國立雲林科技大學

雲端運算概論期末專案

基於無伺服架構的生成式模型與 Line Bot 開發

Development of a Serverless Architecture-Based Generative AI & Line Bot

組別：第四組

專題組員：　B11223020　張哲維

B11223220　鄭人傑

A11223032　林冠澔

B11223052　潘宣融

B11223062　蔡承軒

Dec, 2024

摘要

Line 是全球最受歡迎的即時通訊應用程式之一，用戶在移動端頻繁使用它進行交流。然而，使用生成式人工智慧技術通常需要額外安裝應用程式或透過 Web 版操作，對部分用戶而言略顯不便。本專題針對這一問題，採用無伺服架構（Serverless）設計並開發一個智能對話的 Line Bot 系統，提供用戶高效、無縫的使用體驗。

系統利用 AWS Lambda 與 API Gateway 處理後端邏輯，結合 OpenAI 生成式模型和 LINE Developers 服務，讓用戶能在熟悉的 Line 平台上即時體驗生成式人工智能服務。架構包含用戶端、前端、後端與雲端監控四大模組，從訊息傳遞、邏輯處理到回應生成的整個流程高效運行。

本系統實現了高彈性與可擴展性，不僅減少伺服器管理的負擔，還降低用戶操作的門檻，讓用戶能獲得高效的資訊查詢與問題解答，進一步提升智慧交互的整體體驗。

目錄

摘要 I
一、前言 1
1.1 動機 1
1.2 目的 1
二、技術與服務概述 2
2.1 Amazon Web Services (AWS) 2
2.1.1 AWS Lambda 2
2.1.2 API Gateway 2
2.1.3 Amazon CloudWatch 3
2.2 LINE Developers 3
2.3 OpenAI 3
三、系統設計與實施 4
3.1 系統架構 4
3.2 流程說明 5
3.3 配置流程 6
3.3.1 LINE Developers 設定 6
3.3.2 OpenAI API 設定 6
3.3.3 Lambda 部屬 6
3.3.4 API Gateway 配置 7
四、實際展示 9
五、結論 10
5.1 傳統架構與無伺服器架構的比較 10
5.2 未來改善方向 10
5.2.1 整合更多生成式 AI 模型 10
5.2.2 個性化系統 10
5.2.3 引入 RAG 技術 (Retrieval-Augmented Generation) 11
參考文獻 12

一、前言

1.1 動機

現今，Line 已經成為全球最受歡迎的即時通訊應用程式之一，大多數人都在移動端（手機）安裝並頻繁使用它。然而，對於需要使用 ChatGPT 等先進人工智慧技術的用戶來說，通常需要安裝額外的應用程式或透過 Web 版進行操作，這樣的使用門檻和操作流程對某些用戶來說可能不夠便捷。此外，傳統伺服器架構在串接 ChatGPT 等技術時，可能會面臨擴展性和維護上的挑戰。因此，採用無伺服架構進行整合與開發，能夠大幅提供更高的彈性與擴展性，還能顯著減少伺服器管理的負擔，從而實現更加高效且無縫的使用者體驗。

1.2 目的

本專題旨在開發與整合先進的生成式人工智慧技術（Generative AI）和無伺服架構（Serverless）技術，利用大規模語言模型（LLM）、LINE Developers 服務及 Amazon Web Services（AWS）中的 Lambda 與 API Gateway 服務來處理後端邏輯與無伺服架構。

用戶能夠在熟悉的 Line 聊天室內，直接體驗智能對話機器人的功能。這樣的系統應用不僅能降低服器管理的負擔，還提高更多彈性與擴展性，在使用者端也能夠減少繁瑣的操作步驟，提供即時、高效的問題回答和資訊查詢，從而提升用戶體驗。

二、技術與服務概述

2.1 Amazon Web Services (AWS)

AWS 是全球最全面、最廣泛採納的雲端服務，透過全球資料中心提供超過 200 項功能完整的服務。數百萬個客戶，包括成長最快的新創公司、最大型企業以及領先的政府機構，都使用 AWS 來降低成本、變得更靈活，且更迅速地創新。

圖: AWS

2.1.1 AWS Lambda

Lambda 是理想的運算服務，適用於需要快速縱向擴展的應用程式案例，並在不需要時縮減規模至零。例如，可將 Lambda 用於檔案處理、串流處理、Web 應用程式、IoT 後端、行動後端等。

