作者序

隨著大數據 (Big Data) 與人工智慧 (AI) 的技術蓬勃發展，加上網路與硬體的革新也在持續地突破，未來能夠有效地搜集數據做出價值應用、分析和預測已逐漸成為各個產業與領域的必備要領、能力和應用。為了因應這樣的場景和有效地處理大量資料，達到即時性效果，此時對應的資料工程架構會顯得非常重要。

資料工程 (Data Engineering) 這個領域的知識其實非常的深且廣，不如同一般所說的就僅僅是資料清洗與前處理這樣的任務，它會牽扯到批次 (Batch)或是流 (Streaming) 排程處理、分散式架構處理、資料一致性、網路附載平衡、資料副本與修復、資料版本控制、資料欄位驗證、資料品質監控等各種環節，必須要具備這些相關知識才有機會建置出相對可靠地資料流架構 (Data Pipeline)。

該領域牽扯到的知識非常地多樣，以至於近幾年有許多工具與服務都被推出、甚至更加成熟。例如 Apache Airflow、Apache Kafka、Spark、Apache NiFi、Apahce Druid、Clickhouse、Cassandra 等各種開源工具都被推出，除此之外，我們所熟悉的雲 (Cloud) 服務也會定時推出對應的應用來解決各式各樣的問題，例如 AWS、GCP 和 Azure 等各平台，所以在這樣的環境驅使下，不難發現資料面向的架構與應用逐漸地被各個產業重視。再加上當今的社會都十分重視人工智慧的應用，也堅信著這項技術能帶給社會一定程度的回饋與影響力，但在這條技術的路上，我們除了專精與研究模型的演算法之外，更值得注意的是資料本身的特性，若沒有一定程度地可靠且穩定資料流架構，就很容易造成所謂地 Garbage-In-Garbage-Out，最後就會無法帶來實質上地商業幫助。

碁峰 www.gotop.com.tw

本書主要以介紹 Apache NiFi 這套工具作為主軸，該工具主要用來建置 Data Pipeline 的應用，我們可透過該工具決定什麼時候從來源端資料讀取，且有內建許多元件可以讓我們快速執行資料處理，接著落地到我們預期的目的端資料載體。另外也同時支援 Batch 和 Streaming 的處理，以利於未來場景需要切換時可以彈性地切換。

除了介紹 Apache NiFi 的架構、工具建置與操作方式之外，本書也會帶出資料工程的常見架構與概念，例如 Lambda、Kappa 和 Delta 架構、或 Streaming 的應用等，讓讀者們在閱讀本書時，除了在學習該工具之外，也能對於資料工程面較陌生的讀者們也能有一些初步地認識，以利於整本書的知識吸收。因此，只要好好地瀏覽，能從中體悟到資料工程的基本核心與精神，以及為何有 Apache NiFi 這套工具的出現，和最後要解決的議題。

蘇揮原

碁峰 www.gotop.com.tw

1.1 何謂 Data Pipeline？

在實際的開發架構上，我們會需要將上線的服務或應用程式、第三方資料以及商業需求的相關資料經過一段流程處理之後，最後落地到一個地方，來讓後續的分析或下游任務可以再接下去做執行。其中，這段流程處理主要目的就是『讓原始資料 (Raw Data) 轉換產生成更有價值的資料 (Valuable Data) 』。可以如何做到這件事情呢？最基本的做法就是將資料經過資料搜集、資料清洗、資料格式轉換、資料事前計算或統計、資料落地等階段執行，如此一來當下游任務或使用者在使用資料時就比較不會有資料偏差 (bias)、或不完整與不一致性的狀況發生，進而建立出來的服務、分析與決策也會有可靠性和可信度。

Data Pipeline 就是在這樣的情境下所產生出來的流程設計問題，底下會需要有很多子模組 (Submodule) 或稱步驟 (Step) 來一步一步串聯出來，告知資料當下這一步應該要做什麼處理，一旦處理完之後再往下一個流程繼續執行。可以把它想成類似的生產線流程，以製造汽車的工廠為例，一條製造汽車的生產線流程中，會經過多個不同的作業任務來串連而成，例如要先衝壓車身零件、接著焊接、塗裝、總裝、到最後的檢測，基本上每一個流程之 Input 都是接續著前一個工作任務之 Output，最後做到完整的產出；當然有些 Step 是可以同時運行的，甚至需要等到上游的全部處理完之後才能處理當下的任務。一樣再拿汽車流程為例，若要焊接勢必要等到車身、車頂、車蓋等都有了才能做焊接的動作。因此，Data Pipeline 也是同樣的道理，簡單的設計也可以很簡單，但如果要複雜的設計也能做到，所以一切取決於資料的使用場景與資料流流程規劃。

上述是以簡單的文字來描述，接下來用圖來簡單說明 Data Pipeline 的流程，我們可以從圖 1-1-1 中看到有三個主要元件 (Componenets)—Source Data、Data Pipeline 和 Target Data。簡單來定義一下這三個名詞：

- **Source Data**：原始資料或資料取得的地方，也是整個流程的出發點。常見的可能像是 File (csv, parquet 等)、Database、DataLake 等資料載體，有時甚至能直接從 API 拿到原始資料就直接做處理。

