

# NDH 支援 AVEVA PI System 與 Historian 整合架構

作者: Manus AI

日期: 2025年10月14日

版本: 1.0

文件類型: 技術架構設計

## 執行摘要

本文檔設計 NDH (Neutral Data Hub) 與兩大主流工業時序數據平台的整合架構: - **AVEVA PI System** (前身為 OSIsoft PI) - **Historian** (包括 GE Historian, Honeywell PHD, Siemens WinCC 等)

## 核心設計目標

- 雙向數據流:** 支援從 PI/Historian 讀取歷史數據,也支援將 NDH 數據寫入 PI/Historian
- 即時數據訂閱:** 支援 PI System 的即時數據流 (Streaming)
- 統一抽象:** 透過 NDH 的時序數據庫抽象層,將 PI/Historian 視為另一種 TSDB
- 零侵入:** 不修改現有 PI/Historian 系統,僅透過標準 API 整合
- 企業級功能:** 支援 PI AF (Asset Framework) 的資產模型映射

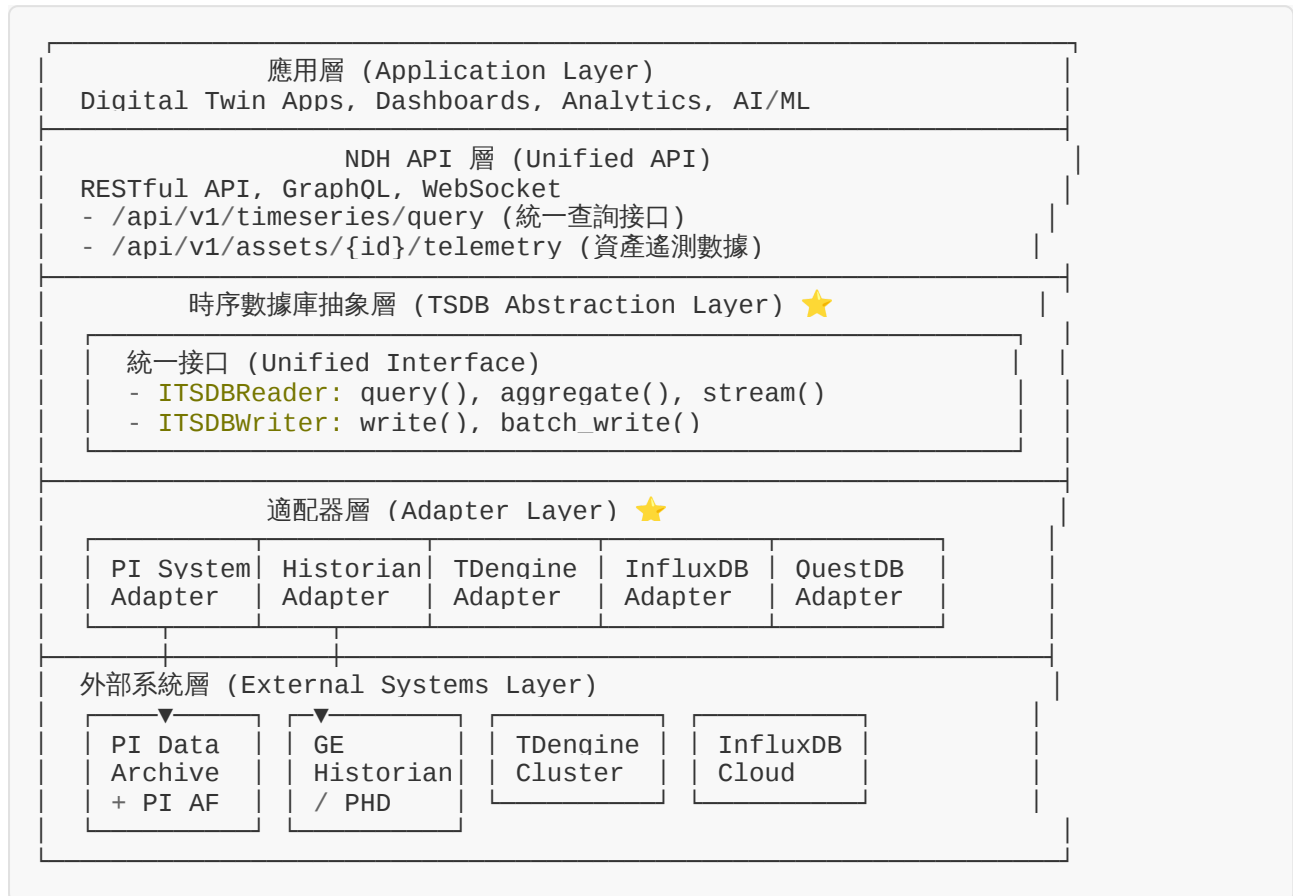
## 整合價值

**對企業的價值:** - ☒ 保護現有 PI/Historian 投資,無需遷移數據 - ☒ 將 PI/Historian 數據與 OT/IT 系統統一整合 - ☒ 透過 IADL 為 PI 數據提供語義化和情境化 - ☒ 支援混合架構 (PI 用於歷史數據,TDengine 用於新數據)

**技術效益:** - 數據訪問統一化 (透過 NDH API 訪問所有數據源) - 降低應用開發複雜度 (無需直接對接 PI SDK) - 支援跨平台數據分析和 AI/ML 應用

# 1. 整合架構總覽

## 1.1 分層架構



## 1.2 核心組件

### 1.2.1 PI System Adapter

- **PI Data Archive Connector:** 連接 PI 時序數據庫
- **PI AF Connector:** 連接 PI Asset Framework (資產模型)
- **PI Web API Client:** 透過 REST API 訪問 PI 數據
- **IADL Mapper:** 將 PI AF 資產模型映射為 IADL

### 1.2.2 Historian Adapter

- **OPC HDA Client:** 透過 OPC Historical Data Access 協議
- **Native API Client:** 支援各廠商原生 API (GE, Honeywell, Siemens)
- **ODBC Connector:** 透過 ODBC 訪問 Historian 數據庫

---

## 2. PI System 整合設計

---





### 2.1 PI System 架構概述

**PI System 核心組件:** 1. **PI Data Archive:** 時序數據存儲引擎 2. **PI Asset Framework (AF):** 資產模型和元數據管理 3. **PI Web API:** RESTful API 接口 4. **PI SDK:** 原生 C#/.NET SDK

### 2.2 整合策略

NDH 支援三種整合方式,按優先順序:

