中原大學 雲端計算平台實務

12/16 - 作業報告

Create serverless applications

資訊碩一 11177035 林彥輝

授課教師：鍾武君 教授

中華民國一一一年十二月

1. Model Intro

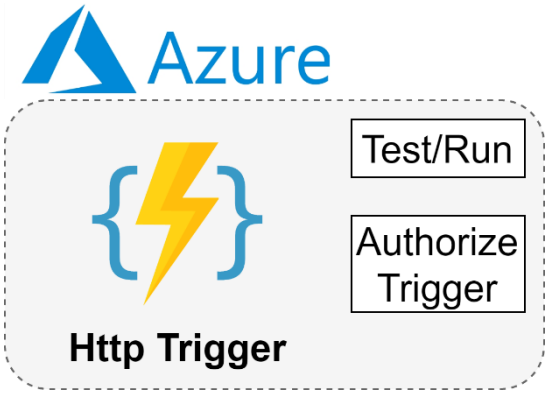
Create serverless applications

https://docs.microsoft.com/en-us/learn/paths/create-serverless-applications/

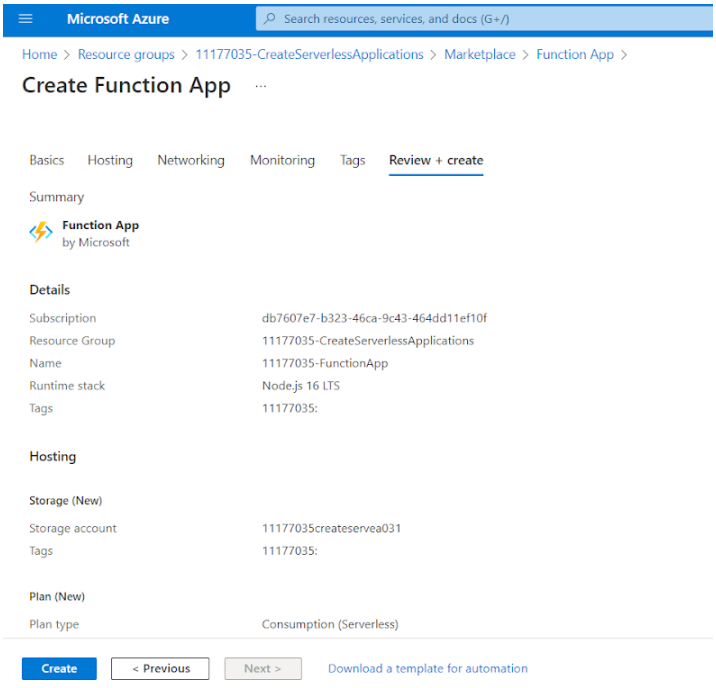
2. Summary Homework Assignment

Module 3: Create serverless logic with Azure Functions

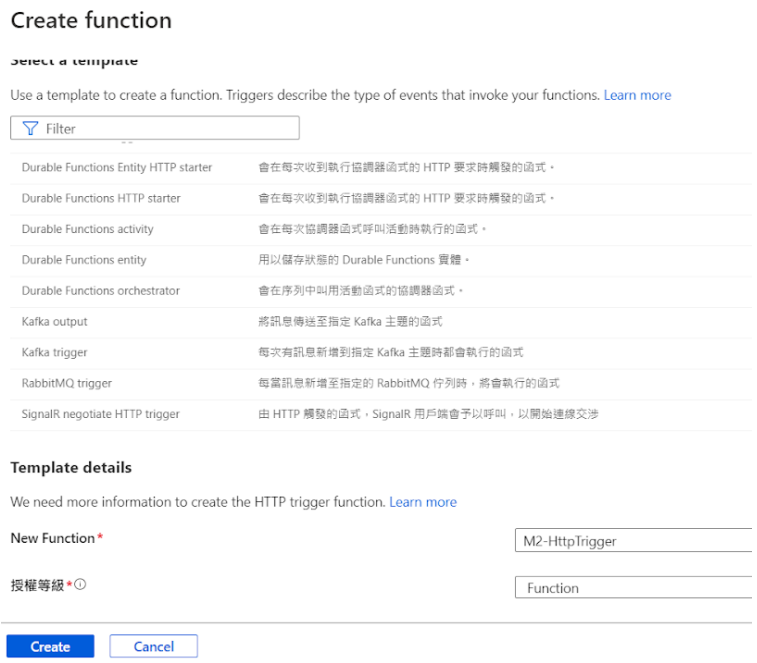
此 Module 最主要初步了解如何在 Azure Portal 開啟一個 Azure Function 的資源，新增一個簡單的 Http Trigger Function，並使用 Azure Portal 提供的開發測試工具在 Portal 中就能開發自己的 Azure Function。在每一個 Function 創建時都能選擇授權等級，決定該 Function 使用的權限，此 Module 也教導如何使用 Function Key 觸發 Function Level 的 Function。最後使用一個物聯網的例子展示 Azure Function 的使用方式。



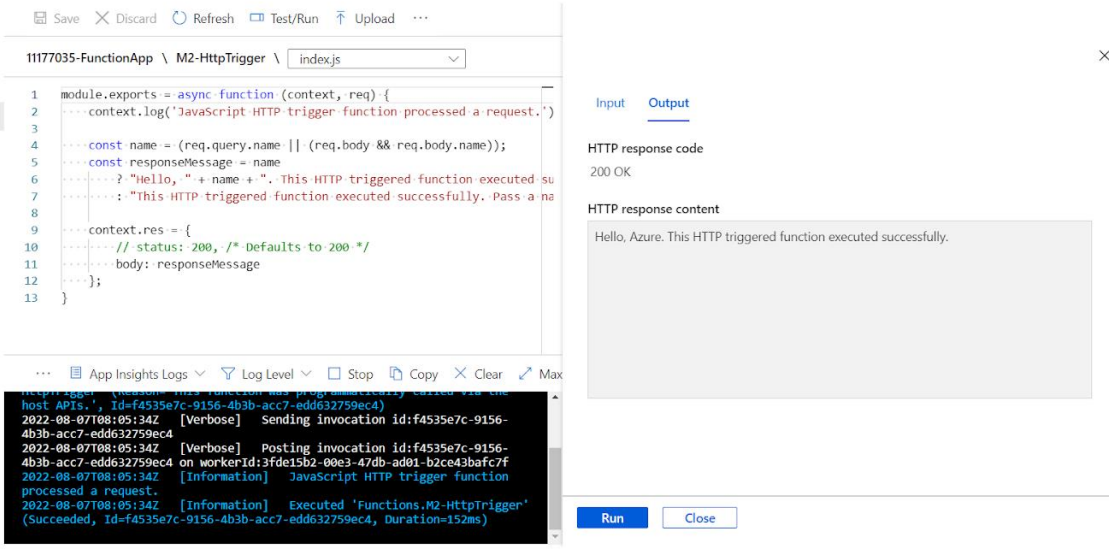
首先，創建一個 Function App 資源。



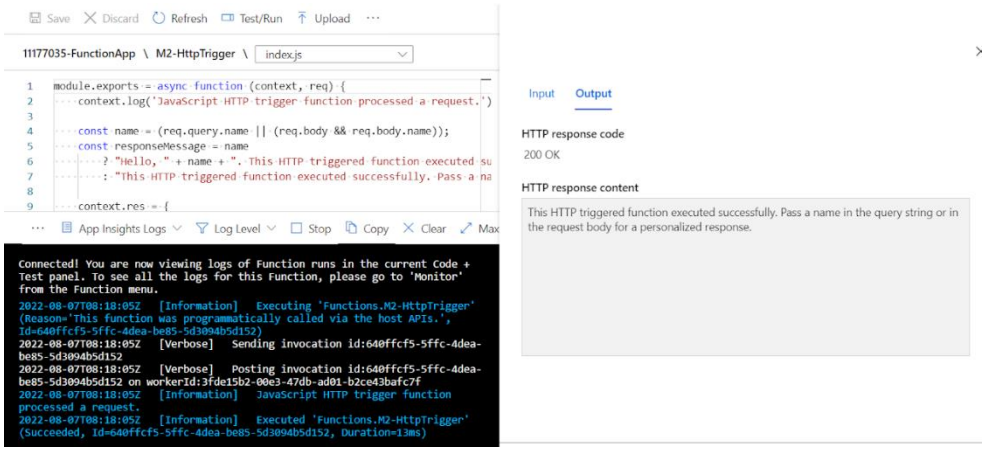
創建一個 Http Trigger，並選定授權等級為”Function” 。



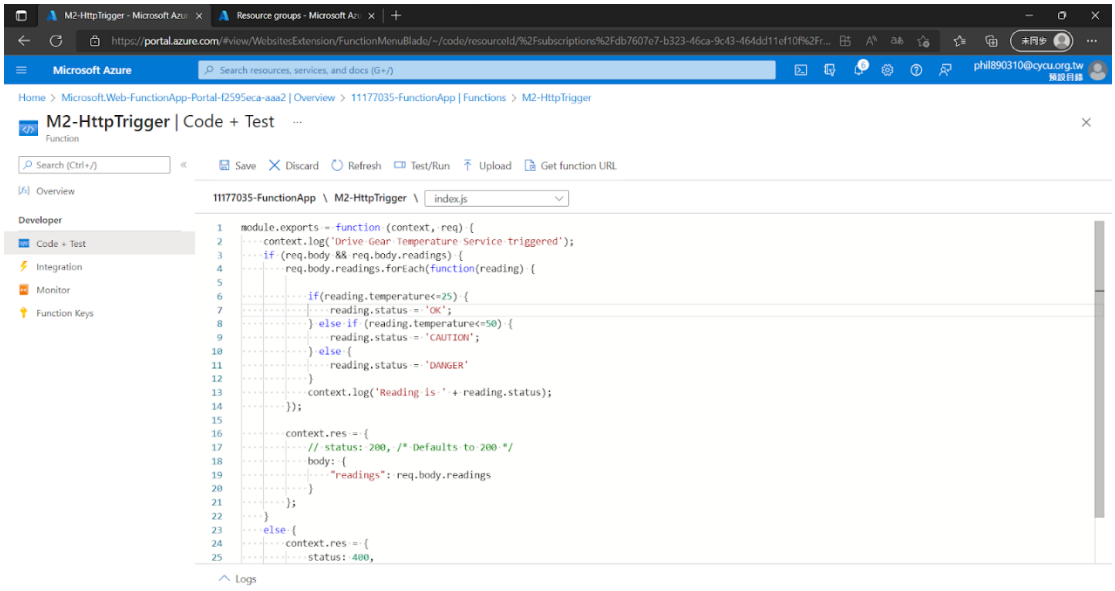
使用 Test/Run 功能，做簡單的測試，使用 Post 的方式傳輸 Name: Azure。



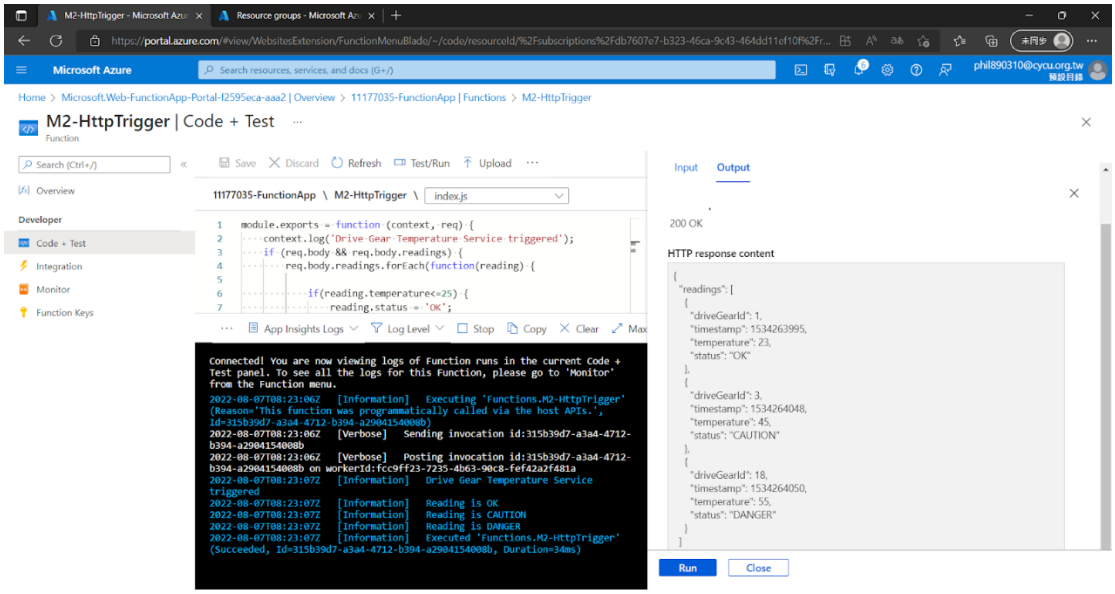
使用 Curl 指令，header 帶入 FunctionKey，進一步觸發 Http Trigger。



修改 HttpTrigger 程式碼，使其成為 IoT 溫度檢測的應用。

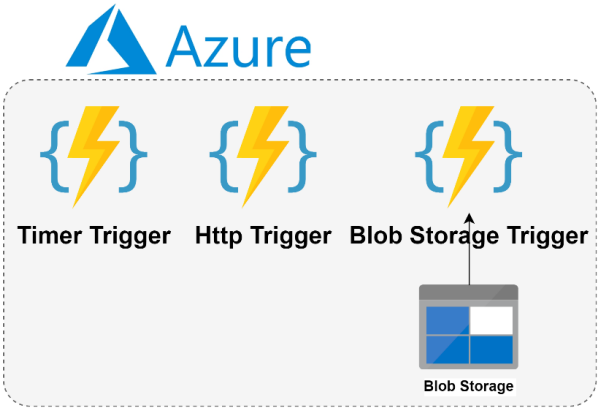


使用 IoT 測試數據驗證 Function，成功觸發 Azure Function，也成功根據溫度回應不同的結果。



Module 4: Execute an Azure Function with triggers

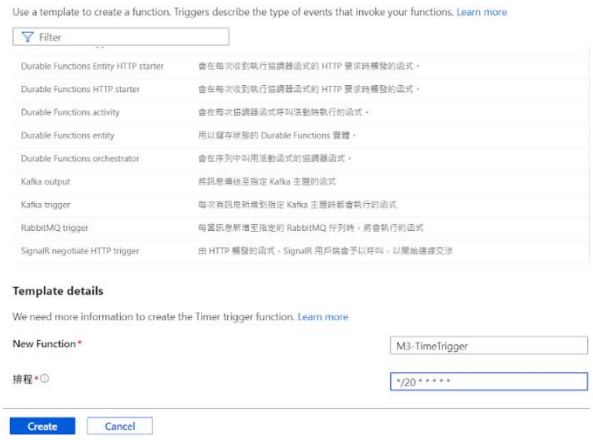
此 Module 重點教導 Azure Function 的更多 Trigger 方法，例如 Timer Trigger、Blob Trigger。



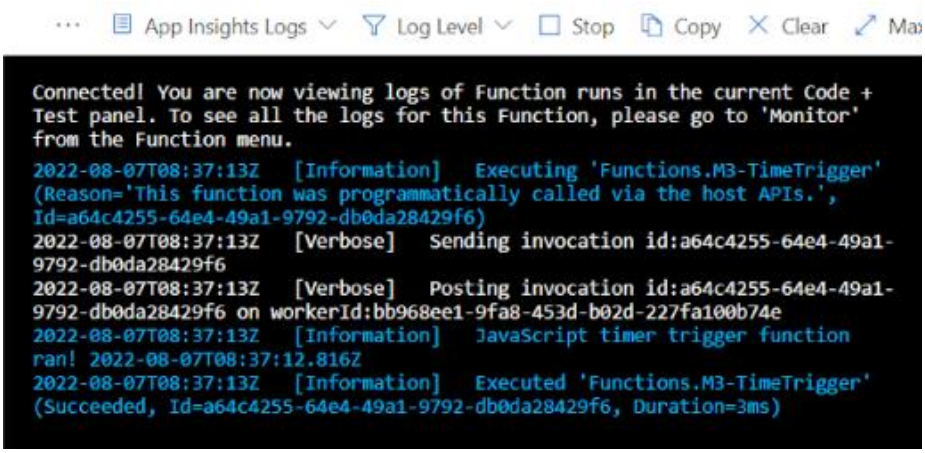
在 Timer Trigger 部分，新增 Timer Trigger 時，需要額外新增 CRON 排程指定觸發頻率。

CRON expression:

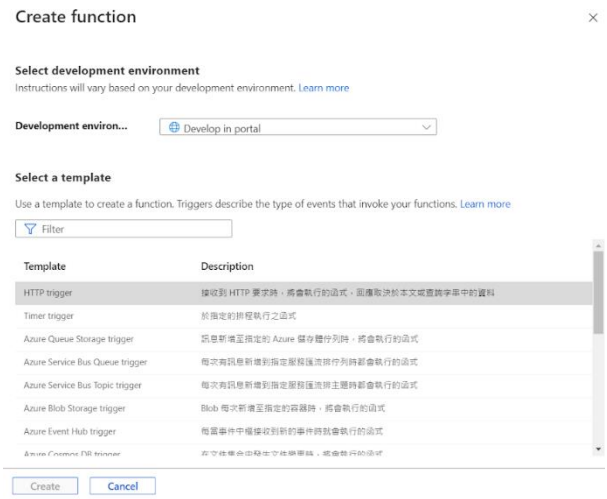
{second} {minute} {hour} {day} {month} {day-of-week}

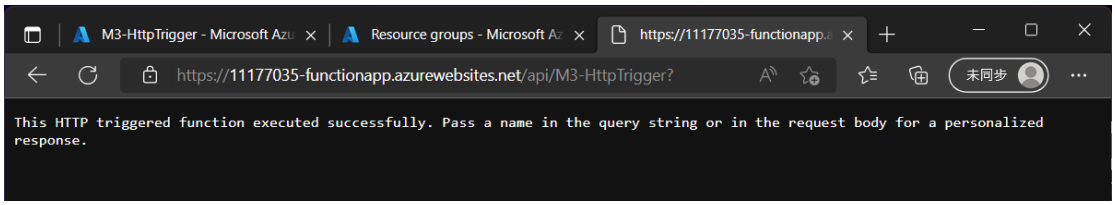


打開 Test/Run > Log 功能，查看 Function 每 20 秒觸發一次的結果。

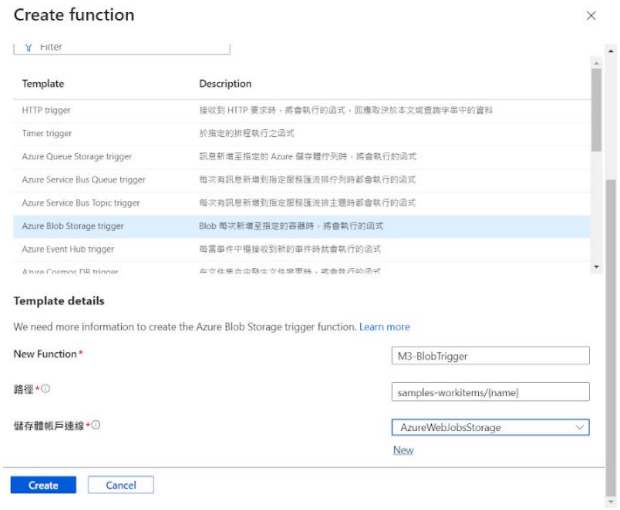


在 Http Trigger 部分，如同上一個 Module 新增一個 Http Trigger，並且簡單的使用 url 做 Http 觸發。

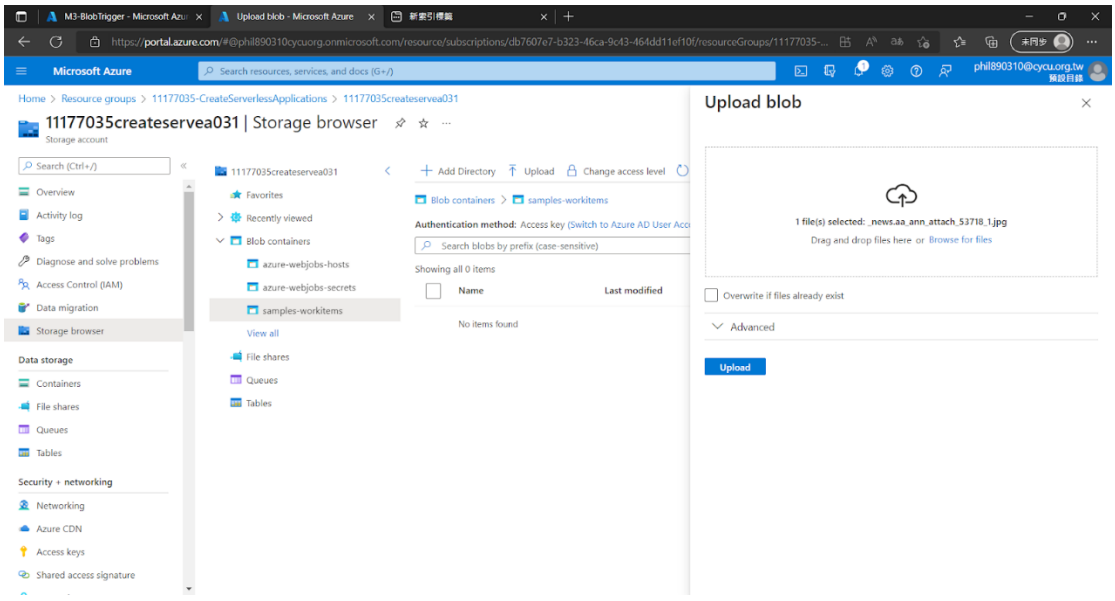




最後新增 Blob Trigger，新增過程需新增與 Blob Storage 的連線，並指定觸發 Container 路徑。

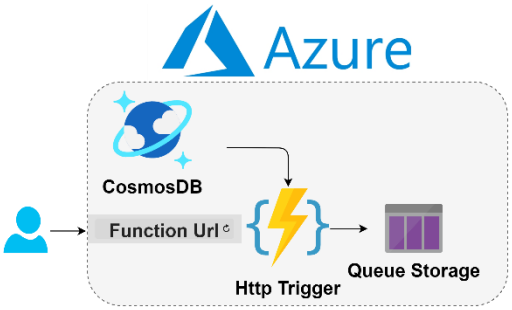


在 Blob 的指定 Container 當中隨意新增檔案，就能在 Azure Function 的 Filesystem Logs 當中觸發的結果，並且透過內容觀看新增的檔案 Metadata。