碁峰 www.gotop.com.tw

- **Data Pipeline**：定義資料處理流程的框架，會包含許多 Steps 來做設計。
- **Target Data**：資料經過一連串處理完之後，要落地且儲存的地方。常見的是 File (csv, parquet 等)、Database、DataWarehouse 等資料載體。

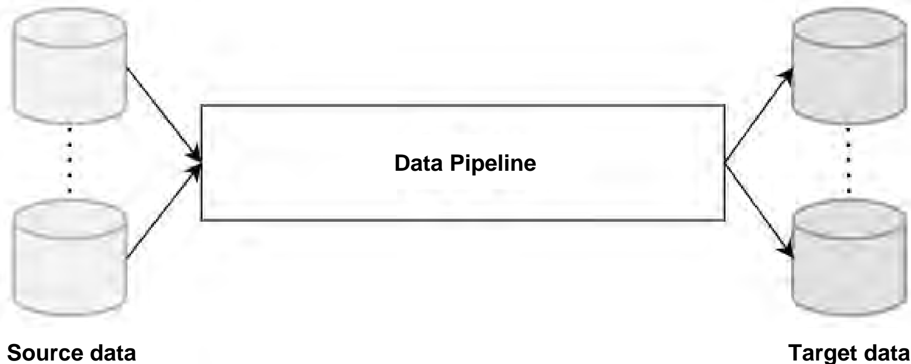

圖 1-1-1　Date Pipeline 架構示意圖

其中 Source Data 和 Target Data 通常就是存放資料的地方，以 Data 領域來說，比較常見的有如下類型：

- Local File (ex. csv, parquet, json 等)
- RDB (ex. MySQL, MariaDB, PostgreSQL 等)
- Document DB (ex. MongoDB 等)
- NoSQL (ex. Cassandra, ScyllaDB 等)
- Search Engine (ex. elasticsearch 等)
- Data Storage (ex. AWS S3, GCP GCS, HDFS 等)
- DataWarehouse (ex. AWS Redshift, GCP BigQuery 等)
- Message Queue (ex. Apache Kafka, AWS Kinesis, GCP Pub/Sub 等)
- SQL Engine (ex. Presto, AWS Athena 等)
- API (ex. 透過請求 API 將資料轉移到下一個服務)

碁峰 www.gotop.com.tw

因此，只要能存放資料的地方，我們就能對它做資料的讀寫，再經由 Data Pipeline 的轉換來做處理，最後落地到使用者或下游任務能存取資料的地方。

接著再把圖 1-1-1 放大成圖 1-1-2，我們就可以進一步地想像在 Date Pipeline 當中其實會有著非常多的 Steps 或 Sub-Module 組合而成，有些很單純，有些則是複雜的 Dependency，也意味著需要上游 (Upstream) 都完成了才可以執行。例如 step 2-2 代表它需要 step 1-1 和 step 2-1 處理完後的資料才能再接下去執行，所以在執行前就必須等待上游的這兩個 Steps 都執行沒問題時才能運作。

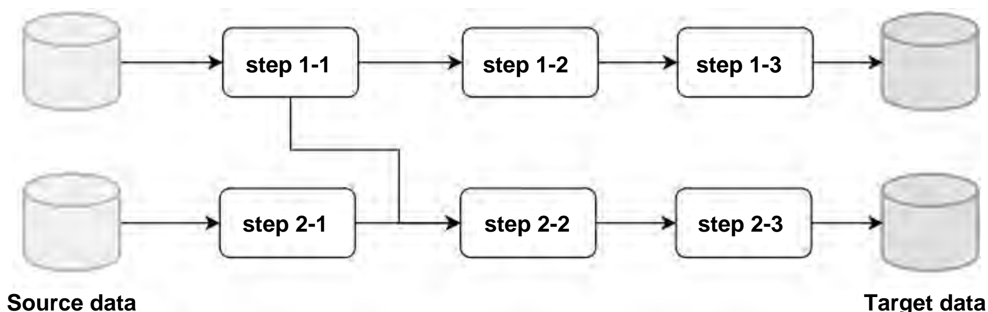

圖 1-1-2　詳細的 Data Pipeline 示意圖

透過這樣的說明，相信讀者對於 Data Pipeline 的定義有建立出基本的概念與認識了。然而，在 Data Pipeline 的設計與實務當中，有兩個主流且最大眾的架構，分別是 ETL 和 ELT，雖然從文字上看起來就是 T (Transforme) 和 L (Load) 順序做反轉，但是對應的實務情境需求也有所不同，接下來簡單說明這兩個架構概念。

1.1.1　ETL (Extract-Transform-Load)

ETL 顧名思義它的執行順序是 Extract、Transform、Load，需要存放資料的地方取得資料出來 (Extract)，接著做邏輯處理與轉換 (Transform)，最後存放到目標資料存放地 (Load)。下圖圖 1-1-3 為 ETL 圖示：

