

AWS 相關

- [【AWS】Day 2：AWS 的基本知識](#)
- [【AWS】Day 3：AWS 中的帳號與資源管理架構](#)
- [【AWS】Day 4：AWS 運算服務簡介](#)
- [【AWS】Day0 AWS 系列文：從實務出發的雲端探索之旅（共 30 篇）](#)
- [【AWS】Day 5：IaaS、PaaS、FaaS、SaaS 的差別](#)
- [【AWS】Day 6：運算服務 - Amazon EC2（虛擬機服務）](#)
- [【AWS】Day 7：運算服務 - Elastic Beanstalk（自動化部署服務）](#)
- [【AWS】【Elastic Beanstalk】常見部署結構推薦指南&費用最佳化建議](#)
- [【AWS】Day 8：運算服務 - AWS Lambda（Serverless 運算服務）](#)

【AWS】Day 2：AWS 的基本知識

Day 2：AWS 的基本知識

上一篇文章我們大致介紹了什麼是雲端服務，並且認識了幾個主流的雲端服務商，像是 AWS、GCP、Azure.....等等。

這篇文章則會來介紹 AWS (Amazon Web Services) 裡面的一些基本概念，包括：

- AWS 的 Region 和 Availability Zone (簡稱 AZ)
- AWS 上的資源範圍 (Region、AZ、Global)
- 管理 AWS 資源的方法有哪些

Region 和 Availability Zone 是什麼？

AWS 和 GCP 一樣，也使用了類似的概念來區分地理區域，他們稱作：

- **Region (區域)**
- **Availability Zone (可用區)**

Region (區域)

Region 就是地理位置的概念，例如：

- 台灣最近的是 **Asia Pacific (Tokyo)**，代碼是 `ap-northeast-1`
- 新加坡是 `ap-southeast-1`
- 美國東岸的維吉尼亞北部是 `us-east-1` (也是最多 AWS 服務最早上線的 Region)

截至 2023 年，AWS 在全球已經有超過 **30 個以上的 Region**，每個 Region 都是 AWS 自建的實體資料中心群。

Availability Zone (AZ，可用區)

每一個 Region 底下會有多個 Availability Zones (通常 2~6 個)，可以想像成是一個個獨立的資料中心 (不同大樓、不同電力、不同網路線路)，目的是增加可靠性。

舉例來說：

- `us-east-1` 底下就有 `us-east-1a`、`us-east-1b`、`us-east-1c`.....等 AZ
- `ap-southeast-1` (新加坡) 底下有 `ap-southeast-1a`、`ap-southeast-1b`、`ap-southeast-1c`

AZ 是 AWS 用來實現高可用 (High Availability) 的基本單位。服務部署在多個 AZ，可以避免單一機房故障造成整個應用中斷。

資源範圍：Region / AZ / Global 的區分

在 AWS 上，每一種服務在建立時都有自己的作用範圍，分為三種：

□ 以 AZ 為單位的服務

像是 EC2 (虛擬機)、EBS (區塊儲存) 就是屬於這一類。

- 在建立 EC2 時，你需要選擇一個 AZ，像是 `ap-northeast-1a`
- EBS 只能掛載在同一個 AZ 裡的 EC2
- 如果你要做高可用，就要部署在多個 AZ，並自己設計 failover 機制

□ 以 Region 為單位的服務

像是 S3（物件儲存）、RDS（資料庫）、Elastic Load Balancer、VPC 網路等等，都是 Region 級別的服務。

- 在一個 Region 中建立的 S3 bucket，可以讓這個 region 內的資源共同使用
- RDS 則可以設定 Multi-AZ（跨 AZ 高可用）

□ 以 Global 為單位的服務

有些服務是全域性的，例如：

- IAM（身份與權限管理）
- Route 53（DNS 服務）
- CloudFront（CDN 服務）

這些服務在建立時不會要求你選擇 Region，它們的作用範圍就是「整個 AWS 帳號或全球網路」。

管理 AWS 的四種方式

在 AWS 上，我們有四種主要方式可以操作與管理服務：

1. AWS Management Console（管理主控台）

透過瀏覽器進入 <https://console.aws.amazon.com>，就可以用圖形化介面來操作各種資源。

是最簡單、最直覺的操作方式。

2. AWS CLI（Command Line Interface）

透過終端機安裝 CLI 工具（`aws` 指令），可以用命令列操作 AWS 的服務。

適合自動化部署、寫腳本、CI/CD 等情境。

3. AWS SDKs

AWS 提供多種語言的開發套件（Java, Python, Node.js, Go...等），可以直接透過程式碼呼叫 AWS API。

適合建構自動化應用或將 AWS 整合進你自己的系統。

4. AWS Mobile App

AWS 官方也有提供手機 App，可以用來查看服務狀態、成本、帳單、甚至手動操作某些資源。

小結

這篇文章我們介紹了 AWS 中的基本概念，包括：

- Region 和 Availability Zone 的差別
- 服務根據作用範圍可以分為 AZ 級、Region 級與 Global 級
- 常見的四種管理 AWS 的方法

理解這些概念之後，會對後續的服務操作更容易上手，也比較知道為什麼某些設定不能跨 AZ 或跨 Region。

那麼下一篇文章，我們就會進一步來介紹，在 AWS 中的帳號與 Project 結構（包含 Account、Organization、OU、Tagging 等觀念），幫助你理解大型團隊怎麼管理 AWS 資源。

我們下篇見啦～□

【AWS】Day 3：AWS 中的帳號與資源管理架構

上一篇我們介紹了 AWS 的基本概念，包括什麼是 Region、Availability Zone，以及怎麼操作 AWS 的幾種方式。

那這篇文章，我們要來聊聊一個很多人初學 AWS 時會感到困惑的問題：**AWS 裡面的資源，到底是怎麼被組織與管理的？**

為什麼要有架構？

當你只是個人開一個帳號、開幾台機器、弄幾個服務，帳號的資源大致上不會太複雜，console 點一點都還可以處理。

但當你是企業使用者、跨部門團隊、有多個環境（dev / uat / prod）、多個專案、甚至上百個服務時，就會遇到一堆問題：

- 權限怎麼分？
- 帳單怎麼分？
- 誰能看？誰能用？
- 怎麼防止誤刪 / 誤操作？

這時候就會用到 AWS 提供的資源管理架構囉～

AWS 的三層資源管理架構

在 GCP 中，我們常會提到 **Project** 的概念。但在 AWS 裡面，這樣的觀念其實是被拆成以下幾層來實現的：

□ 1. Account（AWS 帳號）

AWS 帳號是最基本的管理單位。每個帳號就是一個獨立的資源空間，擁有自己的：

- IAM 設定（權限、角色）
- 資源與服務（EC2、S3、RDS 等等）
- 帳單與計費紀錄

可以想像成一個帳號就是一間房間，裡面什麼資源你都自己負責。

□ 很多企業會用多帳號來隔離環境，例如：

- 一個帳號放開發環境（dev）
- 一個帳號放測試環境（uat）
- 一個帳號放生產環境（prod）

這樣的好處是資源更隔離、權限更清楚、帳單也更容易拆分。

□ 2. AWS Organizations（組織）

AWS Organizations 是一個多帳號的管理工具，讓你可以統一控管多個 AWS 帳號，並集中管理費用與權限。

使用 Organizations 後，可以做到：

- 建立帳號之間的層級架構（像是 OU = 組織單位）
- 統一帳單（Consolidated Billing）
- 實作 SCP（Service Control Policies）來限制某些帳號能用的服務

□ 這就像是你開了一間公司，然後底下開了一堆子公司，每個子公司有自己的帳號，但你這個總部可以看所有分公司的帳單，也能下政策規定大家不能亂搞。

□ 3. Resource Tag（資源標籤）

這個就是 AWS 用來補足「Project」概念的做法。

因為 AWS 沒有像 GCP 的 Project 架構，所以會建議大家在建立資源時，養成加上標籤（Tag）的習慣！

常見的標籤包含：

- Project: live-api、momo-shop、data-pipeline
- Env: dev、uat、prod
- Owner: treeman、team-data、ops-team
- CostCenter: 90123、IT-Backend、Marketing

透過 Tag：

- 可以更清楚知道這台機器是做什麼的
- 可以用在 成本分析（Cost Explorer）
- 可以配合 IAM 權限做限制（例如只能操作特定 Project 的資源）
- 還能透過工具（如 Config、Budgets）追蹤管理

小結

這篇文章我們介紹了 AWS 的資源管理架構，總結如下：



概念名稱	對應用途
Account	資源與權限的邊界、計費單位
Organizations	多帳號統一管理架構
Tag	資源標記，模擬 Project 分類

其實 AWS 的架構雖然比 GCP 稍微複雜一點，但也因為彈性大，更能符合企業或大型專案的需求。

那麼下一篇文章，我們就正式進入 **AWS 的運算服務介紹** 啦！包括 EC2、Lambda、ECS、EKS、App Runner.....每一種服務適合什麼情境、優缺點是什麼，我們下一篇文章再慢慢拆解！

