佇列結構效能大比拼：多場景測試分析與實務應用

RabbitMQ vs Kafka vs GCP Pub/Sub

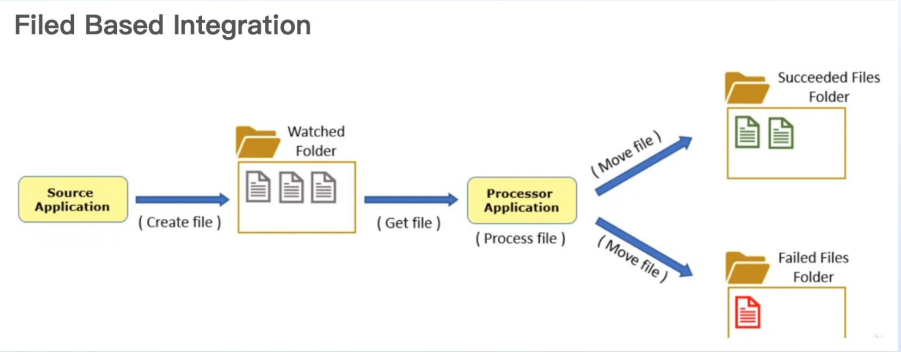
Charlie

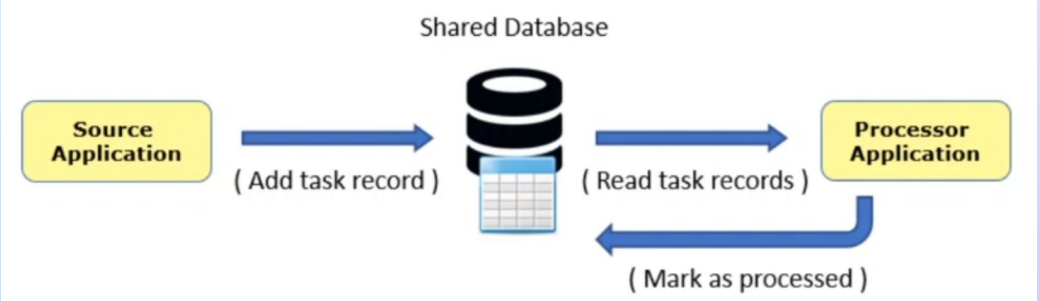
目錄

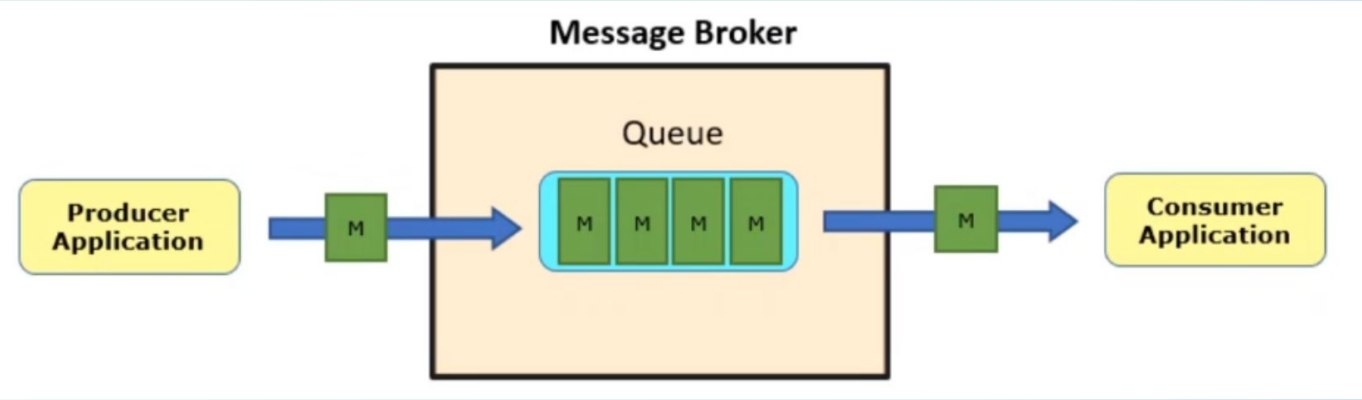
Queue Overview

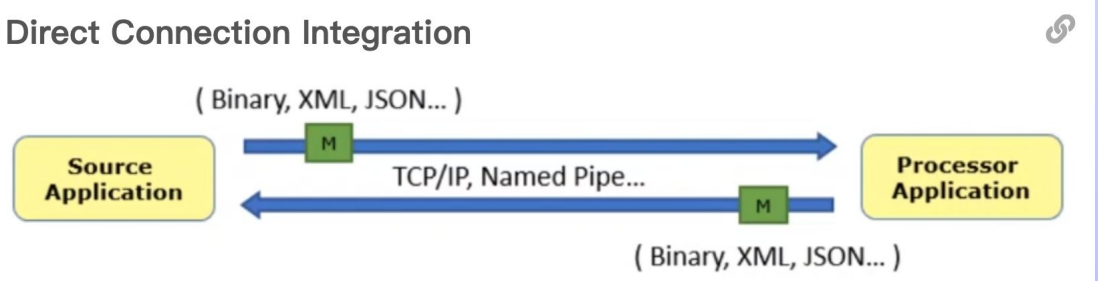
System Intergration類型

- File Based Intergration
- Shared Database Intergration
- Direct Connection Intergration
- Asynchronous Message Broker



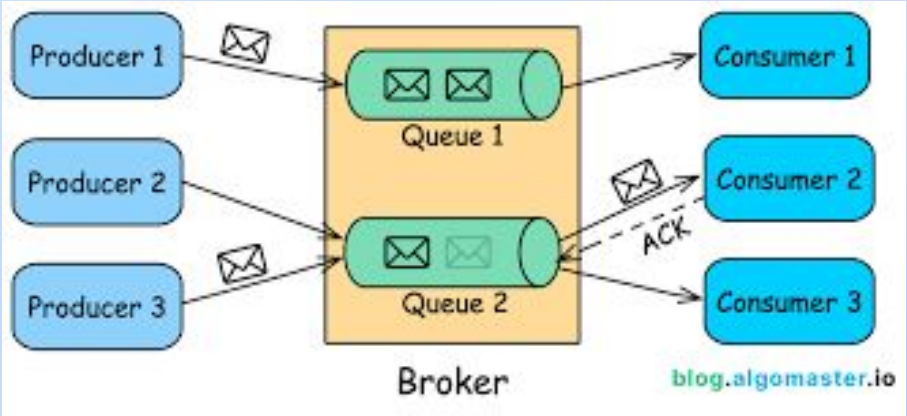






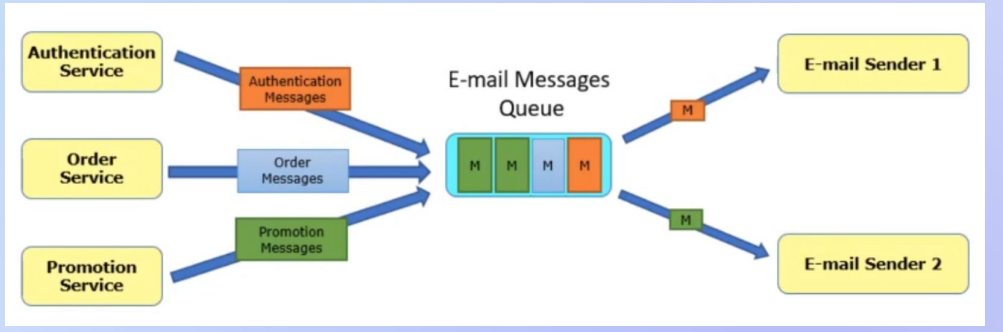
Why Queue?