碁峰 www.gotop.com.tw

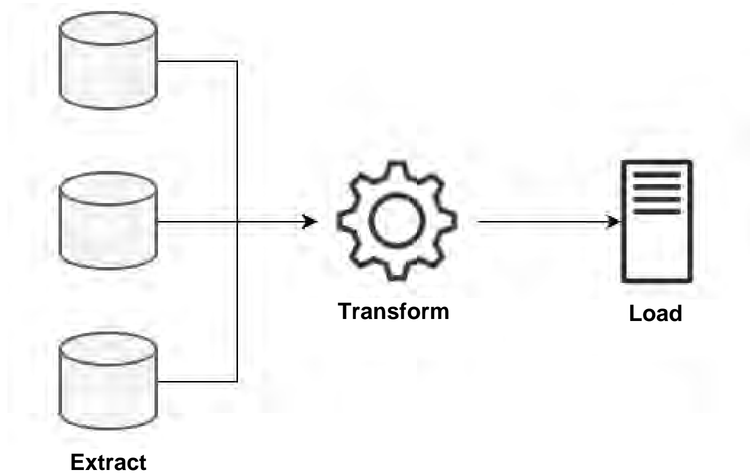

圖 1-1-3　ETL 示意圖

當選擇這樣的架構時，我們需要事前定義好 Transform 的邏輯與流程，以及資料的進出，否則資料會卡在中間那一段就會無法順利寫入到目標資料存放地。此外，當選擇這個架構時也意味著把資料轉換處理這段交派給專責的開發人員或團隊，因為後續任務或使用者不需要掌握流程邏輯，但是如果邏輯改變或新增時，就必須要與專責團隊做溝通，所以也會需要額外的溝通成本。因此我們可以對 ETL 整理出以下幾個特性：

- 必須事前定義好 Transform 邏輯，以及資料的 Input/Output。
- 中間的 Transforme 可能會需要仰賴額外的 code 來實作，ex. python、spark 或 scala 等，相對較多開發的時間與成本。
- Transform 會由額外技術來實作，較易達到 Streaming、Micro-Batch 和 Batch 的設計模式。
- 會有專責團隊做處理，在資料權限控管上比較嚴謹。
- 當新增或變動邏輯時，需要與負責團隊來回溝通，較有溝通成本。

碁峰 www.gotop.com.tw

1.1.2　ELT (Extract-Load-Transform)

ETL 它的執行順序是 Extract、Load、Transform，代表著從存放資料的地方 (Extract) 先轉存到另一個資料載體(Load)，再從直接在資料載體上面做邏輯處理與轉換 (Transform)。下圖圖 1-1-4 為 ELT 圖示：

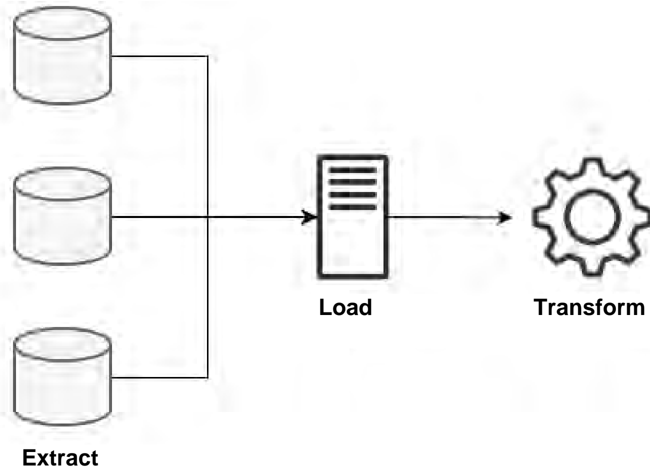

圖 1-1-4　ELT 示意圖

選擇該架構時，意味著會有一個類似資料中心或統一取得的地方，接著可能因為部門或團隊的需求而有所不同，需要將各自所需的資料轉存到各自的資料載體，接著再從上面做資料邏輯與轉換。這邊要留意的是這裡的轉換會是在載體做處理，舉例來說，如果是 load 到 OLAP (ex. AWS Redshift, GCP BigQuery 等)，而使用者就直接在服務上透過 SQL 方式來做轉換處理與設計邏輯。這樣做代表由各個團隊自行決定資料處理邏輯，當有需要改變或變動時，溝通成本較低且彈性較高。不過因為需要讓所有團隊可以做到轉存的動作，所以對於資料的權限管理上就會沒那麼嚴謹，因為每個團隊都有權利對資料中心做操作。此外，對於技術能力沒有那麼深的團隊來說，他們就有可能會直接在資料載體上做處理或是運算，對於載體的運算能力也會相當吃重。因此我們可針對 ELT 整理出以下特性：

- 直接在目標 Load 的資料載體做 Transform 的邏輯 (ex. SQL)，但就不容易做 Streaming 的操作，通常只能 Batch 處理。

碁峰 www.gotop.com.tw

- 彈性較高且較少溝通成本，因為由各個負責人員處理。
- 資料權限管理上較麻煩，需要更嚴格去把關。
- 對於非工程或技術的人員，有可能會直接載體上做處理，所以需衡量載體服務的運算能力。

講述了 ETL 和 ELT，實際上是哪一個架構最好的，並沒有標準答案。因為完全取決於實務的使用場景，但大多數不會特別單獨使用，尤其在一定規模的企業當中，在資源與成本上的考量，大部分都會走上混合 (Hybrid) 的方式使用。例如針對主要資料做 ETL，對於部門自行或是客製化的需求再做 ELT，藉此盡可能達到資料的最大價值與效益。

