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

第二部分:技術架構

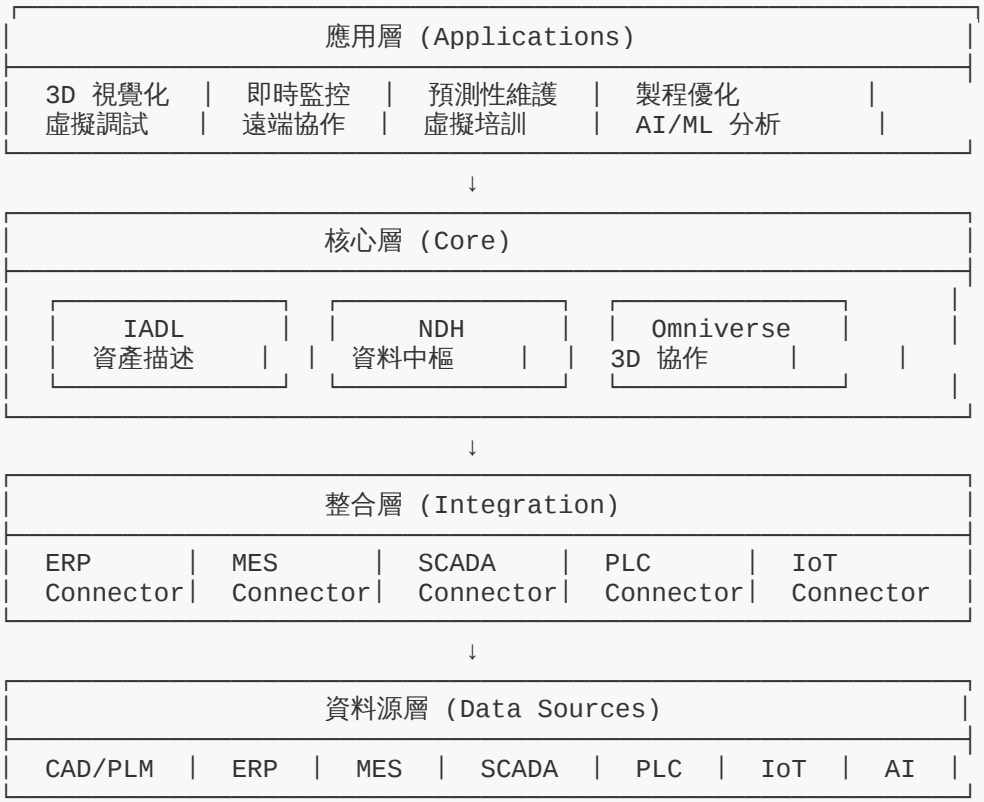
4. IDTF 整體架構

4.1 ISA-95 五層架構

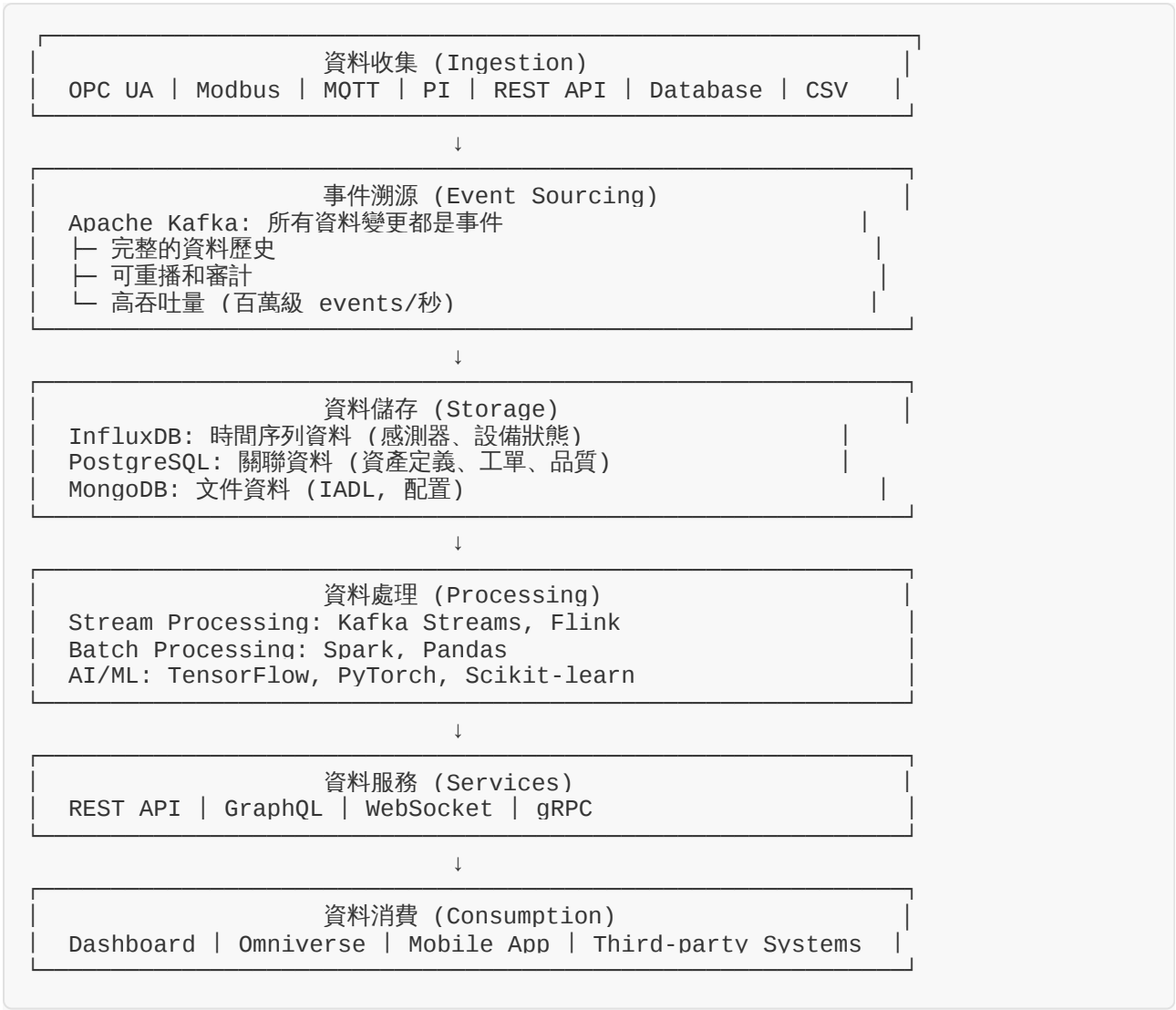
IDTF 基於 ISA-95 標準,整合從現場設備 (Level 0) 到企業管理 (Level 4) 的所有層級:


Level 4: 企業層 (Enterprise)
ERP - 企業資源規劃
├─ 財務管理、供應鏈管理、客戶關係管理
IDTF Hub (NDH)
Level 3: 製造執行層 (Manufacturing Execution)
MES - 製造執行系統
├─ 生產排程、工單管理、品質管理、物料追蹤
IDTF Hub (NDH)
Level 2: 監控層 (Supervisory Control)
SCADA - 監控與資料擷取系統
├─ HMI、即時監控、報警管理、歷史資料記錄
IDTF Hub (NDH)
Level 1: 控制層 (Control)
PLC/DCS - 可程式邏輯控制器 / 分散式控制系統
├─ 製程控制、邏輯控制、順序控制
IDTF Hub (NDH)
Level 0: 現場層 (Field Devices)
感測器、執行器、儀表、設備