- First In First Out
- 非同步處理
  - Producers 傳送訊息至 Message Queue 之後不需要立即得到 response 以繼續處理其他事情。
- 鬆耦合
- 系統緩衝
- Better Performance



What is Queue&Why Queue?

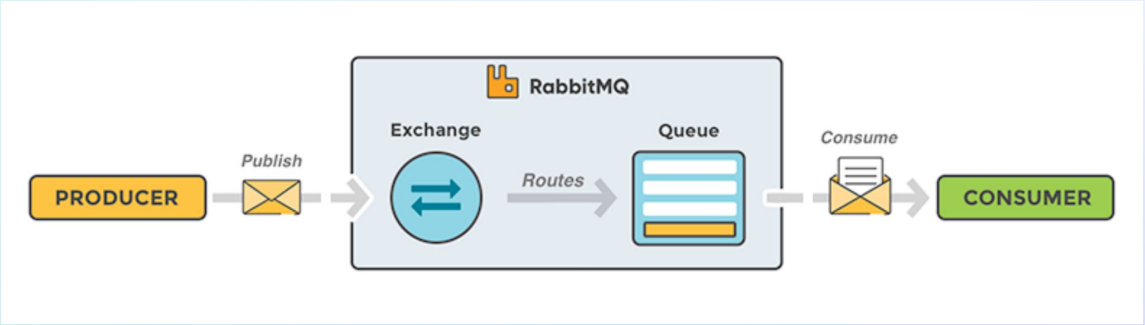
- First In First Out
- 非同步處理 Asynchronous Communications Protocol
  - 不需要立即得到立即回覆
- Better performance
  - consumer 有空時才會處理 message，比起持續等待的方式相對有效率
- 解耦Decouple
  - publisher 與 consumer 不需要知道雙方的實際的位置
- 可靠性Reliablity
  - 預防暫時服務失效
- 可擴展性 Flexiability of scaling
  - producer, queue, consumer 可以依照需求擴張或縮減



RabbitMQ 介紹

RabbitMQ-核心概念

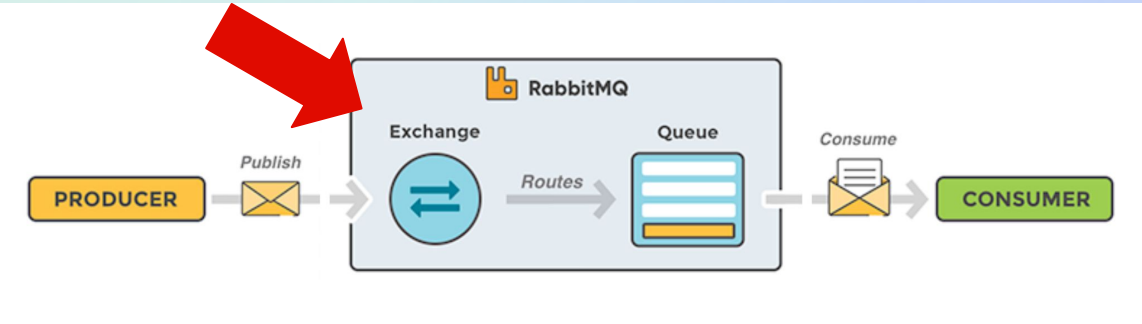
RabbitMQ: 一種 AMQP（Advanced Message Queuing Protocol）實現。



- 隊列（Queue）：
  - 存放消息，供消費者檢索。
- 交換器（Exchange）：
  - 用於將消息根據規則路由到不同的隊列。
- 綁定（Binding）：
  - 連接交換器與隊列，並設定消息路由的規則。
- 消息確認（Acknowledgement）：
  - 確保消息可靠地傳遞並處理，避免丟失。

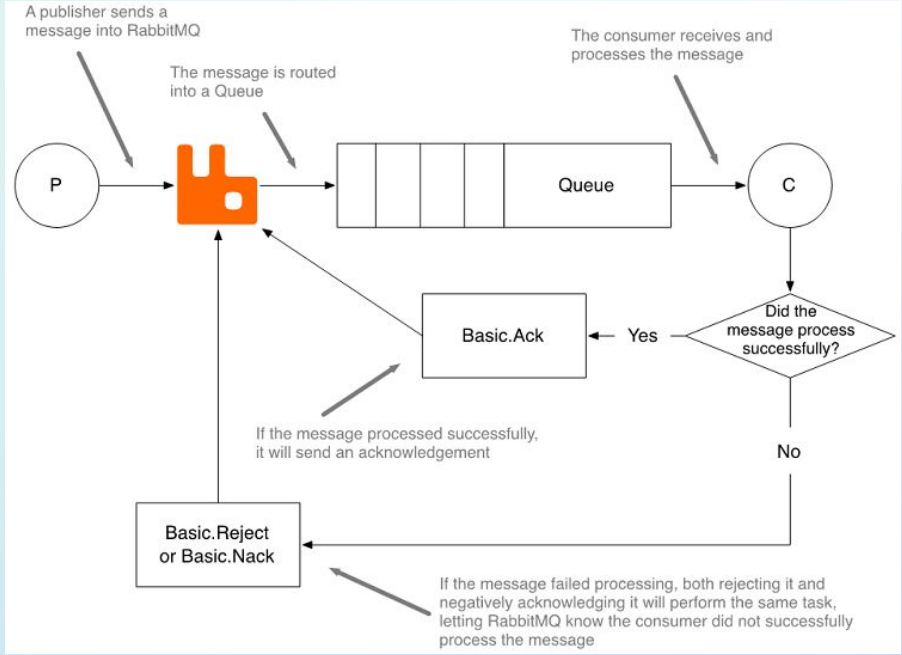
Producer 藉由Binding Routing Key來決定如何丟資料給 queue

- Direct: 直接丟給指定的 Queue
- Topic: 藉由regular expression設定 binding 規則
- Headers: 透過 header 指定 Queue
- Fanout: 忽略Routing Key，將訊息丟給全部負責的 Queue



RabbitMQ-自動確認機制＆資料完整性

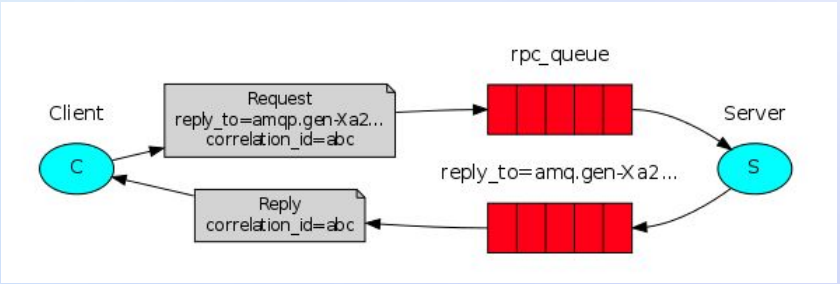
- consumer端：
  - `basic.ack` 用於正面確認
  - `basic.nack`, `basic.reject` 用於負面確認
- acknowledge參數



