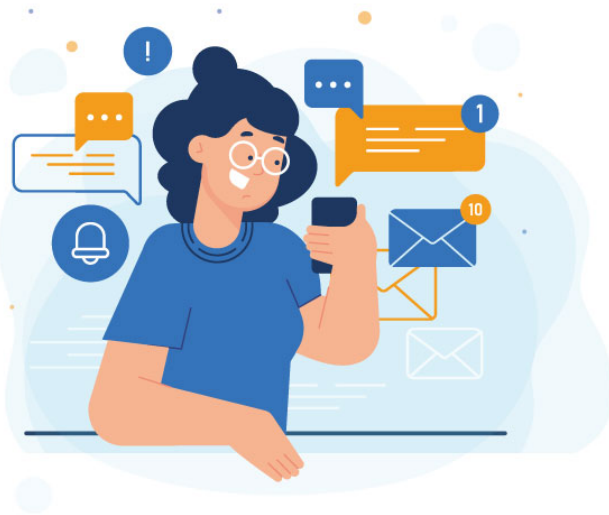


GoogleCloud

- [Pub/Sub 是什麼？6 大常見名詞與訊息生命週期介紹](#)
- [【CCP】Google Cloud Platform 服務總覽](#)

Pub/Sub 是什麼？6 大常見名詞與訊息生命週期介紹

Cloud Ace



Pub/Sub 是甚麼？

6 大常見名詞與訊息生命週期介紹

文章段落



- [Pub/Sub 是什麼？](#)
 - [概述與優勢介紹](#)
 - [6 大常見名詞](#)
 - [訊息生命週期](#)
- [ack / Retry 說明](#)
 - [ack](#)
 - [Retry](#)
- [常見問題](#)
- [結論](#)

Google Cloud 的 Pub/Sub 可為不同的應用程式之間提供收發訊息的功能，以及每則訊息的接收狀態追蹤功能，確保任何規模的訊息皆可穩定地傳送，有效提升用戶服務的靈活與穩健性，這次就讓我們從零開始，完整解析 Pub/Sub 功能優勢、6 大常見名詞與訊息生命週期吧！

Pub/Sub 是什麼？

概述與優勢介紹

Pub/Sub 是一種全球性訊息傳遞的服務，並具有高可用架構，可以在應用程式與服務間交換訊息。由 Publish 與 Subscribe 組成，運用了發佈及訂閱的概念讓訊息透過非同步的方式進行傳輸，只要您的服務可以處理訊息，Pub/Sub 就會持續傳送訊息，如果您 Subscription 下的所有 Subscriber 均無法處理訊息，Pub/Sub 會自動將您的訊息在默認的情況下保留 7 天（該天數最高可以延長至 31 天）。

另外在 Pub/Sub 中提供 3 種類型分別為：**多對一**、**多對多**、**一對多**，這 3 種類型可以讓使用者更靈活的處理訊息，讓服務可以高效運作。