#### 方式 1: PI Web API (推薦)

**優勢:** -  跨平台 (HTTP/REST,無需 Windows) -  現代化架構,易於維護 -  支援即時數據流 (WebSocket) -  內建安全認證 (Kerberos, Basic Auth)

**劣勢:** -  需要 PI Web API 服務器 (PI System 2016+) -  性能略低於原生 SDK

#### 方式 2: PI SDK (高性能場景)

**優勢:** -  最高性能 -  支援所有 PI 功能

**劣勢:** -  僅支援 Windows 平台 -  需要安裝 PI SDK -  維護成本高

#### 方式 3: ODBC (備用方案)

**優勢:** -  簡單易用

**劣勢:** -  性能最低 -  功能受限 -  不支援即時數據

## 2.3 PI System Adapter 實現

### 2.3.1 核心接口實現

```
# pi_adapter.py

from typing import List, Dict, Any, Optional
from datetime import datetime
import httpx
from ndh.tsdb.interface import ITSDBReader, ITSDBWriter

class PISystemAdapter(ITSDBReader, ITSDBWriter):
    """AVEVA PI System 適配器 (基於 PI Web API)"""

    def __init__(self, config: Dict[str, Any]):
        """
        初始化 PI System 適配器

        Args:
            config: 配置字典
                - web_api_url: PI Web API 端點 (e.g., "https://pi-server/piwebapi")
                - username: 用戶名
                - password: 密碼
                - verify_ssl: 是否驗證 SSL 證書
                - af_database: PI AF 數據庫名稱 (可選)
        """
        self.web_api_url = config['web_api_url'].rstrip('/')
        self.auth = (config['username'], config['password'])
        self.verify_ssl = config.get('verify_ssl', True)
        self.af_database = config.get('af_database')

        self.client = httpx.AsyncClient(
            auth=self.auth,
            verify=self.verify_ssl,
            timeout=30.0
        )

    async def query(self,
                    measurement: str,
                    tags: Optional[Dict[str, str]] = None,
                    fields: Optional[List[str]] = None,
                    start_time: Optional[datetime] = None,
                    end_time: Optional[datetime] = None,
                    limit: Optional[int] = None) -> List[Dict[str, Any]]:
        """
        查詢 PI 时序數據

        Args:
            measurement: PI Tag 名稱 (在 PI 中即為 Point Name)
            tags: 標籤過濾 (在 PI 中可映射為 AF Attributes)
            fields: 字段列表 (在 PI 中為 Value, Timestamp, Quality)
            start_time: 開始時間
            end_time: 結束時間
            limit: 返回記錄數限制

        Returns:
            时序數據列表
        """
        # 1. 查找 PI Point WebID
        point_webid = await self._get_point_webid(measurement)

        # 2. 構建查詢 URL
```

```

url = f"{self.web_api_url}/streams/{point_webid}/recorded"

# 3. 構建查詢參數
params = {}
if start_time:
    params['startTime'] = start_time.isoformat()
if end_time:
    params['endTime'] = end_time.isoformat()
if limit:
    params['maxCount'] = limit

# 4. 發送請求
response = await self.client.get(url, params=params)
response.raise_for_status()

# 5. 解析響應
data = response.json()
items = data.get('Items', [])

# 6. 轉換為 NDH 標準格式
results = []
for item in items:
    results.append({
        'measurement': measurement,
        'timestamp':
datetime.fromisoformat(item['Timestamp']).replace('Z', '+00:00'),
        'value': item['Value'],
        'quality': item.get('Good', True),
        'tags': tags or {},
        'fields': {'value': item['Value']}
    })

return results

async def aggregate(self,
                    measurement: str,
                    aggregation: str,
                    interval: str,
                    start_time: datetime,
                    end_time: datetime,
                    tags: Optional[Dict[str, str]] = None) -> List[Dict[str,
Anvll:
    """
    聚合查詢 (使用 PI 的 Summary 功能)

    Args:
        measurement: PI Tag 名稱
        aggregation: 聚合函數 (average, minimum, maximum, total, count)
        interval: 聚合間隔 (e.g., "1h", "15m")
        start_time: 開始時間
        end_time: 結束時間
        tags: 標籤過濾

    Returns:
        聚合結果列表
    """
    point_webid = await self._get_point_webid(measurement)

    # PI Web API 的 Summary 端點
    url = f"{self.web_api_url}/streams/{point_webid}/summary"

    params = {
        'startTime': start_time.isoformat(),
        'endTime': end_time.isoformat(),
        'summaryType': self._map_aggregation(aggregation),
        'summaryDuration': self._convert_interval(interval)
    }

```

```

    }

    response = await self.client.get(url, params=params)
    response.raise_for_status()

    data = response.json()
    items = data.get('Items', [])

    results = []
    for item in items:
        results.append({
            'measurement': measurement,
            'timestamp':
datetime.fromisoformat(item['Timestamp'].replace('Z', '+00:00')),
            'aggregation': aggregation,
            'value': item['Value']['Value'],
            'tags': tags or {}
        })

    return results

async def write(self,
                measurement: str,
                tags: Dict[str, str],
                fields: Dict[str, Any],
                timestamp: Optional[datetime] = None) -> bool:
    """
    寫入數據到 PI (需要寫入權限)

    Args:
        measurement: PI Tag 名稱
        tags: 標籤
        fields: 字段 (必須包含 'value')
        timestamp: 時間戳 (可選, 默認為當前時間)

    Returns:
        是否成功
    """
    point_webid = await self._get_point_webid(measurement)

    url = f"{self.web_api_url}/streams/{point_webid}/value"

    payload = {
        'Timestamp': (timestamp or datetime.now()).isoformat(),
        'Value': fields.get('value'),
        'Good': True
    }

    response = await self.client.post(url, json=payload)
    return response.status_code == 202 # PI 返回 202 Accepted

async def batch_write(self, data_points: List[Dict[str, Any]]) -> bool:
    """
    批量寫入 (使用 PI Web API Batch 功能)
    """
    # PI Web API 支援批量操作
    # 實現細節省略...
    pass

# ===== PI AF 資產模型整合 =====

async def get_af_elements(self, element_path: Optional[str] = None) ->
List[Dict[str, Any]]:
    """
    獲取 PI AF 資產元素