模式	行為	可靠性	適用場景
acknowledge=none	不管結果直接回傳收到，使消息從queue刪除	低	高吞吐量、非關鍵性數據處理場景
acknowledge=auto	消費者根據框架自動處理	中	開發效率高、可靠性需求一般的場景
acknowledge=manual	消費者手動回傳結果	高	關鍵數據處理、需要嚴格消息保證

RabbitMQ-RPC(Remote Procedure Call)

- **請求者（Client）：**
  - 發送請求消息到指定的請求隊列。
  - 為每個請求生成唯一的 correlation_id，用於追蹤請求和回應。
  - 指定一個回應隊列（ reply_to）用來接收服務端的回應。

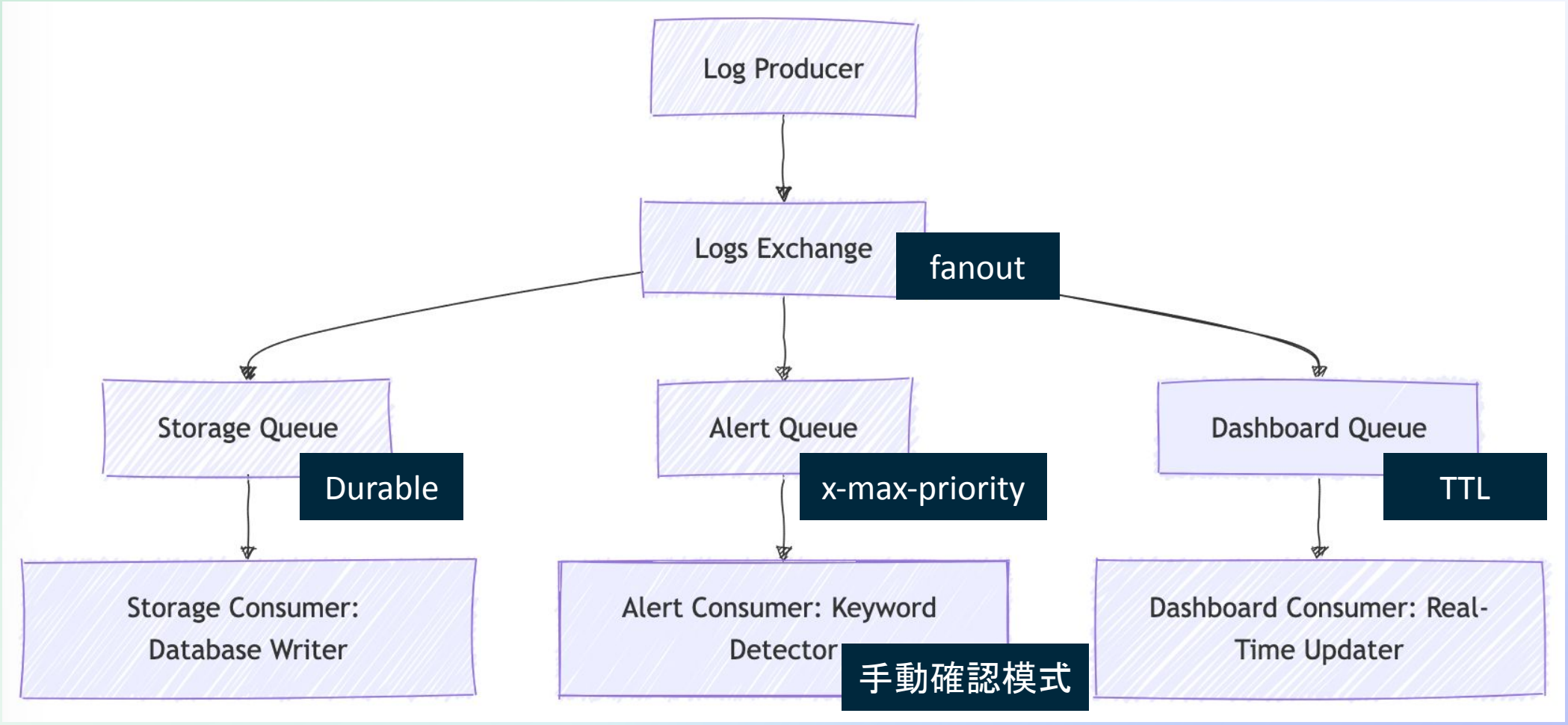


- **處理者（Server）：**
  - 從請求隊列中取出消息，執行相應的處理邏輯。
  - 將處理結果發送到請求消息中指定的回應隊列（ reply_to），並保持 correlation_id 不變。
- **回應處理：**
  - 客戶端在回應隊列中接收消息，通過 correlation_id 對應到原始請求並完成處理。

RabbitMQ-關鍵字

- 基本隊列設定：
  - 持久化（Durable）
  - 獨占（Exclusive）
  - 自動刪除（Auto-delete）
- 消息處理相關設定：
  - 消息 TTL（Time-To-Live）
  - 隊列最大長度 /容量（Max Length）
  - 死信交換器（Dead-Letter Exchange, DLX）
- 消費行為設定：
  - 確認模式（Acknowledgment Mode）
  - 消息公平分發（Prefetch Count）
- Queue類型：
  - 鏡像隊列（Mirrored Queue）
  - 分片隊列（Sharded Queue）
  - 延遲隊列（Delayed Queue）
- Queue參數：
  - 優先級（Priority Queue）
  - 隊列過期時間（Queue TTL）
  - 備份隊列（Alternate Exchange）