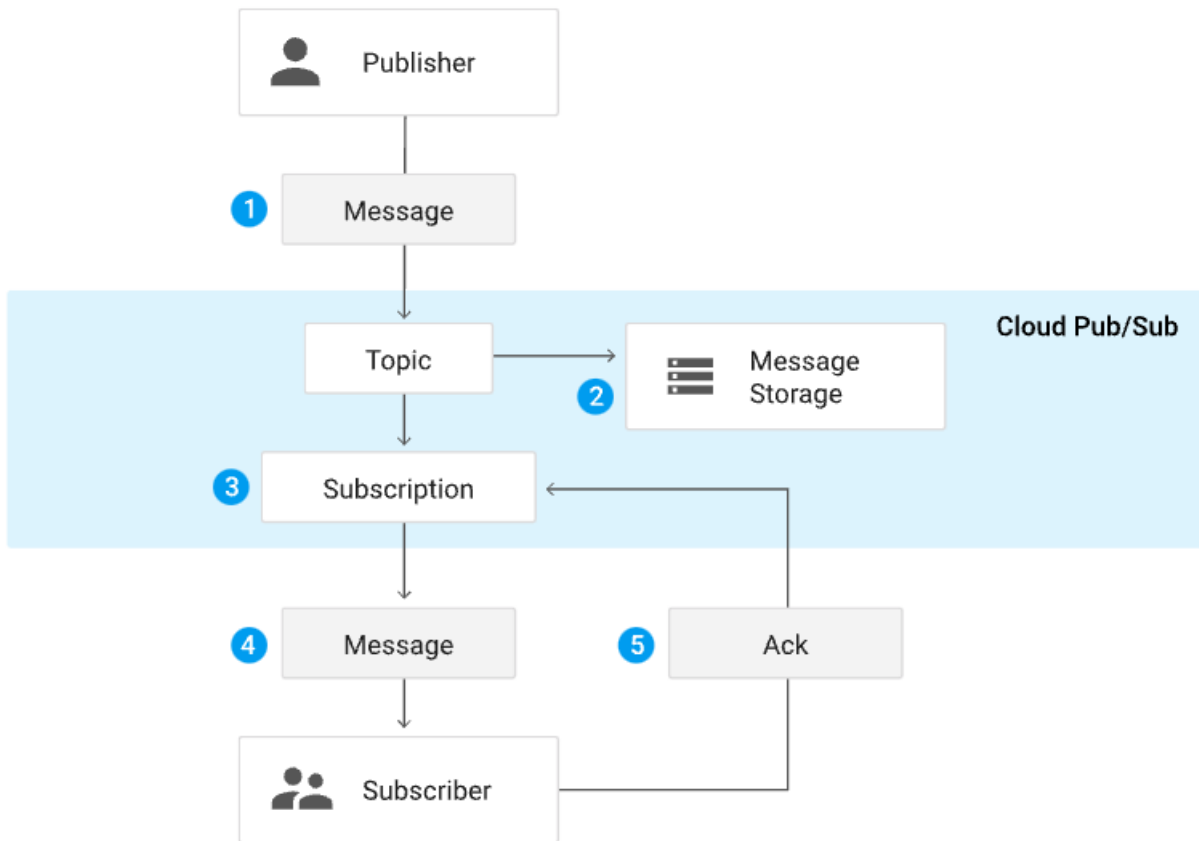
6 大常見名詞

對 Pub/Sub 服務有初步了解之後就要針對 Pub/Sub 的服務進行更深入的探討，首先我們先來了解 Pub/Sub 的六個常見名詞及生命週期：

- Publisher（發布者）：創建訊息並將訊息傳送到指定主題上的服務/人。
- Subscription（訂閱）：為 Subscriber 的上游，主要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）：處理 Publisher 傳送的訊息並將其儲存至 Google Cloud Storage（簡稱 GCS）。
- Acknowledged（已確認）：簡稱「ack」主要用於確認訊息是否有被接收最少一次。
- Retry（重試）：若訊息處理失敗會進行 Retry 用來確保所有訊息都可以被處理的機制。

訊息生命週期

訊息生命週期流程（1-5）說明：（1）將訊息傳到 Pub / Sub，（2）將訊息寫入儲存空間，（3）Pub / Sub 將訊息發布給該主題的所有訂閱，（4）訂閱將訊息發送到訂閱者，（5）訂閱者回覆「ack」確認已經收到此訊息，並將此訊息從 GCS 上刪除；至此生命週期結束。



截圖自：[Cloud Pub/Sub 產品頁](#)

©2023 Google

可以發現特別是在流程（5）需要回覆「ack」給 Pub / Sub，這是因為 Pub / Sub 的基礎架構為「保證訂閱」的方式設計，確保所有訊息可以被傳送一次，因此為了確保所有資料完整被 Subscriber 接收到，會特別要求 Subscriber 進行「ack」，若沒有即時進行「ack」您的訊息則會被丟入「Retry」中。

ack / Retry 說明

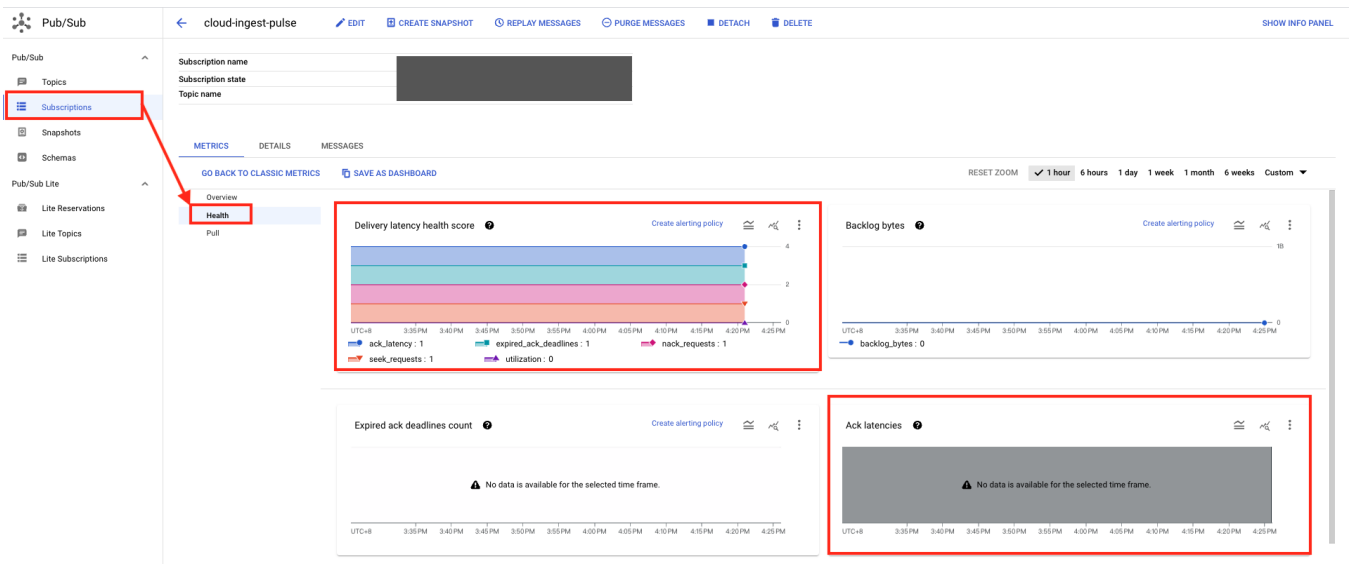
ack

若訊息未 ack 有何影響

若訊息未 ack 在 Pub / Sub 的設計上就會進行 Retry requests 的操作，這個動作會將未在時間內「ack」的訊息重新排序，這可能會導致您的 Subscriber 無法即時找到相關的訊息，進而導致該訊息延遲的狀況發生。