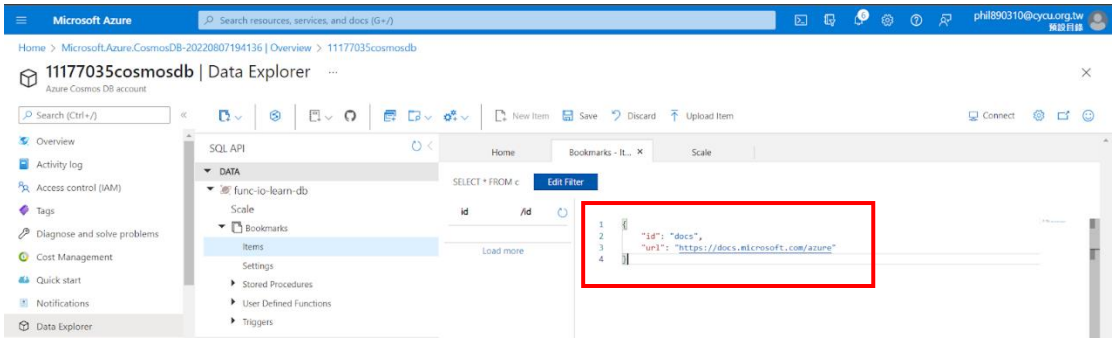


Module 5: Chain Azure Functions together using input and output bindings Create serverless logic with Azure Functions

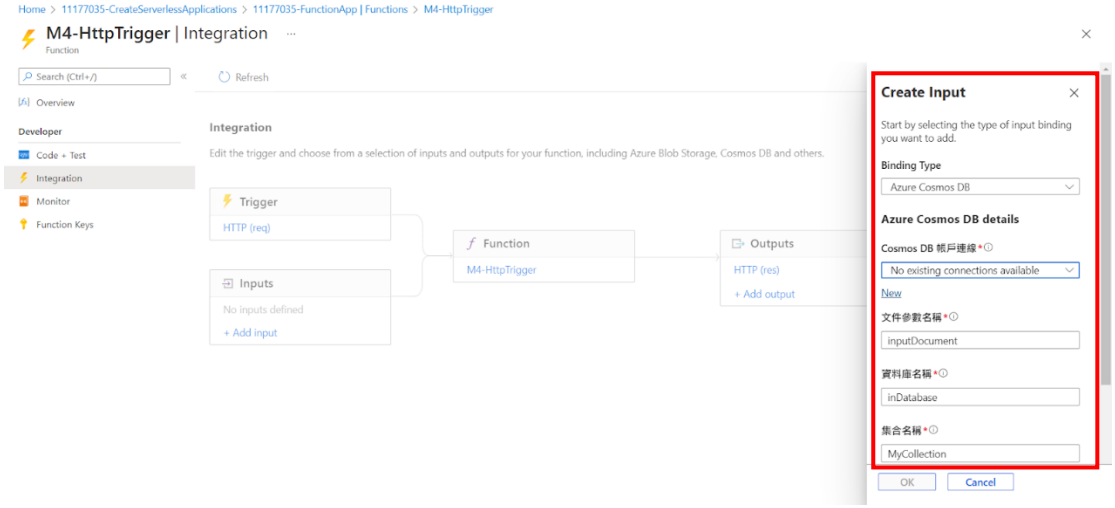
此 Module 只在教導 Azure Function 中的 Binding，並且設定 Input Binding 與 Output Binding 做示範。透過 Binding，可以使 Azure Function 執行時可以與更多的資源做連結，使用上更靈活。以此 Module 使用一個 Http Trigger 搭配 Get/Post 的方式查詢一個 CosmosDB 的資料，並將運算結果傳送至 Queue Storage。查詢 CosmosDB 資料與傳送結果的方式都是透過 Binding 定義其他資源與 Function 間觸發及結束的資料流。



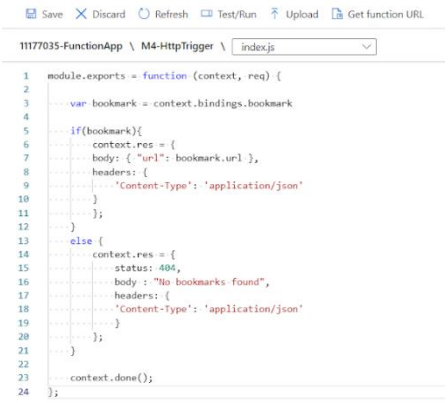
首先，創建一個 CosmosDB，並且在創立一個 Container，輸入一筆資料。



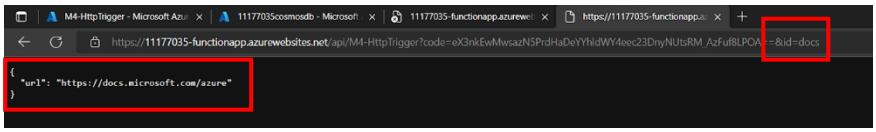
接續創立 Http Trigger，並且在 Input Binding 新增一個 CosmosDB 的連線，並且指定 Binding 的資料位置。



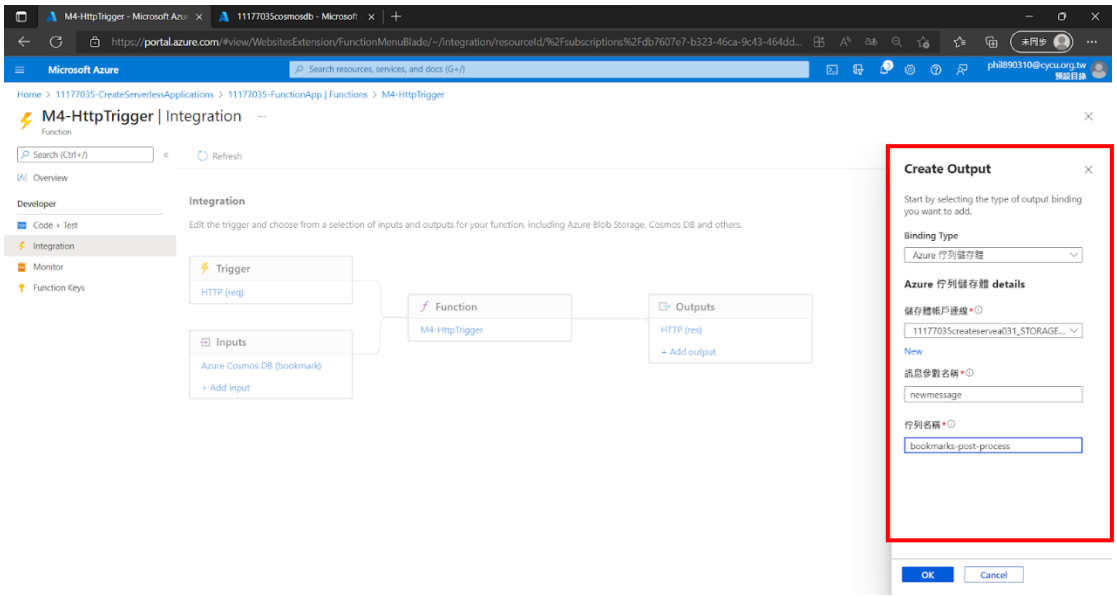
修改 Function 的邏輯內容，判斷 Get/Post 的查詢是否存在於 CosmosDB 的 Input Binding 資料中，若存在則顯示 CosmosDB 中其餘 url 資訊。



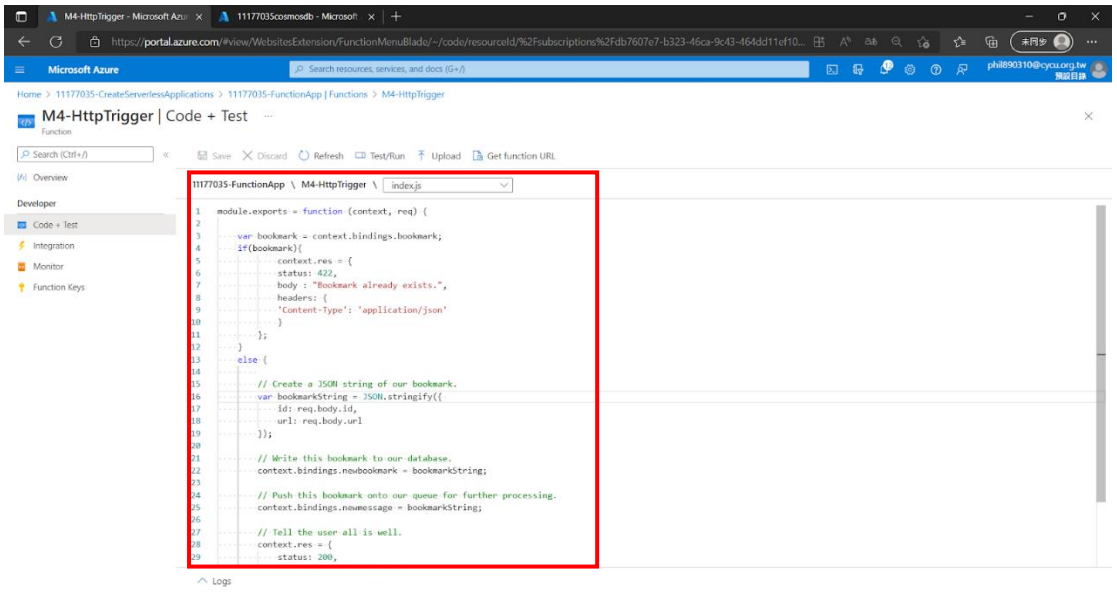
簡單做 Http Trigger 測試，使用 GET 查詢 id=docs 的資料，在 Function 內成功透過 Input Binding 查詢到 id=docs 這筆資料，並將結果回傳。



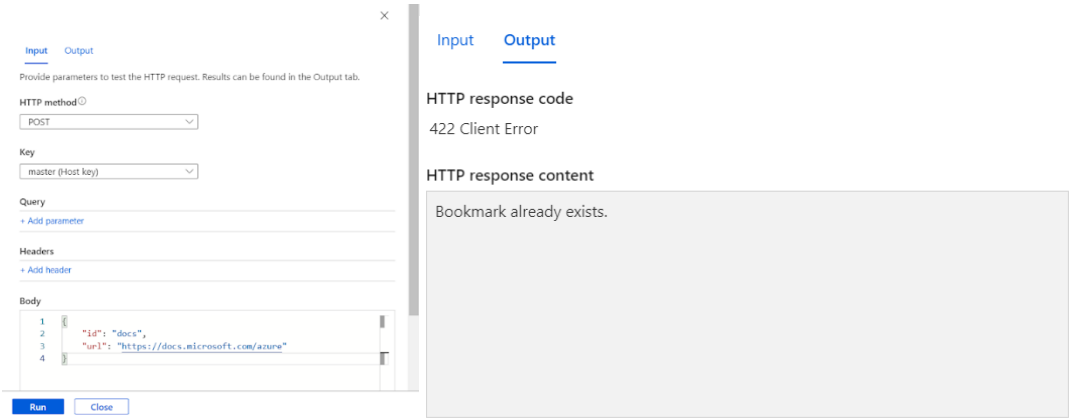
接續定義 Output Binding，新增 Queue Storage 的連線，並指定 Output Binding 的 Queue 名稱。



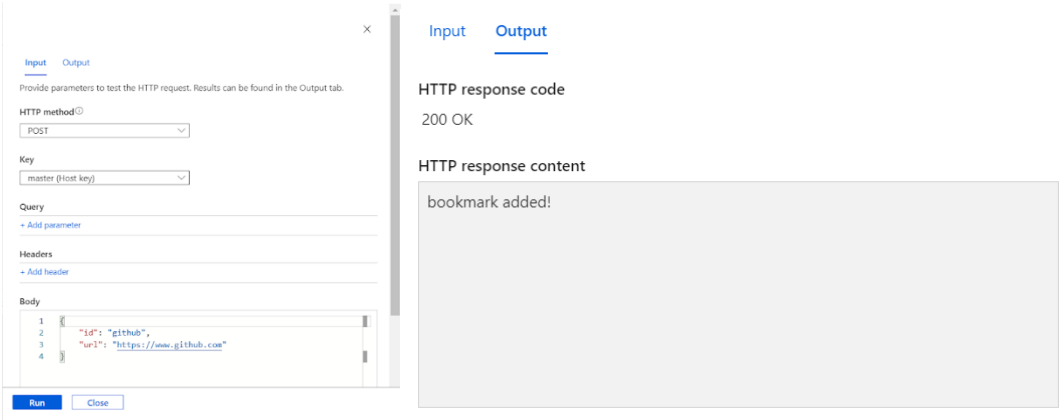
修改 Function 邏輯程式碼，首先 GET/POST 的資料會與 Input Binding 的資料做查詢，若 POST 的資料不存在 Input Binding 的資料，則將 POST 的資料先新增進 CosmosDB，再透過 Output Binding 寫至 Queue Storage 中。



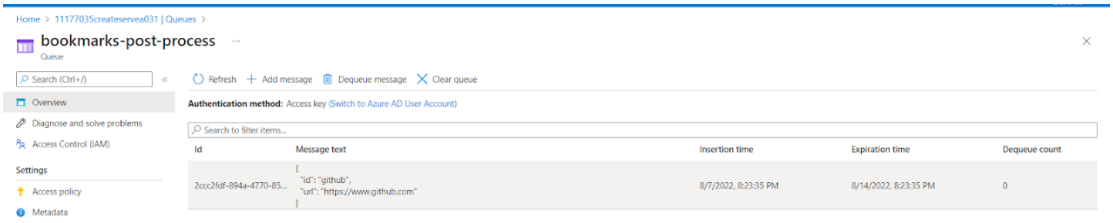
最後進行測試，POST 一筆已經存在的資料，顯示 Bookmark 已經存在。



另一個測試 POST 不存在的資料，顯示 Bookmark 添加至 CosmosDB 中。

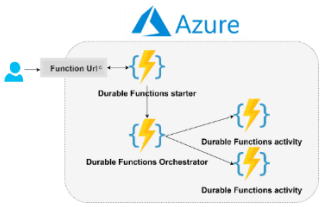


進階查看 Queue Storage，確認 Output Binding 正確將資料寫入 Queue Storage 中。

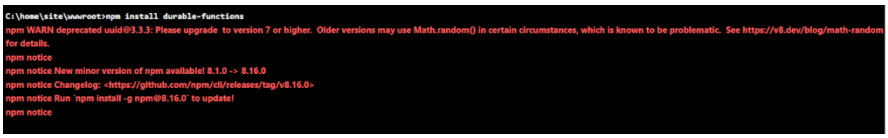


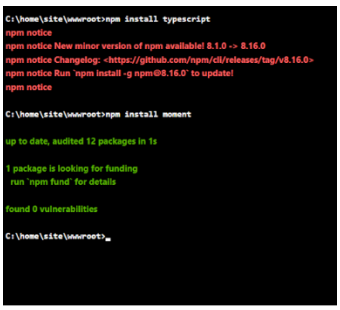
Module 6: Create a long-running serverless workflow with Durable Functions

Durable Function 是 Azure Function 的進階功能，使得 Serverless 從 Stateless 的狀態因 Durable Function 而擁有 Stateful 的狀態。Durable Function 可以用於執行長時間的工作，使得應用更具有彈性，避免工作時間過長導致 Http Timeout。

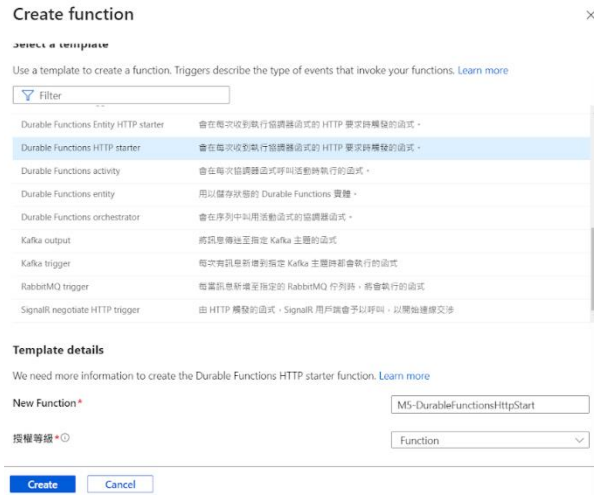


首先須安裝所需套件，開啟 Azure Function 的 Console，安裝 durable-functions、typescript、moment。

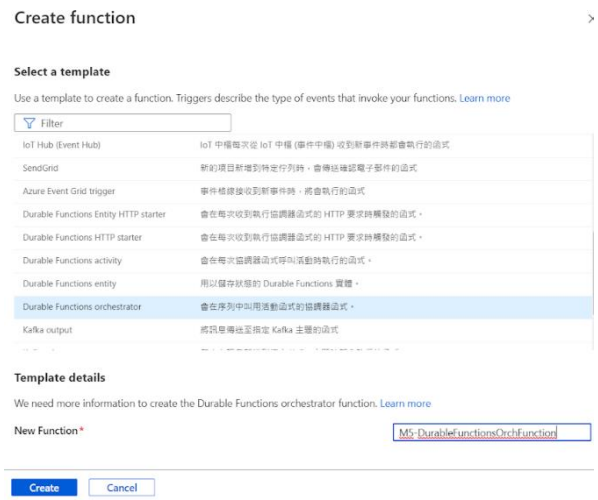




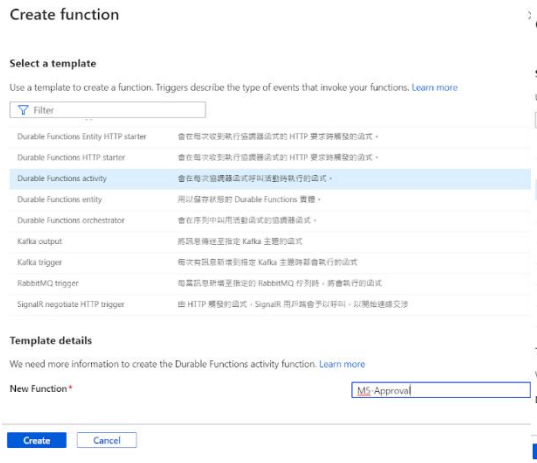
新增一個 Durable Http starter，此 starter 作為 Durable Function 入口網站。