我們 Day 4 再見啦 ☺

【AWS】Day 4：AWS 運算服務簡介

Day 4：AWS 運算服務簡介

在上一篇我們認識了 AWS 中的資源管理架構，包括 Account、Organizations、以及 Tag 的概念。

接下來我們要進入一個非常重要的主題：**AWS 的運算服務 (Compute Services)**

所謂「運算服務」，指的就是你在雲端上 **執行應用程式、後端系統、作業流程** 的基礎設施，也就是「跑程式碼的地方」。

AWS 運算服務的類型有哪些？

AWS 提供非常多樣化的運算服務，為了對應不同的使用情境與技術需求，主要可以分成以下幾大類：



分類	說明	代表服務
IaaS	自己管理作業系統、網路、防火牆	EC2 (虛擬機)
PaaS	自動幫你處理 OS、佈署、監控等事情	Elastic Beanstalk
FaaS	無伺服器架構，只有 Function 和事件觸發	Lambda
Container	使用容器技術來執行應用程式，彈性、可擴展	ECS、EKS、App Runner

為什麼 AWS 有這麼多種運算方式？

因為每個團隊的需求都不同：

- 有些團隊希望像過去一樣，有主機、有 OS 可以自己裝軟體 (像 EC2)
- 有些人只想部署應用，不想管基礎架構 (像 App Runner、Elastic Beanstalk)
- 有些系統用事件驅動設計，希望快速反應且省錢 (像 Lambda)
- 有些公司習慣使用 Docker，需要容器編排平台 (像 ECS、EKS)

各服務適合的情境簡介

EC2 (Amazon Elastic Compute Cloud)

- 適合需要高度自訂的作業系統、網路、軟體安裝情境
- 就像租一台雲端主機，從頭開始建系統
- 最自由，但也最需要自己管理

Elastic Beanstalk

- 適合快速部署 Web 應用 (Java、Python、Node.js...)
- 幫你處理 load balancer、autoscaling、OS patching
- 很適合快速上線產品 MVP，但自訂空間有限

Lambda

- 適合小型服務、事件驅動邏輯 (例如上傳圖片後自動轉換)
- 沒有伺服器的概念，你只要寫 function，AWS 幫你跑
- 非常適合做 event-driven 架構、backend for frontend

ECS / EKS / App Runner

- ECS：適合熟悉容器的團隊，整合 AWS 生態系好
- EKS：Kubernetes 愛好者的最愛，管理成本略高
- App Runner：讓你用最少設定就能跑 Docker 應用，近似 Heroku 使用體驗

如何選擇？

以下是簡單的選擇指南：



使用情境	推薦服務
自己掌控一切、用久習慣了	EC2
想快速部署、不管底層	Elastic Beanstalk / App Runner
要做簡單的資料處理任務	Lambda
已有 Docker 經驗的團隊	ECS / EKS / App Runner
想玩 Kubernetes	EKS

小結

這篇文章我們簡單認識了 AWS 中的運算服務全貌，理解了不同服務的設計思路與適用場景：

- EC2 是最基礎的運算單位
- Lambda 是最極致的 serverless
- Elastic Beanstalk 與 App Runner 是簡化部署流程的幫手
- ECS / EKS 則是容器化架構的核心

下一篇我們將正式進入第一個 AWS 運算服務的詳細介紹 —— **EC2（虛擬主機）**，看看這個最經典、最通用的服務到底可以怎麼用。

Day 5 再見！☺

【AWS】Day0 AWS 系列文：從實務出發的雲端探索之旅（共 30 篇）

1. AWS 簡介

- Day 1：認識 AWS (Amazon Web Services)
 - Day 2：AWS 的基本知識與優勢
 - Day 3：AWS 中的帳號、Organization 與 Project 架構 (Account、OU、Tag)
-

2. AWS 中的運算服務

- Day 4：AWS 運算服務總覽
 - Day 5：IaaS, PaaS, FaaS, SaaS 在 AWS 的對應與比較
 - Day 6：EC2 (虛擬機服務)
 - Day 7：Elastic Beanstalk (簡化部署平台)
 - Day 8：Lambda (Serverless 函式)
 - Day 9：EKS (Elastic Kubernetes Service)
 - Day 10：AWS App Runner & ECS (Container 運行環境)
 - Day 11：AWS 運算服務總結與選擇建議
-

3. AWS 的數據儲存服務

- Day 12：AWS 儲存服務總覽
 - Day 13：S3 (物件儲存服務)
 - Day 14：RDS (關聯式資料庫)
 - Day 15：Aurora (高效能資料庫引擎)
 - Day 16：DynamoDB (NoSQL 資料庫)
 - Day 17：Amazon Redshift & DocumentDB & ElastiCache
 - Day 18：AWS 儲存服務總結與應用情境比較
-

4. AWS 的 API 管理服務

- Day 19：API 管理與整合概觀
 - Day 20：API Gateway (全面 API 管理服務)
 - Day 21：AppSync (GraphQL 服務)
 - Day 22：Amazon CloudFront + Lambda@Edge (API 加速與轉換)
 - Day 23：API 管理服務總結與實戰應用
-

5. AWS 的 Message Queue 與事件服務

- Day 24：SQS、SNS、EventBridge 介紹與比較
-

6. AWS 的權限與身份管理

- Day 25：AWS 權限系統概觀
 - Day 26：IAM (身份與角色)
 - Day 27：AWS Cognito & IAM Identity Center (登入與使用者管理)
 - Day 28：IAM 實作範例與最佳實踐
-

7. AWS 的帳單與費用控制

- Day 29：AWS 成本結構、帳單管理與費用控管 (含 Cost Explorer、Budgets)
-

8. AWS 總結與應用建議

- Day 30：AWS 總結、常見架構與選型建議

【AWS】Day 5：IaaS、PaaS、FaaS、SaaS 的差別

在進入 AWS 各項運算服務前，我們先來搞懂幾個在雲端世界中常見的術語：
IaaS、PaaS、FaaS、SaaS。

這些都是「服務模型（Service Model）」的分類，代表你在使用雲端時，需要負責的事情有多少、而雲端服務商（像 AWS）又幫你負責了多少。

雲端服務模型的基本觀念

以下這張表可以幫助你快速了解四種模型的差別：



管理對象/模型	自架主機	IaaS	PaaS	FaaS	SaaS
硬體設備	☐ 自己買	☐ AWS 提供	☐ AWS 提供	☐ AWS 提供	☐ AWS 提供
作業系統	☐ 自己裝	☐ 自己裝/管理	☐ AWS 管理	☐ AWS 管理	☐ AWS 管理
中介軟體 (DB、Runtime)	☐ 自建	☐ 自建	☐ AWS 幫你配好	☐ AWS 幫你配好	☐ AWS 幫你配好
應用程式邏輯	☐ 自己寫	☐ 自己寫	☐ 自己寫	☐ 自己寫	☐ 你不需要寫
維運、監控	☐ 全部自己	☐ 自己負責	△ 半自動	☐ AWS 自動處理	☐ AWS 處理

各種服務模型的說明與範例

1☐ IaaS: Infrastructure as a Service (基礎設施即服務)

- 你租用虛擬機與網路，但其他都自己來
- 最大彈性、也最麻煩，需要自己維運
- 常見服務：**Amazon EC2、VPC、EBS、ELB**

適合對象：

- 有自己建系統的需求
- 對運維有經驗、需高度自訂環境
- 例：自行部署 Apache + PHP + MySQL 的網站架構

2☐ PaaS: Platform as a Service (平台即服務)

- AWS 幫你準備好 OS、環境、runtime，只要上傳應用程式即可
- 不用管架設，但彈性較小
- 常見服務：**Elastic Beanstalk、App Runner**

適合對象：

- 想快速上線服務，不想煩基礎建設細節
- MVP、新創、團隊快速開發部署階段

3☐ FaaS: Function as a Service (函數即服務)

- 只負責一段段程式碼（function），不需要啟動伺服器

- 事件觸發執行、自動擴展、極度省錢
- 常見服務：**AWS Lambda**

適合對象：

- 事件驅動架構、簡單自動任務（如圖片轉換、排程任務）
- 想實作 Serverless 架構

4☐ SaaS: Software as a Service (軟體即服務)

- 使用者什麼都不用管，只要「使用」就好
- 全部由供應商管理、維運、升級
- 常見服務：**Amazon WorkMail**、**QuickSight**、**Google Workspace**、**Slack**、**Notion**、**Zoom**

適合對象：

- 只想「用服務」，不想管任何底層系統
- 想要穩定、成熟的產品解決問題

AWS 各服務分類對照表



模型	AWS 常見服務
IaaS	EC2、EBS、VPC、ELB
PaaS	Elastic Beanstalk、App Runner
FaaS	AWS Lambda、Step Functions
SaaS	WorkMail、QuickSight、Chime、Honeycode

小結

透過這篇文章，我們理解了 IaaS、PaaS、FaaS、SaaS 的差異，並知道：

- 不同模型代表「管理責任」的差異
- 彈性 vs 管理負擔 是互相拉扯的
- 在 AWS 裡幾乎各種模型的服務都有對應可選

下一篇開始，我們就會進入實際的運算服務操作介紹 —— 先從最核心、最經典的 **EC2 (Elastic Compute Cloud) 虛擬機服務** 開始探索！

我們 Day 6 再見～

【AWS】Day 6：運算服務 - Amazon EC2（虛擬機服務）

在前幾篇中，我們介紹了各種運算服務的分類方式，像是 IaaS、PaaS、FaaS 等等。

今天就從 AWS 中最經典、最基礎的 IaaS 服務開始介紹 —— **Amazon EC2 (Elastic Compute Cloud)**

這個服務幾乎是所有 AWS 使用者早期接觸雲端的第一站，也是許多架構的基石。

什麼是 EC2？

EC2 就是一台 **雲端的虛擬機器 (Virtual Machine)**。

可以想像成你在資料中心租了一台電腦，你可以選擇要用 Linux 還是 Windows，要幾核心、幾 GB 記憶體，要多大的硬碟，要不要固定 IP.....這些你都可以自己決定。

