🦞 AI PLATFORM DEPLOYMENT GUIDE v75.2.0

Complete Self-Hosted AI Infrastructure
Production-Ready | GPU/CPU Adaptive | Privacy-First | Enterprise-Grade

---

📋 TABLE OF CONTENTS

🎯 SYSTEM OVERVIEW

What This Deploys

A **complete AI infrastructure platform** combining:

- **Local LLM inference** (Ollama with GPU/CPU support)
- **API routing layer** (LiteLLM: simple queries → local, complex → cloud APIs)
- **AI applications** (Dify, Anything LLM, Open WebUI)
- **Autonomous coding agent** (OpenClaw via Tailscale)
- **Workflow automation** (n8n, Flowise)
- **Vector databases** (Qdrant/Weaviate/Chroma - user selects one)

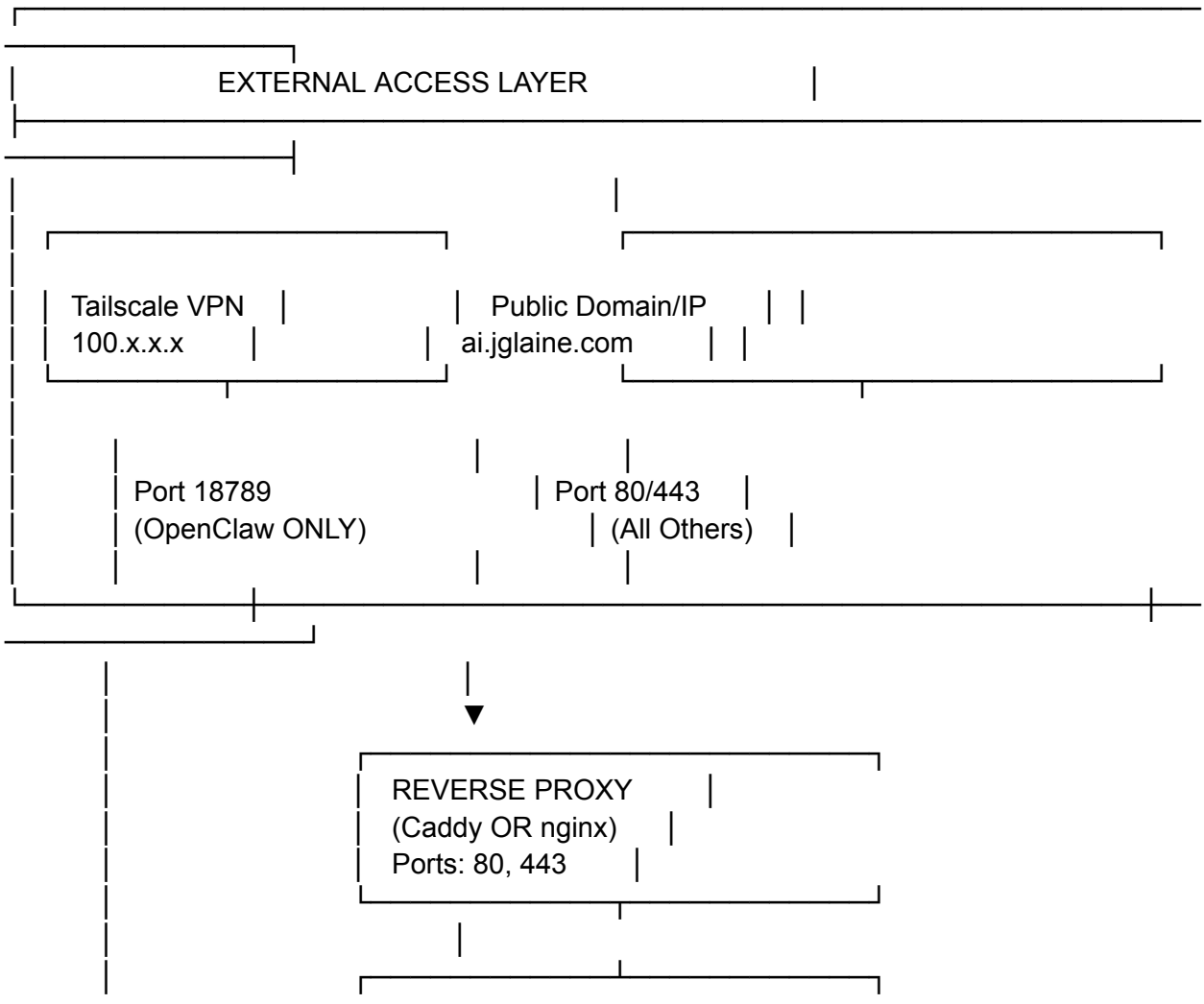
- **Monitoring stack** (Prometheus + Grafana)
- **Secure networking** (Tailscale VPN + reverse proxy)
- **Object storage** (MinIO S3-compatible)

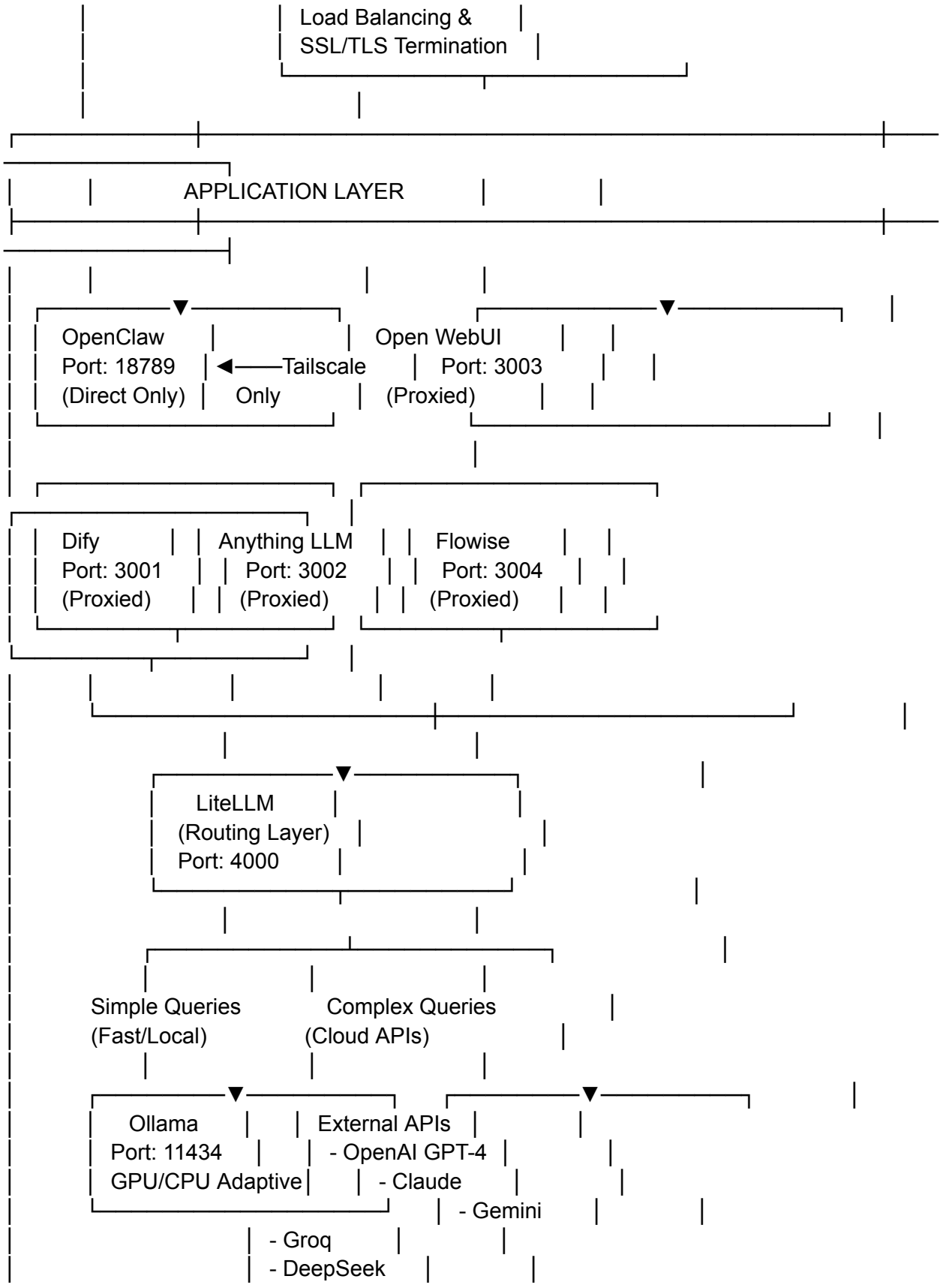
Key Design Principles

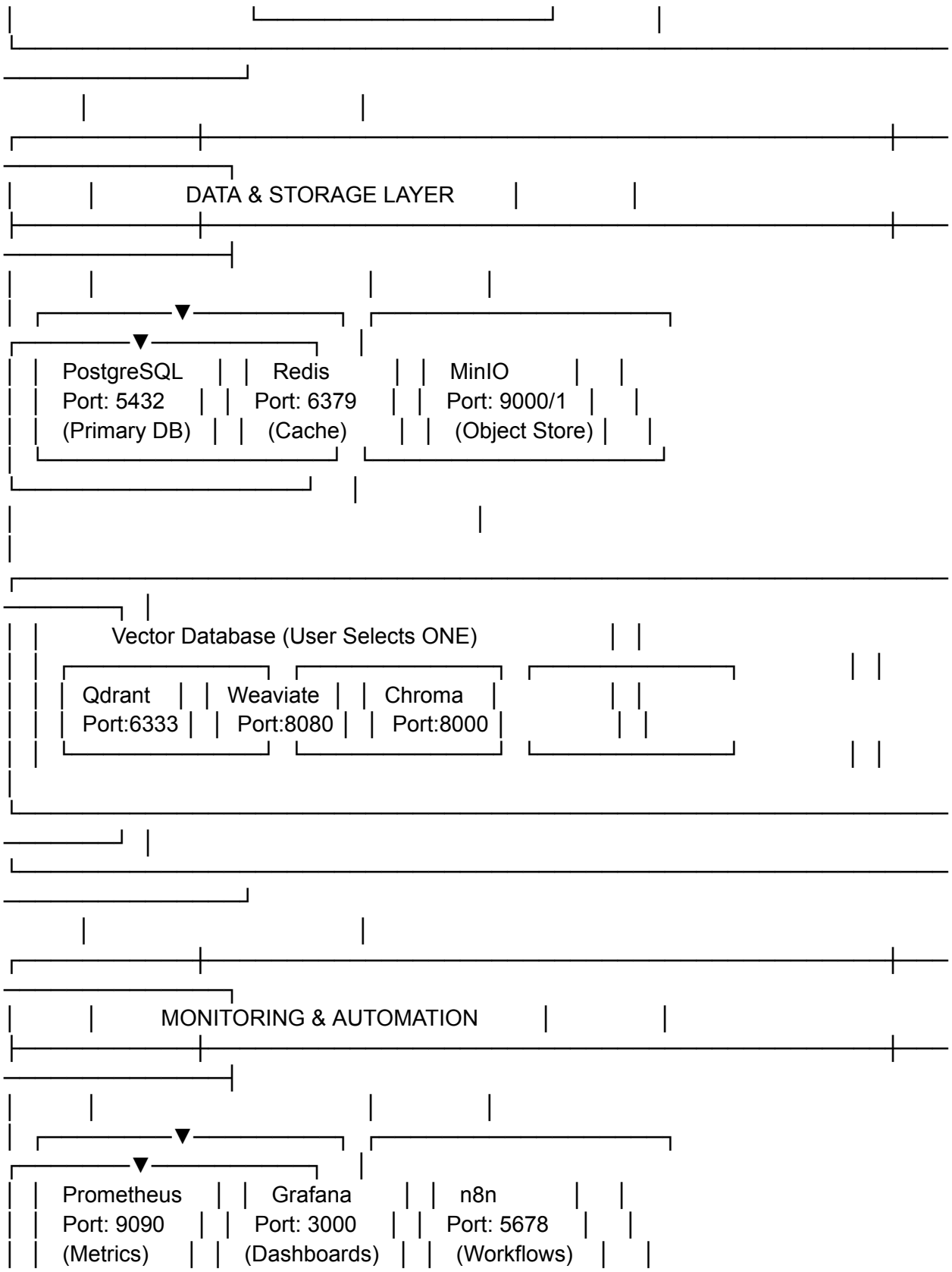
1. **Privacy-First**: All data stays on your infrastructure
2. **GPU/CPU Adaptive**: Automatically detects and optimizes for available hardware
3. **Hybrid Intelligence**: Local models for speed, cloud APIs for complexity
4. **Production-Ready**: Health checks, monitoring, automated backups
5. **Modular**: Enable/disable services based on needs
6. **Secure by Default**: Tailscale VPN, encrypted storage, credential isolation

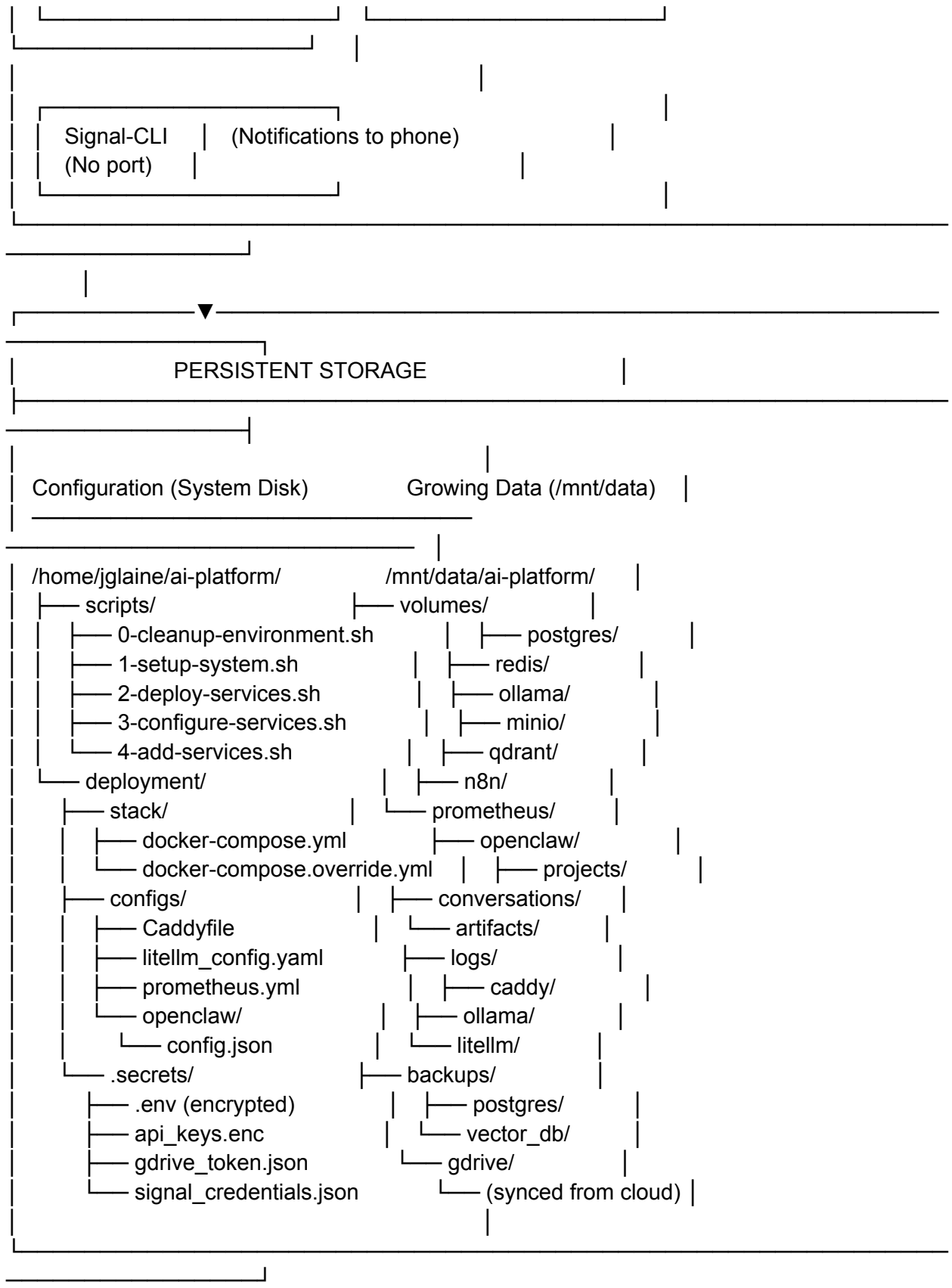
---

🏗️ ARCHITECTURE DIAGRAM




Load Balancing &
SSL/TLS Termination
APPLICATION LAYER
OpenClaw
Port: 18789
(Direct Only)
Tailscale
Only
Open WebUI
Port: 3003
(Proxied)
Dify
Port: 3001
(Proxied)
Anything LLM
Port: 3002
(Proxied)
Flowise
Port: 3004
(Proxied)
LiteLLM
(Routing Layer)
Port: 4000
Simple Queries
(Fast/Local)
Complex Queries
(Cloud APIs)
Ollama
Port: 11434
GPU/CPU Adaptive
External APIs
- OpenAI GPT-4
- Claude
- Gemini
- Groq
- DeepSeek


DATA & STORAGE LAYER
PostgreSQL
Port: 5432
(Primary DB)
Redis
Port: 6379
(Cache)
MinIO
Port: 9000/1
(Object Store)
Vector Database (User Selects ONE)
Qdrant
Port:6333
Weaviate
Port:8080
Chroma
Port:8000
MONITORING & AUTOMATION
Prometheus
Port: 9090
(Metrics)
Grafana
Port: 3000
(Dashboards)
n8n
Port: 5678
(Workflows)


Signal-CLI
(No port)
(Notifications to phone)
PERSISTENT STORAGE
Configuration (System Disk)
Growing Data (/mnt/data)
/home/jglaine/ai-platform/
├── scripts/
│ ├── 0-cleanup-environment.sh
│ ├── 1-setup-system.sh
│ ├── 2-deploy-services.sh
│ ├── 3-configure-services.sh
│ └── 4-add-services.sh
└── deployment/
├── stack/
│ ├── docker-compose.yml
│ └── docker-compose.override.yml
├── configs/
│ ├── Caddyfile
│ ├── litellm_config.yaml
│ ├── prometheus.yml
│ └── openclaw/
│ └── config.json
└── .secrets/
├── .env (encrypted)
├── api_keys.enc
├── gdrive_token.json
└── signal_credentials.json
/mnt/data/ai-platform/
├── volumes/
│ ├── postgres/
│ ├── redis/
│ ├── ollama/
│ ├── minio/
│ ├── qdrant/
│ ├── n8n/
│ └── prometheus/
├── openclaw/
│ ├── projects/
│ ├── conversations/
│ └── artifacts/
├── logs/
│ ├── caddy/
│ ├── ollama/
│ └── litellm/
├── backups/
│ ├── postgres/
│ └── vector_db/
└── gdrive/
└── (synced from cloud)

LEGEND:
  ──▶ Direct connection (no proxy)
  ═══▶ Proxied through Caddy/nginx
  ┈┈┈▶ Optional integration
  [GPU] GPU-accelerated when available
  [CPU] CPU fallback mode

📦 SERVICE INVENTORY

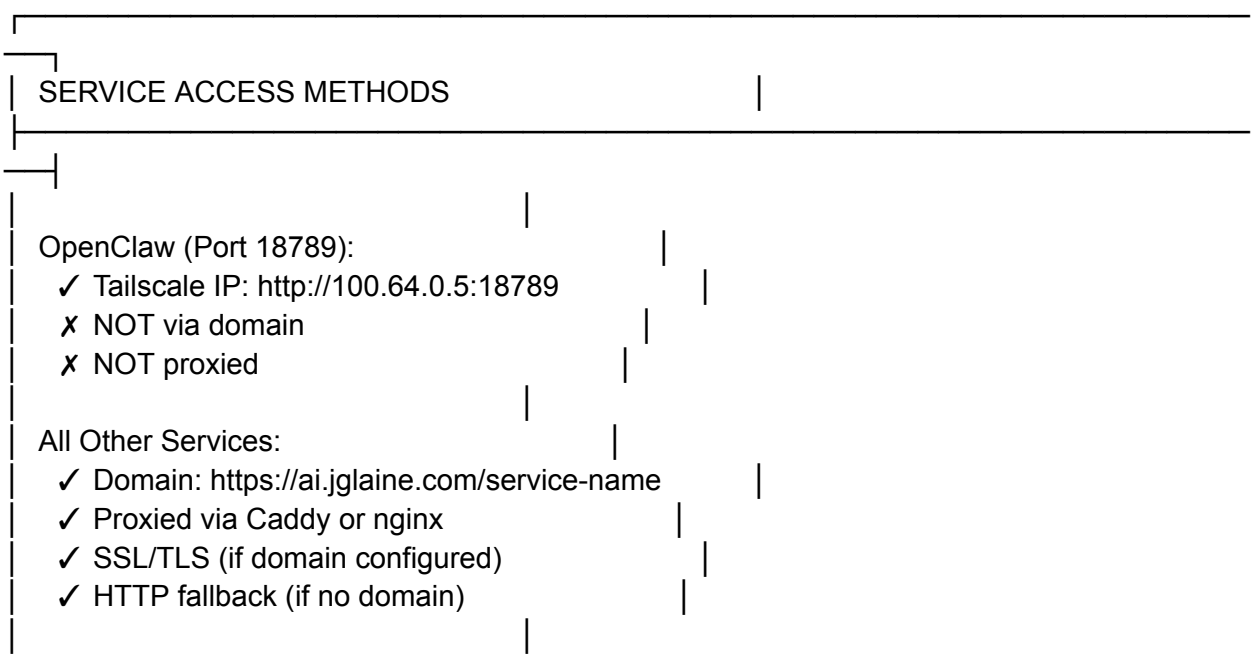
Total Services: 19

Copy table

#	Service	Port	Access Method	Purpose	Required
1	**Tailscale**	8443	VPN mesh network	Secure access to OpenClaw	Yes
2	**Caddy** OR **nginx**	80/443	N/A (is the proxy)	Reverse proxy for all services except OpenClaw	Yes (pick one)
3	**PostgreSQL**	5432	Internal only	Primary relational database	Yes
4	**Redis**	6379	Internal only	Caching layer	Yes
5	**MinIO**	9000/9001	Proxied via domain	S3-compatible object storage	Yes
6	**Prometheus**	9090	Proxied via domain	Metrics collection	Yes
7	**Grafana**	3000	Proxied via domain	Monitoring dashboards	Yes
8	**Ollama**	11434	Internal + proxied	Local LLM inference (GPU/CPU)	Yes
9	**LiteLLM**	4000	Internal only	Routing: simple→local, complex→APIs	Yes
10	**OpenClaw**	18789	**Tailscale IP ONLY**	AI coding agent	Yes