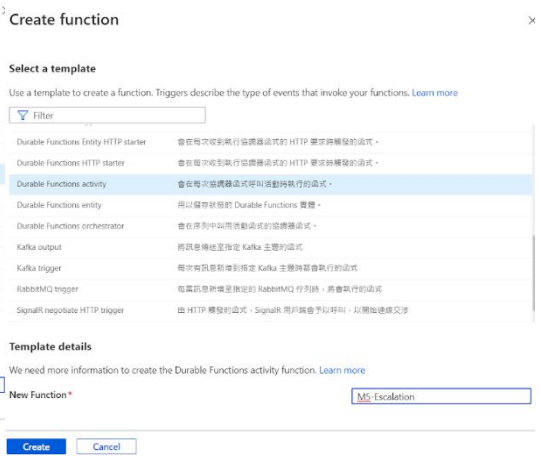


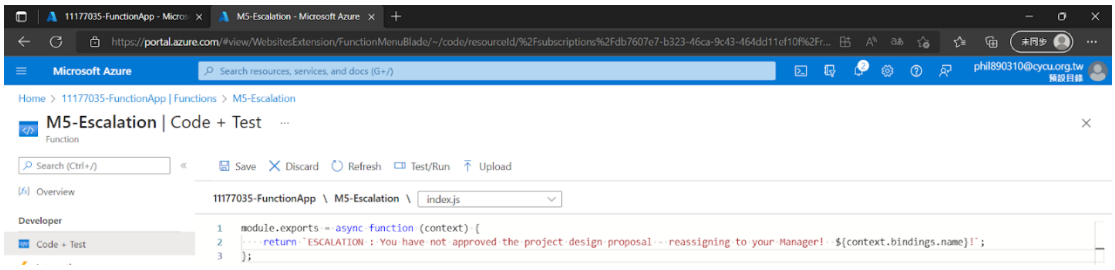
新增一個 Durable orchestrator，此 orchestrator 作為 Durable Function 主要調度的物件，內部明確定義 Function 的動作。



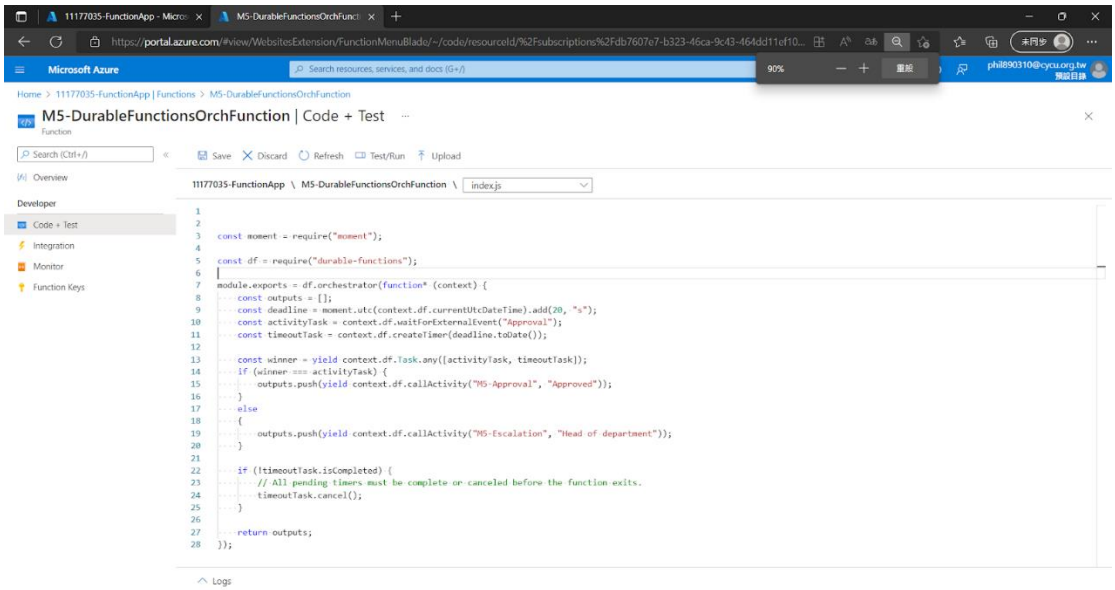
新增 2 個 Activity 作為 Durable Function 的基本工作單位，可供 orchestrator 進行調度。



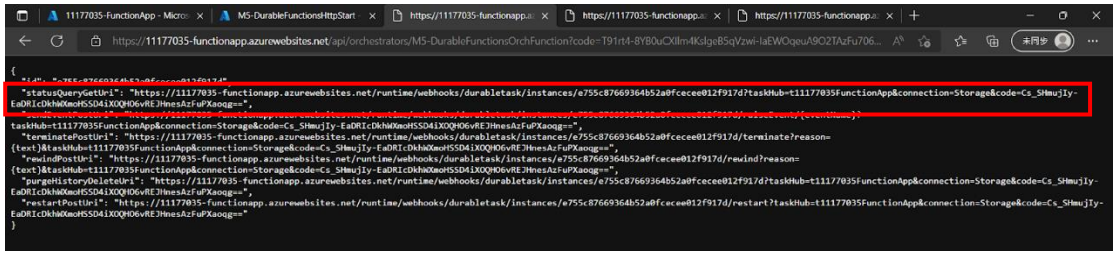




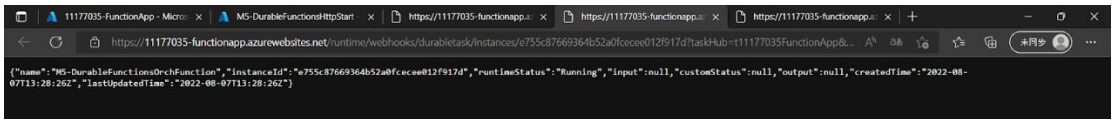
修改 orchestrator 的程式碼，在程式碼中使用 moment library 紀錄時間。在 orchestrator 生成後的前 20 秒，會執行”M5-Approval”的 Activity；20 秒之後執行”M5-Escalation”的 Activity。

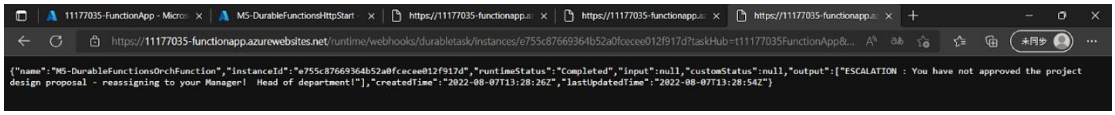


執行 Http Starter 開啟 Durable Function 入口，選 statusQueryGetUri 的 url 觸發 Durable Function。



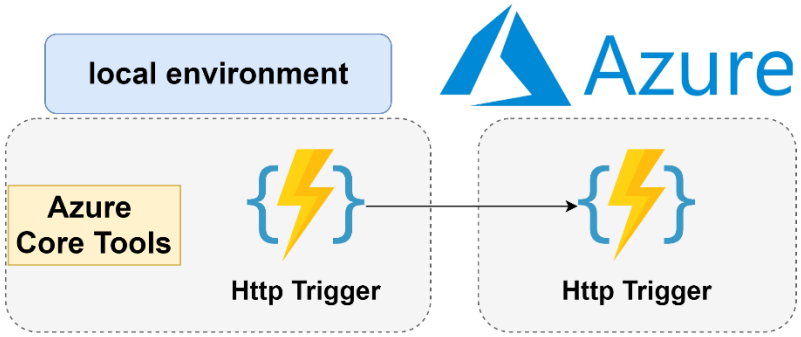
第一次開啟 url，result 顯示 null，；但在 20 秒後，執行 ESCALATION 的 Activity。





Module 7: Develop, test, and publish Azure Functions by using Azure Functions Core Tools

Azure Function 除了能在 Azure Portal 進行開發部署，也能在本地環境執行，尤其是較特殊的語言，例如 Python，Azure Portal 無法支援 Python 的開發，因此需要 Developer 在本地環境開發完成後再搭配 IDE 部署至 Azure 中。



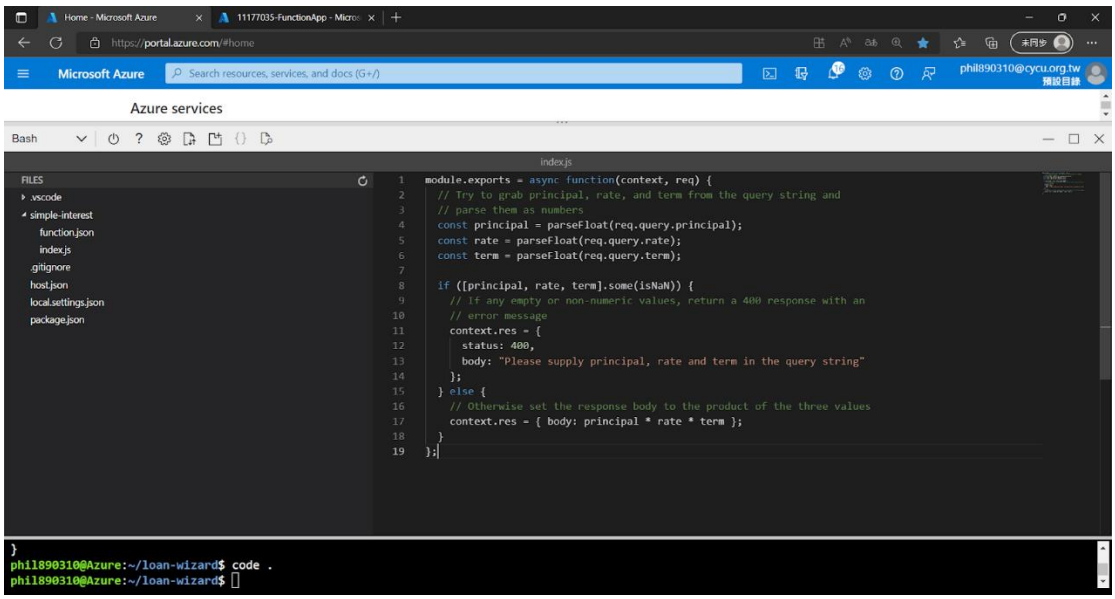
初始化 Azure Function，此指令會生成 host.json、local.settings.json、.gitignore 等環境設定檔案，並且設定開發語言。

$ func init

使用 func new 開啟一個 Function，並設定觸發方式及名稱。

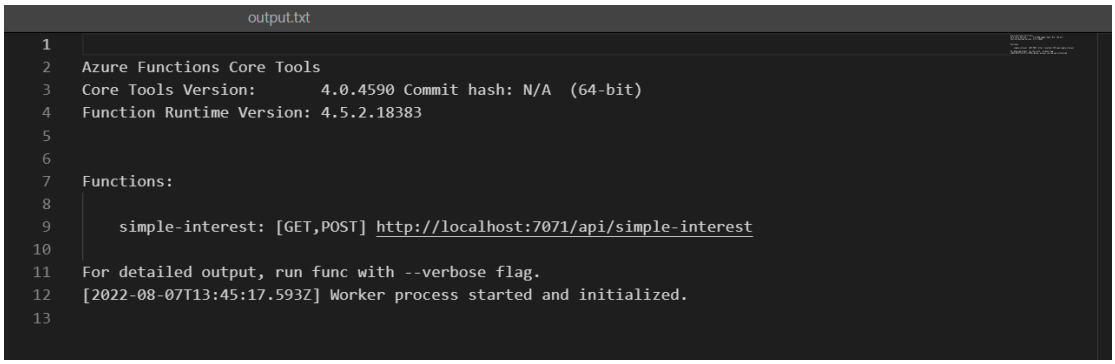
$ func new

替代 Function 執行程式碼：程式會抓取 GET/POST 傳入的 3 個 Key 進行相乘。



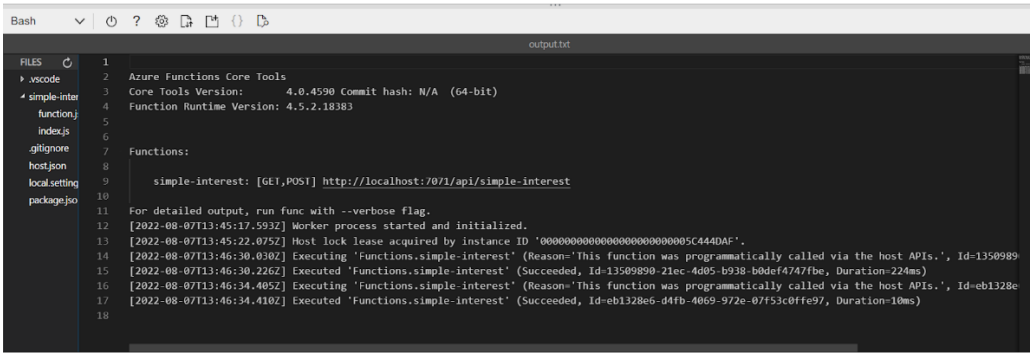
使用 func start &> ~/output.txt &指令，使得程式在背景執行，並將 log 輸出至 output.txt，可隨時查看 output.txt 得到系統 log。

```
phil890310@Azure:~/loan-wizard$ func start &> ~/output.txt &
[1] 810
```



使用 curl 進行 Http Trigger，也可搭配參數進行運算。事後也可至 oputput.txt 查看 log。

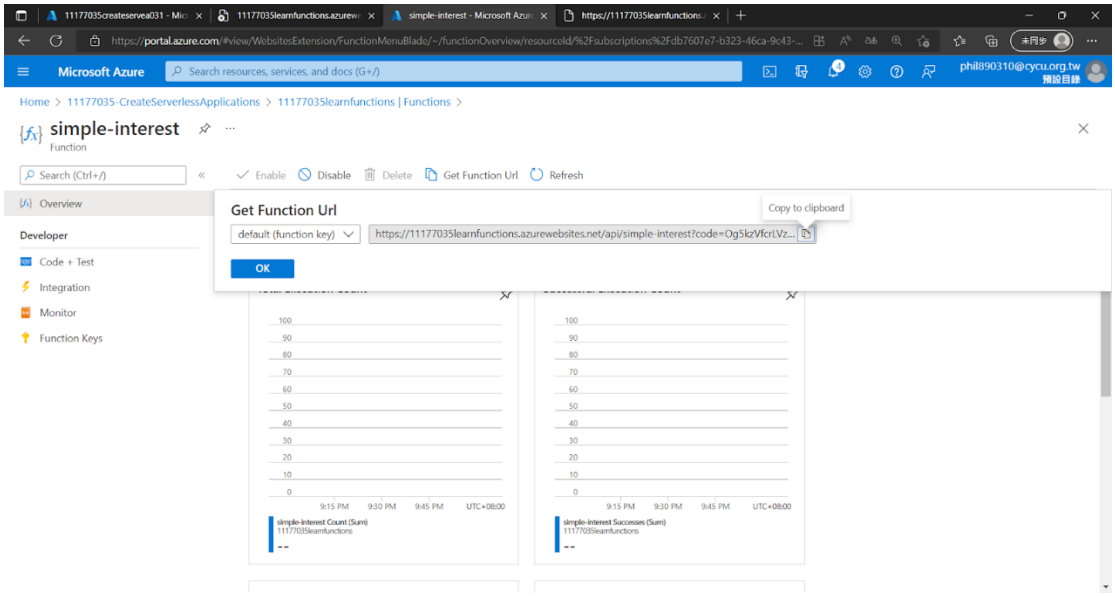
```
phil890310@Azure:~/loan-wizard$ curl "http://localhost:7071/api/simple-interest" -w "\n"
Please supply principal, rate and term in the query string
phil890310@Azure:~/loan-wizard$ curl "http://localhost:7071/api/simple-interest?principal=5000&rate=.035&term=36" -w "\n"
6300.000000000001
```



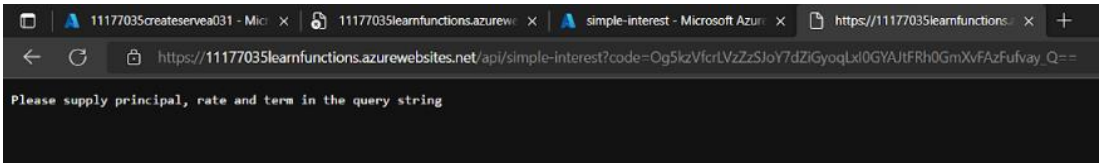
使用 func azure functionapp publish "$FUNCTIONAPP" –force 將本地開發的 azure function 部署至 Azure 雲端。

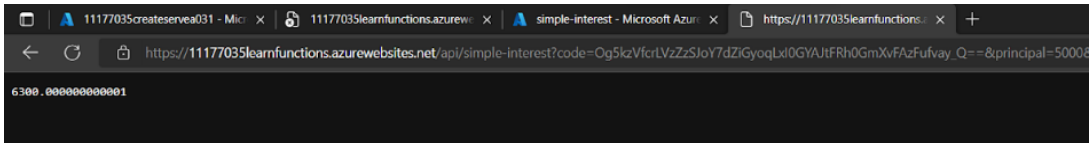
```
phil890310@Azure:~/loan-wizard$ func azure functionapp publish "$FUNCTIONAPP" --force
Setting Functions site property 'netFrameworkVersion' to 'v6.0'
Getting site publishing info...
Creating archive for current directory...
Uploading 1.38 KB [########################################################################################]
Upload completed successfully.
Deployment completed successfully.
Syncing triggers...
Syncing triggers...
Syncing triggers...
Syncing triggers...
Functions in 11177035learnfunctions:
    simple-interest - [httpTrigger]
        Invoke url: https://11177035learnfunctions.azurewebsites.net/api/simple-interest
```

Azure 雲端即可查看部署結果。



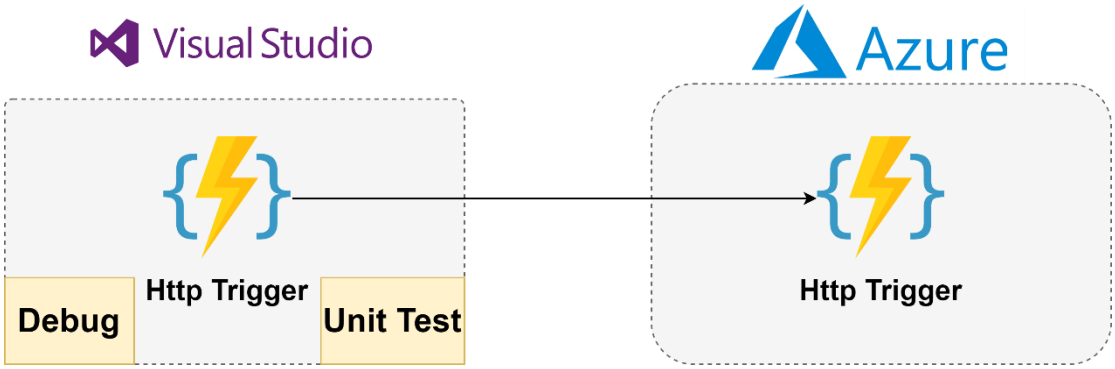
雲端上的 Function 能與本地開發的 Function 擁有一樣的運作結果。