建立 EC2 之後，就能連進去安裝軟體、跑應用、開 Web Server、跑排程、架設資料庫.....什麼都行。

EC2 的核心概念

1. Instance (執行個體)

每一台 EC2 就是一個 Instance，你可以開很多個、關掉它、重新啟動，甚至做 snapshot 備份。

2. AMI (Amazon Machine Image)

AMI 是一種映像檔，可以理解成 EC2 的「作業系統+預設環境」模板。
常見的 AMI 包含：

- Amazon Linux
- Ubuntu
- Debian
- Windows Server
- 自己製作的 AMI

3. Instance Type (機型)

AWS 提供各種 instance type 給你選擇，依照用途分類，例如：

類型代號	用途	代表型號
t 系列	一般型、低成本	t3.micro、t4g.nano
m 系列	平衡型	m6i.large
c 系列	計算密集型	c7g.medium
r 系列	記憶體密集型	r6g.large
g 系列	GPU 運算型	g5.xlarge

建立 EC2 的流程簡介

1. **選擇 AMI**：決定你要用哪一種作業系統
2. **選擇 Instance Type**：例如 t3.micro (免費方案支援)
3. **設定 Key Pair**：建立 SSH 金鑰，用來連線 EC2
4. **設定 Network (VPC) 與 Security Group (防火牆)**

5. 設定磁碟 (EBS) 大小
6. 啟動 EC2 !

完成後你會拿到一組 Public IP，就可以用 SSH 連進去使用了。

價格與計費方式

EC2 的價格組成會根據幾個項目決定：

項目	說明
Instance Type	不同型號每小時價格不同
計費模式	On-Demand / Spot / Reserved
使用時間	每秒或每小時計費
是否有附加儲存	EBS 空間也另外計費
傳輸流量	Outbound (出站) 會收費

EC2 的使用情境

EC2 適合用在哪些情況呢？以下列舉幾個常見案例：

- 需要完整控制環境與作業系統
- 要安裝自訂軟體或中介軟體 (middleware)
- 部署高效能或特殊架構的應用 (如容器主機、資料庫伺服器)
- 架設 CI/CD 服務、自架 Git、Redis、RabbitMQ
- 建立可手動調整、監控的後台系統

小結

EC2 是 AWS 中最基礎的運算服務，具備極大的彈性與控制權：

- 可以自由選擇作業系統與機型
- 適合習慣傳統主機操作的團隊
- 計費彈性、整合其他 AWS 服務方便

EC2 進階篇：實戰操作與進階應用

在上一篇我們介紹了 EC2 的基本觀念與用途，這篇會帶大家進一步實作：

- 如何登入 EC2 ?
- 如何開放 port (像是 80、443) 讓外部可以訪問 ?
- 怎麼設定開機自動執行程式 ?
- EC2 如何實作自動擴展 (Auto Scaling) ?

一、登入 EC2 實例 (Linux)

當你啟動好一台 EC2 Linux 實例後，登入方式如下：

□ 準備工作

- 確保當初有建立 Key Pair，下載了 `.pem` 檔案
- 開啟 Security Group 的 TCP port 22 (SSH)
- 有 Public IP 或 Public DNS (可在 EC2 頁面查看)

❑ 登入指令

```
chmod 400 my-key.pem
ssh -i my-key.pem ec2-user@<你的 Public IP>
```

“不同 AMI 登入帳號不同：

- Amazon Linux 用 `ec2-user`
- Ubuntu 用 `ubuntu`
- Debian 用 `admin` 或 `debian`

二、開放 HTTP / HTTPS 連線

如果你要讓外部可以訪問你的 EC2，例如跑 Web server，需要設定 **Security Group**。

❑ 開啟常見 port：

- HTTP：80
- HTTPS：443
- 自定服務：你用什麼就開什麼（例如 3000、8080）

操作步驟：

1. 到 EC2 頁面點選「Security Group」
2. 找到你 EC2 所使用的 security group
3. 編輯「Inbound rules」
4. 新增：
 - 類型：HTTP、HTTPS
 - 來源：0.0.0.0/0（所有人都可以連）或限制特定 IP

三、設定開機後自動執行程式（User Data）

你可以在建立 EC2 時加入 **User Data**，當機器第一次啟動時會自動執行這段腳本。

範例：自動安裝 Nginx 並啟動 Web Server

```
#!/bin/bash
yum update -y
yum install -y nginx
systemctl enable nginx
systemctl start nginx
echo "Hello from EC2!" > /usr/share/nginx/html/index.html
```

“❑ 注意：User Data 只能在 EC2 **首次啟動時執行一次**。

❑ 如果想修改已存在的 User Data，需要重新建立新的 EC2 或自己寫開機指令。

四、設定 Auto Scaling Group（自動擴展）

Auto Scaling 是 EC2 強大的特點之一，可以根據負載自動增減 EC2 數量。

使用場景：

- 使用者流量增加，自動開新機器
- 流量低時自動縮減節省成本
- 保證服務有固定最少機器數（High Availability）

建立流程簡略：

1. 建立一個 Launch Template（定義要開什麼樣的 EC2）
 2. 設定 Auto Scaling Group：
 - 定義最小 / 最大 EC2 數量
 - 設定 Scaling Policy（例如 CPU > 60% 時新增 1 台）
 3. 可結合 Load Balancer，讓所有流量平均分配到所有 EC2 上
-

五、備份與還原

EC2 本身不會自動備份，要使用以下方式保護資料：

快照（Snapshot）

- 可對 EBS 磁碟建立快照
- 快照可以還原成新的 EBS 磁碟掛回 EC2

建立 AMI

- 對目前 EC2 建立 AMI 映像檔
 - 可隨時用這個 AMI 快速建立新機器（帶所有設定）
-

小結

這篇進階實作帶你實際操作 EC2，包括：

- 使用 SSH 登入實例
- 開放外部連線 port
- 利用 User Data 自動化設定
- 設定 Auto Scaling Group 自動擴展
- 建立快照與 AMI 做備份

EC2 的強大在於它的彈性與可控性，從個人專案到大型分散式系統，都能發揮作用。

下一篇，我們將來看看如果你不想自己處理作業系統、網路設定、磁碟掛載這些瑣事時，AWS 有沒有更輕鬆的方案？
Day 7：Elastic Beanstalk - 全自動部署的神隊友！，我們不見不散！

【AWS】Day 7：運算服務 - Elastic Beanstalk（自動化部署服務）

在前幾篇，我們介紹了 EC2，了解了怎麼自己開一台雲端主機、自己設定環境、自己管防火牆、自己擴展、自己備份。

但如果你覺得自己管這些基礎建設很麻煩，希望可以更專注在「寫程式」跟「部署應用程式」上面，那 AWS 其實也有更方便的選擇：今天要介紹的就是 —— **Elastic Beanstalk**

什麼是 Elastic Beanstalk？

Elastic Beanstalk 是 AWS 提供的一種 **PaaS（平台即服務）**。

它的目標很簡單：

- 你只要準備好應用程式
- 把應用程式丟給 Beanstalk
- Beanstalk 幫你處理所有底層繁瑣的事情

包括：

- 建好 EC2 instance
- 建好負載平衡器 (Load Balancer)
- 建好 Auto Scaling
- 建好 Security Group、防火牆設定
- 幫你佈署、監控、收集日誌
- 還可以一鍵回滾版本

你專注在開發跟部署，AWS 幫你顧好其他基礎設施。

支援的應用類型

Elastic Beanstalk 支援超多種語言與平台：

- Node.js
- Python
- Java
- Ruby
- PHP
- .NET
- Go
- Docker（完整支援 Container）
- 靜態網站 (HTML/CSS/JS)

如果你是 Web 應用開發者，基本上都可以直接使用 Beanstalk 部署。

Elastic Beanstalk 的架構概念

雖然 Beanstalk 幫你管理底層資源，但其實背後還是會幫你建立 AWS 其他服務，包括：

- EC2（跑你的應用）
- Elastic Load Balancer（分流）
- Auto Scaling Group（自動擴展）
- CloudWatch（監控、日誌）
- RDS（如果有需要資料庫）

這些資源都是 Beanstalk 幫你「托管」，你可以選擇要不要深入調整。

“你仍然可以進入 EC2 頁面看到自己應用用到的實例，但建議不要直接手動修改，避免破壞 Beanstalk 的管理邏輯。

建立 Elastic Beanstalk 應用的流程

1. 進入 AWS Console，打開 Elastic Beanstalk
2. 點選「Create Application」
3. 填入：
 - Application Name（應用名稱）
 - Platform（選語言與環境）
 - Upload your code（上傳程式碼壓縮檔）
4. 設定部署設定（可以預設）
5. 點下「Create Application」