1.2　何謂 Streaming 和 Batch？

前一個小節介紹了 Data Pipeline 的流程定義與目的，也分享了 ETL 和 ELT 常見的架構。但除了設計好 Pipeline 的流程之外，還會有一件很重要的事情需要被探討，也就是要多久去觸發執行？是每天執行一次、還是每小時、每分鐘、還是即時性地處理，也就是有資料上來就開始執行。在時間粒度 (Time Granularity) 上的選擇是沒有絕對的，可以一天執行一次，但就會面臨每次執行的量級可能會很大，因為是處理前一天整天的資料量，同時可能會影響到企業的決策與分析效率，因為必須要等到隔天才會有今天的數據；相反地，如果選擇秒級、甚至微秒級的資料處理，也就是當下的時間點有資料時就應該要立即能分析到該筆資料，甚至做應用。但如果 Pipeline 的處理速度趕不上的話，就會有可能產生資料延遲的風險。

因此這議題通常取決於企業場景需求以及 Pipeline 它能處理的速度以及能掌控的資料量級。舉例來說，假設企業期望能達到秒級的資料分析，但如果中間的處理邏輯目前需要執行需要 5 分鐘才能完成，那就會造成資料的延遲，進而不符合企業的決策需求。所以要如何決定觸發時間的這項議題，往往會是一個權衡 (Trade-Off) 的問題。一方面考量商業需求，另一方面也要考量目前工程架構能否做到配合 (Fit)，所以當中的協調與配合就會是非常重要的議題。

碁峰 www.gotop.com.tw

基於這項議題，本章節會講解整理目前在 Data Pipeline 上常用時間處理架構，分別是 Streaming、Batch 和 Micro-Batch。當然目前有一些成熟的技術工具或服務可以做到這些事情，但如果我們先了解這些真正核心的概念之後，再套用到對應工具服務上就會更明確知道如何設計符合對應場景的 Pipeline 架構流程。

1.2.1 Streaming

在 Data 領域當中，Streaming 代表著資料會即時性且持續地產生與流入，因此後續要有對應的高效能處理能力才不會造成資料延遲，也就是要做到保證資料送達 (Guaranteed Delivery)，並且同時要避免資料遺失以確保每一筆產生的資料都能完整地被搜集且處理到。簡單來說注重即時性 (Realtime) 和低延遲 (Lower-Latency)。

可以從圖 1-2-1 來看到 Streaming 的做法，從圖中可以看到 2022/09/01 13:00 有一筆資料產生，以 Streaming 的定義上，他應該也要在 2022/09/01 13:00 時搜到資料切寫入到 Database，下一秒服務或是在做資料分析時就可包含到該筆資料；同理可證，若 13:05 有資料產生時，也會同時間看到該資料的產出。因此資料的產生到落地的過程中是不會有所謂的資料延遲問題發生，但通常為了達到這個目的會需要龐大、複雜與高效的處理架構才能做到。

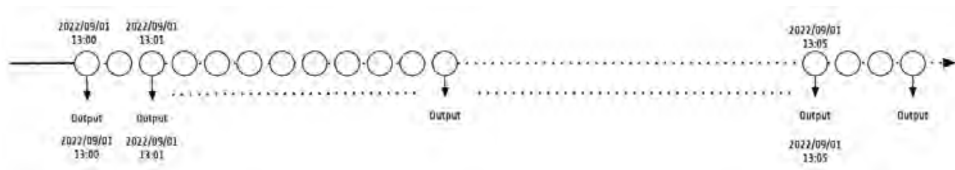

圖 1-2-1 Streaming 示意圖

1.2.2 Batch

Batch 的架構中是允許資料延遲，因此時間粒度相較於 Streaming 來得大，可以分鐘、半小時、小時、一天、甚至也可以一週到一個月，其中處理的

碁峯 www.gotop.com.tw

資料量級會隨著時間粒度越大而成正比。此外，當前時間的資料不會立即地處理與分析到，必須等到下一個觸發時間時才會取得。

這邊以圖 1-2-2 的 Hourly 來做舉例，假設每天整點做 Pipline 的觸發，2022/09/01 13:00 會觸發 Data Pipeline，其中會處理的資料之時間區間會是從上次的觸發時間到該次觸發的時間，也就是取得 12:00 到 13:00 這段時間做處理，下次觸發時就是 13:00 到 14:00，以此類推。因此在 Current 的圓點所產生的資料，就需要等到下次觸發的時間 15:00 時才會拉到該筆資料。

圖 1-2-2　Batch 示意圖

過往由於技術上的考量與限制，大多數的資料處理和應用都會以 Batch 方式做執行。但近幾年隨著 AI 和 ML 以及相關分析的技術成熟，越來越多商業應用著重在即時性這件事情上，以及為了提升企業的高速擴張與規模成長，和提升市場競爭力，資料的價值應用也必須面對到即時性的處理。以我的觀點來看，未來 Streaming 一定會成為主流與主要的設計方式，甚至是標配，而 Batch 架構就會逐漸變成不是主要的設計原理與架構。若讀者們能現在開始培養這部分的技能，未來在做 Date Pipline 或資料工程相關開發時，必會相對有競爭力。

1.3　何謂 Lambda、Kappa 和 Delta 架構？

在前一章節我們介紹了 Streaming 和 Batch 的原理，但回過頭來套用到我們在設計 Data Pipeline 的架構上，看起來似乎只能在 Streaming 和 Batch 中選擇一個做為處理。但實際上在 Data 領域當中有些架構提倡混合 (Hybrid) 的方式做實現，只是過程中會有許多問題要去克服和解決，因此本節帶讀者們來看一下過往在這部分的數據架構的演變，進而到現在有哪些技術性的突破與發展，讓我們在現在這個時間點，對於這部分的技術可以更加成熟。