如何確認「未 ack」的訊息

可以透過「[subscription/delivery_latency_health_score](#)」、「[subscription/ack_latencies](#)」這兩個 Metric 來確認，其中「[subscription/delivery_latency_health_score](#)」若有數值為「0」代表該 Topic 過去 10 分鐘內可能有發生未正常處理的情況，比如：太晚進行「ack」、過去 10 分鐘內有 0.1% 的確認延遲時間超過 30 秒 ... 等；另外也可以從下圖右下角的「[subscription/ack_latencies](#)」看出延遲的時間。



截圖自：[Cloud Pub/Sub 產品頁](#)
© 2023 Google

Retry

甚麼原因會出現 Retry？

Retry 是因為 Subscriber 未在時間內進行「ack」，這些未被「ack」的訊息會被 Pub / Sub 視為 Subscriber 無法處理訊息，且基於 Pub / Sub 為保證訂閱的服務，因此會將這類的訊息進行 Retry；若所有 Subscriber 均無法處理訊息，Pub/Sub 會將訊息保存至 GCS 中。

3 方法避免訊息被 Retry

如果服務處理速度跟不上 Pub / Sub 傳送的速度您可以透過以下方式進行修改：

- 提升服務的機器規格，讓您的服務可以即時處理 Pub / Sub 傳送的訊息。
- 可以透過延長「ack」時間來延後訊息被 Pub / Sub 視為無法處理。
- 可以加入[指數退避](#)的方式讓訊息在第一次失敗後等待最短指數退避的時間並重新傳送。

以上方法都可以盡量避免訊息被 Pub / Sub Retry，可以依照需求選擇最適合的方式。

常見問題

若訊息被 Retry 如何減緩影響？

如果您的訊息是擁有順序性的話，您可以為在訊息加入排序，可以讓來不及處理的訊息依照第一次的順序進行排序，可以避免您的服務請求順序出問題。

如何延長 ack 時間？

預設的「ack」時間為 10 秒，若要延長可以透過修改 [modifyackDeadline](#) 來延長「ack」的時間，最高可將「ack」時間延長至 60 分鐘。

若特定時間出現龐大流量該怎麼應對？

在 Pub / Sub 中提供使用 [Flow Control](#) 的方式針對瞬時流量進行處理。

結論

在設計服務流程時訊息處理是相當重要的一環，傳統地端可能需要自行架設、維護 Kafka、RabbitMQ... 這類型的服務費時又麻煩，不過 Google Cloud 提供 Pub/Sub 服務可以使用，並結合了保證訂閱、多對多、延時處理... 等優點，讓使用者可以減少管理訊息相關的時間，更專注在開發、優化服務上。

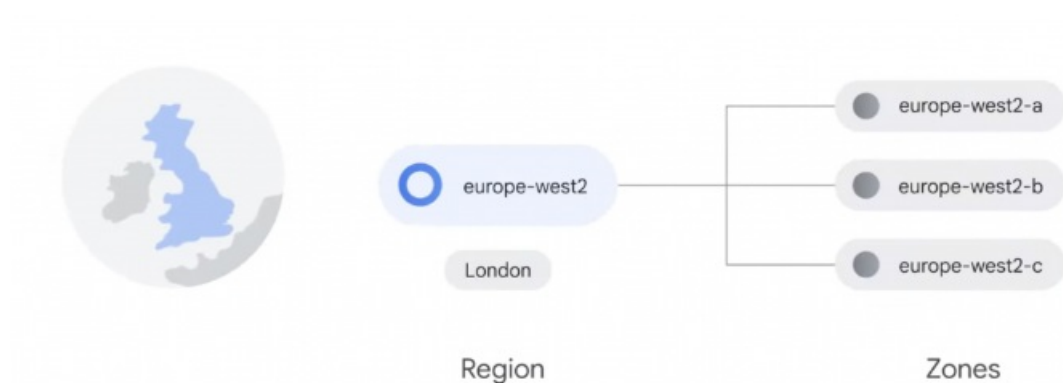
這次介紹了 Pub/Sub 功能優勢，6 大常見名詞與訊息生命週期。各位若有進一步的 Pub/Sub 操作疑問，歡迎[聯絡 Cloud Ace](#) 獲得更進一步的資訊。