幾分鐘後，AWS 就會幫你建好整個環境，還會提供一個 URL，直接可以訪問你的應用！

部署更新與版本管理

Elastic Beanstalk 很貼心地內建了「版本管理」功能：

- 每次部署新程式碼，都會建立一個新的版本
- 可以快速回滾（Rollback）到前一個版本
- 可以在不同環境（例如 dev、prod）快速切換程式版本

常見 Elastic Beanstalk 的使用情境

- 快速啟動 MVP、小型專案
- 開發階段測試不同版本部署
- 中小型團隊的 Web 應用架設
- 需要簡單 Auto Scaling、Load Balancing 但不想自己架設

注意事項

- **不是完全免維護**：雖然簡化很多，但還是要稍微理解 VPC、Security Group 等 AWS 基本概念
- **資源會產生費用**：Beanstalk 本身不收費，但底層建立的 EC2、RDS、S3 等都會收費
- **適合標準架構**：如果應用有非常特殊的環境需求（例如自訂作業系統 kernel）可能還是要回去用 EC2 自己建

小結

Elastic Beanstalk 是 AWS 上手最簡單的運算服務之一：

- 幫你管理底層基礎建設
- 讓你快速部署、快速擴展、快速回滾
- 適合中小型 Web 應用，也適合剛開始學 AWS 的人嘗試

當然可以！這裡直接幫你補上

Elastic Beanstalk 簡易實作教學（以 Node.js 舉例），延續前面文章的語氣與結構，實作內容也保持簡單明瞭，讓讀者可以直接跟著做。

Elastic Beanstalk 簡易實作教學（以

Node.js 範例)

剛剛介紹了 Elastic Beanstalk 是什麼，接下來就實際帶大家操作一次：

- 我們會快速做出一個 Node.js 小應用
- 壓成 ZIP 檔
- 部署到 Elastic Beanstalk 上

目標是：**10 分鐘內完成一個雲端 Web App !**

1. 建立一個簡單的 Node.js 專案

首先，在你的電腦上開一個新資料夾，例如 `eb-demo-app`，然後建立最基本的 Node.js 應用。

```
mkdir eb-demo-app
cd eb-demo-app
npm init -y
npm install express
```

接著建立一個 `app.js`，內容如下：

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('Hello from Elastic Beanstalk!');
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

這樣就完成了最基本的 Node.js Web 應用。

注意：這裡我們用 `process.env.PORT` 是 Elastic Beanstalk 的部署要求，讓它自動分配正確的 port。

2. 建立必要的設定檔

Elastic Beanstalk 在 Node.js 環境中，需要有一個叫做 `package.json` 的設定檔（剛剛 `npm init` 已經自動生成），還需要明確標出啟動指令。

請確認 `package.json` 中有這一段：

```
"scripts": {
  "start": "node app.js"
}
```

如果沒有，手動加上去，這樣 Elastic Beanstalk 才知道要怎麼啟動你的應用程式。

3. 壓縮成 ZIP 檔案

回到 `eb-demo-app` 目錄，將所有檔案壓縮成一個 `.zip` 檔。

注意：**不要把整個資料夾壓縮進去！**

- ZIP 檔裡面應該是直接包含 `app.js`、`package.json`，不是包一層資料夾！

例如正確的內容：

```
app.js
package.json
node_modules/
```

如果你想省空間，其實可以只壓 `app.js` 和 `package.json`，讓 Elastic Beanstalk 自己在雲端跑 `npm install`。

4. 登入 AWS Elastic Beanstalk 控制台

打開 [Elastic Beanstalk 主控台](#)

1. 點選「Create Application」
2. 填上：
 - **Application name**：例如 `demo-node-app`
 - **Platform**：選 `Node.js`
 - **Application code**：選擇「Upload your code」上傳剛剛的 `.zip` 檔
3. 點選「Create Application」

接下來 AWS 會自動：

- 建 EC2
- 建 Load Balancer
- 建 Auto Scaling Group
- 佈署你的應用程式

大概 3-5 分鐘後，就可以從 Elastic Beanstalk 頁面拿到一個網址，直接訪問你的應用！

5. 驗證部署結果

打開 Elastic Beanstalk 頁面給你的 URL，比如：

```
http://demo-node-app.ap-northeast-1.elasticbeanstalk.com/
```

你應該會看到網頁上寫著：

```
Hello from Elastic Beanstalk!
```

部署成功 ☑！

小結

這就是最簡單的 Elastic Beanstalk 實作流程，總結一下步驟：

1. 建立應用 (Node.js / Python / etc)
2. 加好 `start` 指令
3. 壓成 ZIP 檔
4. 上傳到 Elastic Beanstalk
5. 幾分鐘內自動建立並上線

對於想快速上線 MVP 或測試環境的人來說，Elastic Beanstalk 真的是一個超省力的好選擇！

沒問題！這裡幫你補上

Elastic Beanstalk 簡易實作教學 (Python / Flask 範例)
延續剛剛 Node.js 範例的風格和節奏，一樣清楚、快速能跟著做！

Elastic Beanstalk 簡易實作教學（以

Python / Flask 範例)

如果你是使用 **Python** 的開發者，想要快速把 Flask App 部署到 Elastic Beanstalk，也是非常簡單的！

這篇教學會帶你從零建立一個最簡單的 Flask App，並且成功部署到 Elastic Beanstalk 上運行。

1. 建立一個 Flask 專案

首先，在你的電腦上新建一個資料夾，例如 `eb-python-demo`：

```
mkdir eb-python-demo
cd eb-python-demo
python3 -m venv venv
source venv/bin/activate # Windows 用 venv\Scripts\activate
pip install flask
```

接著新增一個 `application.py`（注意：檔名是 `application.py`，Elastic Beanstalk 預設會找這個名字）

```
from flask import Flask

application = Flask(__name__)

@application.route("/")
def hello():
    return "Hello from Elastic Beanstalk with Python!"

if __name__ == "__main__":
    application.run(host='0.0.0.0', port=5000)
```

這樣就完成一個超簡單的 Flask Web App！

2. 建立 requirements.txt

Elastic Beanstalk 需要知道要安裝哪些套件，所以要建立一個 `requirements.txt` 檔案，內容如下：

```
Flask
```

（未來如果有更多套件，就加在這個檔案裡。）

指令快速產生：

```
pip freeze > requirements.txt
```

3. 建立 `.ebextensions` 設定檔（可選）

如果你想在 Elastic Beanstalk 上做更多進階設定（例如自動安裝系統套件），可以用 `.ebextensions`。不過這次我們簡單部署，這步可以先跳過。

4. 壓縮成 ZIP 檔案

跟剛剛 Node.js 一樣，把你的檔案壓縮成一個 `.zip` 檔案。

⚠ 注意：

- ZIP 裡面應該包含：`application.py`、`requirements.txt`
- 不要壓整個資料夾進去

正確的結構是：

```
application.py
requirements.txt
```

5. 部署到 Elastic Beanstalk

打開 [AWS Elastic Beanstalk Console](#)

操作步驟：

1. 點選「Create Application」
2. 填寫：
 - **Application name**：例如 `demo-python-app`
 - **Platform**：選 **Python**
 - **Platform Branch**：例如 Python 3.11
3. 上傳剛剛壓好的 `.zip` 檔案
4. 點選「Create Application」

Elastic Beanstalk 會自動：

- 幫你建立 EC2
- 幫你佈署 Flask
- 幫你設定 Load Balancer

大約 3-5 分鐘部署完成。

6. 驗證部署結果

部署好之後，Elastic Beanstalk 會給你一個 URL，比如：

```
http://demo-python-app.ap-northeast-1.elasticbeanstalk.com/
```

打開之後應該會看到：

```
Hello from Elastic Beanstalk with Python!
```

大成功！☑

小結

Python / Flask 的 Elastic Beanstalk 快速部署流程：

1. 建好 Flask 應用（注意 `application.py`）
2. 建立 `requirements.txt`
3. 打包成 `.zip`
4. 上傳到 Elastic Beanstalk
5. 等待幾分鐘，直接有一個可以訪問的 Web App

Elastic Beanstalk 進階教學

（環境變數、eb CLI 部署、自動串接 RDS）

一、設定環境變數 (Environment Variables)

在實際應用中，很常需要讓應用程式讀取一些敏感資訊（像資料庫連線字串、API 金鑰等等），這些不適合硬寫在程式碼裡。這時就需要使用「環境變數」。

□ 在 Elastic Beanstalk 上設定環境變數

操作步驟：

1. 打開 Elastic Beanstalk Console，進入你的應用 Environment
2. 左邊選單找到「Configuration」
3. 找到「Software」，按下「Edit」
4. 在 Environment properties 區塊新增你的變數，例如：

Key	Value
DB_HOST	mydb.xxxxxxxx.rds.amazonaws.com
DB_USER	admin
DB_PASSWORD	supersecure123

儲存後，Elastic Beanstalk 會自動重啟環境，讓你的應用可以透過環境變數讀取設定。

□ 在程式中讀取環境變數 (Python Flask 範例)

```
import os

db_host = os.getenv('DB_HOST')
db_user = os.getenv('DB_USER')
db_password = os.getenv('DB_PASSWORD')
```

這樣就能安全地使用環境設定，不用把敏感資訊寫死在程式碼裡！

二、使用 eb CLI 進行快速部署

如果你覺得每次都開 Console 點來點去很煩，AWS 提供了專門給 Elastic Beanstalk 用的指令工具：**Elastic Beanstalk CLI (eb CLI)**