使用 Lambda 時，只需負責程式碼的相關操作。Lambda 會管理運算叢集，提供平衡的記憶體 CPU、網路和其他資源來執行程式碼。

圖: AWS Lamada

2.1.2 API Gateway

API Gateway 是一種全受管的服務，可讓開發人員輕鬆地建立、發佈、維護、監控和保護任何規模的 API，並支援容器化、無伺服器工作負載和 Web 應用程式。它可以充當前端應用和後端服務之間的中介。

API Gateway 沒有最低費用或啟動成本，以收到的 API 呼叫和資料傳輸量支付費用，而使用 API Gateway 分級定價模型，可在 API 用量擴展時減少成本。

圖: API Gateway

2.1.3 Amazon CloudWatch

Amazon CloudWatch 是一個強大的工具，可以監控 AWS 資源和正在運行的應用程式。使用 CloudWatch 可以收集和追蹤各種指標，這些指標用於測量資源和應用程式的性能。還可以設置警報來監控這些指標，當指標超過設定的閾值時，CloudWatch 會發送通知，或者自動對被監控的資源進行相應的調整。通過這些功能，可以全面掌握整個系統的資源利用率、應用程式性能和運營狀態。

圖: Amazon CloudWatch

2.2 LINE Developers

LINE Developers 是 LINE 提供的一個開發者平台，讓開發者可以使用 LINE 的 API 和工具來創建和整合各種應用程式和服務。這個平台包括了許多功能，例如 Messaging API、LINE Login、LINE Pay 及 LINE Notify 等，讓開發者能夠與 LINE 使用者進行互動，提供更多的服務和功能。

圖: LINE Developers

2.3 OpenAI

OpenAI 是一個美國人工智慧研究實驗室，由非營利組織 OpenAI Inc，和其營利組織子公司 OpenAI LP 所組成。OpenAI 進行 AI 研究的目的是促進和發展友好的人工智慧，使人類整體受益。其中 OpenAI 推出 OpenAI API 服務，用於存取 OpenAI 開發的 AI 模型，使用者可以要求存取權限，以便將 API 整合到產品中、開發全新的應用程式。

圖: OpenAI

三、系統設計與實施

3.1 系統架構

此架構將模塊劃分為四部分:

I. 用戶端	II. 前端
● Line App	● Line Messaging API ● API Gateway
III. 後端	**IV. 雲端監控**
● AWS Lambda ● OpenAI API	● Amazon CloudWatch