【CCP】Google Cloud Platform 服務總覽

出處：<https://ithelp.ithome.com.tw/users/20151036/ironman/6131>

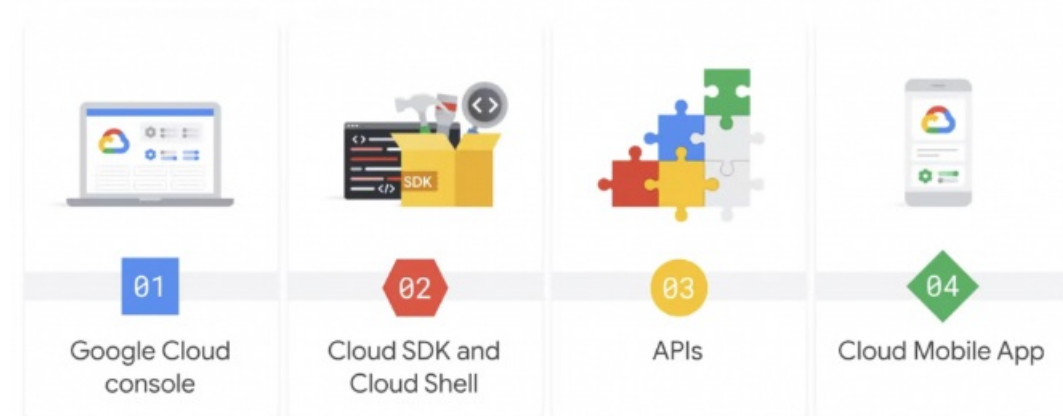
Region 和 Zone 的差別

在 Google Cloud 裡面，使用了 Region 和 Zone 的概念，分別去區分不同地理區塊的服務



管理 Google Cloud 的四種方法

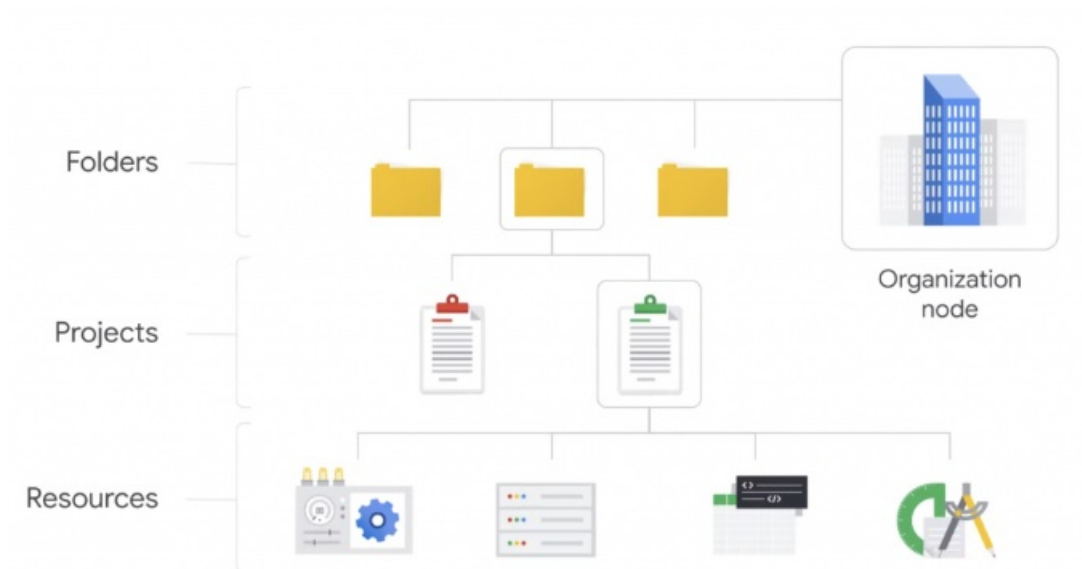
介紹完了 Region 和 Zone 的差別，接著介紹一下管理 Google Cloud 中的服務的四種方法



工具	介面類型	功能強度	使用時機
Google Cloud Console	圖形 UI	中等	建立資源、設定、報表
gcloud SDK	命令列工具	最強	自動化部署、開發整合、CI/CD
Cloud Shell	雲端終端機	高	快速部署/測試指令，不佔本地資源
Cloud APIs	程式介面	高	系統對接、自動化服務整合
Cloud Mobile App	手機 App	輕量	查看狀態、接收通知、緊急操作

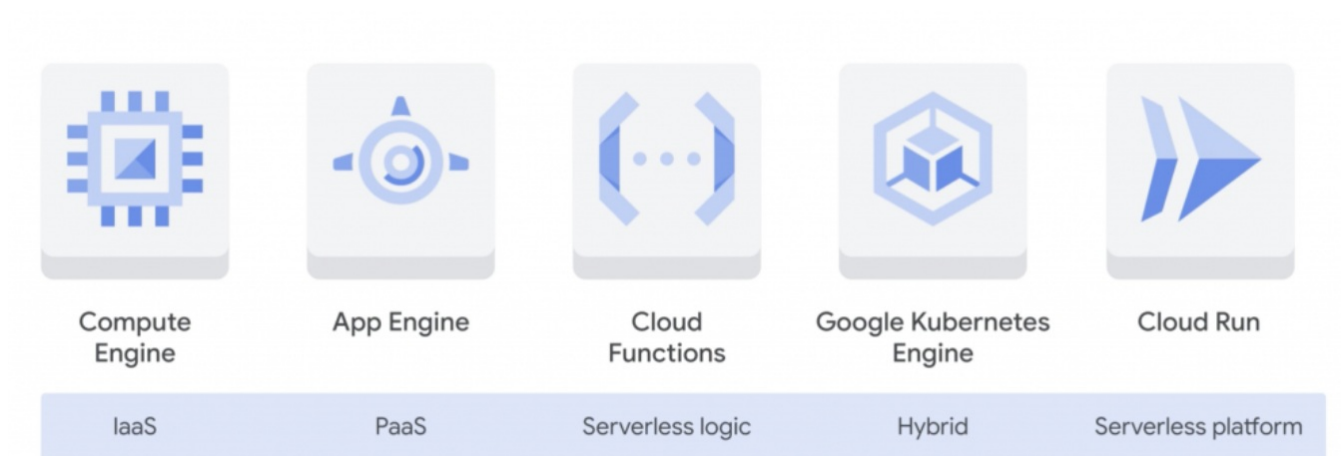
Google Cloud 中 Project 結構

下面這張圖描述了 Google Cloud 中的 Project 結構，不過這張圖要從最下層往上看



Google Cloud 中的運算服務

- Compute Engine
- App Engine
- Cloud Functions
- Google Kubernetes Engine
- Cloud Run