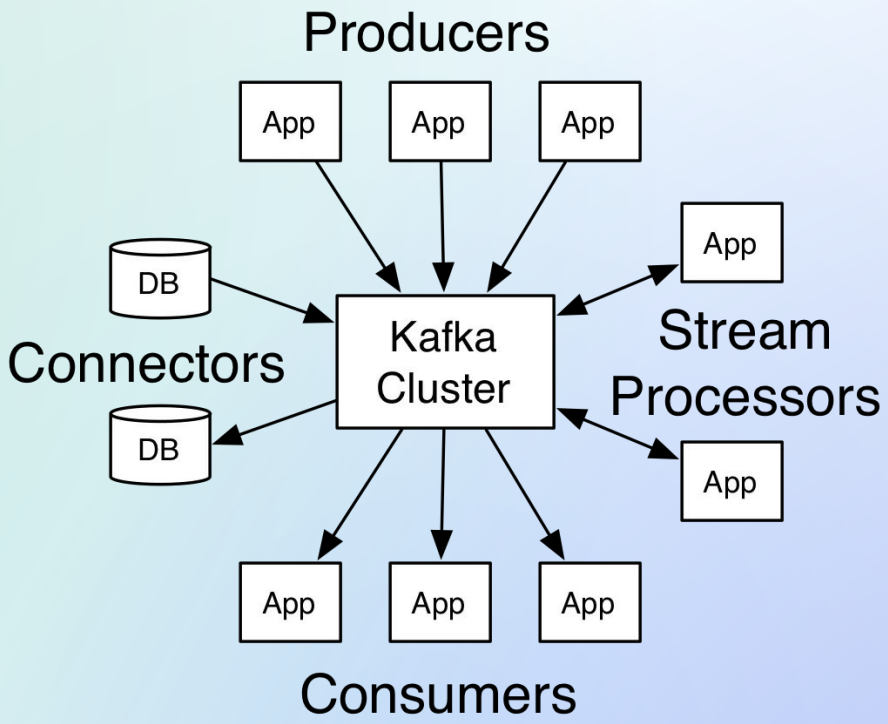
RabbitMQ-案例-日誌分析告警系統



Kafka 介紹

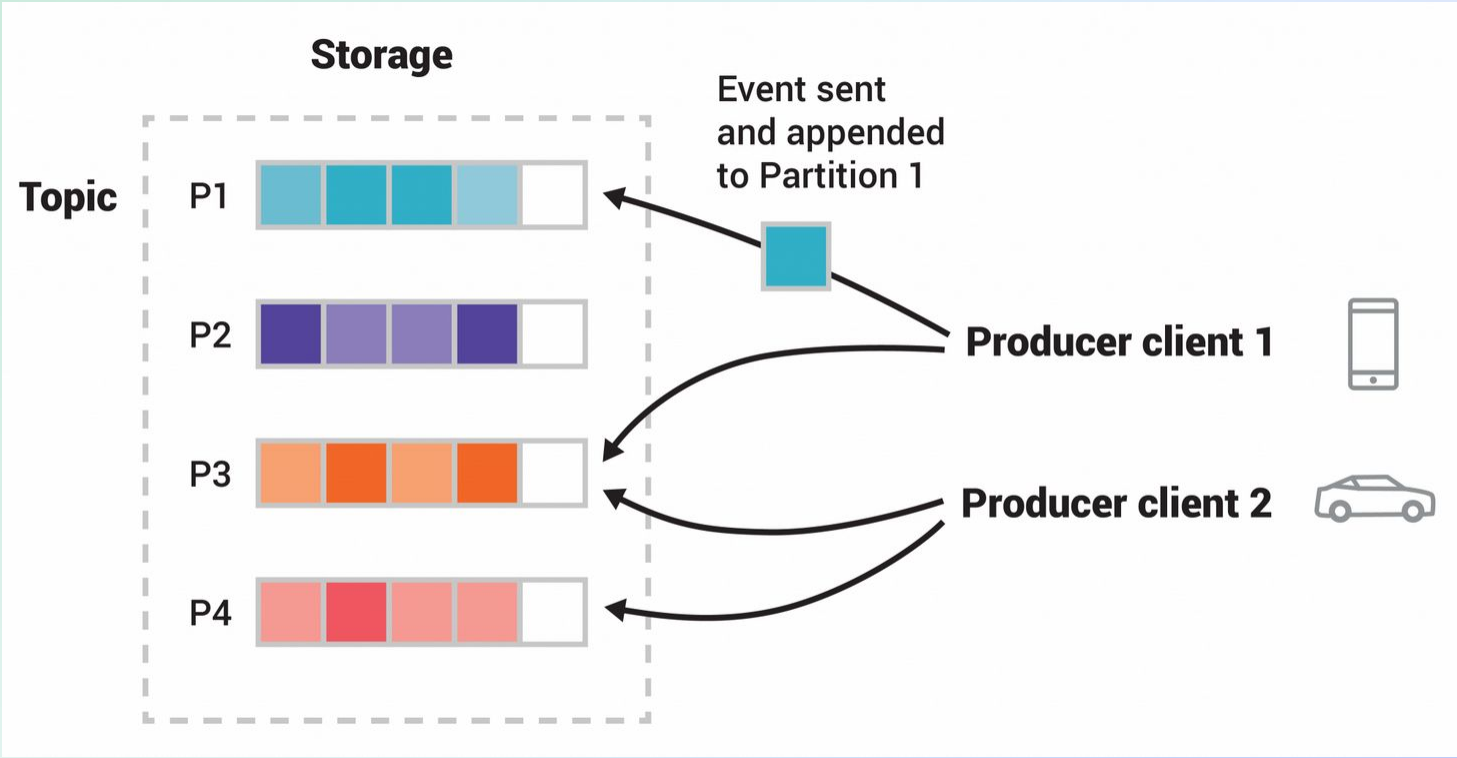
Kafka-核心概念

- Producer&Consumer
- Broker&Cluster
- Topic&Partition
- Event&Message
- 分散式系統
- 事件串流處理



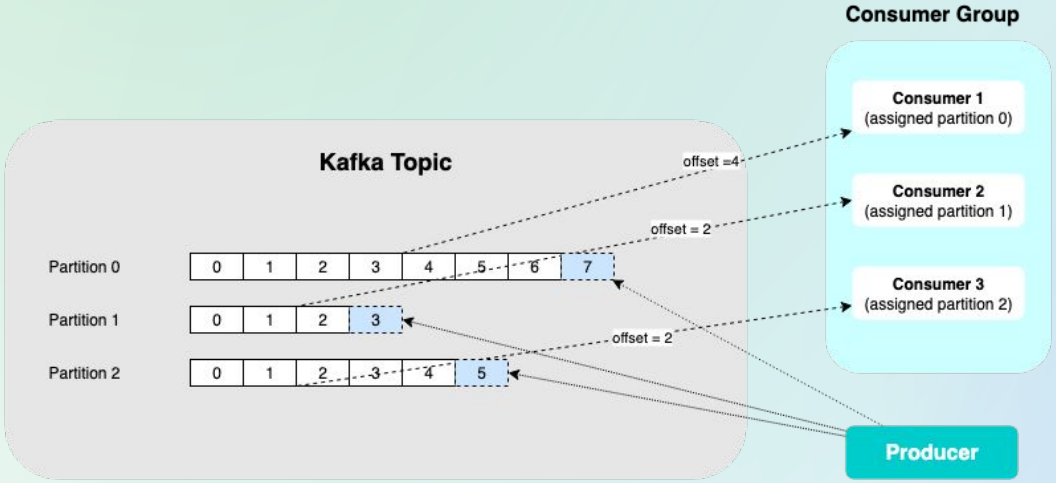
Kafka-topic&partition

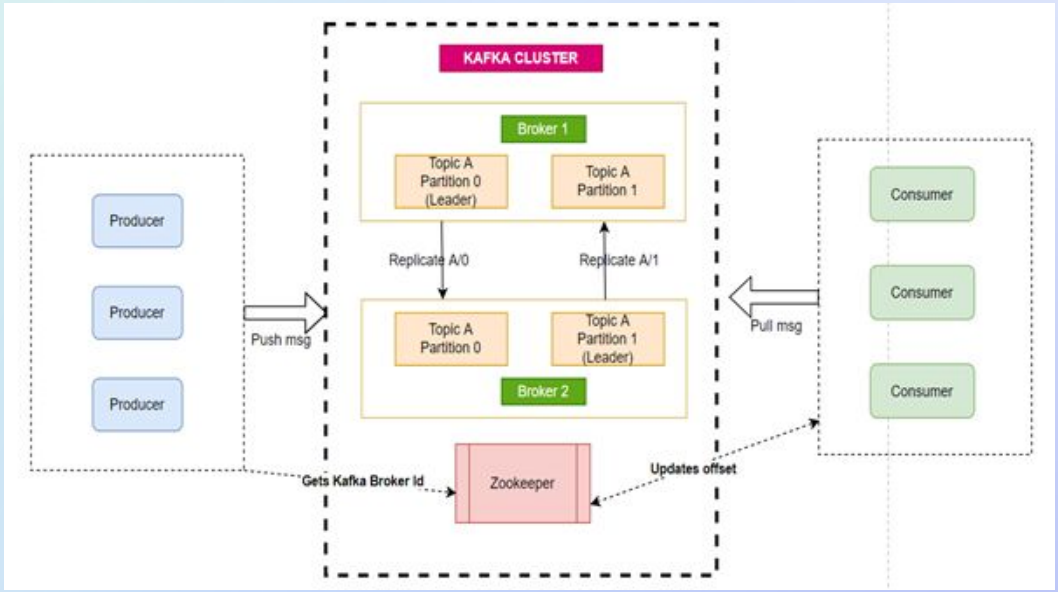
- Event: 用key-value-timestamp組成事件
- 基於磁碟保存資料