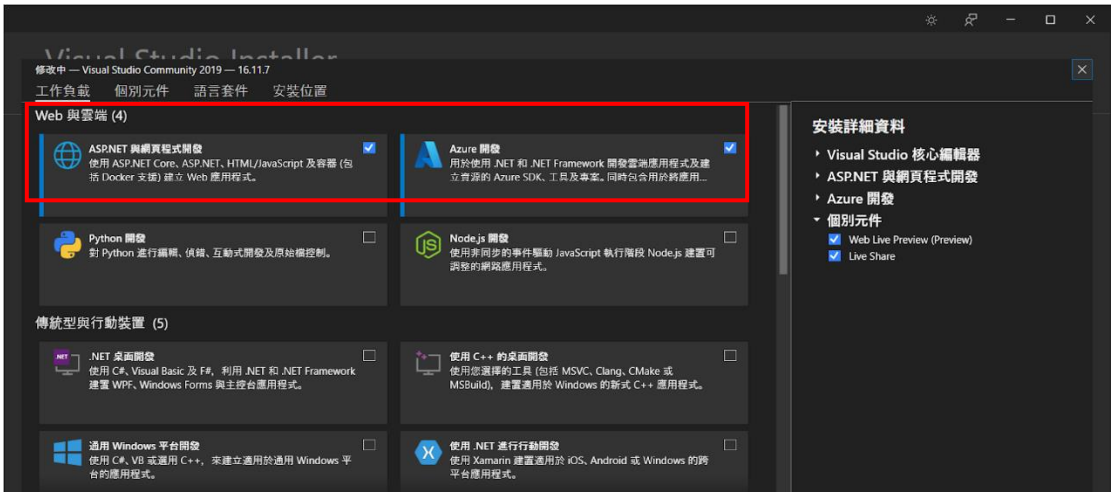


Module 8: Develop, test, and deploy an Azure Function with Visual Studio

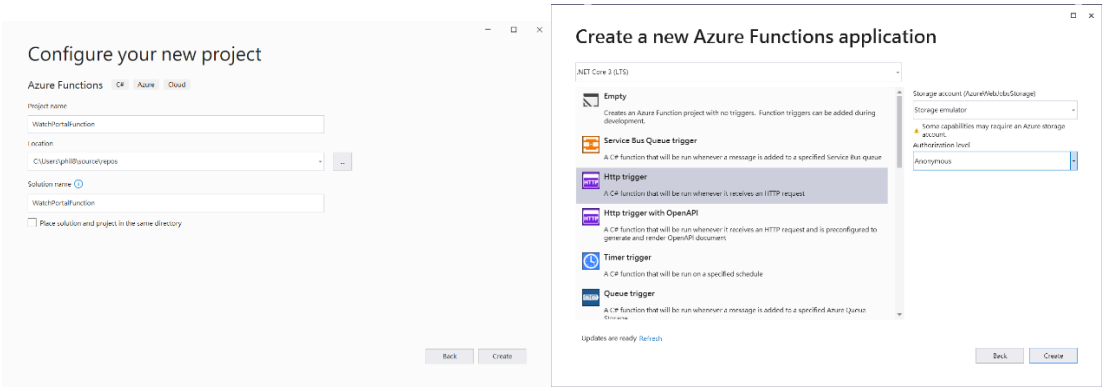
本模組重點重點教導如何使用 Visual Studio 的開發環境做 Azure Function 的開發，除了 Create Function、Publish/Deploy 功能，還教導 BreakPoint、Unit Test 的功能。



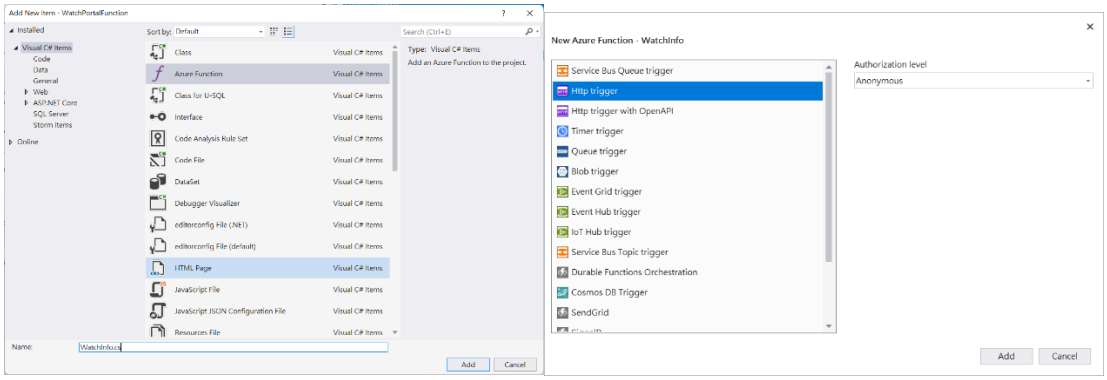
此模組使用 Visual Studio 進行 Azure Function 的開發，需要安裝以下套件。



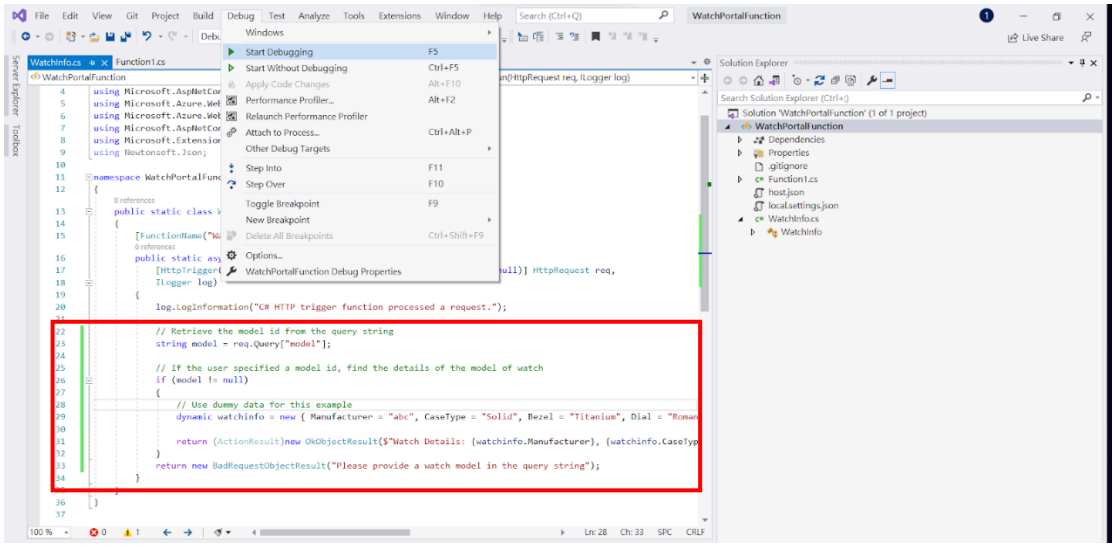
創立一個 Project，並創立第一個 Function（Http Trigger），名為 Function1。

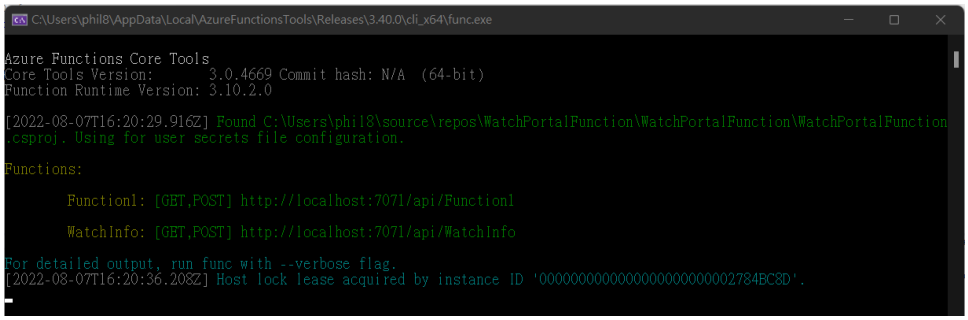


接續創立第二個 Function（Http Trigger），名為 WatchInfo。

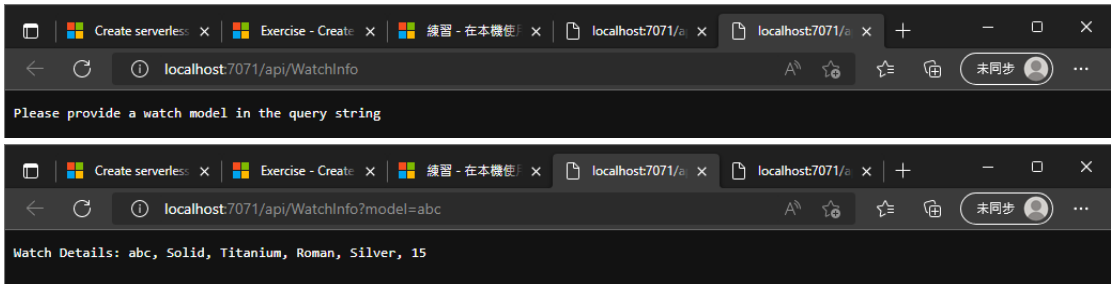


修改程式碼，程式會根據 GET/POST 引入的變數"model"回傳不同的結果。並使用 Debug 啟動 Azure Function。

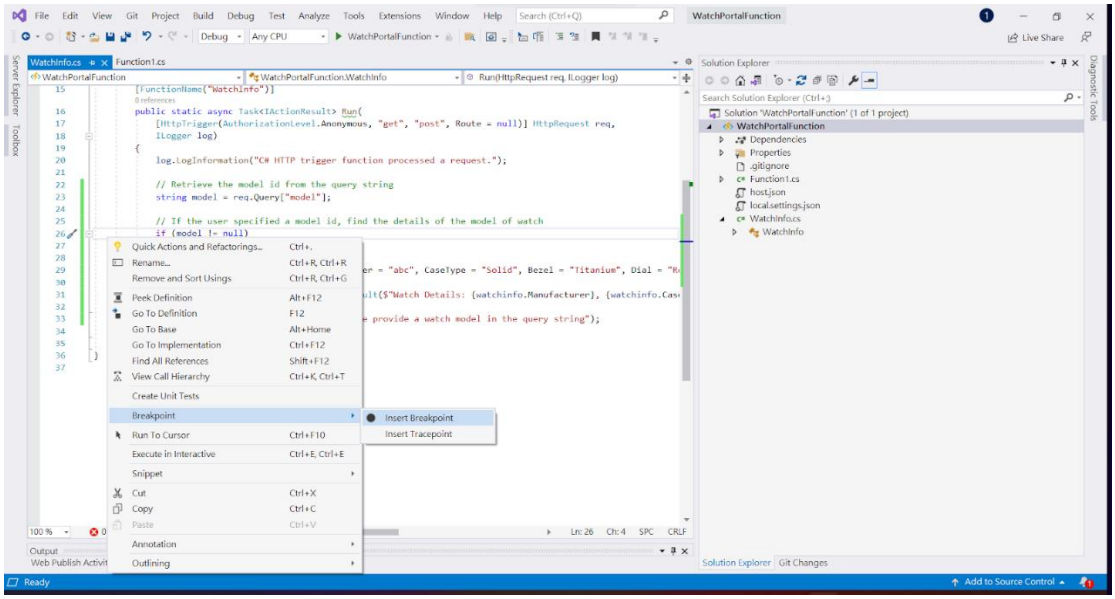




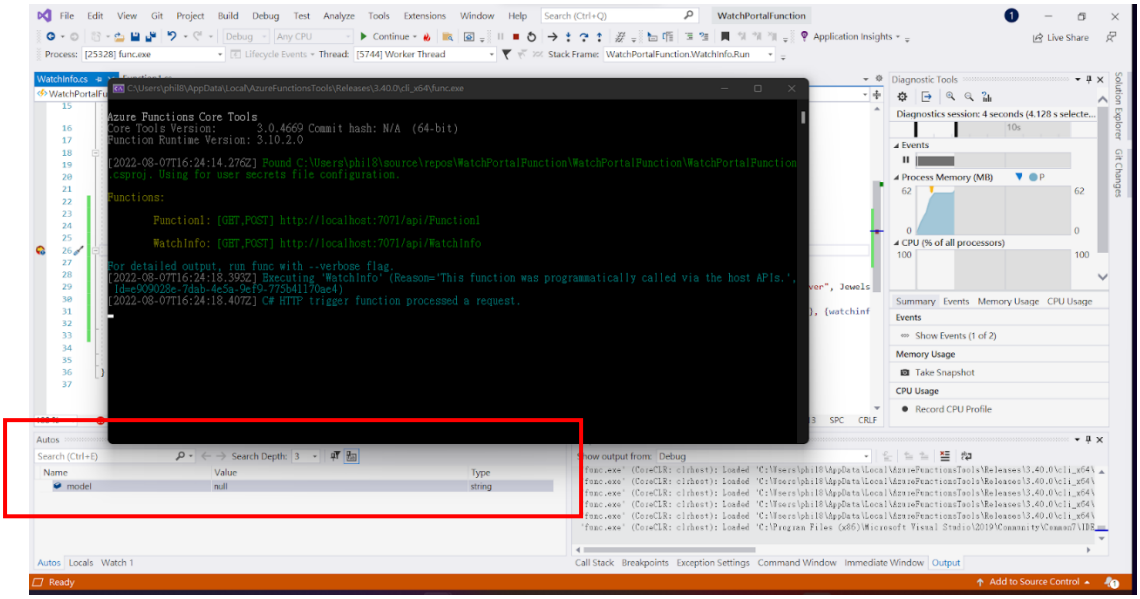
Http Trigger 結果如下圖所示，程式根據 GET/POST 的 model 變數決定輸出內容。



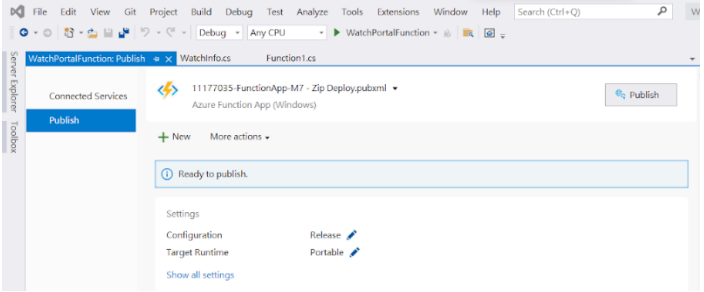
於 Visual Studio 中，可以設定 Breakpoint，可以逐行設定程式中斷點進入 Debugging Mode，查看程式執行當下各物件的狀態。



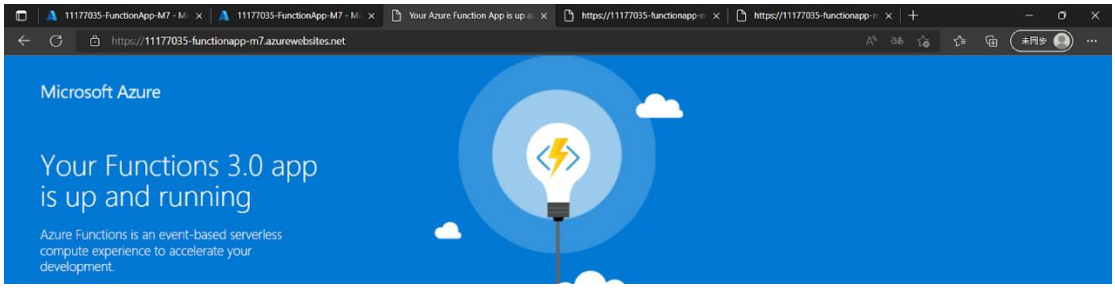
如 model 變數，在中斷點下能偵查當下的值。

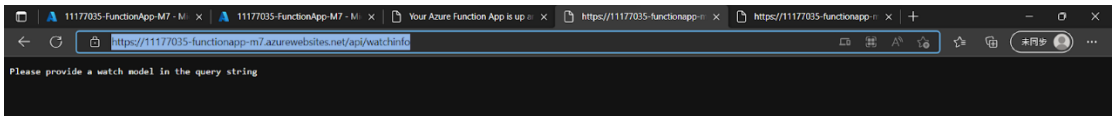


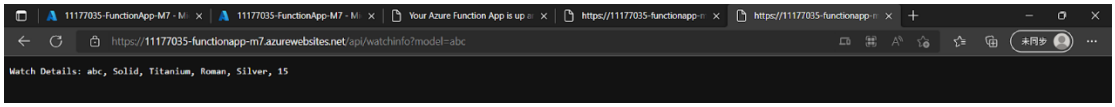
最後也能透過 IDE 佈署 Azure Function 至雲端。



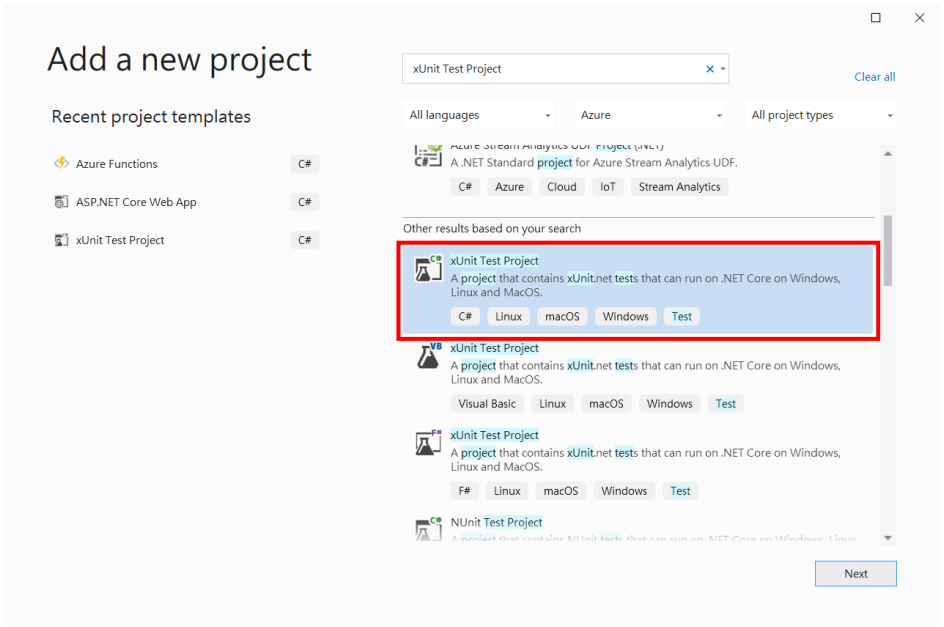
部署結果如下，與本地開發的功能在雲端再度呈現。



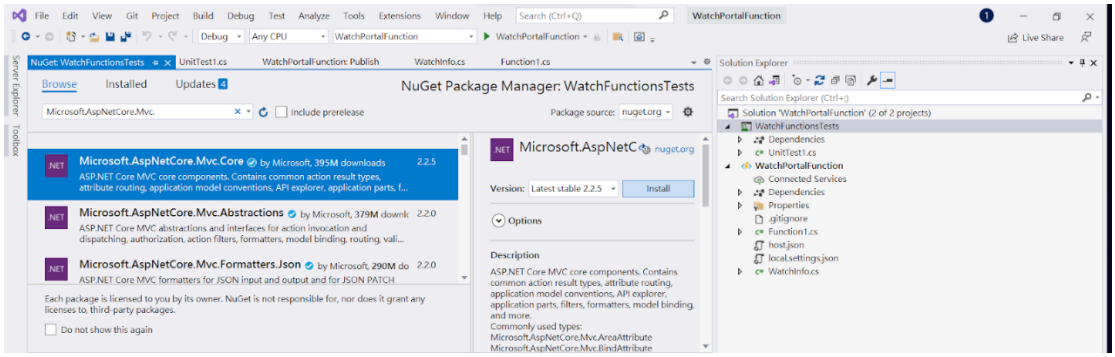




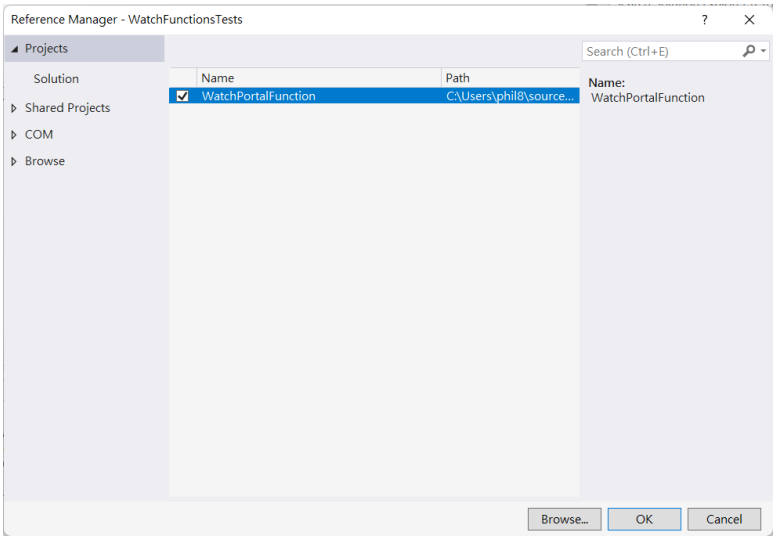
創立一個 Project 進行 Unit Test。



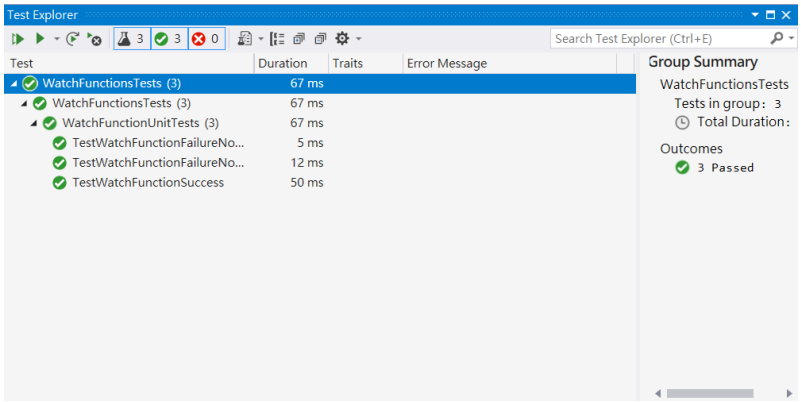
透過 NuGet，安裝 MVC 套件。



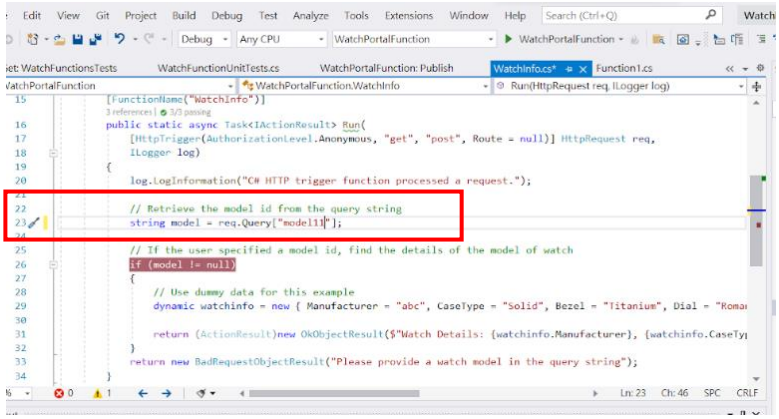
將 WatchPortalFunction 的 Project 新增至 Unit Test 的 Project Dependency，

