <h2>Spokes</h2>
        <button onClick={() => setShowTokenGenerator(true)}>+ Add Spoke</button>
        <button onClick={refreshSpokes}>Refresh</button>
      </div>

      <div className="view-mode-toggle">
        <button onClick={() => setViewMode('all')}>All Spokes</button>
        <button onClick={() => setViewMode('by-type')}>By Type</button>
        <button onClick={() => setViewMode('proxmox-clusters')}>Proxmox Clusters</button>
      </div>

      {viewMode === 'all' && (
        <table>
          <thead>
            <tr>
              <th>Status</th>
```

```
              <th>Name</th>
              <th>IP</th>
              <th>OS</th>
              <th>Last Handshake</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {spokes.map(spoke => (
              <tr key={spoke.id}>
                <td>{getStatusIcon(spoke)}</td>
                <td>{spoke.name}</td>
                <td>{spoke.allowedIPs[0]}</td>
                <td>{spoke.os}</td>
                <td>{getLastHandshakeDisplay(spoke.lastHandshake)}</td>
                <td>
                  <button onClick={() => removeSpoke(spoke.id)}>Remove</button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      )}

      {viewMode === 'proxmox-clusters' && <ProxmoxClusterView />}

      {showTokenGenerator && (
        <InstallationTokenGenerator onClose={() => setShowTokenGenerator(false)} />
      )}
    </div>
  )
}
```

9.5 ProxmoxClusterView Component

File: `src/components/ProxmoxClusterView.tsx`

```
export const ProxmoxClusterView: React.FC = () => {
  const { spokes, proxmoxClusters } = useHubContext()
  const [expandedClusters, setExpandedClusters] = useState<Set<string>>(new Set())

  const toggleCluster = (clusterId: string) => {
    const newExpanded = new Set(expandedClusters)
    if (newExpanded.has(clusterId)) {
      newExpanded.delete(clusterId)
    } else {
      newExpanded.add(clusterId)
    }
    setExpandedClusters(newExpanded)
  }

  const getClusterStatus = (cluster: ProxmoxCluster) => {
    const nodes = spokes.filter(s => s.proxmoxClusterId === cluster.id)
    const activeNodes = nodes.filter(n => n.status === 'active').length
    const totalNodes = nodes.length

    if (activeNodes === totalNodes) return { icon: '✅', text: 'All Active' }
    if (activeNodes === 0) return { icon: '❌', text: 'All Inactive' }
    return { icon: '⚠️', text: 'Degraded' }
  }

  const standaloneProxmoxNodes = spokes.filter(s => s.isProxmox && !s.proxmoxClusterId)

  return (
    <div className="proxmox-cluster-view">
      {proxmoxClusters.map(cluster => {
        const status = getClusterStatus(cluster)
        const nodes = spokes.filter(s => s.proxmoxClusterId === cluster.id)
        const isExpanded = expandedClusters.has(cluster.id)

        return (
          <div key={cluster.id} className="cluster">
            <div className="cluster-header" onClick={() => toggleCluster(cluster.id)}>
              <span className="icon">🏢</span>
              <span className="name">Cluster: {cluster.clusterName}</span>
```

```
            <span className="node-count">({nodes.length} nodes)</span>
            <span className="status">{status.icon} {status.text}</span>
            <span className="toggle">{isExpanded ? '▼' : '▶'}</span>
          </div>

          {isExpanded && (
            <div className="cluster-nodes">
              {nodes.map(node => (
                <div key={node.id} className="node">
                  <span className="icon">🖥️</span>
                  <span className="name">{node.proxmoxNodeName}</span>
                  <span className="ip">{node.allowedIPs[0]}</span>
                  <span className="status">{node.status === 'active' ? '✅ Active' : '❌
                  <span className="handshake">
                    {getLastHandshakeDisplay(node.lastHandshake)}
                  </span>
                </div>
              ))}
            </div>
          )}
        </div>
      )
    })}

    {standaloneProxmoxNodes.length > 0 && (
      <div className="standalone-section">
        <h3>🖥️ Standalone Proxmox Hosts</h3>
        {standaloneProxmoxNodes.map(node => (
          <div key={node.id} className="node">
            <span className="icon">🖥️</span>
            <span className="name">{node.name}</span>
            <span className="ip">{node.allowedIPs[0]}</span>
            <span className="status">{node.status === 'active' ? '✅ Active' : '❌ Inacti
          </div>
        ))}
      </div>
    )}
  </div>
```