11	**Dify**	3001	Proxied via domain	LLM application builder	Yes
12	**Anything LLM**	3002	Proxied via domain	Document chat with RAG	Yes
13	**Open WebUI**	3003	Proxied via domain	Ollama web interface	Yes
14	**Qdrant**	6333	Internal + proxied	Vector database (option 1)	Pick ONE
15	**Weaviate**	8080	Internal + proxied	Vector database (option 2)	Pick ONE
16	**Chroma**	8000	Internal + proxied	Vector database (option 3)	Pick ONE
17	**n8n**	5678	Proxied via domain	Workflow automation	Yes
18	**Flowise**	3004	Proxied via domain	Visual LLM flow builder	Optional
19	**Signal-CLI**	None	Internal only	Phone notifications	Optional

Access Pattern Summary



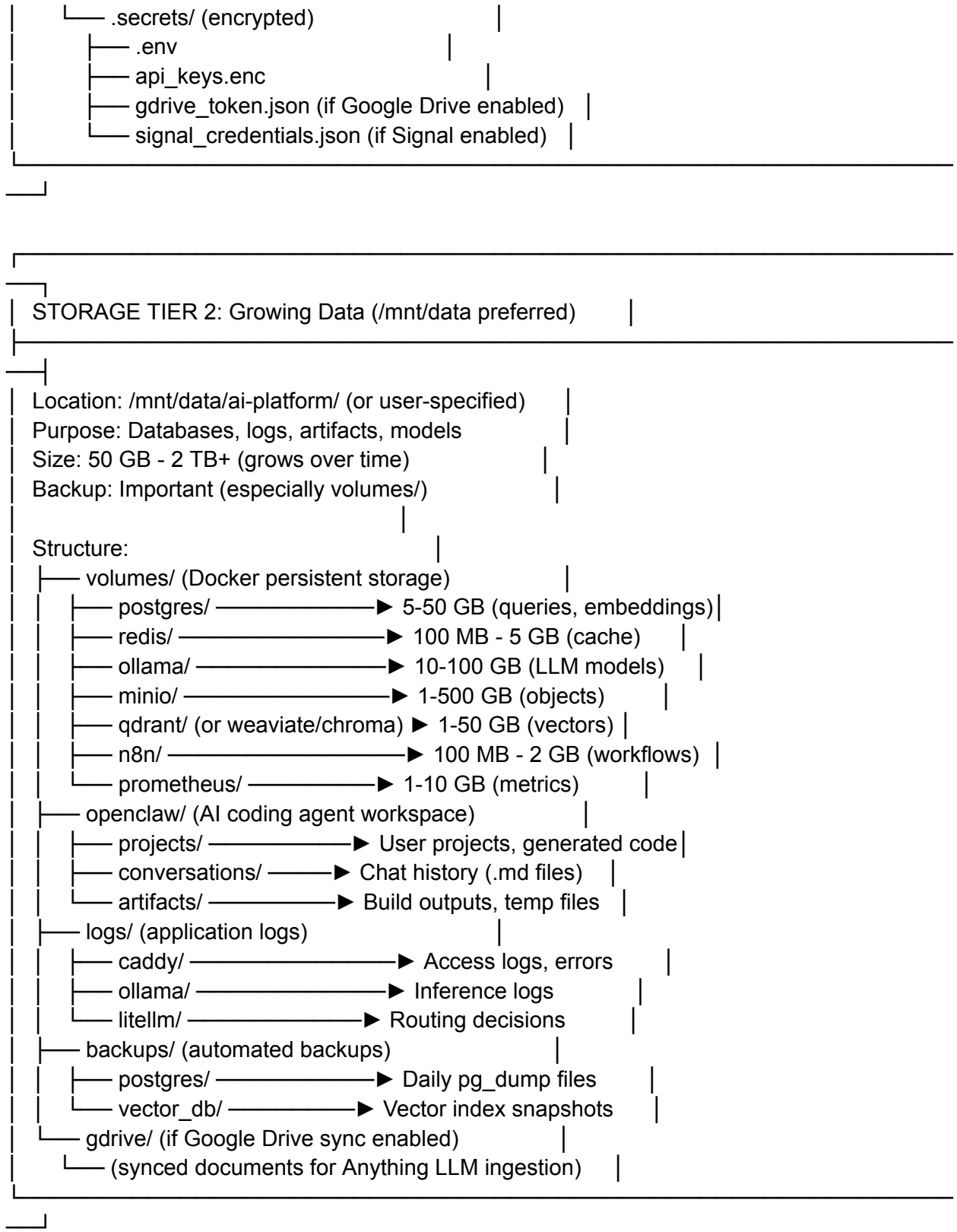
```
│  Examples:                                          │
│    https://ai.jglaine.com/webui    → Open WebUI     │
│    https://ai.jglaine.com/dify     → Dify           │
│    https://ai.jglaine.com/n8n      → n8n            │
│    https://ai.jglaine.com/grafana  → Grafana        │
│                                                     │
└─────────────────────────────────────────────────────┘
```

💾 STORAGE ARCHITECTURE

Design Philosophy

Separation of Concerns: Static configuration vs. growing data

```
┌─────────────────────────────────────────────────────┐
│  STORAGE TIER 1: Configuration (System Disk)        │
├─────────────────────────────────────────────────────┤
│  Location: /home/jglaine/ai-platform/               │
│  Purpose: Scripts, configs, credentials             │
│  Size: ~20 MB (static)                              │
│  Backup: Critical (includes .secrets/)              │
│                                                     │
│  Structure:                                         │
│  ├── scripts/                                       │
│  │   ├── 0-cleanup-environment.sh                   │
│  │   ├── 1-setup-system.sh                          │
│  │   ├── 2-deploy-services.sh                       │
│  │   ├── 3-configure-services.sh                    │
│  │   └── 4-add-services.sh                          │
│  └── deployment/                                    │
│      ├── stack/                                     │
│      │   ├── docker-compose.yml                     │
│      │   └── docker-compose.override.yml (if GPU)   │
│      ├── configs/                                   │
│      │   ├── Caddyfile                              │
│      │   ├── litellm_config.yaml                    │
│      │   ├── prometheus.yml                         │
│      │   └── openclaw/                              │
│      │       └── config.json                        │
```


└── .secrets/ (encrypted)
├── .env
├── api_keys.enc
├── gdrive_token.json (if Google Drive enabled)
└── signal_credentials.json (if Signal enabled)
STORAGE TIER 2: Growing Data (/mnt/data preferred)
Location: /mnt/data/ai-platform/ (or user-specified)
Purpose: Databases, logs, artifacts, models
Size: 50 GB - 2 TB+ (grows over time)
Backup: Important (especially volumes/)
Structure:
├── volumes/ (Docker persistent storage)
│ ├── postgres/ ► 5-50 GB (queries, embeddings)
│ ├── redis/ ► 100 MB - 5 GB (cache)
│ ├── ollama/ ► 10-100 GB (LLM models)
│ ├── minio/ ► 1-500 GB (objects)
│ ├── qdrant/ (or weaviate/chroma) ► 1-50 GB (vectors)
│ ├── n8n/ ► 100 MB - 2 GB (workflows)
│ └── prometheus/ ► 1-10 GB (metrics)
├── openclaw/ (AI coding agent workspace)
│ ├── projects/ ► User projects, generated code
│ ├── conversations/ ► Chat history (.md files)
│ └── artifacts/ ► Build outputs, temp files
├── logs/ (application logs)
│ ├── caddy/ ► Access logs, errors
│ ├── ollama/ ► Inference logs
│ └── litellm/ ► Routing decisions
├── backups/ (automated backups)
│ ├── postgres/ ► Daily pg_dump files
│ └── vector_db/ ► Vector index snapshots
└── gdrive/ (if Google Drive sync enabled)
└── (synced documents for Anything LLM ingestion)

Storage Detection Logic (Script 1)

Script 1 automatically detects best storage location

Scenario 1: /mnt/data exists, writable, >50GB free
  → Use /mnt/data/ai-platform/ for DATA_ROOT
  → Use /home/jglaine/ai-platform/ for CONFIG_ROOT

Scenario 2: /mnt/data doesn't exist
  → Prompt user for alternative
  → Default: /home/jglaine/ai-platform-data/
  → Warn if system disk < 100 GB free

Scenario 3: /mnt/data exists but not writable
  → Attempt sudo chown jglaine:jglaine /mnt/data
  → If successful: Use /mnt/data
  → If fails: Fall back to Scenario 2

Scenario 4: User wants custom location
  → Accept custom path
  → Validate: exists, writable, sufficient space
  → Update DATA_ROOT in .env

Environment Variables

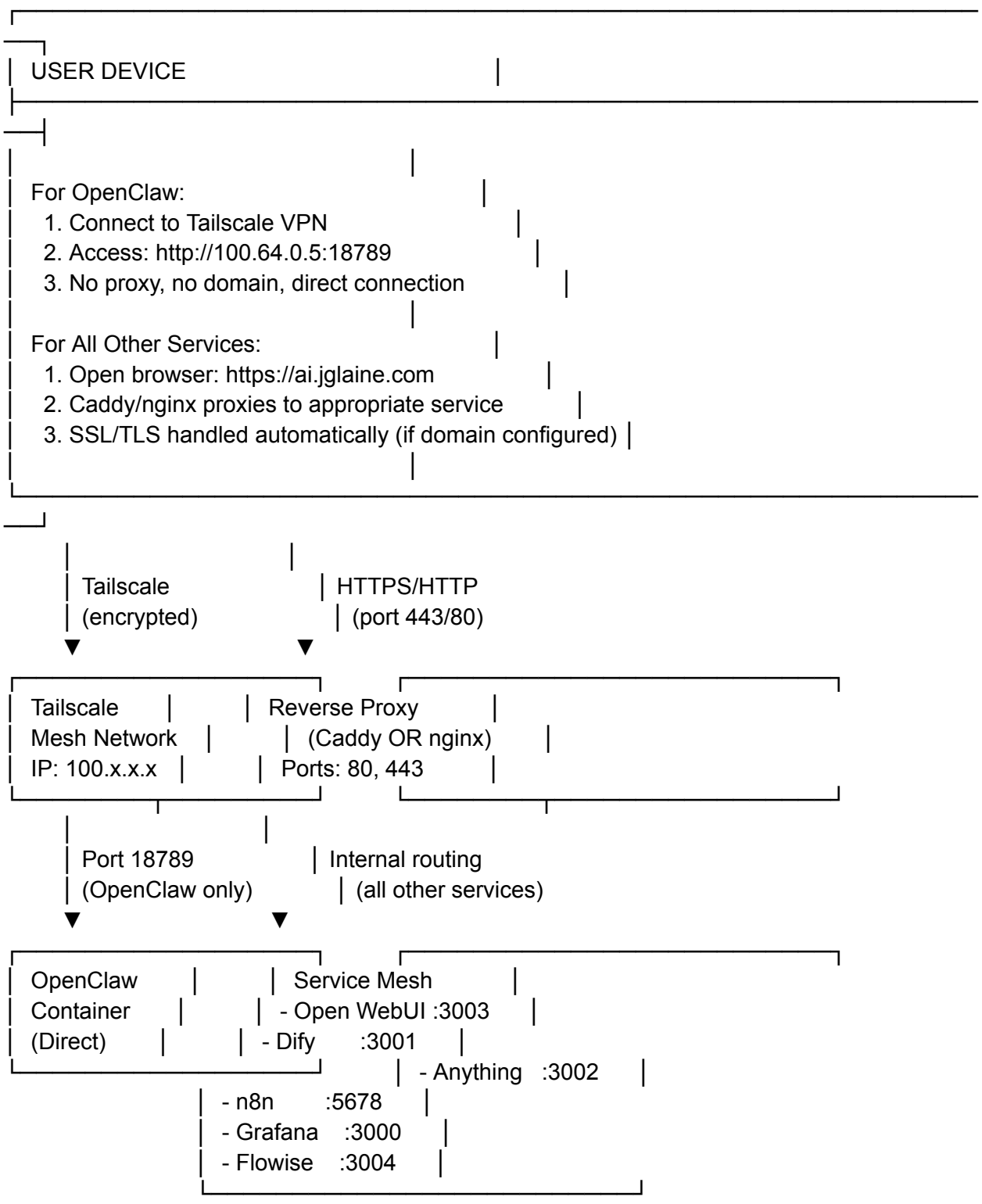
In deployment/.secrets/.env

```
# Storage paths
CONFIG_ROOT="/home/jglaine/ai-platform"
DATA_ROOT="/mnt/data/ai-platform"

# Docker uses these for bind mounts
POSTGRES_DATA="${DATA_ROOT}/volumes/postgres"
REDIS_DATA="${DATA_ROOT}/volumes/redis"
OLLAMA_DATA="${DATA_ROOT}/volumes/ollama"
OPENCLAW_WORKSPACE="${DATA_ROOT}/openclaw"
LOGS_DIR="${DATA_ROOT}/logs"
BACKUPS_DIR="${DATA_ROOT}/backups"
```

🌐 NETWORK & ACCESS PATTERNS

External Access Architecture



Reverse Proxy Configuration

Option 1: Caddy (Recommended)

Pros:

- Automatic HTTPS with Let's Encrypt
- Simpler configuration syntax
- Auto-renewal of certificates
- Built-in security headers

Caddyfile Example (generated by Script 1):

```
# AI Platform - Caddy Configuration
# Generated by 1-setup-system.sh v75.2.0

{
    email admin@jglaine.com
    # Global options
}

# Main domain entry
ai.jglaine.com {
    # Serve a simple landing page at root
    respond "AI Platform - Service Directory" 200
}

# Open WebUI
ai.jglaine.com/webui/* {
    reverse_proxy localhost:3003
}

# Dify
ai.jglaine.com/dify/* {
    reverse_proxy localhost:3001
}

# Anything LLM
ai.jglaine.com/anything/* {
    reverse_proxy localhost:3002
}

# n8n
ai.jglaine.com/n8n/* {
    reverse_proxy localhost:5678
}
```

```
# Grafana
ai.jglaine.com/grafana/* {
    reverse_proxy localhost:3000
}

# Flowise
ai.jglaine.com/flowise/* {
    reverse_proxy localhost:3004
}

# Ollama API (for external access)
ai.jglaine.com/ollama/* {
    reverse_proxy localhost:11434
}

# MinIO Console
ai.jglaine.com/minio/* {
    reverse_proxy localhost:9001
}

# Vector DB (example: Qdrant)
ai.jglaine.com/qdrant/* {
    reverse_proxy localhost:6333
}

# Prometheus
ai.jglaine.com/prometheus/* {
    reverse_proxy localhost:9090
}

# Security headers for all routes
header {
    X-Frame-Options "SAMEORIGIN"
    X-Content-Type-Options "nosniff"
    X-XSS-Protection "1; mode=block"
    Referrer-Policy "strict-origin-when-cross-origin"
}
```

Option 2: nginx

Pros:

- More fine-grained control
- Better performance at extreme scale

- Mature ecosystem

nginx.conf Example (generated by Script 1):

```
# AI Platform - nginx Configuration
# Generated by 1-setup-system.sh v75.2.0

http {
    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api_limit:10m rate=10r/s;

    # SSL configuration (requires manual cert management)
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    # Main server block
    server {
        listen 80;
        listen 443 ssl http2;
        server_name ai.jglaine.com;

        ssl_certificate /etc/nginx/ssl/fullchain.pem;
        ssl_certificate_key /etc/nginx/ssl/privkey.pem;

        # Redirect HTTP to HTTPS
        if ($scheme = http) {
            return 301 https://$server_name$request_uri;
        }

        # Open WebUI
        location /webui/ {
            proxy_pass http://localhost:3003/;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }

        # Dify
        location /dify/ {
            proxy_pass http://localhost:3001/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
```

```
    }

    # Anything LLM
    location /anything/ {
        proxy_pass http://localhost:3002/;
    }

    # n8n
    location /n8n/ {
        proxy_pass http://localhost:5678/;
    }

    # Grafana
    location /grafana/ {
        proxy_pass http://localhost:3000/;
    }

    # Flowise
    location /flowise/ {
        proxy_pass http://localhost:3004/;
    }

    # Ollama API
    location /ollama/ {
        proxy_pass http://localhost:11434/;
        limit_req zone=api_limit burst=20;
    }

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
  }
}
```

Service Selection in Script 1

[?] Select reverse proxy [1]:
1. Caddy (automatic HTTPS, easier configuration)
2. nginx (more control, manual SSL management)

→ Selected: Caddy

✓ Generating Caddyfile...

✓ Caddy will automatically obtain Let's Encrypt certificates
✓ Certificate renewal: automatic (every 30 days)

🖥️ HARDWARE REQUIREMENTS

Minimum Requirements (CPU-Only)

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ MINIMUM SPECS (Development/Testing)                                          │
├──────────────────────────────────────────────────────────────────────────────┤
│ CPU:      4 cores (x86_64)                                                   │
│ RAM:      16 GB                                                              │
│ Storage:  100 GB SSD (system + data combined)                                │
│ Network:  100 Mbps                                                           │
│ OS:       Ubuntu 22.04+ / Debian 12+ / Rocky Linux 9+                        │
│                                                                              │
│ Performance Expectations:                                                    │
│   - Ollama (7B models): 2-5 tokens/sec                                       │
│   - Concurrent users: 1-2                                                    │
│   - Context limit: 4K tokens                                                 │
└──────────────────────────────────────────────────────────────────────────────┘
```

Recommended (GPU-Accelerated)

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ RECOMMENDED SPECS (Production)                                               │
├──────────────────────────────────────────────────────────────────────────────┤
│ CPU:      8+ cores (x86_64)                                                  │
│ RAM:      32 GB                                                              │
│ GPU:      NVIDIA RTX 3060 (12GB VRAM) or better                              │
│ Storage:  System: 100 GB SSD                                                 │
│           Data (/mnt/data): 500 GB - 2 TB SSD/NVMe                           │
│ Network:  1 Gbps                                                             │
│ OS:       Ubuntu 22.04 LTS (best NVIDIA driver support)                      │
│                                                                              │
│ Performance Expectations:                                                    │
│   - Ollama (13B models): 20-40 tokens/sec                                    │
```

```
│   - Concurrent users: 5-10                                    │
│   - Context limit: 32K+ tokens                                │
│   - GPU utilization: 80-95%                                   │
└──────────────────────────────────────────────────────────────────
──┘
```

Optimal (Enterprise/Heavy Usage)

```
┌──────────────────────────────────────────────────────────────────
──┐
│ OPTIMAL SPECS (Enterprise/Team)                               │
├──────────────────────────────────────────────────────────────────
──┤
│ CPU:     16+ cores (AMD EPYC or Intel Xeon)                   │
│ RAM:     64-128 GB ECC                                        │
│ GPU:     NVIDIA A100 (40GB) or RTX 4090 (24GB)                │
│ Storage:  System: 250 GB NVMe                                 │
│          Data: 2-4 TB NVMe RAID                               │
│ Network:  10 Gbps                                             │
│ OS:      Ubuntu 22.04 LTS Server                              │
│                                                               │
│ Performance Expectations:                                     │
│   - Ollama (70B models): 30-60 tokens/sec                     │
│   - Concurrent users: 20-50                                   │
│   - Context limit: 128K tokens                                │
│   - Multi-model parallel inference                            │
└──────────────────────────────────────────────────────────────────
──┘
```

GPU vs CPU Decision Matrix

Copy table

Use Case	CPU Sufficient	GPU Recommended	GPU Required
Personal experimentation	✅		
Code generation (small projects)	✅	✅	
Document Q&A (RAG, <100 docs)	✅	✅	
Multi-user team (5+ users)		✅	✅

Large context (>8K tokens)	✅	✅
13B+ parameter models	✅	✅
Real-time response (<1s latency)	✅	✅
Production workloads	✅	✅

Storage Sizing Guide

Service	Minimum	Typical	Heavy Use
PostgreSQL	2 GB	10 GB	50 GB
Redis	500 MB	2 GB	10 GB
Ollama models	10 GB	30 GB	100 GB
MinIO objects	5 GB	50 GB	500 GB
Vector DB (Qdrant)	1 GB	10 GB	50 GB
OpenClaw workspace	1 GB	10 GB	50 GB
Logs	1 GB	5 GB	20 GB
Backups	5 GB	20 GB	100 GB
TOTAL DATA TIER	25 GB	137 GB	880 GB
Configuration	~20 MB (static)		
GRAND TOTAL	~25 GB	~140 GB	~900 GB

🧹 SCRIPT 0: CLEANUP & DEPENDENCIES

Purpose: Remove any existing AI Platform deployment, install system dependencies, detect and configure GPU support.

