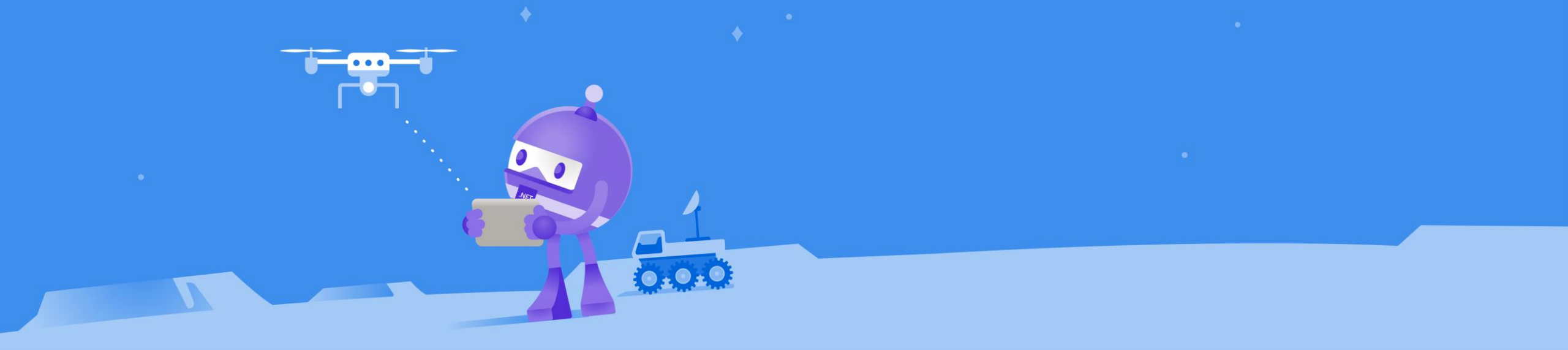
Microsoft STUDY4 Build School

.NET Conf

探索 .NET 新世界



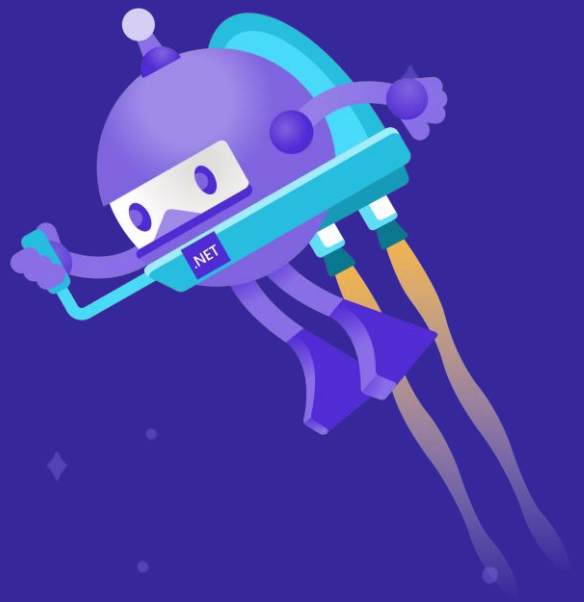
Microsoft STUDY4 Build School

大型異質資料庫移轉

胡百敬

byron.hu@mentortrust.com

集英信誠合夥顧問



我們移轉資料平台到 SQL Server 的經驗

- 政府：DW(Oracle)、核心應用系統移轉(Informix)
- 金融：防洗錢(TD 資料同步到 SQL DM)、轉換 ETL(Informatica to SSIS POC、DTS to SSIS)、SQL to SQL
- 電信：DW(TD)、應用系統移轉(Oracle)、整理中繼資料(TD、Oracle 到 Excel)
- 製造：應用系統移轉(MES Oracle)、應用系統卸載(MES Oracle)
- 零售：DW(TD)、應用系統同步(ERP SAP)
- 醫療：DM(Oracle DW 資料同步到 SQL DM)
- 運輸：應用系統同步(ERP SAP)

困難點

- 企業內 Data warehouse/大型應用系統的存在，通常少則 4~5 年，多至 20 年以上，代表經過數代工程師，多種年代下的：技術、需求、開發規則，不同的資料髒法
  - 移轉資料除了 ETL 平台，原來整資料的方式有 XXX-SQL、DOS cmd、Linux bash 指令、Perl、Python、Java、Excel 的 VBA...
  - 維護資料邏輯不同：PK、Archive、Error Handle、Log、Hash/Mask...以上諸多機制在各個資料表、程式物件的作法往往不同，雖然新系統規劃一致的作法，但要隨機應變，將來源各種作法統一到目的
  - 當代維護的工程師無法預先告知有多少陷阱
- 異質型技術間的差異沒有原廠，不一定 Google 得到，要深探資訊技術本身的原理、原則，且能提供應變解法
  - 團隊要有心理準備大規模重寫、重做。可能改寫上千個物件後，才發現犯了基本上的認知錯誤。全面重寫還不只一次，這不僅考驗個人 EQ，更測試團隊合作
  - 熟悉的技術解法在不同平台不一定找得到，除錯的方式也不相同