```
    )
}
```

9.6 InstallationInstructions Component

File: `src/components/InstallationInstructions.tsx`

```
export const InstallationInstructions: React.FC<{ token: InstallationToken }> = ({ token })
  const [selectedPlatform, setSelectedPlatform] = useState<'linux' | 'macos' | 'windows' |
  const [copied, setCopied] = useState(false)

  const commands = {
    linux: `curl -sSL https://hub.example.com/api/installation/script/${token.token}?platfo
    macos: `curl -sSL https://hub.example.com/api/installation/script/${token.token}?platfo
    windows: `Invoke-WebRequest -Uri "https://hub.example.com/api/installation/script/${toke
    proxmox: `curl -sSL https://hub.example.com/api/installation/script/${token.token}?plat
  }

  const copyToClipboard = () => {
    navigator.clipboard.writeText(commands[selectedPlatform])
    setCopied(true)
    setTimeout(() => setCopied(false), 2000)
  }

  return (
    <div className="installation-instructions">
      <h2>Installation Instructions</h2>

      <div className="tabs">
        <button onClick={() => setSelectedPlatform('linux')}>Linux</button>
        <button onClick={() => setSelectedPlatform('macos')}>macOS</button>
        <button onClick={() => setSelectedPlatform('windows')}>Windows</button>
        <button onClick={() => setSelectedPlatform('proxmox')}>Proxmox VE</button>
      </div>

      <div className="command-box">
        <code>{commands[selectedPlatform]}</code>
        <button onClick={copyToClipboard}>
          {copied ? 'Copied!' : 'Copy'}
        </button>
      </div>

      {selectedPlatform === 'proxmox' && (
        <div className="proxmox-note">
          <strong>Proxmox Multi-Node Deployment</strong>
```

```
          <p>For clustered Proxmox setups:</p>
          <ol>
            <li>Run this command on EACH node in the cluster</li>
            <li>Each node will auto-detect cluster membership</li>
            <li>Each node gets a unique IP and configuration</li>
          </ol>
        </div>
      )}

      <div className="token-info">
        <p>Token expires: {new Date(token.expiresAt).toLocaleString()}</p>
        <p>Allocated IP: {token.allowedIPs[0]}</p>
      </div>
    </div>
  )
}
```

---

10. Backend Services

10.1 TokenService

File: `src/backend/services/TokenService.ts`

Already Implemented (see code in repository)

Key Methods:

- `generateToken()` : Create secure 256-bit token, allocate IP
- `validateAndMarkUsed()` : Atomic validation and usage marking
- `revokeToken()` : Invalidate unused token
- `cleanupExpired()` : Remove expired tokens (cron job)

10.2 WireGuardService

File: `src/backend/services/WireGuardService.ts`

Purpose: Manage hub WireGuard interface

```
import { spawn } from 'child_process'
import { db } from '../config/database'
import type { SpokeRegistration, HubConfig } from '../../types'

export class WireGuardService {
  /**
   * Initialize hub WireGuard interface (first-run setup)
   */
  static async initializeHub(config: HubConfig): Promise<void> {
    // Generate keys
    const privateKey = await this.executeCommand('wg', ['genkey'])
    const publicKey = await this.executeCommand('sh', ['-c', `echo "${privateKey}" | wg publ

    // Create /etc/wireguard/wg0.conf
    const wgConfig = `
[Interface]
Address = ${config.interfaceAddress}
ListenPort = ${config.listenPort}
PrivateKey = ${privateKey}
`.trim()

    await this.writeFile('/etc/wireguard/wg0.conf', wgConfig)
    await this.executeCommand('chmod', ['600', '/etc/wireguard/wg0.conf'])

    // Enable and start service
    await this.executeCommand('systemctl', ['enable', 'wg-quick@wg0'])
    await this.executeCommand('systemctl', ['start', 'wg-quick@wg0'])

    // Store keys in database (encrypted)
    db.prepare(`
      UPDATE hub_config
      SET private_key = ?, public_key = ?
      WHERE id = 1
    `).run(privateKey, publicKey)
  }

