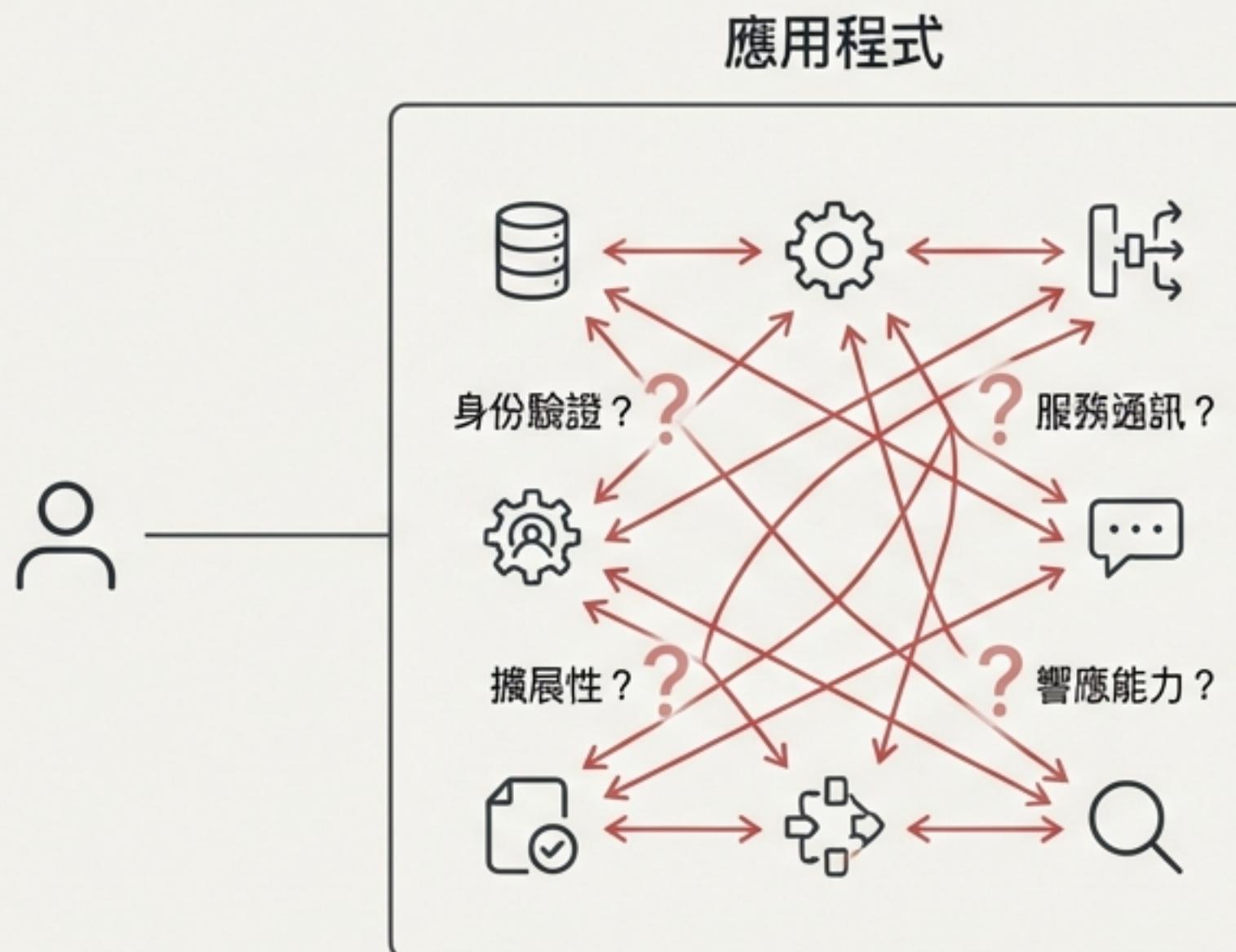


主標題：打造現代化應用程式：從身份驗證到事件驅動架構

副標題：活用 AWS Cognito 與 Amazon EventBridge 的深度實踐



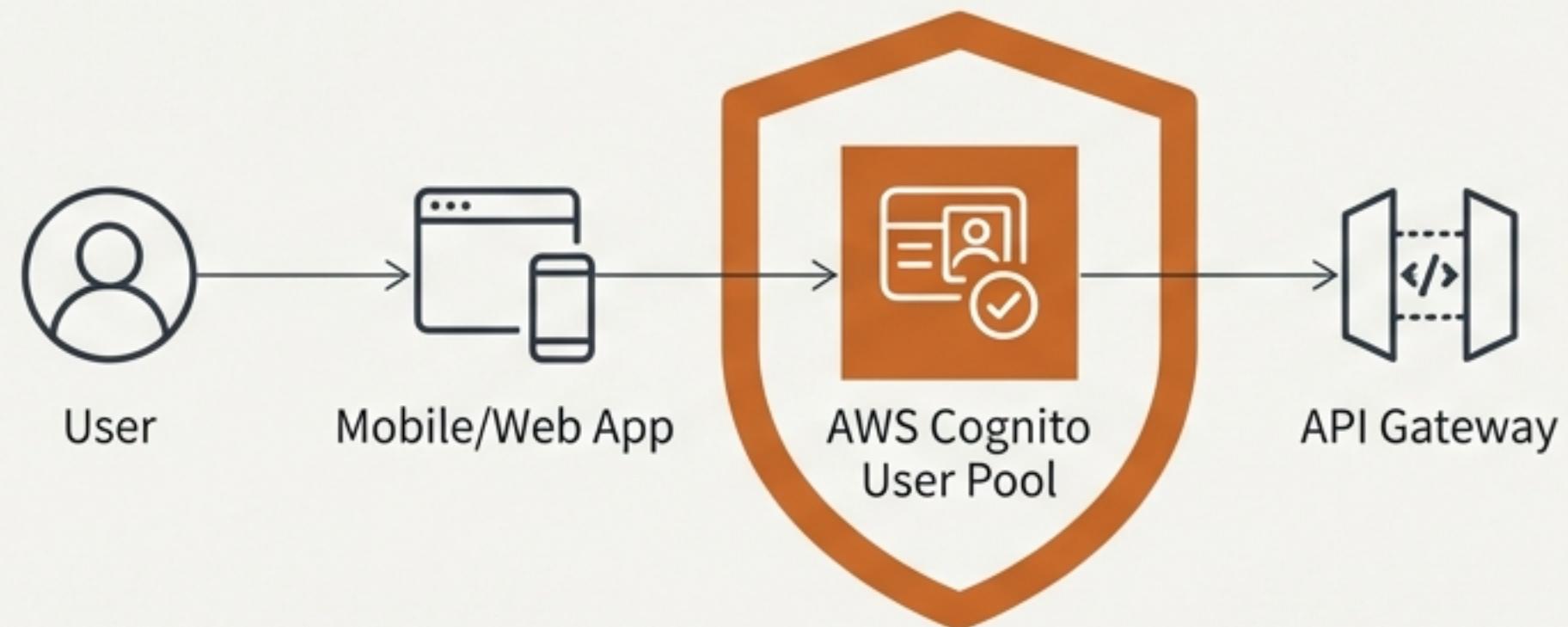
現代應用的核心挑戰：耦合、擴展與安全



現代應用程式架構必須應對一系列複雜挑戰：

- **使用者管理**：如何安全地處理數百萬用戶的註冊、登入和身份驗證？
- **服務通訊**：微服務之間如何高效、可靠地溝通，同時避免形成緊密耦合的「分散式單體」？
- **可擴展性**：當業務需求增長時（例如從 B2C 擴展到 B2B），架構如何平滑地擴展以支持多租戶和複雜的業務邏輯？
- **響應能力**：系統如何對用戶行為或內部事件做出即時反應，觸發下游工作流程？

解決方案之基石：以 AWS Cognito 建立託管式身份層



AWS Cognito 提供了一個完全託管的身份服務，為我們的應用程式奠定安全的基礎。它負責處理所有與使用者身份相關的繁重工作，讓我們能專注於核心業務邏輯。

- **核心功能：**使用者註冊、登入、忘記密碼、Token 管理 (JWT)、聯合身份驗證 (Federation) 等。
- **架構角色：**作為應用程式的身份驗證和授權中心，保護後端 API 和資源。

客製化使用者旅程：透過 Lambda Triggers 注入業務邏輯

Cognito 的強大之處在於其可擴展性。Lambda Triggers 允許我們在使用者生命週期的關鍵節點（如註冊前）執行自訂程式碼，從而實施業務規則。

範例：註冊前自動確認使用者（Pre-Signup Trigger）

這個位於 `presignup/preSignUp.mjs` 的觸發器，可以在用戶註冊時自動確認帳戶並驗證其電子郵件，簡化了用戶入門流程。

在源頭實施業務規則

```
// presignup/preSignUp.mjs
export const handler = async (event) => {
    // Auto confirm users
    event.response.autoConfirmUser = true;

    // Auto verify email
    if (event.request.userAttributes.hasOwnProperty("email")) {
        event.response.autoVerifyEmail = true;
    }
    return event;
};
```

響應使用者事件：用戶確認後的第一個動作

當使用者成功確認帳戶後，系統需要做出反應。Post-Confirmation Trigger 正是實現此目標的關鍵。這不僅是一個觸發器，更是我們架構中第一個有意義的「業務事件」。

範例：在 DynamoDB 中建立使用者資料 (Post-Confirmation Trigger)

此範例

`postconfirmation/postConfirmation.mjs` 觸發器會在用戶確認後，立即在 `Users` 資料表中為其建立一筆記錄。這是一個典型的事件驅動模式：一個事件（用戶確認）觸發了一個動作（寫入資料庫）。

```
// postconfirmation/postConfirmation.mjs
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const docClient = DynamoDBDocumentClient.from(new DynamoDBClient({}));

export const handler = async (event) => {
  if (event.triggerSource === "PostConfirmation_ConfirmSignUp") {
    const { email, sub } = event.request.userAttributes;
    await docClient.send(
      new PutCommand({
        TableName: "Users",
        Item: {
          userId: sub,
          email: email,
          createdAt: new Date().toISOString()
        }
      })
    );
    return event;
  }
};
```