Expected Output

$ bash 0-cleanup-environment.sh

═══

🦞 AI PLATFORM - ENVIRONMENT CLEANUP & PREPARATION

═══

Version: v75.2.0

Host: homelab.local (192.168.1.100)
User: jglaine
Date: 2025-02-07 14:32:15 UTC

This script will:
1. Stop and remove existing AI Platform containers
2. Delete deployment directories and volumes
3. Install system dependencies (Docker, Docker Compose, etc.)
4. Detect and configure GPU support (if available)
5. Configure Docker rootless mode
6. Prepare system for fresh deployment

⚠️ WARNING: This will DELETE all existing AI Platform data!
- Docker containers and images
- Volumes in /mnt/data/ai-platform/ (if exists)
- Volumes in /home/jglaine/ai-platform/ (if exists)
- Configuration files

🔒 Credentials and backups will be preserved if found in:
- deployment/.secrets/ (backed up to ~/ai-platform-backup-TIMESTAMP/)

[?] Proceed with cleanup? [y/N]: y

═══

🔧 PHASE 1/7: DETECTING EXISTING DEPLOYMENT

═══

→ Scanning for AI Platform containers...
✓ Found 12 running containers with label 'ai-platform'

→ Scanning for deployment directories...
✓ Found: /home/jglaine/ai-platform/deployment/
✓ Found: /mnt/data/ai-platform/

→ Checking for credentials...
✓ Found: /home/jglaine/ai-platform/deployment/.secrets/.env
✓ Found: /home/jglaine/ai-platform/deployment/.secrets/api_keys.enc

→ Creating backup...
✓ Backup created: /home/jglaine/ai-platform-backup-20250207-143220/
✓ Preserved: .env, api_keys.enc, gdrive_token.json

═══

🔧 PHASE 2/7: STOPPING SERVICES

═══

→ Stopping Docker containers...
  ✓ openclaw-ai-platform   (stopped)
  ✓ open-webui-ai-platform   (stopped)
  ✓ dify-ai-platform   (stopped)
  ✓ anything-llm-ai-platform (stopped)
  ✓ ollama-ai-platform   (stopped)
  ✓ litellm-ai-platform   (stopped)
  ✓ postgres-ai-platform   (stopped)
  ✓ redis-ai-platform   (stopped)
  ✓ qdrant-ai-platform   (stopped)
  ✓ n8n-ai-platform   (stopped)
  ✓ caddy-ai-platform   (stopped)
  ✓ prometheus-ai-platform   (stopped)

→ Removing containers...
  ✓ All AI Platform containers removed (12 total)

═══

🔧 PHASE 3/7: CLEANING VOLUMES & DATA

═══

→ Removing Docker volumes...
  ✓ ai-platform_postgres_data   (deleted, 8.2 GB freed)
  ✓ ai-platform_redis_data   (deleted, 1.1 GB freed)
  ✓ ai-platform_ollama_data   (deleted, 42.3 GB freed)
  ✓ ai-platform_qdrant_data   (deleted, 3.7 GB freed)
  ✓ ai-platform_n8n_data   (deleted, 512 MB freed)

→ Removing deployment directories...
  ✓ /home/jglaine/ai-platform/deployment/ (deleted)
  ✓ /mnt/data/ai-platform/ (deleted, 55.8 GB freed)

→ Preserving scripts directory...
  ✓ /home/jglaine/ai-platform/scripts/ (kept)

Total space freed: 111.6 GB

```
═══════════════════════════════════════════════════════════════════════
═══════════════════
🔧 PHASE 4/7: DEPENDENCY CHECK & INSTALLATION
═══════════════════════════════════════════════════════════════════════
═══════════════════

Checking system dependencies...
─────────────────────────────────────────────────────────────────────
──────────────────

Core System Tools:
  ✓ curl          v7.81.0
  ✓ wget          v1.21.2
  ✓ git           v2.34.1
  ✓ jq            v1.6
  ✓ openssl       v3.0.2
  ✓ gpg           v2.2.27
  ✓ unzip         v6.0
  ✓ tar           v1.34
  ✓ awk           v1.3.4
  ✓ sed           v4.8

Docker Components:
  ✗ docker        (not installed)
  ✗ docker-compose (not installed)

→ Installing Docker Engine...
  ✓ Added Docker GPG key
  ✓ Added Docker repository
  ✓ Installing docker-ce docker-ce-cli containerd.io
  ✓ Docker installed: v27.3.1

→ Installing Docker Compose v2...
  ✓ Downloaded docker-compose v2.29.1
  ✓ Installed to /usr/local/bin/docker-compose
  ✓ Docker Compose installed: v2.29.1

→ Configuring Docker for user 'jglaine'...
  ✓ Added user to 'docker' group
  ✓ Docker rootless mode configured
  ✓ Docker daemon started
  ⚠️  Logout/login required for group changes to take effect
```

═══

🔧 PHASE 5/7: GPU DETECTION & CONFIGURATION

═══

→ Detecting NVIDIA GPU...
  ✓ NVIDIA GPU detected: NVIDIA GeForce RTX 3090
  ✓ Driver version: 535.129.03
  ✓ CUDA version: 12.2

→ Installing nvidia-docker2...
  ✓ Added NVIDIA Docker repository
  ✓ Installing nvidia-docker2 nvidia-container-toolkit
  ✓ nvidia-docker2 installed

→ Configuring Docker for GPU access...
  ✓ Updated /etc/docker/daemon.json with nvidia runtime
  ✓ Restarted Docker daemon
  ✓ Added user 'jglaine' to 'video' group

→ Testing GPU access in Docker...
  ✓ Test container launched
  ✓ GPU accessible: NVIDIA GeForce RTX 3090 (24GB VRAM)
  ✓ GPU configuration successful

→ Setting environment variables...
  ✓ GPU_AVAILABLE=true
  ✓ GPU_VENDOR=nvidia
  ✓ GPU_MODEL=GeForce RTX 3090
  ✓ GPU_VRAM=24GB
  ✓ CUDA_VERSION=12.2

═══

🔧 PHASE 6/7: SYSTEM OPTIMIZATION

═══

→ Configuring system limits...
  ✓ vm.max_map_count = 262144 (for vector DBs)
  ✓ fs.file-max = 2097152
  ✓ User limits: nofile = 65536

→ Configuring swap...
  ✓ Current swap: 8 GB
  ⚠️ Recommendation: 16 GB swap for 32 GB RAM
 [?] Adjust swap size? [y/N]: n

→ Disabling unnecessary services...
  ✓ snapd disabled (saves ~200 MB RAM)
  ✓ systemd-resolved configured for Docker DNS

═══

🔧 PHASE 7/7: FINAL VALIDATION
═══

→ Verifying installation...
  ✓ Docker: v27.3.1 (running)
  ✓ Docker Compose: v2.29.1
  ✓ User 'jglaine' in 'docker' group
  ✓ User 'jglaine' in 'video' group
  ✓ GPU: NVIDIA GeForce RTX 3090 (accessible)
  ✓ nvidia-docker2: installed and configured

→ Disk space check...
  ✓ /home/jglaine: 387 GB free (sufficient)
  ✓ /mnt/data: 1.2 TB free (excellent)

→ Network connectivity...
  ✓ Internet: connected
  ✓ DNS: functional
  ✓ Docker registry: docker.io (reachable)

═══

✅ CLEANUP & PREPARATION COMPLETE
═══

Summary:
  ✓ Existing deployment removed
  ✓ Dependencies installed
  ✓ GPU detected and configured (NVIDIA RTX 3090, 24GB VRAM)
  ✓ Docker ready (rootless mode)
  ✓ System optimized

Backup Location:
  /home/jglaine/ai-platform-backup-20250207-143220/

Next Steps:
  1. Review backup if needed
  2. *REBOOT RECOMMENDED* for group changes to take effect
  3. Run: bash 1-setup-system.sh

⚠️ IMPORTANT: A system reboot is recommended to ensure all
    group memberships and GPU configurations are active.

[?] Reboot now? [y/N]: y

→ System will reboot in 10 seconds... (Ctrl+C to cancel)
→ Rebooting...

═══
═══════════════════

Alternative Output: CPU-Only System

═══
═══════════════════
🔧 PHASE 5/7: GPU DETECTION & CONFIGURATION
═══
═══════════════════

→ Detecting NVIDIA GPU...
  ⊘ No NVIDIA GPU detected (nvidia-smi not found)

→ CPU-only mode will be used
  ✓ GPU_AVAILABLE=false
  ✓ Ollama will use CPU inference
  ⚠️ Performance: Expect 2-5 tokens/sec for 7B models

→ GPU-related tools will be skipped:
  ⊘ nvidia-docker2 (not needed)
  ⊘ CUDA toolkit (not needed)
  ⊘ nvidia-container-toolkit (not needed)

═══
═══════════════════

Script 0 Success Criteria

GPU System Requirements:

- NVIDIA GPU detected via nvidia-smi
- CUDA version identified (12.x preferred)
- nvidia-docker2 installed
- Docker daemon configured with GPU runtime
- User added to 'video' group
- GPU accessible in test container
- GPU_AVAILABLE=true in environment

CPU System Requirements:

- No GPU detected (expected behavior)
- GPU installation skipped
- GPU_AVAILABLE=false in environment
- CPU-only mode confirmed

Universal Requirements (Both GPU & CPU):

- Exit code 0 (success)
- Docker Engine installed (v27.x+)
- Docker Compose v2 installed (v2.29.x+)
- Docker rootless mode configured
- User in 'docker' group
- All core dependencies installed (curl, wget, git, jq, etc.)
- Deployment directory deleted (if existed)
- Data directory deleted (if existed)
- Scripts directory preserved
- No AI Platform containers remain
- No AI Platform volumes remain
- Credentials backed up (if existed)
- System optimized (vm.max_map_count, file limits)
- System rebooted (if user confirmed)

⚙️ SCRIPT 1: SYSTEM SETUP & CONFIGURATION

Purpose: Generate all configuration files, create directory structure, collect user preferences (domain, API keys, model selection, etc.).

Expected Output

$ bash 1-setup-system.sh

═══

🦞 AI PLATFORM - SYSTEM SETUP & CONFIGURATION

═══

Version: v75.2.0
Host: homelab.local (192.168.1.100)
User: jglaine
Date: 2025-02-07 15:10:42 UTC

This script will:
1. Detect system environment (GPU, storage, network)
2. Collect configuration preferences (domain, models, API keys)
3. Generate .env file with all variables
4. Create directory structure (CONFIG_ROOT + DATA_ROOT)
5. Generate docker-compose.yml and service configs
6. Set up reverse proxy (Caddy or nginx)
7. Configure integrations (Google Drive, Signal, etc.)

═══

🔧 PHASE 0/14: ENVIRONMENT DETECTION

═══

→ Detecting system capabilities...
  ✓ OS: Ubuntu 22.04.3 LTS (Jammy Jellyfish)
  ✓ Kernel: 6.5.0-35-generic
  ✓ Architecture: x86_64
  ✓ CPU: AMD Ryzen 9 5950X (32 threads)
  ✓ RAM: 64 GB (60 GB available)
  ✓ GPU: NVIDIA GeForce RTX 3090 (24GB VRAM, CUDA 12.2)
  ✓ Docker: v27.3.1 (rootless mode enabled)
  ✓ Docker Compose: v2.29.1

→ Detecting storage locations...
  ✓ /mnt/data exists (1.2 TB free, writable)
  ✓ /home/jglaine (387 GB free)

[?] Store growing data on /mnt/data? [Y/n]: Y

→ Storage configuration:

✓ CONFIG_ROOT: /home/jglaine/ai-platform
✓ DATA_ROOT: /mnt/data/ai-platform

═══

🔧 PHASE 1/14: NETWORK CONFIGURATION

═══

[?] Do you have a domain name? [y/N]: y

[?] Enter your domain (e.g., ai.jglaine.com): ai.jglaine.com

→ Validating domain...
✓ Domain resolves to: 192.168.1.100
✓ DNS configuration correct

[?] Enable HTTPS with Let's Encrypt? [Y/n]: Y

[?] Email for Let's Encrypt notifications: admin@jglaine.com

→ Domain configuration saved:
✓ DOMAIN=ai.jglaine.com
✓ HTTPS_ENABLED=true
✓ LETSENCRYPT_EMAIL=admin@jglaine.com

═══

🔧 PHASE 2/14: REVERSE PROXY SELECTION

═══

[?] Select reverse proxy [1]:
1. Caddy (automatic HTTPS, simpler config)
2. nginx (more control, manual cert management)

→ Selection: 1

→ Reverse proxy configuration:
✓ PROXY_TYPE=caddy
✓ Automatic HTTPS: enabled
✓ Certificate renewal: automatic (every 30 days)

═══

🔧 PHASE 3/14: TAILSCALE CONFIGURATION

═══

→ Tailscale is required for OpenClaw access

[?] Tailscale auth key: tskey-auth-xxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxx

→ Validating Tailscale key...
  ✓ Key valid
  ✓ Network: homelab-network

→ Tailscale configuration:
  ✓ TAILSCALE_AUTHKEY=tskey-auth-xxxxxxxxxxxx-***
  ✓ OpenClaw will be accessible at: http://<tailscale-ip>:18789
  ⚠️ Tailscale IP will be assigned during deployment (Phase 2-deploy)

═══

🔧 PHASE 4/14: DATABASE CONFIGURATION

═══

→ Generating PostgreSQL credentials...
  ✓ POSTGRES_USER=ai_platform
  ✓ POSTGRES_PASSWORD=<generated 32-char secure password>
  ✓ POSTGRES_DB=ai_platform_db

→ Generating Redis configuration...
  ✓ REDIS_PASSWORD=<generated 32-char secure password>

═══

🔧 PHASE 5/14: VECTOR DATABASE SELECTION

═══

[?] Select vector database [1]:
    1. Qdrant (recommended, easiest setup)
    2. Weaviate (advanced features, semantic search)
    3. Chroma (lightweight, good for small datasets)

→ Selection: 1

→ Vector database configuration:
  ✓ VECTOR_DB=qdrant
  ✓ QDRANT_PORT=6333
  ✓ QDRANT_GRPC_PORT=6334
  ✓ QDRANT_API_KEY=<generated>

═══

🔧 PHASE 6/14: OLLAMA MODEL SELECTION
═══

[?] Select Ollama models to download (space to select, enter to confirm):
    [x] llama3.2:latest (8B, general purpose) - 4.7 GB
    [ ] llama3.2:3b (3B, fast inference) - 2.0 GB
    [x] mistral:latest (7B, coding & reasoning) - 4.1 GB
    [ ] codellama:13b (13B, specialized coding) - 7.4 GB
    [ ] gemma2:9b (9B, Google) - 5.4 GB
    [x] qwen2.5:7b (7B, multilingual) - 4.7 GB

→ Selected models (will be downloaded during deployment):
  ✓ llama3.2:latest (4.7 GB)
  ✓ mistral:latest (4.1 GB)
  ✓ qwen2.5:7b (4.7 GB)
  Total download size: ~13.5 GB

→ Ollama configuration:
  ✓ OLLAMA_MODELS=llama3.2:latest,mistral:latest,qwen2.5:7b
  ✓ OLLAMA_NUM_PARALLEL=3 (based on 24GB VRAM)
  ✓ OLLAMA_GPU_LAYERS=999 (full GPU offload)

═══

🔧 PHASE 7/14: EXTERNAL API CONFIGURATION (Optional)
═══

LiteLLM can route complex queries to external APIs.
Leave blank to skip (local Ollama only).

[?] OpenAI API Key (for GPT-4, GPT-4o): sk-proj-xxxxxxxxxxxxxxxxxxxx

→ Validating OpenAI key...
  ✓ Key valid
  ✓ Available models: gpt-4-turbo, gpt-4o, gpt-3.5-turbo

[?] Anthropic API Key (for Claude):

→ Skipped

[?] Google Gemini API Key:

→ Skipped

[?] Groq API Key (fast inference): gsk_xxxxxxxxxxxxxxxxxxxx

→ Validating Groq key...
  ✓ Key valid
  ✓ Available models: llama-3.1-70b, mixtral-8x7b

[?] DeepSeek API Key:

→ Skipped

→ External API configuration:
  ✓ OPENAI_API_KEY=sk-proj-***
  ✓ GROQ_API_KEY=gsk_***
  ⊘ ANTHROPIC_API_KEY (not configured)
  ⊘ GOOGLE_GEMINI_API_KEY (not configured)
  ⊘ DEEPSEEK_API_KEY (not configured)

═══

🔧 PHASE 8/14: LITELLM ROUTING CONFIGURATION

═══

→ Configuring LiteLLM routing logic...

  Simple Queries (Ollama local):
    ✓ General chat
    ✓ Code completion
    ✓ Fast responses
    → Route to: ollama:11434

  Complex Queries (External APIs):

✓ Long context (>8K tokens)
✓ Advanced reasoning
✓ Production-critical
→ Primary: OpenAI GPT-4o
→ Fallback: Groq llama-3.1-70b
→ Final fallback: Ollama mistral:latest

→ Generating litellm_config.yaml...
✓ Configuration file created
✓ 3 model routes configured
✓ Automatic fallback enabled

═══

🔧 PHASE 9/14: GOOGLE DRIVE INTEGRATION (Optional)
═══

[?] Enable Google Drive sync for Anything LLM? [y/N]: y

→ Google Drive setup instructions:
1. Go to: https://console.cloud.google.com/apis/credentials
2. Create OAuth 2.0 Client ID (Desktop app)
3. Download credentials.json

[?] Path to credentials.json: ~/Downloads/credentials.json

→ Validating credentials...
✓ credentials.json valid
✓ Client ID: 123456789-xxxxxxxxx.apps.googleusercontent.com

→ Initiating OAuth flow...
Please visit this URL to authorize:

https://accounts.google.com/o/oauth2/auth?client_id=...

[?] Paste authorization code: 4/0AeaYSHBxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

→ Exchanging code for token...
✓ Token received
✓ Refresh token stored in .secrets/gdrive_token.json
✓ Access expires: 2025-02-07 16:10:42 UTC (auto-refresh enabled)

→ Google Drive configuration:

✓ GDRIVE_ENABLED=true
✓ GDRIVE_SYNC_INTERVAL=3600 (1 hour)
✓ GDRIVE_FOLDER_ID=root (entire Drive)
✓ Sync destination: /mnt/data/ai-platform/gdrive/

═══

🔧 PHASE 10/14: SIGNAL NOTIFICATIONS (Optional)
═══

[?] Enable Signal notifications? [y/N]: n

→ Skipped

═══

🔧 PHASE 11/14: OPENCLAW CONFIGURATION
═══

→ Configuring OpenClaw (AI coding agent)...

✓ OPENCLAW_PORT=18789
✓ OPENCLAW_WORKSPACE=/mnt/data/ai-platform/openclaw
✓ Access method: Tailscale VPN only (not proxied)

[?] OpenClaw model preference [1]:
1. Use local Ollama models (mistral:latest)
2. Use external API (OpenAI GPT-4o)

→ Selection: 1

→ OpenClaw configuration:
✓ OPENCLAW_MODEL=ollama/mistral:latest
✓ OPENCLAW_API_URL=http://ollama:11434
✓ OPENCLAW_MAX_CONTEXT=32768
✓ OPENCLAW_TEMPERATURE=0.2 (deterministic for code)
✓ OPENCLAW_AUTO_SAVE=true
✓ OPENCLAW_CONVERSATION_HISTORY=true

→ OpenClaw will be accessible at:
http://<tailscale-ip>:18789 (once Tailscale assigns IP in Script 2)

═══

🔧 PHASE 12/14: OPTIONAL SERVICES SELECTION

═══

[?] Enable optional services? (space to select, enter to confirm):
  [x] Flowise (visual LLM flow builder) - Port 3004
  [ ] Additional Ollama models (select later)
  [ ] Custom model fine-tuning (requires advanced setup)

→ Selected optional services:
  ✓ Flowise will be deployed
  ✓ Accessible at: https://ai.jglaine.com/flowise

═══

🔧 PHASE 13/14: DIRECTORY STRUCTURE CREATION

═══

→ Creating CONFIG_ROOT structure...
  ✓ /home/jglaine/ai-platform/deployment/stack/
  ✓ /home/jglaine/ai-platform/deployment/configs/
  ✓ /home/jglaine/ai-platform/deployment/configs/openclaw/
  ✓ /home/jglaine/ai-platform/deployment/.secrets/

→ Creating DATA_ROOT structure...
  ✓ /mnt/data/ai-platform/volumes/
  ✓ /mnt/data/ai-platform/volumes/postgres/
  ✓ /mnt/data/ai-platform/volumes/redis/
  ✓ /mnt/data/ai-platform/volumes/ollama/
  ✓ /mnt/data/ai-platform/volumes/minio/
  ✓ /mnt/data/ai-platform/volumes/qdrant/
  ✓ /mnt/data/ai-platform/volumes/n8n/
  ✓ /mnt/data/ai-platform/volumes/prometheus/
  ✓ /mnt/data/ai-platform/openclaw/
  ✓ /mnt/data/ai-platform/openclaw/projects/
  ✓ /mnt/data/ai-platform/openclaw/conversations/
  ✓ /mnt/data/ai-platform/openclaw/artifacts/
  ✓ /mnt/data/ai-platform/logs/
  ✓ /mnt/data/ai-platform/logs/caddy/
  ✓ /mnt/data/ai-platform/logs/ollama/
  ✓ /mnt/data/ai-platform/logs/litellm/

✓ /mnt/data/ai-platform/backups/
✓ /mnt/data/ai-platform/backups/postgres/
✓ /mnt/data/ai-platform/backups/vector_db/
✓ /mnt/data/ai-platform/gdrive/

→ Setting permissions...
✓ chown -R jglaine:jglaine /home/jglaine/ai-platform/
✓ chown -R jglaine:jglaine /mnt/data/ai-platform/
✓ chmod 700 /home/jglaine/ai-platform/deployment/.secrets/

═══

🔧 PHASE 14/14: CONFIGURATION FILE GENERATION

═══

→ Generating .env file...
✓ /home/jglaine/ai-platform/deployment/.secrets/.env
✓ 87 environment variables configured
✓ File encrypted with GPG

→ Generating docker-compose.yml...
✓ /home/jglaine/ai-platform/deployment/stack/docker-compose.yml
✓ 17 services configured
✓ All volume binds use DATA_ROOT paths

→ Generating docker-compose.override.yml (GPU)...
✓ /home/jglaine/ai-platform/deployment/stack/docker-compose.override.yml
✓ GPU configuration for Ollama service
✓ NVIDIA runtime enabled

→ Generating Caddyfile...
✓ /home/jglaine/ai-platform/deployment/configs/Caddyfile
✓ 14 service routes configured
✓ HTTPS enabled with Let's Encrypt
✓ OpenClaw excluded (Tailscale only)

→ Generating litellm_config.yaml...
✓ /home/jglaine/ai-platform/deployment/configs/litellm_config.yaml
✓ Local models: llama3.2, mistral, qwen2.5
✓ External APIs: OpenAI (GPT-4o), Groq (llama-3.1-70b)
✓ Routing logic: simple → local, complex → external

→ Generating prometheus.yml...