表: 系統模塊

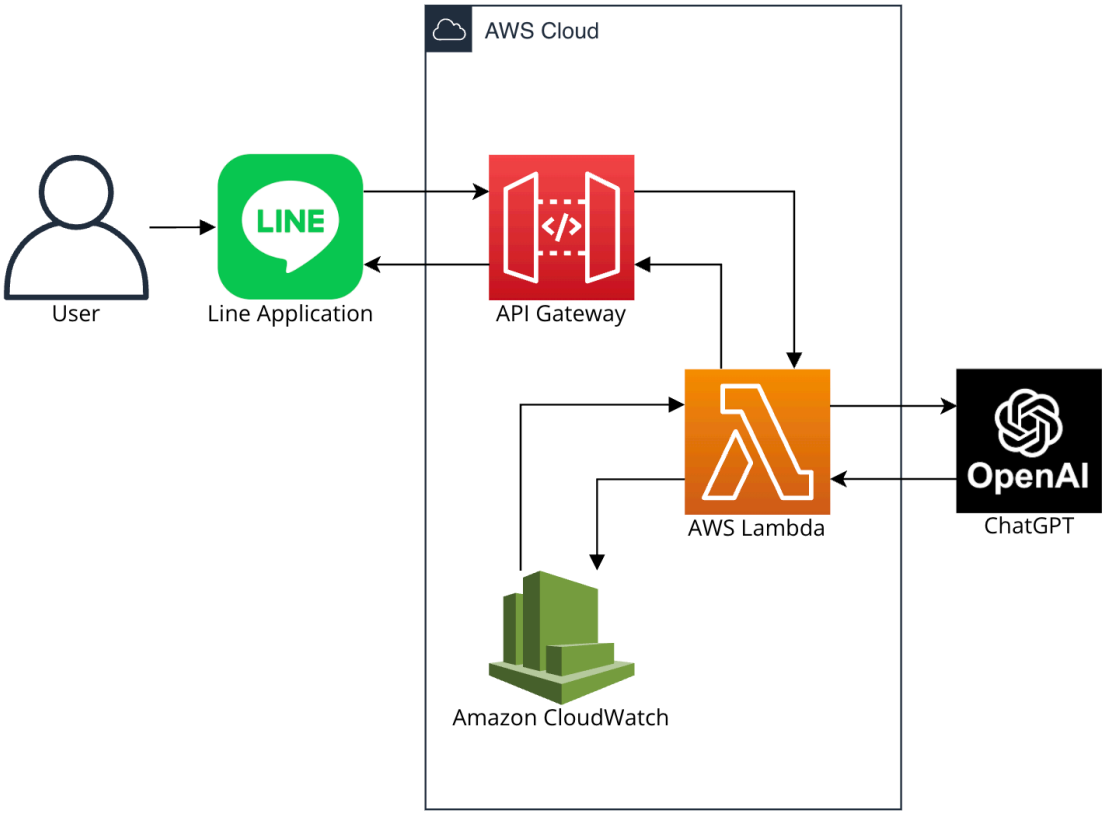


圖: 系統架構圖

3.2 流程說明

I. 用戶在 Line 發送訊息	
執行說明：	在 Line 聊天室輸入訊息後
II. Line 聊天室接收訊息	
執行說明：	Line Message API 將該訊息透過 Webhook (API Gateway 端點)傳遞到後端。伺服器會攜帶用戶訊息等資料，將其封裝成 Http Post 請求。
III. API Gateway 轉發請求至 AWS Lambda	
執行說明：	API Gateway 充當入口，接收傳遞的請求並將其路由到 AWS Lambda。經過 API Gateway 的路徑和權限驗證後，觸發 Lambda 函數。
IV. AWS Lambda 處理邏輯	
執行說明：	讀取 API Gateway 傳遞的訊息、解析用戶訊息、調用 OpenAI API 以及解析 OpenAI API 回傳生成的結果。
V. 調用 OpenAI API	
執行說明：	Lambda 傳遞的訊息作為請求參數發送給 OpenAI 的大型語言模型且返回生成的文字結果。
VI. Lambda 回傳結果至 API Gateway	
執行說明：	Lambda 整理 OpenAI API 的回應並封裝為 Line Message API 的回應格式，包括用戶訊息及處理結果等。
VII. API Gateway 回傳結果至 Line Messaging API	
執行說明：	API Gateway 將 Lambda 的結果作為 Http 回應發送給 Line Messaging API。
VIII.聊天室顯示回覆的訊息	
執行說明：	Line Messaging API 將回應內容發送到用戶的聊天室。

表: 流程說明

3.3 配置流程

3.3.1 LINE Developers 設定

1. 建立 Messaging API 通道

- 登錄 LINE Developers，建立一個新的 Messaging API 通道。
- 獲取 Channel Secret 和 Channel Access Token，用於後續的驗證和訊息傳遞。

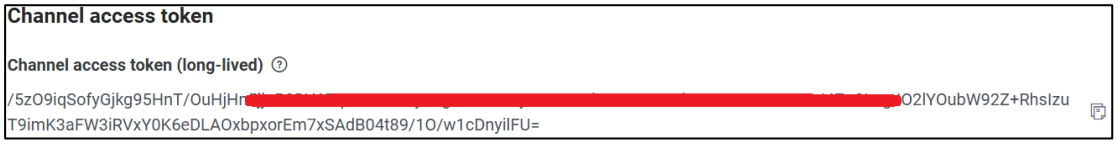


圖: Channel Access Token

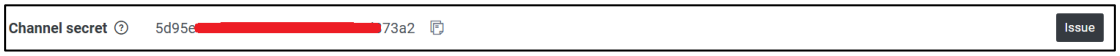


圖: Channel Secret

2. 設定 Webhook URL

- 將 Webhook URL 設置為 API Gateway 的端點（稍後配置）。
- 啟用 Webhook 功能，確保 Line 伺服器能將用戶訊息推送到後端。

3.3.2 OpenAI API 設定

1. 申請 OpenAI API Key

- 登錄 OpenAI，建立一個新的專案。
- 獲取 API Key，用於調用 OpenAI 模型與後續的驗證和訊息傳遞。

圖: OpenAI API Key

3.3.3 Lambda 部屬

1. 環境建置

- 使用 Python 作為主要環境。
- 將 openai、line-bot-sdk 等函數庫打包上傳至 AWS Lambda。
- 以下基於 macOS 上打包函式庫步驟。
  - 使用 virtualenv。