碁峰 www.gotop.com.tw

1.3.1 Lambda 架構

Lambda 架構最早是由 Storm 的創辦人 — Nathan Marz 所提出來的。該架構主要採用了兩套層級做並行，分別是 Batch Layer (也可稱為 Offline Layer) 和 Realtime Layer。其中 Batch Layer 主要利用 Hive、Impala 等框架來做線下的 Batch 處理；而 Realtime Layer 則用 Spark Streaming、Flink 等技術做處理。但我們可以從下圖圖 1-3-1 來看他真實的架構核心與精神：

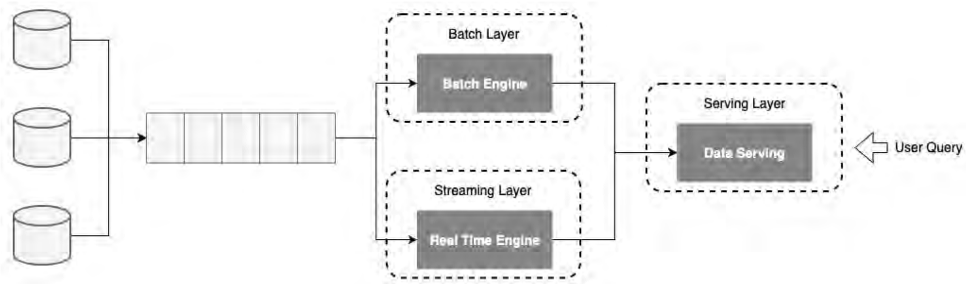

圖 1-3-1　Lambda 架構圖

從架構圖上來看，其實就是在資料蒐集進來之後，會由兩個 Layer 來做接收：

- **Batch Layer**：用來處理 Offline Data，代表著全體的資料集。
- **Streaming Layer**：僅用來處理隨著時間增量的資料。
- **Serving Layer**：用來合併 Batch Layer 和 Streaming Layer 結果以產生最終的資料，而 User 直接對該最終資料做 Query。

若這邊將該架構圖套用對應可使用的工具服務，可以看到如圖 1-3-2。可以看到前端可以用 Kafka 來做一個資料緩衝的地方，後面可以利用 Storm 來做增量處理，而 hadoop 可以做一個定期的 Batch 資料處理，這兩層處理完會將資料統一進入到 DB 中的 Table，最後來讓使用者 Query 與分析。

碁峰 www.gotop.com.tw

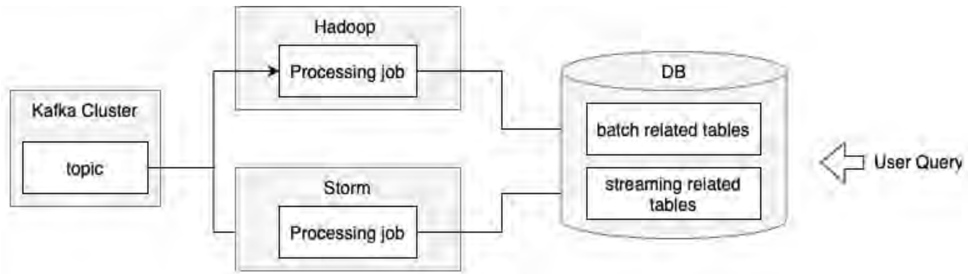

圖 1-3-2　套用到 Lambda 架構的工具範例

從圖上看起來好像是 Batch Layer 和 Streaming Layer 會是同時接收，但其實兩個 Layer 的觸發時間可以獨立切開。例如 Batch Layer 可以每天晚上處理，Streaming Layer 則持續接受資料處理。但這樣的架構會面臨到一個主要的問題：

◆ 需要維護兩套不同的框架 — Batch 與 Streaming

主要這兩個 Layer 要處理的核心問題就有所不同，Batch Layer 要處理大量級的資料問題；Streaming Layer 要處理低延遲的問題，所以這兩個核心的設計架構與流程就會有明顯的不同，對於工程師和相關開發人員來說，這樣會造成維護上的困難，以及如果未來資料處理邏輯做變動或新增時，程序上可能會變得十分複雜。

1.3.2　Kappa 架構

我們在前面的介紹已經知道 Lambda 架構在維護上有一定的困難程度，所以後續 Linkedin 的 Jay Kreps 等人團隊提出了一個 Kappa 架構。他們認為與其利用兩套的方式來做處理，不如集中或專注在一個框架去執行就好，且加入額外的處理來達到可以對歷史資料做計算。因此他們保留了 Streaming Layer，並多了 History Data Storage 在該 layer，讓如果資料有問題時起碼還有資料可以暫存且重複計算。因此架構可以參考圖 1-3-3。