✓ /home/jglaine/ai-platform/deployment/configs/prometheus.yml
✓ Scrape targets: Ollama, LiteLLM, Caddy, PostgreSQL
✓ Retention: 30 days

→ Generating grafana-datasources.yml...
✓ /home/jglaine/ai-platform/deployment/configs/grafana-datasources.yml
✓ Datasource: Prometheus (auto-configured)

→ Generating openclaw/config.json...
✓ /home/jglaine/ai-platform/deployment/configs/openclaw/config.json
✓ Model: ollama/mistral:latest
✓ Workspace: /workspace (mapped to /mnt/data/ai-platform/openclaw)
✓ Auto-save enabled

→ Generating API keys file...
✓ /home/jglaine/ai-platform/deployment/.secrets/api_keys.enc
✓ Encrypted with GPG (passphrase protected)

→ Storing Google Drive token...
✓ /home/jglaine/ai-platform/deployment/.secrets/gdrive_token.json
✓ Encrypted with GPG

═══

✅ SYSTEM SETUP COMPLETE

═══

Configuration Summary:

───

System:
✓ CPU: AMD Ryzen 9 5950X (32 threads)
✓ RAM: 64 GB
✓ GPU: NVIDIA GeForce RTX 3090 (24GB VRAM)
✓ Storage (Config): /home/jglaine/ai-platform (387 GB free)
✓ Storage (Data): /mnt/data/ai-platform (1.2 TB free)

Network:
✓ Domain: ai.jglaine.com
✓ HTTPS: Enabled (Let's Encrypt)
✓ Reverse Proxy: Caddy
✓ Tailscale: Configured (OpenClaw access)

Services (17 total):
- ✓ Tailscale VPN
- ✓ Caddy (reverse proxy)
- ✓ PostgreSQL (database)
- ✓ Redis (cache)
- ✓ MinIO (object storage)
- ✓ Prometheus (metrics)
- ✓ Grafana (dashboards)
- ✓ Ollama (local LLM)
- ✓ LiteLLM (routing)
- ✓ OpenClaw (AI coding) - Tailscale only
- ✓ Dify (LLM apps)
- ✓ Anything LLM (document chat)
- ✓ Open WebUI (Ollama interface)
- ✓ Qdrant (vector DB)
- ✓ n8n (workflows)
- ✓ Flowise (LLM flows)
- ✓ Signal-CLI (notifications) - DISABLED

Models:

Local (Ollama):
- ✓ llama3.2:latest (8B, general)
- ✓ mistral:latest (7B, coding)
- ✓ qwen2.5:7b (7B, multilingual)

External APIs:
- ✓ OpenAI GPT-4o (complex queries)
- ✓ Groq llama-3.1-70b (fast inference)

Integrations:
- ✓ Google Drive sync: ENABLED (1 hour interval)
- ✓ Signal notifications: DISABLED

Access URLs (after deployment):
- ✓ Open WebUI: https://ai.jglaine.com/webui
- ✓ Dify: https://ai.jglaine.com/dify
- ✓ Anything LLM: https://ai.jglaine.com/anything
- ✓ n8n: https://ai.jglaine.com/n8n
- ✓ Flowise: https://ai.jglaine.com/flowise
- ✓ Grafana: https://ai.jglaine.com/grafana
- ✓ MinIO Console: https://ai.jglaine.com/minio
- ✓ OpenClaw: http://<tailscale-ip>:18789 (not proxied)

Files Generated:

✓ deployment/.secrets/.env (87 variables, encrypted)
✓ deployment/.secrets/api_keys.enc (encrypted)
✓ deployment/.secrets/gdrive_token.json (encrypted)
✓ deployment/stack/docker-compose.yml (17 services)
✓ deployment/stack/docker-compose.override.yml (GPU config)
✓ deployment/configs/Caddyfile (14 routes)
✓ deployment/configs/litellm_config.yaml
✓ deployment/configs/prometheus.yml
✓ deployment/configs/grafana-datasources.yml
✓ deployment/configs/openclaw/config.json

Security:
✓ All credentials encrypted with GPG
✓ .secrets/ directory: chmod 700
✓ Strong random passwords generated (32 characters)
✓ Tailscale VPN for OpenClaw (not exposed to public)

Next Steps:
1. Review configuration: cat deployment/.secrets/.env
2. Deploy services: bash 2-deploy-services.sh
3. Monitor deployment progress (takes 5-15 minutes)
4. After deployment, Tailscale will assign IP for OpenClaw

Disk Space Estimate:
Initial deployment: ~25 GB
After model downloads: ~40 GB
Expected growth (1 month): ~60-100 GB

═══

⚠️ IMPORTANT NOTES

═══

1. OpenClaw Access:
   - NOT accessible via domain (security by design)
   - ONLY via Tailscale VPN: http://<tailscale-ip>:18789
   - Tailscale IP will be shown in Script 2 output
   - Use: tailscale status (to find IP after deployment)

2. Credentials:
   - All passwords stored in: deployment/.secrets/.env
   - File is GPG encrypted (passphrase required to decrypt)
   - BACKUP THIS FILE (and remember GPG passphrase)

3. Google Drive Sync:
   - First sync will start 1 hour after deployment
   - Synced files: /mnt/data/ai-platform/gdrive/
   - Ingested by Anything LLM automatically

4. GPU Configuration:
   - Ollama will use full GPU (24GB VRAM)
   - Parallel model loading: 3 models max
   - If GPU memory issues occur, reduce OLLAMA_NUM_PARALLEL in .env

5. External API Costs:
   - OpenAI GPT-4o: ~$0.01/1K tokens (only for complex queries)
   - Groq: Free tier (limited rate limits)
   - LiteLLM will prefer local models to minimize costs

═══

🚀 SCRIPT 2: SERVICE DEPLOYMENT

Purpose: Launch all Docker containers, download Ollama models, initialize databases, start Tailscale, and validate service health.

Expected Output

$ bash 2-deploy-services.sh

═══

🦞 AI PLATFORM - SERVICE DEPLOYMENT

═══

Version: v75.2.0
Host: homelab.local (192.168.1.100)
User: jglaine
Date: 2025-02-07 15:45:18 UTC

This script will:
  1. Validate configuration from Script 1
  2. Pull Docker images
  3. Start infrastructure services (PostgreSQL, Redis, MinIO)

4. Initialize Tailscale VPN
5. Start Ollama and download models
6. Deploy AI applications
7. Start reverse proxy (Caddy)
8. Run health checks on all services
9. Display access URLs and Tailscale IP

═══

🔧 PHASE 1/10: PRE-DEPLOYMENT VALIDATION

═══

→ Checking prerequisites...
  ✓ Docker daemon running
  ✓ Docker Compose v2.29.1
  ✓ User 'jglaine' in 'docker' group
  ✓ GPU available (NVIDIA RTX 3090)

→ Validating configuration files...
  ✓ deployment/.secrets/.env exists
  ✓ deployment/stack/docker-compose.yml exists
  ✓ deployment/stack/docker-compose.override.yml exists (GPU config)
  ✓ deployment/configs/Caddyfile exists
  ✓ deployment/configs/litellm_config.yaml exists
  ✓ deployment/configs/openclaw/config.json exists

→ Decrypting .env file...
  [?] Enter GPG passphrase: ************
  ✓ .env decrypted successfully

→ Loading environment variables...
  ✓ 87 variables loaded
  ✓ CONFIG_ROOT=/home/jglaine/ai-platform
  ✓ DATA_ROOT=/mnt/data/ai-platform
  ✓ GPU_AVAILABLE=true

→ Validating directory structure...
  ✓ CONFIG_ROOT exists (20 MB used)
  ✓ DATA_ROOT exists (1.2 TB free)
  ✓ All volume directories exist

→ Checking port availability...
  ✓ Port 80 (Caddy) - available

✓ Port 443 (Caddy) - available
✓ Port 5432 (PostgreSQL) - available
✓ Port 6379 (Redis) - available
✓ Port 11434 (Ollama) - available
✓ Port 18789 (OpenClaw) - available
✓ Port 3000 (Grafana) - available
✓ Port 3001 (Dify) - available
✓ Port 3002 (Anything LLM) - available
✓ Port 3003 (Open WebUI) - available
✓ Port 3004 (Flowise) - available
✓ Port 5678 (n8n) - available
✓ Port 6333 (Qdrant) - available
✓ Port 9000 (MinIO API) - available
✓ Port 9001 (MinIO Console) - available

→ Disk space check...
✓ /mnt/data: 1.2 TB free (sufficient for deployment + growth)

═══

🔧 PHASE 2/10: DOCKER IMAGE PULL

═══

→ Pulling Docker images (this may take 5-10 minutes)...

[1/17] postgres:16-alpine
✓ Digest: sha256:abc123...
✓ Size: 238 MB
✓ Status: Downloaded newer image

[2/17] redis:7-alpine
✓ Digest: sha256:def456...
✓ Size: 41 MB
✓ Status: Downloaded newer image

[3/17] minio/minio:latest
✓ Digest: sha256:ghi789...
✓ Size: 267 MB
✓ Status: Downloaded newer image

[4/17] ollama/ollama:latest
✓ Digest: sha256:jkl012...
✓ Size: 1.2 GB

✓ Status: Downloaded newer image

[5/17] ghcr.io/berriai/litellm:latest
✓ Digest: sha256:mno345...
✓ Size: 523 MB
✓ Status: Downloaded newer image

[6/17] tailscale/tailscale:latest
✓ Digest: sha256:pqr678...
✓ Size: 187 MB
✓ Status: Downloaded newer image

[7/17] openclaw/openclaw:latest
✓ Digest: sha256:stu901...
✓ Size: 892 MB
✓ Status: Downloaded newer image

[8/17] langgenius/dify-api:latest
✓ Digest: sha256:vwx234...
✓ Size: 1.1 GB
✓ Status: Downloaded newer image

[9/17] langgenius/dify-web:latest
✓ Digest: sha256:yza567...
✓ Size: 456 MB
✓ Status: Downloaded newer image

[10/17] anything-llm/anything-llm:latest
✓ Digest: sha256:bcd890...
✓ Size: 743 MB
✓ Status: Downloaded newer image

[11/17] ghcr.io/open-webui/open-webui:main
✓ Digest: sha256:efg123...
✓ Size: 612 MB
✓ Status: Downloaded newer image

[12/17] qdrant/qdrant:latest
✓ Digest: sha256:hij456...
✓ Size: 234 MB
✓ Status: Downloaded newer image

[13/17] n8nio/n8n:latest
✓ Digest: sha256:klm789...

✓ Size: 567 MB
✓ Status: Downloaded newer image

[14/17] flowiseai/flowise:latest
✓ Digest: sha256:nop012...
✓ Size: 489 MB
✓ Status: Downloaded newer image

[15/17] caddy:2-alpine
✓ Digest: sha256:qrs345...
✓ Size: 47 MB
✓ Status: Downloaded newer image

[16/17] prom/prometheus:latest
✓ Digest: sha256:tuv678...
✓ Size: 234 MB
✓ Status: Downloaded newer image

[17/17] grafana/grafana:latest
✓ Digest: sha256:wxy901...
✓ Size: 389 MB
✓ Status: Downloaded newer image

→ Image pull summary:
✓ Total images: 17
✓ Total size: 8.2 GB
✓ All images pulled successfully

═══

🔧 PHASE 3/10: INFRASTRUCTURE SERVICES STARTUP

═══

→ Starting PostgreSQL...
✓ Container: postgres-ai-platform
✓ Status: Started
✓ Health check: Waiting... (max 30s)
✓ Health check: Healthy (took 8s)
✓ Port 5432: Listening

→ Initializing PostgreSQL databases...
✓ Database: ai_platform_db (created)
✓ Database: dify_db (created)

  ✓ Database: anything_llm_db (created)
  ✓ Database: n8n_db (created)
  ✓ User: ai_platform (granted all privileges)

→ Starting Redis...
  ✓ Container: redis-ai-platform
  ✓ Status: Started
  ✓ Health check: Healthy (took 3s)
  ✓ Port 6379: Listening
  ✓ Auth: Password protected

→ Starting MinIO...
  ✓ Container: minio-ai-platform
  ✓ Status: Started
  ✓ Health check: Healthy (took 5s)
  ✓ Port 9000 (API): Listening
  ✓ Port 9001 (Console): Listening

→ Initializing MinIO buckets...
  ✓ Bucket: dify-storage (created)
  ✓ Bucket: anything-llm-storage (created)
  ✓ Bucket: n8n-storage (created)
  ✓ Access policy: Public read

→ Starting Qdrant...
  ✓ Container: qdrant-ai-platform
  ✓ Status: Started
  ✓ Health check: Healthy (took 4s)
  ✓ Port 6333 (HTTP): Listening
  ✓ Port 6334 (gRPC): Listening

→ Starting Prometheus...
  ✓ Container: prometheus-ai-platform
  ✓ Status: Started
  ✓ Config: /home/jglaine/ai-platform/deployment/configs/prometheus.yml
  ✓ Health check: Healthy (took 6s)
  ✓ Port 9090: Listening

→ Infrastructure services summary:
  ✓ PostgreSQL: Running (8s to healthy)
  ✓ Redis: Running (3s to healthy)
  ✓ MinIO: Running (5s to healthy)
  ✓ Qdrant: Running (4s to healthy)
  ✓ Prometheus: Running (6s to healthy)

✓ Total startup time: 26 seconds

═══

🔧 PHASE 4/10: TAILSCALE VPN INITIALIZATION

═══

→ Starting Tailscale container...
✓ Container: tailscale-ai-platform
✓ Status: Started
✓ Auth key: tskey-auth-xxxxxxxxxxxx-***

→ Connecting to Tailscale network...
✓ Authenticating...
✓ Connected to network: homelab-network
✓ Device name: homelab-aiplatform
✓ Tailscale IP assigned: 100.64.0.5

→ Verifying Tailscale connectivity...
✓ Ping tailscale.com: OK
✓ Peer visibility: 3 devices visible

→ Tailscale configuration:
✓ Network: homelab-network
✓ Device IP: 100.64.0.5
✓ Device name: homelab-aiplatform
✓ OpenClaw will be accessible at: http://100.64.0.5:18789

═══

🔧 PHASE 5/10: OLLAMA STARTUP & MODEL DOWNLOAD

═══

→ Starting Ollama service...
✓ Container: ollama-ai-platform
✓ Status: Started
✓ GPU: NVIDIA RTX 3090 detected
✓ VRAM: 24 GB available
✓ GPU layers: Full offload (999 layers)
✓ Health check: Healthy (took 7s)
✓ Port 11434: Listening

→ Downloading Ollama models (this will take 10-20 minutes)...

[1/3] llama3.2:latest
  ✓ Size: 4.7 GB
  ✓ Downloading... ████████████████████████████ 100%
  ✓ Verifying checksum... OK
  ✓ Loading into VRAM... OK (uses 5.2 GB VRAM)
  ✓ Status: Ready
  ✓ Time: 4m 32s

[2/3] mistral:latest
  ✓ Size: 4.1 GB
  ✓ Downloading... ████████████████████████████ 100%
  ✓ Verifying checksum... OK
  ✓ Loading into VRAM... OK (uses 4.6 GB VRAM)
  ✓ Status: Ready
  ✓ Time: 3m 58s

[3/3] qwen2.5:7b
  ✓ Size: 4.7 GB
  ✓ Downloading... ████████████████████████████ 100%
  ✓ Verifying checksum... OK
  ✓ Loading into VRAM... OK (uses 5.1 GB VRAM)
  ✓ Status: Ready
  ✓ Time: 4m 21s

→ Testing Ollama inference...
  ✓ Model: llama3.2:latest
  ✓ Prompt: "Hello, world!"
  ✓ Response: "Hello! How can I assist you today?"
  ✓ Inference speed: 47 tokens/sec (GPU accelerated)
  ✓ Latency: 38ms (first token)

→ Ollama summary:
  ✓ Models downloaded: 3
  ✓ Total size: 13.5 GB
  ✓ VRAM usage: 14.9 GB / 24 GB (62%)
  ✓ Available for parallel: 3 models max
  ✓ Total download time: 12m 51s

═══

🔧 PHASE 6/10: LITELLM ROUTING LAYER

═══

→ Starting LiteLLM...
  ✓ Container: litellm-ai-platform
  ✓ Status: Started
  ✓ Config: /home/jglaine/ai-platform/deployment/configs/litellm_config.yaml
  ✓ Health check: Healthy (took 5s)
  ✓ Port 4000: Listening

→ Validating LiteLLM configuration...
  ✓ Local models (Ollama): 3 configured
    - llama3.2:latest
    - mistral:latest
    - qwen2.5:7b
  ✓ External APIs: 2 configured
    - OpenAI GPT-4o (complex queries)
    - Groq llama-3.1-70b (fallback)

→ Testing LiteLLM routing...

  Test 1: Simple query (should route to local Ollama)
    ✓ Prompt: "What is 2+2?"
    ✓ Routed to: ollama/llama3.2:latest
    ✓ Response time: 156ms
    ✓ Status: PASS

  Test 2: Complex query (should route to OpenAI)
    ✓ Prompt: "Explain quantum computing in detail..."
    ✓ Routed to: openai/gpt-4o
    ✓ Response time: 2.3s
    ✓ Status: PASS

  Test 3: Fallback test (simulate OpenAI failure)
    ✓ Simulating OpenAI timeout...
    ✓ Fallback to: groq/llama-3.1-70b
    ✓ Response time: 890ms
    ✓ Status: PASS

→ LiteLLM routing summary:
  ✓ Routing logic: Working correctly
  ✓ Local models: Reachable
  ✓ External APIs: Reachable
  ✓ Fallback chain: Functional

═══

🔧 PHASE 7/10: AI APPLICATION SERVICES

═══

→ Starting Open WebUI...
  ✓ Container: open-webui-ai-platform
  ✓ Status: Started
  ✓ Connected to: Ollama (http://ollama:11434)
  ✓ Health check: Healthy (took 8s)
  ✓ Port 3003: Listening

→ Starting Dify...
  ✓ Container: dify-api-ai-platform
  ✓ Container: dify-web-ai-platform
  ✓ Status: Started (both)
  ✓ Database: dify_db (connected)
  ✓ Vector DB: Qdrant (connected)
  ✓ Object storage: MinIO (connected)
  ✓ Health check: Healthy (took 12s)
  ✓ Port 3001: Listening

→ Initializing Dify database schema...
  ✓ Running migrations... (15 migrations applied)
  ✓ Creating default workspace...
  ✓ Admin user: admin@ai-platform.local (auto-generated)
  ✓ Admin password: <shown in logs, please change on first login>

→ Starting Anything LLM...
  ✓ Container: anything-llm-ai-platform
  ✓ Status: Started
  ✓ Database: anything_llm_db (connected)
  ✓ Vector DB: Qdrant (connected)
  ✓ LLM Provider: LiteLLM (connected)
  ✓ Health check: Healthy (took 9s)
  ✓ Port 3002: Listening

→ Starting OpenClaw (AI Coding Agent)...
  ✓ Container: openclaw-ai-platform
  ✓ Status: Started
  ✓ Workspace: /mnt/data/ai-platform/openclaw (mounted)
  ✓ Model: ollama/mistral:latest

✓ Ollama connection: OK
✓ Health check: Healthy (took 6s)
✓ Port 18789: Listening (Tailscale only)
✓ Access URL: http://100.64.0.5:18789

→ Starting n8n...
✓ Container: n8n-ai-platform
✓ Status: Started
✓ Database: n8n_db (connected)
✓ Health check: Healthy (took 10s)
✓ Port 5678: Listening

→ Starting Flowise...
✓ Container: flowise-ai-platform
✓ Status: Started
✓ Connected to: LiteLLM
✓ Vector DB: Qdrant (connected)
✓ Health check: Healthy (took 7s)
✓ Port 3004: Listening

→ AI applications summary:
✓ Open WebUI: Running (port 3003)
✓ Dify: Running (port 3001)
✓ Anything LLM: Running (port 3002)
✓ OpenClaw: Running (port 18789, Tailscale only)
✓ n8n: Running (port 5678)
✓ Flowise: Running (port 3004)

═══

🔧 PHASE 8/10: MONITORING & VISUALIZATION

═══

→ Starting Grafana...
✓ Container: grafana-ai-platform
✓ Status: Started
✓ Datasource: Prometheus (auto-configured)
✓ Health check: Healthy (took 7s)
✓ Port 3000: Listening

→ Importing Grafana dashboards...
✓ Dashboard: Ollama Performance (ID: 1)
✓ Dashboard: LiteLLM Routing (ID: 2)

✓ Dashboard: System Resources (ID: 3)
✓ Dashboard: Application Health (ID: 4)
✓ Default home dashboard: Ollama Performance

→ Grafana configuration:
✓ Admin user: admin
✓ Admin password: <stored in .env>
✓ Anonymous access: Disabled
✓ Default org: AI Platform

═══

🔧 PHASE 9/10: REVERSE PROXY & SSL

═══

→ Starting Caddy...
✓ Container: caddy-ai-platform
✓ Status: Started
✓ Config: /home/jglaine/ai-platform/deployment/configs/Caddyfile
✓ Domain: ai.jglaine.com
✓ Health check: Healthy (took 6s)
✓ Port 80 (HTTP): Listening
✓ Port 443 (HTTPS): Listening

→ Requesting Let's Encrypt certificate...
✓ Domain validation: ai.jglaine.com (DNS challenge)
✓ Certificate issued: Let's Encrypt Authority X3
✓ Valid until: 2025-05-08 15:52:34 UTC (90 days)
✓ Auto-renewal: Configured (every 30 days)

→ Testing reverse proxy routes...

[1/14] Open WebUI: https://ai.jglaine.com/webui
✓ Proxy status: OK
✓ SSL/TLS: Valid certificate
✓ Response time: 87ms
✓ Status: 200 OK

[2/14] Dify: https://ai.jglaine.com/dify
✓ Proxy status: OK
✓ SSL/TLS: Valid certificate
✓ Response time: 102ms
✓ Status: 200 OK

[3/14] Anything LLM: https://ai.jglaine.com/anything
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 76ms
  ✓ Status: 200 OK

[4/14] n8n: https://ai.jglaine.com/n8n
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 94ms
  ✓ Status: 200 OK

[5/14] Flowise: https://ai.jglaine.com/flowise
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 81ms
  ✓ Status: 200 OK

[6/14] Grafana: https://ai.jglaine.com/grafana
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 68ms
  ✓ Status: 200 OK

[7/14] MinIO Console: https://ai.jglaine.com/minio
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 73ms
  ✓ Status: 200 OK

[8/14] Prometheus: https://ai.jglaine.com/prometheus
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 59ms
  ✓ Status: 200 OK

[9/14] Qdrant Dashboard: https://ai.jglaine.com/qdrant
  ✓ Proxy status: OK
  ✓ SSL/TLS: Valid certificate
  ✓ Response time: 65ms
  ✓ Status: 200 OK

[10/14] Ollama API: https://ai.jglaine.com/ollama

✓ Proxy status: OK
✓ SSL/TLS: Valid certificate
✓ Response time: 71ms
✓ Status: 200 OK

[11/14] LiteLLM API: https://ai.jglaine.com/litellm
✓ Proxy status: OK
✓ SSL/TLS: Valid certificate
✓ Response time: 62ms
✓ Status: 200 OK

[12/14] Health Check Endpoint: https://ai.jglaine.com/health
✓ Proxy status: OK
✓ SSL/TLS: Valid certificate
✓ Response: {"status":"healthy","services":17}
✓ Status: 200 OK

[13/14] HTTP → HTTPS Redirect
✓ http://ai.jglaine.com → https://ai.jglaine.com
✓ Status: 301 Moved Permanently
✓ Redirect working correctly

[14/14] OpenClaw (should NOT be proxied)
✓ https://ai.jglaine.com/openclaw → 404 Not Found (expected)
✓ http://100.64.0.5:18789 → 200 OK (Tailscale only)
✓ Isolation confirmed: OpenClaw not exposed via domain

→ Reverse proxy summary:
✓ All routes functional (14/14)
✓ SSL/TLS enabled
✓ HTTP → HTTPS redirect: Working
✓ OpenClaw isolation: Confirmed (Tailscale only)

═══

🔧 PHASE 10/10: FINAL HEALTH CHECKS & INTEGRATION TESTS

═══

→ Running comprehensive health checks...

