



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

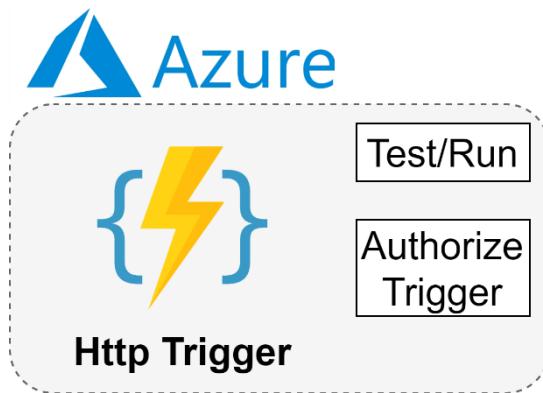
Create serverless applications

<https://docs.microsoft.com/en-us/learn/patterns/create-serverless-applications/>

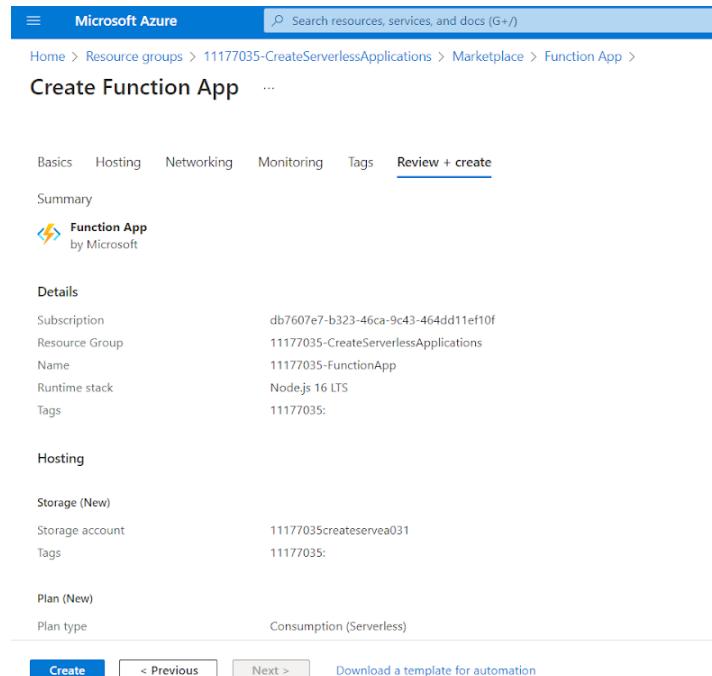
## 2. Summary Homework Assignment

### 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。



Microsoft Azure

Search resources, services, and docs (G+ /)

Home > Resource groups > 11177035-CreateServerlessApplications > Marketplace > Function App >

Create Function App ...

Basics Hosting Networking Monitoring Tags Review + create

Summary

Function App by Microsoft

Details

Subscription	db7607e7-b323-46ca-9c43-464dd11ef10f
Resource Group	11177035-CreateServerlessApplications
Name	11177035-FunctionApp
Runtime stack	Node.js 16 LTS
Tags	11177035:

Hosting

Storage (New)

Storage account	11177035createservea031
Tags	11177035:

Plan (New)

Plan type	Consumption (Serverless)
-----------	--------------------------

Create < Previous Next > Download a template for automation

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

### Create function

#### Select a template

Use a template to create a function. Triggers describe the type of events that invoke your functions. [Learn more](#)

Filter

Durable Functions Entity HTTP starter 會在每次收到執行協調器函式的 HTTP 請求時觸發的函式。

Durable Functions HTTP starter 會在每次收到執行協調器函式的 HTTP 請求時觸發的函式。

Durable Functions activity 會在每次協調器函式呼叫活動時執行的函式。

Durable Functions entity 用以儲存狀態的 Durable Functions 實體。

Durable Functions orchestrator 會在序列中呼叫活動函式的協調器函式。

Kafka output 將訊息傳送至指定 Kafka 主題的函式

Kafka trigger 每次有訊息新增到指定 Kafka 主題時都會執行的函式

RabbitMQ trigger 每當訊息新增至指定的 RabbitMQ 行列時，將會執行的函式

SignalR negotiate HTTP trigger 由 HTTP 觸發的函式。SignalR 用戶端會予以呼叫，以開始連線交涉

#### Template details

We need more information to create the HTTP trigger function. [Learn more](#)

New Function \*

M2-HttpTrigger

授權等級 \*①

Function

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

The screenshot shows the Azure Functions Test/Run interface. The top navigation bar includes Save, Discard, Refresh, Test/Run, Upload, and other options. The main area shows the code editor for the file index.js:

```
1 module.exports = async function (context, req) {
2   context.log(`JavaScript HTTP trigger function processed a request.`)
3
4   const name = (req.query.name || (req.body && req.body.name));
5   const responseMessage = name
6   ? `Hello, ${name}! This HTTP triggered function executed successfully. Pass a name as a query parameter or in the request body for a personalized response.`
7   : `This HTTP triggered function executed successfully. Pass a name as a query parameter or in the request body for a personalized response.`
8
9   context.res = {
10     // status: 200, /* Defaults to 200 */
11     body: responseMessage
12   };
13 }
```

Below the code editor is the App Insights Logs panel, which displays the following log entries:

```
2022-08-07T08:05:34Z [Verbose] Sending invocation id:f4535e7c-9156-4b3b-acc7-edd632759ec4
2022-08-07T08:05:34Z [Verbose] Posting invocation id:f4535e7c-9156-4b3b-acc7-edd632759ec4 on workerId:3fde15b2-00e3-47db-ad01-b2ce43bafc7f
2022-08-07T08:05:34Z [Information] JavaScript HTTP trigger function processed a request.
2022-08-07T08:05:34Z [Information] Executed 'Functions.M2-HttpTrigger' (Succeeded, Id=f4535e7c-9156-4b3b-acc7-edd632759ec4, Duration=152ms)
```

The right side of the interface shows the Output tab for the HTTP response, displaying the message "Hello, Azure. This HTTP triggered function executed successfully."

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

The screenshot shows the Azure Functions Test/Run interface. The top navigation bar includes Save, Discard, Refresh, Test/Run, Upload, and other options. The main area shows the code editor for the file index.js:

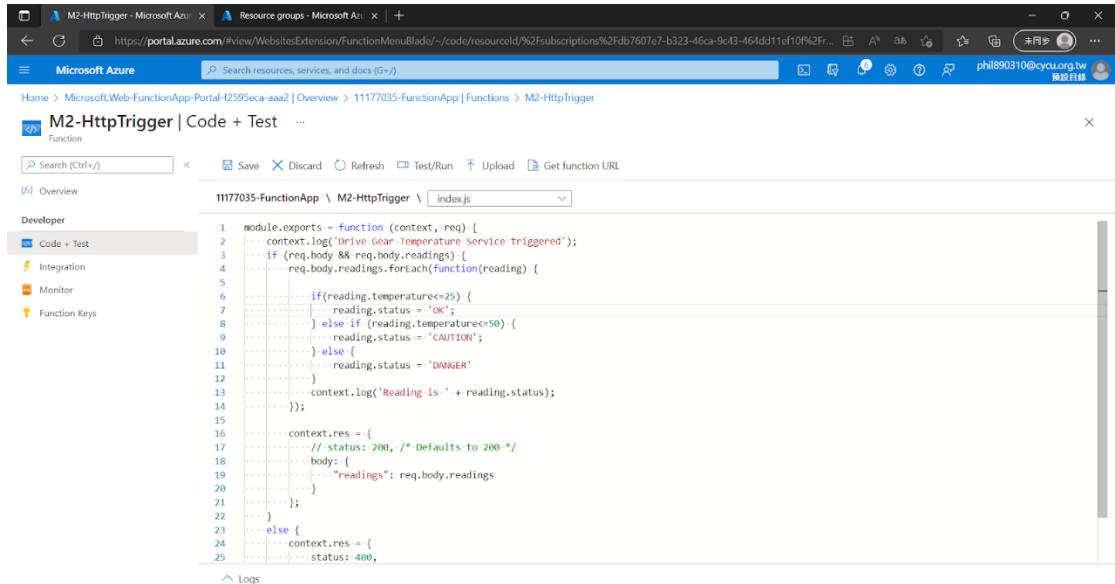
```
1 module.exports = async function (context, req) {
2   context.log(`JavaScript HTTP trigger function processed a request.`)
3
4   const name = (req.query.name || (req.body && req.body.name));
5   const responseMessage = name
6   ? `Hello, ${name}! This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response.`
7   : `This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response.`
8
9   context.res = {
10     // status: 200, /* Defaults to 200 */
11     body: responseMessage
12   };
13 }
```

Below the code editor is the App Insights Logs panel, which displays the following log entries:

```
Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the logs for this Function, please go to 'Monitor' from the Function menu.
2022-08-07T08:18:05Z [Information] Executing 'Functions.M2-HttpTrigger' (Reason:'This function was programmatically called via the host APIs.', Id=640ffcf5-5ffc-4dea-be85-5d3094b5d152)
2022-08-07T08:18:05Z [Verbose] Sending invocation id:640ffcf5-5ffc-4dea-be85-5d3094b5d152
2022-08-07T08:18:05Z [Verbose] Posting invocation id:640ffcf5-5ffc-4dea-be85-5d3094b5d152 on workerId:3fde15b2-00e3-47db-ad01-b2ce43bafc7f
2022-08-07T08:18:05Z [Information] Javascript HTTP trigger function processed a request.
2022-08-07T08:18:05Z [Information] Executed 'Functions.M2-HttpTrigger' (Succeeded, Id=640ffcf5-5ffc-4dea-be85-5d3094b5d152, Duration=13ms)
```

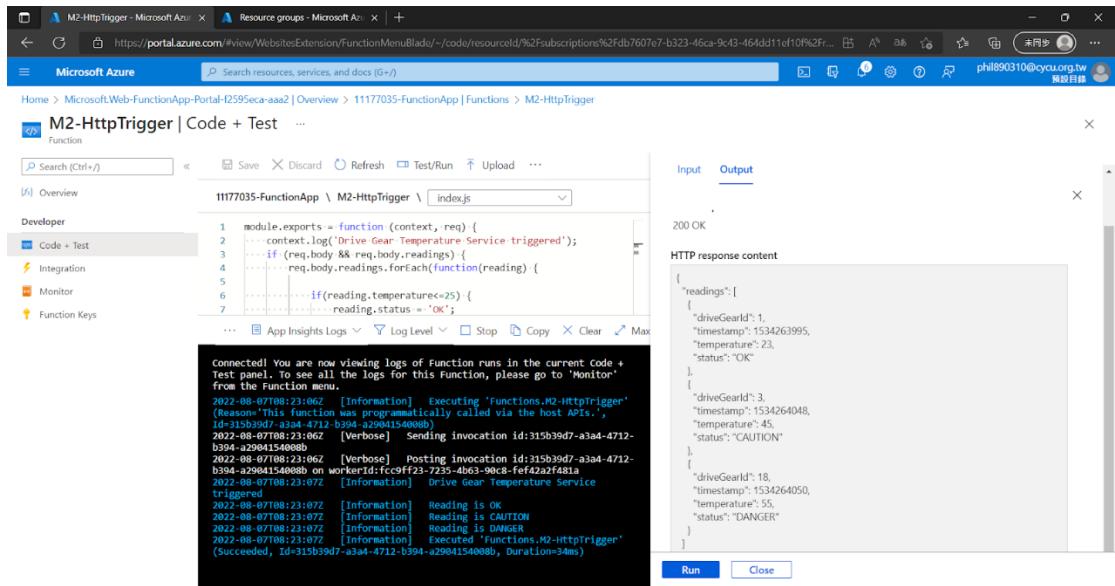
The right side of the interface shows the Output tab for the HTTP response, displaying the message "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."

## 修改 HttpTrigger 程式碼，使其成為 IoT 溫度檢測的應用。



```
module.exports = function (context, req) {
    context.log('Drive Gear Temperature Service triggered');
    if (req.body && req.body.readings) {
        req.body.readings.forEach(function(reading) {
            if(reading.temperature<=25) {
                reading.status = 'OK';
            } else if (reading.temperature<=50) {
                reading.status = 'CAUTION';
            } else {
                reading.status = 'DANGER';
            }
            context.log('Reading is ' + reading.status);
        });
        context.res = {
            // status: 200, /* Defaults to 200 */
            body: {
                "readings": req.body.readings
            }
        };
    } else {
        context.res = {
            status: 400,
        };
    }
};
```

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。



Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the logs for this Function, please go to 'Monitor' from the Function menu.