Kafka-offset

- 每條消息在 partition中都有一個唯一的、順序遞增的 offset數字
- Consumer用offset來追踪自己讀取到哪個位置
- offset提交 : 自動 /手動提交
- 儲存在 Kafka的topic/Zookeeper中

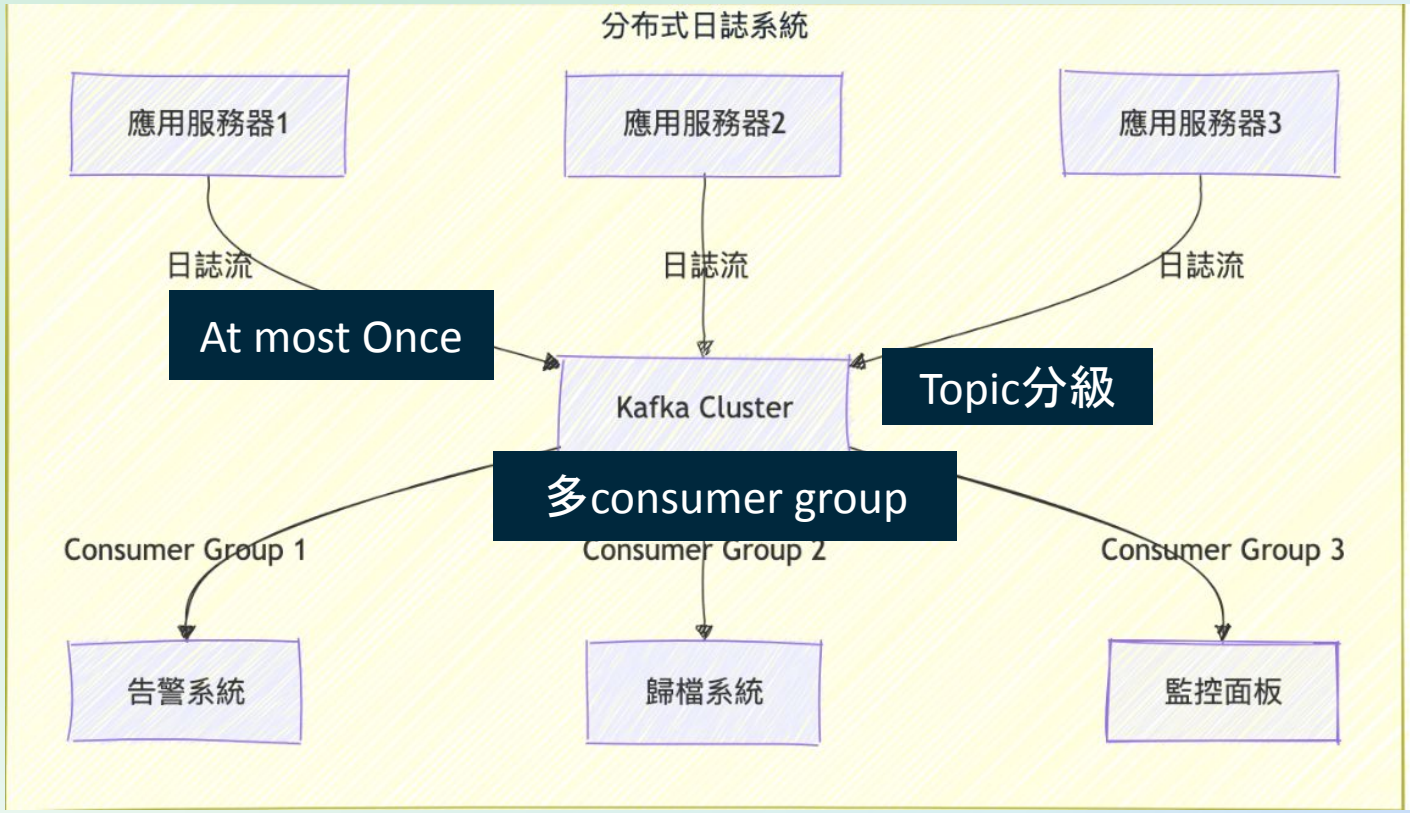


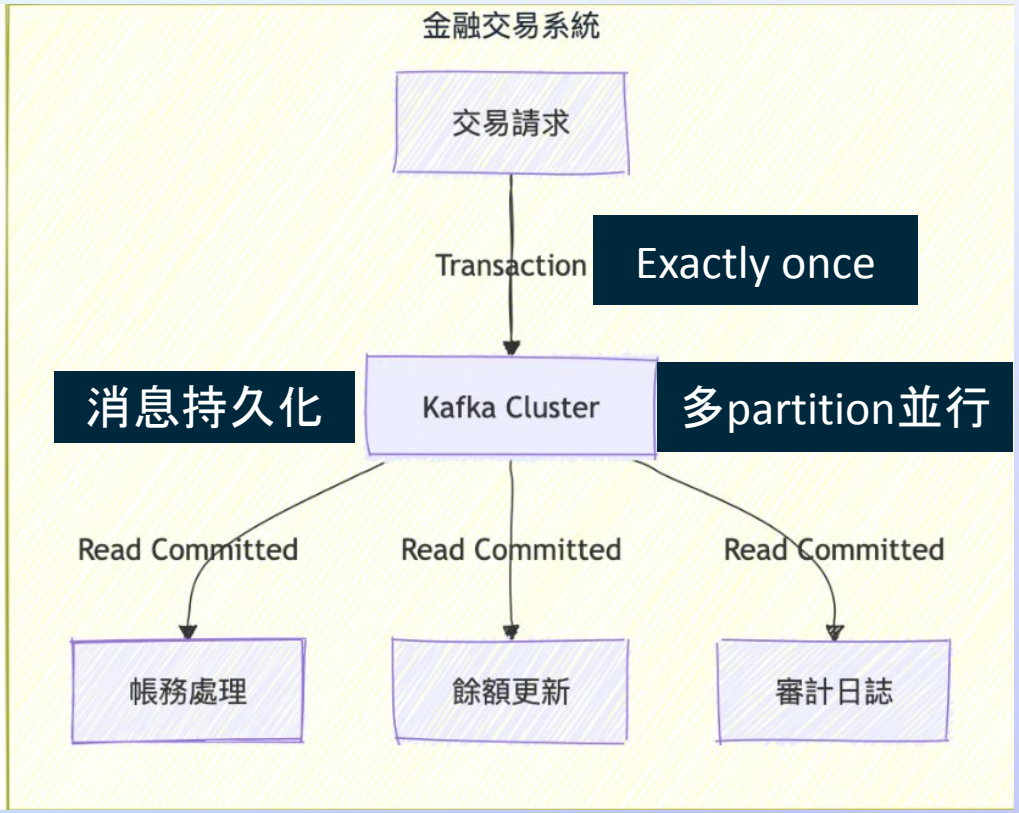


Kafka-message確認機制

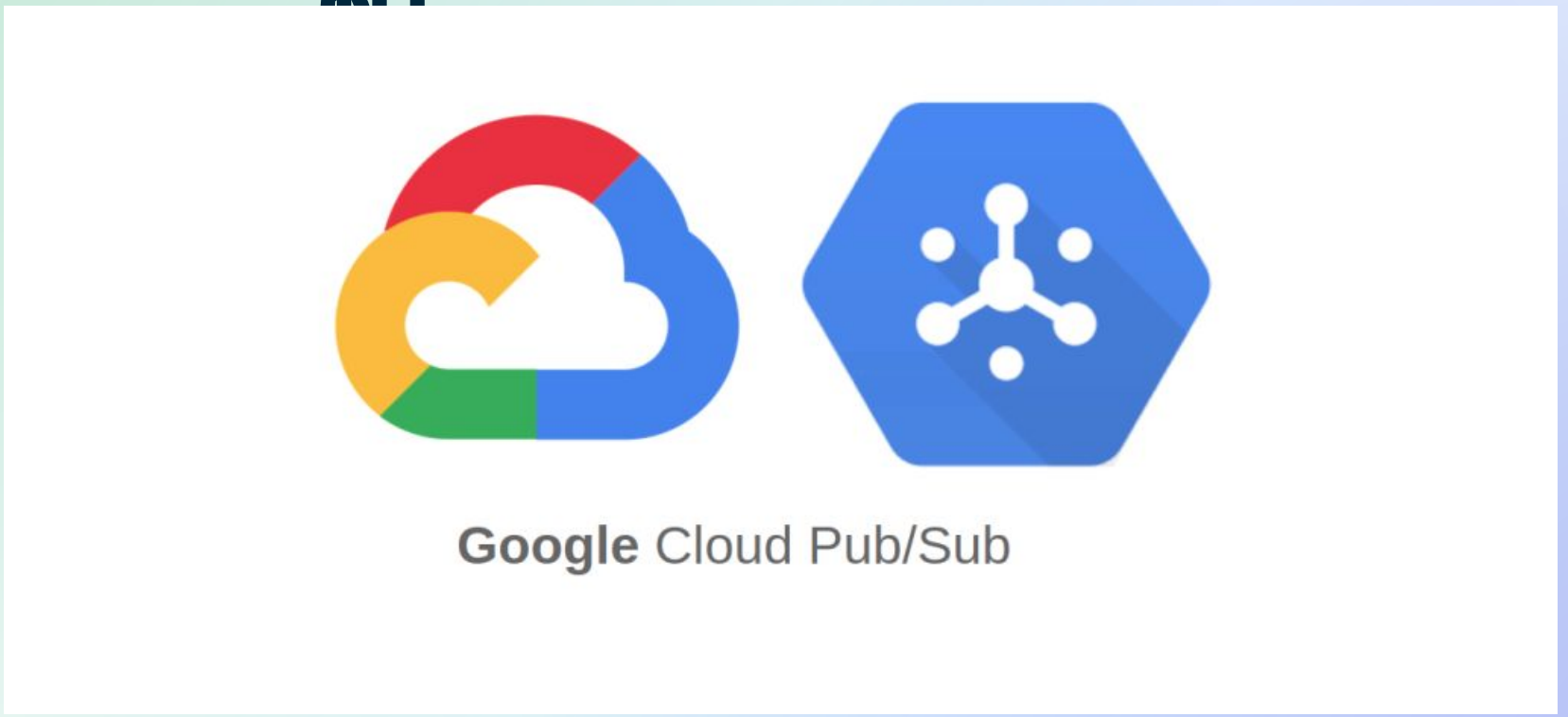
- At most once (至多一次)
  - 消息可能會丟失，但 絕不會重複傳遞
  - 適用場景：可以容忍數據丟失的場景，如日誌收集
- At least once (最少一次)
  - Kafka預設的傳遞語意，確保消息至少會被傳遞一次，但可能重複
  - 適用場景：不能容忍數據丟失，且能處理重複的場景
- Exactly once (剛好一次)
  - 消息只會被傳遞一次，不丟失也不重複，透過 (Transactions)實現
  - 適用場景：金融交易、計費系統等對準確性要求高的場景

Kafka-案例


分布式日誌系統
應用服務器1
應用服務器2
應用服務器3
日誌流
日誌流
日誌流
At most Once
Kafka Cluster
Topic分級
多consumer group
Consumer Group 1
Consumer Group 2
Consumer Group 3
告警系統
歸檔系統
監控面板


金融交易系統
交易請求
Transaction
Exactly once
消息持久化
Kafka Cluster
多partition並行
Read Committed
Read Committed
Read Committed
帳務處理
餘額更新
審計日誌