碁峰 www.gotop.com.tw

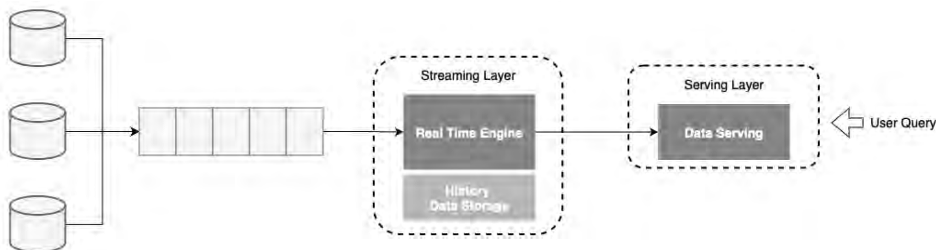

圖 1-3-3　Kappa 架構

所以從架構來看，Kappa 相較於 Lambda 來說，架構上更為統一，同時也比較好去維護。中間的 History Data Storage 通常可透過 Kafka 來做處理，因為 Kafka 有 Persist Duration 來決定資料可以保留多久，可以設定成 forever 進而永久保存資料。但在這樣的架構中仍然有一些缺陷：

- 對於歷史資料的處理仍然有限，因為 Streaming layer 通常用於增量處理比較有效率，對於過往歷史資料的重新計算會相對複雜

在先前的 Lambda 的架構中，是有一個專責的 Batch Layer 可以過往的所有歷史資料做計算與處理。Kafka 也可以做到類似的事情，但相對於傳統 Lambda 架構來說，它對於歷史資料的處理確實欠佳，所以即便統一個與簡化了架構，一旦需要歷史資料處理時的場景就會顯得相對不足。

為了能夠區分 Lambda 和 Kappa 的特性，這裡簡單整理了表格來比較當中的差異：

	Lambda 架構	**Kappa 架構**
資料處理	能處理 Streaming 資料之外，同時有單獨的 Layer 可以處理大規模的歷史資料。	對於歷史資料處理有限。
維護成本	成本高，需要維護兩套系統，因此對於機器服務的成本較高。	成本低，僅需要維護一個框架。
技術難度	因為有兩套系統，所以複雜性較高。	僅有單一系統，複雜性較低。

碁峰 www.gotop.com.tw

1.3.3 Delta 架構

接下來介紹 Databricks 在 2019 年提出來的 Delta 架構，主要是透過他們自行研發的 Delta Lake 來取代 Lambda 架構。我們都知道傳統的 Lambda 架構有著要維護兩套系統的困難度，同時也增加人力與資源的成本。而 Delta 架構就是要解決該問題，它是由 Spark Structured Streaming 與 Delta Lake 的整合出來的概念架構。

Delta 架構具備以下特性：

◆ Streaming 和 Batch 合併，不需要個別維護系統

◆ 可隨時重新處理歷史資料

◆ 具備彈性地儲存與計算的資料擴展

簡單來說，透過 Delta Lake 這套框架，可讓 Streaming 和 Batch 的寫法統一，不需要獨立設計，直接在指定的參數做對應變化，程式邏輯無需更動，就可以自由地在 Streaming 和 Batch 當中做切換。此外在寫入時 Delta lake 會產生 transaction log 來記錄每次資料的版本，當有發生問題時可以在回溯先前的資料做重新運算，所以在處理的邏輯與程式設計上就會變得十分單純，同時也能兼顧 Streaming 的增量處理與全量級歷史資料處理。只是在這樣的架構下就需要透過 Spark Structured Streaming 來做實作與設計，但這部分詳細實作就不在本書的範疇，有興趣者可以近一步去學習與了解，這邊提供目前技術與架構的發展供讀者們了解與認識。

1.4 為什麼需要使用 Apache NiFi？

前面讓大家對於資料的處理架構有了基本認識，我們從何謂 Data Pipeline 到 ETL、ELT 的介紹，再來介紹 Streaming 和 Batch 的定義與用途，最後到更上一層的架構介紹，包含 Lambda、Kappa 到近期的 Delta 架構。而建立在這些基礎上有許多延伸的工具服務，雖然有些工具服務看起來類似，但取決於一開始它們想解決的目的不同而有個別合適的使用場景。然而，回歸到本書的核心內容——Apache NiFi，該服務就可以讓我們可以設計出 Lambda 和 Kappa 等架構，藉此來應變我們的使用場景。

碁峰 www.gotop.com.tw

Apache NiFi 主要被用來針對資料流 (Data Flow) 這件事情來做處理，它也支援多種類型 Datasource，無論 Cloud 服務或是 Open Source，並且能以 streaming 或 batch 的方式來設計 Data Pipeline，本身也有內建提供完整的 Processor 來做資料轉換，所以對於資料的處理或格式清洗上有一定的幫助。再者，除非遇到較複雜的處理邏輯之外，通常都是可以直用拖拉的方式與參數設定的方式就可建立好 Data Pipeline，大部分無需額外寫 Code 來做處理，除非將 Apahce NiFi 視為 WorkFlow 工具，也就是作為 job 的順序流程控制。但這邊以 Data Pipeline 的角度來做介紹與使用，較符合該工具的理念與核心價值。

1.4.1　什麼是 Apache NiFi？

Apache NiFi 是美國國安局 (NSA) 開發且在 2014 年貢獻給 Apache 的頂級專案之一，它被稱作一種 Data Pipeline 或是 Data Flow Control 的工具服務，我們可以透過它去建立 Streaming 或 Batch 的資料處理流程，其中建立過程中是屬於 No-Code 的方式去建立，因為有提供完整的 WEB UI 介面可以操作，同時也可以做到高擴展性，以至於面對大數據處理可以更有效率。它也具備一個最重要的特性是可以做到資料追蹤，大部分的 Data Pipeline 工具主要只做到 job 的追蹤，但 NiFi 是可以追蹤到每一筆資料的變化與輸出後的狀況，所以這有助於在做 Streaming 時可以更加確認處理邏輯。