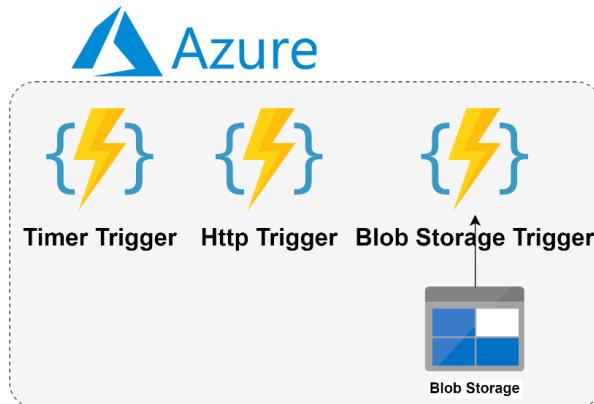
```
2022-08-07T08:23:06Z [Information] Executing 'Functions.M2-HttpTrigger' (Reason: "This function was programmatically called via the host APIs.", Id:315b30d7-a3a4-4712-b394-a2984154008b)
2022-08-07T08:23:06Z [Verbose] Sending invocation id:315b30d7-a3a4-4712-b394-a2984154008b on workerId:fcc9ff23-7235-4b63-90c8-fefa2af2f481a
2022-08-07T08:23:07Z [Information] Drive Gear Temperature Service triggered
2022-08-07T08:23:07Z [Information] Reading is OK
2022-08-07T08:23:07Z [Information] Reading is CAUTION
2022-08-07T08:23:07Z [Information] Reading is DANGER
2022-08-07T08:23:07Z [Information] Executed 'Functions.M2-HttpTrigger' (Succeeded, Id:315b30d7-a3a4-4712-b394-a2984154008b, Duration=3ms)
```

HTTP response content

```
[{"readings": [{"driveGearId": 1, "timestamp": 1534263995, "temperature": 23, "status": "OK"}, {"driveGearId": 3, "timestamp": 1534264048, "temperature": 45, "status": "CAUTION"}, {"driveGearId": 18, "timestamp": 1534264050, "temperature": 55, "status": "DANGER"}]]
```

## 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}

Use a template to create a function. Triggers describe the type of events that invoke your functions. [Learn more](#)

Durable Functions Entity HTTP starter	會在每次收到執行指揮器函式的 HTTP 要求時觸發的函式。
Durable Functions HTTP starter	會在每次收到執行指揮器函式的 HTTP 要求時觸發的函式。
Durable Functions activity	會在每次啟動狀態的 Durable Functions 實體。
Durable Functions entity	用以儲存狀態的 Durable Functions 實體。
Durable Functions orchestrator	會在序列中呼叫活動函式的協調器函式。
Kafka output	將訊息串送至指定 Kafka 主題的函式。
Kafka trigger	每次有訊息新增到指定 Kafka 主題時都會執行的函式。
RabbitMQ trigger	每當訊息新增至指定的 RabbitMQ 行列時，都會執行的函式。
SignalR negotiate HTTP trigger	由 HTTP 發送的函式。SignalR 用戶端會予以呼叫，以開始連線交涉。

**Template details**

We need more information to create the Timer trigger function. [Learn more](#)

New Function\*

排程\*

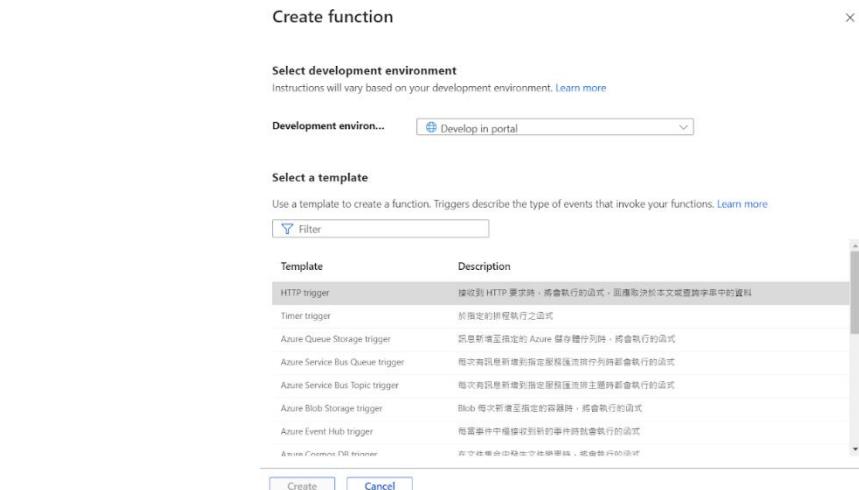
打開 Test/Run > Log 功能，查看 Function 每 20 秒觸發一次的結果。

...

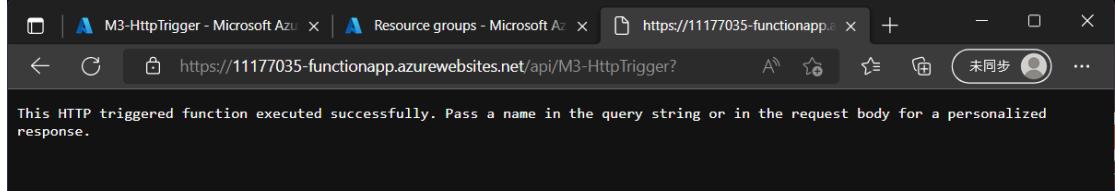
Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the logs for this Function, please go to 'Monitor' from the Function menu.

```
2022-08-07T08:37:13Z [Information] Executing 'Functions.M3-TimeTrigger'
(Reason='This function was programmatically called via the host APIs.',
Id=a64c4255-64e4-49a1-9792-db0da28429f6)
2022-08-07T08:37:13Z [Verbose] sending invocation id:a64c4255-64e4-49a1-
9792-db0da28429f6
2022-08-07T08:37:13Z [Verbose] Posting invocation id:a64c4255-64e4-49a1-
9792-db0da28429f6 on workerId:bb968ee1-9fa8-453d-b02d-227fa100b74e
2022-08-07T08:37:13Z [Information] JavaScript timer trigger function
ran! 2022-08-07T08:37:12.816Z
2022-08-07T08:37:13Z [Information] Executed 'Functions.M3-TimeTrigger'
(Succeeded, Id=a64c4255-64e4-49a1-9792-db0da28429f6, Duration=3ms)
```

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。

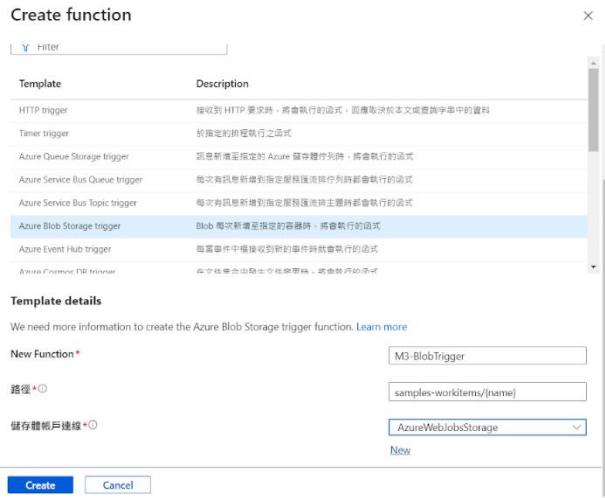


The screenshot shows the 'Create function' dialog in the Azure portal. The 'Select a template' step is active, displaying a list of available triggers. The 'HTTP trigger' template is selected, highlighted with a grey background. Other templates listed include Timer trigger, Azure Queue Storage trigger, Azure Service Bus Queue trigger, Azure Service Bus Topic trigger, Azure Blob Storage trigger, Azure Event Hub trigger, and Azure Cosmos DB trigger. Below the list are 'Create' and 'Cancel' buttons.

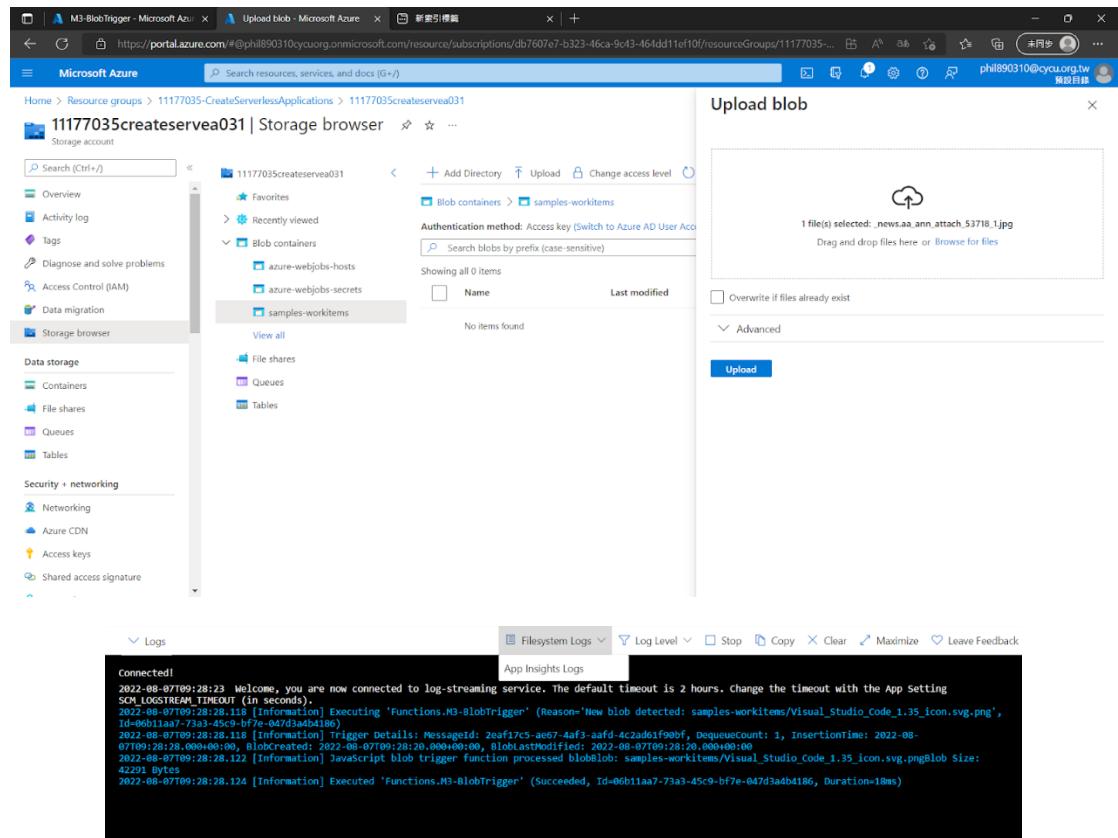
This screenshot shows a browser window displaying the result of an Azure Function execution. The URL is <https://11177035-functionapp.azurewebsites.net/api/M3-HttpTrigger?code=...>. The page content is: "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."

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。



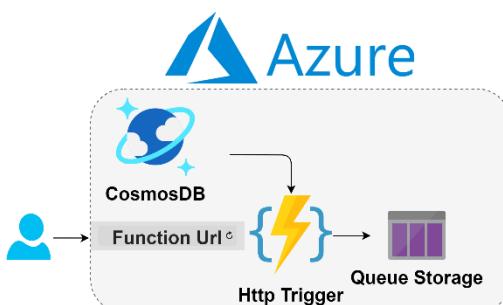
The screenshot shows the 'Create function' dialog in the Azure portal, specifically the 'Template details' step for a Blob Storage trigger. The 'Template' dropdown is set to 'Azure Blob Storage trigger'. In the 'Template details' section, there are fields for 'New Function\*' (containing 'M3-BlobTrigger'), 'Path\*' (containing 'samples-workitems/{name}'), and 'Storage account connection' (containing 'AzureWebJobsStorage' with a 'New' option). Below these fields are 'Create' and 'Cancel' buttons.

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。

The screenshot shows the Microsoft Azure Data Explorer interface. On the left, there's a sidebar with various options like Overview, Activity log, Tags, and Data Explorer. The main area is titled "func-io-learn-db" under the "DATA" section. A table is displayed with columns "id" and "url". There is one row with the id "docs" and the url "https://docs.microsoft.com/azure". This row is highlighted with a red box.

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。

The screenshot shows the Azure Functions portal. On the left, the developer tools bar is visible with "Code + Test", "Integration" (which is selected), "Monitor", and "Function Keys". In the center, there's a diagram showing a "Trigger (HTTP)" connected to a "Function" (M4-HttpTrigger) which has an "Outputs" connection. Below the trigger, there's an "Inputs" section with a "+ Add input" button. To the right, a modal window titled "Create Input" is open, showing the "Binding Type" set to "Azure Cosmos DB". Under "Azure Cosmos DB details", it shows "Cosmos DB 資戶連線" with a dropdown menu showing "No existing connections available" and a "New" button. It also shows fields for "文件參數名稱" (inputDocument), "資料庫名稱" (inDatabase), and "集合名稱" (MyCollection). The "OK" button is at the bottom right of the modal.

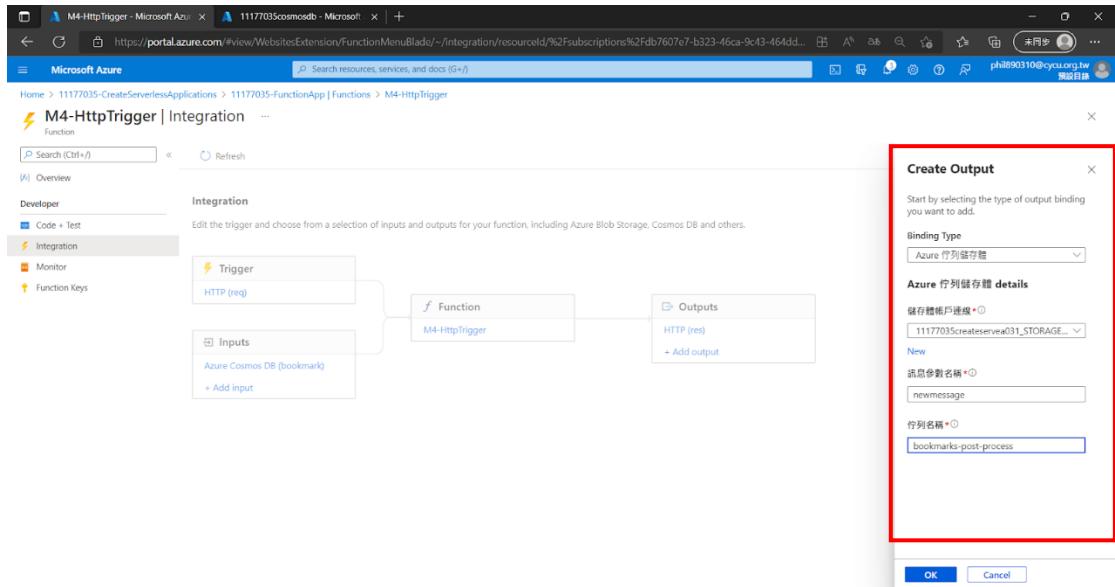
修改 Function 的邏輯內容，判斷 Get/Post 的查詢是否存在於 CosmosDB 的 Input Binding 資料中，若存在則顯示 CosmosDB 中其餘 url 資訊。

```
module.exports = function (context, req) {
  var bookmark = context.bindings.bookmark;
  if(bookmark){
    context.res = {
      body: { "url": bookmark.url },
      headers: {
        'Content-Type': 'application/json'
      }
    };
  } else {
    context.res = {
      status: 404,
      body : "No bookmarks found",
      headers: {
        'Content-Type': 'application/json'
      }
    };
  }
  context.done();
};
```

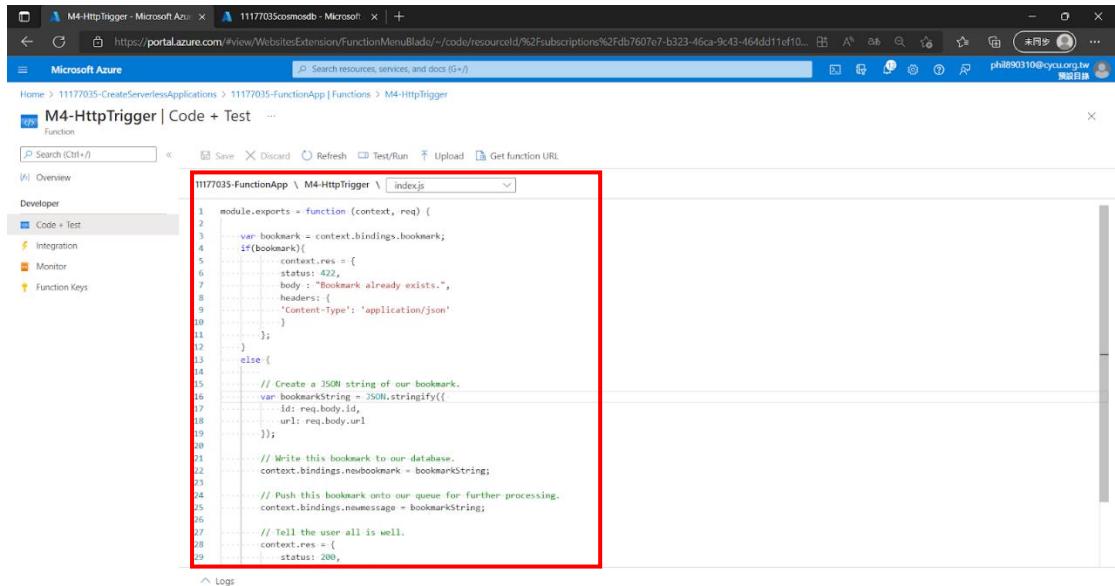
簡單做 Http Trigger 測試，使用 GET 查詢 id=docs 的資料，在 Function 內成功透過 Input Binding 查詢到 id=docs 這筆資料，並將結果回傳。

The screenshot shows a browser window with the URL "https://11177035-functionapp.azurewebsites.net/api/M4-HttpTrigger?code=eX3nkEwMwsazN5PrdHaDeYhidWY4eecc23DryNUtsRM\_AzFu8LPOA--&id=docs". The response is a JSON object with a single key "url" pointing to "https://docs.microsoft.com/azure". This result is highlighted with a red box.

