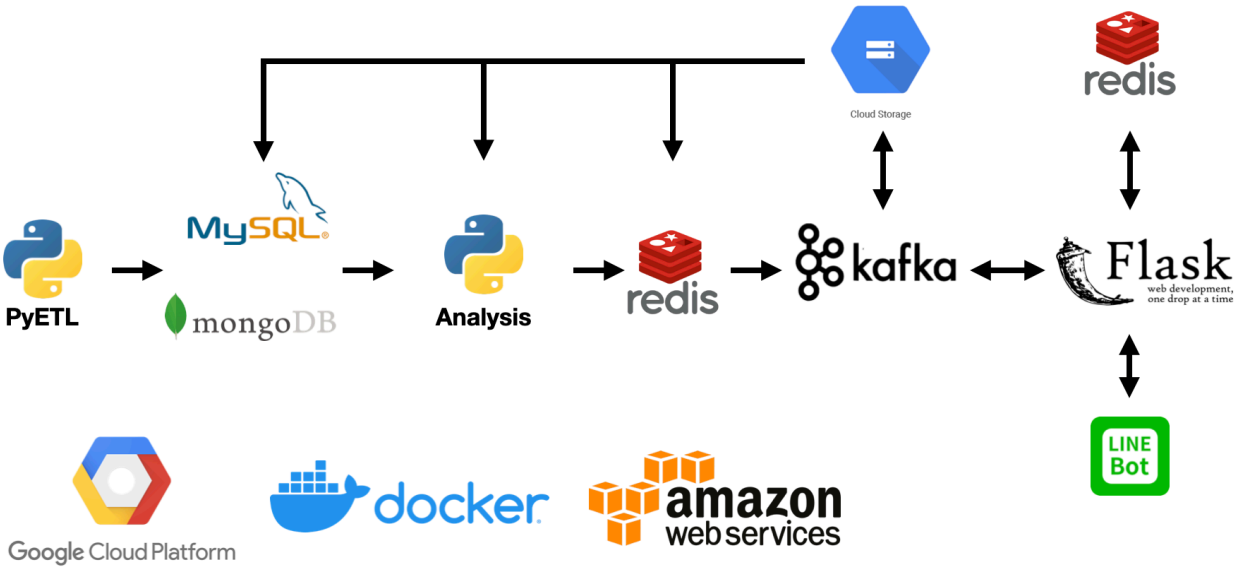
Framework

Summary:

在此次專題中，我們使用了較為複雜的資料串流架構處理，因此這個部分主要是來說明此次我們專題的架構與自動化過程。接下來將以三個部分來說明，分別為前後端溝通平台、本地端資料串接與每日自動化爬蟲。



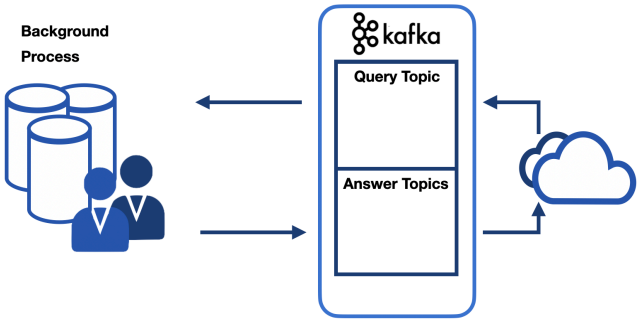
前後端溝通平台—Kafka：

起因：

在這次股市小子的專題中，我們首要面對的問題即是前後端資料串接的困難，原因為資策會這邊本地端的主機並無對外IP，使得前端的外部程式並無法藉由IP來與本地端主機進行資料的交換。