  /**
   * Add spoke peer to WireGuard interface
```

```
 */
static async addSpokePeer(spoke: SpokeRegistration): Promise<void> {
  // Extract IP without CIDR prefix for allowed-ips
  const ip = spoke.allowedIPs[0].split('/')[0]

  // Add peer
  await this.executeCommand('wg', [
    'set', 'wg0',
    'peer', spoke.publicKey,
    'allowed-ips', `${ip}/32`,
    'persistent-keepalive', '25'
  ])

  // Persist config
  await this.executeCommand('wg-quick', ['save', 'wg0'])
}

/**
 * Remove spoke peer from WireGuard interface
 */
static async removeSpokePeer(publicKey: string): Promise<void> {
  await this.executeCommand('wg', ['set', 'wg0', 'peer', publicKey, 'remove'])
  await this.executeCommand('wg-quick', ['save', 'wg0'])
}

/**
 * Reload WireGuard config without disrupting connections
 */
static async reloadConfig(): Promise<void> {
  await this.executeCommand('sh', ['-c', 'wg syncconf wg0 <(wg-quick strip wg0)'])
}

/**
 * Get current WireGuard interface status
 */
static async getStatus(): Promise<any> {
  const output = await this.executeCommand('wg', ['show', 'wg0', 'dump'])
```

```
    // Parse output (tab-separated values)
    const lines = output.trim().split('\n')
    const [interfaceLine, ...peerLines] = lines

    const [privateKey, publicKey, listenPort, fwmark] = interfaceLine.split('\t')

    const peers = peerLines.map(line => {
      const [peerPublicKey, presharedKey, endpoint, allowedIPs, latestHandshake, rxBytes, tx

      return {
        publicKey: peerPublicKey,
        endpoint: endpoint || null,
        allowedIPs: allowedIPs.split(','),
        latestHandshake: latestHandshake !== '0' ? new Date(parseInt(latestHandshake) * 100(
        transferRx: parseInt(rxBytes),
        transferTx: parseInt(txBytes),
        persistentKeepalive: parseInt(persistentKeepalive)
      }
    })

    return {
      interface: 'wg0',
      publicKey,
      listenPort: parseInt(listenPort),
      peerCount: peers.length,
      peers
    }
  }

  /**
   * Execute shell command safely (prevent injection)
   */
  private static executeCommand(command: string, args: string[]): Promise<string> {
    return new Promise((resolve, reject) => {
      const child = spawn(command, args, { shell: false })

      let stdout = ''
      let stderr = ''
```

```
      child.stdout.on('data', (data) => { stdout += data })
      child.stderr.on('data', (data) => { stderr += data })

      child.on('close', (code) => {
        if (code !== 0) {
          reject(new Error(`Command failed: ${command} ${args.join(' ')}\n${stderr}`))
        } else {
          resolve(stdout.trim())
        }
      })

      child.on('error', (err) => {
        reject(err)
      })
    })
  }

  private static async writeFile(path: string, content: string): Promise<void> {
    const fs = await import('fs/promises')
    await fs.writeFile(path, content, { mode: 0o600 })
  }
}
```

10.3 ScriptGenerator

File: `src/backend/services/ScriptGenerator.ts`

```
import { readFileSync } from 'fs'
import { join } from 'path'
import type { InstallationToken } from '../../types'

export class ScriptGenerator {
  private static templates = {
    linux: join(__dirname, '../scripts/install-spoke-linux.sh.template'),
    macos: join(__dirname, '../scripts/install-spoke-macos.sh.template'),
    windows: join(__dirname, '../scripts/install-spoke-windows.ps1.template'),
    proxmox: join(__dirname, '../scripts/install-spoke-proxmox.sh.template')
  }

  /**
   * Generate platform-specific installation script
   */
  static generateScript(
    platform: 'linux' | 'macos' | 'windows' | 'proxmox',
    tokenData: InstallationToken
  ): string {
    const templatePath = this.templates[platform]
    const template = readFileSync(templatePath, 'utf-8')