可以直接從 Terminal 完成初始化、部署、查看環境狀態等等，非常方便！

□ 安裝 eb CLI

先安裝 AWS EB CLI：

```
pip install awsebcli --upgrade
```

“ 有時候 MacOS 或 Linux 需要加上 `--user`，或用 `virtualenv` 安裝。

安裝好後，試試：

```
eb --version
```

如果能正常顯示版本號，代表安裝成功。

☐ eb CLI 操作基本流程

1. 登入 AWS（要先設定好 AWS CLI credentials）

```
aws configure
```

2. 初始化 eb 專案

在你的應用資料夾內：

```
eb init
```

過程中會問你：

- 要使用哪個 Region？
- 是哪個 Platform？（Node.js、Python 等）
- 是否需要設定 SSH？

3. 建立 Environment

```
eb create dev-env
```

（可以自己取環境名字）

4. 部署更新

```
eb deploy
```

5. 查看環境資訊

```
eb status
```

6. 開啟瀏覽器查看應用

```
eb open
```

超級方便，可以完全不開網頁直接部署更新！

三、自動串接 RDS 資料庫（一起建立）

如果你的應用需要資料庫（例如 MySQL、PostgreSQL），Elastic Beanstalk 可以在建立 Environment 時順便幫你建立 RDS！

☐ 如何設定

1. 建立 Environment 時，選擇「Configure more options」
2. 找到「Database」選項，點選「Edit」
3. 選擇：
 - 資料庫引擎（MySQL、PostgreSQL）
 - 資料庫版本
 - 使用者名稱、密碼
 - 硬碟空間大小

建立好後：

- Elastic Beanstalk 會自動把 RDS 與你的應用放在同一個 VPC、Security Group
- 會自動把資料庫連線資訊存到環境變數（`RDS_HOSTNAME`、`RDS_USERNAME`、`RDS_PASSWORD`、`RDS_PORT`）

你的 Flask / Node.js 應用只要讀取這些環境變數，就能連上資料庫！

☐ Flask 範例（連接 Elastic Beanstalk RDS）

```
import os
import pymysql
```

```
db_host = os.getenv('RDS_HOSTNAME')
db_user = os.getenv('RDS_USERNAME')
db_password = os.getenv('RDS_PASSWORD')
db_name = os.getenv('RDS_DB_NAME')

conn = pymysql.connect(
    host=db_host,
    user=db_user,
    password=db_password,
    database=db_name
)
```

就可以在應用程式中直接操作資料庫啦！

小結

Elastic Beanstalk 除了快速部署應用，進階玩法還可以：

- 設定環境變數，安全管理敏感資訊
- 使用 eb CLI，極速 Terminal 部署
- 一起建立 RDS，讓應用直接串接雲端資料庫

Elastic Beanstalk 常見錯誤排除指南

Elastic Beanstalk 幫我們自動化了很多事，但實際使用過程中還是很容易遇到一些常見錯誤。

這篇帮大家整理了：

- 常見的 Elastic Beanstalk 部署問題
- 原因說明
- 解決方法

給你一份遇到問題時能快速排雷的工具包！

1. 部署後出現 502 Bad Gateway

問題現象

- 部署成功，但一打開應用的網址就跳出 502 Bad Gateway
- 或者頁面載入超級慢然後 timeout

常見原因

- 應用程式沒有正常啟動
- 聽錯了 Port (Elastic Beanstalk 預設會轉送到 80 或由 proxy pass)

解決方法

☐ 檢查你的應用程式是否有監聽正確的 Port：

- Node.js / Flask / Django 等應用，要監聽環境變數給定的 port
- 例如 Flask：

python

複製

編輯

```
application.run(host='0.0.0.0', port=int(os.environ.get('PORT', 5000)))
```

☐ 查看 Logs：

- Elastic Beanstalk Console → Logs → Request logs → Last 100 lines
- 查看應用啟動過程是否有錯誤訊息，例如 module not found、port 綁定錯誤等

2. 部署失敗，顯示「Instance deployment failed」

問題現象

- 上傳 ZIP 包後，環境 Health 直接變紅色
- console 顯示 deployment failed

常見原因

- 上傳的 ZIP 包結構錯誤
- 必要檔案（像 application.py 或 package.json）缺少或錯誤
- requirements.txt 有誤（例如版本衝突、缺失）

解決方法

☐ 確認 ZIP 檔內部是正確的結構：



☐ 如果是 Python，確保有 requirements.txt，且內容正確。

☐ 部署失敗可以看 Instance logs：

- Console → Logs → Instance logs → 全部下載下來
- 查看 /var/log/eb-engine.log 或 /var/log/web.stdout.log 裡的錯誤訊息

3. 無法 SSH 連線到 EC2

問題現象

- 按下「Connect」或者手動 SSH 連線失敗
- 卡在 timeout，無法進去 Instance

常見原因

- Security Group 沒有開放 port 22
- 沒有設定 Key Pair
- 錯誤使用 Public IP 或 DNS 名稱

解決方法

☐ 確認 Environment → Configuration → Security → EC2 Key Pair 有設定。

☐ Security Group 要開 TCP 22 port，允許你的 IP 連進去。

☐ 連線指令正確，例如：

```
bash
```


4. 資源遺留：環境刪掉但 EC2、RDS 還在

問題現象

- 刪除 Elastic Beanstalk 環境後
- 發現 EC2、RDS、Security Group、EBS Volume 還殘留在帳號中

常見原因

- 預設刪除環境時，只會刪除 Beanstalk 直接管理的資源
- 外掛式 RDS、手動建立的資源不會跟著刪

解決方法

□ 手動到 EC2、RDS、VPC 頁面，檢查並刪除殘留的資源。

□ 未來建立環境時，如果有開 RDS，選「**內嵌式 RDS**」（跟環境生命週期綁在一起），才會自動清掉。

5. 更新應用程式後，環境變紅，回不去

問題現象

- 部署新版本應用後
- Elastic Beanstalk 環境變成 Severe（紅色）
- 想緊急回滾但不知道怎麼做

常見原因

- 新版程式出錯，導致服務無法正常啟動
- Elastic Beanstalk 的 Health Check 判定失敗

解決方法

□ 使用 Elastic Beanstalk 的**版本回滾功能**：

- Console → Application → Versions
- 找到上一個成功部署的版本，按「Deploy」

□ 部署失敗時，通常 Console 會自動記錄上一次成功的狀態，可以快速 Rollback。

6. 頻繁超過資源限制（Instance CPU 爆表、Memory 爆滿）

問題現象

- Elastic Beanstalk 環境 Health 不穩
- 應用載入變慢或直接掛掉

常見原因

- Instance 選太小 (例如 t2.micro 記憶體太小)
- 程式設計有記憶體洩漏 (Memory Leak)
- 同一台機器上負載過高

解決方法

□ 調整 Instance Type :

- 將 EC2 Instance 換成 t3.medium、t3.large 或更高階型號
- Elastic Beanstalk 支援一鍵調整 Scaling 設定

□ 設定 Auto Scaling Policy :

- 當 CPU > 60% 自動擴展一台
- 當 CPU < 30% 自動縮減

□ 用 CloudWatch 監控 CPU、Memory 使用率，設通知 (Alarm)

小結

Elastic Beanstalk 雖然大幅降低了管理負擔，但遇到問題時：

- 善用 **Logs**
- 善用 **Environment Health Status**
- 善用**版本控制回滾機制**

這三個是最快速排除問題的關鍵！

下一篇，我們將進入 AWS 的 **Serverless 運算世界 —— Lambda** !
看看如果連 EC2、Load Balancer 都不想管，只想「寫 function」，AWS 能怎麼幫你做到！

Day 8 再見 ☺

【AWS】【Elastic Beanstalk】常見部署結構推薦指南&費用最佳化建議

Elastic Beanstalk 常見部署結構推薦指南

Elastic Beanstalk 很方便，但如果搭配得好，會讓你的應用又穩又省錢；搭配得不好，可能會出現：

- 資源不足導致當機
- 成本過高
- 擴展緩慢等問題

這篇針對不同情境，推薦最佳部署結構與選型建議。

1. 小型網站 / MVP 初版

適用情境

- 一般公司網站
- 初版產品 Demo
- 內部管理系統
- 低流量 Web API

推薦設定

項目	建議值
Instance Type	t3.micro 或 t3.small (免費額度或低價)
Instance 數量	最小 1 台，最大 1 台 (不開 Auto Scaling)
Database	內嵌式 RDS 或直接外接小型 DB (如 Lightsail)
Load Balancer	不使用 (單台直接對外即可)
部署方式	單一環境，使用 Rolling 更新

原因分析

- 成本極低 (每月不到 10 美金)
- 流量低，單台就夠
- 部署簡單，維運容易
- 適合快速迭代、驗證市場

2. 中型後端服務 / API Server

適用情境

- 中等流量的 Web API
- 手機 App 的後端
- 需要支援突發高峰流量

推薦設定

項目	建議值
Instance Type	<code>t3.medium</code> 或 <code>t3.large</code>
Instance 數量	最小 2 台，最大 4 台（開啟 Auto Scaling）
Database	外接獨立 RDS（例如 db.t3.medium）
Load Balancer	啟用 Application Load Balancer (ALB)
部署方式	Rolling with additional batch（保證不中斷）

原因分析

- 2 台機器可避免單點故障
- Auto Scaling 可以因應突發性流量（例如促銷活動）
- 獨立 RDS 提高資料庫效能與可管理性
- ALB 可支援 HTTPS 與多目標健康檢查