解決過程：

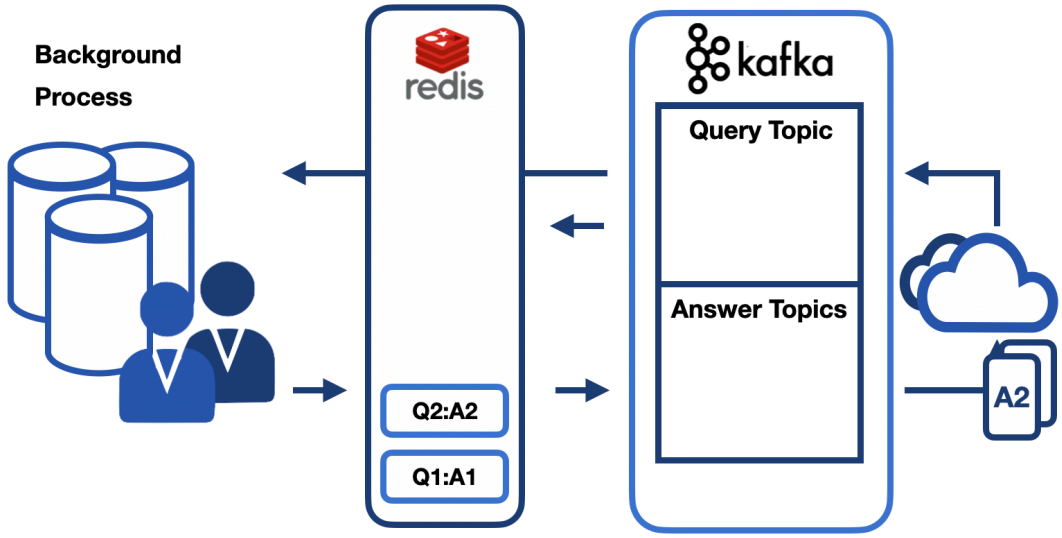
因此我們採取的方法為利用Kafka在資料交換上獨有的被動性，藉由Producer與Consumer的交互使用，讓Kafka的位置像是接線生的概念，以實現資料的交換。但是新的問題也慢慢浮現出來。



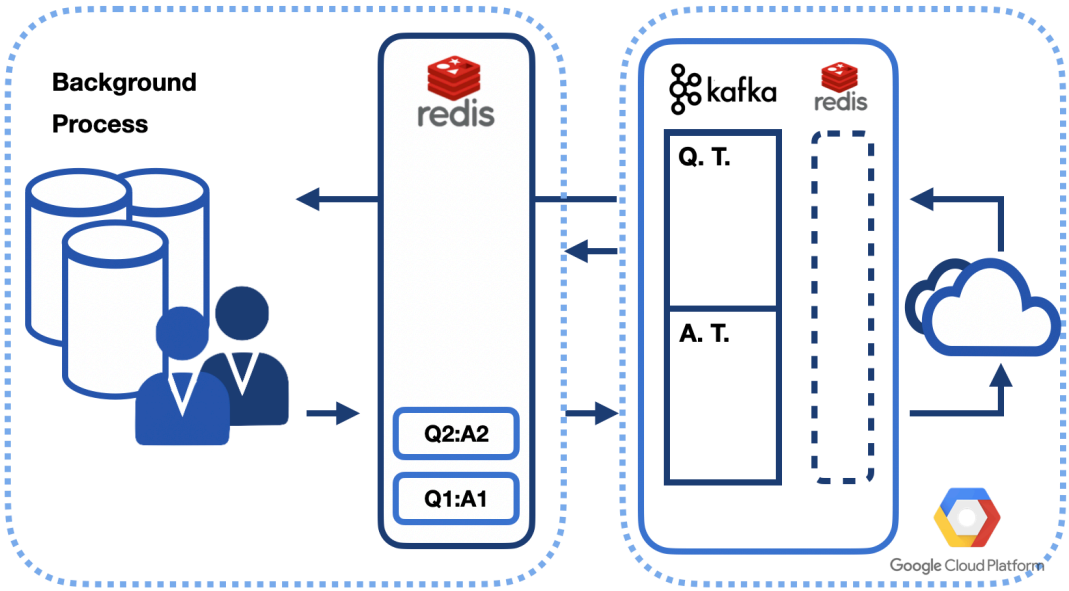
新的問題與解決方法：

由於Line的限制，資料的回應必須落在一定的時間內，不然會被視為是失效的連線，但由於我們在前後端加上了Kafka作為中繼，因此反應時間被大幅加長了。後來我們分別以程式面向與架構面向試圖解決反應時間的問題。

程式面向，我們利用全域變數的宣告，使得在每此的資料串接時不必每次都需要再做一個新的連線；架構面向，我們試圖解決問題的根源—運算，因此我們將使用者預期會問的問題與答案是先先計算完畢，並將其結果存入本地端的Redis以作為快取使用。藉由同時使用上述的兩個方法後，初步的解決了連線的問題。