Google Cloud Run、App Engine、Cloud Functions 三者的完整比較：

三種 Serverless 服務比較表



項目	Cloud Run	App Engine	Cloud Functions
適用層級	容器 (Container) 層級	應用 (Application) 層級	函數 (Function) 層級
啟動方式	HTTP 請求觸發	HTTP、排程、Task Queue	HTTP、事件觸發 (如 Storage、PubSub)

項目	Cloud Run	App Engine	Cloud Functions
開發模式	自行打包容器，任何語言框架皆可	GCP 支援語言（Java, Python, Go, Node）	限制在特定語言框架（Node, Python, Go 等）
彈性調整	自動擴展至零	自動擴展（可設定最小/最大執行數）	自動擴展至零（每次執行為全新環境）
是否支援容器	可自定義容器	（App Engine Flex 支援，但較複雜）	（只支援函數級）
計費方式	請求數 + CPU + 記憶體 + 時間	請求數 + 時間 + 記憶體	函數呼叫次數 + 執行時間 + 資源
冷啟動	較低，若設定最小實例可避免	標準環境會冷啟，高級環境可控制	冷啟明顯（尤其低頻率觸發）
狀態保持	不可保留（Stateless）	不可保留	不可保留
部署單位	整個容器（任意語言）	應用程式檔案 + config.yaml	單一函數檔案（Function）
自訂網域 / HTTPS 支援	可設定自訂網域與 HTTPS		
IAM 權限控管	精細控制		

簡易選擇建議



你要的是...	建議使用
可以自由使用任何語言與架構	Cloud Run
最少設定、支援完整應用部署	App Engine
僅需要處理單一任務 / 事件驅動功能	Cloud Functions
想用 Docker 容器、可相容本地環境	Cloud Run
想部署 Flask、Express 但不想處理容器細節	App Engine
想做圖片上傳通知、排程簡訊等小任務	Cloud Functions

實際開發場景舉例：



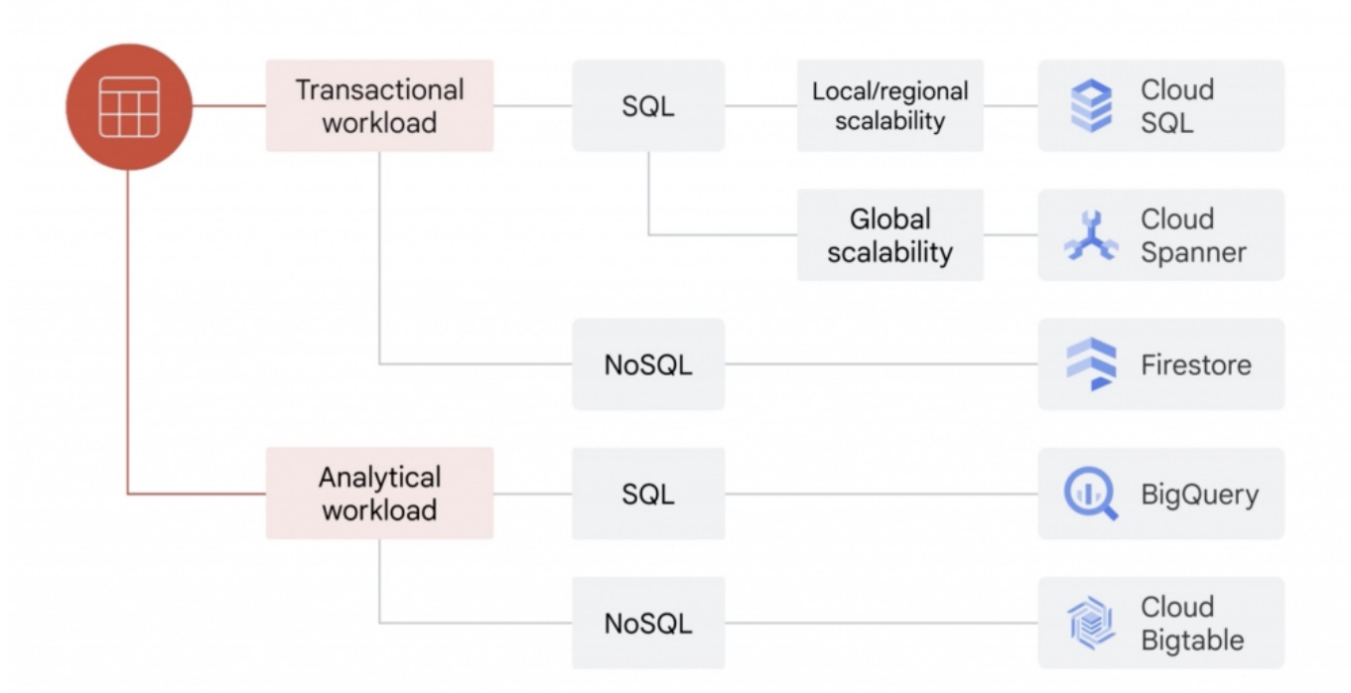
使用情境	適合服務
部署 Spring Boot 應用，並自定義 Dockerfile	Cloud Run
部署 Python Web（Flask）快速上線	App Engine
使用 GCS 上傳觸發縮圖處理	Cloud Functions
開發 RESTful API，架構使用 FastAPI + Redis + Gunicorn	Cloud Run
設定每日 08:00 發送報表	Cloud Functions（排程）