```

```

    Args:
        element_path: AF 元素路徑 (e.g., "\\AF-
Server\\Database\\Plant1\\Area1")

    Returns:
        AF 元素列表
    """
    if not self.af_database:
        raise ValueError("AF Database not configured")

    # 獲取 AF Database WebID
    db_url = f"{self.web_api_url}/assetdatabases"
    params = {'path': f"\\\\{self.af_database}"}
    response = await self.client.get(db_url, params=params)
    db_webid = response.json()['WebID']

    # 獲取元素
    elements_url = f"{self.web_api_url}/assetdatabases/{db_webid}/elements"
    response = await self.client.get(elements_url)

    return response.json().get('Items', [])

async def map_af_to_iadl(self, af_element_webid: str) -> Dict[str, Any]:
    """
    將 PI AF 元素映射為 IADL 格式

    Args:
        af_element_webid: AF 元素的 WebID

    Returns:
        IADL 格式的資產定義
    """
    # 獲取 AF 元素詳情
    url = f"{self.web_api_url}/elements/{af_element_webid}"
    response = await self.client.get(url)
    element = response.json()

    # 獲取 Attributes
    attrs_url = f"{url}/attributes"
    attrs_response = await self.client.get(attrs_url)
    attributes = attrs_response.json().get('Items', [])

    # 映射為 IADL
    iadl = {
        'asset_id': element['Name'],
        'asset_type': element.get('TemplateName', 'Generic'),
        'description': element.get('Description', ''),
        'data_tags': [],
        'properties': {}
    }

    # 映射 Attributes 為 Data Tags
    for attr in attributes:
        if attr.get('Type') == 'PI Point':
            iadl['data_tags'].append({
                'tag_id': attr['Name'],
                'tag_name': attr['Name'],
                'data_type': self.map_pi_type(attr.get('Type')),
                'unit': attr.get('DefaultUnitsName', ''),
                'source': {
                    'type': 'pi_point',
                    'point_name': attr.get('PIPoint', {}).get('Name', '')
                }
            })
        else:
            iadl['properties'][attr['Name']] = attr.get('Value')

```

```

        return iadl

# ===== 輔助方法 =====

async def _get_point_webid(self, point_name: str) -> str:
    """獲取 PI Point 的 WebID"""
    url = f"{self.web_api_url}/points"
    params = {'path': f"\\\\\\{point_name}"}

    response = await self.client.get(url, params=params)
    response.raise_for_status()

    return response.json()['WebId']

def _map_aggregation(self, agg: str) -> str:
    """映射聚合函數名稱"""
    mapping = {
        'mean': 'Average',
        'average': 'Average',
        'min': 'Minimum',
        'minimum': 'Minimum',
        'max': 'Maximum',
        'maximum': 'Maximum',
        'sum': 'Total',
        'total': 'Total',
        'count': 'Count'
    }
    return mapping.get(agg.lower(), 'Average')

def _convert_interval(self, interval: str) -> str:
    """轉換時間間隔格式 (e.g., "1h" -> "1h")"""
    # PI Web API 使用類似格式, 可能需要微調
    return interval

def _map_pi_type(self, pi_type: str) -> str:
    """映射 PI 數據類型為 IADL 類型"""
    mapping = {
        'Float32': 'float',
        'Float64': 'double',
        'Int16': 'int',
        'Int32': 'int',
        'String': 'string',
        'Boolean': 'bool',
        'Timestamp': 'datetime'
    }
    return mapping.get(pi_type, 'float')

```



### 2.3.2 配置範例

```
# ndh_config.yaml

tsdb:
  # 主時序數據庫 (新數據)
  primary:
    type: tdengine
    host: localhost
    port: 6030
    database: ndh_data

  # PI System (歷史數據)
  secondary:
    - type: pi_system
      name: plant1_pi
      web_api_url: https://pi-server.company.com/piwebapi
      username: ndh_service
      password: ${PI_PASSWORD} # 從環境變量讀取
      verify_ssl: true
      af_database: PlantAssets

  # 數據路由規則
  routing:
    # 歷史數據從 PI 讀取
    read_historical: true
    # 新數據寫入 TDengine
    write_new: false
    # 數據範圍
    time_range:
      start: "2020-01-01"
      end: "2024-12-31"
```

## 3. Historian 整合設計

### 3.1 支援的 Historian 系統

Historian 系統	整合方式	優先級
GE Historian	OPC HDA + Native API	高
Honeywell PHD	OPC HDA + ODBC	高
Siemens WinCC	OPC HDA	中
Wonderware Historian	OPC HDA	中
Rockwell FactoryTalk	OPC HDA	中