在解決基本的連線問題後，還想再基於原本的架構上再次的優化系統。我們再次想到在資料庫中擁有強大讀取速度的Redis，並將Redis安排在Kafka同等級的位置，並在裡面儲存近一個月活躍客戶的資料。以利於在我們Line客戶使用時，我們可以以最短的時間為他們做專屬化的股票推薦，大幅度的提升使用者體驗。



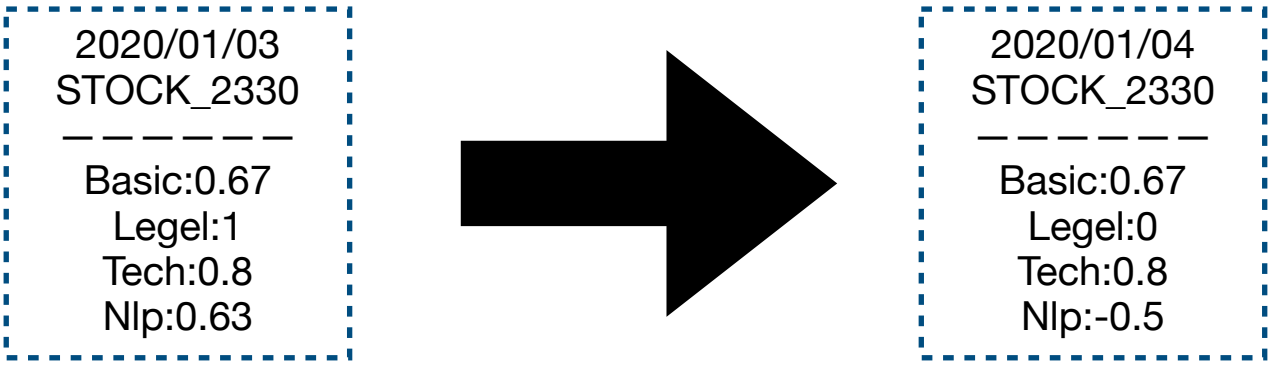
本地端資料串接—Redis：

起因：

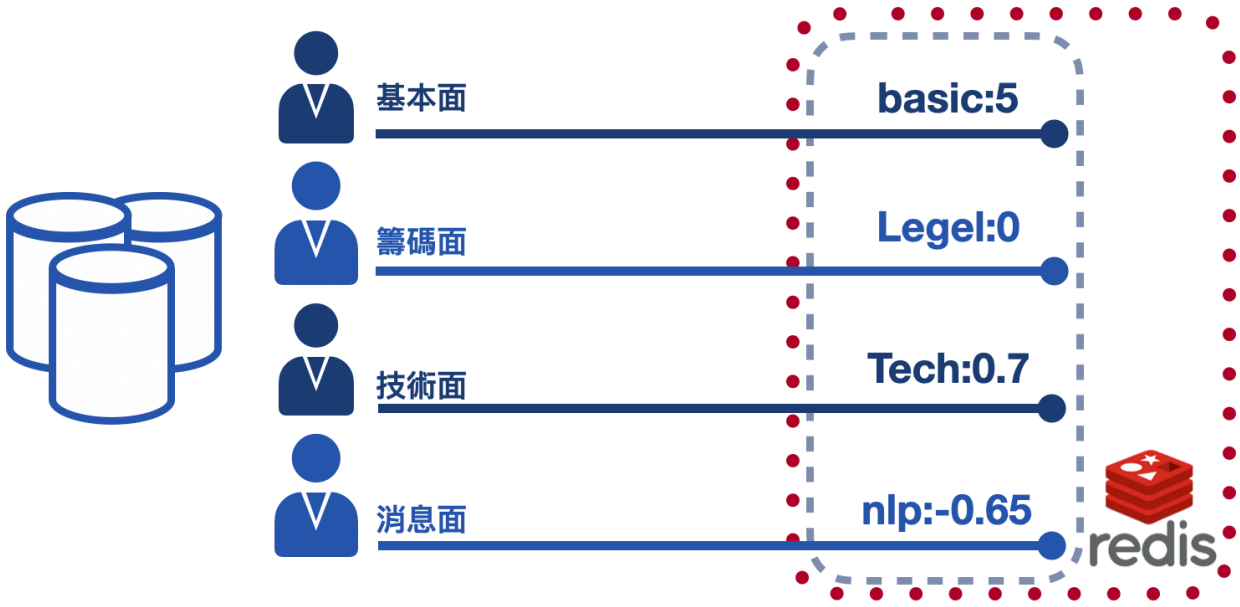
在此次我們的分析中，以分別不同的基本面、技術面、籌碼面與消息面來進行分析，並希望結合不同的客戶屬性，提供出不同的股票推薦清單。