4.2 IDTF 核心架構

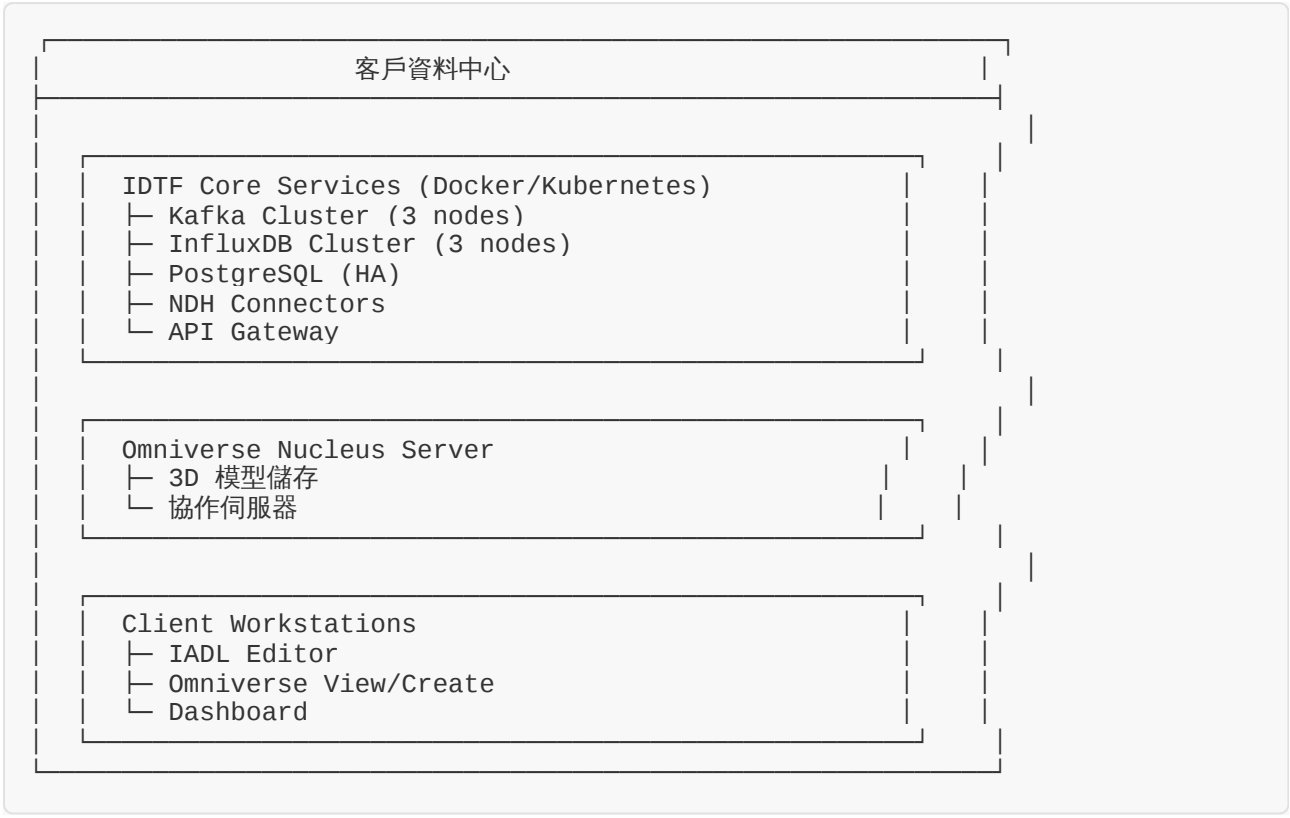

應用層 (Applications)
3D 視覺化
即時監控
預測性維護
製程優化
虛擬調試
遠端協作
虛擬培訓
AI/ML 分析
核心層 (Core)
IADL
資產描述
NDH
資料中樞
Omniverse
3D 協作
整合層 (Integration)
ERP Connector
MES Connector
SCADA Connector
PLC Connector
IoT Connector
資料源層 (Data Sources)
CAD/PLM
ERP
MES
SCADA
PLC
IoT
AI