然而，關於 Apache NiFi 的特點可以列舉如下：

- 支援 AWS、GCP、Azure 等 cloud 平台的所有 db 和 storage 相關服務，ex. S3、GCS、PubSub、Knesis 等。
- 支援 Open Source DB，ex. Cassandra、MySQL、MongoDB 等。
- 主要透過 Web UI 視覺化之設定與拖拉的方式即可建立 Data pipeline。
- 可支援 Batch 和 Streaming 做切換。
- 具備 Monitoring 機制來監控 data 的處理狀況。

碁峰 www.gotop.com.tw

- 具備 Module Group 的機制，讓 Pipeline 變成模組來重複使用。
- 可建置成 clustering(叢集)，來使大量資料做 load balance。
- 擁有子服務 - NiFi Registry，可針對 Data pipeline 達到像 Git 一樣的版本控制，讓使用者可以對 pipeline 做 commit 與版本轉換。
- 可搭配自己撰寫的 Script (ex. python, golang 等) 去做資料處理與控制。
- 容易追蹤每筆資料的流向與轉換狀況。
- 對於錯誤處理有提供完善的措施，ex. Replay 來重新執行資料。
- 如果 Pipeline 有問題時，不需要停止 Pipeline 的運作，只要針對特定的 Processor 作處理即可。

從上述的列點看起來好像 Apache NiFi 優點許多，有很多對於 Database、Data Lake、Data Warehouse 等都有計有的元件可以做使用，而且對於跨平台或是自建的服務都有良好的兼容性與整合。但其實也有一些缺點需要做克服，這邊也列舉如下：

- 介面需要設定的參數較多，涵蓋的 Componenet 也較多，需要搭配官方文件來做實作較容易理解。
- 因為是 Open Source，所以需要 Self-Hosted 和 Self-Maintain。
- 在 Clustering 的狀況下，因為需要透過 Zookeeper 做 Node 連線觀察，如果要加入新的 Node，需要將服務做 Downtime，更改設定後再重新啟動。
- 台灣的 Community、Use-Case 和資源相對國外較少，所以遇到問題時需要花點時間排查。

碁峯 www.gotop.com.tw

1.4.2　Apache NiFI 的元件介紹

這邊會先簡單帶出 Apache NiFi 的基本元件，這邊可以透過圖 1-4-1 來搭配描述：

圖 1-4-1　NiFi 元件操作圖

◆ **FlowFile**

何謂 FlowFile？我們可以想像是資料中或是 File 中的一筆 Record，甚至是一包資料同時含有很多筆 record，今天假設有一張 Table 且其中有 100 筆資料時，當 NiFi 從中讀取時，這 100 筆 Record 就會在 NiFi 產生

碁峰
www.gotop.com.tw

100 筆 FlowFile，而 FlowFile 會帶有自己的 attribute 和 content，這兩個有什麼差異呢？

- attribute：可以想像成是 metadata，以 Key-Value 的方式來對 FlowFile 的描述，包含 size、path、permission 等
- content：真實 data 的內容，可能是 csv、json 等格式。

◆ **Processor**

想像成是一個邏輯處理的 Unit，在 NiFi 中它提供了許多內建的 Processor，這可使我們透過設定的方式來產生、處理、轉換、輸出 FlowFile 等相關操作。

因此，我們可以從上圖看到在一個 Pipeline 當中，FlowFiles 會經過中間多個 Processor 的處理，第一個 Processor 會被用來產生 FlowFiles (ex. 讀取 DB 或 files)；而最後一個通常會是一個落地輸出的 Processor (ex. 寫入 DB 或 files)。

◆ **Connection**

在 Apache NiFi 建立 Data Pipeline 時，其實會透過一連串 Processor 來建置，Processor 彼此之間會建立一個關係，就稱作為 Connection，可從圖中看到 Processor 之間一定會有個 Connection 的存在。

在 Connection 當中，我們可以透過設定來描述該 Connection 的定義，圖 1-4-1 中可看到常見的 Success 和 Failed，若今天有 FlowFiles 是走 Success 的 Connection，也就代表上一個 Processor 是處理成功的。因此，我們可自行建立多種 Connection 來決定每一條下游路徑的狀態與意義。

此外，在 Connection 中我們還可以設定 Queue 的狀態，像是 FIFO 或是 New First 等，來緩解當 Connection 中因有太多 Flowfiles 時所導致效能的問題。詳細的設定會在往後的章節再作進一步說明。

碁峰 www.gotop.com.tw

◆ **Processor Group**

Processor Group 通常被用來作為 Processor 的 Module，為 Processors 的集合。通常會有 3 種情境需要 ProcessorGroup：

- 今天假如有兩個 pipeline，其中有一段的流程是一模一樣的，這時候我們就可以把那一段 Processors 獨立做成 Processor Group，後續若遇到一樣的需求時，就只要拉這個 Processor Group 做串接即可，使用這就不需要再一一建立流程。簡單來說，就是視為『Module』的意思。
- 第二個會用到的情境就是『分專案或部門』為使用，若今天有一個 Team，同時有 10 個專案需要建立 Data Pipeline，理所當然每一個專案的流程都會不一樣，這時候就可以透過 Processor Group 來做專案的劃分；或是有不同 Team 要採用時，也可以利用這個方式來劃分不同 Team。
- 第 3 種是第 2 種的延伸，Processor Group 通常也會是一個 User 權限的最小單位，我們可以針對特定 Processor Group 來決定哪些 User 擁有 Write 或 read 的權限。