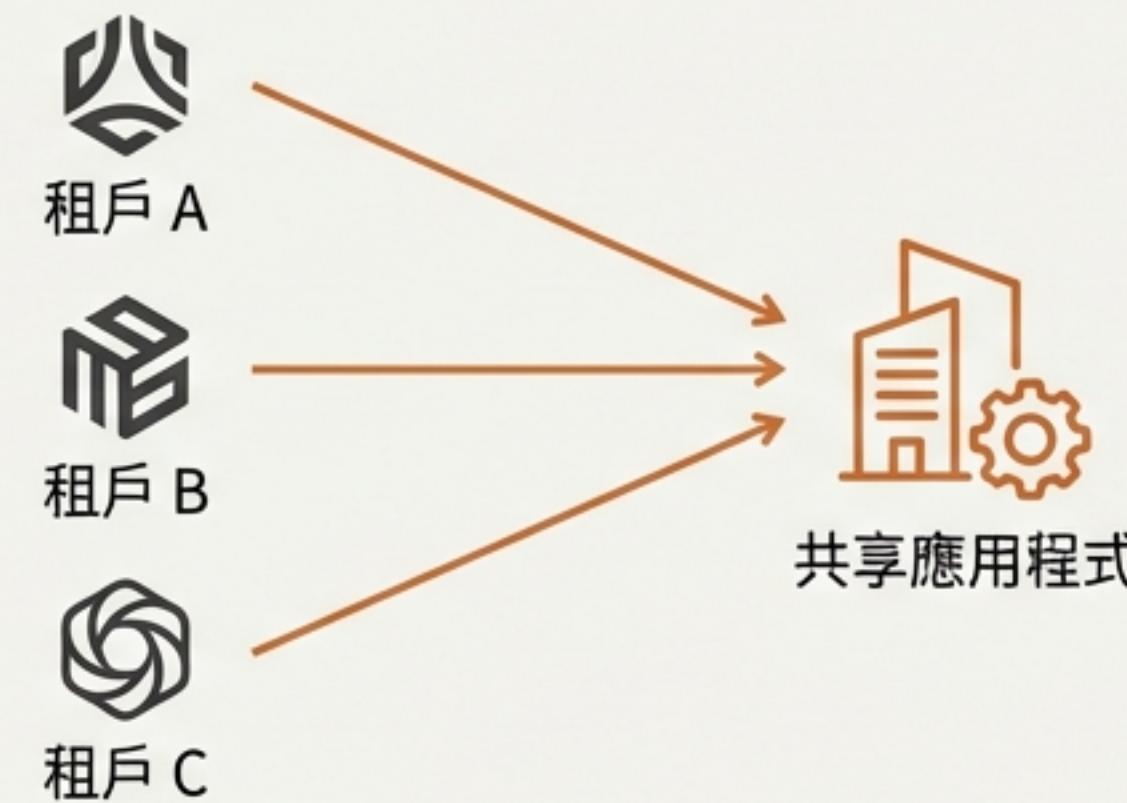
使用者確認 (事件) → 建立使用者設定檔 (動作)



DynamoDB



業務擴展的必然：為 B2B SaaS 設計多租戶架構



隨著應用程式的成功，下一步通常是進入 B2B 市場。這帶來了一個核心的架構挑戰：多租戶 (Multi-Tenancy)。我們需要讓多個企業客戶（租戶）在共享的基礎設施上安全、隔離地使用我們的服務。

場景：

- 一個 B2B Web App，客戶是不同公司 (TenantA, TenantB...)。
- 所有客戶都使用同一個網址登入 (e.g., `https://app.example.com`)。
- 每個客戶都希望使用自己的身份提供者 (IdP) 進行員工身份驗證，如 Okta、Entra ID 或 Google Workspace。