4.3 資料流架構

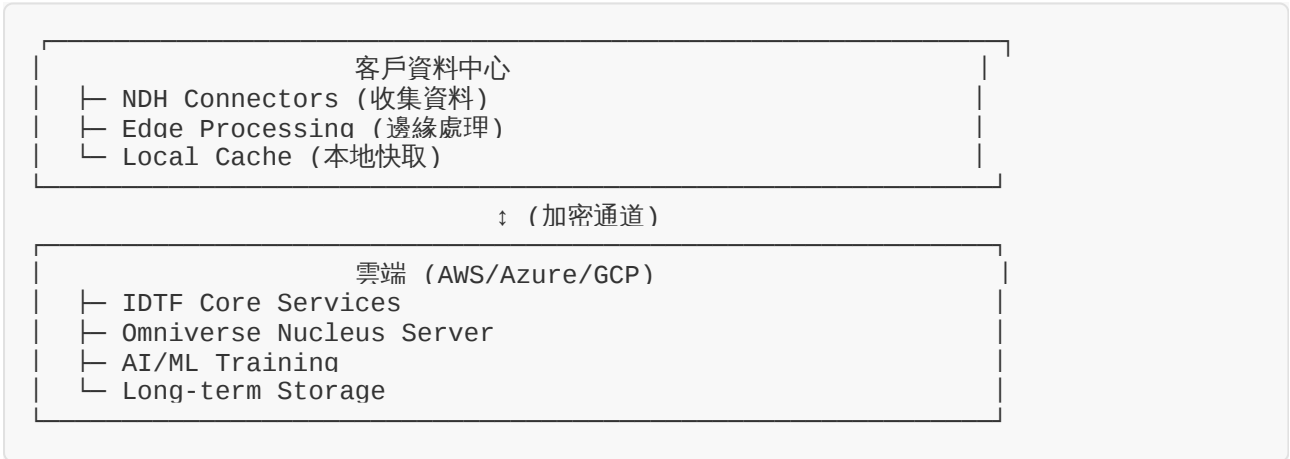

資料收集 (Ingestion)
OPC UA | Modbus | MQTT | PI | REST API | Database | CSV
事件溯源 (Event Sourcing)
Apache Kafka: 所有資料變更都是事件
├─ 完整的資料歷史
├─ 可重播和審計
└─ 高吞吐量 (百萬級 events/秒)
資料儲存 (Storage)
InfluxDB: 時間序列資料 (感測器、設備狀態)
PostgreSQL: 關聯資料 (資產定義、工單、品質)
MongoDB: 文件資料 (IADL, 配置)
資料處理 (Processing)
Stream Processing: Kafka Streams, Flink
Batch Processing: Spark, Pandas
AI/ML: TensorFlow, PyTorch, Scikit-learn
資料服務 (Services)
REST API | GraphQL | WebSocket | gRPC
資料消費 (Consumption)
Dashboard | Omniverse | Mobile App | Third-party Systems

4.4 部署架構

4.4.1 本地部署 (On-Premise)



4.4.2 混合部署 (Hybrid)



---

5. IADL: 工業資產描述語言

5.1 IADL 簡介

Industrial Asset Description Language (IADL) 是一種標準化的工業資產描述語言,用 YAML 格式定義工業資產的所有屬性、參數、標籤和最佳實踐。

5.2 IADL 資料模型

```yaml
# IADL 資料模型
asset:
  # 基本資訊
  id: string                    # 唯一識別碼
  name: string                  # 資產名稱
  type: string                  # 資產類型
  category: string              # 資產類別

  # 製造商資訊
  manufacturer: string          # 製造商
  model: string                 # 型號
  serial_number: string         # 序號
  manufacture_date: date        # 製造日期

  # 位置資訊
  location:
    plant: string               # 工廠
    area: string                # 區域
    line: string                # 產線
    position:                   # 3D 位置
      x: float
      y: float
      z: float

  # 技術參數
  parameters:
    [parameter_name]: value     # 參數名稱和值

  # 標籤結構
  tags:
    plc:                        # PLC 標籤
      [tag_name]:
        address: string         # 標籤地址
        data_type: string       # 資料類型
        unit: string            # 單位
    scada:                      # SCADA 標籤
      [tag_name]:
        path: string            # 標籤路徑
        data_type: string
        unit: string
    iot:                        # IoT 標籤
      [tag_name]:
        topic: string           # MQTT Topic
        data_type: string
        unit: string

  # 維護資訊
  maintenance:
    schedule:                   # 維護排程
      - type: string            # 維護類型
        frequency: string       # 頻率
        last_date: date         # 上次維護日期
        next_date: date         # 下次維護日期
    history:                    # 維護歷史
      - date: date
        type: string
        description: string
        cost: float

  # 最佳實踐
  best_practices:
    - description: string       # 最佳實踐描述
```

```yaml
  # 故障案例
  failure_cases:
    - date: date
      symptom: string           # 症狀
      root_cause: string        # 根本原因
      solution: string          # 解決方案
      downtime: float           # 停機時間 (小時)
      cost: float               # 成本
```