Google Cloud 中的數據儲存服務

Google Cloud 支援五種數據的儲存服務，而這五種服務，又可以分成三大類，分別是「一般檔案的儲存服務」、「SQL 的儲存服務」以及「NoSQL 的儲存服務」



數據儲存服務總結



Bigtable 與 Bigquery 比較

基本說明



名稱	說明
Bigtable	分散式 NoSQL 資料庫，適合高頻率讀寫、低延遲的應用（類似 HBase）
BigQuery	分析型資料倉儲，專門處理超大資料集的查詢與分析（類似 Redshift、Snowflake）

詳細比較表



項目	Bigtable	BigQuery
資料類型	NoSQL，Key-Value / Wide-column store	SQL-based，結構化資料表
適用場景	即時讀寫、大量設備資料、時序資料（IoT、日誌）	大數據分析、商業智慧報表、ETL 處理

項目	Bigtable	BigQuery
□ 查詢方式	API / gRPC / HBase client	SQL (支援標準 SQL)
□ 查詢靈活度	限制多 (只能依 row key 查詢)	高 (可聚合、JOIN、Window Function)
✗ 效能焦點	寫入快、讀取延遲低	掃描大量資料效率高
□ 儲存設計	儲存在 Colossus + SStable (HBase-like 分散式設計)	儲存在 Dremel 架構 (專為資料分析優化)
□ 索引支援	□ 沒有二級索引，僅支援 row key 前綴	□ 支援複雜查詢與分析
□ 每秒處理量	百萬級 TPS (高併發)	適合大批量 (TB~PB) 分析
□ 查詢延遲	毫秒等級	秒 ~ 分鐘等級
□ 計費方式	根據儲存容量 + 節點數 (可橫向擴充)	根據掃描的資料量計價 (可選定價模式)
□ 管理需求	類似資料庫 (需設計 schema、選 row key、規劃節點)	類似報表工具 (丟資料就能查，幾乎零維運)

□ 使用情境建議



需求/情境	建議使用
IoT 裝置每秒回傳大量資料	□ Bigtable
要統計每月業績報表、跑分析 SQL	□ BigQuery
需要實時查詢某一筆用戶記錄	□ Bigtable
想跑 SELECT count, avg, group by	□ BigQuery
需要和 Data Studio 或 Looker 整合報表	□ BigQuery
想儲存百億筆資料但需低延遲查詢	□ Bigtable

□ 搭配使用範例

很多系統會這樣搭配：

1. **Bigtable：實時寫入**
 - 寫入裝置或行為記錄 (如瀏覽行為、IoT 資料)
2. **定期匯入到 BigQuery**
 - 跑 ETL，進行分析與報表產出

Google Cloud 中的 API 管理服務

影片中有提到三種 Google Cloud 中常見的 API 管理服務，分別是：

- Apigee API Management
- API Gateway
- Cloud Endpoints

□ 快速對比表：Apigee vs API Gateway vs Cloud Endpoints



項目	Apigee	API Gateway	Cloud Endpoints
□ 定位	企業級 API 管理平台	簡化、輕量的 GCP 原生 API Gateway	開發者導向的輕量級 API Proxy
□ 建立方式	使用 Apigee UI 建立 API Proxy	部署 <code>api-gateway.yaml</code> + GCP service	使用 ESP (Extensible Service Proxy) 與 OpenAPI 配置
□ 功能完整性	□ 完整 API 生命週期管理 (分析、流量管控、金鑰等)	△ 僅基本 API Proxy 與驗證	△ 基本功能 + Firebase / Auth0 整合

項目	Apigee	API Gateway	Cloud Endpoints
□ 分析與報表	□ 內建精細分析報表	□ 無分析功能	△ 可搭配 Stackdriver/Cloud Logging
□ 驗證/授權支援	□ OAuth2, API Key, JWT, SAML, 多種認證 ◀ <input type="text"/> ▶	□ JWT / API Key / IAM	□ JWT / API Key / Firebase
⚙ API 開發方式	提供 GUI、Flow 編輯器、邏輯處理	使用 OpenAPI 定義 + 設定 YAML	OpenAPI / gRPC 定義 (ESP Proxy)
□ 效能	高 (企業級 SLA)	中等, 適合輕量流量	輕量, 設計給內部/開發測試用
□ 收費模式	依照 API 呼叫數、功能計費 (需額外購買)	依流量計費 (依照 GCP 定價)	依照 ESP Proxy 的 GKE/GCE/Cloud Run 運行費用計費
□ 運行平台	Apigee X (GCP原生)、Hybrid	Cloud Run / GKE / GCE	GKE / Cloud Run / App Engine
□ 複雜功能 (限流、轉換)	□ 支援 (flow policy 可自定義 JSON/XML 轉換等)	□ 不支援	□ 不支援
□ 部署簡易性	中等 (企業級 GUI + 設定需學習)	高 (YAML + CLI)	中等 (需熟 ESP/OpenAPI)