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。

Input Output

Provide parameters to test the HTTP request. Results can be found in the Output tab.

HTTP method: POST

Key: master (Host key)

Query: + Add parameter

Headers: + Add header

Body:

```
1 {
2   "id": "docs",
3   "url": "https://docs.microsoft.com/azure"
4 }
```

Run Close

HTTP response code: 422 Client Error

HTTP response content: Bookmark already exists.

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

Input Output

Provide parameters to test the HTTP request. Results can be found in the Output tab.

HTTP method: POST

Key: master (Host key)

Query: + Add parameter

Headers: + Add header

Body:

```
1 {
2   "id": "github",
3   "url": "https://www.github.com"
4 }
```

Run Close

HTTP response code: 200 OK

HTTP response content: bookmark added!

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

Home > 11177035createserver031 | Queues >

bookmarks-post-process ...

Queue

Search (Ctrl+F) Refresh + Add message Dequeue message Clear queue

Authentication method: Access key (Switch to Azure AD User Account)

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

Access policy

Metadata

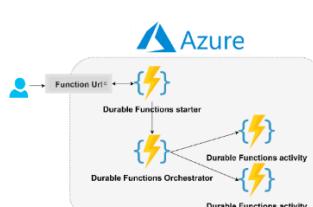
Authentications method: Access key (Switch to Azure AD User Account)

Search to filter items...

ID	Message text	Insertion time	Expiration time	Dequeue count
2cc02fdf-894a-4770-85...	{ "id": "github", "url": "https://www.github.com" }	8/7/2022, 8:23:35 PM	8/14/2022, 8:23:35 PM	0

## 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。

```
C:\home\site\wwwroot>npm install durable-functions
npm WARN deprecated uuid@0.1.3: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random
for details.
npm notice
npm notice New minor version of npm available! 8.1.0 -> 8.16.0
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v8.16.0>
npm notice Run 'npm install -g npm@8.16.0' to update
npm notice

C:\home\site\wwwroot>npm install typescript
npm notice
npm notice New minor version of npm available! 8.1.0 -> 8.16.0
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v8.16.0>
npm notice Run 'npm install -g npm@8.16.0' to update
npm notice

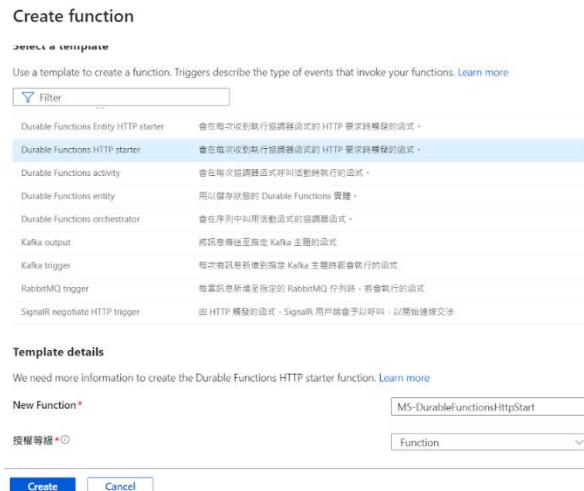
C:\home\site\wwwroot>npm install moment
up to date, audited 12 packages in 1s

1 package is looking for funding
  run "npm fund" for details

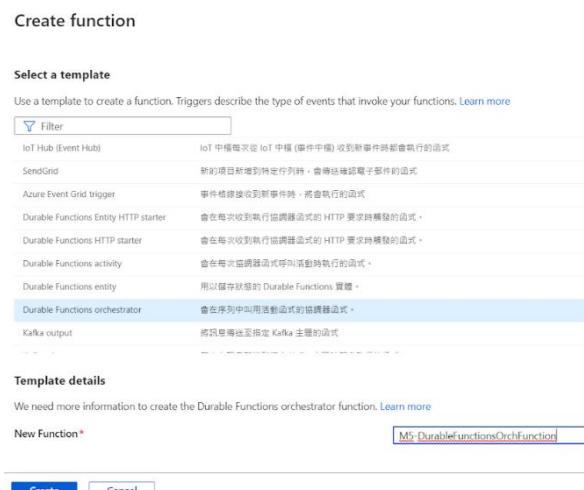
found 0 vulnerabilities

C:\home\site\wwwroot>
```

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。

The screenshot shows the Azure portal interface for creating functions. Two separate 'Create function' dialog boxes are displayed side-by-side.

- Left Dialog (M5-Approval):**
  - Title:** Create function
  - Template:** Select a template (Durable Functions activity)
  - Description:** 請在每次收到執行協調器函式的 HTTP 要求時觸發的函式。
  - Trigger:** Durable Functions activity
  - Code Editor:** Contains the code: `module.exports = async function (context) { ...return 'ESCALATION : You have not approved the project design proposal - reassigning to your Manager! - \${context.bindings.name}!'; }`
  - Buttons:** Create, Cancel
- Right Dialog (M5-Escalation):**
  - Title:** Create function
  - Template:** Select a template (Durable Functions activity)
  - Description:** 請在每次收到執行協調器函式的 HTTP 要求時觸發的函式。
  - Trigger:** Durable Functions activity
  - Code Editor:** Contains the code: `module.exports = df.orchestrator(function\* (context) { ... })`
  - Buttons:** Create, Cancel

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。

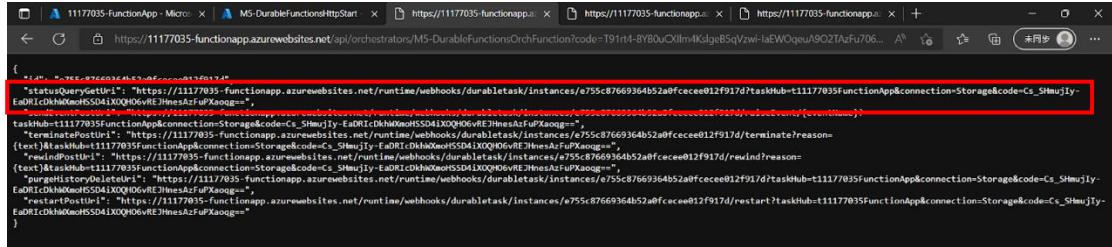
The screenshot shows the Azure portal interface for the 'M5-DurableFunctionsOrchFunction' orchestrator function. The code editor displays the following JavaScript code:

```

1 module.exports = df.orchestrator(function* (context) {
2     const outputs = [];
3     const deadline = moment.utc(context.df.currentUtcDateTime).add(20, "s");
4     const activityTask = context.df.waitForExternalEvent("Approval");
5     const timeoutTask = context.df.createTimer(deadline.toDate());
6
7     while (!activityTask.isCompleted() && !timeoutTask.isCompleted()) {
8         const outputs = yield context.df.task.any([activityTask, timeoutTask]);
9         if (outputs === activityTask) {
10             outputs.push(yield context.df.callActivity("M5-Approval", "Approved"));
11         } else {
12             outputs.push(yield context.df.callActivity("M5-Escalation", "Head of department"));
13         }
14     }
15     if (!timeoutTask.isCompleted()) {
16         // All pending timers must be complete or canceled before the function exits.
17         timeoutTask.cancel();
18     }
19     return outputs;
20 });

```

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

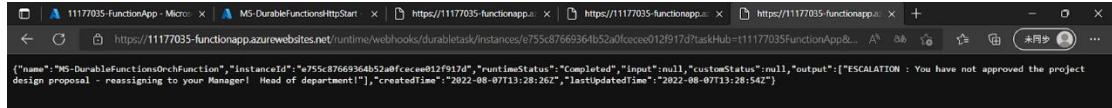


The screenshot shows a browser window with multiple tabs. The active tab is 'https://11177035-functionapp.azurewebsites.net/api/orchestrators/M5-DurableFunctionsOrchFunction?code=T91rt4-8YB0uCXlmlKSlge8SqVzwi-laEWoqeuA9O2TazFu706...'. The page content displays JSON data related to a durable task, specifically a 'terminatePostUri' call. A red box highlights the URL 'https://11177035-functionapp.azurewebsites.net/runtime/webhooks/durabletask/instances/e755c87669364b52a0fcceee012f917d?taskHub=t11177035FunctionApp&connection=Storage&code=Cs\_SHmuJly-Eu0IcKhkMwotDS04IxQ0DgREJhmesAzfpXaog=='. The JSON also includes other fields like 'terminatePostUri1' and 'taskHub1'.

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



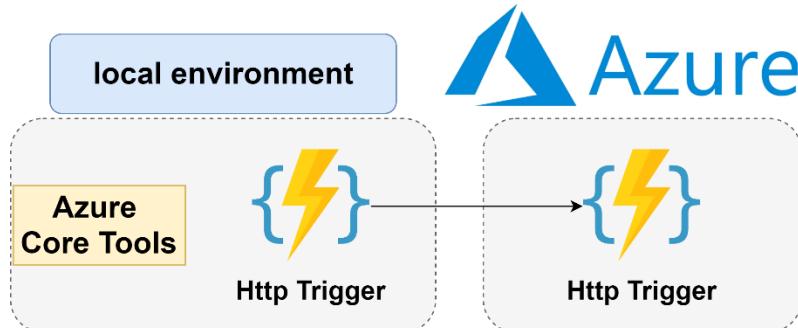
The screenshot shows a browser window with multiple tabs. The active tab is 'https://11177035-functionapp.azurewebsites.net/runtime/webhooks/durabletask/instances/e755c87669364b52a0fcceee012f917d?taskHub=t11177035FunctionApp&connection=Storage&code=Cs\_SHmuJly-Eu0IcKhkMwotDS04IxQ0DgREJhmesAzfpXaog=='. The page content displays a JSON object with a single entry: {"name": "M5-DurableFunctionsOrchFunction", "instanceId": "e755c87669364b52a0fcceee012f917d", "runtimeStatus": "Running", "input": null, "customStatus": null, "output": null, "createdTime": "2022-08-07T13:28:26Z", "lastUpdatedTime": "2022-08-07T13:28:26Z"}.



The screenshot shows a browser window with multiple tabs. The active tab is 'https://11177035-functionapp.azurewebsites.net/runtime/webhooks/durabletask/instances/e755c87669364b52a0fcceee012f917d?taskHub=t11177035FunctionApp&connection=Storage&code=Cs\_SHmuJly-Eu0IcKhkMwotDS04IxQ0DgREJhmesAzfpXaog=='. The page content displays a JSON object with a single entry: {"name": "M5-DurableFunctionsOrchFunction", "instanceId": "e755c87669364b52a0fcceee012f917d", "runtimeStatus": "Completed", "input": null, "customStatus": null, "output": [{"message": "ESCALATION : You have not approved the project design proposal - reassigning to your Manager! Head of department!"}], "createdTime": "2022-08-07T13:28:26Z", "lastUpdatedTime": "2022-08-07T13:28:54Z"}.