    // Replace placeholders
    const script = template
      .replace(/\{\{TOKEN\}\}/g, tokenData.token)
      .replace(/\{\{HUB_ENDPOINT\}\}/g, tokenData.hubEndpoint)
      .replace(/\{\{HUB_PUBLIC_KEY\}\}/g, tokenData.hubPublicKey)
      .replace(/\{\{SPOKE_IP\}\}/g, tokenData.allowedIPs[0])
      .replace(/\{\{NETWORK_CIDR\}\}/g, tokenData.networkCIDR)
      .replace(/\{\{DNS_SERVERS\}\}/g, tokenData.dns?.join(', ') || '')
      .replace(/\{\{CALLBACK_URL\}\}/g, `${process.env.API_BASE_URL}/api/spoke/register`)
      .replace(/\{\{PERSISTENT_KEEPALIVE\}\}/g, tokenData.persistentKeepalive.toString())

    return script
  }

  /**
   * Validate template exists
```

```
   */
  static templateExists(platform: string): boolean {
    return platform in this.templates
  }
}
```

10.4 IPAddressPool

File: `src/backend/services/IPAddressPool.ts`

Already Implemented (see code in repository)

Key Methods:

- `getNextAvailableIP()` : Find next free IP in CIDR range
- `isIPInNetwork()` : Validate IP belongs to network
- `isValidIP()` : Validate IP format

---

11. Deployment Guide

11.1 Prerequisites

Server Requirements:

- Ubuntu 22.04 LTS or newer
- Public IPv4 address
- Domain name (e.g., `hub.example.com`)
- Root or sudo access

Software Requirements:

- Node.js 18+ and npm
- WireGuard kernel module
- Nginx or Caddy (reverse proxy)
- Certbot (Let's Encrypt TLS)

11.2 Installation Steps

Step 1: Install Dependencies

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs

# Install WireGuard
sudo apt install -y wireguard wireguard-tools

# Install Nginx
sudo apt install -y nginx

# Install Certbot
sudo apt install -y certbot python3-certbot-nginx
```

Step 2: Clone Repository and Build

```
# Clone repository
git clone https://github.com/your-org/hub-and-spoke-wireguard.git
cd hub-and-spoke-wireguard

# Install dependencies
npm install

# Build frontend
npm run build

# Build backend
npm run build:backend
```

Step 3: Configure Environment Variables

```
# Create .env file
cat > .env.backend <<EOF
NODE_ENV=production
PORT=3000
DATABASE_PATH=/var/lib/wireguard-hub/database.sqlite
API_BASE_URL=https://hub.example.com
CORS_ORIGIN=https://hub.example.com
TOKEN_EXPIRATION_HOURS=24
RATE_LIMIT_MAX_TOKENS_PER_HOUR=10
EOF

# Create database directory
sudo mkdir -p /var/lib/wireguard-hub
sudo chown $USER:$USER /var/lib/wireguard-hub
```

Step 4: Initialize Database

```
npm run db:migrate
```

Step 5: Configure Nginx

```
# Create Nginx config
sudo cat > /etc/nginx/sites-available/wireguard-hub <<'EOF'
server {
    listen 80;
    server_name hub.example.com;

    # Let's Encrypt challenge
    location /.well-known/acme-challenge/ {
        root /var/www/html;
    }

    # Redirect to HTTPS
    location / {
        return 301 https://$server_name$request_uri;
    }
}

server {
    listen 443 ssl http2;
    server_name hub.example.com;

    # TLS certificates (will be configured by Certbot)
    ssl_certificate /etc/letsencrypt/live/hub.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/hub.example.com/privkey.pem;

    # TLS settings
    ssl_protocols TLSv1.3;
    ssl_prefer_server_ciphers off;
    ssl_ciphers 'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256

    # HSTS
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    # Serve static files (React SPA)
    root /home/ubuntu/hub-and-spoke-wireguard/dist;
    index index.html;
```

```
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Proxy API requests to Express backend
    location /api/ {
        proxy_pass http://127.0.0.1:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}
EOF

# Enable site
sudo ln -s /etc/nginx/sites-available/wireguard-hub /etc/nginx/sites-enabled/
sudo rm /etc/nginx/sites-enabled/default

# Test config
sudo nginx -t
```