資料整合：

由於此四個方向看似相關，但我們是以分別進行的方式去處理它們，所以這四個面向是彼此獨立不相關的。因此我們藉由Redis中Hash格式的形式對每隻股票的四個面向的處理分數進行集中式管理。也因為Redis的特性，我們不需要再花額外的心思去之前資料的刪除，他會自動迭代。



第二步便是以我們預先準備好對於不同客戶屬性所對應的權重，我們將其整理成向量的格式與每日各股所形成的向量進行Inner Product，並將結果的純量進行排序，再將排序後的前十支股票重新進行Join，以此作為每日股票推薦的依據。



每日自動化爬蟲—Kafka+GCP:

起因：

對於我們股票推薦來說，每日的即時新聞與股價是不可缺少的，因此我們迫切的需要一個強大的爬蟲系統來滿足我們分析資料的需求。

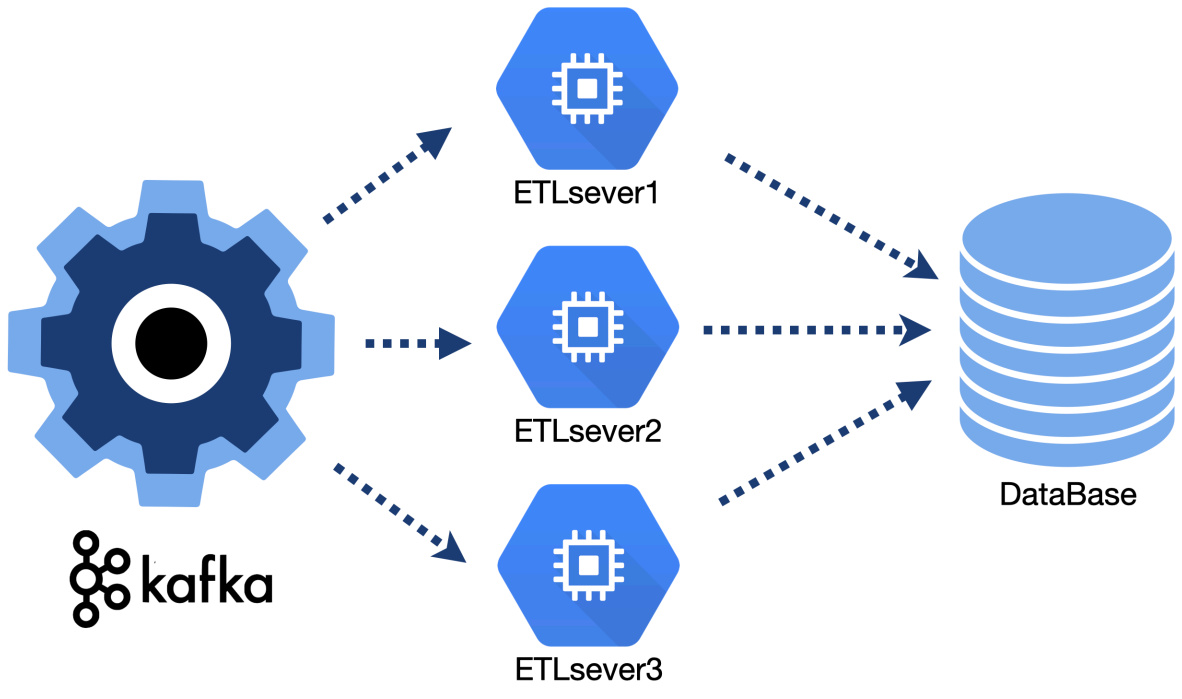
困難點與解決方法：

在爬蟲時往往會遇到諸多困難，其中以IP的問題尤其棘手。再以不花錢的前提下，有些網站所提供的代理IP往往早就已經無法使用，如果還需要花而外的時間對其一一篩選的話，我們認為會本末倒置，所以否決了這個方法。這個問題也因此困擾了我們許久。