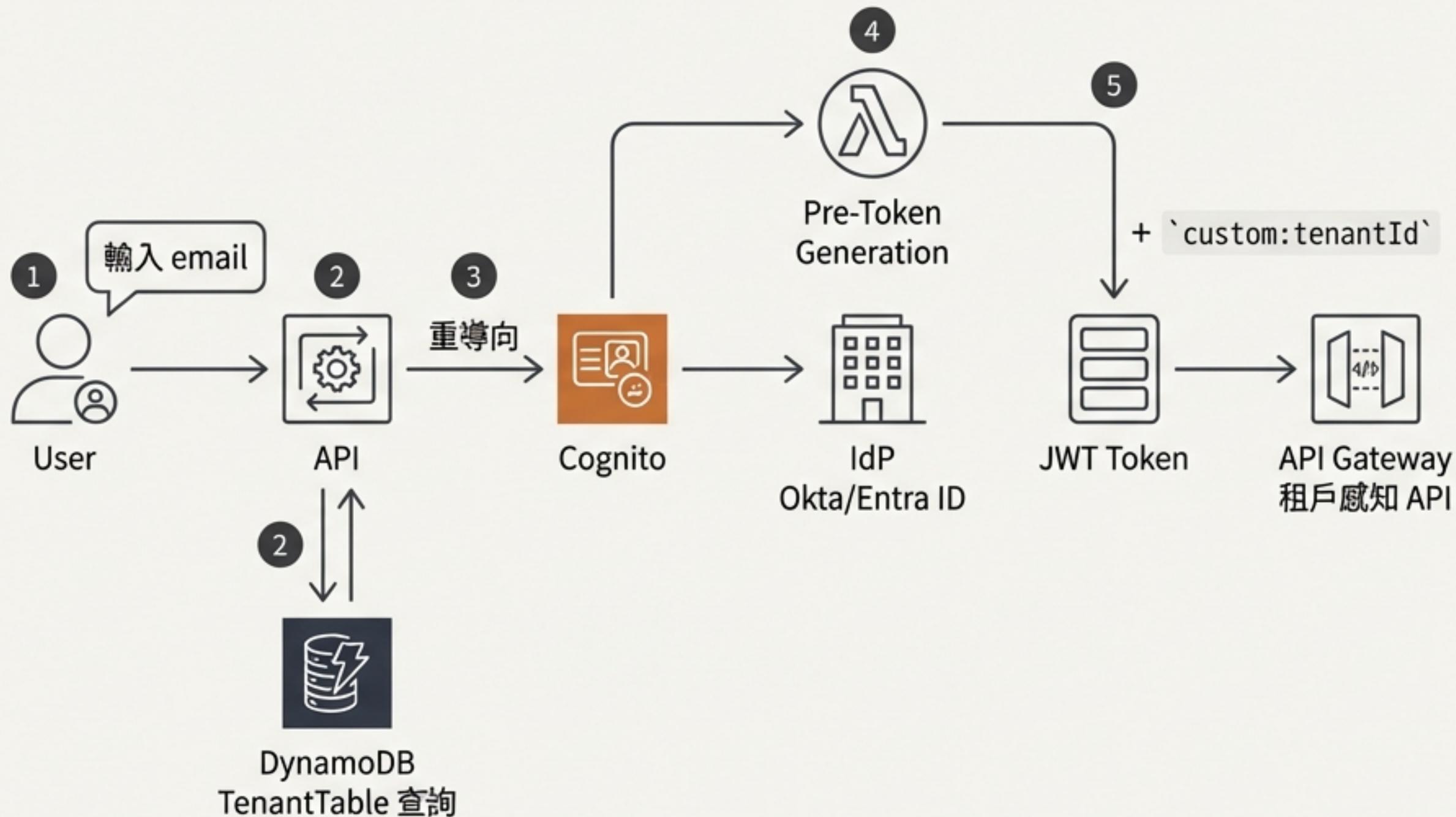
Cognito 多租戶身份驗證的三種常見模型

在 Cognito 中實現多租戶有多種策略，每種策略在隔離性、管理成本和使用者體驗方面都有不同的權衡。

模型	作法	優點	缺點
每租戶一個 User Pool	為每家公司建立獨立 User Pool	隔離度最高、各租戶可有不同的密碼/MFA 政策	管理成本高、租戶多時難維運
共用 User Pool + App Client 分租戶	同一個 User Pool，為每個租戶建一個 App Client，並綁定各自 IdP	共用使用者資料，仍可對租戶客製 IdP 與 Redirect URL	App Client 變多、設定較複雜
共用 User Pool + 自訂屬性分租戶	單一 User Pool，使用 custom:tenantId 等屬性標記租戶，後端以此做隔離	Sign-up / Sign-in URL 完全共用，體驗最一致，擴充租戶最簡單	授權與資料隔離需在應用和 API 層嚴格實作