Step 6: Obtain TLS Certificate

```
sudo certbot --nginx -d hub.example.com
```

Step 7: Create Systemd Service

```
# Create service file
sudo cat > /etc/systemd/system/wireguard-hub-api.service <<EOF
[Unit]
Description=WireGuard Hub API
After=network.target

[Service]
Type=simple
User=ubuntu
WorkingDirectory=/home/ubuntu/hub-and-spoke-wireguard
Environment="NODE_ENV=production"
EnvironmentFile=/home/ubuntu/hub-and-spoke-wireguard/.env.backend
ExecStart=/usr/bin/node dist-backend/server.js
Restart=on-failure
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF

# Reload systemd
sudo systemctl daemon-reload

# Enable and start service
sudo systemctl enable wireguard-hub-api
sudo systemctl start wireguard-hub-api

# Check status
sudo systemctl status wireguard-hub-api
```

Step 8: Configure Firewall

```
# Allow SSH (if not already allowed)
sudo ufw allow 22/tcp

# Allow HTTPS
sudo ufw allow 443/tcp

# Allow WireGuard
sudo ufw allow 51820/udp

# Enable firewall
sudo ufw enable
```

Step 9: Initialize Hub via Dashboard

1. Open browser: `https://hub.example.com`
2. Fill in first-run setup wizard:
   - Network CIDR: `10.0.1.0/24`
   - Listen Port: `51820`
   - Public Endpoint: `<your-public-ip>:51820`
   - DNS: `1.1.1.1, 8.8.8.8`
3. Click “Initialize Hub”

The hub is now operational!

11.3 Backup Strategy

Database Backup (Daily Cron):

```
# Create backup script
cat > /usr/local/bin/backup-wireguard-hub.sh <<'EOF'
#!/bin/bash
set -e

BACKUP_DIR="/var/backups/wireguard-hub"
DB_PATH="/var/lib/wireguard-hub/database.sqlite"
DATE=$(date +%Y-%m-%d_%H-%M-%S)

mkdir -p $BACKUP_DIR

# Backup database
sqlite3 $DB_PATH ".backup '$BACKUP_DIR/database_$DATE.sqlite'"

# Backup WireGuard config
cp /etc/wireguard/wg0.conf $BACKUP_DIR/wg0_$DATE.conf

# Encrypt with GPG
gpg --symmetric --cipher-algo AES256 $BACKUP_DIR/database_$DATE.sqlite
gpg --symmetric --cipher-algo AES256 $BACKUP_DIR/wg0_$DATE.conf

# Remove unencrypted files
rm $BACKUP_DIR/database_$DATE.sqlite
rm $BACKUP_DIR/wg0_$DATE.conf

# Keep only last 30 days
find $BACKUP_DIR -name "*.gpg" -mtime +30 -delete

echo "Backup completed: $DATE"
EOF

chmod +x /usr/local/bin/backup-wireguard-hub.sh

# Add cron job (daily at 2 AM)
echo "0 2 * * * /usr/local/bin/backup-wireguard-hub.sh" | sudo crontab -
```

12. Testing Strategy

12.1 Unit Tests

TokenService Tests:

```
describe('TokenService', () => {
  test('generateToken creates valid 256-bit token', () => {
    const token = TokenService.generateToken({ spokeName: 'test', hubConfig })
    expect(token.token.length).toBe(43) // 32 bytes base64url
  })

  test('validateAndMarkUsed prevents token reuse', () => {
    const token = TokenService.generateToken({ spokeName: 'test', hubConfig })
    TokenService.validateAndMarkUsed(token.token) // First use

    expect(() => TokenService.validateAndMarkUsed(token.token))
      .toThrow('TOKEN_ALREADY_USED') // Second use fails
  })

  test('expired tokens are rejected', () => {
    // Create token with past expiration
    const expiredToken = { ...tokenData, expiresAt: new Date('2020-01-01') }
    db.prepare('INSERT INTO installation_tokens ...').run(expiredToken)

    expect(() => TokenService.validateAndMarkUsed(expiredToken.token))
      .toThrow('TOKEN_EXPIRED')
  })
})
```

IPAddressPool Tests:

```
describe('IPAddressPool', () => {
  test('allocates IPs sequentially', () => {
    const ip1 = IPAddressPool.getNextAvailableIP('10.0.1.0/24', [])
    const ip2 = IPAddressPool.getNextAvailableIP('10.0.1.0/24', [ip1])

    expect(ip1).toBe('10.0.1.1/24') // First usable IP
    expect(ip2).toBe('10.0.1.2/24')
  })

  test('throws error when pool exhausted', () => {
    const usedIPs = Array.from({ length: 254 }, (_, i) => `10.0.1.${i + 1}/24`)

    expect(() => IPAddressPool.getNextAvailableIP('10.0.1.0/24', usedIPs))
      .toThrow('No available IP addresses')
  })
})
```