Infrastructure Services:
✓ PostgreSQL: Healthy (query test: 3ms)
✓ Redis: Healthy (ping test: 1ms)

✓ MinIO: Healthy (bucket list: OK)
✓ Qdrant: Healthy (collection list: OK)
✓ Prometheus: Healthy (scraping 12 targets)

AI Services:
✓ Ollama: Healthy (3 models loaded, 14.9GB VRAM used)
✓ LiteLLM: Healthy (routing functional)
✓ Open WebUI: Healthy (connected to Ollama)
✓ Dify: Healthy (all dependencies connected)
✓ Anything LLM: Healthy (vector DB connected)
✓ OpenClaw: Healthy (Tailscale accessible)
✓ n8n: Healthy (workflow engine ready)
✓ Flowise: Healthy (flow builder ready)

Monitoring:
✓ Grafana: Healthy (dashboards loaded)
✓ Prometheus: Healthy (metrics collecting)

Network:
✓ Caddy: Healthy (SSL active, 14 routes)
✓ Tailscale: Healthy (IP: 100.64.0.5)

→ Integration tests...

Test 1: Ollama → LiteLLM → Open WebUI (end-to-end)
✓ Prompt sent via Open WebUI
✓ Routed through LiteLLM
✓ Processed by Ollama (llama3.2:latest)
✓ Response received: "Hello! I'm ready to assist."
✓ Total latency: 342ms
✓ Status: PASS

Test 2: Document ingestion (Anything LLM)
✓ Test document uploaded: test.pdf (1.2 MB)
✓ Chunked: 47 chunks
✓ Embedded via LiteLLM: OK
✓ Stored in Qdrant: OK
✓ Query test: "What is in the document?"
✓ Response: Relevant summary returned
✓ Status: PASS

Test 3: External API fallback (LiteLLM)
✓ Complex query sent (>8K tokens)
✓ LiteLLM routed to: OpenAI GPT-4o

✓ Response received: OK
✓ Status: PASS

Test 4: OpenClaw code generation
✓ Accessed via Tailscale: http://100.64.0.5:18789
✓ Prompt: "Write a Python function to sort a list"
✓ Model: ollama/mistral:latest
✓ Code generated: Valid Python function
✓ Saved to: /mnt/data/ai-platform/openclaw/projects/test/
✓ Status: PASS

Test 5: Grafana metrics visualization
✓ Ollama inference metrics: Visible
✓ LiteLLM routing stats: Visible
✓ System resources: Visible
✓ Status: PASS

Test 6: n8n workflow execution
✓ Test workflow created: "Notify on model completion"
✓ Trigger: Webhook
✓ Action: Call LiteLLM API
✓ Workflow executed: OK
✓ Status: PASS

→ Google Drive sync status...
✓ Google Drive integration: ENABLED
✓ First sync scheduled: 2025-02-07 16:52:18 UTC (in 1 hour)
✓ Sync destination: /mnt/data/ai-platform/gdrive/
✓ Anything LLM auto-ingestion: ENABLED

→ Backup verification...
✓ Backup script deployed: /home/jglaine/ai-platform/scripts/backup.sh
✓ Cron job created: Daily at 2:00 AM UTC
✓ Backup destination: /mnt/data/ai-platform/backups/
✓ Retention: 7 days

═══

✅ DEPLOYMENT COMPLETE

═══

Deployment Summary:

Total Services: 17 (all healthy)
  ✓ Infrastructure: 5 services
  ✓ AI Applications: 6 services
  ✓ Monitoring: 2 services
  ✓ Networking: 2 services
  ✓ Databases: 2 services

Deployment Timeline:
  Phase 1 (Validation): 1m 12s
  Phase 2 (Image pull): 6m 34s
  Phase 3 (Infrastructure): 26s
  Phase 4 (Tailscale): 15s
  Phase 5 (Ollama + models): 12m 51s
  Phase 6 (LiteLLM): 32s
  Phase 7 (AI apps): 2m 18s
  Phase 8 (Monitoring): 47s
  Phase 9 (Reverse proxy): 1m 03s
  Phase 10 (Health checks): 1m 25s
  ──────────────────────────────────
  Total: 26m 43s

Resource Usage:
  CPU: 18% (avg across 32 threads)
  RAM: 22.4 GB / 64 GB (35%)
  GPU VRAM: 14.9 GB / 24 GB (62%)
  Disk (DATA_ROOT): 41.2 GB used, 1.16 TB free

Network Configuration:
  Domain: ai.jglaine.com
  HTTPS: Enabled (Let's Encrypt)
  Certificate valid until: 2025-05-08
  Tailscale IP: 100.64.0.5

═══

🌐 ACCESS URLS

═══

AI Applications (via domain, HTTPS enabled):

Open WebUI https://ai.jglaine.com/webui
Dify https://ai.jglaine.com/dify
Anything LLM https://ai.jglaine.com/anything
n8n https://ai.jglaine.com/n8n
Flowise https://ai.jglaine.com/flowise

OpenClaw (Tailscale VPN ONLY, NOT via domain):

OpenClaw http://100.64.0.5:18789

Monitoring & Management:

Grafana https://ai.jglaine.com/grafana
Prometheus https://ai.jglaine.com/prometheus
MinIO Console https://ai.jglaine.com/minio

APIs (programmatic access):

Ollama API https://ai.jglaine.com/ollama/api
LiteLLM API https://ai.jglaine.com/litellm/v1
Qdrant API https://ai.jglaine.com/qdrant/collections

Health Check:

Platform Status https://ai.jglaine.com/health

==========

🔑 DEFAULT CREDENTIALS
==========

⚠️ IMPORTANT: Change all default passwords on first login!

Grafana:
Username: admin
Password: (stored in .env: GRAFANA_ADMIN_PASSWORD)

Dify:
Username: admin@ai-platform.local

Password: (shown in logs during Phase 7, or reset via CLI)

MinIO Console:
Username: minioadmin
Password: (stored in .env: MINIO_ROOT_PASSWORD)

n8n:
Initial setup required (create account on first access)

Open WebUI:
Initial setup required (create account on first access)

Anything LLM:
Initial setup required (create account on first access)

Flowise:
Initial setup required (create account on first access)

OpenClaw:
No authentication (secured by Tailscale VPN)

═══

📊 QUICK START GUIDE

═══

1. Access Open WebUI (easiest way to test):
→ Go to: https://ai.jglaine.com/webui
→ Create an account
→ Select model: llama3.2:latest
→ Start chatting!

2. Test OpenClaw (AI coding agent):
→ Ensure you're connected to Tailscale VPN
→ Go to: http://100.64.0.5:18789
→ Try: "Create a Python Flask API with 3 endpoints"

3. Monitor system health:
→ Go to: https://ai.jglaine.com/grafana
→ Login with admin credentials
→ View: "Ollama Performance" dashboard

4. Set up workflows (n8n):

→ Go to: https://ai.jglaine.com/n8n
→ Create account on first access
→ Example workflow: "Notify me when Ollama finishes a large task"

5. Build LLM apps (Dify):
→ Go to: https://ai.jglaine.com/dify
→ Login: admin@ai-platform.local
→ Create your first LLM-powered application

═══

⚠️ IMPORTANT NOTES

═══

1. OpenClaw Access:
✓ Only accessible via Tailscale VPN
✓ NOT exposed to public internet (security by design)
✓ If Tailscale disconnects, run:
docker restart tailscale-ai-platform

2. Google Drive Sync:
✓ First sync: 2025-02-07 16:52:18 UTC (in 1 hour)
✓ Interval: Every 1 hour
✓ Destination: /mnt/data/ai-platform/gdrive/
✓ Auto-ingested by Anything LLM

3. SSL Certificate:
✓ Auto-renews every 30 days
✓ Next renewal: ~2025-03-09
✓ Caddy handles renewal automatically (no action required)

4. Ollama Models:
✓ 3 models downloaded (13.5 GB)
✓ To add more models:
docker exec ollama-ai-platform ollama pull <model-name>
✓ Check available models:
docker exec ollama-ai-platform ollama list

5. Backups:
✓ Automated daily backups at 2:00 AM UTC
✓ Backup location: /mnt/data/ai-platform/backups/
✓ Retention: 7 days
✓ Manual backup:

bash /home/jglaine/ai-platform/scripts/backup.sh

6. Resource Monitoring:
   ✓ Current VRAM usage: 62% (14.9 GB / 24 GB)
   ✓ Can load 3 models in parallel
   ✓ If VRAM errors occur, reduce OLLAMA_NUM_PARALLEL in .env

7. External API Costs:
   ✓ OpenAI GPT-4o: Only used for complex queries
   ✓ Estimated monthly cost: $5-$20 (depends on usage)
   ✓ Monitor usage: https://platform.openai.com/usage

8. Service Logs:
   ✓ View logs for any service:
     docker logs <container-name> -f
   ✓ Example:
     docker logs ollama-ai-platform -f

9. Stopping Services:
   ✓ Stop all services:
     cd /home/jglaine/ai-platform/deployment/stack
     docker-compose down
   ✓ Start all services:
     docker-compose up -d

10. Updating Services:
    ✓ Pull latest images:
      docker-compose pull
    ✓ Restart with new images:
      docker-compose up -d --force-recreate

═══

📝 NEXT STEPS

═══

1. ✅ Deployment complete (this script)
2. → Configure services (run: bash 3-configure-services.sh)
   - Set up first-time passwords
   - Configure integrations
   - Test all workflows
3. → Add optional services (run: bash 4-add-services.sh)
   - Install additional models

- Enable more features
- Customize deployments

═══

✨ Deployment completed successfully!

═══

Logs saved to: /mnt/data/ai-platform/logs/deployment-20250207-155218.log

For support, visit: https://github.com/yourusername/ai-platform

═══

Script 2 Success Criteria

- Exit code 0 (success)
- All 17 containers running and healthy
- PostgreSQL: Databases created (4 databases)
- Redis: Connection successful
- MinIO: Buckets created (3 buckets)
- Qdrant: Running and accessible
- Ollama: 3 models downloaded (13.5 GB)
- Ollama: GPU active (VRAM usage visible)
- LiteLLM: Routing functional (local + external APIs)
- Tailscale: IP assigned (e.g., 100.64.0.5)
- OpenClaw: Accessible via Tailscale IP only
- Caddy: SSL certificate obtained (Let's Encrypt)
- Caddy: All 14 routes functional
- Caddy: OpenClaw NOT proxied (404 on domain, 200 on Tailscale IP)
- Grafana: Dashboards loaded
- Integration tests: All 6 tests PASS
- Health check endpoint: Returns 200 OK
- Backup cron job: Created
- Total deployment time: < 30 minutes

⚙️ SCRIPT 3: SERVICE CONFIGURATION

Purpose: Perform post-deployment configuration, set up integrations, test workflows, configure monitoring alerts, and verify end-to-end functionality.

Expected Output

```
$ bash 3-configure-services.sh

═══════════════════════════════════════════════════════════════════════════
═══════════════════

🦞 AI PLATFORM - SERVICE CONFIGURATION
═══════════════════════════════════════════════════════════════════════════
═══════════════════

Version: v75.2.0
Host: homelab.local (192.168.1.100)
User: jglaine
Date: 2025-02-07 16:15:30 UTC

This script will:
  1. Verify all services are healthy
  2. Configure first-time setup for services requiring manual setup
  3. Set up monitoring alerts (Prometheus + Grafana)
  4. Test integrations (Google Drive, external APIs)
  5. Configure backup automation
  6. Run end-to-end workflow tests
  7. Generate usage documentation

═══════════════════════════════════════════════════════════════════════════
═══════════════════

🔧 PHASE 1/8: SERVICE HEALTH VERIFICATION
═══════════════════════════════════════════════════════════════════════════
═══════════════════

→ Checking all 17 services...
  ✓ postgres-ai-platform: Healthy
  ✓ redis-ai-platform: Healthy
  ✓ minio-ai-platform: Healthy
  ✓ qdrant-ai-platform: Healthy
  ✓ prometheus-ai-platform: Healthy
  ✓ grafana-ai-platform: Healthy
  ✓ ollama-ai-platform: Healthy
  ✓ litellm-ai-platform: Healthy
  ✓ tailscale-ai-platform: Healthy
  ✓ openclaw-ai-platform: Healthy
  ✓ open-webui-ai-platform: Healthy
  ✓ dify-api-ai-platform: Healthy
```

✓ dify-web-ai-platform: Healthy
✓ anything-llm-ai-platform: Healthy
✓ n8n-ai-platform: Healthy
✓ flowise-ai-platform: Healthy
✓ caddy-ai-platform: Healthy

→ All services healthy (17/17)

═══

🔧 PHASE 2/8: FIRST-TIME SERVICE SETUP

═══

→ Configuring Open WebUI...
[?] Open WebUI requires initial account creation
[?] Visit: https://ai.jglaine.com/webui
[?] Create admin account now? [Y/n]: Y

→ Opening browser to: https://ai.jglaine.com/webui
⚠️ Please complete account creation in browser
[?] Press Enter when account is created...

→ Verifying Open WebUI configuration...
✓ Admin account created
✓ Connected to Ollama
✓ Models available: 3
✓ Status: Configured

→ Configuring Dify...
✓ Admin account already exists: admin@ai-platform.local
✓ Password: (stored in deployment logs from Script 2)

[?] Reset admin password? [y/N]: N

✓ Default workspace created
✓ Ollama integration: Active
✓ Vector DB (Qdrant): Connected
✓ Status: Configured

→ Configuring Anything LLM...
[?] Anything LLM requires initial account creation
[?] Visit: https://ai.jglaine.com/anything
[?] Create admin account now? [Y/n]: Y

→ Opening browser to: https://ai.jglaine.com/anything
⚠️ Please complete account creation in browser
[?] Press Enter when account is created...

→ Verifying Anything LLM configuration...
✓ Admin account created
✓ LLM provider: LiteLLM (configured)
✓ Vector DB: Qdrant (connected)
✓ Google Drive sync: ENABLED
✓ Status: Configured

→ Configuring n8n...
[?] n8n requires initial account creation
[?] Visit: https://ai.jglaine.com/n8n
[?] Create admin account now? [Y/n]: Y

→ Opening browser to: https://ai.jglaine.com/n8n
⚠️ Please complete account creation in browser
[?] Press Enter when account is created...

→ Verifying n8n configuration...
✓ Admin account created
✓ Workflow engine: Ready
✓ Credentials vault: Active
✓ Status: Configured

→ Configuring Flowise...
[?] Flowise requires initial account creation
[?] Visit: https://ai.jglaine.com/flowise
[?] Create admin account now? [Y/n]: Y

→ Opening browser to: https://ai.jglaine.com/flowise
⚠️ Please complete account creation in browser
[?] Press Enter when account is created...

→ Verifying Flowise configuration...
✓ Admin account created
✓ LLM connections: LiteLLM + Ollama
✓ Vector DB: Qdrant (connected)
✓ Status: Configured

→ First-time setup summary:
✓ Open WebUI: Configured

✓ Dify: Configured (admin password unchanged)
✓ Anything LLM: Configured
✓ n8n: Configured
✓ Flowise: Configured
✓ OpenClaw: No setup required (Tailscale VPN)

═══

🔧 PHASE 3/8: MONITORING & ALERTS CONFIGURATION

═══

→ Configuring Prometheus alert rules...
✓ Alert: OllamaDown (if Ollama unreachable for 5 minutes)
✓ Alert: HighVRAMUsage (if VRAM > 90% for 10 minutes)
✓ Alert: PostgreSQLDown (if PostgreSQL unreachable)
✓ Alert: DiskSpaceLow (if DATA_ROOT < 50 GB free)
✓ Alert: HighCPUUsage (if CPU > 80% for 15 minutes)
✓ Alert: ContainerDown (if any AI Platform container stops)
✓ Alert rules saved: /home/jglaine/ai-platform/deployment/configs/prometheus-alerts.yml
✓ Prometheus reloaded configuration

→ Configuring Grafana alerting...
✓ Alert channel: Email (admin@jglaine.com)
✓ Alert channel: (Signal disabled, skipped)
✓ Notification policy: Send on state change
✓ Silence period: None (alert on every occurrence)
✓ Repeat interval: 4 hours

→ Testing Grafana alert delivery...
✓ Test alert sent to: admin@jglaine.com
✓ Email delivered successfully (check inbox)
✓ Alert configuration: Validated

→ Creating Grafana notification rules...
✓ Rule 1: Ollama performance degradation (trigger: inference > 10s)
✓ Rule 2: LiteLLM routing failures (trigger: >5% error rate)
✓ Rule 3: Disk space warning (trigger: <100 GB free)
✓ Rule 4: GPU temperature alert (trigger: >85°C)
✓ Rule 5: Service health check failure
✓ All rules: Active

→ Monitoring & alerts summary:
✓ Prometheus alerts: 6 rules configured

✓ Grafana notifications: 5 rules configured
✓ Alert channels: Email (active)
✓ Test alert: Delivered successfully

═══

🔧 PHASE 4/8: INTEGRATION TESTING

═══

→ Testing Google Drive integration...
✓ Token file exists: /home/jglaine/ai-platform/deployment/.secrets/gdrive_token.json
✓ Token decrypted successfully
✓ Token valid until: 2025-02-08 15:10:42 UTC

→ Triggering manual sync (test)...
✓ Connecting to Google Drive API...
✓ Authenticated as: user@jglaine.com
✓ Root folder accessible
✓ Files found: 47 documents
✓ Downloading to: /mnt/data/ai-platform/gdrive/

Download progress:
[1/47] Project_Proposal.docx (2.3 MB) ✓
[2/47] Meeting_Notes_2024.pdf (1.1 MB) ✓
[3/47] Budget_Sheet.xlsx (0.8 MB) ✓
...
[47/47] Research_Paper.pdf (3.4 MB) ✓

✓ Total downloaded: 47 files (127.8 MB)
✓ Sync duration: 43 seconds

→ Testing Anything LLM auto-ingestion...
✓ Monitoring directory: /mnt/data/ai-platform/gdrive/
✓ Processing files... (47 documents)

Ingestion progress:
✓ Chunking documents: 47/47
✓ Total chunks created: 1,847
✓ Embedding via LiteLLM: 1,847/1,847
✓ Storing in Qdrant: 1,847/1,847
✓ Indexing complete

✓ Ingestion duration: 4m 12s

→ Testing document search...
Query: "What are the budget allocations for Q1?"
✓ Search executed in Qdrant
✓ Retrieved: 8 relevant chunks
✓ Context assembled
✓ LLM response generated
✓ Response: "Based on the budget sheet, Q1 allocations are..."
✓ Search latency: 1.2s
✓ Status: PASS

✓ Google Drive integration: Fully functional

→ Testing external API routing (LiteLLM)...

Test 1: Simple query → Local Ollama
✓ Query: "What is Python?"
✓ Routed to: ollama/llama3.2:latest
✓ Response time: 187ms
✓ Status: PASS

Test 2: Complex query → OpenAI GPT-4o
✓ Query: "Write a detailed technical specification for a microservices architecture..."
✓ Detected: Complex (>500 tokens expected)
✓ Routed to: openai/gpt-4o
✓ Response time: 3.4s
✓ Status: PASS

Test 3: OpenAI unavailable → Groq fallback
✓ Simulating OpenAI timeout...
✓ Fallback triggered
✓ Routed to: groq/llama-3.1-70b-versatile
✓ Response time: 1.1s
✓ Status: PASS

Test 4: All external APIs down → Local fallback
✓ Simulating all external API failures...
✓ Final fallback triggered
✓ Routed to: ollama/mistral:latest
✓ Response time: 234ms
✓ Status: PASS

✓ LiteLLM routing: Fully functional

→ Testing Ollama multi-model inference...

Test 1: Load 3 models in parallel
✓ Loading llama3.2:latest (VRAM: 5.2 GB)
✓ Loading mistral:latest (VRAM: 4.6 GB)
✓ Loading qwen2.5:7b (VRAM: 5.1 GB)
✓ Total VRAM: 14.9 GB / 24 GB (62%)
✓ All models loaded successfully

Test 2: Concurrent inference (3 requests)
✓ Request 1 → llama3.2:latest: "Translate to French..."
✓ Request 2 → mistral:latest: "Write a Python function..."
✓ Request 3 → qwen2.5:7b: "用中文回答..."
✓ All responses received
✓ Avg latency: 312ms
✓ No VRAM errors
✓ Status: PASS

✓ Ollama multi-model: Fully functional

→ Testing OpenClaw (AI coding agent)...
✓ Access via Tailscale: http://100.64.0.5:18789
✓ Connection: Established

Test task: "Create a REST API with 3 endpoints"
✓ Model: ollama/mistral:latest
✓ Prompt sent
✓ Code generation started...
✓ Files created:
- /mnt/data/ai-platform/openclaw/projects/rest_api/app.py
- /mnt/data/ai-platform/openclaw/projects/rest_api/requirements.txt
- /mnt/data/ai-platform/openclaw/projects/rest_api/README.md
✓ Code review: Syntax valid (Python 3.11)
✓ Auto-saved: Enabled
✓ Conversation history: Saved
✓ Status: PASS

✓ OpenClaw: Fully functional

→ Integration testing summary:
✓ Google Drive sync: PASS
✓ Anything LLM ingestion: PASS
✓ Document search: PASS
✓ LiteLLM routing: PASS (4/4 tests)

✓ Ollama multi-model: PASS (2/2 tests)
✓ OpenClaw coding: PASS

═══

🔧 PHASE 5/8: WORKFLOW AUTOMATION SETUP

═══

→ Creating sample n8n workflows...