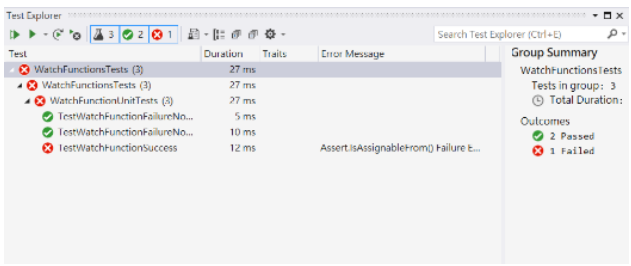


進行正常 Unit Test 的結果。



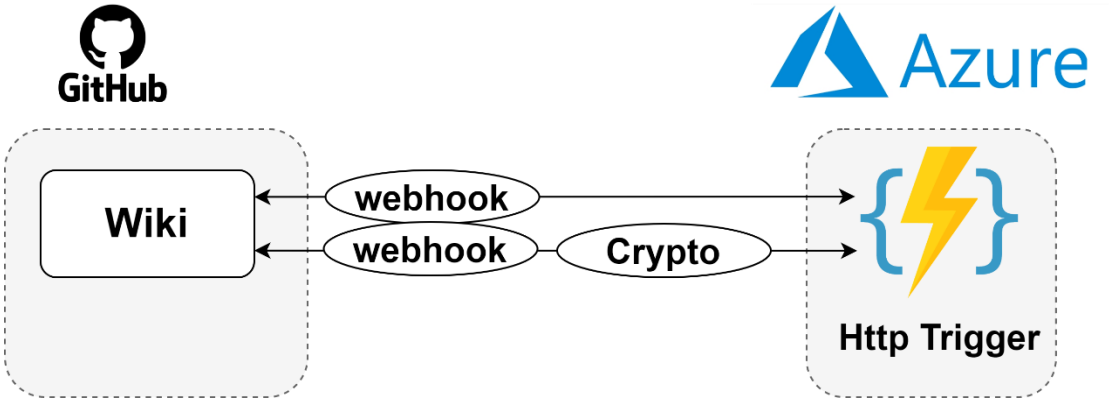
接著故意修改錯誤的程式碼，模擬開發錯誤的情境。



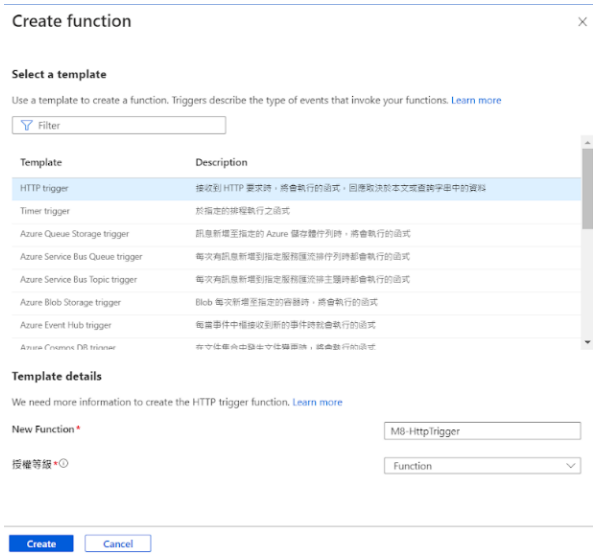


Module 9: Monitor GitHub events by using a webhook with Azure Functions

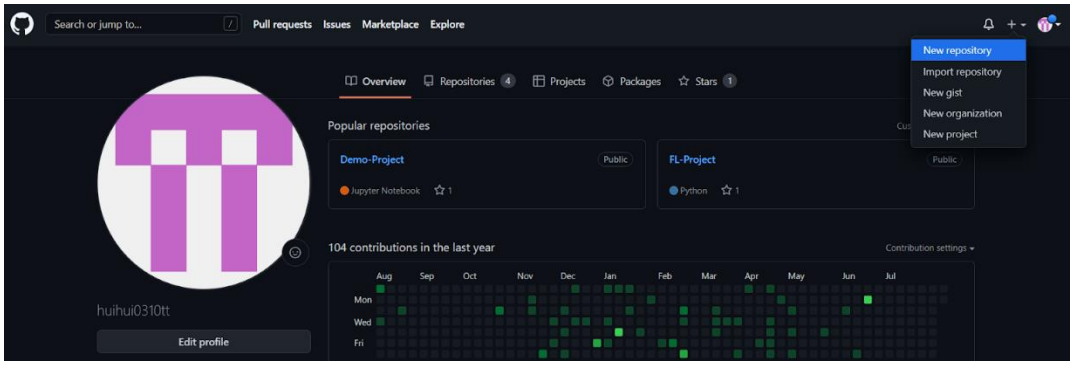
此模組重點在教導如何使用 Webhook 觸發，並進一步與 Azure Function 做配合。



創建一個 Http Trigger 供 Github Webhook 做觸發。

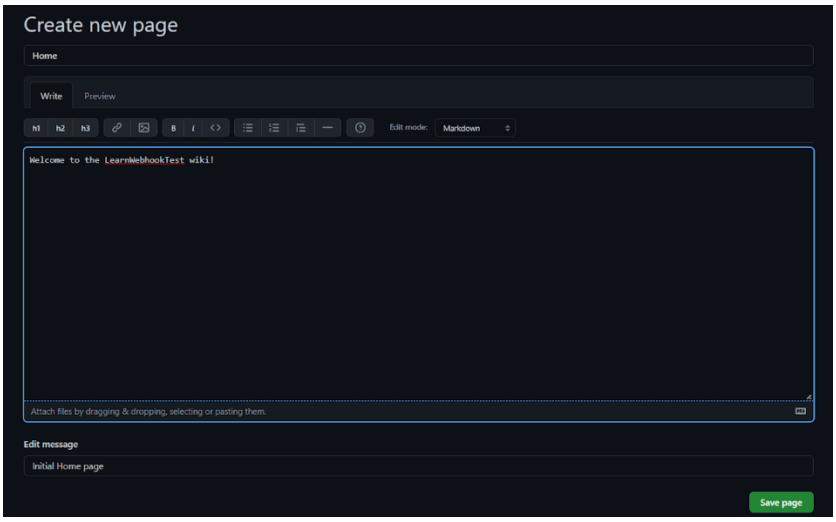


在 Github，創建一個倉儲。

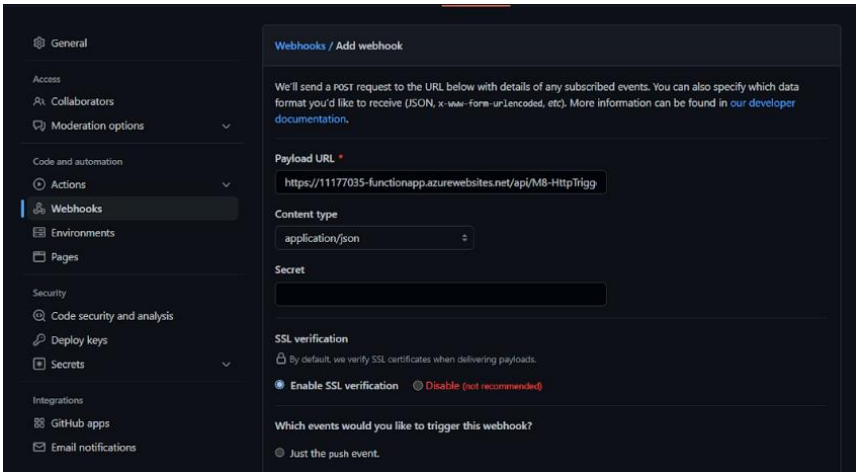


‘

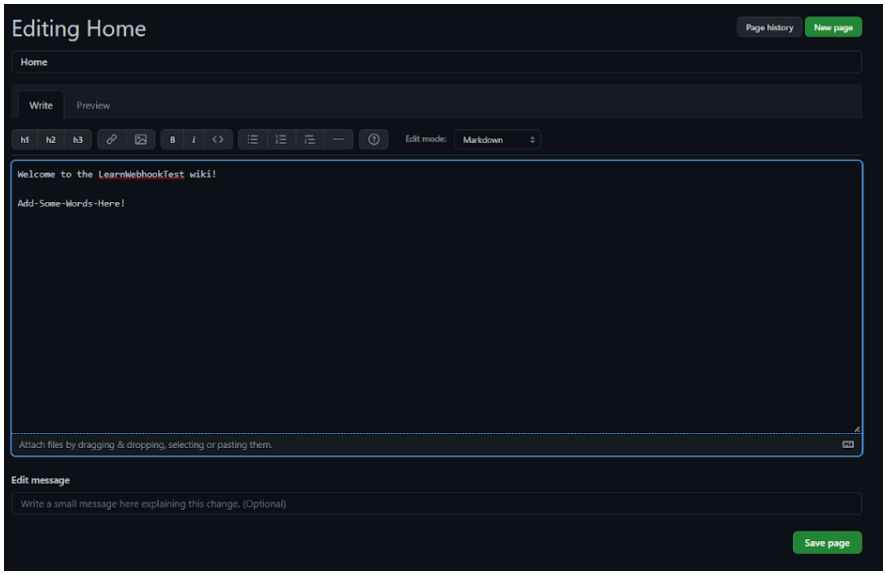
開啟一個 Wiki 頁面，作為觸發 Http 的來源。



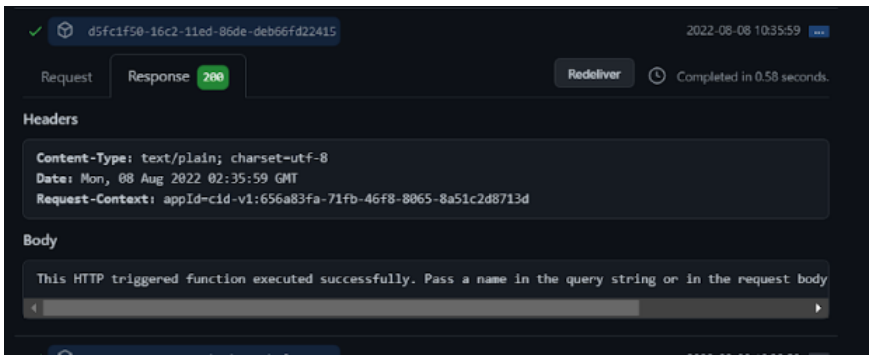
設定倉儲的 Webhook（無加密），並將 Payload URL 設定為 Azure Function 的 http url。



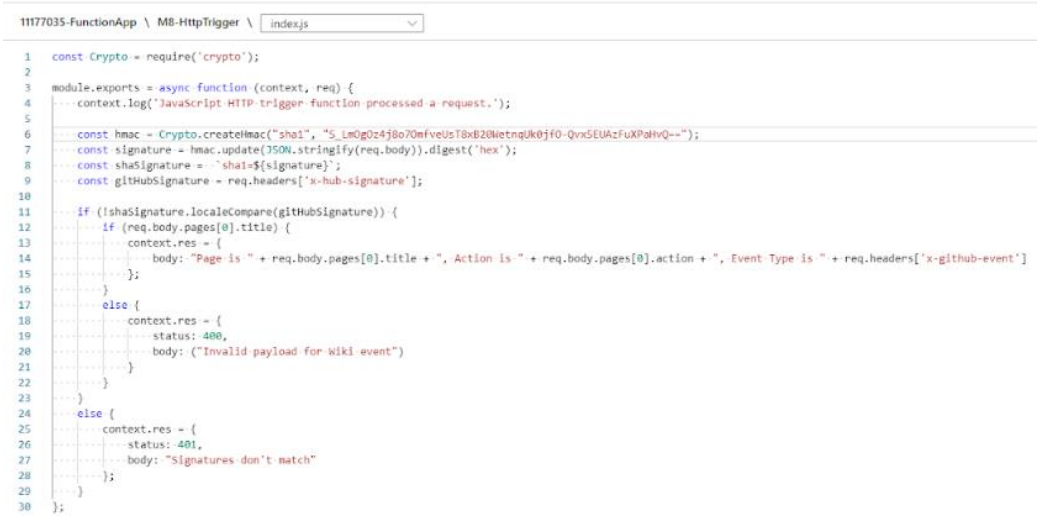
修改 Wiki 得觸發第一次的 Webhook。



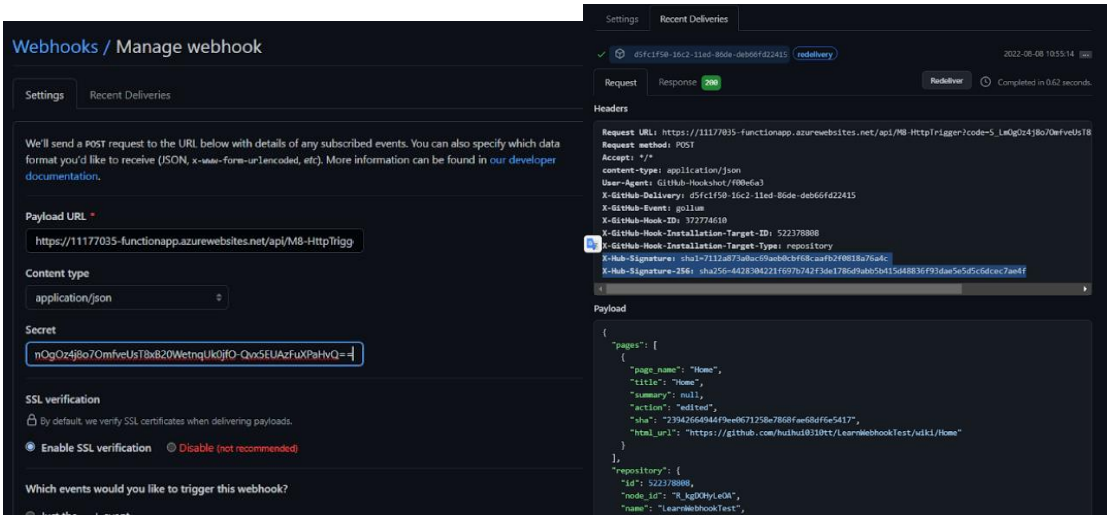
Webhook 紀錄中查詢過往觸發紀錄，紀錄顯示 Http200，正確觸發。



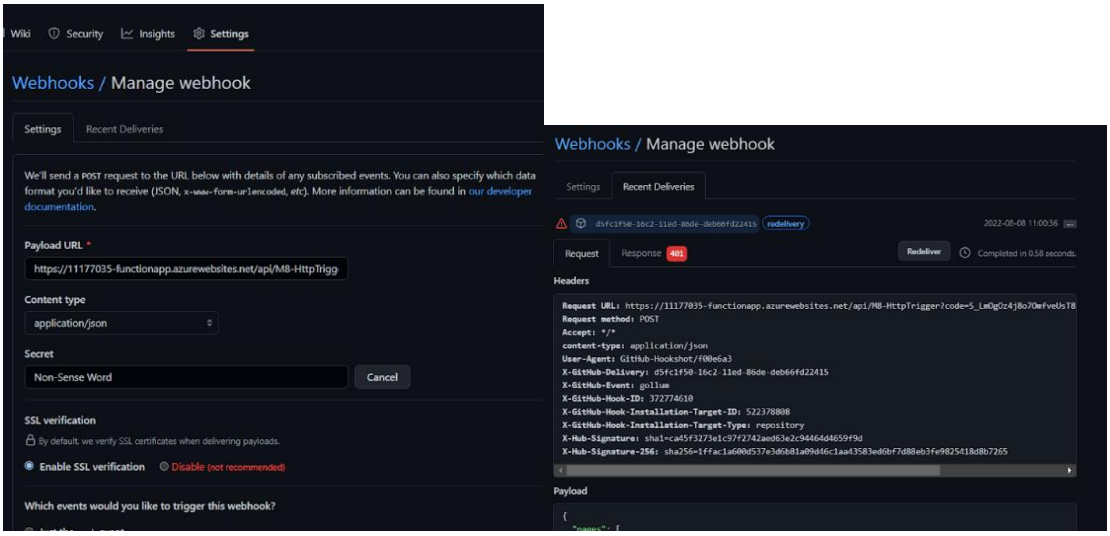
修改 Azure Function 程式內容，使用 Private Key（Azure Function 的 Function Key）做驗證，確定發出 Trigger 的來源是受信任的對象。



在另一端，Webhook 設定 Secret（Azure Function 的 Function Key）對內容作簽名，Azure Function 就能根據內容驗證。

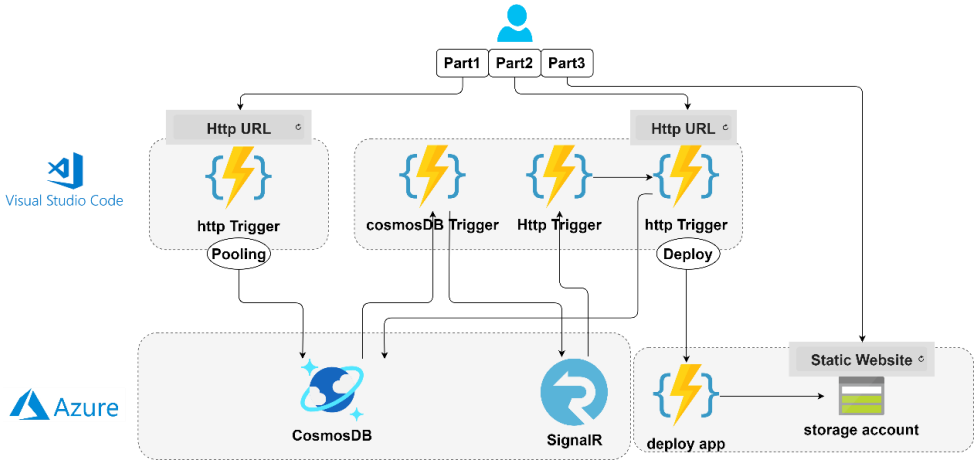


驗證 Secret 的正確性，測試錯誤的 Secret 將得到錯誤的 Response。



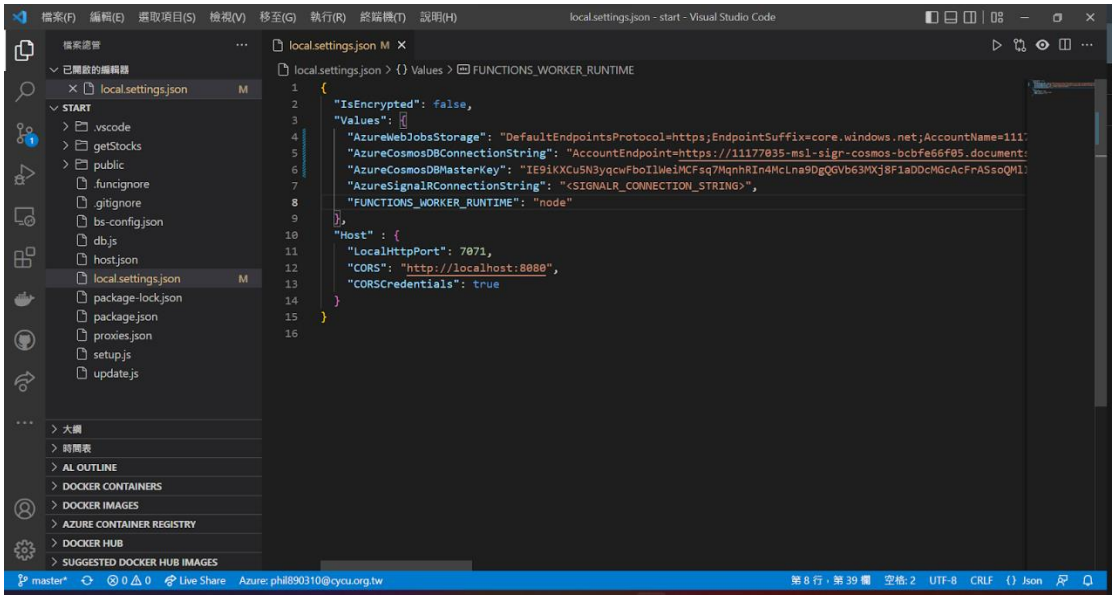
Module 10: Enable automatic updates in a web application using Azure Functions and SignalR Service