12.2 Integration Tests

API Endpoint Tests:

```
describe('POST /api/installation/token', () => {
  test('generates token with valid payload', async () => {
    const response = await request(app)
      .post('/api/installation/token')
      .send({ spokeName: 'test-spoke' })
      .expect(201)

    expect(response.body).toHaveProperty('token')
    expect(response.body.token.token.length).toBe(43)
  })
})

describe('GET /api/installation/script/:token', () => {
  test('returns script for valid unused token', async () => {
    const token = await TokenService.generateToken({ spokeName: 'test', hubConfig })

    const response = await request(app)
      .get(`/api/installation/script/${token.token}?platform=linux`)
      .expect(200)

    expect(response.text).toContain('#!/bin/bash')
    expect(response.text).toContain(token.token)
  })

  test('rejects used token', async () => {
    const token = await TokenService.generateToken({ spokeName: 'test', hubConfig })
    TokenService.validateAndMarkUsed(token.token)

    const response = await request(app)
      .get(`/api/installation/script/${token.token}?platform=linux`)
      .expect(410)

    expect(response.body.code).toBe('TOKEN_ALREADY_USED')
  })
})
```

12.3 Manual Testing (VMs)

Test Matrix:

Platform	Version	Test Cases
Ubuntu	22.04	Happy path, expired token, used token, invalid public key
Debian	11	Happy path
CentOS	8	Happy path
macOS	13 (Ventura)	Happy path
Windows	11	Happy path, PowerShell execution policy
Proxmox VE	8.1	Standalone node, 3-node cluster

Test Procedure:

1. Generate token via dashboard
2. Run installation script on VM
3. Verify WireGuard interface created
4. Verify spoke appears in dashboard as “Active”
5. Test connectivity: `ping 10.0.1.1` (hub)
6. Test spoke removal: Delete spoke in dashboard, verify interface removed

Proxmox-Specific Tests:

1. **Standalone Node**: Install on single Proxmox host, verify `is_proxmox=true`, `proxmox_cluster_id=null`
2. **3-Node Cluster**: Install on all 3 nodes, verify cluster auto-detected, all nodes grouped in dashboard
3. **Cluster Communication**: Test ping between nodes over VPN
4. **VM Migration**: Test live migration of VM between nodes (optional, performance test)

---

13. Operations and Monitoring

13.1 Monitoring Metrics

System Metrics (via Prometheus + Node Exporter):

- CPU usage
- Memory usage
- Disk I/O
- Network throughput

WireGuard Metrics:

- Active peer count
- Handshake failures
- Data transfer (RX/TX bytes per peer)
- Last handshake timestamp

Application Metrics (Custom):

- Token generation rate
- Token usage rate
- Spoke registration rate
- API request latency
- API error rate

13.2 Prometheus Configuration

prometheus.yml:

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'wireguard-hub'
    static_configs:
      - targets: ['localhost:9100']  # Node exporter
      - targets: ['localhost:3000']  # Express app (custom metrics)
```

Express Metrics Endpoint (`/api/metrics`):

```
import { collectDefaultMetrics, register, Counter, Histogram } from 'prom-client'

collectDefaultMetrics()

const tokenGenerationCounter = new Counter({
  name: 'wireguard_tokens_generated_total',
  help: 'Total number of installation tokens generated'
})

const spokeRegistrationCounter = new Counter({
  name: 'wireguard_spokes_registered_total',
  help: 'Total number of spokes registered'
})

const apiRequestDuration = new Histogram({
  name: 'wireguard_api_request_duration_seconds',
  help: 'API request duration',
  labelNames: ['method', 'route', 'status']
})

// Middleware to track metrics
app.use((req, res, next) => {
  const start = Date.now()
  res.on('finish', () => {
    const duration = (Date.now() - start) / 1000
    apiRequestDuration.labels(req.method, req.route?.path || req.path, res.statusCode.toStri
  })
  next()
})