後來在接觸雲端GCP與AWS後，這個問題有了一絲希望！我們發現在每台雲端主機會配有浮動的外部IP。如果我們將我們的爬蟲程式部署在不同的雲端VM的話，其實就可以解決代理IP的問題。但是當我們真正將爬蟲程式部署上雲端後，即產生了一個很嚴重的問題。

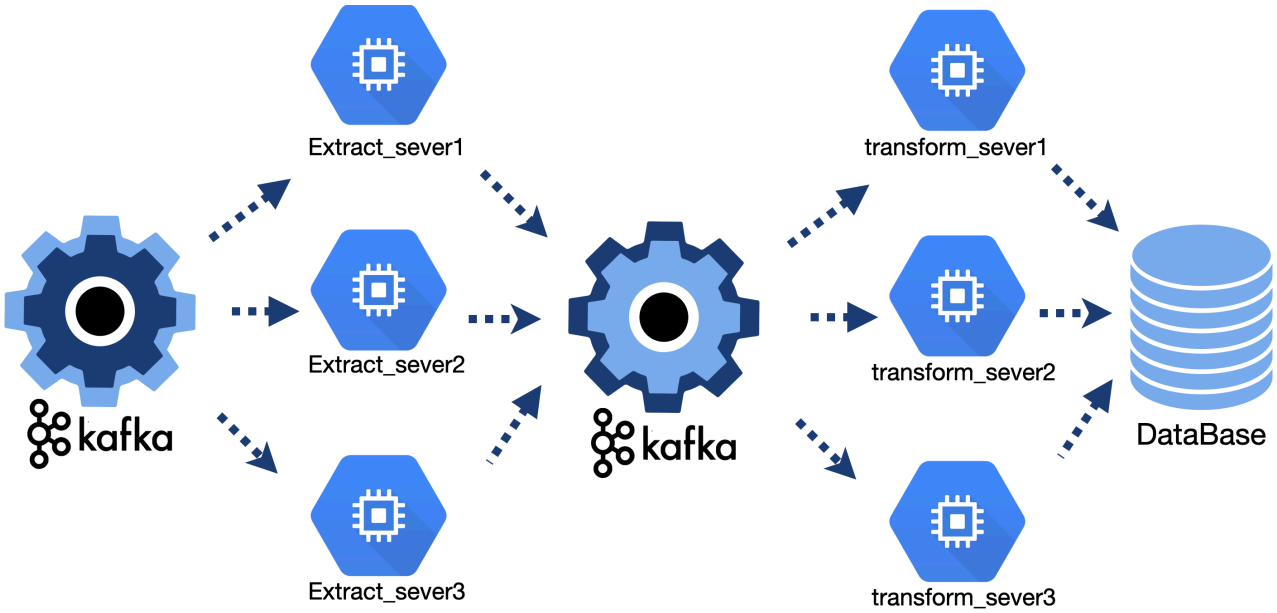
任務分配：

對於相同的爬蟲程式，但因為其部署在不同台機器上，因此他們並無法互相通知彼此爬蟲的進度，可能會造成重複爬取相同的內容。因此我們迫切的要為爬蟲程式們進行任務分配與管理。這時我們認為使用Kafka是一個非常適合的選擇，事先將所有需要爬取的清單以每日排程的模式先行存入Kafka當中，再藉由Kafka中Partition與Consumer的關係，且相同的GroupID會有相同的Commit，因此爬蟲主機間彼此會分配到不同的爬取子清單，已實現分散式的運算。



優化：

但在此架構中依然存在著些許問題。對於整體爬蟲的過程，如果將ETL三個動作皆放在同一台主機進行的話，造成的明顯問題是單位時間內能處理的數據上升幅度有限。因此我們最終使用的架構為入圖下：



此流程重點於不管三七二十一，先將資料自網路刮下來後，就直接再進行下一次的爬蟲，而刮取的內容則是Produce回Kafka中，再由Kafka做第二次資料轉換工作的任務分配，以解決資料吞吐的限制。

主講人：劉宇哲

https://github.com/ben4932042