困難點

- 要能儘量無縫接軌前端的應用程式
  - Power BI、Analysis Services、Cognos、Tableau、R、Python
  - 舊有的交易系統、自行以 .NET、Java、PHP...開發的應用程式
- 改寫原有主要系統的支援系統
  - Audit 監控與警示
  - 資料上傳
- 將近萬個物件：資料表、資料庫內物件、ETL 物件、排程、程式...多種資料與物件的開發生命週期管理
- 穿著衣服改衣服，系統持續在變動，物件與資料都持續在改變，某個時間點後，原系統的變更需要在兩端一起做
- 企業越來越重安全，但整個 DevOp 生命週期中，大家卡安全的方式各有巧妙

為何要換

- 技術作古
- 原廠凋零
- 維護費用過高
- 高層人士更動/組織政策要求
- 汰換舊系統或新添系統的技術選擇

與 MS SQL 世界可能存在的差異

- 資料語言：T-SQL、TD-SQL、PL-SQL、Informix、PostgreSQL...
- 不支援的功能：
  - Oracle 的 Reference Cursor
  - 單筆資料更新而觸發的 Trigger，非以批次觸發
  - ...
- 資料類型差異：數值精準度、時間的 Interval 資料型態
- 編碼差異：ANSI、UTF-8、UTF-16、IBM EBCDIC、中文 EUC 碼
- 文字檔的欄(例如：0x06 ACK)、列分隔符號(例如：僅有 0x0A LF)
- 原系統的難字(自造字)
- ETL 平台：Trinity、Informatica
- 批次排程平台：Control-M、CPS
- 相關系統開發平台與語言：Linux、BASH、Telnet、FTP、SMTP、SAP、PTC Windchill、Java、Python、R 、SAS、Crystal Report、Cognos、Tableau
- 備份/還原、可用性
- 認證、權限、安全概念

移轉流程

- POC
- 建置環境：開發、測試、正式
- 普查舊系統
- 全部一次性移轉
  - 驗證
- 增補線上系統的持續變動：在新系統增修一次性移轉後的變動
- ETL 與資料物件持續性改寫
  - 持續測試、驗證
  - 部分一次性重轉
- 效能調校
- 使用者教育訓練
- 上線前中後
- 保固/維護

人員團隊配置

- 專案團隊
- 技術團隊
  - 新架構規劃、平台安裝、最佳化與維護
  - 全部與部分一次性移轉
  - 普查與驗證
  - 改寫資料物件(Stored procedure、View、Function、Trigger)
  - 改寫 ETL
  - 相關應用系統改寫
- 動態支援人力

POC 需要驗證的內容

- 移轉後的資料維持來源資料正確性
  - 並非跟原來資料一樣，歷史悠久的資料通常很髒：不可見字元、違反唯一、非字元碼、超過資料格式容量的值(例如：字串長度、數值精準位數、日期邊界)
  - 目標系統不存在來源系統的資料類型
- 達到原來平台的功能
- 符合效能要求
- 滿足架構需求：可用性、安全(DW 可能有複雜的授權、稽核要求)
- 其他系統的存取：存取原資料庫的各種異質前端系統
- 比較不同解決方案的優劣：各家產品、雲/地差異、機房配置

新架構規劃

- 比較新舊系統的平台與技術差異，提供新的架構
- Sizing
- 大型資料架構的資料庫擺放、分割、切月/年、Archive 、壓縮
- 存取方式：多 DB、Schema、View
- 建置/維護索引
- 安全：帳戶、角色、權限、加密、Mask、稽核
- 維護、監控
- 備份/還原、可用性規劃
- 壓測

普查

- 整體系統架構、網路拓樸、窗口負責人資料(兩方建立 Line 群)
- 上游來源系統、下游應用系統
- DB 物件
- 資料
  - Schema
    - 以正確性為主，除欄位、型態，還有 default、check、trigger...
    - 效能相關設計如 Partition、Index 可參考，但大多因為系統間差異而需要重新規劃)
  - 資料量、紀錄筆數
- ETL 平台物件
- 物件數量與內容在開發期程內仍持續變動，需定期普查
- 批次排程
- 原程序執行耗用時間
- 相關應用程式可能使用的語法(Ex：應用程式常自行組織執行的 SQL 語法)