// Metrics endpoint
app.get('/api/metrics', (req, res) => {
  res.set('Content-Type', register.contentType)
  res.end(register.metrics())
})
```

13.3 Grafana Dashboard

Key Panels:

1. **Hub Overview**:
    - Total spokes (gauge)
    - Active spokes (gauge)
    - WireGuard uptime (uptime metric)
2. **Spoke Activity**:
    - Spoke registrations over time (time series)
    - Handshake failures (counter)
    - Average handshake interval (histogram)
3. **System Health**:
    - CPU usage (line graph)
    - Memory usage (line graph)
    - Network throughput (area graph)
4. **API Performance**:
    - Request rate (time series)
    - Latency percentiles (p50, p95, p99)
    - Error rate (time series)

13.4 Alerting Rules

Critical Alerts:

- Hub WireGuard interface down
- Database file permissions changed
- > *50% of spokes inactive*
- API error rate >5%

Warning Alerts:

- Disk space <10%

- Memory usage >80%
- Any Proxmox cluster with >50% nodes offline

Prometheus Alerting Rules:

```
groups:
  - name: wireguard_hub
    rules:
      - alert: HubInterfaceDown
        expr: up{job="wireguard-hub"} == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "WireGuard hub interface is down"

      - alert: HighSpokeInactiveRate
        expr: (wireguard_spokes_inactive / wireguard_spokes_total) > 0.5
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "More than 50% of spokes are inactive"
```

13.5 Log Management

Log Locations:

- **Express API**: `journalctl -u wireguard-hub-api -f`
- **WireGuard**: `journalctl -u wg-quick@wg0 -f`
- **Nginx**: `/var/log/nginx/access.log`, `/var/log/nginx/error.log`

Structured Logging (Winston):

```
import winston from 'winston'

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: '/var/log/wireguard-hub/error.log', level: 'erro
    new winston.transports.File({ filename: '/var/log/wireguard-hub/combined.log' }),
    new winston.transports.Console({ format: winston.format.simple() })
  ]
})