深度解析推薦模型：單一 User Pool + 自訂屬性

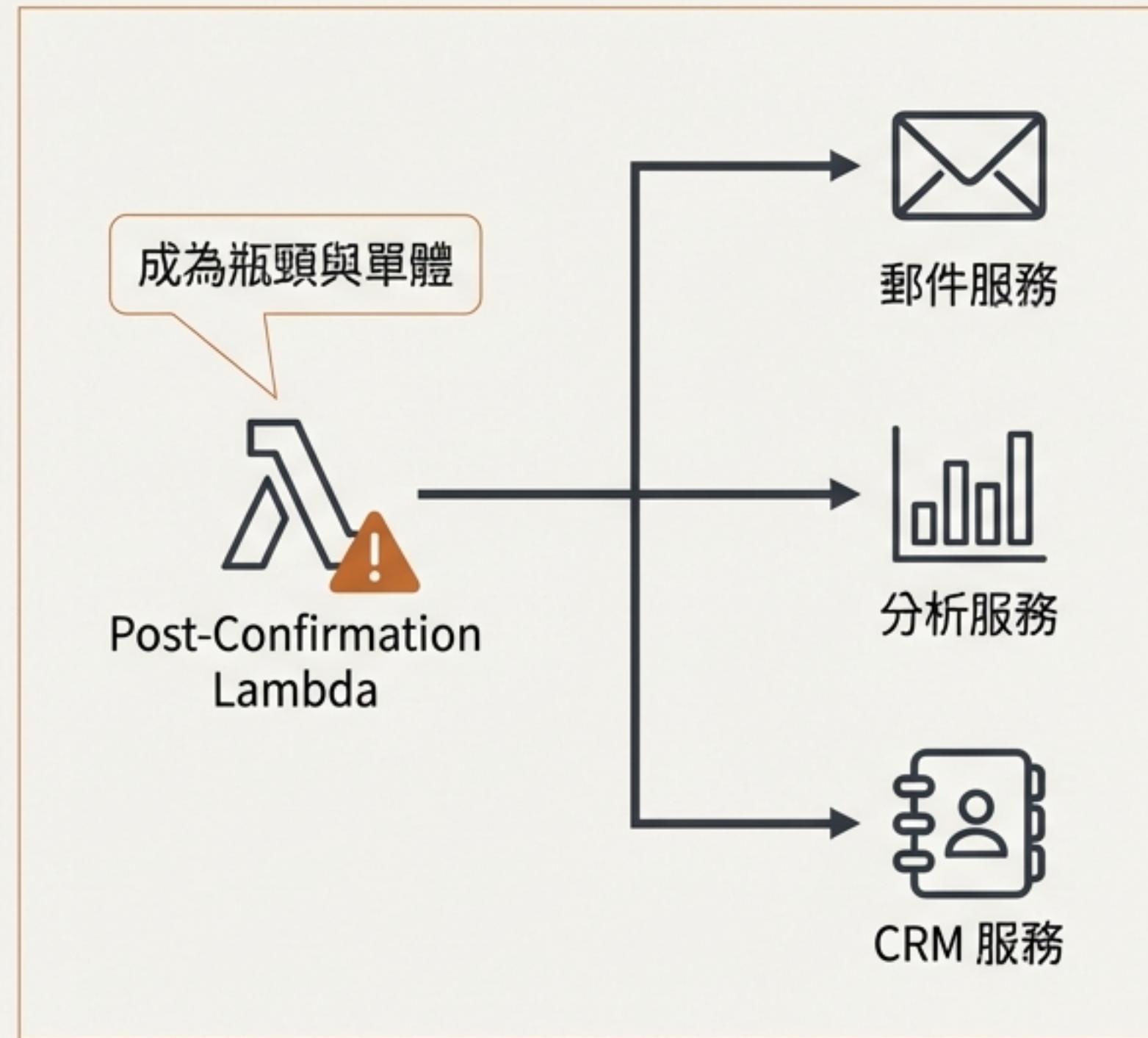
此模型在擴展性和使用者體驗之間取得了最佳平衡，是大多數 B2B SaaS 應用的首選。



登入與授權流程：

- 識別租戶**：使用者輸入 email (`alice@a-company.com`)。API 透過 email domain 查詢 DynamoDB 中的 `'TenantTable'`，找到對應的 IdP。
- 定向登入**：系統將使用者重導向至 Cognito，並指定使用該租戶的 IdP (如 Okta)。
- 注入租戶資訊**：使用者在 IdP 成功登入後，Cognito 觸發 **Pre-Token Generation Lambda**。
- 豐富 Token**：Lambda 查詢 `'TenantTable'`，將 `'custom:tenantId'` 和 `'custom:usergroups'` 等資訊寫入 JWT。
- 應用層隔離**：後端 API 驗證 JWT 並讀取 `'custom:tenantId'`，以此作為所有資料查詢和授權的依據。

超越點對點觸發：系統級解耦的需求



Cognito Lambda Triggers 非常適合處理與身份直接相關的單一任務（如在 DynamoDB 中建立使用者）。但如果多個下游服務都需要對「使用者已確認」這個事件做出反應呢？