□ 應用場景建議

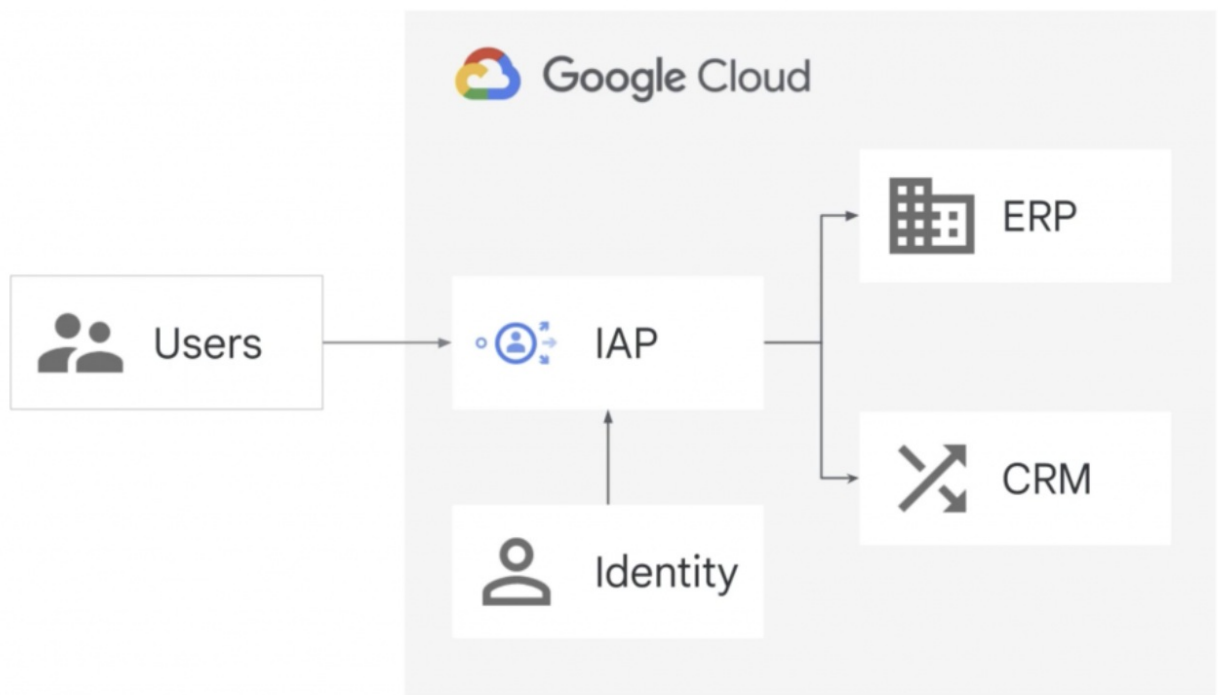


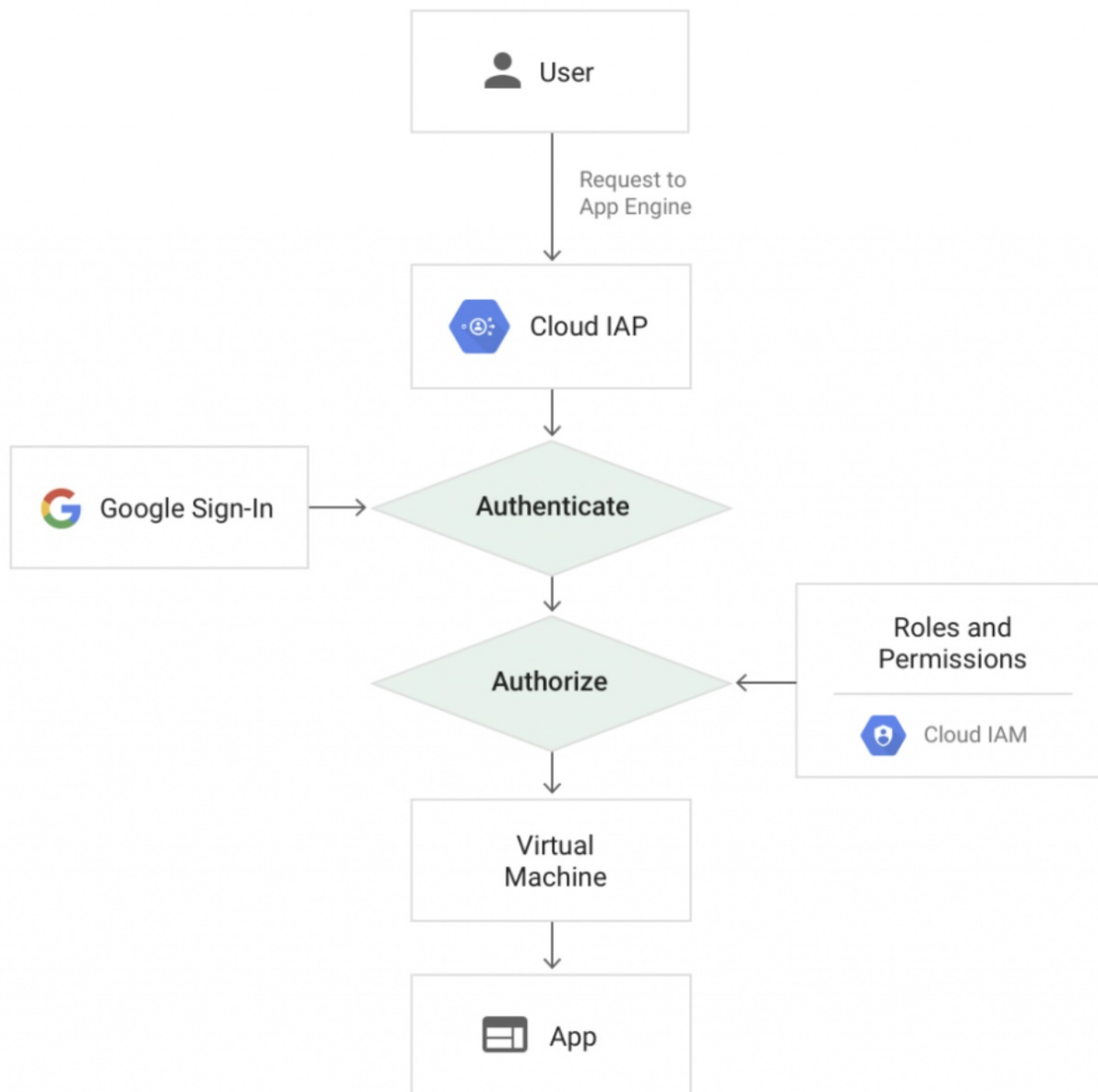
需求 / 狀況	建議使用
企業需要完整 API 管理功能 (流控、報表等)	□ Apigee
GCP 環境輕量部署 + IAM 驗證 API	□ API Gateway
開發階段/原型開發 + 自訂認證方式	□ Cloud Endpoints
需要整合 Firebase 認證	□ Cloud Endpoints
多租戶 API 管理、API 商業化 (API Monetization)	□ Apigee
想快速部署在 Cloud Run + IAM 控制	□ API Gateway

□ 技術選擇心法：

- □ 選 **Apigee** 如果你是企業客戶、要做 API 授權金鑰控管、頻率限制、報表、開發者入口網站 (dev portal) 等。
- □ 選 **API Gateway** 如果你只是想快速幫 Cloud Run 上的 API 加上 HTTPS、安全性。
- □ 選 **Cloud Endpoints** 如果你熟 OpenAPI、只想讓 API 有認證與基本日誌記錄，不需要 GUI。

權限管理服務





說明 Iaas Paas Saas Faas

這四個名詞是雲端服務的不同層級與服務模型，分別為：

□ IaaS (Infrastructure as a Service) 基礎設施即服務

- **提供內容：** 虛擬機 (VM)、儲存空間、網路、作業系統等基礎資源
- **使用者負責：** 自行安裝應用程式、中介軟體、設定安全性等
- **彈性高，自由度大**
- **常見例子：**
 - AWS EC2
 - Google Compute Engine
 - Microsoft Azure VM

“ □ 適用於：需要完整控制權限的系統管理員與 DevOps 團隊