3. 高流量大型網站 / 高併發應用

適用情境

- 電商網站
- 多人即時服務（如聊天室、直播）
- 高 API 併發應用

推薦設定

項目	建議值
Instance Type	<code>m6i.large</code> 或 <code>c6g.large</code> （效能型）
Instance 數量	最小 4 台，最大 10 台（Auto Scaling）
Database	專屬大型 RDS（或 Aurora Serverless）
Load Balancer	Application Load Balancer（或加上 CloudFront）
部署方式	Immutable deployment（零中斷更新）

原因分析

- 使用效能型機器降低單位併發成本
- 多台擴展配合 Auto Scaling 可應付大幅波動
- Immutable 部署方式可避免更新失敗影響現有線上服務
- Load Balancer 可搭配 CDN 提升全球加速

4. Serverless + Beanstalk 混合模式 (Hybrid)

適用情境

- 需要兼顧複雜計算與 Serverless 架構
- 某些任務使用 Lambda 執行（如影像處理、寄信）
- 主系統仍以 Beanstalk EC2 部署

推薦設定

- 核心 API / Web server 放在 Elastic Beanstalk
- 背景工作 (cron job、queue consumer) 用 AWS Lambda + EventBridge
- File upload / Static Content 用 S3 + CloudFront
- Serverless 服務與 EC2 協作，共用 IAM role / VPC

原因分析

- 彈性最佳化：主服務 EC2 高效能、子任務 Lambda 靈活啟動
- 成本控制佳，避免小工作佔用大型主機資源
- 容錯能力提升，Lambda 失敗也不影響主要 API

小結

不同規模、不同需求下，Elastic Beanstalk 最佳部署建議：

規模	Instance建議	Auto Scaling	Load Balancer	Database建議
小型	1台小機器	☐	☐	內嵌 RDS 或外接小型 DB
中型	2-4台中型機器	☐	☐	獨立 RDS
大型	4-10台高效能機器	☐	☐ (加 CDN)	大型 RDS 或 Aurora Serverless
混合	Beanstalk + Lambda	☐	☐	跨服務組合

Elastic Beanstalk 的強大就在於
從個人開發到企業級系統，都可以靈活搭配出適合自己的方案！

Elastic Beanstalk 費用最佳化建議

(含 Auto Scaling 節省技巧)

Elastic Beanstalk 本身不收管理費，真正產生費用的是底層的 EC2、Load Balancer、RDS、S3、傳輸流量等資源。

所以如果想讓 Beanstalk 部署既穩定又省錢，關鍵在於：

- 控制 EC2 成本
- 適當調整擴展策略
- 避免無意義的浪費

這篇就帶你掌握 Elastic Beanstalk 最有效的省錢方法！

1. 使用 Reserved Instance (RI)

☐ 做法

如果你的應用是「穩定長期運行」（例如公司網站、後端 API），建議購買 **Reserved Instances** (RI) 來取代 On-Demand。

- 可省下 **最高 72%** 成本（依付款方案而定）
- 1 年期或 3 年期，分為 All Upfront / Partial Upfront / No Upfront 付款選項

“☐ 注意：RI 是鎖定在「Region + Instance Type family」（例如 t3 family）

☐ 適用時機

- 流量穩定的長期應用
- 預期至少持續使用 1 年以上

2. 配合 Auto Scaling 使用 Spot Instances

□ 做法

Elastic Beanstalk 的 Auto Scaling Group 可以設定混合策略：

- 混合使用 On-Demand 與 Spot Instances
- 例如：
 - 30% On-Demand 保底
 - 70% Spot 便宜又彈性擴展

在環境設定裡選擇：

“Capacity → Auto Scaling → Instances → Purchase Options → "Combine purchase options and instance types"

這樣可以極大降低「額外擴展台數」的費用。

Spot 便宜很多（常常是 On-Demand 價格的 10%~30%），超適合做大量讀取、彈性擴展的情境！

□ 適用時機

- 流量不穩定、會爆量但可以短期容忍失敗（如直播、促銷活動）
- 不要求 100% 可用性的非關鍵服務（如統計、備援服務）

3. 適當設定 Auto Scaling 門檻

Auto Scaling 是節省資源的重要機制，但如果設定不當，反而可能讓你爆量開機器！

□ 推薦設定策略

- **Scaling Up (擴展)**：CPU 使用率 > 60% 才新增機器
- **Scaling Down (縮減)**：CPU 使用率 < 30% 才關閉機器
- **Cooldown Time (冷卻時間)**：
 - 擴展冷卻時間：180 秒
 - 縮減冷卻時間：300 秒

這樣可以避免瞬間小波動就導致無謂的開關機，減少 Auto Scaling 的震盪效應（flapping）。

4. 選擇適合的 Instance Type

不要一開始就開很大的機器，建議從小型機型開始測試：

- 小型 / 中型服務：選 `t3.medium` 或 `t4g.medium`（ARM 架構更便宜）
- 記憶體吃重服務：選 `r6g.large`
- 計算密集服務：選 `c6g.large`

t 系列機型特別便宜，還有 **CPU Credit** 機制，非常適合大部分輕中量應用。

“□ 如果有資格使用 AWS Savings Plan，搭配起來成本又能再下降！

5. 適時關閉閒置資源

Elastic Beanstalk 的環境如果沒有在使用，底層資源（EC2、EBS、ALB）還是會一直產生費用！

📌 建議

- 測試環境（staging / dev）用完記得直接 **terminate**。
- 用 Lifecycle Policy 自動清除超過 30 天未使用的 Application Versions。
- RDS 如果是測試用，也記得關掉（RDS 比 EC2 還貴！）。

6. 用 CloudFront 做流量加速，減少 ALB 費用

如果你的 Elastic Beanstalk 是做靜態檔案 / API Server，流量高峰期可以前面加一層 **CloudFront CDN**，減少直接打到 Load Balancer 的次數。

- 減少 ALB Request 數量 → 降低 ALB 成本
- 加速全球存取速度
- 保護背後應用不被 DDoS 打爆（搭配 AWS Shield）

小結

Elastic Beanstalk 節省費用重點整理：

方法	核心效果
Reserved Instances	穩定運作環境省錢最大化 (>50%)
Spot Instances混搭	彈性負載環境大幅降低成本
正確設 Auto Scaling	避免不必要的機器啟動/關閉
選對 Instance Type	減少過度規格浪費
定期清理閒置資源	減少測試環境與多餘資源費用
加上 CloudFront	減少 ALB 成本與提高存取效率

Elastic Beanstalk 不只是簡單，只要稍微搭配得好，也能做到「低成本、高可靠、高效能」的部署！

【AWS】Day 8：運算服務 - AWS Lambda（Serverless 運算服務）

從前幾篇我們一路看了 EC2（自己管主機）、Elastic Beanstalk（半自動化部署）。

但如果你希望：

- 完全不用開伺服器
- 完全不用自己處理機器、網路、Scaling
- 只要專心「寫功能」就好

那 AWS Lambda 正是為了這樣的世界而生的！

今天正式進入 AWS Serverless 世界 —— **Lambda**！

什麼是 AWS Lambda？

AWS Lambda 是一個「事件驅動」的計算服務。

你只需要撰寫小段的**程式碼 Function**，告訴 AWS：

- 什麼時候要執行（例如 API 呼叫、排程時間、檔案上傳）
- 要執行什麼邏輯（你的程式碼）
- 設定好記憶體、超時時間

剩下的：

- 執行環境（OS、Patch、Security）
- 擴展（Scaling up & down）
- 負載均衡
- 日誌收集
- 資源回收

全部 AWS 幫你自動處理。

你只要負責寫功能，其他都不用煩惱！

Lambda 的基本特性

項目	說明
運算單位	Function（函數）
部署方式	上傳程式碼（ZIP）或直接在 Console 編輯
支援語言	Node.js, Python, Java, Go, .NET, Ruby...
記憶體設定	128MB ~ 10GB（每64MB為單位）
執行時間限制	最長 15 分鐘
價格模式	依執行次數 + 記憶體用量 + 執行時間計價
自動擴展	是（幾乎無上限，橫向自動擴展）

Lambda 的典型使用情境

- API Gateway → Lambda → 回應 Web API
- S3 上傳檔案 → Lambda 處理轉檔（圖片縮圖、影片轉檔）
- DynamoDB 資料異動 → Lambda 處理後續邏輯

- 排程任務（類似 crontab）→ 用 EventBridge 觸發 Lambda
- 輕量型後端服務（例如 Form submit、小型 Game Server）

Lambda 的基本運作流程

以「API Server」為例：

1. 使用 **API Gateway** 建立 API Endpoint
2. 當使用者呼叫 API 時，API Gateway 觸發 Lambda
3. Lambda 執行你的程式邏輯（例如查資料庫、處理資料）
4. Lambda 執行完畢，把結果回傳給 API Gateway，再傳給使用者

整個過程中：

- 沒有 EC2
- 沒有 Load Balancer
- 沒有固定伺服器
- 自動擴展，不需要自己設定台數

Lambda 的計費方式

Lambda 的收費方式非常彈性，也很適合小量或突發性使用的場景。

計費公式：

“總費用 = 呼叫次數 + (記憶體用量 × 執行時間)”