## 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。



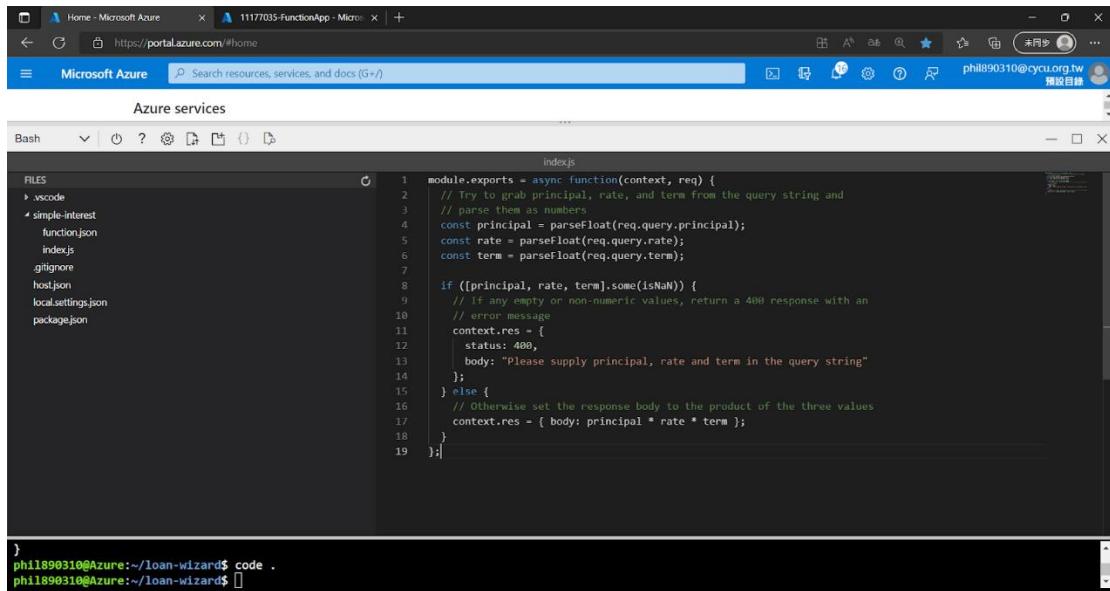
初始化 Azure Function，此指令會生成 host.json、local.settings.json、.gitignore 等環境設定檔案，並且設定開發語言。

```
$ 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。



```
module.exports = async function(context, req) {
    // Try to grab principal, rate, and term from the query string and
    // parse them as numbers
    const principal = parseFloat(req.query.principal);
    const rate = parseFloat(req.query.rate);
    const term = parseFloat(req.query.term);

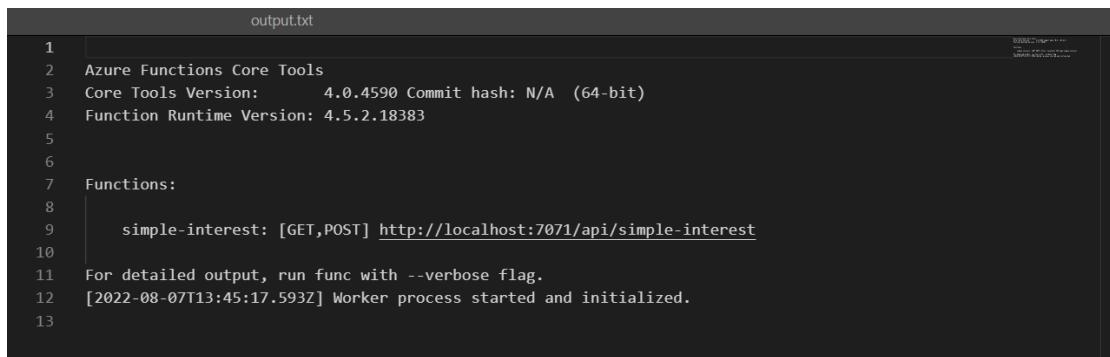
    if ([principal, rate, term].some(isNaN)) {
        // If any empty or non-numeric values, return a 400 response with an
        // error message
        context.res = {
            status: 400,
            body: 'Please supply principal, rate and term in the query string'
        };
    } else {
        // Otherwise set the response body to the product of the three values
        context.res = { body: principal * rate * term };
    }
};

}

phil1890310@Azure:~/loan-wizard$ code .
phil1890310@Azure:~/loan-wizard$ 
```

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。

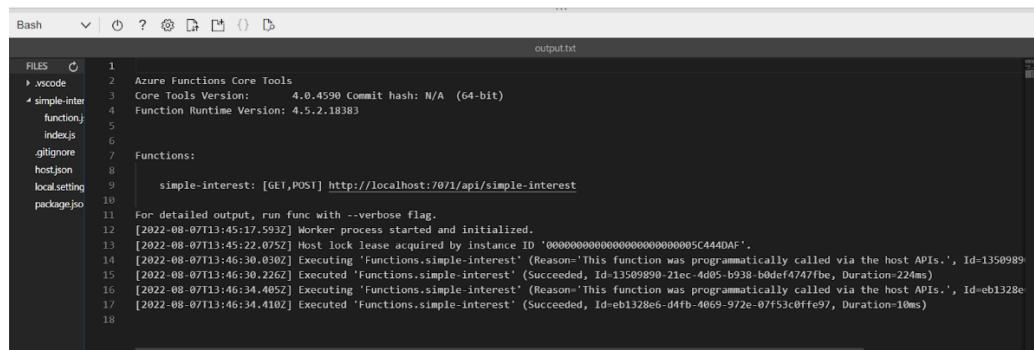
```
phil1890310@Azure:~/loan-wizard$ func start &> ~/output.txt &
[1] 810
```



```
1 Azure Functions Core Tools
2 Core Tools Version: 4.0.4590 Commit hash: N/A (64-bit)
3 Function Runtime Version: 4.5.2.18383
4
5
6 Functions:
7
8     simple-interest: [GET,POST] http://localhost:7071/api/simple-interest
9
10
11 For detailed output, run func with --verbose flag.
12 [2022-08-07T13:45:17.593Z] Worker process started and initialized.
13
14
```

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

```
phil1890310@Azure:~/loan-wizard$ curl "http://localhost:7071/api/simple-interest" -w "\n"
Please supply principal, rate and term in the query string
phil1890310@Azure:~/loan-wizard$ curl "http://localhost:7071/api/simple-interest?principal=5000&rate=.035&term=36" -w "\n"
6300.000000000001
```