SignalR [1]是一個 RPC 的 Implement 方式，使得需要頻繁更新即時資訊的 WebApp 能使用 SignalR 進行 WebSocket。在此模組中，使用股票查詢的例子，股票的資訊會呈現在 FunctionApp 的 URL 中。在第一部分簡單透過 Pooling 的方式進行，每隔一段時間查詢資料庫中股票的資訊；在第二部分加入了 SignalR 的方式能使得資料庫的系統更新才通知 WebApp 更新資料；最後把本地資源部署至雲端，再透過 Storage Account 中的 Static Website 查詢同樣的股票資訊。

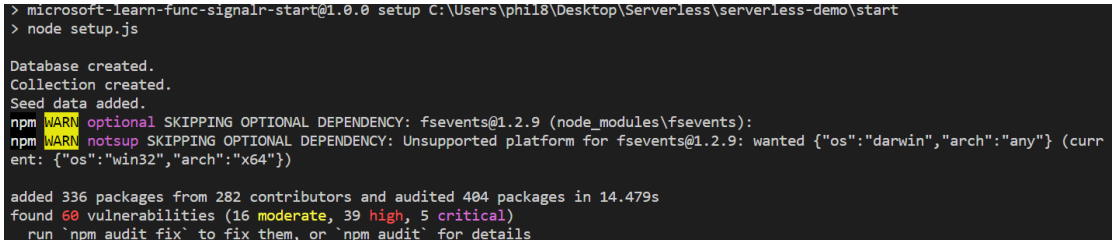


首先，創立以下資源：

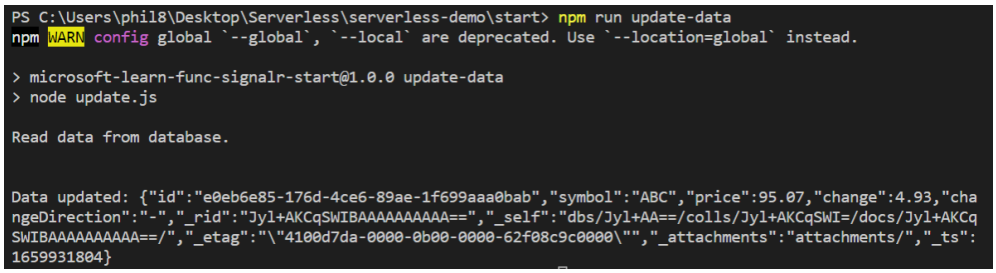
1. Azure Storage Account
2. Azure CosmosDB
3. 下載範例程式碼，將 local.settings.json 的 Connection String 填入。
   範例程式中會使用 Pooling 的方式定時向資料庫下載股票資料。



使用 npm install 將股票資訊寫入 DB 中。

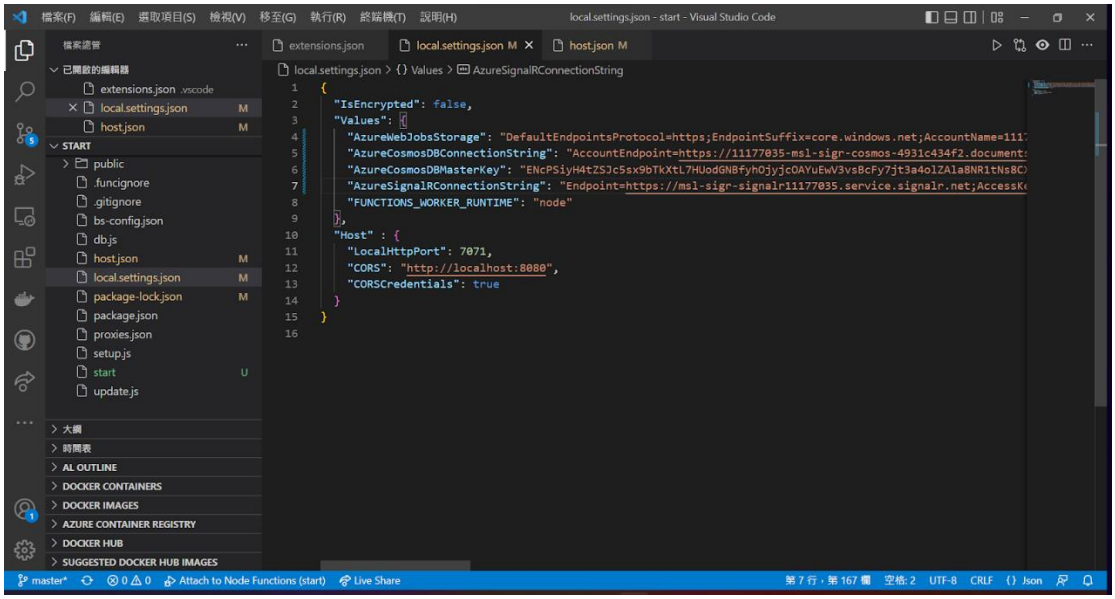


啟動 Azure Function，使用 npm run update-date 更新股票資料。

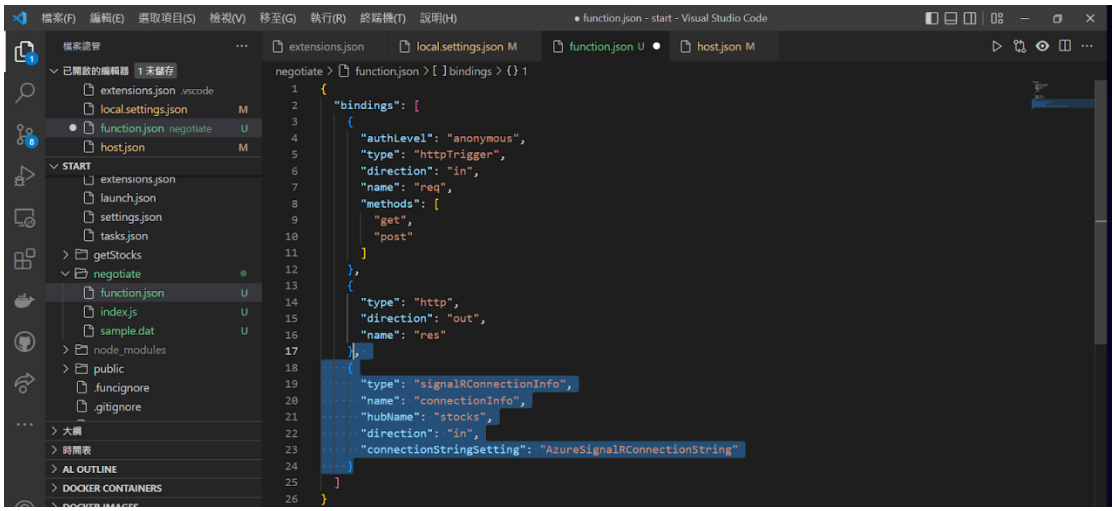


透過刷新網頁，在網頁中的資訊會因為股票資訊而有所更動。

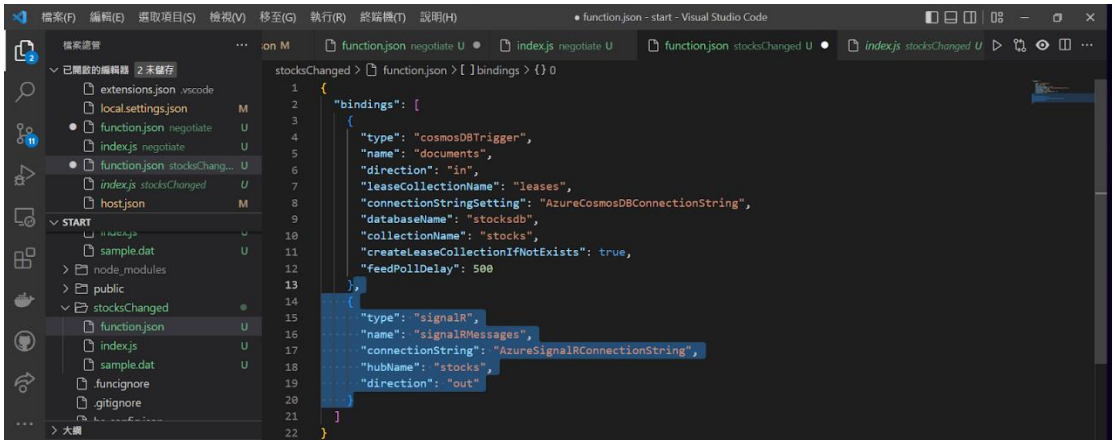
第二部分，創立 SignalR，增添 SignalR 的 ConnectionString 至 local.setting.json 設定檔中。



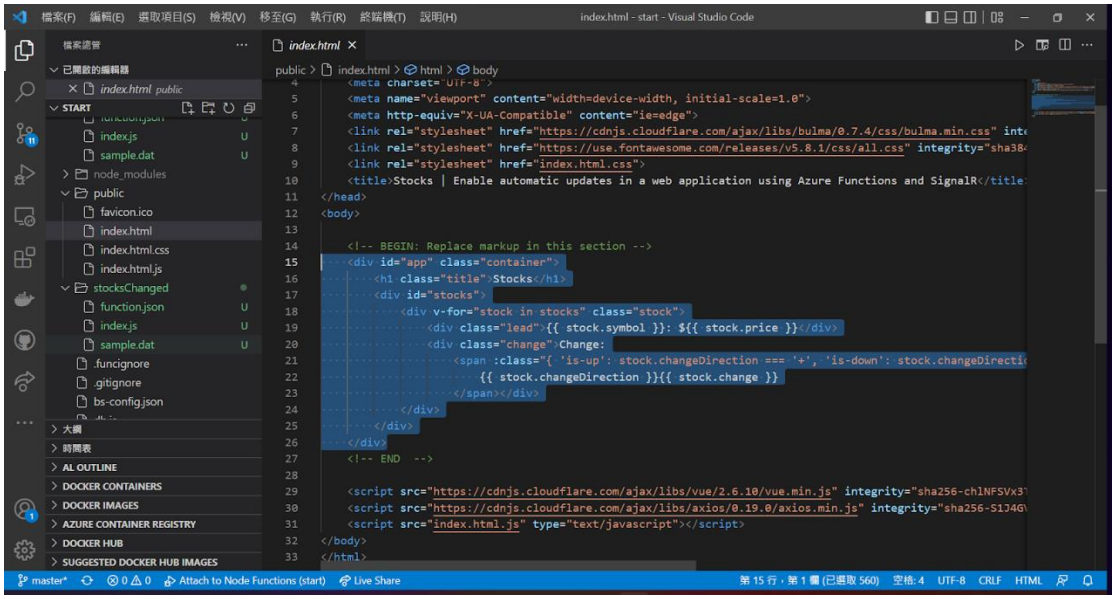
新增 Http Trigger，添增 SignalR 的 Input Binding。



新增 CosmosDB Trigger，添增 SignalR 的 Output Binding。

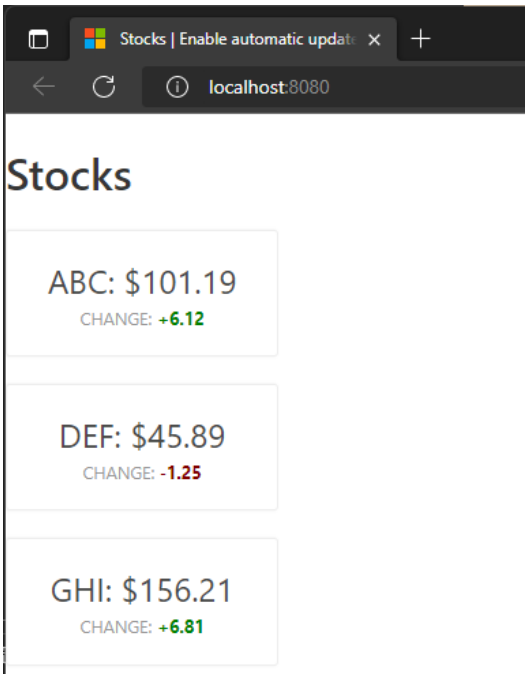


第二部分，創立 SignalR，增添 SignalR 的 ConnectionString 至 local.setting.json 設定檔中。

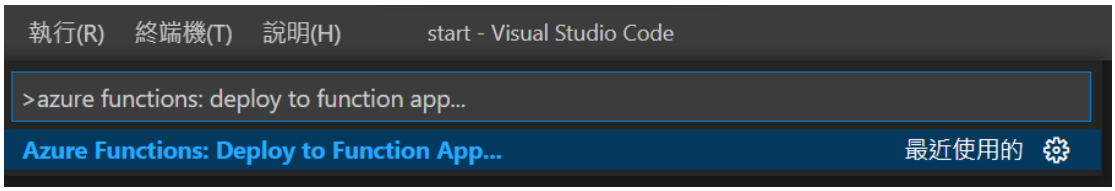


開啟網頁後，執行資料更新的程式，在網頁中的股票資訊即時更動。

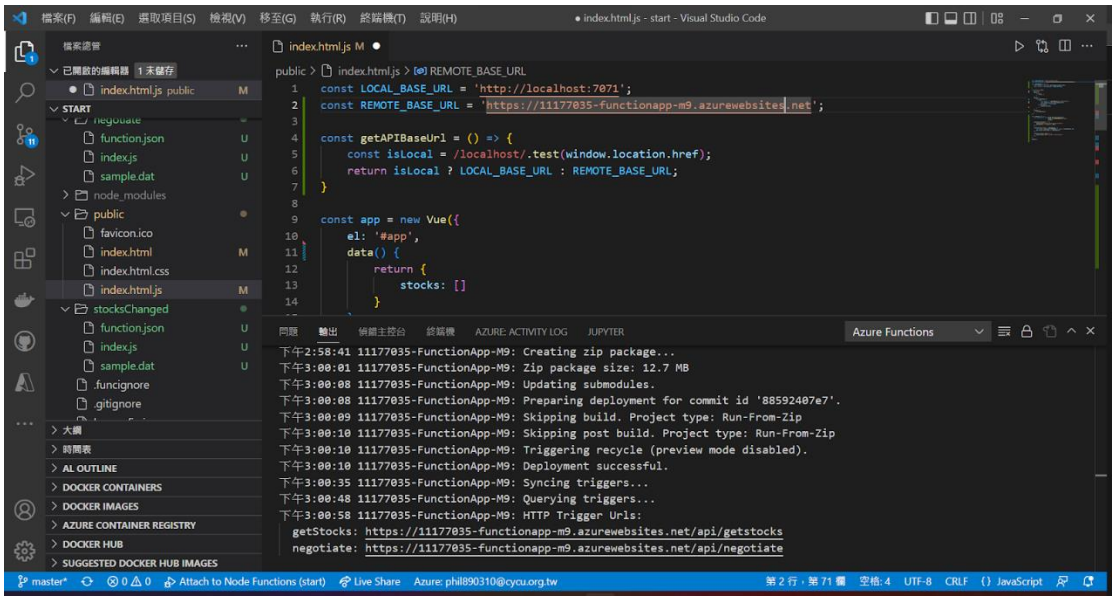
```
PS C:\Users\phil8\Desktop\Serverless2\serverless-demo\start> npm run update-data
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

> microsoft-learn-func-signalr-start@1.0.0 update-data
> node update.js

Read data from database.


Data updated: {"id":"e0eb6e85-176d-4ce6-89ae-1f699aaa0bab","symbol":"ABC","price":101.19,"change":6.12,"ch
angeDirection":"+","_rid":"Jyl+AKCqSWIBAAAAAAAAAA==","_self":"dbs/Jyl+AA==/colls/Jyl+AKCqSWI=/docs/Jyl+AKC
qSWIBAAAAAAAAAA==/","_etag":"\"410078dc-0000-0b00-0000-62f092fc0000\"","_attachments":"attachments/","_ts"
:1659933436}
```



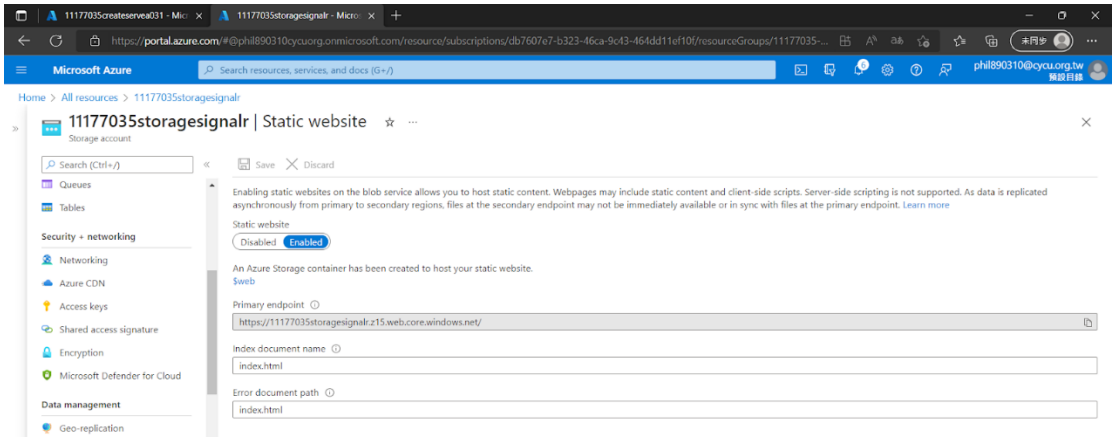
第三部分，部署 Function App 至雲端。



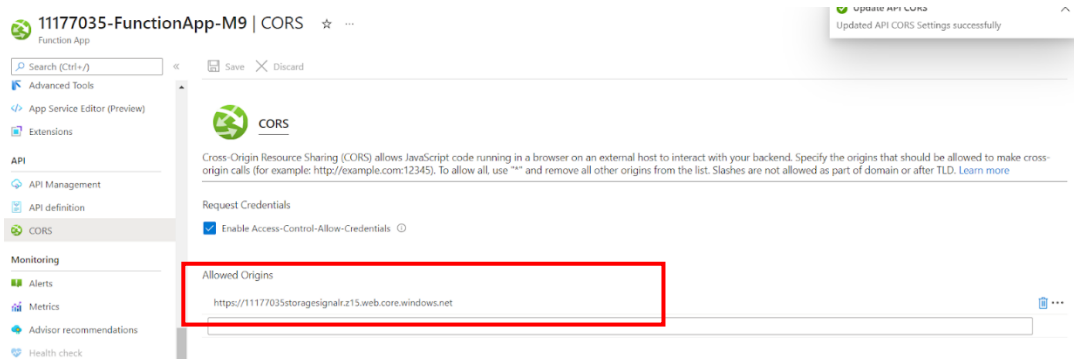
修改 base url，並且上傳設定至雲端。



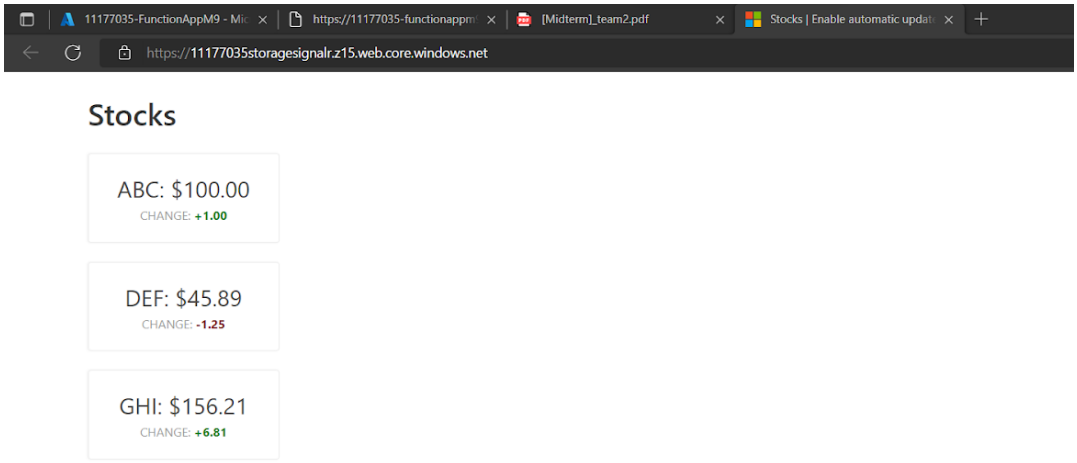
設定 Static Website 作為 Endpoint。



設定 CORS，使 Function App 內容能被 Static Website 存取。

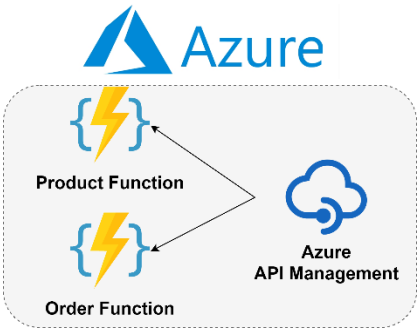


最終在 Static Website 運行 App。



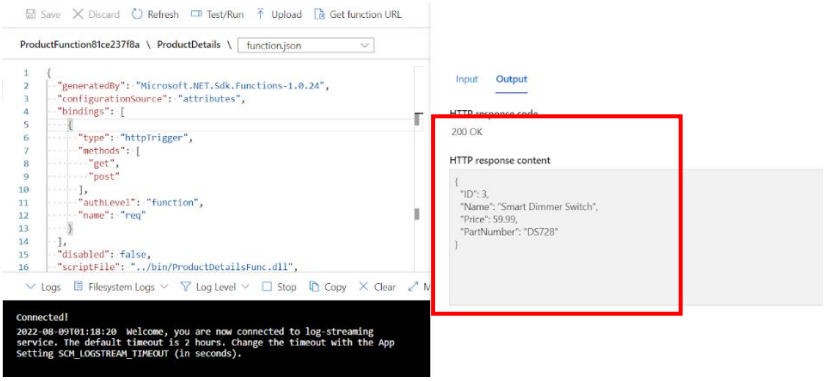
Module 11: Expose multiple Azure Function apps as a consistent API by using Azure API Management