5.3 IADL 範例

5.3.1 固晶機 (Die Bonder)

```yaml
asset:
  id: "DB-001"
  name: "固晶機 #1"
  type: "Die Bonder"
  category: "Production Equipment"

  manufacturer: "ASM"
  model: "AD830"
  serial_number: "AD830-2024-001"
  manufacture_date: "2024-01-15"

  location:
    plant: "Hsinchu Factory"
    area: "Production Area A"
    line: "Line 1"
    position:
      x: 10.5
      y: 20.3
      z: 0.0

  parameters:
    bonding_temperature: 180      # °C
    bonding_pressure: 50          # N
    bonding_time: 2.5             # seconds
    pick_height: 0.5              # mm
    bond_height: 0.1              # mm

  tags:
    plc:
      temperature:
        address: "DB1.DBD0"
        data_type: "REAL"
        unit: "°C"
      pressure:
        address: "DB1.DBD4"
        data_type: "REAL"
        unit: "N"
      status:
        address: "DB1.DBW8"
        data_type: "INT"
        unit: ""
    scada:
      temperature:
        path: "Line1.DB001.Temperature"
        data_type: "float"
        unit: "°C"
      oee:
        path: "Line1.DB001.OEE"
        data_type: "float"
        unit: "%"
    iot:
      vibration:
        topic: "factory/line1/db001/vibration"
        data_type: "float"
        unit: "mm/s"

  maintenance:
    schedule:
      - type: "Daily Cleaning"
```

```yaml
      frequency: "Daily"
      last_date: "2025-10-10"
      next_date: "2025-10-11"
    - type: "Preventive Maintenance"
      frequency: "Monthly"
      last_date: "2025-09-15"
      next_date: "2025-10-15"
  history:
    - date: "2025-09-15"
      type: "Preventive Maintenance"
      description: "更換加熱元件,校準溫度感測器"
      cost: 5000

best_practices:
  - "每班開機前預熱 30 分鐘"
  - "溫度波動超過 ±2°C 需要校準"
  - "每週檢查真空吸嘴"
  - "每月更換過濾網"

failure_cases:
  - date: "2025-08-20"
    symptom: "溫度無法達到設定值"
    root_cause: "加熱元件老化"
    solution: "更換加熱元件"
    downtime: 4.0
    cost: 8000
```

5.4 Asset Template Library

Asset Template Library 是預定義的資產範本庫,涵蓋常見的工業資產類型:

5.4.1 半導體/LED 製造

- **固晶機** (Die Bonder): DB-001, DB-002, ...
- **打線機** (Wire Bonder): WB-001, WB-002, ...
- **封膠機** (Encapsulation Machine): EM-001, EM-002, ...
- **烘烤爐** (Curing Oven): CO-001, CO-002, ...
- **分類機** (Sorter): SR-001, SR-002, ...

5.4.2 流程工業

- **泵** (Pump): 離心泵、容積泵、真空泵
- **閥門** (Valve): 控制閥、開關閥、安全閥
- **容器** (Vessel): 壓力容器、儲槽、反應器
- **換熱器** (Heat Exchanger): 管殼式、板式、空冷器
- **塔器** (Column): 蒸餾塔、吸收塔、萃取塔

5.4.3 分析儀表

- **流量計** (Flow Meter): 渦輪、科氏力、超音波
- **壓力變送器** (Pressure Transmitter)
- **溫度變送器** (Temperature Transmitter)
- **液位計** (Level Transmitter): 雷達、超音波、浮球
- **分析儀** (Analyzer): pH、導電度、濁度

5.5 IADL Editor

IADL Editor 是一個圖形化的資產編輯器,提供 Drag & Drop 介面:

5.5.1 主要功能

1. **Asset Designer**: 拖拉式資產設計
2. **Template Browser**: 瀏覽和選擇範本
3. **Tag Generator**: 自動生成 PLC/SCADA/IoT 標籤
4. **Validation**: 驗證 IADL 定義的正確性
5. **Export**: 匯出到 Wonderware, Ignition, WinCC 等

5.5.2 工作流程

```
1. 從 Template Library 選擇範本
   ↓
2. Drag & Drop 到工廠佈局
   ↓
3. 修改參數和標籤
   ↓
4. 驗證 IADL 定義
   ↓
5. 匯出到 SCADA/PLC
   ↓
6. 同步到 NDH 和 Omniverse
```

---

6. NDH: 中立資料中樞

6.1 NDH 簡介

Neutral Data Hub (NDH) 是一個中立的資料整合中樞,基於事件溯源架構,整合所有工業系統的資料。

6.2 NDH 核心組件

6.2.1 Apache Kafka (事件溯源)

功能: - 所有資料變更都是事件 - 完整的資料歷史 - 可重播和審計 - 高吞吐量 (百萬級 events/秒)

Topic 設計:

```
 # ERP Topics
erp.production_plan
erp.material_requirement
erp.purchase_order

# MES Topics
mes.work_order
mes.quality
mes.material_tracking

# SCADA Topics
scada.equipment_status
scada.alarm
scada.trend

# PLC Topics
plc.tag_change
plc.event

# IoT Topics
iot.sensor_data
iot.device_status
```