    #python3 -m venv

    #venv source venv/bin/activate

- 安裝套件。

  pip install openai line-bot-sdk

- 建立資料夾。

  #mkdir llm_linebot_pkg

- 將套件導出及壓縮成 ZIP 文件。

  #cp -r venv/lib/python3.*/site-packages/* llm_linebot_pkg/

  #zip -r llm_linebot_pkg zip llm_linebot_pkg/

2. 撰寫 Lambda 程式碼

- 這裡使用 Python 進行串接。
- 預設檔名為 lambda_function.py 的程式檔案，並執函式 lambda_handler 作為進入點。

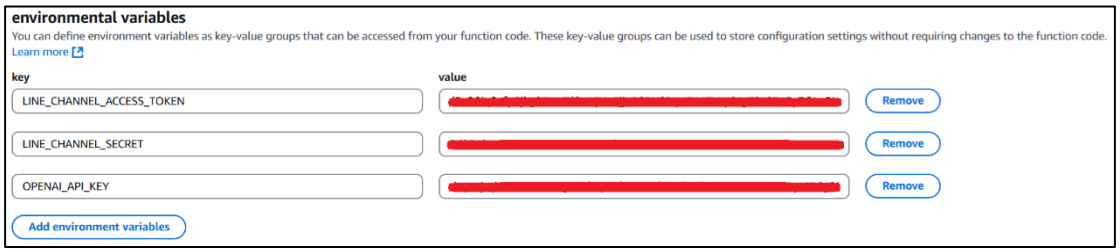
```
def lambda_handler(event, context):
    #textmsg trigger
    @handler.add(MessageEvent, message=TextMessage)
    def handle_message(event):
```

圖：函式架構

- 以下為程式主要邏輯。
  - 解讀 LINE 傳入的訊息。
  - 提取用戶訊息，並調用 OpenAI API 進行處理。
  - 將結果轉換為回應 Text Message 格式。

3. 配置環境變數

- 利用環境變數傳遞設定值，將敏感資訊及上述獲取的金鑰編碼到程式中。



圖：環境配置

4. 測試與部屬

- 使用事件觸發 Lambda 函數，驗證邏輯是否正確。

3.3.4 API Gateway 配置

1. 建立觸發器

- 使 API Gateway 作為 Lambda 的觸發器。
- 建立 HTTP API。

2. 設置資源和方法

- 定義資源路徑。
- 為資源添加 POST 方法 (LINE Webhook 請求使用 POST)。

3. 啟用 API 並配置端點

- 部署 API 到指定的階段 (/default)
- 獲得 API 端點 URL

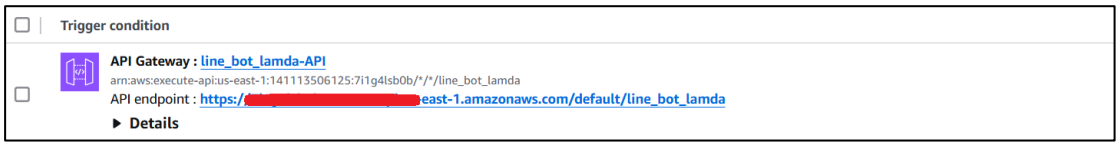


圖: API endpoint

- 將端點配置於 LINE Messaging API (LINE Webhook 必須為 SSL 支援，透過 API Gateway 提供的 HTTPS URL 加密連接)。