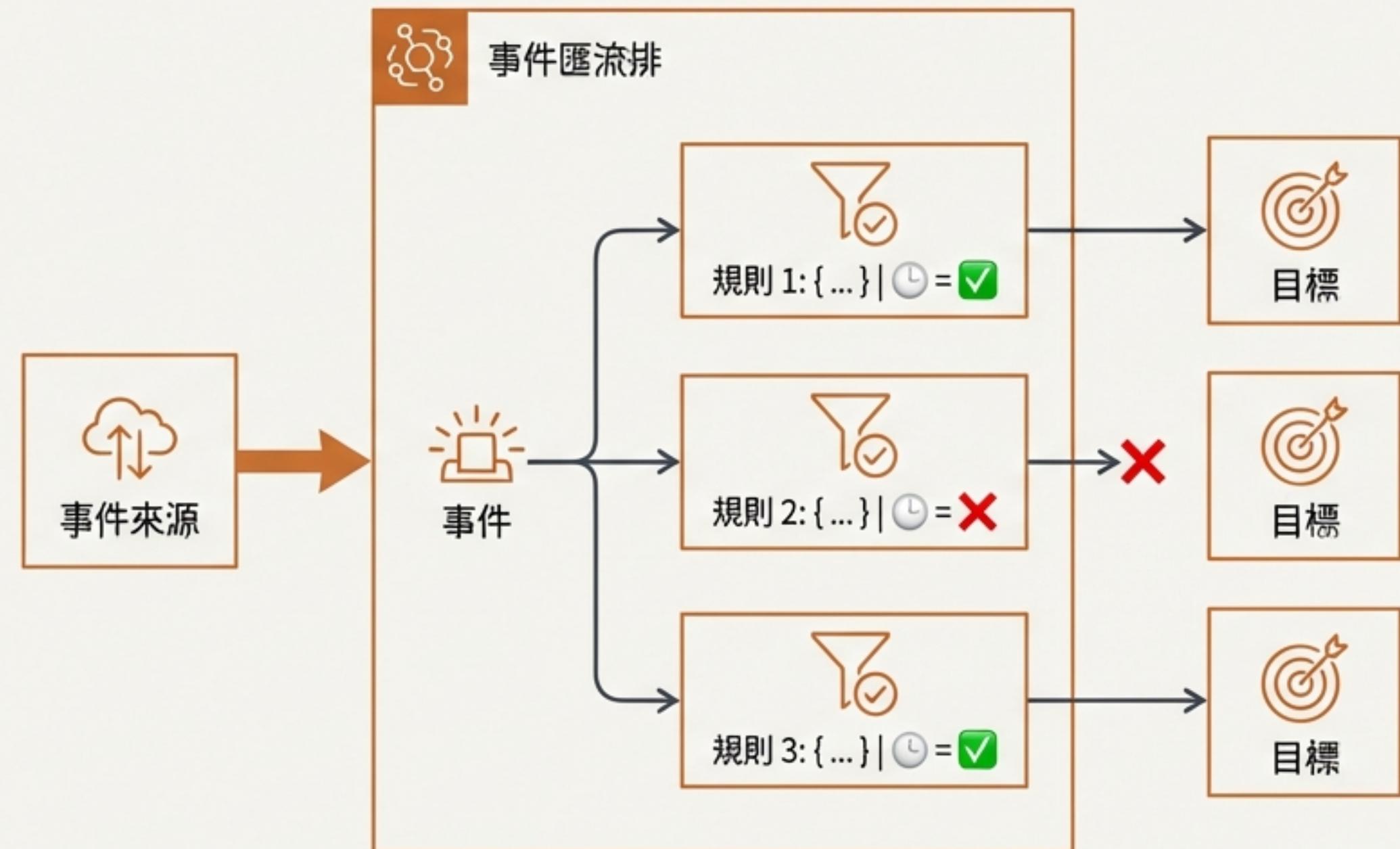
- **通知服務**：需要發送歡迎郵件。
- **分析服務**：需要追蹤新用戶註冊指標。
- **CRM 服務**：需要在客戶關係管理系統中建立聯絡人。

如果將所有邏輯都放入 Post-Confirmation Lambda 中，它會變得臃腫、脆弱且難以維護。我們需要一個更強大的機制來廣播事件，實現真正的服務解耦。

解決方案之核心：以 Amazon EventBridge 作為系統神經中樞

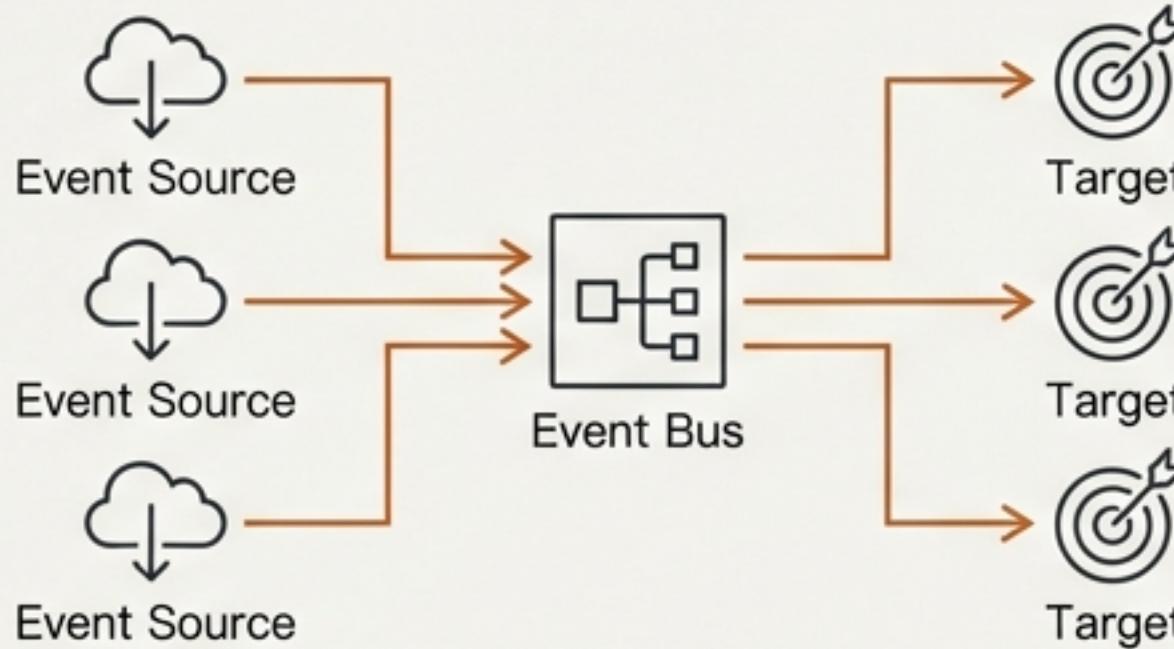
Amazon EventBridge 是一個無伺服器事件匯流排 (Event Bus) ，它使應用程式的不同組件能夠透過發布和響應事件來進行通訊。它讓我們能夠建立可擴展、可靠且鬆散耦合的事件驅動架構。