詳細來說：

- 每個月前 1M 次請求免費
- 每個月 400,000 GB-seconds 免費
- 超出後，依照實際用量超便宜的單位收費

簡單說就是：

- 程式小、執行快，超便宜
- 不執行，不收費（真正的 pay-as-you-go）

Lambda 的限制與注意事項

雖然 Lambda 很香，但也有一些天然限制要注意：

限制項目	說明
最大執行時間	15 分鐘
單次部署包大小上限	250MB (含 layer)
同步觸發最大負載	預設 1000 concurrency (可申請提升)
須設計無狀態	每次執行是全新環境，不保留記憶體資料

小結

今天介紹了 AWS Lambda 的核心概念：

- 完全免維護伺服器
- 按次計費、真正用多少付多少
- 適合 API、小型運算任務、背景作業

- 超級擴展性，適合不確定流量的應用

Lambda 完全顛覆了傳統開發部署方式，讓開發者能更專注在**功能本身**，而不是維護伺服器！

好！這裡直接幫你產出

AWS Lambda 實作教學（Hello World + API Gateway 快速串接）

讓讀者可以跟著馬上做出第一個 Lambda Function，並且用瀏覽器呼叫成功！

AWS Lambda 實作教學（Hello World + API Gateway 快速串接）

前一篇我們認識了什麼是 AWS Lambda。

這一篇直接來實戰，帶你：

- 建立一個 Hello World Lambda
- 用 API Gateway 快速串接
- 用瀏覽器直接呼叫！

超簡單，只需要 10 分鐘 ☐

1. 建立第一個 Lambda Function

登入 AWS Console

打開 [AWS Lambda Console](#)

1. 點選「Create function」
2. 選擇「Author from scratch」
3. 填寫基本資訊：

項目	設定
Function name	<input type="text" value="hello-world-lambda"/>
Runtime	Node.js 18.x（或 Python 3.11 也可）
Architecture	x86_64（預設即可）
Permissions	建議選「Create a new role with basic Lambda permissions」

4. 點選「Create Function」

幾秒鐘後，一個新的 Lambda 就建立好了！

2. 編輯 Lambda 程式碼

在 Lambda Console 的編輯器中，把預設程式碼改成這樣：

Node.js 版

```
exports.handler = async (event) => {
  return {
    statusCode: 200,
    body: JSON.stringify('Hello from AWS Lambda!'),
  };
};
```

Python 版

```
def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from AWS Lambda!'
    }
```

編輯完後，記得按下「Deploy」！

這樣 Lambda 程式就設定完成了 ☐

3. 建立 API Gateway 串接 Lambda

接下來，我們讓這個 Lambda 能夠用 HTTP API 被呼叫。

- 1. 在 Lambda Function 頁面上方，點選「Add trigger」
- 2. 選擇「API Gateway」
- 3. 設定：

項目	設定
API type	HTTP API（比較簡單快速）
Security	Open（不設驗證，之後可以補強）
API Name	自動生成或自己命名也可以

- 4. 點選「Add」

AWS 會自動幫你建立一個 HTTP API，並串好 Lambda！

完成後，Lambda 頁面會顯示一個 API Endpoint，例如：

```
https://xxxxx12345.execute-api.ap-northeast-1.amazonaws.com/
```

4. 測試 API 呼叫

打開你的瀏覽器，直接輸入剛剛那個 URL！

你應該會看到畫面顯示：

```
"Hello from AWS Lambda!"
```

恭喜 ☐
你的第一個 Lambda + API Gateway 已經成功部署！

5. 小結

這次的流程總結：

步驟	說明
建立 Lambda	撰寫 Hello World 程式碼
建立 API Gateway Trigger	快速建立 HTTP API 串接 Lambda
測試呼叫	瀏覽器直接打 API URL 測試結果

透過 Lambda + API Gateway，可以超快速上線一個基本 API，而且完全不用管 EC2、Scaling、Load Balancer 等基礎設施！

太好了！這裡接著給你完整的
AWS Lambda 進階教學：環境變數、IAM 權限設定、連接 RDS 資料庫，
讓你的 Lambda 應用從 Hello World 升級到真正可以跑正式服務！

AWS Lambda 進階教學

(環境變數設定 + IAM 權限管理 + 連接 RDS 資料庫)

一、設定 Lambda 環境變數 (Environment Variables)

□ 為什麼需要環境變數？

- 存放敏感資訊 (資料庫帳號密碼、API Key)
- 存放不同環境 (dev / prod) 設定
- 程式更乾淨，不要把設定寫死在程式碼裡

□ 如何設定環境變數

1. 打開 Lambda Function 頁面
2. 選擇「Configuration」→「Environment variables」
3. 點「Edit」→「Add environment variable」
4. 加入變數，例如：

Key	Value
DB_HOST	your-db.xxxx.rds.amazonaws.com
DB_USER	admin
DB_PASSWORD	your-password
DB_NAME	your-database-name

5. 儲存即可，Lambda 會自動讀取這些變數

□ 程式裡讀取環境變數 (Python 範例)

```
import os

db_host = os.environ['DB_HOST']
db_user = os.environ['DB_USER']
db_password = os.environ['DB_PASSWORD']
db_name = os.environ['DB_NAME']
```

讀取環境變數超簡單，推薦大家一定要用這種方式管理設定！

二、設定 Lambda 的 IAM Role (權限)

□ 為什麼需要設定 IAM Role？

Lambda 本身如果要呼叫其他 AWS 資源 (像 S3、DynamoDB、RDS)，需要有「授權」。

這個授權是透過「IAM Role」給的。

“ □ Lambda 本身就會綁定一個 IAM Role，進行細部設定即可。

□ 最小權限原則（Best Practice）

- 只開放 Lambda 需要用到的權限
- 不要直接給 Lambda admin 權限（避免安全漏洞）

□ 範例：讓 Lambda 存取 RDS（或 VPC）

如果 Lambda 需要連到 RDS，需要設定：

- 讓 Lambda 進入正確的 VPC、Subnet、Security Group
- IAM Role 不需要特別開 RDS 權限（除非要操作 RDS API，例如開啟 RDS Snapshot）

所以通常 IAM Role 保持「基本執行」權限就好，例如：

- `AWSLambdaBasicExecutionRole`（CloudWatch log 權限）
- `AWSLambdaVPCAccessExecutionRole`（存取 VPC 權限）

可以在 Lambda → Configuration → Permissions → Execution Role 編輯。

三、Lambda 連接 RDS 資料庫（Python 範例）

假設我們已經設定好 RDS MySQL 或 PostgreSQL 資料庫，並且把連線資訊設在環境變數。

□ 安裝必備套件

AWS Lambda 本身環境很乾淨，需要把 Python 連線資料庫的套件（例如 `pymysql`、`psycopg2`）打包進去。

開發環境步驟：

```
mkdir lambda_rds
cd lambda_rds
python3 -m venv venv
source venv/bin/activate
pip install pymysql
deactivate
```

把 `venv/lib/python3.11/site-packages` 的 `pymysql` 複製到專案目錄。

最後 ZIP 包含：

- `lambda_function.py`（你的程式）
- `pymysql/`（套件目錄）

□ Lambda Python 連線 RDS (MySQL) 範例

```
import pymysql
import os

def lambda_handler(event, context):
    # 讀取環境變數
    db_host = os.environ['DB_HOST']
    db_user = os.environ['DB_USER']
    db_password = os.environ['DB_PASSWORD']
    db_name = os.environ['DB_NAME']

    # 建立連線
    connection = pymysql.connect(
        host=db_host,
```

```
user=db_user,
password=db_password,
database=db_name,
connect_timeout=5
)

try:
    with connection.cursor() as cursor:
        sql = "SELECT NOW();"
        cursor.execute(sql)
        result = cursor.fetchone()
        return {
            'statusCode': 200,
            'body': f'Database Time: {result[0]}'
        }
finally:
    connection.close()
```

這樣就能讓 Lambda 成功連接雲端資料庫，完成資料操作！

四、常見注意事項

注意事項	說明
Lambda + RDS Latency	Lambda 通常要設在與 RDS 同一個 VPC / AZ
連線池設計	建議使用 RDS Proxy (可以省連線數)
超時設置	Lambda 最大執行時間 15 分鐘，連資料庫要小心逾時
打包套件	自訂套件一定要打包進 ZIP 中

小結

這篇進階教學重點整理：

- Lambda 用環境變數管理設定
- Lambda 設定 IAM Role，最小權限原則
- Lambda 可以直接連 RDS，但要注意 VPC 設定
- 打包套件進 ZIP 是必須的！

掌握這些，
你的 Lambda 就可以從 Hello World 升級成**正式後端服務**了！

AWS Serverless 全應用實戰

(API Gateway + Lambda + DynamoDB + RDS Proxy)

□ 架構圖



一、建立 DynamoDB Table

1. 打開 [DynamoDB Console](#)
2. 點選「Create Table」
3. 設定：



項目	值
Table name	Todos
Partition key	id (String)

4. 其他預設即可，點「Create table」

二、撰寫 Lambda Function

□ Lambda 功能：新增一筆 Todo 資料

程式語言以 Python 為例，並假設請求是 `POST`，Body 包含 `title` 欄位。

python



```
import json
import uuid
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Todos')

def lambda_handler(event, context):
    body = json.loads(event['body'])
    todo_id = str(uuid.uuid4())
    title = body.get('title')

    table.put_item(
        Item={
            'id': todo_id,
            'title': title
        }
    )

    return {
        'statusCode': 200,
        'body': json.dumps({'id': todo_id, 'title': title})
    }
```

