

【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)
- `AWSLambdaVPCLAccessExecutionRole` (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 ☐

🕒 修訂版本 #3

★ 由 treeman 建立於 25 🕒🕒🕒 2025 19:08:41

✍ 由 treeman 更新於 25 🕒🕒🕒 2025 19:32:13