其中情境 1，可以從圖 1-4-1 看到它被整合在 Pipeline 的其中一環，但其實我們將其放大來看，它就是由一連串的 Processor 和 Connection 組合而成，此外再設定好 Input/Output 的格式與定義即可。所以未來若有其他 Pipeline 需要類似的情境時，它可以直接拉取這個 Processor Group 來套用，就不需要再重頭拉一次原先 Processor Group 內部的流程。

◆ **Funnel**

用來將多個 source connection 的組合成單一個 connection，這對於可讀性可以提供相當大的幫助，如同上圖的 Processor 5 之前的漏斗符號，我們可以想像假設有 n 個 Processor 同時要連到同一個 Processor，如果不透過 Funnel 的話，在下游的那一個 Processor 身上會有很多條線，這對於使用者在檢視或是 Debug 是不理想的。

碁峯 www.gotop.com.tw

接下來介紹的 Component，就不會呈現在上圖，但在 Apache NiFi 的操作上有一定的重要性與角色：

◆ **Controller Service**

Controller Service 可以把它想像成是一個與第三方對接的 connection，注意不是前面所提到的 connection，而是真的透過網路服務建立的 connection。所以當我們選擇 NiFi 作為我們 Data Pipeline 的工具時，照理來說就會在服務上建立許多的 Pipeline，甚至有些當中的 Processor 會共同存取同一個 DB 或是 cloud 的服務，如果每一個 Processor 都對其建立太多 connection 的話可能會造成問題。

所以此時就可以統一透過 Controller Service 來做一個管理，它可以事先建立好一個與第三方服務的 connection，而當有 Processor 需要做使用的就可以直接套用對應的 Controller Service，一方面不用再重新設定，另一方面可以對目標存取控制好連線數以節省開銷。

◆ **Reporting Task**

Reporting Task 是 NiFi 在做 Monitoring 很重要的角色，它可以將一些 Metrics、Memory & Disk Utilization、以及一些 Monitoring 的資訊發送出來，常見應用是發送到 Prometheus、DataDog、Cloudwatch 等第三方服務來做視覺化呈現。

◆ **Template**

Templates 通常用於轉換環境做使用，假設我有一個既有的 NiFi 在 A 機器上，在上面已經有設計好一些 Pipeline，但我需要轉換環境到 B 機器上，此時使用者可以把在 A 機器上所有的 Pipeline 輸出成 Templates（為 XML 檔），接著再匯入到 B 機器上的 NiFi，就可以在新的環境中繼續使用相同的 Pipeline 了。

這當中的轉換是以 Processor Group 作為單位，其中也會把這個底下的所需要用到的參數和設定一起匯出成 templates，所以在環境轉換時就會是無痛轉移。簡單的呈現如下圖：

碁峰 www.gotop.com.tw

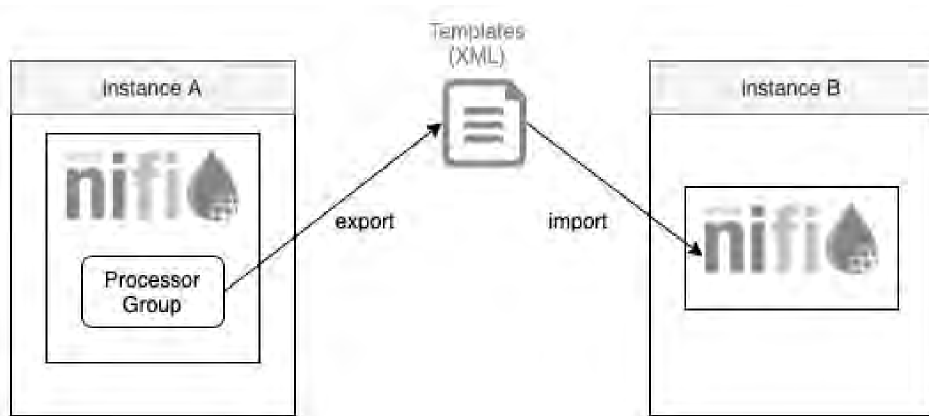

圖 1-4-2　Template 轉換示意圖

1.5　小結

這章節我們介紹了 Data Pipeline 的用途，以及 ETL 和 ELT 在實務上對應的場景說明，此外也帶到 Streaming 和 Batch 的特性，最後提到整體目前在 Data 領域的架構發展。

在建立這些概念完之後，也介紹本書的重點 Apache NiFi 的特性與用途，以及說明了在 Apache NiFi 中重要的元件，因為在後續的章節會開始圍繞這些元件來做更詳細的操作與設定介紹，最後一步一步地帶領讀者們建立出 Data Pipeline。

接下來下一章節，會教如何在自己的電腦或是在機器上建置 Apache NiFi 和 NiFi Registry 的服務，以及分享在實務上建置的注意事項與架構，讓讀者們可以擁有自己的 Apache NiFi 來做後續的操作與應用。

碁峰
www.gotop.com.tw