GCP Pub/Sub 介紹

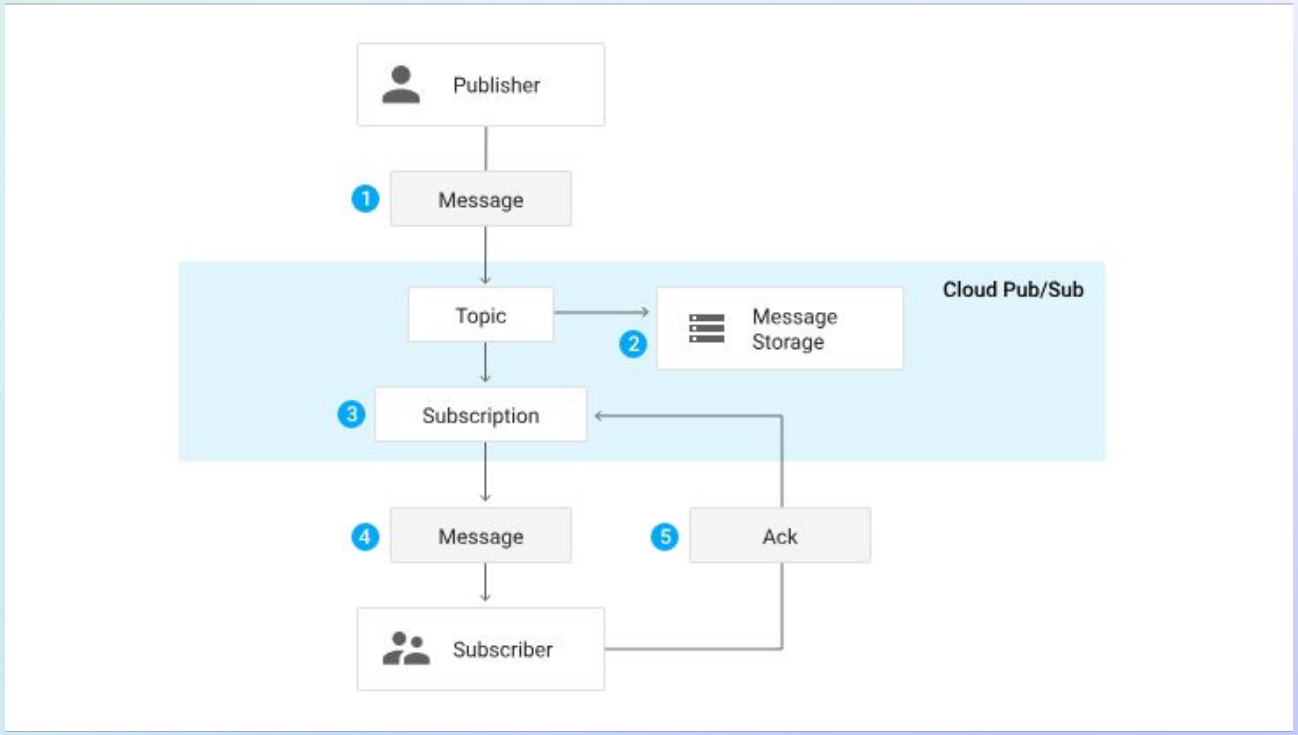


GCP-Pub/Sub

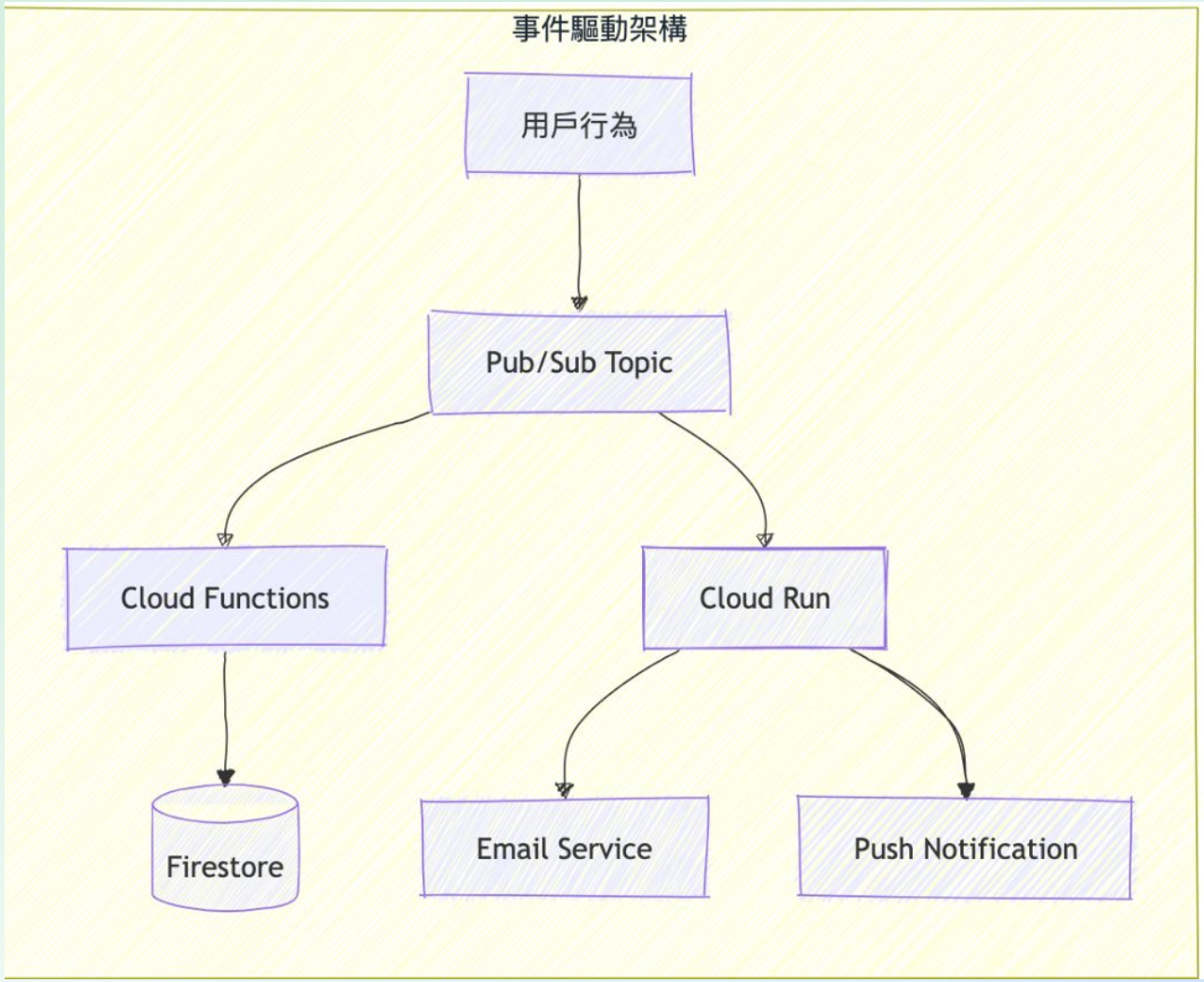
- Publisher（發布者）
- Subscription（訂閱）：為 Subscriber 的上游，負責將訊息傳給指定的 Subscriber。
- Subscriber（訂閱者）：訂閱特定主題的服務 /人。
- Topic（主題）：將 Publisher 的訊息儲存至 Google Cloud Storage
- Partition
- 失敗重試機制（ Acknowledge）
- Retry（重試）