☐ PaaS (Platform as a Service) 平台即服務

- **提供內容：**開發平台（含OS、Runtime、DB、Framework）
- **使用者負責：**上傳應用程式，開發、部署、維護應用邏輯
- **省去基礎建設與系統維護負擔**
- **常見例子：**
 - Google App Engine
 - Heroku
 - AWS Elastic Beanstalk

“☐ 適用於：開發人員專注寫程式，不想管底層架構

☐ SaaS (Software as a Service) 軟體即服務

- **提供內容：**完整的軟體應用，通常透過瀏覽器存取
- **使用者負責：**只需要使用，不用管安裝、更新、維護
- **完全託管、快速上手**
- **常見例子：**
 - Gmail / Outlook
 - Google Docs / Office 365
 - Salesforce
 - Dropbox

“☐ 適用於：終端使用者或企業快速取得現成解決方案

✂ FaaS (Function as a Service) 函數即服務

- **提供內容：**運行特定函數或任務的執行環境
- **使用者負責：**撰寫函數邏輯，其他都交給雲端處理（包含資源擴展）
- **屬於 Serverless 架構的一種**
- **常見例子：**
 - AWS Lambda
 - Google Cloud Functions
 - Azure Functions

“☐ 適用於：事件驅動的任務（如：圖檔轉換、資料處理、API 回應等）

對照圖表總覽：



層級	你負責的部分	例子
IaaS	App + OS + Middleware + DB	AWS EC2、GCP VM
PaaS	App + DB	Heroku、GCP App Engine
SaaS	僅使用	Gmail、Google Docs
FaaS	僅寫 function	AWS Lambda、GCF