□ Lambda 要有以下 IAM 權限：

json



```
{
  "Effect": "Allow",
  "Action": [
    "dynamodb:PutItem"
  ],
  "Resource": "arn:aws:dynamodb:*:*:table/Todos"
}
```

“ 可以在 Lambda 執行角色（Execution Role）中加上這段 policy。

三、建立 API Gateway 串接 Lambda

1. 打開 [API Gateway Console](#)
2. 建立一個「HTTP API」
3. 新增路由：
 - 方法：POST
 - 路徑：/todos
 - 整合：選擇剛剛的 Lambda
4. Deploy 完後，你會得到一個 API URL

使用 curl 或 Postman 測試：

bash

複製

編輯

```
curl -X POST https://xxx.execute-api.ap-northeast-1.amazonaws.com/todos \
-H "Content-Type: application/json" \
-d '{"title":"Write Lambda guide"}'
```

四、加碼：改用 RDS + RDS Proxy 作為資料庫層

☐ 為什麼使用 RDS Proxy ？

- 解決 Lambda 對 RDS 連線過多問題
- 支援連線池（Connection pooling）
- 提高效率，降低 DB crash 風險

步驟 1：建立 RDS Proxy

1. 前往 [RDS Console](#)
2. 選單點「Proxies」→「Create proxy」
3. 設定：



項目	值
Proxy name	todo-proxy
Engine	MySQL / PostgreSQL（需有資料庫）
Target RDS	選擇現有的 RDS 實例
IAM Auth	啟用（建議使用）
VPC	與 Lambda 相同 VPC

“ 建好後會給你一個 Proxy endpoint：例如 `todo-proxy.proxy-cdefg.ap-northeast-1.rds.amazonaws.com` ”

步驟 2：Lambda 改連 Proxy（Python）

安裝並打包 pymysql：

bash

複製

編輯

```
pip install pymysql -t .
zip -r lambda-rds.zip .
```


Lambda 程式碼範例：

python

複製

編輯

```
import pymysql
import os
import json
import uuid

def lambda_handler(event, context):
    conn = pymysql.connect(
        host=os.environ['DB_PROXY_HOST'], # Proxy endpoint
        user=os.environ['DB_USER'],
        password=os.environ['DB_PASSWORD'],
        database=os.environ['DB_NAME'],
        connect_timeout=5
    )

    try:
        body = json.loads(event['body'])
        todo_id = str(uuid.uuid4())
        title = body['title']

        with conn.cursor() as cursor:
            cursor.execute("INSERT INTO todos (id, title) VALUES (%s, %s)", (todo_id, title))
            conn.commit()

    return {
        'statusCode': 200,
        'body': json.dumps({'id': todo_id, 'title': title})
    }

    finally:
        conn.close()
```

□ 注意：

- Lambda 要設定進入 **同 VPC + Subnet + SG**
- 建議設定 `AWSLambdaVPCLambdaAccessExecutionRole`
- Lambda 的 IAM Role 不需要存取 RDS Proxy，只需要 DB 連線資訊與網路存取權

五、小結

這篇教你從 0 建立一個完整的無伺服器應用：



元件	用途
API Gateway	提供外部 API 接口
Lambda	執行商業邏輯 Function
DynamoDB / RDS	儲存資料
RDS Proxy	提升連線穩定性與效率

這套架構非常適合：

- 小型 SaaS API
- 行動 App 後端
- MVP 初期應用快速開發
- 大型應用中的 Serverless 子系統（例如上傳任務、登入、Webhook 處理等）

太好了！這裡是你完整的加強版：

AWS Serverless Todo API 進階整合：Cognito 驗證、錯誤處理、Serverless Framework 快速部署

這篇會在前面 Lambda + API Gateway + DynamoDB/RDS 架構上，補上以下三個常用進階功能：

1. □ **API Gateway JWT 驗證 (Cognito)**
2. □ **Lambda 錯誤處理與 Logging 最佳實踐**

Serverless Todo API 完整實戰進階篇

一、加入 Cognito 驗證保護 API

為什麼需要？

不管是 Web 前端還是 App，只要對外開 API，基本保護機制不可少。

AWS 的 Cognito 可以幫你處理：

- 使用者註冊 / 登入
- JWT Token 簽發與驗證
- 無需自建帳號系統！

建立 Cognito User Pool

1. 開啟 [Cognito Console](#)
2. 建立一個 **User Pool**
 - Pool name: `todo-api-users`
 - 啟用 Email 登入即可
 - 完成後記住 Pool ID 和 Region
3. 建立 App Client
 - 禁用 client secret (Lambda 驗證用不到)
 - 儲存產生的 `App client ID`

API Gateway 整合 JWT 驗證

1. 回到 API Gateway HTTP API Console
2. 選「Authorization」→「Create authorizer」
3. 建立 Cognito JWT authorizer：

項目	設定
Type	JWT
Identity source	<code>Authorization</code> header
Issuer URL	<code>https://cognito-idp.<region>.amazonaws.com/<pool-id></code>
Audience	你剛剛的 App Client ID

4. 把 `/todos` 路徑設定為需要驗證 (選上剛剛的 authorizer)

呼叫 API 時加上 JWT token

前端從 Cognito 登入取得 Token 後，呼叫 API 時帶上：

```
Authorization: Bearer <token>
```

這樣 API Gateway 就會自動驗證使用者身份，Lambda 裡也可以透過 `event['requestContext']` 取得使用者資訊！

二、Lambda Logging 與 Error Handling 最

佳實踐

☐ 記錄 Log 到 CloudWatch

在 Lambda 中使用標準輸出即可：

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

logger.info("New todo created")
```

CloudWatch Logs 會自動收集這些 log。

☐ 最佳錯誤處理模式

```
import json
import traceback

def lambda_handler(event, context):
    try:
        # 主邏輯
        ...
        return {
            'statusCode': 200,
            'body': json.dumps({'message': 'Success'})
        }
    except Exception as e:
        logger.error("Error: %s", traceback.format_exc())
        return {
            'statusCode': 500,
            'body': json.dumps({'error': 'Internal Server Error'})
        }
```

這樣可避免直接顯示錯誤堆疊給使用者，又方便自己 debug。

三、使用 Serverless Framework 快速部署整套應用

☐ 為什麼用 Serverless Framework？

- 自動建 Lambda、API Gateway、DynamoDB、IAM、環境變數
- 一鍵部署、一鍵刪除
- 適合團隊開發與 CI/CD

☐ 安裝 CLI 工具

```
npm install -g serverless
```

☐ 初始化專案

```
serverless create --template aws-python --path todo-api
cd todo-api
```

編輯 serverless.yml

以下是 Todo API + DynamoDB + JWT 驗證的簡化範例：

```
service: todo-api

provider:
  name: aws
  runtime: python3.11
  region: ap-northeast-1
  environment:
    DB_TABLE: Todos
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:PutItem
      Resource: arn:aws:dynamodb:*:*:table/Todos

functions:
  createTodo:
    handler: handler.create
    events:
      - http:
          path: todos
          method: post
          authorizer:
            type: COGNITO_USER_POOLS
            userPoolArn: arn:aws:cognito-idp:ap-northeast-1:xxxx:userpool/xxxx

resources:
  Resources:
    TodosTable:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: Todos
        AttributeDefinitions:
          - AttributeName: id
            AttributeType: S
        KeySchema:
          - AttributeName: id
            KeyType: HASH
        BillingMode: PAY_PER_REQUEST
```

Lambda 程式 handler.py

```
import json
import uuid
import boto3
import os

table = boto3.resource('dynamodb').Table(os.environ['DB_TABLE'])

def create(event, context):
    data = json.loads(event['body'])
    todo_id = str(uuid.uuid4())

    table.put_item(Item={'id': todo_id, 'title': data['title']})

    return {
        'statusCode': 200,
        'body': json.dumps({'id': todo_id, 'title': data['title']})
    }
```

☐ 部署指令

```
sls deploy
```

幾十秒後，你就會拿到公開的 API URL！☐

☐ 刪除整個堆疊（清資源）

```
sls remove
```

超級適合測試環境清除資源時使用，防止殘留帳單！

☐ 加碼：Lambda 搭配 RDS Proxy 用 Serverless Framework 管理

你可以在 `serverless.yml` 裡加上 VPC 設定：

```
provider:
  ...
vpc:
  securityGroupIds:
    - sg-xxxxxxx
  subnetIds:
    - subnet-aaaa
    - subnet-bbbb
```

這樣部署後的 Lambda 就能存取你已經建立好的 RDS Proxy！

☐ 小結

這次進階整合包含：

- ☐ Cognito 驗證 → 提供安全 API
- ☐ Lambda Logging + Error Handling → 更穩定、更易除錯
- ☐ Serverless Framework → 快速一鍵部署 / 一鍵刪除
- ☐ 支援 DynamoDB / RDS Proxy → 彈性資料層選擇

這套完整架構，適合你用來開發：

- SaaS MVP
- 微服務架構中的 User Service
- 內部工具（支援 Cognito 驗證）
- 全無伺服器 RESTful API

下一篇，我們將繼續探討

Day 9：AWS 容器服務 - EKS (Elastic Kubernetes Service)，看看當需要更複雜的大型應用，想用 Kubernetes 架構時，AWS 是怎麼支援的！

Day 9 再見 ☐