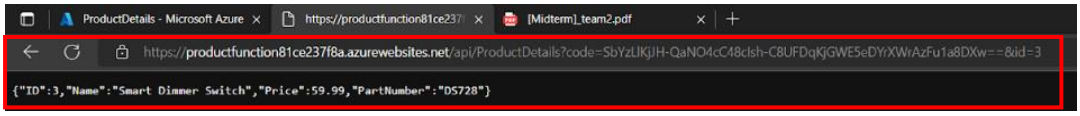
此模組主要使用 Azure API Management 管理多個 Azure Function，將多個 Microservice 集合至一個 API 當中。



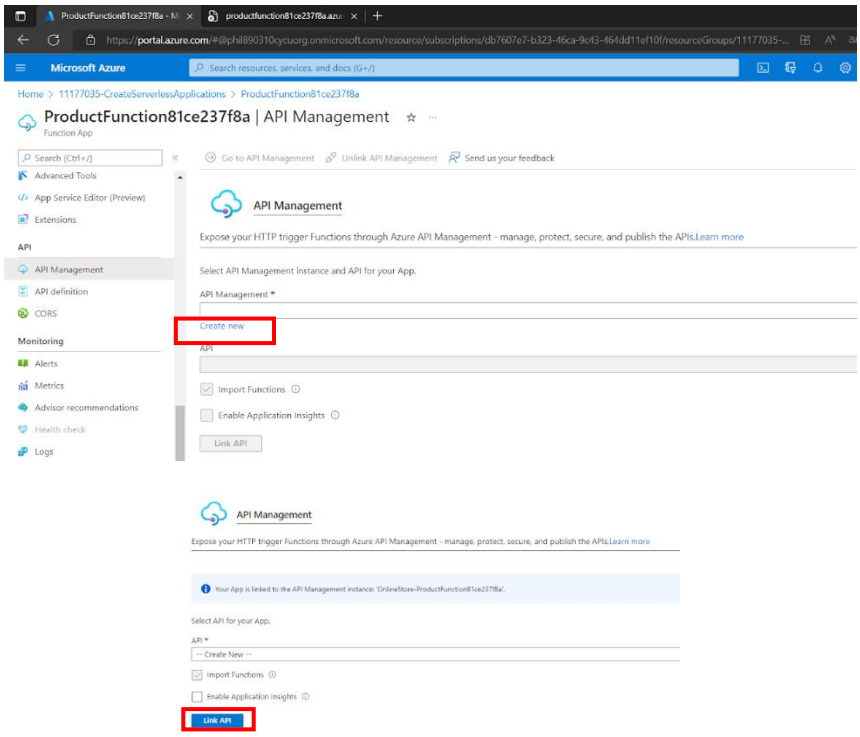
此模組使用 Github 資源包 https://github.com/MicrosoftDocs/mslearn-apim-and-functions，Github 中包含一個 Shell Script，可以用於生成兩個已經撰寫好的 Azure Function（Product、Order），供訂單、查詢的倉儲應用作展示。

測試 Product Function，使用 GET: id=3 進行測試，成功 Request 資料。

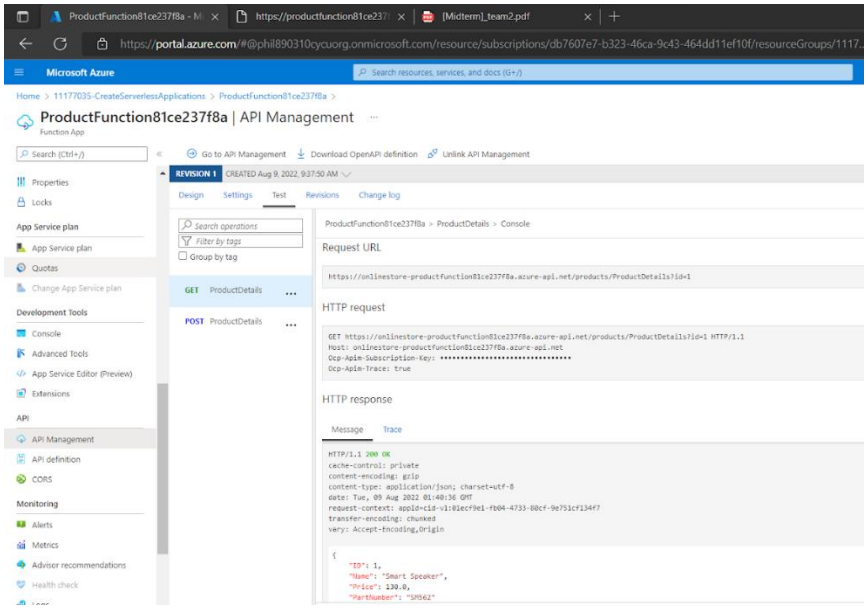




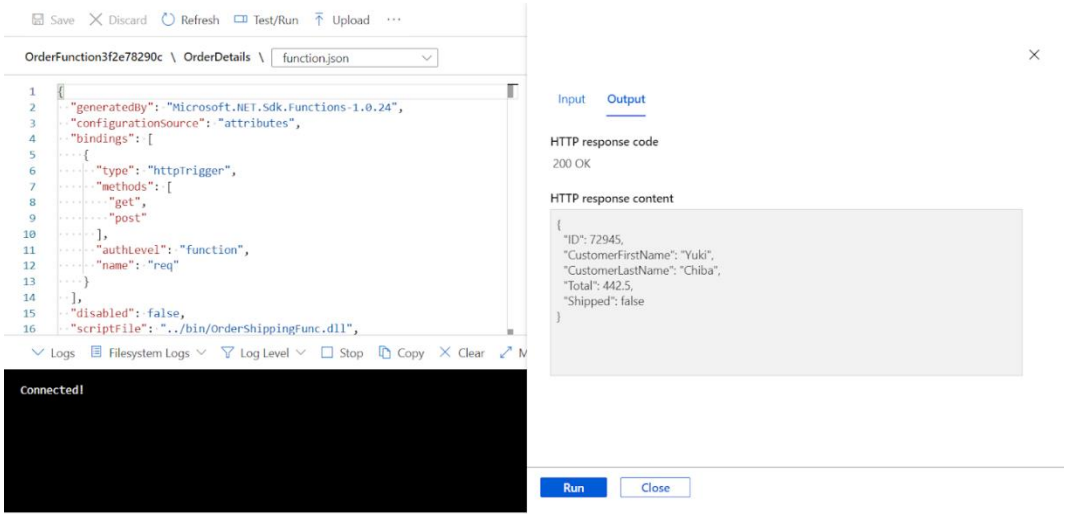
使用 API Management 將 API 公開，過程連結 Product Function。

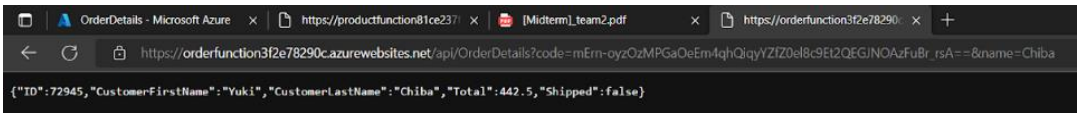


在 API Management 中，使用 GET: id=1 進行測試，成功 Request 資料。

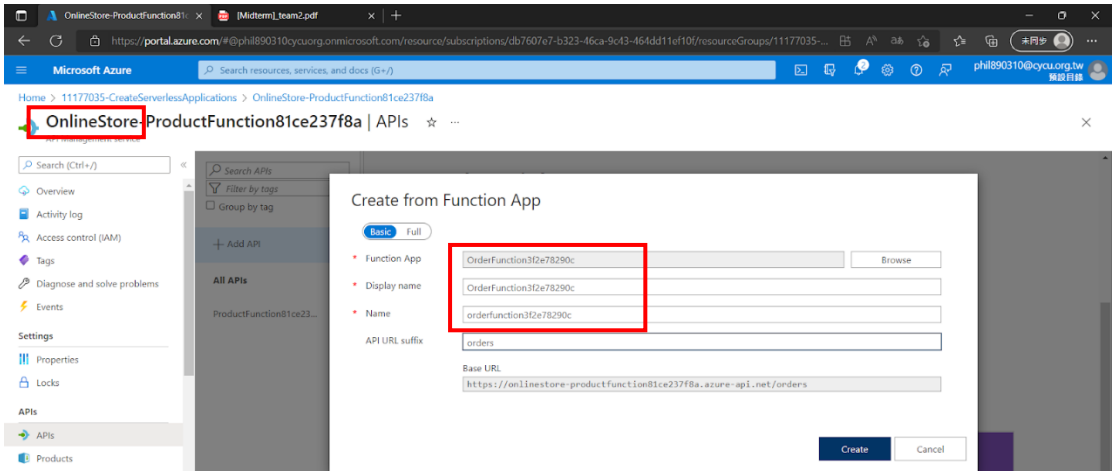


測試另一個 OrderFunction，GET name: Chiba。

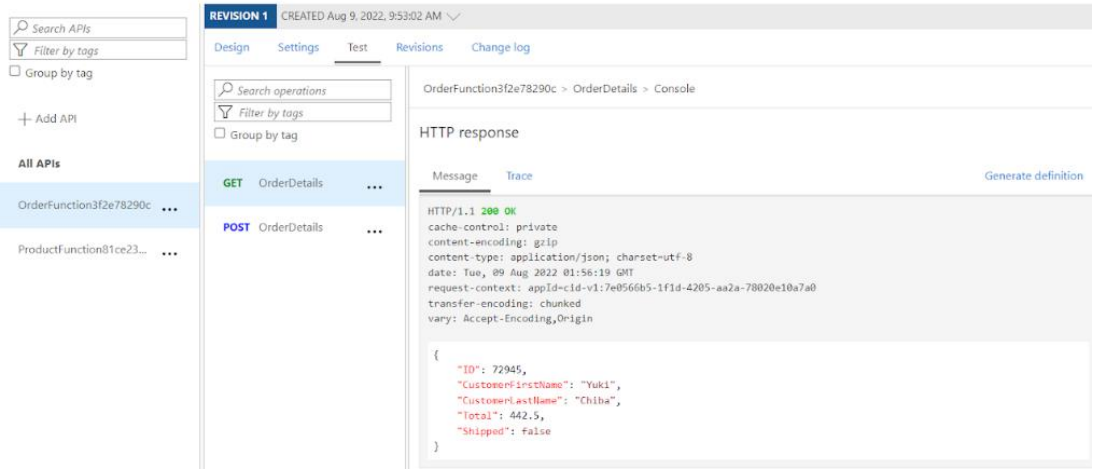




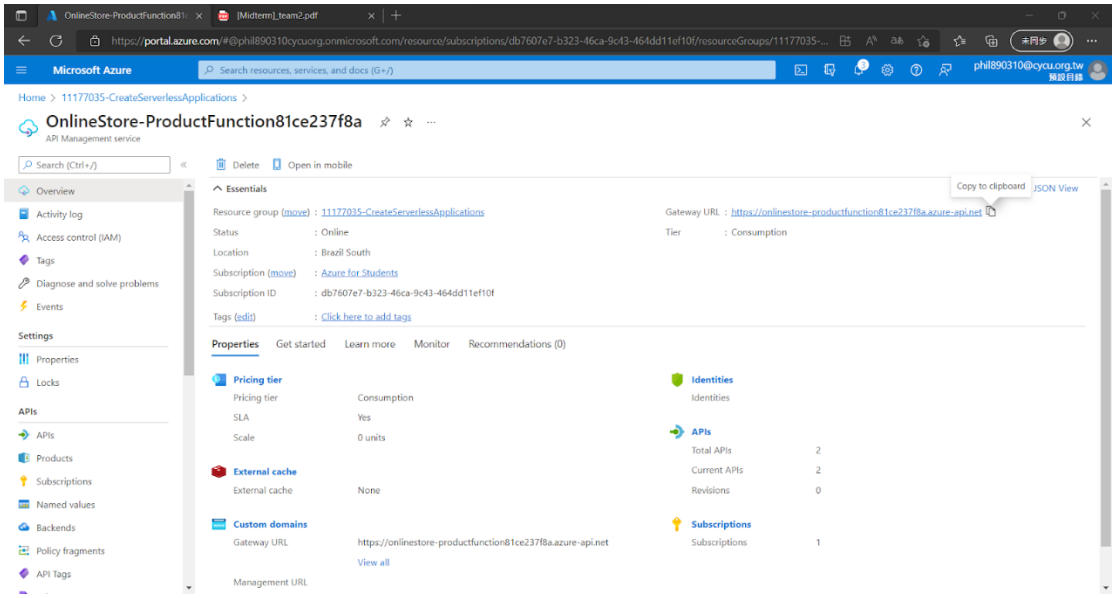
添加 Function 至既有的 OnlineStore API。



在 API Management 中，使用 GET: name=Chiba 進行測試，成功 Request 資料。



使用 API Management 的 Gateway URL 以及 Key 測試 Microservice 服務。



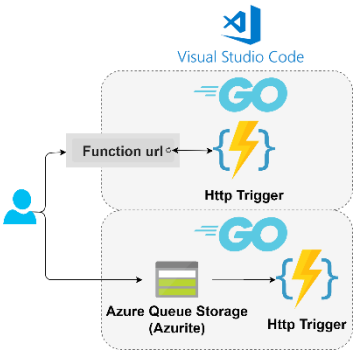
輸入<GATEWAY_URL>變數以及<SUB_KEY>變數，即可在相同的 Domain 中，透過不同的 suffix 使用不同的 Azure Function，得到不同的結果。

```
phil890310@Azure:~$ GATEWAY_URL=https://onlinestore-productfunction81ce237f8a.azure-api.net
phil890310@Azure:~$ SUB_KEY=c40800420a8b477fbf87c77d0b21fb73
phil890310@Azure:~$ curl -X GET "$GATEWAY_URL/orders/OrderDetails?name=Henri" -H "Ocp-Apim-Subscription-Key:$SUB_KEY"
{"ID":56224,"CustomerFirstName":"Pascale","CustomerLastName":"Henri","Total":307.98,"Shipped":true}phil890310@Azure:~$
```

```
phil890310@Azure:~$ curl -X GET "$GATEWAY_URL/products/ProductDetails?id=2" -H "Ocp-Apim-Subscription-Key:$SUB_KEY"
{"ID":2,"Name":"Home Security Camera","Price":105.99,"PartNumber":"SC967"}phil890310@Azure:~$
```

Module 12: Build serverless apps with Go

此模組重點強調開發人員可以使用任何熟悉的語言進行開發，只要語言支持 Http 工具，此模組使用 GoLang 為例子進行展示，並示範了另一個 Storage Account 的使用（Azurite，Local Environment 版本的 Storage Account）



第一部分，首先簡單運行一個 GoLang 的 Hello World 測試環境。

helloworld.go > main

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World")
}
```

新增一個 Project，在語言的部分特別選定 Custom Handler，再新增一個 Http Trigger，更換範例程式碼，加入 Http 服務。

helloworld.go U　server.go U

server.go > helloHandler

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")
    if r.Method == "GET" {
        w.Write([]byte("hello world"))
    } else {
        body, _ := ioutil.ReadAll(r.Body)
        w.Write(body)
    }
}

func main() {
    customHandlerPort, exists := os.LookupEnv("FUNCTIONS_CUSTOMHANDLER_PORT")
    if !exists {
        customHandlerPort = "8080"
    }
    mux := http.NewServeMux()
    // mux.HandleFunc("/api/hello", helloHandler)
    fmt.Println("Go server Listening on: ", customHandlerPort)
    log.Fatal(http.ListenAndServe(":"+customHandlerPort, mux))
}
```

在 host.json 設定檔，額外設定 Go 編譯執行檔位置。

host.json - Azure-Go-Lang - Visual Studio Code

server.go U　local.settings.json　host.json U

host.json > ...

```json
{
  "version": "2.0",
  "logging": {
    "applicationInsights": {
      "samplingSettings": {
        "isEnabled": true,
        "excludedTypes": "Request"
      }
    }
  },
  "customHandler": {
    "description": {
      "defaultExecutablePath": "./server.exe",
      "workingDirectory": "",
      "arguments": []
    },
    "enableForwardingHttpRequest" : true
  }
}
```

```
問題　輸出　偵錯主控台　終端機　AZURE　JUPYTER

PS C:\Users\phil8\Desktop\Azure-Go-Lang>
PS C:\Users\phil8\Desktop\Azure-Go-Lang> func start

Azure Functions Core Tools
Core Tools Version:       4.0.3971 Commit hash: d0775d487c93ebd49e9c1166d5c3c01f3c76eaaf  (64-bit)
Function Runtime Version: 4.0.1.16815

Core Tools Version:       4.0.3971 Commit hash: d0775d487c93ebd49e9c1166d5c3c01f3c76eaaf  (64-bit)
Function Runtime Version: 4.0.1.16815

[2022-08-09T03:19:19.989Z] Go server Listening on:  55357

Functions:

        hello: [GET,POST] http://localhost:7071/api/hello

For detailed output, run func with --verbose flag.
```