- **核心概念：**
 - **事件來源 (Event Source)**：產生事件的實體（如 AWS 服務、自訂應用程式）。
 - **事件匯流排 (Event Bus)**：接收事件並根據規則進行路由的管道。
 - **規則 (Rules)**：根據事件內容 (Event Pattern) 進行過濾，決定將事件發送到哪些目標。
 - **目標 (Targets)**：接收並處理事件的資源（如 Lambda、SQS、SNS）。



EventBridge 實戰：Event Buses vs. Pipes

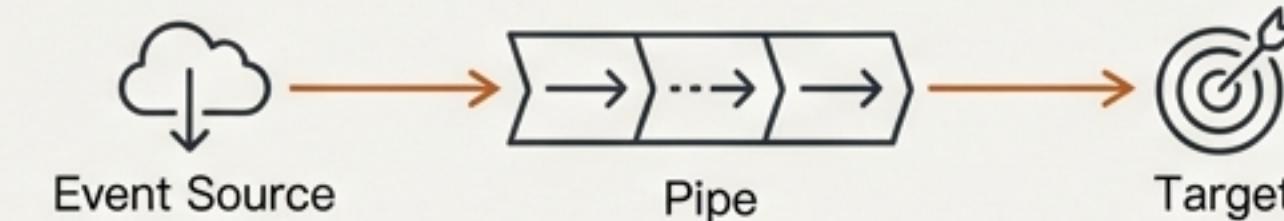
EventBridge 提供了兩種主要的方式來處理和傳遞事件，適用於不同的整合場景。



事件匯流排 (Event Buses) :

用途：多對多路由。適用於將來自多個來源的事件路由到多個目標。

範例：「使用者已確認」事件發布到事件匯流排，多個服務（郵件、分析、CRM）可以訂閱並各自處理。



管道 (Pipes) :