6.2.2 InfluxDB (時間序列資料)

功能: - 高效儲存時間序列資料 - 快速查詢和聚合 - 自動資料保留策略

資料模型:

```
# Measurement: equipment_status
time                 equipment_id  parameter      value
2025-10-11 10:00:00 DB-001        temperature    180.5
2025-10-11 10:00:01 DB-001        temperature    180.6
2025-10-11 10:00:02 DB-001        temperature    180.4
```

6.2.3 PostgreSQL (關聯資料)

功能: - 儲存資產定義、工單、品質資料 - 支援複雜查詢和關聯

資料模型:

```sql
-- Assets Table
CREATE TABLE assets (
    id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(200),
    type VARCHAR(100),
    manufacturer VARCHAR(100),
    model VARCHAR(100),
    location_plant VARCHAR(100),
    location_area VARCHAR(100),
    iadl_definition JSONB
);

-- Work Orders Table
CREATE TABLE work_orders (
    wo_id VARCHAR(50) PRIMARY KEY,
    product VARCHAR(200),
    quantity INT,
    start_time TIMESTAMP,
    due_time TIMESTAMP,
    status VARCHAR(50),
    progress FLOAT
);
```

6.2.4 MongoDB (文件資料)

功能: - 儲存 IADL 定義、配置、文件 - 靈活的 schema

6.3 NDH Connectors

6.3.1 OPC UA Connector

```python
class OPCUAConnector:
    def __init__(self, server_url, namespace):
        self.client = Client(server_url)
        self.namespace = namespace

    def connect(self):
        self.client.connect()

    def subscribe(self, node_ids):
        # 訂閱 OPC UA 節點
        for node_id in node_ids:
            node = self.client.get_node(node_id)
            node.subscribe_data_change(self.on_data_change)

    def on_data_change(self, node, value, data):
        # 發布到 Kafka
        event = {
            "source": "OPC UA",
            "node_id": node.nodeid.to_string(),
            "value": value,
            "timestamp": data.monitored_item.Value.SourceTimestamp
        }
        kafka_producer.send("plc.tag_change", event)
```

6.3.2 Modbus Connector

```python
class ModbusConnector:
    def __init__(self, host, port):
        self.client = ModbusTcpClient(host, port)

    def read_holding_registers(self, address, count):
        result = self.client.read_holding_registers(address, count)
        return result.registers

    def poll(self, register_map):
        # 輪詢 Modbus 暫存器
        for register in register_map:
            value = self.read_holding_registers(
                register["address"],
                register["count"]
            )

            # 發布到 Kafka
            event = {
                "source": "Modbus",
                "equipment_id": register["equipment_id"],
                "parameter": register["parameter"],
                "value": value,
                "timestamp": datetime.now()
            }
            kafka_producer.send("plc.tag_change", event)
```

6.3.3 MQTT Connector

```
class MQTTConnector:
    def __init__(self, broker, port):
        self.client = mqtt.Client()
        self.client.on_message = self.on_message
        self.client.connect(broker, port)

    def subscribe(self, topics):
        for topic in topics:
            self.client.subscribe(topic)

    def on_message(self, client, userdata, message):
        # 發布到 Kafka
        event = {
            "source": "MQTT",
            "topic": message.topic,
            "payload": message.payload.decode(),
            "timestamp": datetime.now()
        }
        kafka_producer.send("iot.sensor_data", event)
```

6.4 NDH API

6.4.1 REST API

```
# FastAPI REST API
@app.get("/api/assets/{asset_id}")
async def get_asset(asset_id: str):
    # 從 PostgreSQL 讀取資產定義
    asset = db.query(Asset).filter(Asset.id == asset_id).first()
    return asset

@app.get("/api/assets/{asset_id}/realtime")
async def get_asset_realtime(asset_id: str):
    # 從 InfluxDB 讀取即時資料
    query = f'SELECT * FROM equipment_status WHERE equipment_id =
\'{asset_id}\' ORDER BY time DESC LIMIT 1'
    result = influxdb_client.query(query)
    return result

@app.get("/api/assets/{asset_id}/history")
async def get_asset_history(asset_id: str, start: datetime, end: datetime):
    # 從 InfluxDB 讀取歷史資料
    query = f'SELECT * FROM equipment_status WHERE equipment_id =
\'{asset_id}\' AND time >= \'{start}\' AND time <= \'{end}\''
    result = influxdb_client.query(query)
    return result
```