```
1 Azure Functions Core Tools
2 Core Tools Version: 4.0.4590 Commit hash: N/A (64-bit)
3 Function Runtime Version: 4.5.2.18383
4
5
6 Functions:
7
8     simple-interest: [GET,POST] http://localhost:7071/api/simple-interest
9
10
11 For detailed output, run func with --verbose flag.
12 [2022-08-07T13:45:17.593Z] Worker process started and initialized.
13 [2022-08-07T13:45:22.075Z] Host lock lease acquired by instance ID '000000000000000000000000000000005C444DAF'.
14 [2022-08-07T13:46:30.030Z] Executing 'Functions.simple-interest' (Reason='This function was programmatically called via the host APIs.', Id=1350989)
15 [2022-08-07T13:46:30.226Z] Executed 'Functions.simple-interest' (Succeeded, Id=1350989-21ec-4d05-b938-b0def4747fbe, Duration=224ms)
16 [2022-08-07T13:46:34.405Z] Executing 'Functions.simple-interest' (Reason='This function was programmatically called via the host APIs.', Id=eb1328e
17 [2022-08-07T13:46:34.410Z] Executed 'Functions.simple-interest' (Succeeded, Id=eb1328e-d4fb-4069-872e-07f53c0ffe97, Duration=10ms)
```

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。

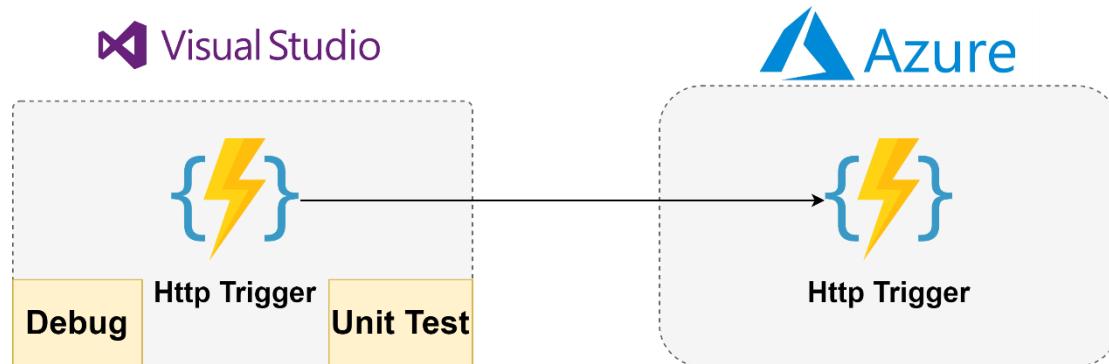
The screenshot shows the Azure portal interface for the 'simple-interest' function. On the left, there's a sidebar with options like Overview, Code + Test, Integration, Monitor, and Function Keys. The main area shows the 'Get Function Url' section with a button to copy the URL to clipboard. Below it are two line charts: one for 'Simple-Interest Count (Sum)' and another for 'simple-interest Successes (Sum)'. Both charts show data points for the time period from 9:15 PM to 9:45 PM UTC+0800, with values ranging from 0 to 100.

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

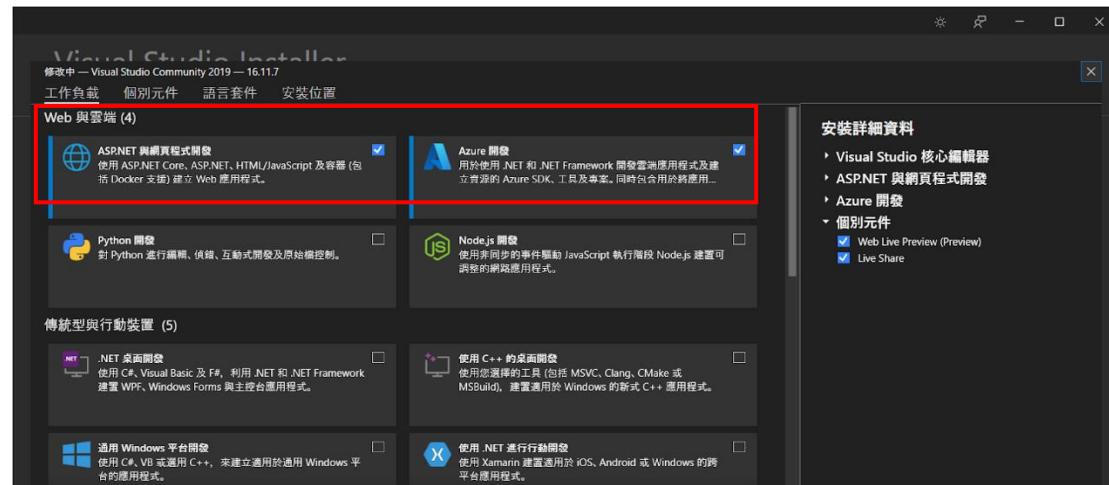
The top screenshot shows a browser window with the URL `https://11177035learnfunctions.azurewebsites.net/api/simple-interest?code=Og5kzVfrLVzzSj0Y7dZiGyoqlx0GYAJtFrh0GmXvFAzFufvay_Q==`. The page displays an error message: "Please supply principal, rate and term in the query string". The bottom screenshot shows the same URL but with additional parameters: `?code=Og5kzVfrLVzzSj0Y7dZiGyoqlx0GYAJtFrh0GmXvFAzFufvay_Q==&principal=5000&rate=0.05&term=10`. The response is the calculated interest value: "6300.000000000001".

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

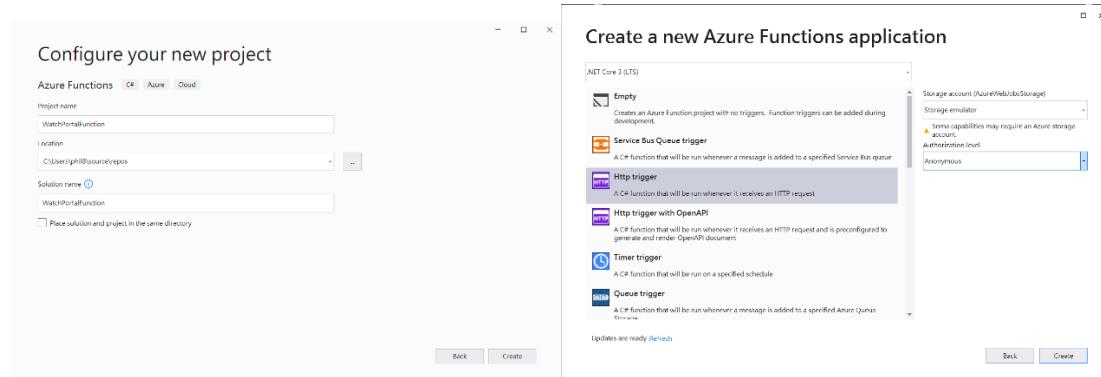
本模組重點教導如何使用 Visual Studio 的開發環境做 Azure Function 的開發，除了 Create Function、Publish/Deploy 功能，還教導 BreakPoint、Unit Test 的功能。



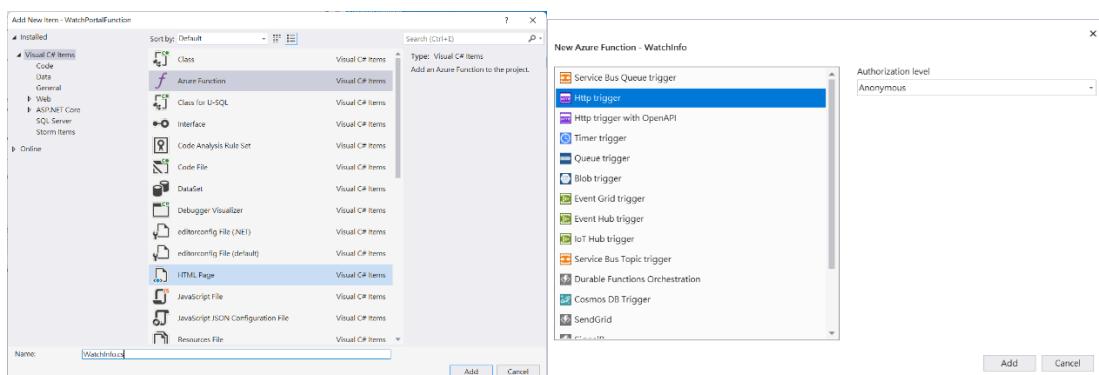
此模組使用 Visual Studio 進行 Azure Function 的開發，需要安裝以下套件。



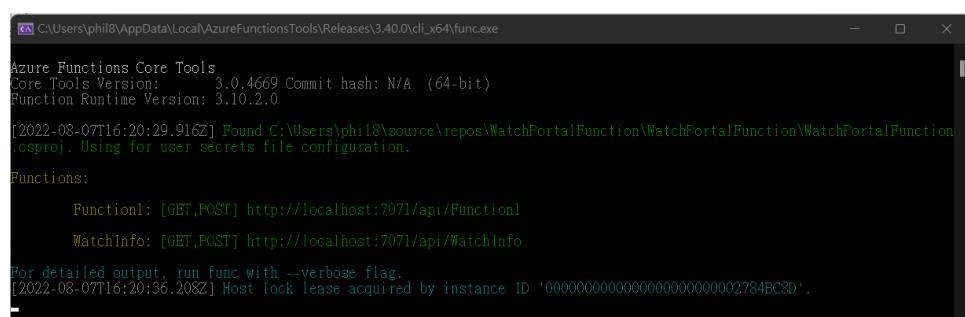
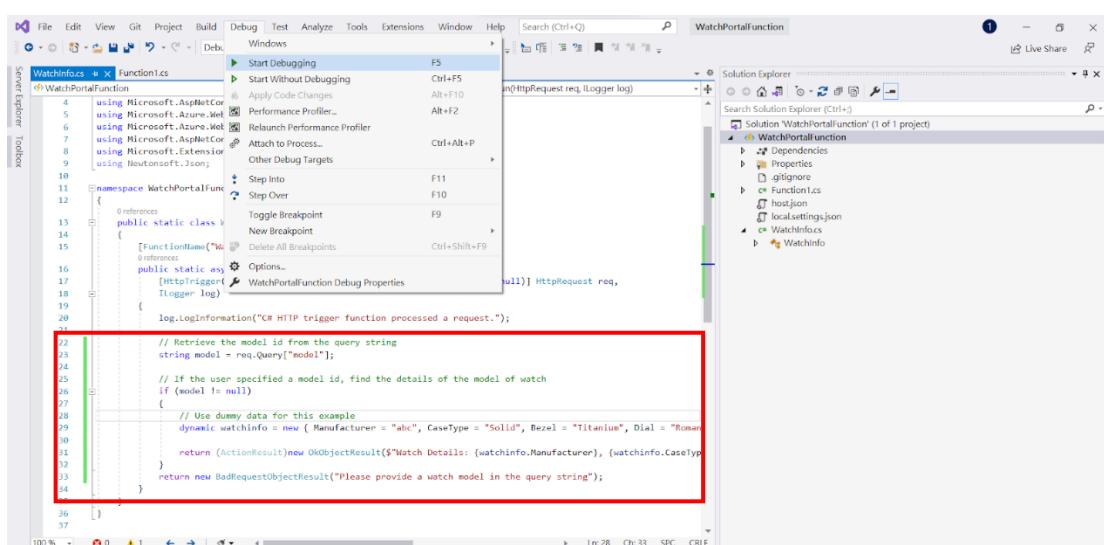
創立一個 Project，並創立第一個 Function (Http Trigger)，名為 Function1。



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



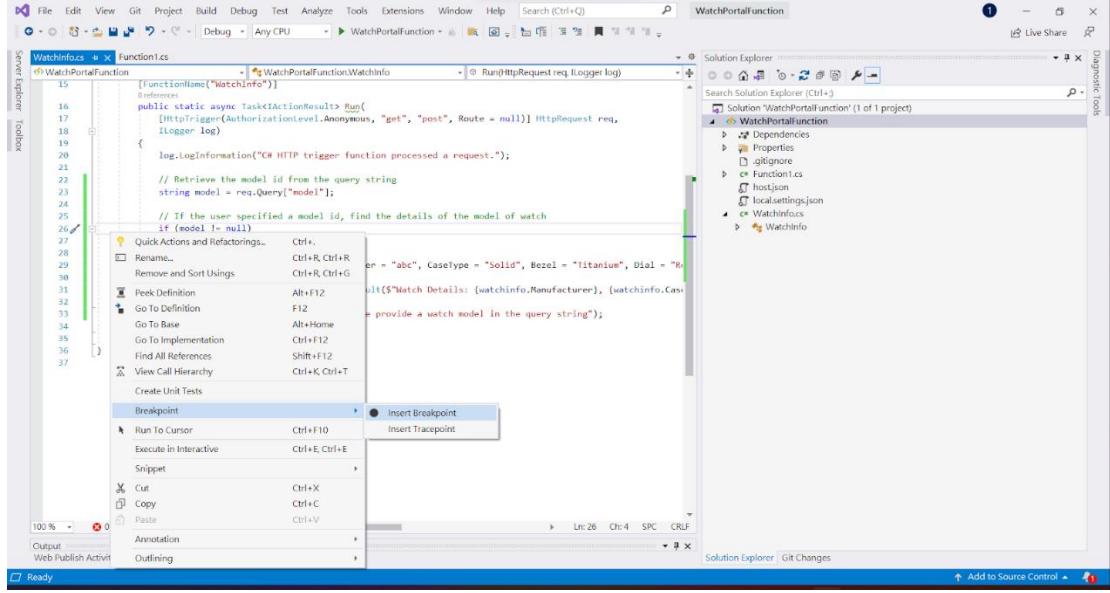
修改程式碼，程式會根據 GET/POST 引入的變數”model”回傳不同的結果。  
並使用 Debug 啟動 Azure Function。



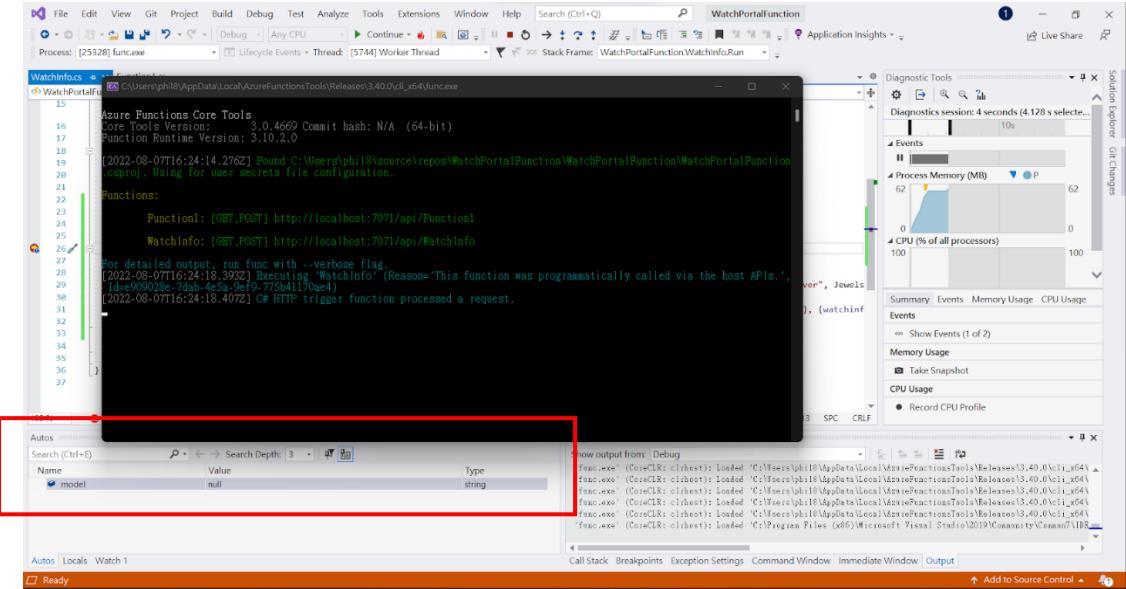
Http Trigger 結果如下圖所示，程式根據 GET/POST 的 model 變數決定輸出內容。



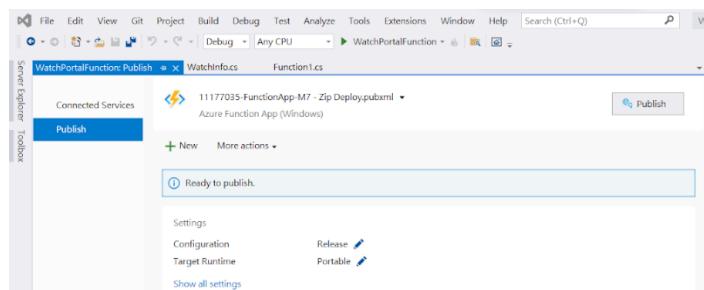
於 Visual Studio 中，可以設定 Breakpoint，可以逐行設定程式中斷點進入 Debugging Mode，查看程式執行當下各物件的狀態。



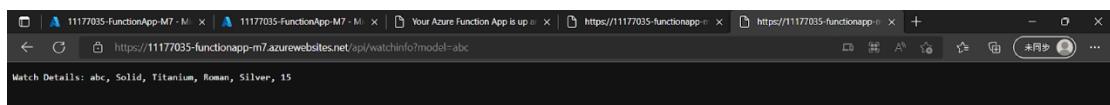
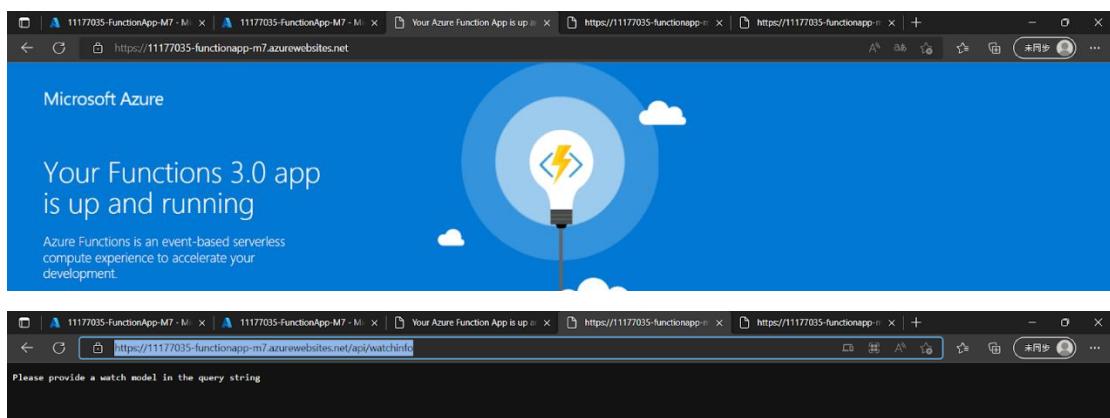
如 model 變數，在中斷點下能偵查當下的值。



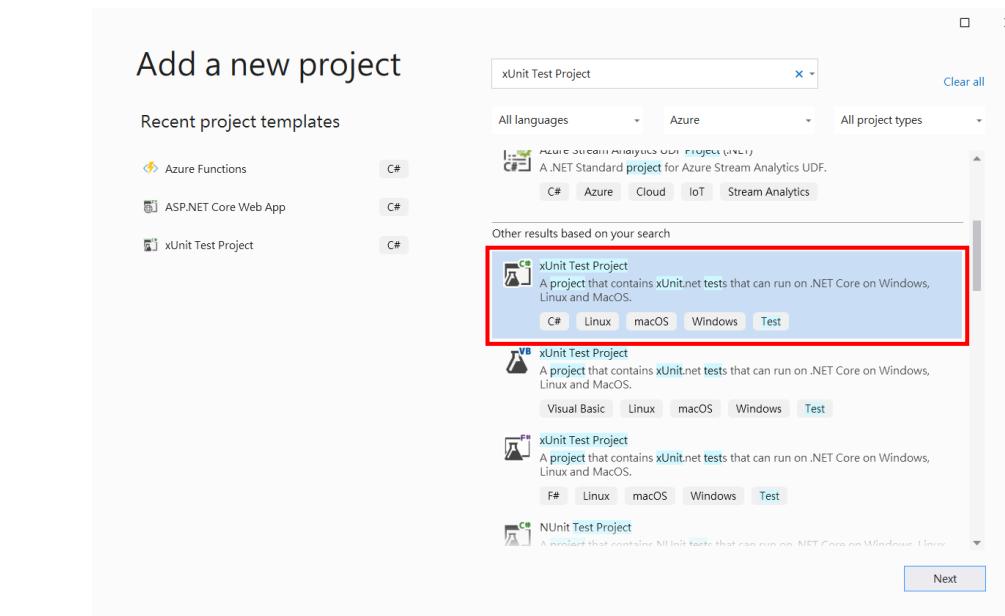
最後也能透過 IDE 佈署 Azure Function 至雲端。



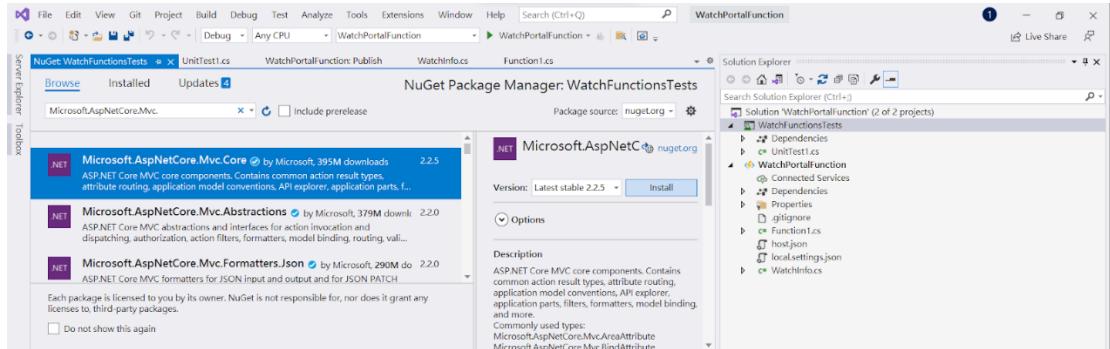
部署結果如下，與本地開發的功能在雲端再度呈現。



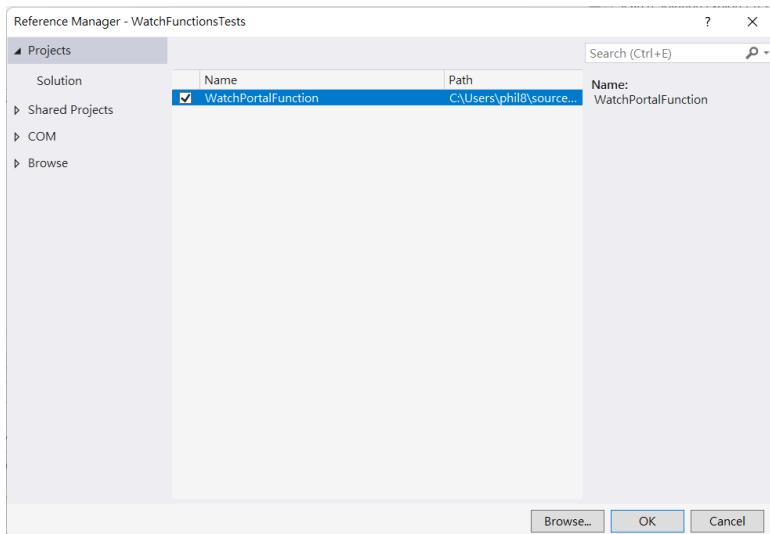
創立一個 Project 進行 Unit Test。



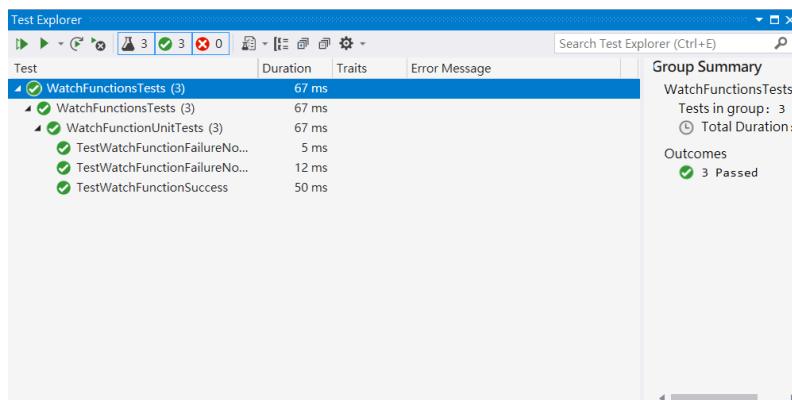
透過 NuGet，安裝 MVC 套件。



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



進行正常 Unit Test 的結果。

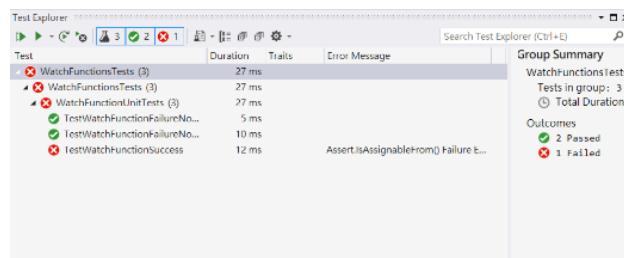


接著故意修改錯誤的程式碼，模擬開發錯誤的情境。

The screenshot shows the Visual Studio code editor with the 'WatchInfo.cs' file open. On line 23, there is a comment block that includes a line of code: 'string model = req.Query["modelId"];'. This line is highlighted with a red rectangular box, indicating it is the source of the bug being tested.

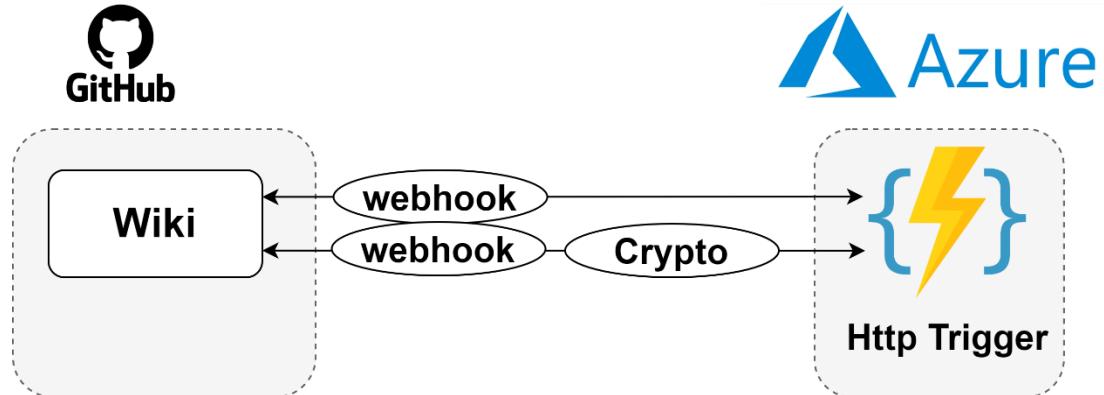
```
15     [FunctionName("WatchInfo")]
16     public static async Task<ActionResult> Run(
17         [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
18         ILogger log)
19     {
20         log.LogInformation("C# HTTP trigger function processed a request.");
21
22         // Retrieve the model id from the query string
23         string model = req.Query["modelId"];
24
25         // If the user specified a model id, find the details of the model of watch
26         if (model != null)
27         {
28             // Use dummy data for this example
29             dynamic watchinfo = new { Manufacturer = "abc", CaseType = "Solid", Bezel = "Titanium", Dial = "Roman" };
30
31             return (ActionResult)new OkObjectResult($"Watch Details: {watchinfo.Manufacturer}, {watchinfo.CaseType}");
32         }
33         return new BadRequestObjectResult("Please provide a watch model in the query string");
34     }

```

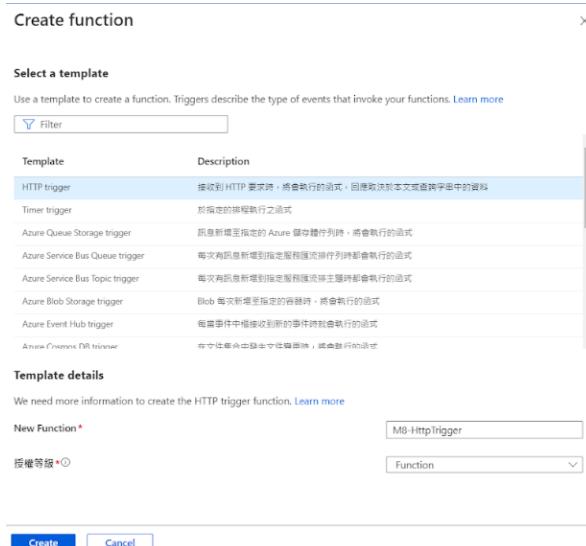


# Module 9: Monitor GitHub events by using a webhook with Azure Functions

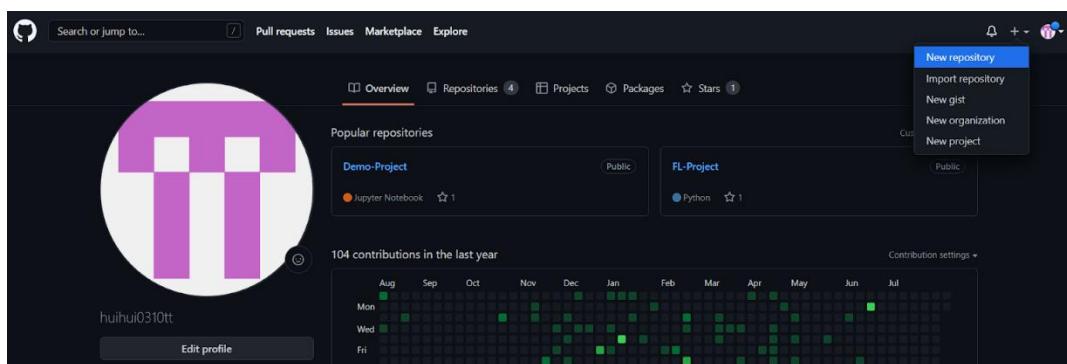
此模組重點在教導如何使用 Webhook 觸發，並進一步與 Azure Function 做配合。



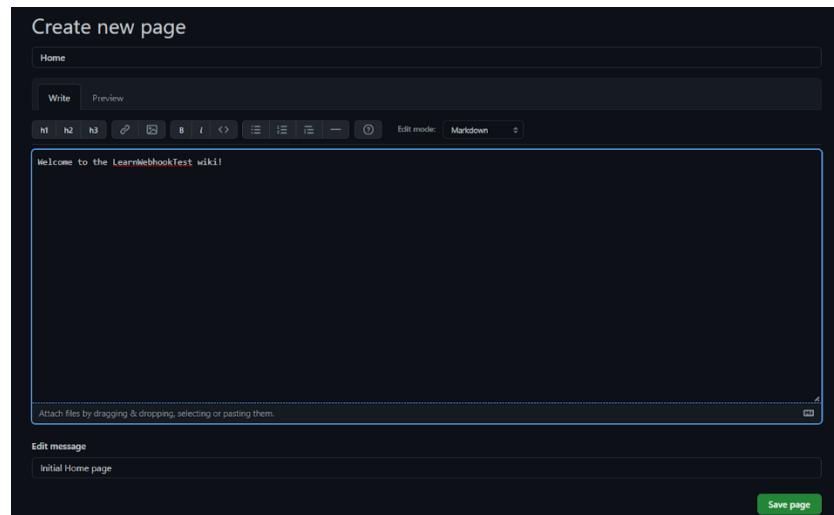
創建一個 Http Trigger 供 Github Webhook 做觸發。



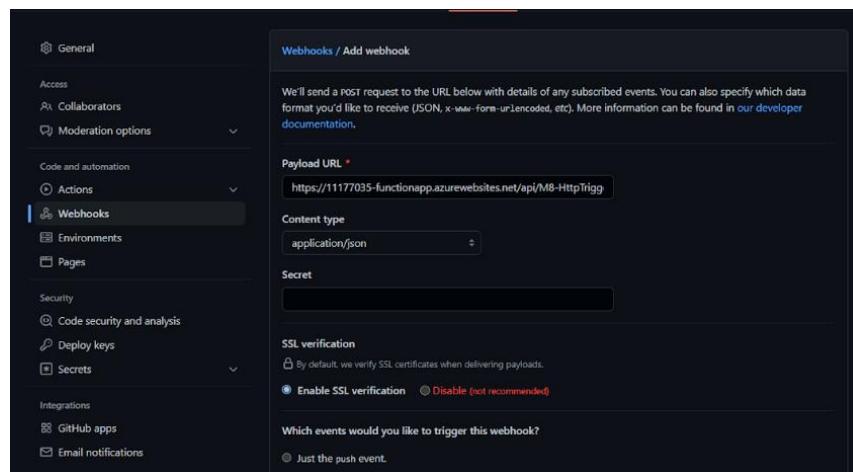
在 Github，創建一個倉儲。



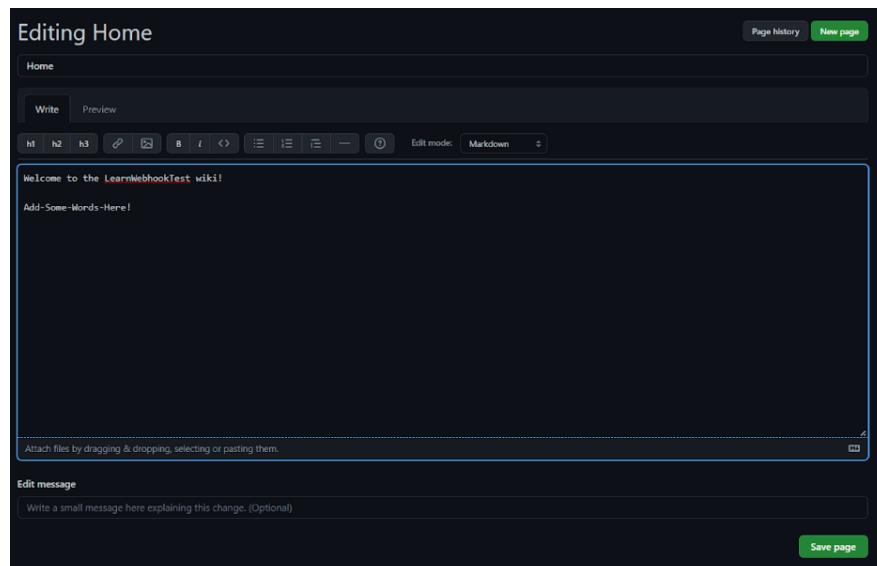
開啟一個 Wiki 頁面，作為觸發 Http 的來源。



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。

The screenshot shows the Azure Functions log viewer. A successful request is displayed with a status of 200. The Headers section shows standard HTTP headers like Content-Type and Date. The Body section contains the message: "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body".

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。