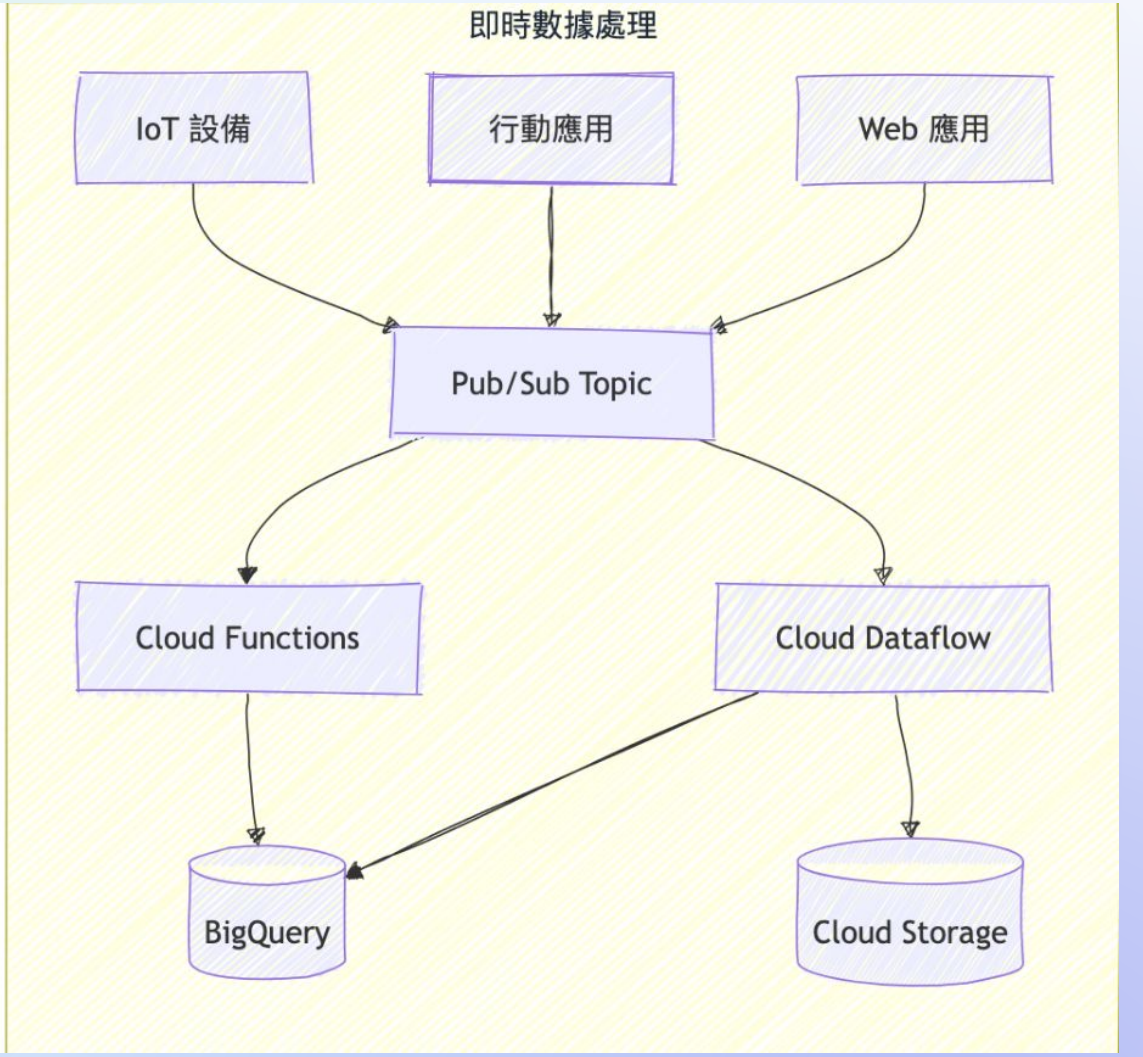
GCP-Pub/Sub-流程

- 預設語意: At-least once
- 注意事項
  - 消息不保證順序
  - Acknowledge等待問題
- 無需設定節點，自行依據數量拓展



GCP-Pub/Sub-案例

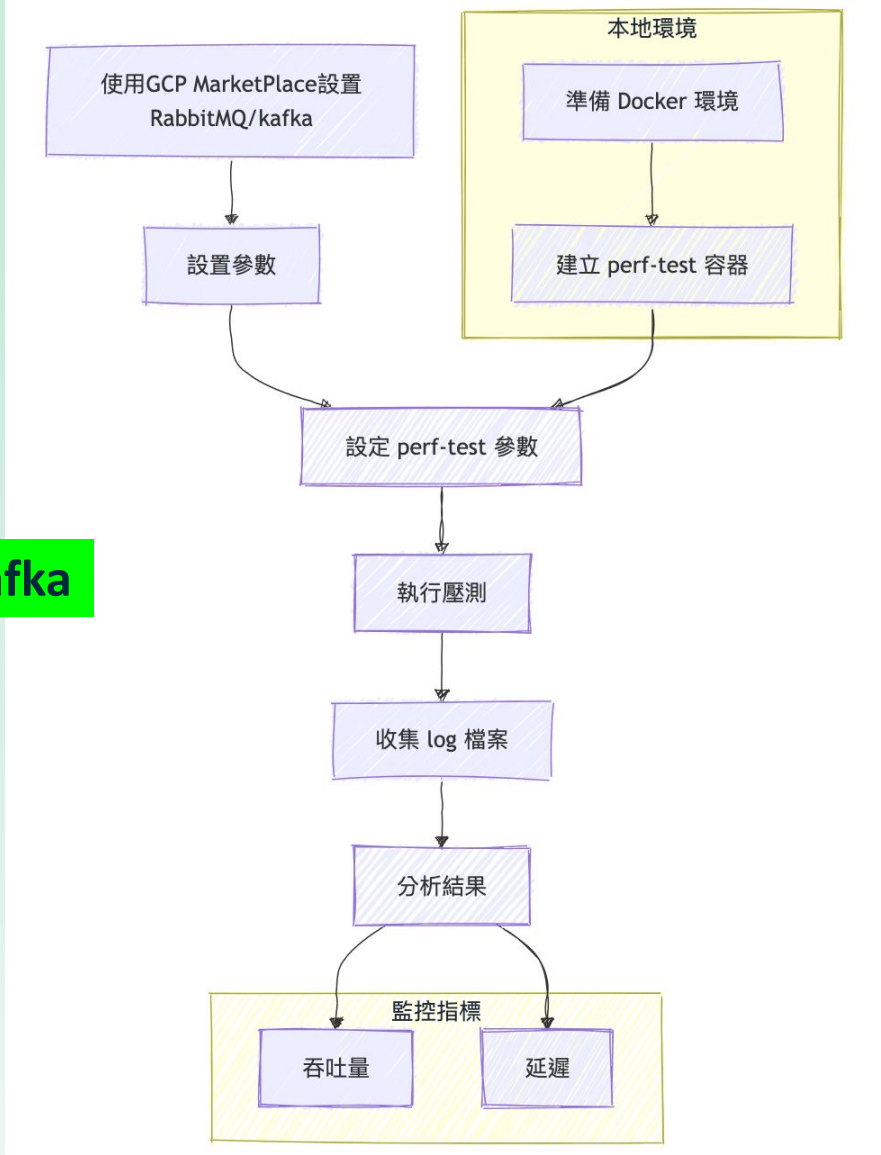




Message Queue
效能壓測實驗

實驗流程

RabbitMQ/Kafka



GCP PubSub

實驗結果

	Rabbitmq	Kafka	GCP PubSub
運行地點	GCP compute Engine(GCP Marketplace)		Google PubSub
壓測工具	RabbitMQ PerfTest	Kafka PerfTest	自行撰寫Python腳本
測試種類	持續中等流量		
處理速度	Produce:436msg/s -> 140msg/s consume:穩定120msg/s	262msg/s -> 4518msg/s	Produce:80-140msg/s consume:10-140msg/s
延遲	128ms -> 2204ms	1186ms	168ms -> 3642ms
效能觀測	consumer速度跟不上producer 消息堆積導致高延遲	速度穩定上升	生產/消費圖形類似
note	低流量情況反而勝過Kafka	消費者無寫入操作所以很快	速度緩慢上升

實驗結果

	Rabbitmq		Kafka
運行地點	GCP compute Engine(GCP Marketplace)		
壓測工具	RabbitMQ PerfTest		Kafka PerfTest
測試種類	高Client同時併發	突發高流量	
處理速度	Produce:7,846 msg/s -> 15msg/s consume:136-290msg/s	Produce:414msg/s -> 140msg/s consume:120-140msg/s	Produce:526msg/s -> 5824msg/s
延遲	381ms -> 13,878ms	15ms -> 2046ms	1827ms
效能觀測	消息堆積情況下延遲飆升	表現與持續流量相近	高流量下表現穩定

總結

綜合比較

	Rabbitmq	Kafka	GCP PubSub
消息模型	AMQP	事件驅動串流佇列	完全託管發布/訂閱
吞吐量	中	極高	中
延遲	低流量:低 高流量:高	中	中
可擴展性	有限	高	高
管理難度	中	高(cluster, zookeeper…etc)	方便
顆粒度	高	高	低
常見案例	可靠性要求高的小規模應用: 即時通信	高吞吐量的數據流管道: 日誌處理	GCP 整合:雲原生微服務
預設語意	至多一次	最少一次	最少一次

Thank you!

Q&A Time