圖: Webhook URL 設置

四、實際展示

圖: 操作 1

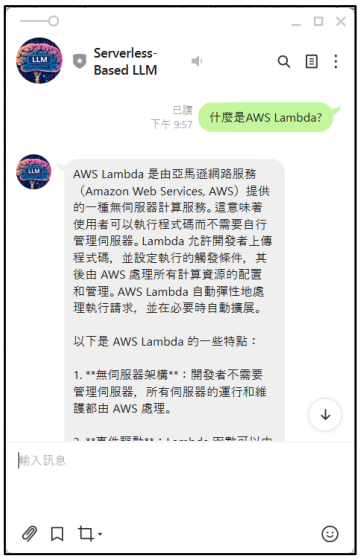


圖: 操作 2

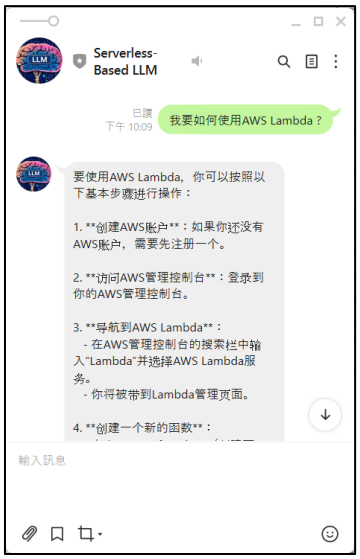


圖: 操作 3

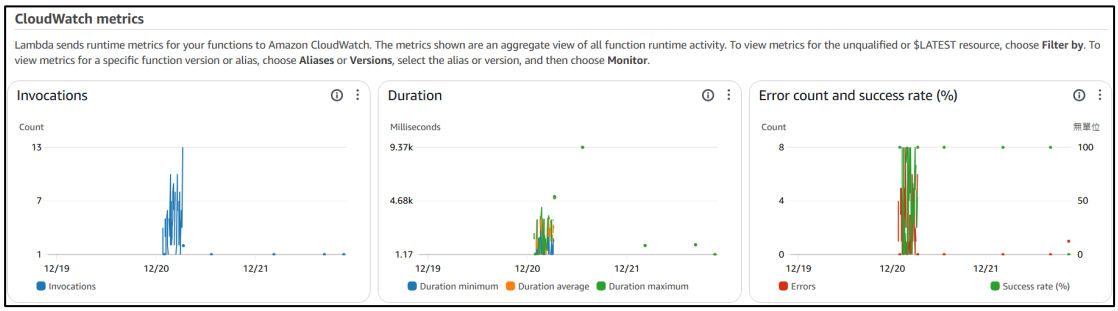


圖: CloudWatch 指標

五、結論

5.1 傳統架構與無伺服器架構的比較

	傳統架構	Serverless 架構
使用方法	啟動 EC2 實例，手動管理伺服器和應用。	Lambda+API Gateway，自動執行邏輯運行。
執行成本	固定(即使無請求，還是需要支付伺服器運行費)。	按執行次數和執行時間計費。
配置複雜度	高，需手動配置伺服器、網路等基礎設施。	低，基礎設施 AWS 會自動處理）。
管理負擔	高，需管理伺服器、更新、擴展。	低，AWS 自動管理。
效率	低流量時浪費資源，因為伺服器一直在運作，高流量時可能會產生延遲問題，如果沒有及時手動擴展實例。	高，Lambda 可自動擴展執行實例。
靈活性	高，可以完全控制環境設定。	低。

表: 傳統架構與無伺服器架構的比較

5.2 未來改善方向

將更多模型的整合、個性化系統，及 RAG（Retrieval-Augmented Generation）技術引入到 LINE Bot 中，將顯著提升系統的智能化、準確性和使用體驗。

5.2.1 整合更多生成式 AI 模型

為了提供多樣化的服務，除了目前使用的 OpenAI 模型，還可以引入其他生成式 AI 模型來豐富系統的功能，可以在 Line 聊天室進行切換或選配，可這針對不同領域或需求進行優化與改進。

5.2.2 個性化系統

因目前暫未將用戶訊息進行儲存與收集，為了提高用戶的交互體驗，個性化系統至關重要。透過用戶的歷史數據和行為分析，可以實現一些個性化功能，如根據用戶過去的對話內容或偏好，推薦相關問題的答案或個性化的建議、用戶經常詢問某類問題，可以預測並主動提供相關資訊或功能。

5.2.3 引入 RAG 技術 (Retrieval-Augmented Generation)

RAG 技術結合了信息檢索 (Retrieval) 和生成模型 (Generation) 的優勢，可以顯著提升聊天機器人的回答準確性和知識覆蓋範圍。具體來說，這可以通過資料庫中檢索相關信息、動態知識庫更新，強化查詢生成且根據檢索結果生成回答，以提高生成的回答更加準確且基於最新的信息與解決長尾問題等，使系統更為智能和靈活。

參考文獻

[1]什麼是 AWS？檢自：https://aws.amazon.com/tw/what-is-aws/

[2]什麼是 AWS Lambda？檢自：https://docs.aws.amazon.com/zh_tw/lambda/latest/dg/welcome.html

[3]Amazon API Gateway | API 管理| Amazon Web Services。檢自：https://aws.amazon.com/tw/api-gateway/

[4]什麼是 Amazon CloudWatch？檢自：https://docs.aws.amazon.com/zh_tw/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[5] LINE Developers。檢自：https://developers.line.biz/en/?form=MG0AV3

[6] OpenAI API。檢自：https://openai.com/index/openai-api/