```
1  const Crypto = require('crypto');
2
3  module.exports = async function (context, req) {
4    // context.log('JavaScript HTTP trigger function processed a request.');
5
6    const hmac = Crypto.createHmac("sha1", "5_Lm0g0z4j8o70mfveUsT8xB20WetnqUk0jf0-QvxSEUAzFuXPaHvQ=");
7    const signature = hmac.update(JSON.stringify(req.body)).digest("hex");
8    const sha1Signature = `sha1=${signature}`;
9    const gitHubSignature = req.headers['x-hub-signature'];
10
11    if (!sha1Signature.localeCompare(gitHubSignature)) {
12      if (req.body.pages[0].title) {
13        context.res = {
14          body: `Page is ${req.body.pages[0].title}, Action is ${req.body.pages[0].action}, Event Type is ${req.headers['x-github-event']}`
15        };
16      } else {
17        context.res = {
18          status: 400,
19          body: ("Invalid payload for Wiki event")
20        };
21      }
22    } else {
23      context.res = {
24        status: 401,
25        body: "Signatures don't match"
26      };
27    }
28  };
29};
30};
```

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。

The screenshot shows the Azure Webhooks settings page. It displays a configuration for a webhook with the following details:

- Payload URL:** https://11177035-functionapp.azurewebsites.net/api/M8-HttpTrigger
- Content type:** application/json
- Secret:** nOg0z4j8o70mfveUsT8xB20WetnqUk0jf0-QvxSEUAzFuXPaHvQ=
- SSL verification:** Enabled (radio button selected)
- Which events would you like to trigger this webhook?** (This field is partially visible at the bottom)

On the right side, there is a detailed view of a recent delivery log:

- Request URL:** https://11177035-functionapp.azurewebsites.net/api/M8-HttpTrigger?code=5\_Lm0g0z4j8o70mfveUsT8...
- Request method:** POST
- Accept:** \*/\*
- content-type:** application/json
- User-Agent:** GitHub-Hookshot/0.0.0a
- X-GitHub-Delivery:** d5fc1f50-16c2-11ed-86de-deb66fd22415
- X-GitHub-Event:** pull  
X-GitHub-Hook-ID: 372378080
- X-GitHub-Hook-Installation-Target-ID:** 522378080
- X-GitHub-Hook-Installation-Target-Type:** repository
- X-Hub-Signature:** sha1=112a071a0c0a0a0bcb6fcacab3f0081a76ca8c
- X-Hub-Signature-256:** sha256=d42e30a221f607b742f3de178cfabb5b15d48836f93da5e5d5fc6ce7aef

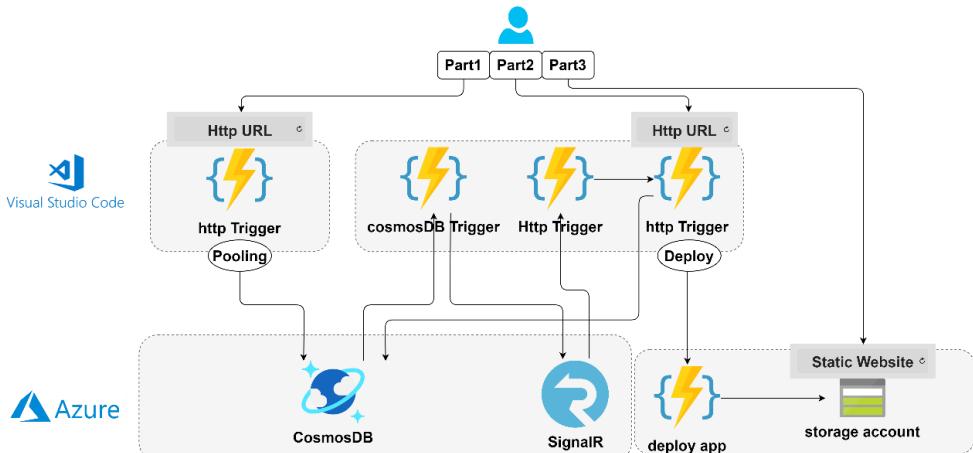
The log also shows the response from the Azure Function, indicating it completed successfully in 0.62 seconds.

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

The screenshot shows two side-by-side screenshots of the GitHub 'Webhooks / Manage webhook' settings page. The left screenshot shows the configuration of a webhook with a payload URL of <https://11177035-functionapp.azurewebsites.net/api/M8-HttpTrigger>, content type 'application/json', and a secret 'Non-Sense Word'. The right screenshot shows a successful test delivery with a status of 'Completed in 0.58 seconds' and a red '401' error message in the response. The response headers include GitHub-specific metadata like X-GitHub-Delivery, X-GitHub-Event, and X-GitHub-Hook-ID.

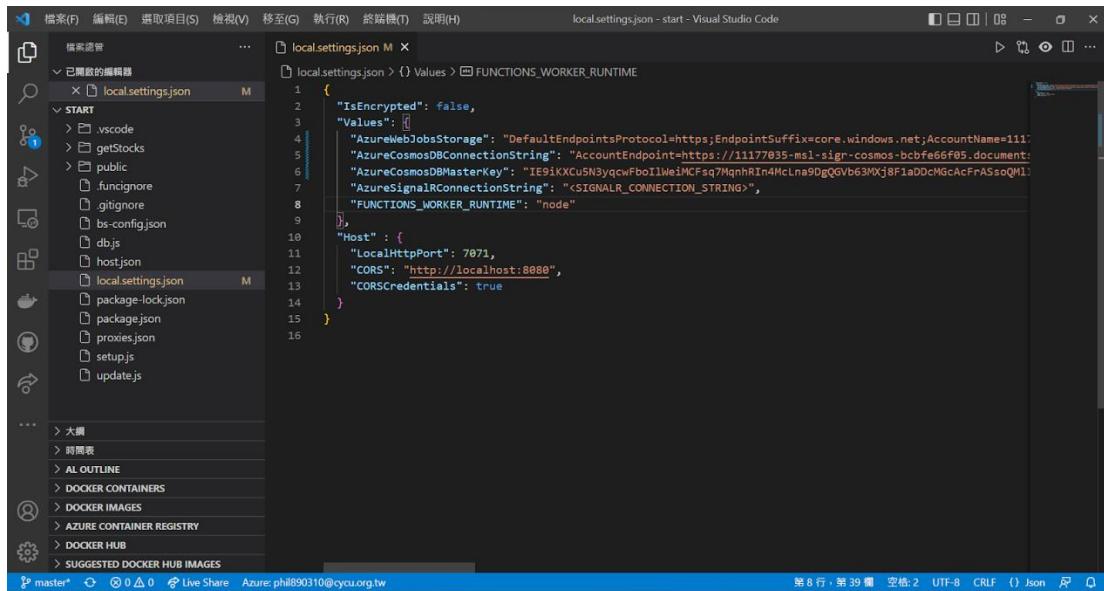
## Module 10: Enable automatic updates in a web application using Azure Functions and SignalR Service

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。