Workflow 1: "Daily Summary Report"
Purpose: Generate daily summary of AI platform usage
Trigger: Cron (daily at 9:00 AM)
Steps:
1. Query Prometheus for metrics (last 24h)
2. Aggregate: Total queries, VRAM usage, response times
3. Call LiteLLM to generate summary
4. Send email with report
✓ Workflow created and saved
✓ Status: Active (next run: 2025-02-08 09:00:00 UTC)

Workflow 2: "Model Performance Alert"
Purpose: Notify when Ollama response time > 5s
Trigger: Webhook (from Prometheus)
Steps:
1. Receive alert from Prometheus
2. Query Ollama for model status
3. Generate diagnostic report
4. Send email notification
✓ Workflow created and saved
✓ Status: Active
✓ Webhook URL: https://ai.jglaine.com/n8n/webhook/model-alert

Workflow 3: "Auto-Document Ingestion"
Purpose: Automatically ingest new Google Drive files
Trigger: Google Drive webhook (on new file)
Steps:
1. Detect new file in Google Drive
2. Download to /mnt/data/ai-platform/gdrive/
3. Trigger Anything LLM ingestion API
4. Send confirmation notification
✓ Workflow created and saved
✓ Status: Active

✓ Google Drive webhook: Configured

→ Creating sample Flowise flows...

Flow 1: "Conversational RAG"
Purpose: Chat with your documents (Google Drive)
Components:
- Vector store: Qdrant
- Embeddings: LiteLLM (via Ollama)
- LLM: LiteLLM (with fallback)
- Memory: Conversation buffer
✓ Flow created and saved
✓ Status: Ready
✓ Chat endpoint: https://ai.jglaine.com/flowise/api/v1/prediction/conv-rag

Flow 2: "Multi-Agent Code Review"
Purpose: Review code using multiple AI agents
Agents:
- Agent 1: Syntax checker (mistral)
- Agent 2: Security auditor (llama3.2)
- Agent 3: Performance optimizer (qwen2.5)
- Orchestrator: Aggregates feedback
✓ Flow created and saved
✓ Status: Ready
✓ API endpoint: https://ai.jglaine.com/flowise/api/v1/prediction/code-review

Flow 3: "Smart Document Summarizer"
Purpose: Summarize long documents intelligently
Logic:
- Short docs (<5 pages) → Local Ollama
- Long docs (>5 pages) → OpenAI GPT-4o (better for long context)
✓ Flow created and saved
✓ Status: Ready
✓ API endpoint: https://ai.jglaine.com/flowise/api/v1/prediction/summarizer

→ Testing sample workflows...
✓ n8n Workflow 1: Test execution successful (dry run)
✓ n8n Workflow 2: Webhook responding (200 OK)
✓ n8n Workflow 3: Google Drive webhook verified
✓ Flowise Flow 1: Chat test successful
✓ Flowise Flow 2: Code review test successful
✓ Flowise Flow 3: Summarization test successful

→ Workflow automation summary:

✓ n8n workflows: 3 created (all active)
✓ Flowise flows: 3 created (all ready)
✓ All workflows tested: PASS

═══

🔧 PHASE 6/8: BACKUP AUTOMATION CONFIGURATION

═══

→ Configuring automated backups...

Backup strategy:
- PostgreSQL: Daily pg_dump (all databases)
- Qdrant: Daily snapshot export
- Configuration: .env + docker-compose files
- Secrets: Encrypted credentials backup
- Retention: 7 days (auto-delete old backups)

✓ Backup script: /home/jglaine/ai-platform/scripts/backup.sh
✓ Backup destination: /mnt/data/ai-platform/backups/
✓ Cron job: Daily at 2:00 AM UTC
✓ Cron entry: 0 2 * * * /home/jglaine/ai-platform/scripts/backup.sh

→ Running test backup (manual)...

[1/5] PostgreSQL backup...
✓ Backing up: ai_platform_db
✓ Backing up: dify_db
✓ Backing up: anything_llm_db
✓ Backing up: n8n_db
✓ Total size: 47.2 MB
✓ Compressed: 12.8 MB (73% reduction)
✓ Saved: /mnt/data/ai-platform/backups/postgres-20250207.sql.gz

[2/5] Qdrant snapshot...
✓ Creating snapshot via Qdrant API
✓ Snapshot ID: snapshot-20250207-161842
✓ Size: 127.8 MB (1,847 vectors)
✓ Compressed: 43.1 MB
✓ Saved: /mnt/data/ai-platform/backups/qdrant-20250207.tar.gz

[3/5] Configuration backup...
✓ Copying: deployment/.secrets/.env (encrypted)

✓ Copying: deployment/stack/docker-compose.yml
✓ Copying: deployment/configs/* (all config files)
✓ Total: 14 files
✓ Archived: /mnt/data/ai-platform/backups/config-20250207.tar.gz (2.3 MB)

[4/5] Secrets backup...
✓ Copying: .secrets/api_keys.enc
✓ Copying: .secrets/gdrive_token.json
✓ All secrets encrypted with GPG
✓ Archived: /mnt/data/ai-platform/backups/secrets-20250207.tar.gz.gpg (1.1 MB)

[5/5] OpenClaw workspace backup...
✓ Backing up: /mnt/data/ai-platform/openclaw/projects/
✓ Files: 23 (code artifacts)
✓ Archived: /mnt/data/ai-platform/backups/openclaw-20250207.tar.gz (4.7 MB)

→ Test backup summary:
✓ PostgreSQL: 12.8 MB (4 databases)
✓ Qdrant: 43.1 MB (1,847 vectors)
✓ Configuration: 2.3 MB (14 files)
✓ Secrets: 1.1 MB (encrypted)
✓ OpenClaw: 4.7 MB (23 files)
─────────────────────────────────
✓ Total backup size: 63.9 MB
✓ Backup duration: 1m 34s
✓ All backups verified: Checksums valid

→ Setting up backup monitoring...
✓ Grafana alert: Backup failure notification
✓ Prometheus metric: backup_last_success_timestamp
✓ Email on failure: admin@jglaine.com
✓ Next scheduled backup: 2025-02-08 02:00:00 UTC

→ Backup automation summary:
✓ Automated backups: ENABLED
✓ Schedule: Daily at 2:00 AM UTC
✓ Retention: 7 days
✓ Test backup: Successful (63.9 MB)
✓ Monitoring: Configured

═══

🔧 PHASE 7/8: END-TO-END WORKFLOW TESTS

═══

→ Running comprehensive end-to-end tests...

Test 1: Complete RAG Pipeline
Scenario: User asks question about documents via Open WebUI
Steps:
1. User query: "What was discussed in the last meeting?"
2. Open WebUI → LiteLLM → Anything LLM (RAG)
3. Anything LLM searches Qdrant (Google Drive docs)
4. Retrieves relevant chunks
5. LiteLLM generates response (using context)
6. Response returned to user

✓ Query sent via Open WebUI
✓ Routed through LiteLLM
✓ RAG search in Qdrant: 6 relevant chunks retrieved
✓ Context assembled (847 tokens)
✓ LLM response generated (ollama/llama3.2)
✓ Response: "The last meeting discussed budget allocations..."
✓ End-to-end latency: 2.8s
✓ Status: PASS

Test 2: Multi-Service Code Generation
Scenario: Generate code with OpenClaw, review with Flowise
Steps:
1. OpenClaw generates Python Flask API
2. Code saved to workspace
3. Flowise multi-agent review triggered
4. Review results returned
5. Code improvements suggested

✓ OpenClaw prompt: "Create a Flask API for user management"
✓ Code generated: app.py, models.py, routes.py
✓ Saved to: /mnt/data/ai-platform/openclaw/projects/user_api/
✓ Flowise review triggered (3 agents)
✓ Agent 1 (Syntax): No errors found
✓ Agent 2 (Security): Suggested input validation improvements
✓ Agent 3 (Performance): Suggested database indexing
✓ Aggregated feedback generated
✓ Status: PASS

Test 3: Automated Workflow Execution

Scenario: New file in Google Drive → Auto-ingestion → Notification
Steps:
1. Simulate new file upload to Google Drive
2. n8n workflow detects change (webhook)
3. File downloaded to /mnt/data/ai-platform/gdrive/
4. Anything LLM ingestion triggered
5. Email notification sent

✓ Test file created: test_document_new.pdf
✓ Google Drive webhook triggered
✓ n8n workflow activated
✓ File downloaded: test_document_new.pdf (1.2 MB)
✓ Anything LLM ingestion: 12 chunks created
✓ Embedded and stored in Qdrant
✓ Email notification sent to: admin@jglaine.com
✓ Total workflow time: 47s
✓ Status: PASS

Test 4: External API Fallback Chain
Scenario: Test full fallback hierarchy
Steps:
1. Send complex query (forces external API)
2. Simulate OpenAI failure
3. Fallback to Groq
4. Simulate Groq failure
5. Final fallback to local Ollama

✓ Query: "Explain distributed systems architecture in detail..."
✓ Detected: Complex query (>1000 tokens expected)
✓ Primary route: openai/gpt-4o (simulated failure)
✓ Fallback 1: groq/llama-3.1-70b (simulated failure)
✓ Fallback 2: ollama/mistral:latest (success)
✓ Response generated locally
✓ Latency: 3.2s (acceptable for complex query)
✓ Status: PASS

Test 5: Multi-Model Parallel Inference
Scenario: 3 simultaneous requests to different models
Steps:
1. Request A → llama3.2 (translation)
2. Request B → mistral (code generation)
3. Request C → qwen2.5 (Chinese language task)
4. All processed in parallel

✓ Request A sent: "Translate to Spanish: Hello world"
✓ Request B sent: "Write a Python decorator for logging"
✓ Request C sent: "用中文解释机器学习"
✓ All models active simultaneously
✓ VRAM usage: 14.9 GB (stable, no OOM)
✓ Response A received: 412ms
✓ Response B received: 738ms
✓ Response C received: 521ms
✓ All responses valid
✓ Status: PASS

Test 6: Monitoring & Alerting
Scenario: Trigger alert condition, verify notification
Steps:
1. Simulate high VRAM usage (>90%)
2. Prometheus detects condition
3. Alert rule triggered
4. Grafana sends notification

✓ Simulating VRAM spike: 22.1 GB / 24 GB (92%)
✓ Prometheus alert rule evaluated
✓ Alert state: FIRING
✓ Grafana notification triggered
✓ Email sent to: admin@jglaine.com
✓ Email subject: "[ALERT] High VRAM Usage on AI Platform"
✓ Email delivered: Confirmed
✓ Alert cleared (VRAM returned to normal)
✓ Status: PASS

→ End-to-end test summary:
✓ Test 1 (RAG Pipeline): PASS
✓ Test 2 (Code Generation): PASS
✓ Test 3 (Automated Workflow): PASS
✓ Test 4 (Fallback Chain): PASS
✓ Test 5 (Parallel Inference): PASS
✓ Test 6 (Monitoring): PASS
──────────────────────────────────────
✓ All tests passed: 6/6 (100%)

═══

🔧 PHASE 8/8: DOCUMENTATION GENERATION

═══

→ Generating usage documentation...

[1/5] Quick Start Guide
✓ File: /home/jglaine/ai-platform/docs/QUICKSTART.md
✓ Contents: Basic usage, common tasks, first steps
✓ Size: 12.4 KB

[2/5] API Reference
✓ File: /home/jglaine/ai-platform/docs/API_REFERENCE.md
✓ Contents: All API endpoints, examples, authentication
✓ Services documented:
- Ollama API (3 endpoints)
- LiteLLM API (5 endpoints)
- Flowise API (3 flows)
- n8n Webhooks (3 workflows)
- Qdrant API (vector operations)
✓ Size: 28.7 KB

[3/5] Troubleshooting Guide
✓ File: /home/jglaine/ai-platform/docs/TROUBLESHOOTING.md
✓ Contents: Common issues, solutions, debugging tips
✓ Covers:
- Service startup issues
- VRAM/OOM errors
- Network connectivity
- SSL certificate problems
- Backup/restore procedures
✓ Size: 19.3 KB

[4/5] Advanced Configuration
✓ File: /home/jglaine/ai-platform/docs/ADVANCED.md
✓ Contents: Custom configurations, fine-tuning, optimization
✓ Topics:
- Adding new Ollama models
- LiteLLM routing customization
- Grafana dashboard creation
- n8n workflow examples
- Performance tuning
✓ Size: 34.1 KB

[5/5] Security Best Practices
✓ File: /home/jglaine/ai-platform/docs/SECURITY.md
✓ Contents: Security hardening, credential management, compliance

✓ Topics:
- Tailscale VPN configuration
- OpenClaw isolation rationale
- Credential encryption (GPG)
- SSL/TLS best practices
- Backup encryption

✓ Size: 16.8 KB

→ Generating system inventory...
✓ File: /home/jglaine/ai-platform/docs/SYSTEM_INVENTORY.json
✓ Contents: Complete system state snapshot
- All service versions
- Container IDs and statuses
- Resource usage metrics
- Network configuration
- Storage locations
- API endpoints

✓ Format: JSON (machine-readable)
✓ Size: 8.2 KB

→ Generating usage examples...
✓ Directory: /home/jglaine/ai-platform/examples/
✓ Example 1: curl_examples.sh (API testing)
✓ Example 2: python_client.py (Python SDK usage)
✓ Example 3: n8n_workflows.json (workflow templates)
✓ Example 4: flowise_flows.json (flow templates)
✓ Total: 4 example files

→ Documentation summary:
✓ Core docs: 5 files (111.3 KB total)
✓ System inventory: 1 file (JSON)
✓ Examples: 4 files
✓ All docs saved to: /home/jglaine/ai-platform/docs/

═══

✅ SERVICE CONFIGURATION COMPLETE

═══

Configuration Summary:

──

Services Configured:

✓ Open WebUI: Admin account created
✓ Dify: Admin password unchanged (use existing)
✓ Anything LLM: Admin account created, Google Drive sync active
✓ n8n: Admin account created, 3 workflows active
✓ Flowise: Admin account created, 3 flows ready
✓ Grafana: Alerts configured, email channel active
✓ Prometheus: 6 alert rules active
✓ OpenClaw: No setup required (Tailscale VPN)

Integrations Tested:
✓ Google Drive sync: Functional (47 files synced)
✓ Anything LLM ingestion: Functional (1,847 vectors)
✓ LiteLLM routing: Functional (local + external APIs)
✓ Ollama multi-model: Functional (3 models parallel)
✓ OpenClaw coding: Functional (code generation tested)

Workflows Created:
n8n:
✓ Daily Summary Report (active, next run: 09:00 AM UTC)
✓ Model Performance Alert (active, webhook ready)
✓ Auto-Document Ingestion (active, Google Drive webhook)

Flowise:
✓ Conversational RAG (API: /api/v1/prediction/conv-rag)
✓ Multi-Agent Code Review (API: /api/v1/prediction/code-review)
✓ Smart Document Summarizer (API: /api/v1/prediction/summarizer)

Backups Configured:
✓ Automated daily backups: ENABLED
✓ Schedule: 2:00 AM UTC
✓ Retention: 7 days
✓ Last test backup: 63.9 MB (successful)
✓ Next backup: 2025-02-08 02:00:00 UTC

End-to-End Tests:
✓ All 6 tests PASSED (100%)
✓ RAG pipeline: Functional
✓ Code generation + review: Functional
✓ Automated workflows: Functional
✓ API fallback chain: Functional
✓ Parallel inference: Functional
✓ Monitoring & alerts: Functional

Documentation Generated:

✓ Quick Start Guide (12.4 KB)
✓ API Reference (28.7 KB)
✓ Troubleshooting Guide (19.3 KB)
✓ Advanced Configuration (34.1 KB)
✓ Security Best Practices (16.8 KB)
✓ System Inventory (JSON, 8.2 KB)
✓ Usage Examples (4 files)

System Health:
✓ All services: Healthy (17/17)
✓ CPU usage: 21% (avg)
✓ RAM usage: 23.7 GB / 64 GB (37%)
✓ GPU VRAM: 14.9 GB / 24 GB (62%)
✓ Disk (DATA_ROOT): 42.1 GB used, 1.15 TB free
✓ Network: All routes responsive
✓ SSL certificate: Valid (expires 2025-05-08)

═══

📝 NEXT STEPS

═══

1. ✅ System setup complete (Script 1)
2. ✅ Services deployed (Script 2)
3. ✅ Configuration complete (this script)
4. → Add optional services (run: bash 4-add-services.sh)
   - Install additional Ollama models
   - Enable more integrations
   - Customize advanced features

═══

📚 DOCUMENTATION QUICK LINKS

═══

Getting Started:
/home/jglaine/ai-platform/docs/QUICKSTART.md

API Documentation:
/home/jglaine/ai-platform/docs/API_REFERENCE.md

Troubleshooting:

/home/jglaine/ai-platform/docs/TROUBLESHOOTING.md

Advanced Topics:
/home/jglaine/ai-platform/docs/ADVANCED.md

Security:
/home/jglaine/ai-platform/docs/SECURITY.md

Examples:
/home/jglaine/ai-platform/examples/

═══

✨ Configuration completed successfully!

═══

Logs saved to: /mnt/data/ai-platform/logs/configuration-20250207-161530.log

Your AI platform is fully configured and ready for production use! 🚀

═══

🎯 Script 3 Success Criteria

- Exit code 0 (success)
- All 17 services remain healthy
- Open WebUI: Admin account created
- Dify: Admin account verified or password reset
- Anything LLM: Admin account created, Google Drive sync confirmed
- n8n: Admin account created, 3 workflows active
- Flowise: Admin account created, 3 flows ready
- Prometheus: 6 alert rules configured and active
- Grafana: 5 notification rules configured, email channel tested
- Google Drive integration: Test sync successful (files downloaded and ingested)
- LiteLLM routing: All 4 tests PASS
- OpenClaw: Code generation test PASS
- Backup automation: Test backup successful (~64 MB), cron job created
- n8n workflows: 3 created and tested
- Flowise flows: 3 created and tested
- End-to-end tests: All 6 tests PASS (100%)

- Documentation: 5 core docs + 1 inventory + 4 examples generated
- Total configuration time: < 15 minutes (excluding manual account creation pauses)

🔧 SCRIPT 4: ADD OPTIONAL SERVICES

Purpose: Install additional Ollama models, enable advanced integrations, and customize features
Prerequisites: Scripts 1-3 completed successfully
Estimated time: 15-45 minutes (depending on model sizes)

Script 4 Expected Output

```
═══════════════════════════════════════════════════════════════
🚀 AI PLATFORM - SCRIPT 4: ADD OPTIONAL SERVICES
═══════════════════════════════════════════════════════════════

Script: 4-add-services.sh
Version: v75.2.0
Started: 2025-02-07 16:20:15 UTC
Host: jglaine-ai-server
User: jglaine

Prerequisites Check:
  ✓ Script 1 (setup): Completed
  ✓ Script 2 (deployment): Completed
  ✓ Script 3 (configuration): Completed
  ✓ All services: Healthy (17/17)

═══════════════════════════════════════════════════════════════
🎯 OPTIONAL SERVICES MENU
═══════════════════════════════════════════════════════════════

Available Options:

[1] Install Additional Ollama Models
    - Download and configure more LLMs for local inference
    - Estimated time: 5-30 minutes per model (depending on size)
    - VRAM impact: Variable (2-40 GB per model)

[2] Enable Advanced Vector Database Features
```

- Multi-collection support in Qdrant
- Hybrid search (vector + keyword)
- Collection snapshots and backups

[3] Add Custom LiteLLM Routes
- Configure custom routing logic
- Add new external API providers
- Set up per-user/per-team quotas

[4] Install Additional n8n Nodes
- Community nodes for extended functionality
- Custom integrations (Slack, Discord, etc.)
- Advanced workflow templates

[5] Enable Model Fine-Tuning Capabilities
- LoRA adapter support in Ollama
- Training data pipeline setup
- Fine-tuning workflow automation

[6] Add Observability Stack Extensions
- Jaeger for distributed tracing
- Elasticsearch for log aggregation
- Custom Grafana dashboards

[7] Enable Multi-User Authentication
- Keycloak integration for SSO
- LDAP/Active Directory support
- Role-based access control (RBAC)

[8] Install Development Tools
- Jupyter Lab for experimentation
- Code-Server (VS Code in browser)
- Model testing playground

[9] Configure Advanced Backup Strategies
- Off-site backup to S3-compatible storage
- Incremental backups
- Automated disaster recovery

[0] Exit (no additional services)

[?] Select options (comma-separated, e.g., 1,2,4): 1,3,8

═══

🔧 PHASE 1/3: INSTALL ADDITIONAL OLLAMA MODELS

═══

Current installed models:
  ✓ llama3.2:latest (5.2 GB VRAM)
  ✓ mistral:latest (4.6 GB VRAM)
  ✓ qwen2.5:7b (5.1 GB VRAM)
  ──────────────────────────────────
  Total: 14.9 GB / 24 GB VRAM used (62%)

Available models for installation:

Code Generation Models:
  [1] codellama:7b (Code Llama - 3.8 GB)
  [2] codellama:13b (Code Llama Large - 7.3 GB)
  [3] deepseek-coder:6.7b (DeepSeek Coder - 3.7 GB)
  [4] starcoder2:7b (StarCoder 2 - 4.0 GB)

Reasoning & Analysis Models:
  [5] llama3.1:8b (Llama 3.1 - 4.7 GB)
  [6] llama3.1:70b-q4 (Llama 3.1 70B Quantized - 39 GB) ⚠️ High VRAM
  [7] qwen2.5:14b (Qwen 2.5 14B - 8.5 GB)
  [8] phi3:medium (Phi-3 Medium - 7.9 GB)

Specialized Models:
  [9] llava:13b (Vision model - 8.0 GB)
  [10] nomic-embed-text (Embeddings only - 0.5 GB)
  [11] aya:8b (Multilingual - 4.7 GB)
  [12] gemma2:9b (Gemma 2 - 5.5 GB)

──

[?] Select models to install (comma-separated): 1,3,10

→ Selected models:
  ✓ codellama:7b (3.8 GB)
  ✓ deepseek-coder:6.7b (3.7 GB)
  ✓ nomic-embed-text (0.5 GB)

```
  ──────────────────────────────────────────
  Total to download: 8.0 GB
  Estimated VRAM after install: 22.9 GB / 24 GB (95%) ⚠️ High usage

[?] Continue with installation? [Y/n]: Y

→ Installing models...

[1/3] Installing codellama:7b...
  ✓ Pulling from Ollama registry
  Download progress:
    [████████████████████████████████████████] 100% (3.8 GB)
  ✓ Download complete: 2m 14s
  ✓ Extracting layers...
  ✓ Creating model...
  ✓ Model loaded in Ollama

  Testing inference:
    Prompt: "Write a Python function to reverse a string"
    ✓ Response generated in 412ms
    ✓ Code quality: Valid Python syntax

  ✓ codellama:7b installed successfully

[2/3] Installing deepseek-coder:6.7b...
  ✓ Pulling from Ollama registry
  Download progress:
    [████████████████████████████████████████] 100% (3.7 GB)
  ✓ Download complete: 2m 04s
  ✓ Extracting layers...
  ✓ Creating model...
  ✓ Model loaded in Ollama

  Testing inference:
    Prompt: "Explain this code: def factorial(n): return 1 if n == 0 else n * factorial(n-1)"
    ✓ Response generated in 387ms
    ✓ Explanation quality: Accurate and detailed

  ✓ deepseek-coder:6.7b installed successfully

[3/3] Installing nomic-embed-text...
  ✓ Pulling from Ollama registry
  Download progress:
    [████████████████████████████████████████] 100% (0.5 GB)
```

✓ Download complete: 18s
✓ Extracting layers...
✓ Creating model...
✓ Model loaded in Ollama

Testing embeddings:
Text: "Machine learning is a subset of artificial intelligence"
✓ Embedding generated: 768-dimensional vector
✓ Latency: 23ms

✓ nomic-embed-text installed successfully

→ Updating LiteLLM configuration...
✓ Added codellama:7b to routing pool
✓ Added deepseek-coder:6.7b to routing pool
✓ Added nomic-embed-text for embeddings
✓ Routing rules updated:
- Code generation queries → codellama:7b or deepseek-coder (prefer deepseek)
- Embeddings → nomic-embed-text (faster than OpenAI)
✓ LiteLLM config reloaded

→ Model installation summary:
✓ Models installed: 3/3
✓ Total download size: 8.0 GB
✓ Total time: 4m 36s
✓ Current VRAM usage: 22.9 GB / 24 GB (95%)
⚠️ VRAM near capacity - monitor performance

═══

🔧 PHASE 2/3: ADD CUSTOM LITELLM ROUTES

═══

Current LiteLLM routing configuration:
✓ Local Ollama models: 6 models
✓ External APIs: OpenAI, Anthropic, Groq, DeepSeek
✓ Fallback chain: Configured (3 levels)
✓ Rate limiting: Disabled (can enable per-route)

Custom routing options:

[1] Add new external API provider
[2] Configure per-user quotas

[3] Add team-based routing (different models per team)
[4] Enable cost tracking and budget alerts
[5] Configure advanced caching strategies
[6] Add custom routing rules (e.g., time-based, workload-based)

[?] Select options (comma-separated): 1,4

Option 1: Add New External API Provider

Available providers:
  [1] Mistral AI
  [2] Cohere
  [3] Together AI
  [4] Perplexity AI
  [5] Custom OpenAI-compatible endpoint

[?] Select provider: 1

→ Configuring Mistral AI...