移轉資料

- 要小心編碼，例如 TD 的 ODBC 回傳；只會全部 ANSI 或全部 Unicode
- 不同系統間，資料用以排序的方式不同
- 平行移轉資料表，但要能動態調整
  - 平行數量：會有效能 bottleneck：
    - 來源、目的平台的 CPU、RAM、Disk IO、Network 極限
    - 軟體限制：輸出量、平行上線存取量、單一連接可用的資源(如 TD 的 Spool space)
  - 可執行時間：移轉的作業不能干擾線上作業
- 可重複對特定目標群移轉
  - 所有的資料表要先一次性移轉
  - 因為線上系統持續更動，ETL、資料庫物件開發驗證造成資料錯誤，要能針對目標群資料表排程移轉
- 錯誤處理
  - 為了效能放棄錯誤紀錄
  - 為保留錯誤而犧牲效能：設計中介轉換(可找出違反 constraint)，或單筆處理(髒資料)
- 若放棄舊平台，則移轉資料有舊系統下線的時間壓力

改寫物件

- 要有跨技術的先探
  - 在時程允許下，儘量找出系統間的技術、邏輯差異
  - 跨團隊開發的盲點，例如：可能 ETL 執行驗證時(後)，才發現物件(先開發)間的依存關係完全錯誤。或是開發完了，才發現不合組織的部署規範
- 跨平台的 ETL 是對程式寫作的考驗
  - Linux 上的排程批次作業與 Windows 有根本的差異，最好熟悉 PowerShell 來串起一切
- ETL 的 Job 可能有因果關係，要記錄來源與目的資料表，以追溯錯誤資料
- 最好有 .NET 程式設計師以備不時之需

ETL 架構

- 上百個排程，成千個物件
- 來源多樣：csv、excel、dbf、Oracle、SQL Server、ftp
- 集中設定、集中部署、集中紀錄
- 雖然有周期新增、全刪全加、區段更新...等各種形式，盡量設計批次執行，Table Lock、無索引
- 排程發起端可能不在 ETL 自己本身的平台，例如：來自集中式的 Control-M、CPS
  - 要配合排程管理系統的呼叫方式
  - 執行結果要交還排程管理系統
  - 跨系統執行的帳戶權限
- 資料庫平台是 SQL Server，但 ETL 平台沿用舊架構
  - 結合 ETL 系統的呼叫與紀錄，例如執行 Console 批次或 T-SQL 的結果要呈現在 Trinity 內

驗證

- 持續排程自動驗證與比對
- 資料
  - 基本驗證：移轉流程正確、筆數一致
  - 進階驗證：
    - 不同資料類型的 Checksum，文字資料類型固定長度、不同長度的 Checksum
    - 跨欄位的 except
- 物件
  - 數量
  - 執行結果的驗證
  - 執行花費時間
- 進行中的線上系統可能造成驗證資料的時間差
- 各種執行結果，如：批次排程工作、一次性移轉工作...所有的執行結果可以存在資料表，彼此可以串聯
- 物件量太大，各種安全限制，可能要商請終端使用者一起參與

效能調校

- 原本的架構(例如索引設計、交易作法)可以參考，但需要驗證過再決定是否遵循
- 除了一般的效能調校技巧，語法與功能需要與使用者確認邏輯，重新改寫可能較佳
- 因資料量大，使用者多，分析語法複雜，系統上線初期要小心效能

具備...比較好

- VPN 登入開發，例如：
  - 移轉一個 6TB 的 Table，很難預估 User DB 和 Tempdb 將耗用的 MDF、LDF 量，單一資料表一次移轉、建 clustered index...通常超過 24 個小時，需要監控並及時配置系統容量
  - 數千個資料表平行移轉，每天早上 10:00 到晚上 10:00 持續一個月，隨時出錯，需要補正後重轉
- 存取來源/目的系統有夠大權限(盡量不求人)
- 可建立 Linked Server 到相關來源系統，不管是 DB/DW 或 ETL 平台的 DB
- 來源、中繼、目標的容量越大越好，不管是 CPU、RAM、Disk 或 Network
- 兩方團隊可共稿，有太多的訊息需要交換和追蹤

https://byronhu.wordpress.com/2020/03/21/%e7%ad%89%e5%be%85-2/

Q&A

.NET Conf 2020

特別感謝

91APP Technical Network

KKTIX

JET BRAINS

HackMD

MVP Microsoft® Most Valuable Professional

Microsoft

Build School

STUDY4 為學習而生

以及各位參與活動的你們