IDTF 技術白皮書 (第二版)

Industrial Digital Twins Framework - Technical Whitepaper

版本: 2.0
日期: 2025 年 1 月
作者: IDTF 技術團隊

第五部分:實施指南

16. 實施路線圖

16.1 四階段實施路線圖

Phase 1: 基礎建設 (2-3個月)

目標: 建立 IDTF 核心基礎設施

工作內容: 1. **硬體準備**: - 伺服器 (Kafka, InfluxDB, PostgreSQL) - Omniverse Nucleus Server - 網路基礎設施

1. **軟體安裝**:
2. Docker/Kubernetes
3. Kafka Cluster
4. InfluxDB Cluster
5. PostgreSQL
6. Omniverse Nucleus
7. **基礎 Connector 開發**:
8. OPC UA Connector
9. Modbus Connector

10. MQTT Connector

交付物: - IDTF 核心平台運行 - 基礎 Connector 可用 - 文件和培訓材料

投資: $50K

Phase 2: 試點驗證 (3-4個月)

目標: 在 1-2條產線試點,驗證價值

工作內容: 1. **資產定義**: - 建立 10-20個設備的 IADL 定義 - 建立 Asset Template

1. **資料整合**:
2. 連接 SCADA/PLC
3. 收集即時資料
4. 儲存歷史資料
5. **3D 數位分身**:
6. 建立試點產線 3D 模型
7. 同步即時資料到 Omniverse
8. **應用開發**:
9. 即時監控 Dashboard
10. 基礎 AI 模型 (預測性維護)

交付物: - 試點產線數位分身 - 即時監控 Dashboard - 預測性維護 POC - ROI 驗證報告

投資: $40K

Phase 3: 全廠擴展 (6-9個月)

目標: 擴展到全廠所有設備

工作內容: 1. **全廠資產定義**: - 建立 800+設備的 IADL 定義 - 完善 Asset Template Library

1. **全廠資料整合**:
2. 連接所有 SCADA/PLC
3. 整合 ERP/MES

4. **全廠 3D 數位分身**:
5. 建立全廠 3D 模型
6. 同步所有設備即時資料
7. **進階應用**:
8. 品質異常檢測
9. 製程優化
10. 能源優化

交付物: - 全廠數位分身 - 完整的監控和分析平台 - 多個 AI/ML 應用 - 培訓和文件

投資: $50K

Phase 4: 多廠擴展 (可選)

目標: 擴展到其他工廠 (如廈門廠)

工作內容: 1. **IADL 複製**: - 複製母廠 IADL 定義到新廠

1. **虛擬調試**:
2. 在 Omniverse 中虛擬調試新廠
3. **經驗傳遞**:
4. 母廠和新廠資料同步
5. 遠端協作和支援

交付物: - 新廠數位分身 - 經驗傳遞完成 - 新廠快速達到母廠水平

投資: $35K

16.2 時間表

```
Month 1-3:   Phase 1 (基礎建設)
             ├─ 硬體準備
             ├─ 軟體安裝
             └─ Connector 開發

Month 4-7:   Phase 2 (試點驗證)
             ├─ 資產定義
             ├─ 資料整合
             ├─ 3D 數位分身
             └─ 應用開發

Month 8-16:  Phase 3 (全廠擴展)
             ├─ 全廠資產定義
             ├─ 全廠資料整合
             ├─ 全廠 3D 數位分身
             └─ 進階應用

Month 17+:   Phase 4 (多廠擴展, 可選)
             ├─ IADL 複製
             ├─ 虛擬調試
             └─ 經驗傳遞
```

---

17. 技術實現

17.1 系統需求

17.1.1 硬體需求

IDTF Core Server: - CPU: 16 cores - RAM: 64 GB - Storage: 2 TB SSD - Network: 10 Gbps - 數量: 3台 (Kafka cluster)

Omniverse Nucleus Server: - CPU: 8 cores - RAM: 32 GB - Storage: 4 TB SSD (3D 模型儲存) - GPU: NVIDIA RTX 4000 (可選,用於渲染) - Network: 10 Gbps - 數量: 1台

Client Workstations (IADL Editor, Omniverse View): - CPU: 8 cores - RAM: 32 GB - GPU: NVIDIA RTX 4000 或更高 - Storage: 512 GB SSD - Network: 1 Gbps - 數量: 5-10台

總成本: $50K

17.1.2 軟體需求

開源軟體 (免費): - Ubuntu 22.04 LTS - Docker / Kubernetes - Apache Kafka - InfluxDB - PostgreSQL - MongoDB - Python 3.11 - Node.js

商業軟體: - NVIDIA Omniverse Enterprise: $9,000/年 (10 users) - Windows Server (可選): $1,000