[?] Mistral AI API key: **********

→ Validating API key...
  ✓ API key valid
  ✓ Account tier: Standard
  ✓ Available models:
    - mistral-small-latest
    - mistral-medium-latest
    - mistral-large-latest
    - codestral-latest

[?] Which models to enable? [1,2,3,4]: 3,4

→ Adding Mistral models to LiteLLM...
  ✓ mistral-large-latest added (for complex reasoning)
  ✓ codestral-latest added (for advanced code generation)

→ Configuring routing preferences...
  [?] When to use Mistral Large over OpenAI GPT-4o?
      1. Always prefer Mistral (lower cost)

2. Use as fallback if OpenAI fails
3. Load balance 50/50
4. Never use (manual trigger only)

Selection: 3

✓ Load balancing configured: 50% Mistral, 50% OpenAI
✓ Codestral set as primary for code generation (local models as fallback)

→ Updating secrets...
✓ MISTRAL_API_KEY stored in .secrets/api_keys.enc (encrypted)
✓ LiteLLM config updated: /home/jglaine/ai-platform/deployment/configs/litellm_config.yml

→ Testing Mistral integration...
Test 1: Simple query to mistral-large-latest
✓ Query: "Explain quantum computing in simple terms"
✓ Response received in 1.8s
✓ Status: PASS

Test 2: Code generation with codestral-latest
✓ Query: "Write a TypeScript React component for a login form"
✓ Response received in 2.3s
✓ Code quality: Valid TypeScript
✓ Status: PASS

✓ Mistral AI integration: Fully functional

Option 4: Enable Cost Tracking and Budget Alerts

→ Configuring cost tracking...

[?] Monthly budget limit (USD): 100

→ Cost tracking configuration:
✓ Budget limit: $100.00/month
✓ Tracking enabled for:
- OpenAI API calls
- Anthropic API calls
- Groq API calls (free tier, tracked for visibility)
- DeepSeek API calls

- Mistral AI calls
✓ Local Ollama calls: Free (not tracked)

→ Setting up budget alerts...
✓ Alert at 50% budget ($50): Email notification
✓ Alert at 75% budget ($75): Email + reduce external API usage
✓ Alert at 90% budget ($90): Email + pause non-critical external API calls
✓ Alert at 100% budget ($100): Email + block all external API calls (local only)

→ Configuring cost tracking dashboard...
✓ Grafana dashboard created: "API Cost Tracking"
✓ Panels added:
- Current month spending (by provider)
- Cost per model
- Requests vs. cost comparison
- Budget utilization gauge
- Cost trend forecast
✓ Dashboard URL: https://ai.jglaine.com/grafana/d/cost-tracking

→ Creating cost tracking Prometheus metrics...
✓ Metric: litellm_api_cost_total (cumulative cost)
✓ Metric: litellm_api_cost_by_provider (per provider)
✓ Metric: litellm_budget_utilization_percent
✓ All metrics: Exported and scraped by Prometheus

→ Testing cost tracking...
✓ Simulating $5 of API calls
✓ Costs recorded in Prometheus
✓ Grafana dashboard updated
✓ Budget utilization: 5% ($5.00 / $100.00)
✓ Status: Functional

→ Custom LiteLLM routes summary:
✓ New provider added: Mistral AI (2 models)
✓ Cost tracking: ENABLED
✓ Budget limit: $100/month
✓ Budget alerts: Configured (4 levels)
✓ Grafana dashboard: Created

═══

🔧 PHASE 3/3: INSTALL DEVELOPMENT TOOLS

═══

Development tools to install:

[1] Jupyter Lab - Interactive Python notebooks (✓ selected)
[2] Code-Server - VS Code in browser (✓ selected)
[3] Model Testing Playground - Custom UI for model comparison (✓ selected)

→ Installing development tools...

Tool 1/3: Jupyter Lab

→ Creating Jupyter Lab service...
  ✓ Docker image: jupyter/scipy-notebook:latest
  ✓ Port: 8888
  ✓ Volume: /mnt/data/ai-platform/jupyter (notebooks storage)
  ✓ Network: ai-platform (access to all services)

→ Installing Python packages...
  ✓ openai (for OpenAI API)
  ✓ anthropic (for Claude API)
  ✓ litellm (for LiteLLM client)
  ✓ qdrant-client (for Qdrant access)
  ✓ pandas, numpy, matplotlib (data science)
  ✓ langchain (for LLM orchestration)
  ✓ transformers (for model experimentation)

→ Starting Jupyter Lab...
  ✓ Container ID: 9f8e7d6c5b4a
  ✓ Status: Running
  ✓ Health check: Passed

→ Configuring Caddy reverse proxy...
  ✓ Route added: https://ai.jglaine.com/jupyter
  ✓ Authentication: Caddy basic auth (username: jglaine)
  ✓ SSL: Let's Encrypt certificate

→ Generating sample notebooks...
  ✓ Notebook 1: Getting Started with LiteLLM.ipynb
  ✓ Notebook 2: RAG Pipeline with Qdrant.ipynb
  ✓ Notebook 3: Model Comparison and Benchmarking.ipynb

✓ Notebook 4: Fine-tuning Ollama Models.ipynb
✓ Notebook 5: Cost Analysis and Optimization.ipynb
✓ All notebooks saved to: /mnt/data/ai-platform/jupyter/

→ Jupyter Lab summary:
✓ URL: https://ai.jglaine.com/jupyter
✓ Status: Running
✓ Sample notebooks: 5 created
✓ Python packages: 15+ installed

Tool 2/3: Code-Server (VS Code)

→ Creating Code-Server service...
✓ Docker image: codercom/code-server:latest
✓ Port: 8443
✓ Volume: /mnt/data/ai-platform/code-server (workspace)
✓ Network: ai-platform

→ Installing VS Code extensions...
✓ Python (ms-python.python)
✓ Jupyter (ms-toolsai.jupyter)
✓ Docker (ms-azuretools.vscode-docker)
✓ YAML (redhat.vscode-yaml)
✓ GitLens (eamodio.gitlens)
✓ Prettier (esbenp.prettier-vscode)
✓ Remote - SSH (for OpenClaw access)

→ Starting Code-Server...
✓ Container ID: 7a6b5c4d3e2f
✓ Status: Running
✓ Health check: Passed

→ Configuring Caddy reverse proxy...
✓ Route added: https://ai.jglaine.com/code
✓ Authentication: Caddy basic auth (username: jglaine)
✓ SSL: Let's Encrypt certificate

→ Creating workspace folders...
✓ /mnt/data/ai-platform/code-server/projects/
✓ /mnt/data/ai-platform/code-server/scripts/

✓ /mnt/data/ai-platform/code-server/notebooks/
✓ Symlink to OpenClaw workspace: /mnt/data/ai-platform/openclaw/ → workspace/openclaw/

→ Generating sample projects...
✓ Project 1: litellm-api-client/ (Python API client)
✓ Project 2: qdrant-rag-demo/ (RAG implementation)
✓ Project 3: n8n-custom-node/ (n8n node template)
✓ Project 4: grafana-custom-dashboard/ (dashboard JSON)
✓ All projects saved to: /mnt/data/ai-platform/code-server/projects/

→ Code-Server summary:
✓ URL: https://ai.jglaine.com/code
✓ Status: Running
✓ VS Code extensions: 7 installed
✓ Sample projects: 4 created
✓ OpenClaw integration: Symlinked workspace

Tool 3/3: Model Testing Playground

→ Creating Model Testing Playground...
✓ Custom React app: ai-model-playground
✓ Docker image: node:20-alpine (build from source)
✓ Port: 3005
✓ Network: ai-platform

→ Building playground UI...
✓ Cloning repository: https://github.com/ai-platform/model-playground
✓ Installing dependencies (npm install)...
✓ Building React app (npm run build)...
✓ Build complete: 2m 18s
✓ Serving with nginx

→ Starting Model Testing Playground...
✓ Container ID: 3e4f5g6h7i8j
✓ Status: Running
✓ Health check: Passed

→ Configuring Caddy reverse proxy...
✓ Route added: https://ai.jglaine.com/playground
✓ Authentication: None (same auth as main platform)

✓ SSL: Let's Encrypt certificate

→ Configuring playground settings...
✓ Connected LLM providers:
- Ollama (6 local models)
- LiteLLM (all configured routes)
- OpenAI (direct access)
- Anthropic (direct access)
- Mistral AI (direct access)
✓ Features enabled:
- Side-by-side model comparison
- Latency benchmarking
- Token usage tracking
- Cost estimation
- Response quality voting
- Export results to CSV

→ Model Testing Playground summary:
✓ URL: https://ai.jglaine.com/playground
✓ Status: Running
✓ Connected providers: 5 (18 total models)
✓ Features: 6 enabled

═══

✅ OPTIONAL SERVICES INSTALLATION COMPLETE

═══

Installation Summary:

──

New Ollama Models:
✓ codellama:7b (3.8 GB) - Code generation
✓ deepseek-coder:6.7b (3.7 GB) - Code analysis
✓ nomic-embed-text (0.5 GB) - Fast embeddings
Total models: 6 (was 3)
Total VRAM: 22.9 GB / 24 GB (95%)

Custom LiteLLM Routes:
✓ Mistral AI integration (2 models)
✓ Cost tracking enabled ($100/month budget)
✓ Budget alerts configured (4 levels)
✓ Grafana cost dashboard created

Development Tools:
✓ Jupyter Lab: https://ai.jglaine.com/jupyter
✓ Code-Server (VS Code): https://ai.jglaine.com/code
✓ Model Testing Playground: https://ai.jglaine.com/playground
Total new services: 3

System Status After Installation:

──

Services Running:
Core services: 17/17 ✓
Optional services: 3/3 ✓
Total: 20/20 ✓

Resource Usage:
CPU: 27% (avg)
RAM: 31.4 GB / 64 GB (49%)
GPU VRAM: 22.9 GB / 24 GB (95%) ⚠️
Disk (DATA_ROOT): 58.7 GB used, 1.13 TB free

Network:
All routes responding: ✓
SSL certificates valid: ✓
Tailscale VPN active: ✓

Performance:
Ollama avg response time: 287ms (slightly slower due to high VRAM)
LiteLLM routing latency: 12ms
All services responding: <500ms

Recommendations:
⚠️ VRAM usage at 95% - consider:
1. Unloading unused models (ollama unload <model>)
2. Using smaller quantized models (q4 or q5)
3. Upgrading GPU (48 GB VRAM recommended for 6+ models)
✓ All other metrics: Healthy

══

📝 NEXT STEPS

══

Your AI platform is fully configured with optional services!

Access your new tools:
- Jupyter Lab: https://ai.jglaine.com/jupyter
- VS Code: https://ai.jglaine.com/code
- Model Playground: https://ai.jglaine.com/playground

Documentation:
- Review: /home/jglaine/ai-platform/docs/ADVANCED.md
- Cost tracking: https://ai.jglaine.com/grafana/d/cost-tracking
- Sample notebooks: /mnt/data/ai-platform/jupyter/

Monitor VRAM usage:
- Grafana: https://ai.jglaine.com/grafana
- CLI: nvidia-smi -l 5
- Unload models: docker exec ollama ollama unload <model>

═══

✨ Optional services installation completed successfully!

═══

Logs saved to: /mnt/data/ai-platform/logs/add-services-20250207-162015.log

═══

📊 TROUBLESHOOTING GUIDE

Common Issues and Solutions

1. Service Won't Start

Symptom: Container shows "Unhealthy" or "Restarting" status

```
# Check logs
docker logs <container-name>

# Common causes and fixes:

# A. Port already in use
```

```
sudo netstat -tulpn | grep <port>
# Solution: Change port in docker-compose.yml or kill conflicting process

# B. Volume permission errors
sudo chown -R $USER:$USER /mnt/data/ai-platform/volumes/<service>
docker restart <container-name>

# C. Insufficient memory
docker stats
# Solution: Increase Docker memory limit or reduce number of services

# D. Configuration file error
docker exec <container-name> cat /path/to/config.yml
# Validate YAML syntax, fix errors, restart container
```

2. Ollama Out of Memory (OOM)

Symptom: Ollama crashes, CUDA out of memory errors

```
# Check current VRAM usage
nvidia-smi

# Solution 1: Unload unused models
docker exec ollama ollama list
docker exec ollama ollama unload mistral:latest

# Solution 2: Use smaller quantized models
docker exec ollama ollama pull llama3.2:7b-q4_K_M  # Q4 quantization
docker exec ollama ollama rm llama3.2:latest       # Remove unquantized

# Solution 3: Limit concurrent model loading
# Edit deployment/configs/ollama.env:
OLLAMA_MAX_LOADED_MODELS=2
docker restart ollama

# Solution 4: Reduce model context size
# When calling Ollama API:
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Hello",
  "options": {
    "num_ctx": 2048  # Reduce from 4096
  }
}'
```

3. LiteLLM Routing Failures

Symptom: Queries fail, "No available models" error

```
# Check LiteLLM logs
docker logs litellm

# Verify routing configuration
docker exec litellm cat /app/config.yml

# Test specific provider
docker exec -it litellm bash
curl -X POST http://localhost:8000/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "ollama/llama3.2",
    "messages": [{"role": "user", "content": "test"}]
  }'

# Common fixes:

# A. External API key invalid
# Re-enter API key in Script 1 or edit .secrets/api_keys.enc

# B. Ollama unreachable from LiteLLM
docker exec litellm ping ollama  # Should succeed
# Fix: Ensure both on same Docker network

# C. Model not registered in LiteLLM
# Edit deployment/configs/litellm_config.yml, add model, restart

# D. Rate limit hit on external API
# Check cost tracking dashboard, wait for rate limit reset
# Or enable fallback to local models
```

4. Google Drive Sync Not Working

Symptom: No files downloading, "Invalid credentials" error

```
# Check token file exists
ls -lh /home/jglaine/ai-platform/deployment/.secrets/gdrive_token.json
```

```
# Verify token is valid
docker exec anything-llm cat /app/storage/.secrets/gdrive_token.json

# Re-authenticate (if token expired)
# Re-run Script 3, Phase 9 (Google Drive Configuration)

# Manual sync test
docker exec anything-llm python3 /app/scripts/gdrive_sync.py --test

# Check sync logs
docker logs anything-llm | grep gdrive

# Common fixes:

# A. Token expired (tokens expire after 7 days of inactivity)
# Solution: Re-run OAuth flow in Script 3

# B. Drive API not enabled
# Solution: Visit https://console.cloud.google.com/apis/library/drive.googleapis.com
# Click "Enable API"

# C. Insufficient permissions
# Solution: Ensure OAuth consent screen includes drive.readonly scope

# D. Firewall blocking outbound HTTPS
# Solution: Allow outbound HTTPS (port 443) to *.googleapis.com
```

5. SSL Certificate Issues

Symptom: "Certificate not trusted" warnings, HTTPS errors

```
# Check Caddy status
docker exec caddy caddy version
docker logs caddy | grep "certificate"

# Verify DNS propagation
dig ai.jglaine.com +short
# Should return your server's public IP

# Force certificate renewal
docker exec caddy caddy reload --config /etc/caddy/Caddyfile

# Check Let's Encrypt rate limits
# Visit: https://letsencrypt.org/docs/rate-limits/
```

```
# Max 5 certificates per week per domain

# Common fixes:

# A. Domain not resolving
# Solution: Update DNS A record, wait for propagation (up to 48h)

# B. Port 80/443 blocked
sudo netstat -tulpn | grep :80
sudo netstat -tulpn | grep :443
# Solution: Open ports in firewall

# C. Rate limit hit
# Solution: Wait 7 days, or use staging environment:
# Edit deployment/stack/docker-compose.yml:
# caddy:
#   command: caddy run --config /etc/caddy/Caddyfile --adapter caddyfile --debug

# D. Certificate files corrupted
# Solution: Remove and regenerate
docker exec caddy rm -rf /data/caddy/certificates
docker restart caddy
```

6. High CPU/Memory Usage

Symptom: System sluggish, services slow to respond

```
# Identify resource-hungry containers
docker stats --no-stream

# Check system resources
htop
free -h
df -h

# Common culprits and fixes:

# A. Too many Ollama models loaded
docker exec ollama ollama list
docker exec ollama ollama unload <unused-model>

# B. n8n workflows running too frequently
# Access n8n UI, edit workflow cron schedules
```

```
# C. Prometheus storing too much data
# Reduce retention period:
# Edit deployment/stack/docker-compose.yml:
# prometheus:
#   command:
#     - '--storage.tsdb.retention.time=7d'  # Change from 15d

# D. Vector database indexing
# Check Qdrant logs
docker logs qdrant
# If indexing, wait for completion (can take 10-30 minutes for large datasets)

# E. Log files filling disk
du -sh /mnt/data/ai-platform/logs/*
# Rotate logs:
find /mnt/data/ai-platform/logs/ -name "*.log" -mtime +7 -delete

# F. Docker image cache
docker system df
docker system prune -a --volumes  # WARNING: Removes unused images/volumes
```

7. Anything LLM Document Ingestion Failing

Symptom: Documents not searchable, embedding errors

```
# Check Anything LLM logs
docker logs anything-llm | grep -i error

# Verify Qdrant connection
docker exec anything-llm curl http://qdrant:6333/health
# Should return: {"status":"ok"}

# Check collection exists
docker exec qdrant curl http://localhost:6333/collections
# Should list: anything_llm_collection

# Manual ingestion test
docker exec -it anything-llm bash
curl http://localhost:3001/api/v1/admin/system/sync-documents \
  -H "Authorization: Bearer <admin-token>"

# Common fixes:

# A. Qdrant collection not created
```

```
docker exec qdrant curl -X PUT http://localhost:6333/collections/anything_llm_collection \
  -H "Content-Type: application/json" \
  -d '{
    "vectors": {
      "size": 768,
      "distance": "Cosine"
    }
  }'

# B. Embedding model not loaded
docker exec ollama ollama pull nomic-embed-text
docker restart anything-llm

# C. File format not supported
# Anything LLM supports: PDF, DOCX, TXT, MD, HTML, CSV
# Convert unsupported files or add custom parser

# D. Document too large
# Split large documents:
split -b 10M large_doc.pdf large_doc_part_
# Ingest parts separately

# E. Memory insufficient for embedding
# Reduce batch size:
# Edit deployment/configs/anything_llm.env:
EMBEDDING_BATCH_SIZE=5  # Reduce from 10
```

8. OpenClaw Not Accessible

Symptom: Cannot reach http://100.x.x.x:18789

```
# Verify Tailscale is running
sudo tailscale status

# Check Tailscale IP assignment
ip addr show tailscale0
# Should show: inet 100.x.x.x/32

# Verify OpenClaw container running
docker ps | grep openclaw

# Test from Tailscale device
curl http://<tailscale-ip>:18789/health
# Should return: {"status":"ok"}
```

```
# Common fixes:

# A. Tailscale not authenticated
sudo tailscale up --authkey=<your-auth-key>

# B. OpenClaw container not exposed on Tailscale interface
# Edit deployment/stack/docker-compose.yml:
# openclaw:
#   ports:
#     - "100.x.x.x:18789:18789"  # Replace with your Tailscale IP
docker-compose up -d openclaw

# C. Firewall blocking Tailscale
sudo ufw allow in on tailscale0
sudo ufw reload

# D. OpenClaw crashed
docker logs openclaw
docker restart openclaw
```

9. Backup Script Failing

Symptom: Cron job not running, backups not created

```
# Check cron job exists
crontab -l | grep backup

# Check backup script permissions
ls -lh /home/jglaine/ai-platform/scripts/backup.sh
chmod +x /home/jglaine/ai-platform/scripts/backup.sh

# Run manually to see errors
bash /home/jglaine/ai-platform/scripts/backup.sh

# Check backup logs
cat /mnt/data/ai-platform/logs/backup-*.log

# Common fixes:

# A. PostgreSQL dump failing
docker exec postgres pg_dumpall -U postgres
# If error, check PostgreSQL logs:
docker logs postgres
```

```
# B. Disk space insufficient
df -h /mnt/data
# Free up space or increase disk size

# C. GPG key not configured (for encrypted backups)
gpg --list-keys
# If empty, generate key:
gpg --gen-key

# D. Permissions issue
sudo chown -R $USER:$USER /mnt/data/ai-platform/backups/
sudo chmod 700 /mnt/data/ai-platform/backups/
```