6.4.2 GraphQL API

```
type Asset {
  id: ID!
  name: String!
  type: String!
  manufacturer: String
  model: String
  location: Location
  parameters: JSON
  realtime: RealtimeData
  history(start: DateTime!, end: DateTime!): [HistoricalData]
}

type Query {
  asset(id: ID!): Asset
  assets(type: String, location: String): [Asset]
}

type Subscription {
  assetRealtime(id: ID!): RealtimeData
}
```

6.4.3 WebSocket API

```python
# WebSocket for real-time updates
@app.websocket("/ws/assets/{asset_id}")
async def websocket_endpoint(websocket: WebSocket, asset_id: str):
    await websocket.accept()

    # 訂閱 Kafka topic
    consumer = KafkaConsumer(f"asset.{asset_id}.realtime")

    for message in consumer:
        # 推送即時資料到 WebSocket
        await websocket.send_json(message.value)
```

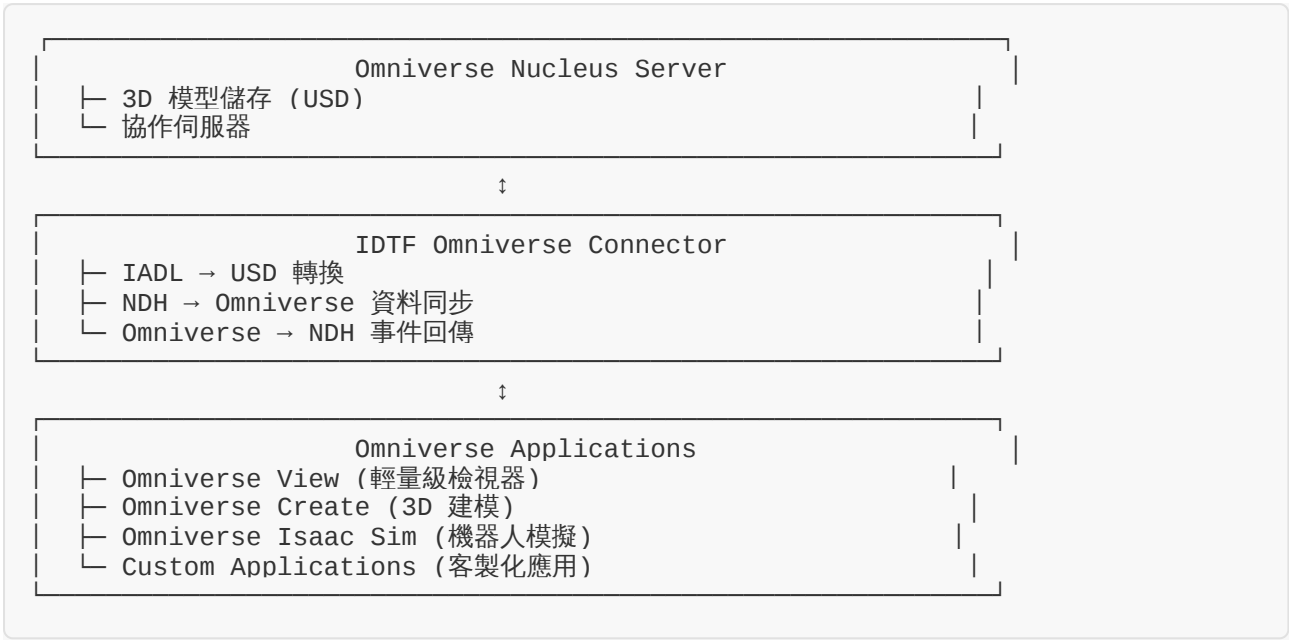
---

7. Omniverse: 3D 協作平台

7.1 Omniverse 簡介

NVIDIA Omniverse 是一個基於 USD (Universal Scene Description) 的 3D 協作平台,提供:
- 即時多人協作 - 物理模擬 (PhysX) - 即時光線追蹤 (RTX) - 虛擬調試和培訓