```
local.settings.json M
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=111",
    "AzureCosmosDBConnectionString": "AccountEndpoint=https://11177035-ms1-sigr-cosmos-bcbfe6f05.documentdb.azure.com/",
    "AzureCosmosDBMasterKey": "IE9iKXCu5N3yqcwFbOIWeimCFSq7MqnRIIn4McLna9DgQGvb63MXj8F1aDDcMGcAcFrASoQML",
    "AzureSignalRConnectionString": "<SIGNALR_CONNECTION_STRING>",
    "FUNCTIONS_WORKER_RUNTIME": "node"
  },
  "Host": {
    "LocalHttpPort": 7071,
    "CORS": "http://localhost:8080",
    "CORSCredentials": true
  }
}
```

使用 npm install 將股票資訊寫入 DB 中。

```
> microsoft-learn-func-signalr-start@1.0.0 setup C:\Users\phil8\Desktop\Serverless\serverless-demo\start
> node setup.js

Database created.
Collection created.
Seed data added.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
added 336 packages from 282 contributors and audited 404 packages in 14.479s
found 60 vulnerabilities (16 moderate, 39 high, 5 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
```

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

```
PS C:\Users\phil8\Desktop\Serverless\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

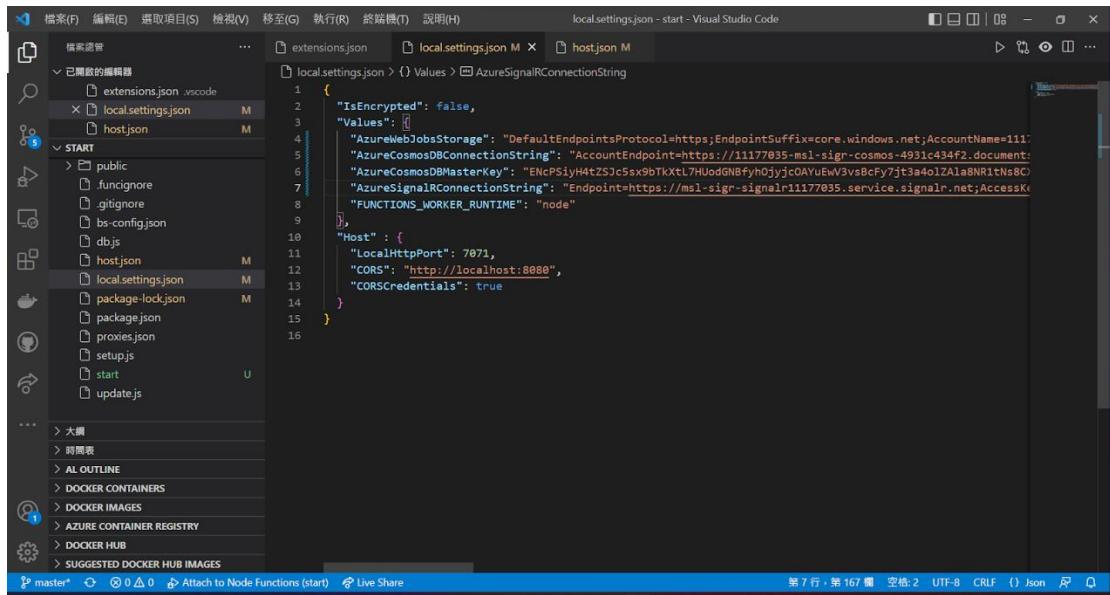
Read data from database.

Data updated: {"id": "e0eb6e85-176d-4ce6-89ae-1f699aaa0bab", "symbol": "ABC", "price": 95.07, "change": 4.93, "changeDirection": "-", "_rid": "Jyl+AKCqSWIBAAAAAAA=", "_self": "dbs/Jyl+AA=/colls/Jyl+AKCqSWI=/docs/Jyl+AKCqSWIBAAAAAAA=/", "_etag": "\"4100d7da-0000-0b00-0000-62f08c9c000\"", "_attachments": "attachments/", "_ts": 1659931804}
```

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。

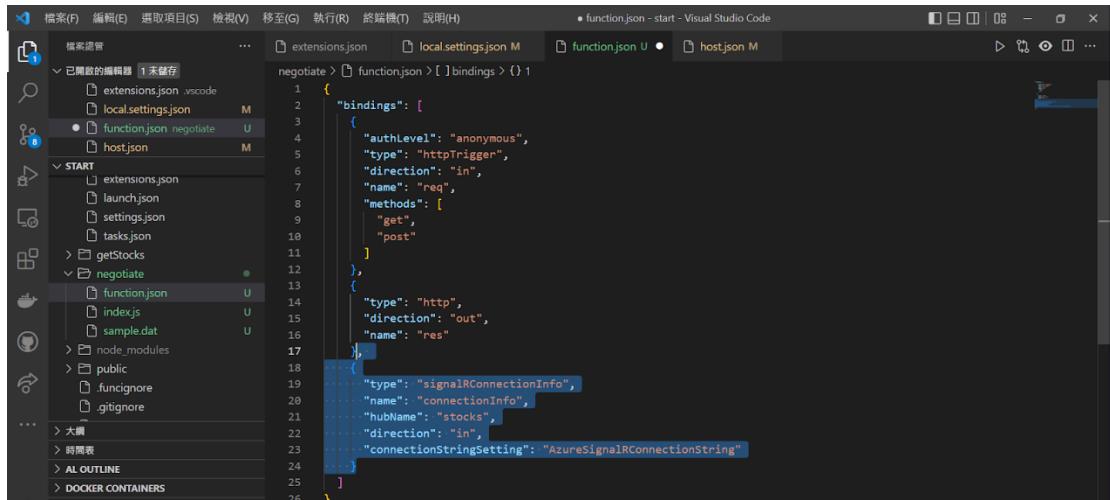


```

{
    "IsEncrypted": false,
    "Values": [
        "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=11177035-msl-signr-cosmos-4931c434f2.documentdb.azure.com;AccountKey=ENCPsiyH4tZSJ5sx9bTktL7HJUdgNBfyhOjyc0AVUEwV3vSBFy7jt3a4o1ZAla8NR1tNs8C",
        "AzureCosmosDBConnectionString": "AccountEndpoint=https://11177035-msl-signr-cosmos-4931c434f2.documentdb.azure.com;AccountKey=ENCPsiyH4tZSJ5sx9bTktL7HJUdgNBfyhOjyc0AVUEwV3vSBFy7jt3a4o1ZAla8NR1tNs8C",
        "AzureSignalRConnectionString": "Endpoint=https://msl-signr-signr-11177035.service.signalr.net;AccessKey=FUNCTIONS_WORKER_RUNTIME:node",
        "Host": {
            "LocalHttpPort": 7071,
            "Cors": "http://localhost:8080",
            "CorsCredentials": true
        }
    ]
}

```

新增 Http Trigger，添增 SignalR 的 Input Binding。

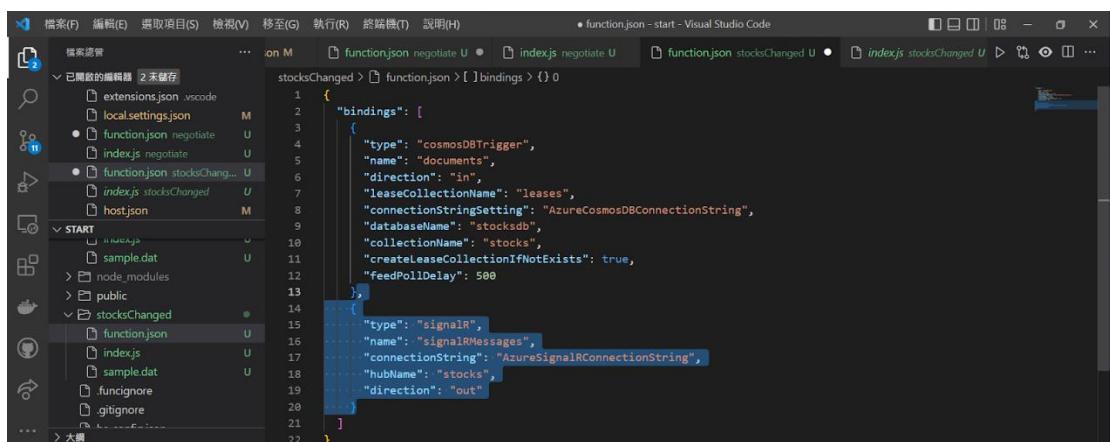


```

{
    "bindings": [
        {
            "authLevel": "anonymous",
            "type": "httpTrigger",
            "direction": "in",
            "name": "req",
            "methods": [
                "get",
                "post"
            ]
        },
        {
            "type": "http",
            "direction": "out",
            "name": "res"
        },
        {
            "type": "signalRConnectionInfo",
            "name": "connectionInfo",
            "hubName": "stocks",
            "direction": "in",
            "connectionStringSetting": "AzureSignalRConnectionString"
        }
    ]
}

```

新增 CosmosDB Trigger，添增 SignalR 的 Output Binding。

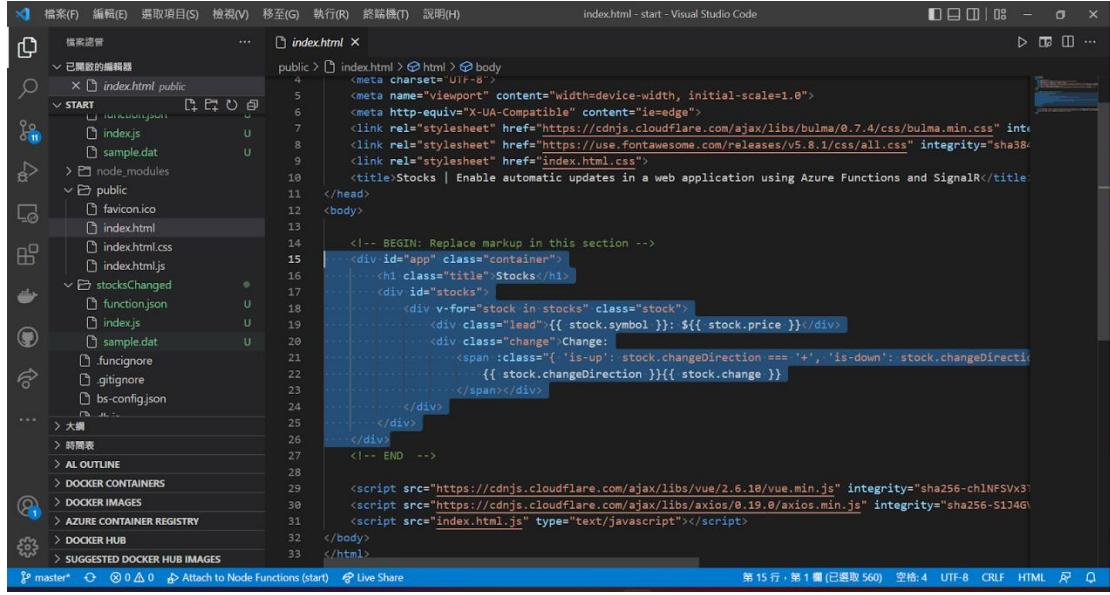


```

{
    "bindings": [
        {
            "type": "cosmosDBTrigger",
            "name": "documents",
            "direction": "in",
            "leaseCollectionName": "leases",
            "connectionStringSetting": "AzureCosmosDBConnectionString",
            "databaseName": "stocksdb",
            "collectionName": "stocks",
            "createLeaseCollectionIfNotExists": true,
            "feedPollDelay": 500
        },
        {
            "type": "signalR",
            "name": "signalRMessages",
            "connectionString": "AzureSignalRConnectionString",
            "hubName": "stocks",
            "direction": "out"
        }
    ]
}