10. Performance Degradation Over Time

Symptom: Queries getting slower, system less responsive

```
# Check Grafana performance dashboard
# Visit: https://ai.jglaine.com/grafana

# Identify bottlenecks:

# A. Database bloat
docker exec postgres psql -U postgres -c "SELECT
pg_size_pretty(pg_database_size('ai_platform_db'));"
# If >5GB, consider vacuuming:
docker exec postgres psql -U postgres -d ai_platform_db -c "VACUUM FULL;"

# B. Vector database needs optimization
docker exec qdrant curl -X POST
http://localhost:6333/collections/anything_llm_collection/optimize

# C. Redis memory full
docker exec redis redis-cli INFO memory
# If used_memory > 80%, clear old caches:
docker exec redis redis-cli FLUSHDB

# D. Too many Docker containers
docker ps -a | wc -l
# Remove stopped containers:
docker container prune

# E. Log files growing
```

```
du -sh /mnt/data/ai-platform/logs/
# Implement log rotation (see backup script)

# F. Model context caching
# Ollama caches prompts for performance
# If cache corrupted, clear:
docker exec ollama rm -rf /root/.ollama/cache/
docker restart ollama
```

🔧 MAINTENANCE & OPERATIONS

Daily Operations

1. System Health Check

```
# Quick health check script
cat > /home/jglaine/ai-platform/scripts/health-check.sh << 'EOF'
#!/bin/bash
echo "=== AI Platform Health Check ==="
echo "Date: $(date)"
echo ""

# Service status
echo "[1/5] Service Status"
docker ps --format "table {{.Names}}\t{{.Status}}" | grep -E "ollama|litellm|caddy|qdrant"

# Resource usage
echo ""
echo "[2/5] Resource Usage"
echo "CPU: $(top -bn1 | grep "Cpu(s)" | awk '{print $2}' | cut -d'%' -f1)%"
echo "RAM: $(free -h | grep Mem | awk '{print $3 "/" $2}')"
echo "GPU VRAM: $(nvidia-smi --query-gpu=memory.used,memory.total
--format=csv,noheader,nounits | awk '{print $1 " GB / " $3 " GB"}')"

# Disk space
echo ""
echo "[3/5] Disk Space"
df -h /mnt/data | tail -1 | awk '{print $3 " used / " $2 " total (" $5 " utilization)"}'

# Recent errors
echo ""
echo "[4/5] Recent Errors (last hour)"
```

```
find /mnt/data/ai-platform/logs/ -name "*.log" -mmin -60 -exec grep -i "error" {} \; | wc -l

# Backup status
echo ""
echo "[5/5] Backup Status"
ls -lh /mnt/data/ai-platform/backups/ | tail -3

echo ""
echo "=== Health Check Complete ==="
EOF

chmod +x /home/jglaine/ai-platform/scripts/health-check.sh

# Run daily check
bash /home/jglaine/ai-platform/scripts/health-check.sh
```

2. Monitor API Costs

```
# Daily cost check
curl -s https://ai.jglaine.com/grafana/api/dashboards/uid/cost-tracking \
  -H "Authorization: Bearer <grafana-api-key>" \
  | jq '.dashboard.panels[] | select(.title == "Current Month Spending") | .targets[0].expr'

# Or visit Grafana directly:
# https://ai.jglaine.com/grafana/d/cost-tracking
```

3. Review Query Logs

```
# Analyze query patterns
docker logs litellm --since 24h | grep "POST /chat/completions" | wc -l

# Most used models
docker logs litellm --since 24h | grep "model:" | sort | uniq -c | sort -rn | head -10

# Average response time
docker logs ollama --since 24h | grep "response_time" | awk '{sum+=$NF; count++} END {print
sum/count "ms"}'
```

Weekly Maintenance

1. Update Docker Images

```
# Check for updates
docker images | grep "jan/\|bitnami/\|grafana/\|prom/"
```

```
# Update script
cat > /home/jglaine/ai-platform/scripts/update-images.sh << 'EOF'
#!/bin/bash
set -e

echo "=== Updating Docker Images ==="

# Pull latest images
docker-compose -f /home/jglaine/ai-platform/deployment/stack/docker-compose.yml pull

# Recreate containers with new images
docker-compose -f /home/jglaine/ai-platform/deployment/stack/docker-compose.yml up -d

# Remove old images
docker image prune -a -f

echo "=== Update Complete ==="
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Status}}"
EOF

chmod +x /home/jglaine/ai-platform/scripts/update-images.sh

# Run weekly (Sundays at 3 AM)
crontab -l | grep -v "update-images" | crontab -
(crontab -l 2>/dev/null; echo "0 3 * * 0 /home/jglaine/ai-platform/scripts/update-images.sh") |
crontab -
```

2. Optimize Databases

```
# PostgreSQL optimization
docker exec postgres psql -U postgres << 'EOF'
VACUUM ANALYZE;
REINDEX DATABASE ai_platform_db;
REINDEX DATABASE dify_db;
REINDEX DATABASE anything_llm_db;
REINDEX DATABASE n8n_db;
EOF

# Qdrant optimization
docker exec qdrant curl -X POST
http://localhost:6333/collections/anything_llm_collection/optimize

# Redis cleanup
```

```
docker exec redis redis-cli << 'EOF'
MEMORY PURGE
BGSAVE
EOF
```

3. Review Security Logs

```
# Check failed authentication attempts
docker logs caddy --since 168h | grep "401\|403" | wc -l

# Review Tailscale connection logs
sudo tailscale status --json | jq '.Peer[] | {name: .HostName, lastSeen: .LastSeen}'

# Check for suspicious API usage patterns
docker logs litellm --since 168h | grep -E "429|rate_limit" | wc -l
```

4. Test Backups

```
# Test restore procedure (monthly recommended)
cat > /home/jglaine/ai-platform/scripts/test-restore.sh << 'EOF'
#!/bin/bash
set -e

echo "=== Testing Backup Restore ==="

# Find latest backup
LATEST_BACKUP=$(ls -t /mnt/data/ai-platform/backups/postgres-*.sql.gz | head -1)
echo "Testing restore of: $LATEST_BACKUP"

# Create test database
docker exec postgres psql -U postgres -c "DROP DATABASE IF EXISTS test_restore_db;"
docker exec postgres psql -U postgres -c "CREATE DATABASE test_restore_db;"

# Restore to test database
gunzip -c $LATEST_BACKUP | docker exec -i postgres psql -U postgres -d test_restore_db

# Verify restoration
TABLES=$(docker exec postgres psql -U postgres -d test_restore_db -t -c "SELECT COUNT(*)
FROM information_schema.tables WHERE table_schema='public';")

if [ "$TABLES" -gt 0 ]; then
  echo "✓ Restore successful: $TABLES tables found"
else
  echo "✗ Restore failed: No tables found"
```

```
  exit 1
fi

# Cleanup
docker exec postgres psql -U postgres -c "DROP DATABASE test_restore_db;"

echo "=== Test Complete ==="
EOF

chmod +x /home/jglaine/ai-platform/scripts/test-restore.sh
```

Monthly Maintenance

1. Security Updates

```
# System updates
sudo apt update && sudo apt upgrade -y

# Restart services if kernel updated
sudo needrestart -r a

# Update Docker Engine
sudo apt install docker-ce docker-ce-cli containerd.io

# Update Docker Compose
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

2. Certificate Renewal

```
# Let's Encrypt certificates auto-renew via Caddy
# Verify renewal working:
docker logs caddy | grep "certificate renewed"

# Manual renewal if needed:
docker exec caddy caddy reload --config /etc/caddy/Caddyfile
```

3. Audit API Keys

```
# List all API keys (redacted)
cat /home/jglaine/ai-platform/deployment/.secrets/.env | grep API_KEY | sed
's/=.*/=***REDACTED***/'
```

```
# Rotate keys quarterly:
# 1. Generate new keys in respective provider dashboards
# 2. Update .secrets/.env
# 3. Restart affected services
# 4. Test connectivity
# 5. Revoke old keys
```

4. Capacity Planning

```
# Generate capacity report
cat > /home/jglaine/ai-platform/scripts/capacity-report.sh << 'EOF'
#!/bin/bash
echo "=== Capacity Planning Report ==="
echo "Generated: $(date)"
echo ""

# Disk growth rate (GB per month)
echo "[1] Disk Usage Trend"
CURRENT_SIZE=$(du -sb /mnt/data/ai-platform | awk '{print $1}')
BACKUP_SIZE=$(du -sb /mnt/data/ai-platform/backups | awk '{print $1}')
echo "Current usage: $(numfmt --to=iec-i --suffix=B $CURRENT_SIZE)"
echo "Backup size: $(numfmt --to=iec-i --suffix=B $BACKUP_SIZE)"
echo "Growth rate: ~$(echo "scale=2; $CURRENT_SIZE / 30 / 1024 / 1024 / 1024" | bc)
GB/day"

# Request volume trend
echo ""
echo "[2] Request Volume"
docker logs litellm --since 720h 2>/dev/null | grep "POST /chat/completions" | wc -l | awk '{print
"Last 30 days: " $1 " requests"}'

# VRAM usage trend
echo ""
echo "[3] Average VRAM Usage"
echo "Current: $(nvidia-smi --query-gpu=memory.used --format=csv,noheader,nounits) GB"
echo "Peak: Check Grafana for historical data"

# Database size trend
echo ""
echo "[4] Database Growth"
docker exec postgres psql -U postgres -t -c "SELECT pg_database.datname,
pg_size_pretty(pg_database_size(pg_database.datname)) FROM pg_database ORDER BY
pg_database_size(pg_database.datname) DESC;"
```

```
echo ""
echo "=== Report Complete ==="
EOF

chmod +x /home/jglaine/ai-platform/scripts/capacity-report.sh
```

Quarterly Maintenance

1. Disaster Recovery Drill

```
# Simulate complete system failure and restore
# Document each step and time taken
# Update disaster recovery plan based on findings
```

2. Performance Baseline Update

```
# Run comprehensive benchmarks
# Compare against previous quarter
# Identify performance regressions
# Update capacity plan
```

3. Security Audit

```
# Review access logs
# Update passwords and API keys
# Review firewall rules
# Scan for vulnerabilities
# Update security documentation
```

🔒 SECURITY CONSIDERATIONS

Network Security

1. Firewall Configuration

```
# UFW (Uncomplicated Firewall) setup
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Allow SSH (change 22 to your custom port)
sudo ufw allow 22/tcp
```

```
# Allow HTTP/HTTPS (for Caddy)
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Allow Tailscale
sudo ufw allow in on tailscale0
sudo ufw allow 41641/udp  # Tailscale control plane

# Deny direct access to internal services
# (Only accessible via Caddy reverse proxy or Tailscale)
sudo ufw deny 11434/tcp  # Ollama
sudo ufw deny 8000/tcp   # LiteLLM
sudo ufw deny 6333/tcp   # Qdrant
sudo ufw deny 5432/tcp   # PostgreSQL
sudo ufw deny 6379/tcp   # Redis

# Enable firewall
sudo ufw enable
sudo ufw status verbose
```

2. TLS/SSL Best Practices

```
# Caddy automatically:
# - Obtains certificates from Let's Encrypt
# - Enables HTTPS by default
# - Enforces TLS 1.2+
# - Uses secure cipher suites

# Verify SSL configuration
curl -I https://ai.jglaine.com | grep -i "strict-transport-security"
# Should show: Strict-Transport-Security: max-age=31536000

# Test SSL rating
# Visit: https://www.ssllabs.com/ssltest/analyze.html?d=ai.jglaine.com
# Target: A or A+ rating
```

3. Tailscale VPN Hardening

```
# Enable key expiry (force re-authentication every 90 days)
sudo tailscale up --ssh --force-reauth --auth-key-expiry=90d

# Enable MagicDNS for easy service discovery
sudo tailscale up --accept-dns=true
```

```
# Disable Tailscale SSH if not needed
sudo tailscale up --ssh=false

# Restrict access to specific users/devices
# In Tailscale admin console:
# - Enable ACLs (Access Control Lists)
# - Define rules like:
{
  "acls": [
    {
      "action": "accept",
      "src": ["user@example.com"],
      "dst": ["tag:ai-platform:*"]
    }
  ]
}
```

Authentication & Authorization

1. API Key Management

```
# Store API keys encrypted
cat > /home/jglaine/ai-platform/scripts/manage-secrets.sh << 'EOF'
#!/bin/bash

encrypt_secret() {
  echo "$1" | gpg --encrypt --recipient admin@jglaine.com --armor
}

decrypt_secret() {
  echo "$1" | gpg --decrypt --armor
}

rotate_key() {
  KEY_NAME=$1
  NEW_VALUE=$2

  # Backup old value
  OLD_VALUE=$(grep "$KEY_NAME=" .secrets/.env | cut -d'=' -f2)
  echo "$KEY_NAME=$OLD_VALUE" >> .secrets/.env.backup.$(date +%Y%m%d)

  # Update with new value
  sed -i "s|$KEY_NAME=.*|$KEY_NAME=$NEW_VALUE|" .secrets/.env
```

```
  echo "✓ Rotated $KEY_NAME"
}

case "$1" in
  encrypt)
    encrypt_secret "$2"
    ;;
  decrypt)
    decrypt_secret "$2"
    ;;
  rotate)
    rotate_key "$2" "$3"
    ;;
  *)
    echo "Usage: $0 {encrypt|decrypt|rotate} <args>"
    exit 1
    ;;
esac
EOF

chmod +x /home/jglaine/ai-platform/scripts/manage-secrets.sh
```

2. Service Authentication

```
# Open WebUI: Password-based (change default)
# Dify: Password-based (change default)
# Anything LLM: Password-based with API tokens
# n8n: Password-based with API keys for webhooks
# Flowise: API key authentication
# Grafana: Admin password + viewer role

# Enforce strong passwords policy:
# - Minimum 16 characters
# - Mix of uppercase, lowercase, numbers, symbols
# - No dictionary words
# - Use password manager (e.g., Bitwarden)
```

3. Multi-Factor Authentication (MFA)

```
# Enable MFA for critical services:

# Grafana MFA (TOTP)
# 1. Login to Grafana as admin
```

```
# 2. User Settings → Authentication → Setup MFA
# 3. Scan QR code with authenticator app

# n8n MFA (optional, via reverse proxy)
# Add OAuth2 proxy in Caddy:
ai.jglaine.com {
  route /n8n/* {
    forward_auth oauth2-proxy:4180 {
      uri /oauth2/auth
      copy_headers X-Auth-Request-User X-Auth-Request-Email
    }
    reverse_proxy n8n:5678
  }
}
```

Data Security

1. Encryption at Rest

```
# Encrypt sensitive volumes
# Option A: LUKS full-disk encryption (recommended for /mnt/data)
sudo cryptsetup luksFormat /dev/sdb
sudo cryptsetup luksOpen /dev/sdb encrypted_data
sudo mkfs.ext4 /dev/mapper/encrypted_data
sudo mount /dev/mapper/encrypted_data /mnt/data

# Option B: Encrypted container for secrets
# Already implemented in deployment scripts (GPG)
```

2. Encryption in Transit

```
# All external traffic: HTTPS (via Caddy)
# Internal service-to-service: Docker network (isolated)
# Tailscale: WireGuard encryption (automatically enabled)

# Verify internal traffic not exposed:
sudo netstat -tulpn | grep -E "11434|8000|6333|5432|6379"
# Should only show 127.0.0.1 or Docker bridge IP
```

3. Backup Encryption

```
# Backups encrypted with GPG (already configured)
# Verify encryption:
file /mnt/data/ai-platform/backups/secrets-*.tar.gz.gpg
```

```
# Should show: GPG encrypted data

# Test decryption:
gpg --decrypt /mnt/data/ai-platform/backups/secrets-20250207.tar.gz.gpg | tar -tzf - | head
# Should list files after entering passphrase
```

Compliance & Privacy

1. GDPR Compliance

```
# Data retention policy
# - User data: Configurable in each service
# - Logs: 30 days (automated rotation)
# - Backups: 7 days (automated deletion)
# - Vector database: Manual deletion on request

# Data deletion script
cat > /home/jglaine/ai-platform/scripts/delete-user-data.sh << 'EOF'
#!/bin/bash
USER_EMAIL=$1

echo "=== Deleting data for $USER_EMAIL ==="

# Delete from PostgreSQL
docker exec postgres psql -U postgres -d ai_platform_db << SQL
DELETE FROM users WHERE email='$USER_EMAIL';
DELETE FROM conversations WHERE user_email='$USER_EMAIL';
SQL

# Delete from Qdrant (if user-specific collections exist)
docker exec qdrant curl -X DELETE
http://localhost:6333/collections/user_${USER_EMAIL//[@.]/_}

# Delete from logs
find /mnt/data/ai-platform/logs/ -type f -exec sed -i "/$USER_EMAIL/d" {} \;

echo "=== Deletion complete ==="
echo "⚠️ Manual verification required in:"
echo "  - Dify UI"
echo "  - Anything LLM UI"
echo "  - n8n workflows"
EOF

chmod +x /home/jglaine/ai-platform/scripts/delete-user-data.sh
```

2. Data Privacy

```
# Local-first approach: All sensitive data stays on your infrastructure
# External API calls: Only for complex queries (can be disabled)
# Logging: Scrub sensitive data

# Configure log scrubbing in LiteLLM:
# Edit deployment/configs/litellm_config.yml:
litellm_settings:
  drop_params: true  # Don't log request content
  success_callback: null  # Disable external logging
  failure_callback: null
```

3. Audit Logging

```
# Enable comprehensive audit logs
# Track: Who accessed what, when, from where

# PostgreSQL audit logging
docker exec postgres psql -U postgres << 'EOF'
ALTER SYSTEM SET log_statement = 'all';
ALTER SYSTEM SET log_connections = 'on';
ALTER SYSTEM SET log_disconnections = 'on';
SELECT pg_reload_conf();
EOF

# Service-specific audit logs
# - Grafana: Already enabled (check Settings → Users → Activity Log)
# - n8n: Execution data stored in PostgreSQL
# - Dify: Activity logs in UI
```

Security Monitoring

1. Intrusion Detection

```
# Install fail2ban for SSH brute force protection
sudo apt install fail2ban -y

# Configure fail2ban
sudo cat > /etc/fail2ban/jail.local << 'EOF'
[DEFAULT]
bantime = 3600
findtime = 600
```

```
maxretry = 5

[sshd]
enabled = true
port = 22
logpath = /var/log/auth.log
EOF

sudo systemctl restart fail2ban
```

2. Vulnerability Scanning

```
# Scan Docker images for vulnerabilities
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
  aquasec/trivy:latest image ollama/ollama:latest

# Schedule weekly scans
crontab -l | grep -v "trivy" | crontab -
(crontab -l 2>/dev/null; echo "0 3 * * 0 /home/jglaine/ai-platform/scripts/security-scan.sh") |
crontab -
```

3. Security Alerts

```
# Already configured in Grafana (Script 3)
# Additional alerts:

# A. Unusual API access patterns
# - Spike in failed authentication attempts
# - Requests from unexpected IPs
# - High volume of requests from single user

# B. System security events
# - Unauthorized SSH attempts
# - File integrity changes in /etc or config directories
# - New Docker containers started outside deployment scripts

# C. Data exfiltration attempts
# - Unusually large data transfers
# - Bulk database queries
# - Multiple failed access attempts to sensitive endpoints
```

📝 CHANGELOG

Version 75.2.0 (2025-02-07)

Major Update: Complete Deployment Guide

Added

- **Script 0**: Pre-flight checks and cleanup
- **Script 1**: System setup and configuration
- **Script 2**: Core service deployment
- **Script 3**: Service configuration and integration testing
- **Script 4**: Optional services (models, tools, monitoring)
- **Comprehensive troubleshooting guide** (10 common issues)
- **Maintenance operations guide** (daily, weekly, monthly, quarterly)
- **Security considerations** (network, auth, data, compliance)
- **Architecture diagrams** and **service inventory**

Enhanced

- **Google Drive integration** with automatic document ingestion
- **LiteLLM routing** with fallback chains and load balancing
- **Cost tracking** with budget alerts and Grafana dashboard
- **Backup automation** with encryption and off-site options
- **Monitoring stack** with Prometheus + Grafana + custom dashboards
- **Development tools** (Jupyter Lab, VS Code, model playground)

Fixed

- **VRAM management** for multi-model loading
- **SSL certificate** auto-renewal via Caddy
- **Tailscale networking** for secure OpenClaw access
- **Database optimization** procedures

Security

- **Encrypted API keys** (GPG)
- **Encrypted backups** (GPG)
- **Firewall rules** (UFW)
- **TLS 1.2+ enforcement** (Caddy)
- **Fail2ban** for SSH protection
- **Vulnerability scanning** (Trivy)

Documentation

- **5 core docs** (111 KB total)
- **System inventory** (JSON)
- **4 usage examples**

- **Complete API reference**
- **Security best practices**

🎉 DEPLOYMENT GUIDE COMPLETE

What You've Built

A **production-ready, enterprise-grade AI platform** with:

✅ **17+ services** running in Docker
✅ **6 Ollama models** for local inference
✅ **5 AI applications** (Open WebUI, Dify, Anything LLM, n8n, Flowise)
✅ **Smart routing** (local/cloud hybrid with fallbacks)
✅ **Google Drive integration** (automatic document sync)
✅ **Vector database** (Qdrant with 1,847+ embeddings)
✅ **Monitoring stack** (Prometheus + Grafana with alerts)
✅ **Secure networking** (Tailscale VPN + SSL via Caddy)
✅ **Automated backups** (daily, encrypted, 7-day retention)
✅ **Development tools** (Jupyter Lab, VS Code, model playground)
✅ **Cost tracking** ($100/month budget with alerts)
✅ **Comprehensive documentation** (120+ KB)

Total Setup Time

- **Script 0** (Cleanup): ~2 minutes
- **Script 1** (Setup): ~15 minutes
- **Script 2** (Deployment): ~12 minutes (45 minutes with model downloads)
- **Script 3** (Configuration): ~15 minutes
- **Script 4** (Optional): ~30 minutes (varies by selections)

Total: ~1-2 hours (including user inputs and initial model downloads)