總成本: $10K/年

17.2 安裝部署

17.2.1 Docker Compose 部署

```yaml
# docker-compose.yml
version: '3.8'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092

  influxdb:
    image: influxdb:latest
    ports:
      - "8086:8086"
    volumes:
      - influxdb-data:/var/lib/influxdb2

  postgres:
    image: postgres:latest
    environment:
      POSTGRES_PASSWORD: password
    volumes:
      - postgres-data:/var/lib/postgresql/data

  mongodb:
    image: mongo:latest
    volumes:
      - mongodb-data:/data/db

  ndh-api:
    build: ./ndh-api
    ports:
      - "8000:8000"
    depends_on:
      - kafka
      - influxdb
      - postgres
      - mongodb

volumes:
  influxdb-data:
  postgres-data:
  mongodb-data:
```

17.2.2 Kubernetes 部署 (生產環境)

```yaml
# kafka-deployment.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: kafka
spec:
  serviceName: kafka
  replicas: 3
  selector:
    matchLabels:
      app: kafka
  template:
    metadata:
      labels:
        app: kafka
    spec:
      containers:
      - name: kafka
        image: confluentinc/cp-kafka:latest
        ports:
        - containerPort: 9092
        env:
        - name: KAFKA_ZOOKEEPER_CONNECT
          value: "zookeeper:2181"
        volumeMounts:
        - name: kafka-data
          mountPath: /var/lib/kafka/data
  volumeClaimTemplates:
  - metadata:
      name: kafka-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 500Gi
```

---

18. 最佳實踐

18.1 IADL 最佳實踐

18.1.1 命名規範

設備 ID: - 格式: `{設備類型縮寫}-{編號}` - 範例: `DB-001` (固晶機 #1), `WB-002` (打線機 #2)

標籤命名: - 格式: `{設備ID}.{參數名稱}` - 範例: `DB-001.Temperature`, `DB-001.Pressure`

SCADA 路徑: - 格式: `{產線}.{設備ID}.{參數名稱}` - 範例: `Line1.DB001.Temperature`

18.1.2 參數單位

統一使用 SI 單位: - 溫度: °C - 壓力: Pa (或 kPa, MPa) - 長度: m (或 mm) - 時間: s (或 h) - 速度: m/s - 流量: m^3/s (或 L/min)

18.1.3 版本控制

使用 Git 管理 IADL 定義:

```
# 初始化 Git Repository
git init iadl-repo
cd iadl-repo

# 建立資產定義
mkdir assets
echo "..." > assets/DB-001.yaml

# 提交變更
git add assets/DB-001.yaml
git commit -m "Add DB-001 asset definition"

# 推送到遠端
git remote add origin https://github.com/company/iadl-repo.git
git push -u origin main
```

18.2 NDH 最佳實踐

18.2.1 Kafka Topic 設計

命名規範: - 格式: `{系統}.{資料類型}.{詳細資訊}` - 範例: `plc.tag_change.db001`, `mes.work_order.created`

Partition 設計: - 根據設備 ID 分區,確保同一設備的事件順序 - Partition 數量 = 設備數量 / 10 (約)

Retention 設計: - 即時資料: 7天 - 重要事件: 永久保留 - 日誌: 30天

18.2.2 資料品質

資料驗證:

```python
def validate_sensor_data(data):
    # 檢查必要欄位
    required_fields = ['equipment_id', 'timestamp', 'value']
    for field in required_fields:
        if field not in data:
            raise ValueError(f"Missing required field: {field}")

    # 檢查資料範圍
    if data['value'] < 0 or data['value'] > 1000:
        raise ValueError(f"Value out of range: {data['value']}")

    # 檢查時間戳
    if data['timestamp'] > datetime.now():
        raise ValueError("Timestamp in the future")

    return True
```

資料清洗:

```python
def clean_sensor_data(data):
    # 移除異常值
    if data['value'] < 0:
        data['value'] = 0
    if data['value'] > 1000:
        data['value'] = 1000

    # 填補缺失值
    if 'unit' not in data:
        data['unit'] = get_default_unit(data['equipment_id'],
data['parameter'])

    return data
```

18.3 Omniverse 最佳實踐

18.3.1 USD 檔案組織

```
factory.usd (根檔案)
├── buildings/
│   ├── building_a.usd
│   └── building_b.usd
├── equipment/
│   ├── die_bonders/
│   │   ├── db_001.usd
│   │   └── db_002.usd
│   └── wire_bonders/
│       ├── wb_001.usd
│       └── wb_002.usd
└── materials/
    ├── metal.usd
    └── plastic.usd
```

18.3.2 效能優化

LOD (Level of Detail): - 遠距離: 低精度模型 - 中距離: 中精度模型 - 近距離: 高精度模型

Instancing: - 相同設備使用 Instance,減少記憶體使用

Culling: - 不在視野內的物件不渲染
