SFG Aluminium Ecosystem Architecture

Version: 1.0
Date: November 3, 2025
Status: In Development

Overview

The SFG Aluminium Ecosystem is a zero-drift orchestration system connecting 51+ applications through a central conductor (Nexus) and satellite apps.

System Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                         NEXUS (Conductor)                         │
│                                                                   │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────────────┐  │
│  │  Memory      │  │  App         │  │  MCP Client          │  │
│  │  Manager     │  │  Registry    │  │  (Orchestration)     │  │
│  └──────────────┘  └──────────────┘  └──────────────────────┘  │
│                                                                   │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │  PostgreSQL Database                                      │   │
│  │  - Conversations  - Plans  - Decisions  - App Registry   │   │
│  └──────────────────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────────────┘
                              │
                              │ MCP Protocol
                              │
        ┌─────────────────────┼─────────────────────┐
        │                     │                     │
        ▼                     ▼                     ▼
┌───────────────┐     ┌───────────────┐     ┌───────────────┐
│  Satellite    │     │  Satellite    │     │  Satellite    │
│  App 1        │     │  App 2        │     │  App 3        │
│               │     │               │     │               │
│  MCP Server   │     │  MCP Server   │     │  MCP Server   │
└───────────────┘     └───────────────┘     └───────────────┘
```

Core Components

1. Nexus (Central Conductor)

- **Role:** Orchestrate all satellite apps
- **Technology:** Next.js 14, TypeScript, PostgreSQL
- **Capabilities:**
- Persistent memory (conversations, plans, decisions)
- App registry (track all 51 apps)
- MCP client (connect to satellites)
- Gap analysis (identify missing capabilities)

- Instruction distribution

2. Satellite Applications

- **Count:** 51 apps (1 active, 50 placeholders)
- **Categories:**
- Core (manufacturing, operations)
- Support (HR, finance, admin)
- Experimental (AI, automation)
- **Capabilities:**
- Self-registration
- MCP server implementation
- Webhook receivers
- Business logic extraction

3. GitHub Repository (Truth Source)

- **URL:** https://github.com/sfgaluminium1-spec/sfg-app-portfolio
- **Purpose:** Central source of truth
- **Contents:**
- App registry (JSON files)
- App backups (source code)
- Instructions (for Nexus and satellites)
- Documentation (architecture, specs)
- Analysis (inventory, gaps, ROI)

Communication Protocols

MCP (Model Context Protocol)

- **Purpose:** Bidirectional communication between Nexus and satellites
- **Flow:**
  1. Nexus connects to satellite via MCP
  2. Satellite exposes tools/resources
  3. Nexus invokes tools to orchestrate
  4. Satellite executes and reports back

GitHub Webhooks

- **Purpose:** Notify satellites of new instructions
- **Flow:**
  1. Nexus pushes instruction to GitHub
  2. GitHub webhook triggers satellite endpoint
  3. Satellite pulls and implements instruction
  4. Satellite reports completion via GitHub issue

REST APIs

- **Purpose:** Direct app-to-app communication
- **Use Cases:**
- Data synchronization
- Real-time updates

- Health checks

Data Flow

Registration Flow

1. Satellite clones repository
2. Satellite creates registration JSON
3. Satellite backs up code
4. Satellite extracts business logic
5. Satellite pushes to GitHub
6. Satellite creates notification issue
7. Nexus processes registration
8. Nexus adds to app registry

Orchestration Flow

1. Warren gives instruction to Nexus
2. Nexus analyzes requirement
3. Nexus identifies relevant satellites
4. Nexus creates instruction files
5. Nexus pushes to GitHub
6. Webhooks notify satellites
7. Satellites implement instructions
8. Satellites report completion
9. Nexus verifies and updates registry

Security

Repository Access

- **Visibility:** Private
- **Access Control:**
- Warren: Admin (full access)
- Nexus: Write (via PAT)
- Satellites: Write (via machine user)

Authentication

- **GitHub:** Personal Access Tokens (PAT)
- **MCP:** Token-based authentication
- **APIs:** API keys + JWT

Data Protection

- **Encryption:** TLS 1.3 for all communications
- **Secrets:** Environment variables, never in code
- **Backups:** Daily automated backups

Scalability

Current Capacity

- 51 apps (1 active, 50 ready)
- 1 conductor (Nexus)
- 1 repository (monorepo)

Growth Plan

- Add apps incrementally (10 per week)
- Scale Nexus horizontally if needed
- Partition repository if >100 apps

Monitoring

Health Checks

- Nexus: `/api/health` endpoint
- Satellites: `/api/health` endpoint
- GitHub: Repository insights

Metrics

- App registration rate
- Instruction completion rate
- Error rate
- Response time

Disaster Recovery

Backup Strategy

- **Code:** GitHub (version controlled)
- **Database:** Daily PostgreSQL dumps
- **Secrets:** Encrypted backup in secure storage

Recovery Plan

1. Restore repository from GitHub
2. Restore database from latest dump
3. Redeploy Nexus
4. Verify satellite connections
5. Resume orchestration

Next Steps

1. Implement persistent memory in Nexus
2. Register first 10 satellite apps
3. Implement MCP protocol
4. Deploy orchestration workflows
5. Monitor and optimize