// Usage
logger.info('Token generated', { tokenId: token.id, spokeName: token.spokeName })
logger.error('Spoke registration failed', { error: err.message, tokenId })
```

---

14. Risk Assessment

14.1 Technical Risks

Risk	Likelihood	Impact	Mitigation
Hub server failure	Medium	Critical	Implement HA with standby hub, automated failover
Database corruption	Low	High	Daily encrypted backups, WAL mode enabled
WireGuard key compromise	Low	Critical	Secure key storage, encrypted backups, rotation policy
Token brute force	Very Low	Low	256-bit entropy, rate limiting
SQLite scalability limits	Medium	Medium	Migrate to PostgreSQL at 1000+ spokes
Network bandwidth saturation	Low	Medium	Monitor bandwidth, upgrade link as needed

14.2 Operational Risks

Risk	Likelihood	Impact	Mitigation
Misconfigured firewall blocks VPN	Medium	High	Automated firewall tests, monitoring
TLS certificate expiration	Low	High	Certbot auto-renewal, expiration alerts
Incorrect spoke removal	Low	Low	Confirmation dialog, audit trail
IP address exhaustion	Low	Medium	Monitor IP usage, plan network expansion

14.3 Security Risks

Risk	Likelihood	Impact	Mitigation
Stolen installation token	Medium	Medium	24-hour expiration, one-time use, HTTPS required
Public key impersonation	Very Low	Low	Uniqueness constraint, validation
Hub admin account compromise	Low	Critical	Implement authentication (future), 2FA, audit logs
DDoS attack on hub	Medium	High	Rate limiting, Cloudflare DDoS protection

14.4 Scalability Risks

SQLite Limitations:

- **Concurrent Writes**: Single-writer limitation
- **Database Size**: Practical limit ~1TB (but performance degrades at 100GB+)
- **Connection Pooling**: Not supported

Mitigation: Migrate to PostgreSQL when:

- Spoke count exceeds 1000
- API request rate exceeds 100 req/s
- Database size exceeds 10GB

---

15. Implementation Roadmap

Week 1: Backend Core (Days 1-5)

Day 1-2: Database and Core Services

- ✅ Database schema implementation
- ✅ TokenService implementation
- ✅ IPAddressPool implementation
- [ ] WireGuardService implementation
- [ ] ScriptGenerator implementation

Day 3-4: API Controllers and Routes

- [ ] HubController (initialize, config, status)
- [ ] InstallationController (token generation, script serving)
- [ ] SpokeController (registration, list, delete)
- [ ] ProxmoxController (cluster management)
- [ ] Express server setup with middleware

Day 5: Testing

- [ ] Unit tests for all services
- [ ] Integration tests for API endpoints
- [ ] Security testing (token validation, SQL injection)

Week 2: Installation Scripts (Days 6-10)

Day 6-7: Linux and Proxmox Scripts

- [ ] Linux installation script template
- [ ] Proxmox installation script with cluster detection
- [ ] Test on Ubuntu 22.04, Debian 11, CentOS 8

Day 8: macOS Script

- [ ] macOS installation script template
- [ ] launchd plist configuration

- [ ] Test on macOS Ventura

Day 9: Windows Script

- [ ] PowerShell installation script
- [ ] Chocolatey integration
- [ ] Test on Windows 11

Day 10: Script Testing

- [ ] VM testing matrix (all platforms)
- [ ] Error handling verification
- [ ] Rollback testing

Week 3: Frontend Components (Days 11-15)

Day 11-12: Core Components

- [ ] HubContext state management
- [ ] HubInitializer wizard
- [ ] SpokeManager component
- [ ] InstallationTokenGenerator

Day 13-14: Proxmox UI

- [ ] ProxmoxClusterView component
- [ ] Cluster grouping logic
- [ ] Status indicators
- [ ] InstallationInstructions with Proxmox tab

Day 15: UI Polish

- [ ] Styling with TailwindCSS
- [ ] Responsive design
- [ ] Loading states and error handling
- [ ] Copy-to-clipboard functionality

Week 4: Integration and Testing (Days 16-20)

Day 16-17: End-to-End Testing

- [ ] Complete provisioning flow (Linux)
- [ ] Complete provisioning flow (Proxmox cluster)
- [ ] Token expiration testing
- [ ] Token reuse prevention testing

Day 18-19: Security Testing

- [ ] Penetration testing (token theft scenarios)
- [ ] Rate limiting verification
- [ ] HTTPS enforcement
- [ ] Input validation testing

Day 20: Performance Testing

- [ ] Load testing (100 concurrent token generations)
- [ ] Database performance testing
- [ ] WireGuard scalability testing (100+ peers)

Week 5: Deployment and Documentation (Days 21-25)

Day 21-22: Production Deployment

- [ ] Deploy to production server
- [ ] Configure Nginx reverse proxy
- [ ] Set up TLS certificates
- [ ] Configure systemd services

Day 23-24: Monitoring and Operations

- [ ] Prometheus metrics implementation
- [ ] Grafana dashboards
- [ ] Alerting rules configuration
- [ ] Backup automation

Day 25: Documentation and Handoff

- [ ] User guide (dashboard usage)
- [ ] Operations guide (troubleshooting)
- [ ] API documentation

- [ ] Chief architect review and approval

Conclusion

This architecture provides a secure, scalable, and user-friendly solution for hub-and-spoke WireGuard VPN management with special focus on Proxmox multi-cluster deployments.

Key Strengths

1. **Security-First Design**: Zero-trust key generation, one-time-use tokens, atomic validation
2. **Multi-Platform Support**: Linux, macOS, Windows, Proxmox VE with auto-detection
3. **Proxmox Integration**: Auto-cluster detection, hierarchical visualization, multi-datacenter support
4. **Developer Experience**: TypeScript throughout, comprehensive type definitions, modular architecture
5. **Operations-Ready**: Monitoring, backups, logging, alerting built-in

Next Steps

1. **Chief Architect Review**: Obtain approval on this architecture document
2. **Implementation**: Follow 5-week roadmap
3. **Testing**: Comprehensive testing on all platforms
4. **Deployment**: Production deployment with monitoring
5. **Iteration**: Gather user feedback, implement enhancements

Future Enhancements (Post-MVP)

- **Authentication**: Add login system for dashboard (OAuth, SAML)
- **RBAC**: Role-based access control (admin, viewer, operator)
- **Spoke-to-Spoke Routing**: Optional mesh networking
- **IPv6 Support**: Dual-stack configuration
- **DNS Server**: Built-in DNS for spoke name resolution
- **Container Support**: Docker/Podman integration
- **Terraform Provider**: Infrastructure-as-code integration

Document Status: Ready for Chief Architect Review **Last Updated**: December 22, 2025 **Prepared By**: Claude Code AI **Version**: 1.0