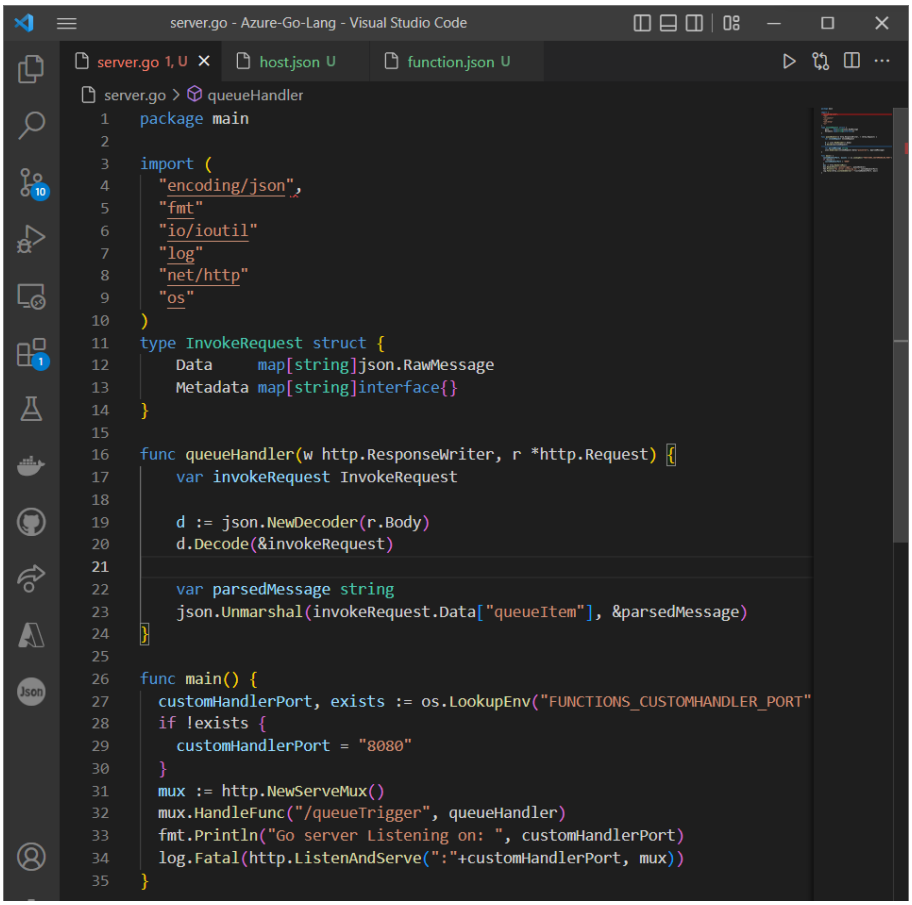
使用 func start，執行 Azure Function。

「在 Azure 中使用關聯式資料」　練習 - 使用自訂處理常式建置應　127.0.0.1:7071/api/hello

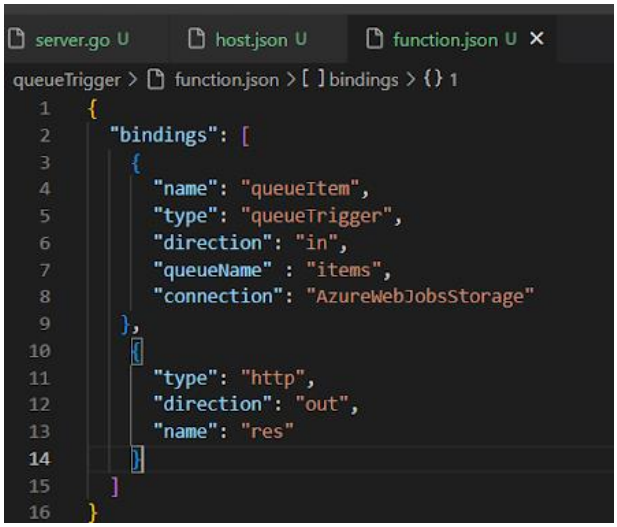
127.0.0.1:7071/api/hello　未同步

```
hello world
```

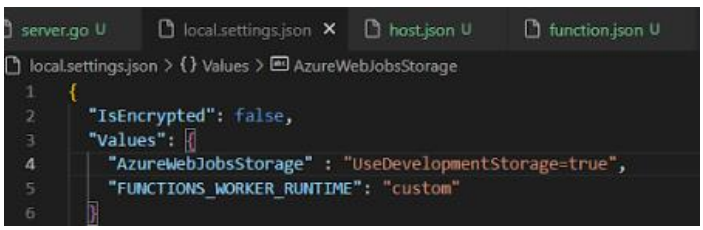
第二部分，再新增一個 Project，在語言的部分特別選定 Custom Handler，新增一個 Http Trigger，更換範例程式碼，加入 Http 服務，讀取 QueueStorage 資訊。



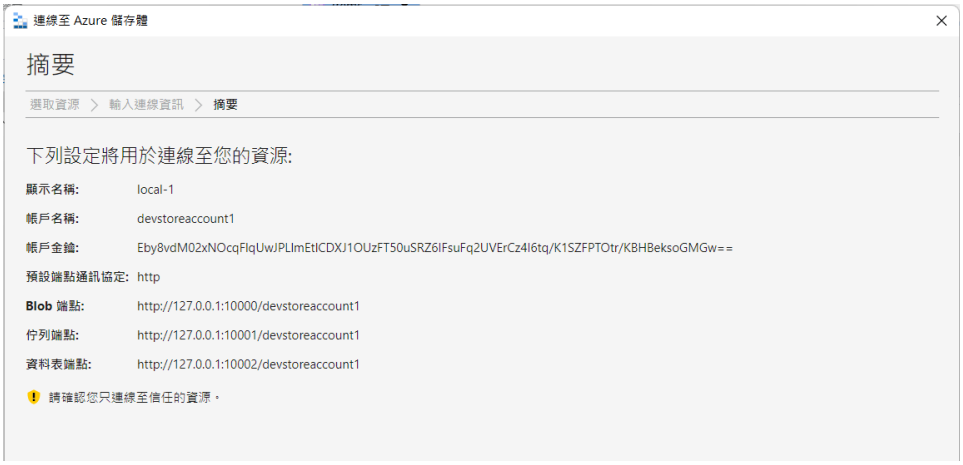
在 function.json 中，更換 Trigger 從 HttpTrigger 至 QueueTrigger。



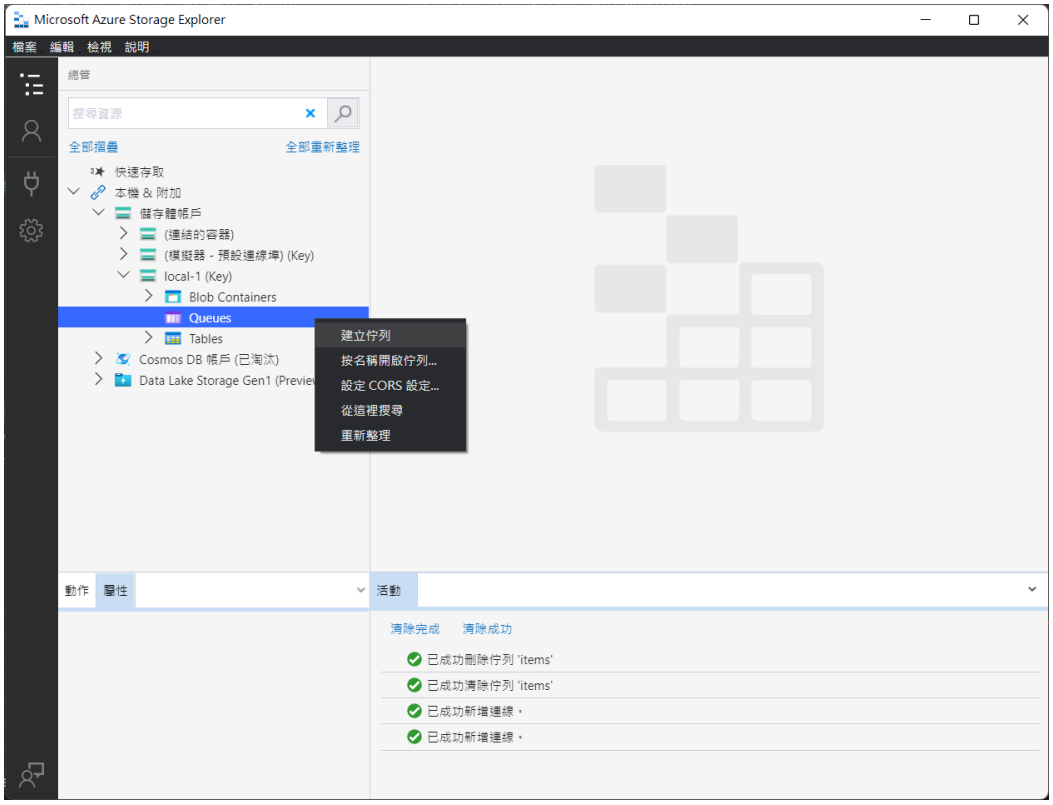
在 local.settings.json 中，而外設定 AzureWebJobStorage 環境變數。



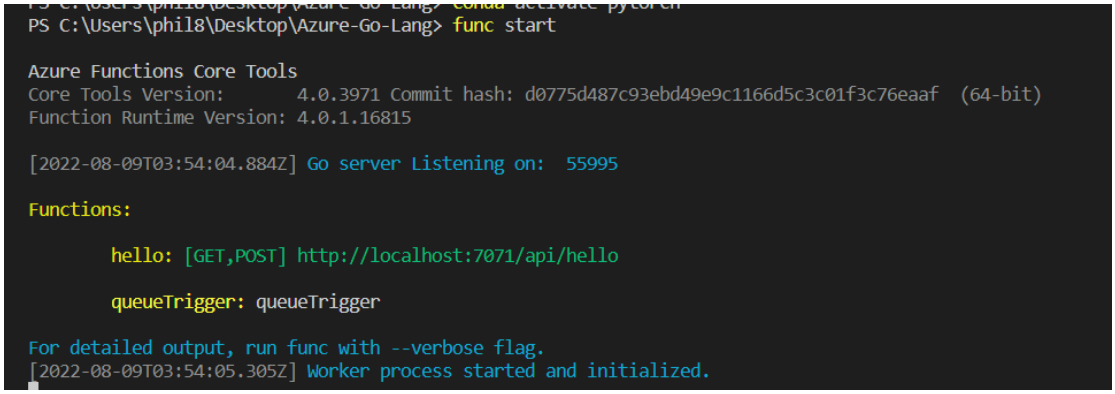
連線至 Azurite，設定相關 port 端點與 Key。



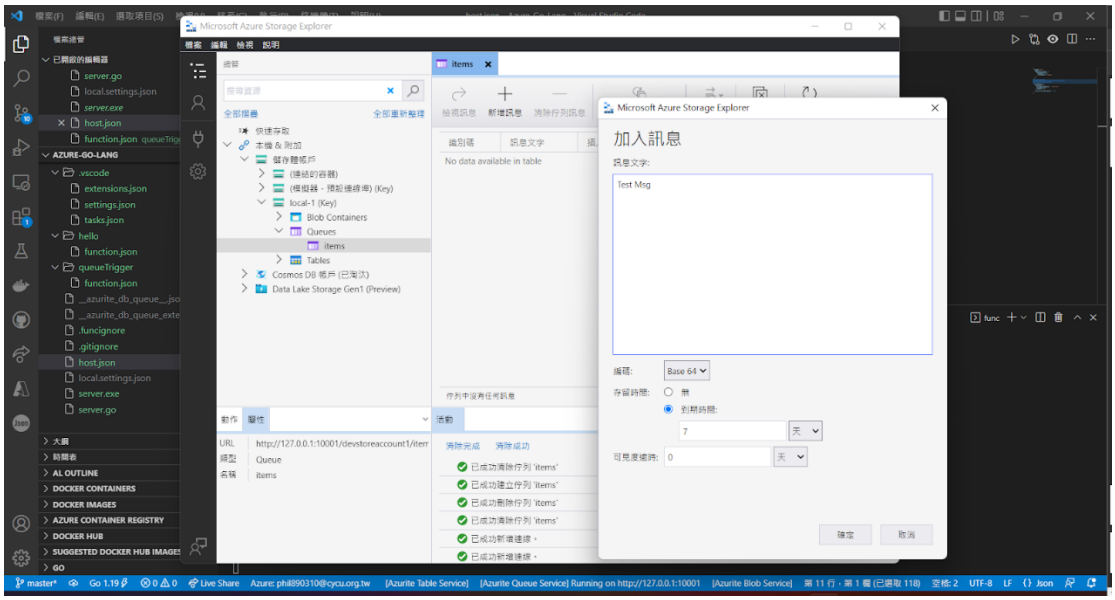
在 App 介面中，新增名為 Item 的 Queue 佇列。



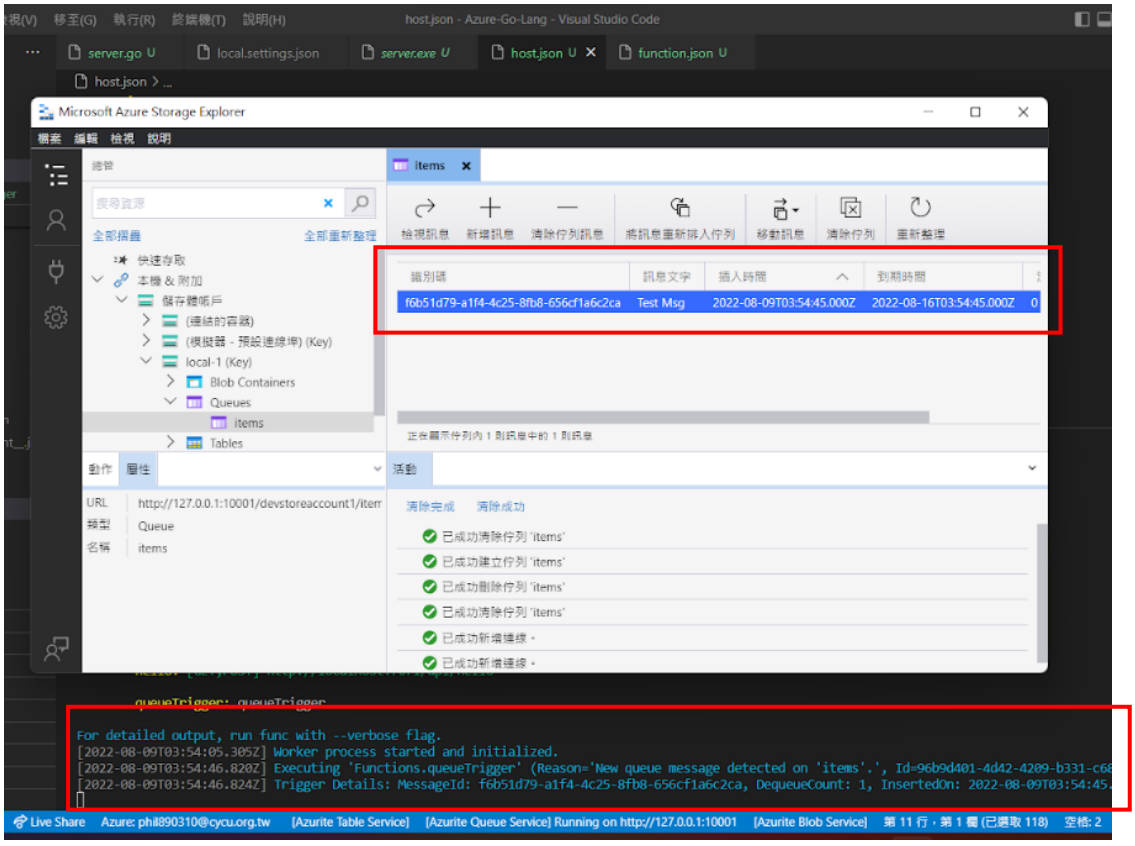
回到 VS Code，運行 Azure Function。



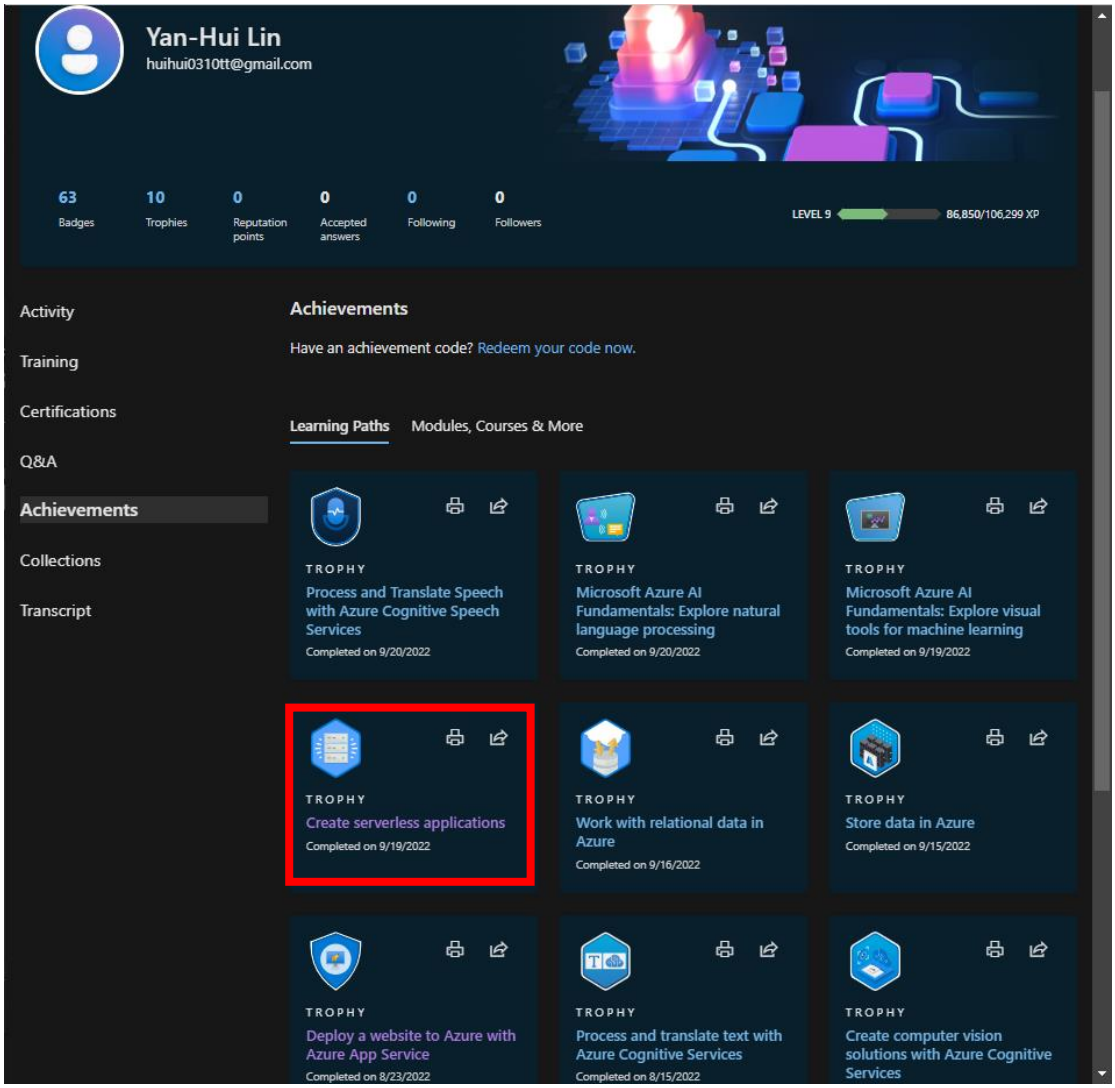
在 Azurite 加入 Message，進一步觸發 Azure Function。



在 VS Code 中可以看到 Trigger Log，在 Azurite 介面也能看到文字訊息。



Take screenshots of Badges and Trophies



Learned from the Learning Path

期中邀請的講員曾經推廣過 Serverless，使用 Serverless 進行程式開發除了可以搭配 Microservices 使得各個物件的開發快速且穩定，物件彼此的依賴程度更低。另外，計價便宜也是一大優點，是相當適合給程式設計師用於解決方法的策略。在 Azure，提供了相當多樣式的觸發方式，並且提供本地開發的選項，完整學習整個 Learning Path 的內容，學到的不只是 Serverless 的概念，也附帶夾雜更多關於專案架構的構思策略。

3. Problems

1. 實際帶 Lab 中，會發現 CosmosDB 開不起來，進一步查看發現在教程中會要求使用免費的資源方案，但因為實際上會有訂閱過多導致

EastUS 免費資源不足的問題，修改成少許計費制並減少備援後已修復。

2. 在 Azure Core Tools 中的 Extension 有做過更新，對於運行 Azure Function 的環境有許多的衝突。

FeedBack

此 Learning Path 雖然豐富充實，但對於初學者而言可能有點過多，應該可以學習 AZ-900、AZ-104 或是 AI-900、AI-102 的 Learning Path 可以把章節切分成兩個部分。

Reference

[1] https://www.c-sharpcorner.com/UploadFile/abhijmk/what-why-and-how-about-signalr/