7.2 Omniverse 在 IDTF 中的角色



7.3 IADL 到 Omniverse 的轉換

7.3.1 資產 3D 模型

```
# IADL to USD Converter
class IADLToUSDConverter:
    def convert_asset_to_usd(self, iadl_asset):
        # 建立 USD Stage
        stage = Usd.Stage.CreateNew(f"{iadl_asset['id']}.usd")

        # 建立 Xform (Transform)
        xform = UsdGeom.Xform.Define(stage, f"/{iadl_asset['id']}")

        # 設置位置
        xform.AddTranslateOp().Set(Gf.Vec3d(
            iadl_asset['location']['position']['x'],
            iadl_asset['location']['position']['y'],
            iadl_asset['location']['position']['z']
        ))

        # 載入 3D 模型 (從 Asset Library)
        model_path = self.get_model_path(iadl_asset['type'],
iadl_asset['model'])
        xform.GetPrim().GetReferences().AddReference(model_path)

        # 添加自訂屬性 (IADL 參數)
        for param_name, param_value in iadl_asset['parameters'].items():
            attr = xform.GetPrim().CreateAttribute(
                f"iadl:{param_name}",
                Sdf.ValueTypeNames.Float
            )
            attr.Set(param_value)

        # 儲存 USD
        stage.Save()

        return stage
```

7.3.2 即時資料同步

```
# NDH to Omniverse Sync
class NDHOmniverseSync:
    def __init__(self, nucleus_url):
        self.nucleus_url = nucleus_url

    def sync_realtime_data(self, asset_id, realtime_data):
        # 開啟 USD Stage
        stage = Usd.Stage.Open(f"{self.nucleus_url}/{asset_id}.usd")

        # 更新屬性
        xform = UsdGeom.Xform.Get(stage, f"/{asset_id}")

        for param_name, param_value in realtime_data.items():
            attr = xform.GetPrim().GetAttribute(f"iadl:{param_name}")
            if attr:
                attr.Set(param_value)

        # 儲存 (自動同步到所有 Omniverse 客戶端)
        stage.Save()
```

7.4 Omniverse 應用場景

7.4.1 3D 視覺化監控

```python
# Omniverse Dashboard
class OmniverseDashboard:
    def __init__(self, factory_usd):
        self.stage = Usd.Stage.Open(factory_usd)

    def update_equipment_status(self, equipment_id, status):
        # 根據狀態改變顏色
        xform = UsdGeom.Xform.Get(self.stage, f"/{equipment_id}")
        mesh = UsdGeom.Mesh.Get(self.stage, f"/{equipment_id}/Mesh")

        if status == "Running":
            color = Gf.Vec3f(0, 1, 0)  # 綠色
        elif status == "Idle":
            color = Gf.Vec3f(1, 1, 0)  # 黃色
        elif status == "Alarm":
            color = Gf.Vec3f(1, 0, 0)  # 紅色
        else:
            color = Gf.Vec3f(0.5, 0.5, 0.5)  # 灰色

        mesh.GetDisplayColorAttr().Set([color])
        self.stage.Save()
```

7.4.2 虛擬調試

```python
# Virtual Commissioning
class VirtualCommissioning:
    def __init__(self, factory_usd):
        self.stage = Usd.Stage.Open(factory_usd)

    def simulate_production(self, work_order):
        # 模擬生產流程
        for step in work_order['steps']:
            equipment_id = step['equipment_id']
            duration = step['duration']

            # 在 Omniverse 中模擬設備運行
            self.run_equipment(equipment_id, duration)

            # 檢查碰撞和干涉
            if self.check_collision(equipment_id):
                return {"status": "Error", "message": "Collision detected"}

        return {"status": "Success"}

    def run_equipment(self, equipment_id, duration):
        # 使用 PhysX 模擬設備運動
        pass

    def check_collision(self, equipment_id):
        # 使用 PhysX 檢查碰撞
        pass
```

7.4.3 虛擬培訓

```python
# Virtual Training
class VirtualTraining:
    def __init__(self, factory_usd):
        self.stage = Usd.Stage.Open(factory_usd)

    def create_training_scenario(self, scenario_type):
        if scenario_type == "normal_operation":
            # 正常運行情境
            self.simulate_normal_operation()
        elif scenario_type == "emergency_stop":
            # 緊急停機情境
            self.simulate_emergency_stop()
        elif scenario_type == "equipment_failure":
            # 設備故障情境
            self.simulate_equipment_failure()

    def simulate_normal_operation(self):
        # 模擬正常運行
        pass

    def simulate_emergency_stop(self):
        # 模擬緊急停機
        # 學員需要正確操作
        pass

    def simulate_equipment_failure(self):
        # 模擬設備故障
        # 學員需要診斷和解決
        pass
```

由於白皮書內容非常龐大,我將分段繼續撰寫。讓我先將目前的內容儲存,然後繼續完成剩餘部分。