用途：點對點整合。用於將單一來源的事件傳遞到單一目標，中間可選性地進行篩選、擴充和轉換。

範例：將 DynamoDB Stream 的變更事件直接、可靠地傳送到 SQS 倉列。

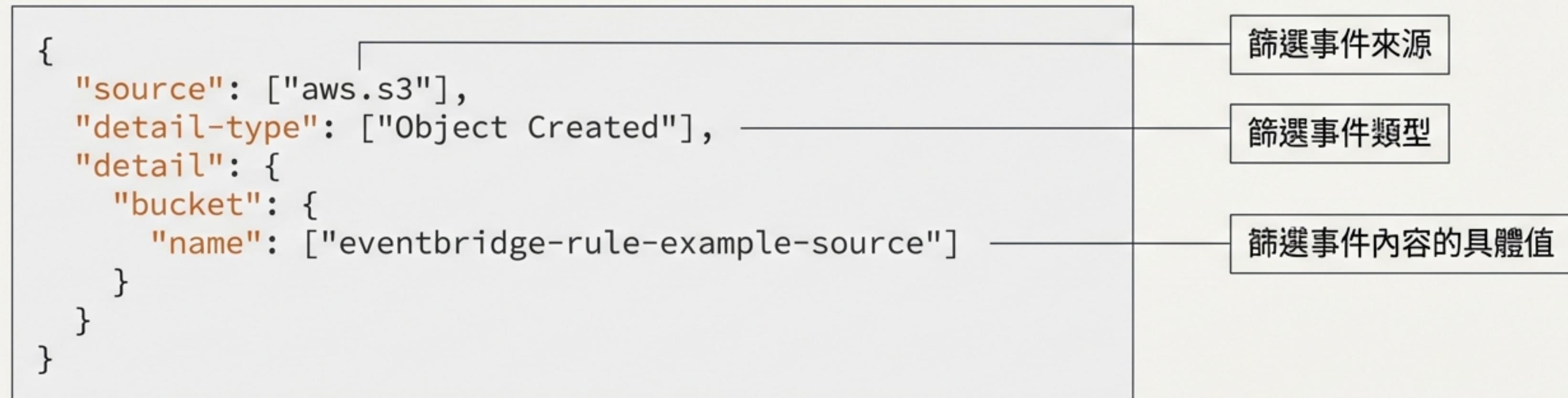
精準路由的藝術：使用事件模式 (Event Patterns) 進行過濾

EventBridge 的強大威力來自其靈活的規則引擎。透過定義「事件模式」，目標可以只訂閱它們感興趣的事件，而忽略所有其他流量，從而實現真正的解耦。

事件模式是一個 JSON 物件，用於匹配傳入事件的結構和內容。

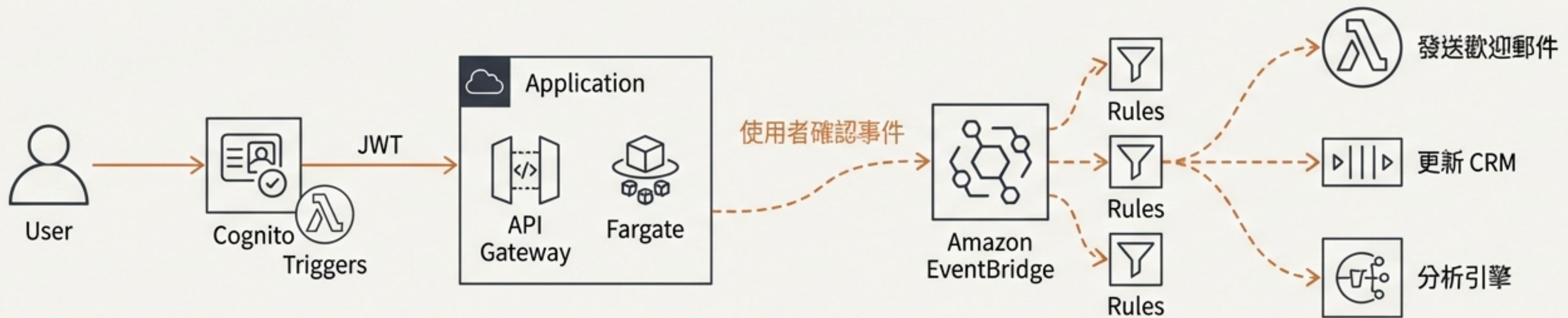
範例：匹配 S3 物件建立事件

以下規則只會匹配來自特定 S3 儲存桶的 `Object Created` 事件。任何其他來源或其他類型的 S3 事件都會被忽略。



完整架構圖：從使用者到事件驅動的全景

這張圖展示了我們沿途構建的完整、現代化的應用程式架構。它將安全的身份驗證與可擴展的事件驅動設計無縫結合。



流程：

1. 使用者透過 Cognito 進行身份驗證，Lambda Triggers 處理客製化邏輯（如 B2B 租戶識別）。
2. 應用程式中的用戶行為（如「用戶確認」）產生一個事件。
3. 該事件被發布到 **Amazon EventBridge**。
4. EventBridge 上的 規則 根據事件內容進行過濾。
5. 匹配的事件被路由到多個解耦的下游 目標服務 （如 Lambda、SQS、SNS），每個服務獨立執行其業務邏輯。

現實考量：大規模下的成本管理

雖然 Cognito 功能強大，但在百萬級月活躍使用者（MAU）的規模下，成本可能成為一個重要因素（每月約 4,415 美元，不含進階安全功能）。以下是來自社群和實踐的成本優化策略：



本地驗證 JWT

對於頻繁的 API 請求，應在 API Gateway 或後端服務中本地驗證 Access Token 的簽名和宣告，而不是每次都呼叫 Cognito 的 `GetUser` API。這可以顯著減少被計為 MAU 的操作。



考慮自建方案（極端規模）

對於有特定需求或需要極致成本控制的超大規模應用，可以評估自建身份驗證伺服器（如 Keycloak on ECS/Fargate）。這會將成本從按用戶收費轉移到基礎設施和維運成本上。



洽談用量折扣

對於可預測的高用量，請務必與您的 AWS 銷售代表聯繫，洽談「承諾使用折扣」（committed use discount），這通常能帶來比定價更優惠的價格。

關鍵架構原則回顧

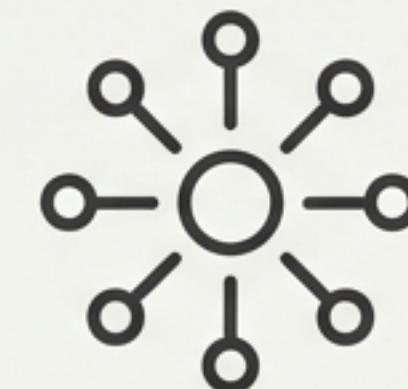
透過這趟架構之旅，我們建立了三個核心原則，以打造穩健、可擴展的現代化應用：



1.



2.



3.

從安全的身份層開始

使用 AWS Cognito 作為起點，建立一個安全、可客製化的身份驗證基礎。不要自己動手造輪子。

為 B2B 擴展設計身份

採用「單一 User Pool + 自訂屬性」的多租戶策略，以應對未來業務增長，同時保持一致的使用者體驗。

用事件匯流排解耦一切

引入 Amazon EventBridge 作為系統的神經中樞，讓您的服務能獨立演進、部署和擴展，實現終極的靈活性。