## 3.2 Historian Adapter 實現

```
# historian_adapter.py

from typing import List, Dict, Any, Optional
from datetime import datetime
from opcua import Client as OPCClient
from ndh.tsdb.interface import ITSDBReader

class HistorianAdapter(ITSDBReader):
    """通用 Historian 適配器 (基於 OPC HDA)"""

    def __init__(self, config: Dict[str, Any]):
        """
        初始化 Historian 適配器

        Args:
            config: 配置字典
                - opc_hda_url: OPC HDA 服務器 URL
                - historian_type: Historian 類型 (ge, honeywell, siemens)
                - username: 用戶名 (可選)
                - password: 密碼 (可選)
        """
        self.opc_url = config['opc_hda_url']
        self.historian_type = config.get('historian_type', 'generic')
        self.username = config.get('username')
        self.password = config.get('password')

        # 初始化 OPC HDA 客戶端
        self.client = OPCClient(self.opc_url)
        if self.username:
            self.client.set_user(self.username)
            self.client.set_password(self.password)

    async def query(self,
                    measurement: str,
                    tags: Optional[Dict[str, str]] = None,
                    fields: Optional[List[str]] = None,
                    start_time: Optional[datetime] = None,
                    end_time: Optional[datetime] = None,
                    limit: Optional[int] = None) -> List[Dict[str, Any]]:
        """
        查詢 Historian 時序數據 (透過 OPC HDA)

        Args:
            measurement: Tag 名稱
            start time: 開始時間
            end_time: 結束時間
            limit: 記錄數限制

        Returns:
            時序數據列表
        """
        # 連接 OPC HDA 服務器
        self.client.connect()

        try:
            # 讀取歷史數據
            # OPC HDA 的具體實現取決於使用的庫
            # 這裡提供概念性代碼

            node = self.client.get_node(f"ns=2;s={measurement}")
            values = node.read_raw_history(start_time, end_time)
```

```

        results = []
        for value in values:
            results.append({
                'measurement': measurement,
                'timestamp': value.SourceTimestamp,
                'value': value.Value,
                'quality': value.StatusCode.is_good(),
                'tags': tags or {},
                'fields': {'value': value.Value}
            })

        if limit:
            results = results[:limit]

        return results

    finally:
        self.client.disconnect()

    async def aggregate(self,
                        measurement: str,
                        aggregation: str,
                        interval: str,
                        start_time: datetime,
                        end_time: datetime,
                        tags: Optional[Dict[str, str]] = None) -> List[Dict[str,
Any]]:
    """
    聚合查詢 (使用 OPC HDA Aggregate 功能)
    """
    # OPC HDA 支援標準聚合函數
    # 實現細節省略...
    pass

```

## 4. 混合架構部署

### 4.1 典型部署場景

場景 1: PI 用於歷史數據,TDengine 用於新數據

```

# 數據路由策略
routing_rules:
- condition:
    time_range: "< 2024-01-01"
    target: pi_system

- condition:
    time_range: ">= 2024-01-01"
    target: tdengine

```

場景 2: PI 作為主數據源,NDH 作為分析層

PI System (主數據源)  
↓ (只讀同步)  
NDH + TDengine (分析層)  
↓  
AI/ML Applications

## 4.2 數據同步策略

```
# data_sync_service.py

class PIToNDHSyncService:
    """PI 數據同步服務"""

    async def sync_historical_data(self,
                                   pi_tag: str,
                                   start_time: datetime,
                                   end_time: datetime,
                                   target_measurement: str):
        """
        從 PI 同步歷史數據到 NDH

        Args:
            pi_tag: PI Tag 名稱
            start_time: 開始時間
            end_time: 結束時間
            target_measurement: 目標 Measurement 名稱
        """
        # 1. 從 PI 讀取數據
        pi_data = await self.pi_adapter.query(
            measurement=pi_tag,
            start_time=start_time,
            end_time=end_time
        )

        # 2. 寫入 NDH (TDengine)
        for point in pi_data:
            await self.tdengine_adapter.write(
                measurement=target_measurement,
                tags=point['tags'],
                fields=point['fields'],
                timestamp=point['timestamp']
            )

    async def setup_realtime_subscription(self, pi_tag: str):
        """
        設置 PI 即時數據訂閱
        """
        # 使用 PI Web API 的 WebSocket 訂閱
        # 實現細節省略...
        pass
```

## 5. API 使用範例

---

### 5.1 統一查詢接口

```
# 應用層代碼無需關心底層是 PI 還是 TDengine

from ndh.api import NDHClient

client = NDHClient(base_url="http://ndh-server:8080")

# 查詢資產遙測數據（自動路由到正確的數據源）
telemetry = await client.get_asset_telemetry(
    asset_id="PUMP-001",
    telemetry_name="discharge_pressure",
    start_time="2024-01-01",
    end_time="2024-01-31"
)

# NDH 會根據時間範圍自動決定從 PI 還是 TDengine 讀取
```

### 5.2 PI AF 資產映射

```
# 將 PI AF 資產導入為 IADL

from ndh.integrations.pi import PISystemAdapter

pi_adapter = PISystemAdapter(config)

# 獲取 PI AF 元素
af_elements = await pi_adapter.get_af_elements(
    element_path="\\\\PI-Server\\PlantAssets\\Area1"
)

# 映射為 IADL
for element in af_elements:
    iadl_asset = await pi_adapter.map_af_to_iadl(element['WebId'])

    # 註冊到 NDH
    await ndh_client.create_asset(iadl_asset)
```

---

## 6. 效能優化

---

### 6.1 快取策略

```
# 對 PI 查詢結果進行快取

from functools import lru_cache
from datetime import timedelta

class CachedPIAdapter(PISystemAdapter):
    """帶快取的 PI 適配器"""

    @lru_cache(maxsize=1000)
    async def query_cached(self, measurement: str, start: str, end: str):
        """快取查詢結果 (適用於歷史數據)"""
        return await self.query(
            measurement=measurement,
            start_time=datetime.fromisoformat(start),
            end_time=datetime.fromisoformat(end)
        )
```

### 6.2 批量讀取

```
# 批量讀取多個 PI Tags

async def batch_query_pi_tags(pi_adapter: PISystemAdapter,
                              tag_list: List[str],
                              start_time: datetime,
                              end_time: datetime):
    """並行查詢多個 PI Tags"""

    import asyncio

    tasks = [
        pi_adapter.query(tag, start_time=start_time, end_time=end_time)
        for tag in tag_list
    ]

    results = await asyncio.gather(*tasks)
    return dict(zip(tag_list, results))
```

---

## 7. 安全性考量

---

### 7.1 認證與授權

```
# PI System 安全配置

pi_system:
  authentication:
    method: kerberos # 或 basic, certificate
    service_account: ndh_service@DOMAIN.COM

  authorization:
    # 只讀訪問
    permissions: read_only

    # 允許訪問的 PI Points
    allowed_points:
      - "Plant1.*"
      - "Area2.Temperature.*"
```

### 7.2 數據加密




```
# 使用 SSL/TLS 連接 PI Web API

config = {
  'web_api_url': 'https://pi-server/piwebapi',
  'verify_ssl': True,
  'ssl_cert': '/path/to/client.crt',
  'ssl_key': '/path/to/client.key'
}
```



## 8. 實施路線圖


---

### 階段 1: PI Web API 基礎整合 (1-2 個月)




-  實現 PI Web API 基礎連接
-  支援時序數據讀取
-  統一 API 封裝

### 階段 2: PI AF 資產模型映射 (1 個月)




-  PI AF 元素讀取
-  AF 到 IADL 映射

-  自動同步機制

### 階段 3: Historian 整合 (1-2 個月)

-  OPC HDA 客戶端實現
-  支援主流 Historian 系統
-  統一抽象層

### 階段 4: 高級功能 (1 個月)

-  即時數據訂閱
-  數據同步服務
-  快取和性能優化

總計: 4-6 個月完整實現

---

## 9. 總結

---

NDH 對 AVEVA PI System 和 Historian 的整合,為企業提供了:

1. **保護現有投資:** 無需遷移 PI/Historian 數據,直接整合
2. **統一數據訪問:** 透過 NDH API 統一訪問所有時序數據
3. **靈活架構:** 支援混合部署,新舊系統並存
4. **語義化增強:** 透過 IADL 為 PI 數據提供情境化
5. **開放生態:** 避免供應商鎖定,保持技術自主權

這種整合策略既尊重企業現有的技術投資,又為未來的數位化轉型提供了開放、靈活的路徑。