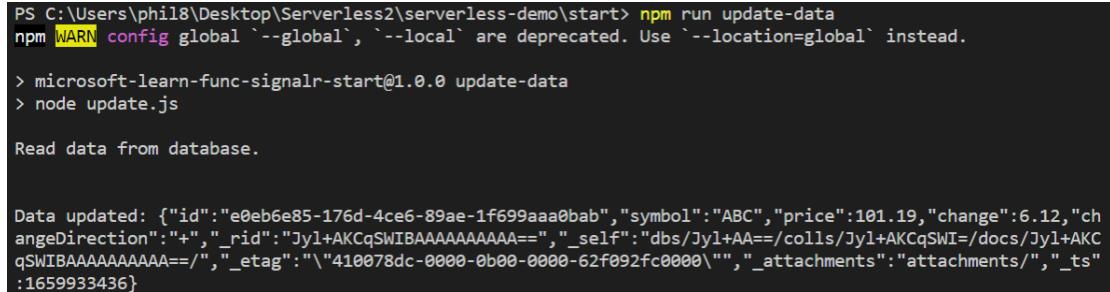
```

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



The screenshot shows the Visual Studio Code interface with the project structure on the left and the code editor on the right. The code editor displays the `index.html` file, which contains HTML and JavaScript code for a stock information application. The code includes imports for CSS and JS files, and a template for displaying stock data.

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。

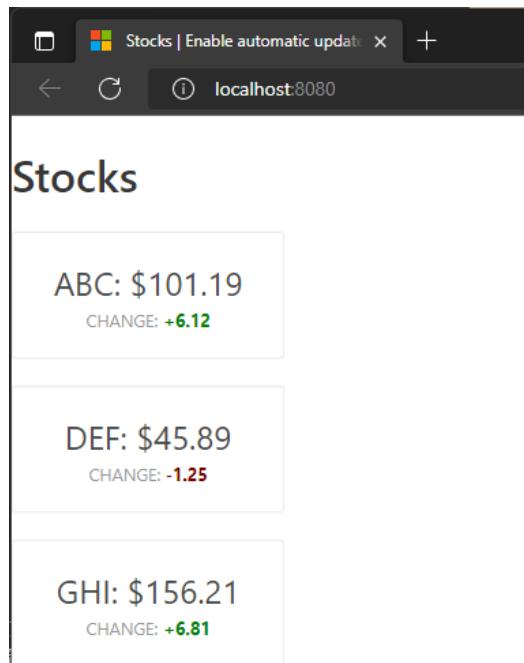


```
PS C:\Users\phil8\Desktop\Serverless2\serverless-demo\start> npm run update-data
npm [WARN] config global `--global`, `--local` are deprecated. Use `--location=global` instead.

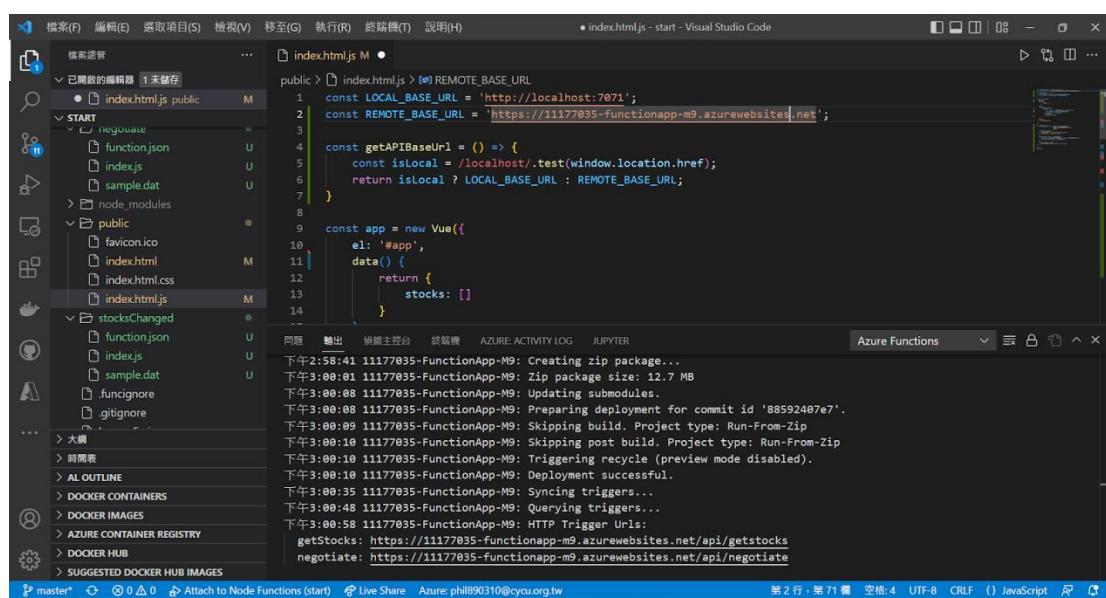
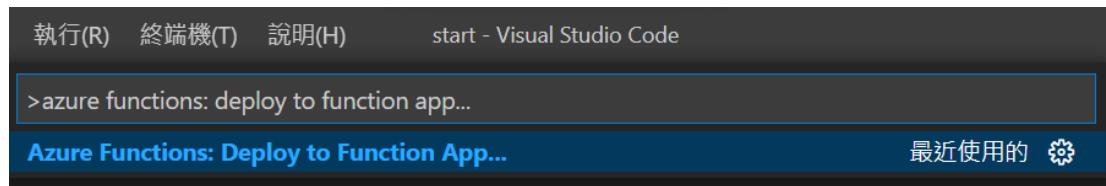
> microsoft-learn-func-signalr-start@1.0.0 update-data
> node update.js

Read data from database.

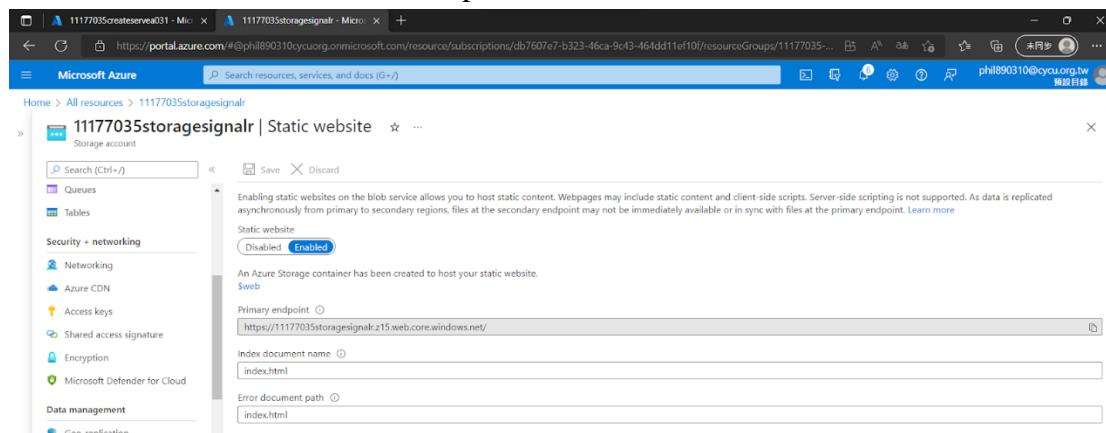
Data updated: {"id":"e0eb6e85-176d-4ce6-89ae-1f699aaa0bab","symbol":"ABC","price":101.19,"change":6.12,"changeDirection":"+","_rid":"Jyl+AKCqSWIBAAAAAAA==","_self":"dbs/Jyl+AA==/colls/Jyl+AKCqSWI=/docs/Jyl+AKCqSWIBAAAAAAA==/","_etag":"/410078dc-0000-0b00-0000-62f092fc0000","","_attachments":"attachments/","_ts":1659933436}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



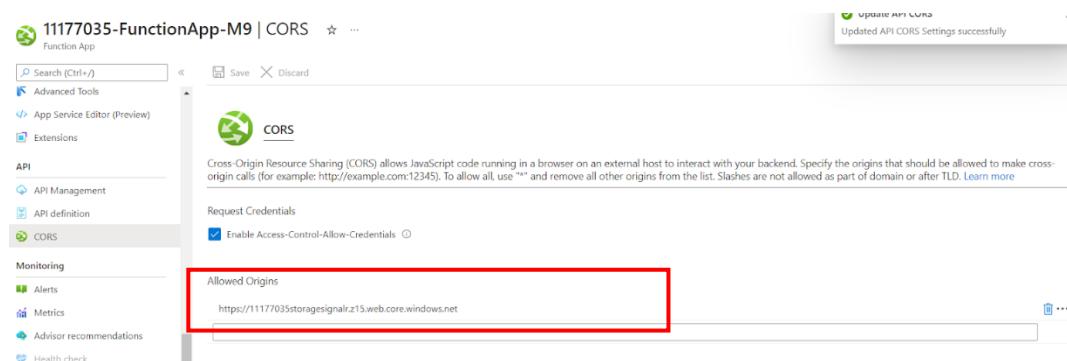
### 第三部分，部署 Function App 至雲端。



### 設定 Static Website 作為 Endpoint。



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

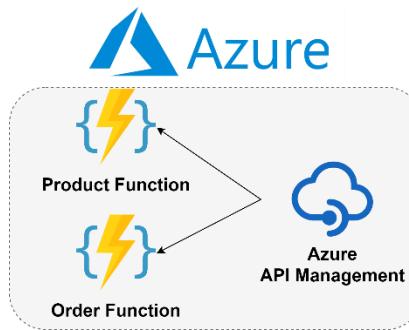


最終在 Static Website 運行 App。



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

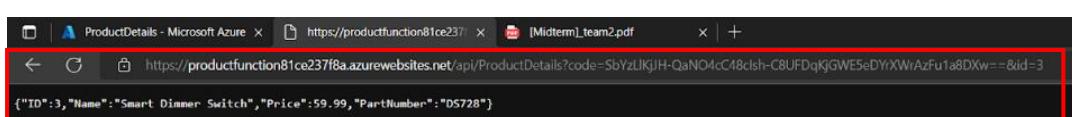
此模組主要使用 Azure API Management 管理多個 Azure Function，將多個 Microservice 集合至一個 API 當中。



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。

The screenshot shows the Azure Functions developer portal interface. A red box highlights the 'Output' tab under the 'HTTP response code' section, which displays '200 OK'. Below it, the 'HTTP response content' section shows a JSON object: { "ID": 3, "Name": "Smart Dimmer Switch", "Price": 59.99, "PartNumber": "DS728" }. The bottom left corner of the interface shows a log message: 'Connected! 2022-08-09T01:18:20 Welcome, you are now connected to log-streaming service. The default timeout is 2 hours. Change the timeout with the App setting SCM\_LOGSTREAM\_TIMEOUT (in seconds)'.



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

The first screenshot shows the Microsoft Azure portal with the 'API Management' section selected. A red box highlights the 'Create new' button under the 'API' management section. The second screenshot shows the 'Link API' step, where a red box highlights the 'Link API' button. The third screenshot is a modal titled 'Create from Function App' with a red box highlighting the 'Basic' tab. It shows fields for 'Function App' (selected as 'ProductFunction81ce237f8a'), 'Display name' (ProductFunction81ce237f8a), 'Name' (productfunction81ce237f8a), 'API URL suffix' (products), and 'Base URL' (https://onlinestore-productfunction81ce237f8a.azure-api.net/products). The 'Create' button is at the bottom right.

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

The screenshot shows the 'Test' tab for the 'ProductDetails' API operation. The 'Request URL' is https://onlinestore-productfunction81ce237f8a.azure-api.net/products/ProductDetails?id=1. The 'HTTP request' pane shows a GET request with headers including 'Host', 'Ocp-Apim-Subscription-Key', and 'Ocp-Apim-Trace'. The 'HTTP response' pane shows a successful 200 OK response with a JSON payload containing product details: { "ID": 1, "Name": "Smart Speaker", "Price": 129.99, "StockLevel": 500 }.

測試另一個 OrderFunction，GET name: Chiba。

The screenshot shows the Azure Functions Test Explorer interface. On the left, the code for `function.json` is displayed:

```
1  {
2      "generatedBy": "Microsoft.NET.Sdk.Functions-1.0.24",
3      "configurationSource": "attributes",
4      "bindings": [
5          {
6              "type": "httpTrigger",
7              "methods": [
8                  "get",
9                  "post"
10             ],
11             "authLevel": "function",
12             "name": "req"
13         },
14     ],
15     "disabled": false,
16     "scriptFile": "../bin/OrderShippingFunc.dll",
17 }
```

On the right, the **Output** tab shows the HTTP response content:

```
[{"ID":72945, "CustomerFirstName": "Yuki", "CustomerLastName": "Chiba", "Total": 442.5, "Shipped": false}]
```

Below the interface, a browser window shows the result of the API call:

```
{"ID":72945, "CustomerFirstName": "Yuki", "CustomerLastName": "Chiba", "Total": 442.5, "Shipped": false}
```

添加 Function 至既有的 OnlineStore API。

The screenshot shows the Microsoft Azure portal. A new API is being created from an existing Function App. The **Create from Function App** dialog is open, with the following fields filled:

- Function App: OrderFunction3f2e78290c
- Display name: OrderFunction3f2e78290c
- Name: orderfunction3f2e78290c
- API URL suffix: orders
- Base URL: https://onlinestore-productfunction81ce237f8a.azure-api.net/orders

The entire dialog is highlighted with a red box.

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

The screenshot shows the API Management console. A successful GET request to the `OrderDetails` API is displayed:

**HTTP response**

```
HTTP/1.1 200 OK
cache-control: private
content-encoding: gzip
content-type: application/json; charset=utf-8
date: Tue, 09 Aug 2022 01:56:19 GMT
request-context: appId=cid-v1:7e0566b5-1f1d-4205-aa2a-78020e10a7a0
transfer-encoding: chunked
vary: Accept-Encoding,Origin

{
    "ID": 72945,
    "CustomerFirstName": "Yuki",
    "CustomerLastName": "Chiba",
    "Total": 442.5,
    "Shipped": false
}
```

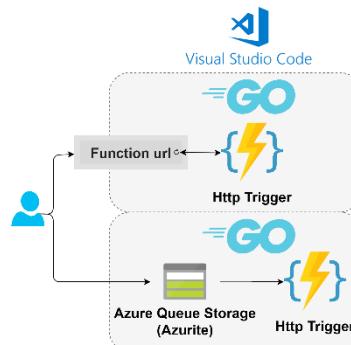
## 使用 API Management 的 Gateway URL 以及 Key 測試 Microservice 服務。

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

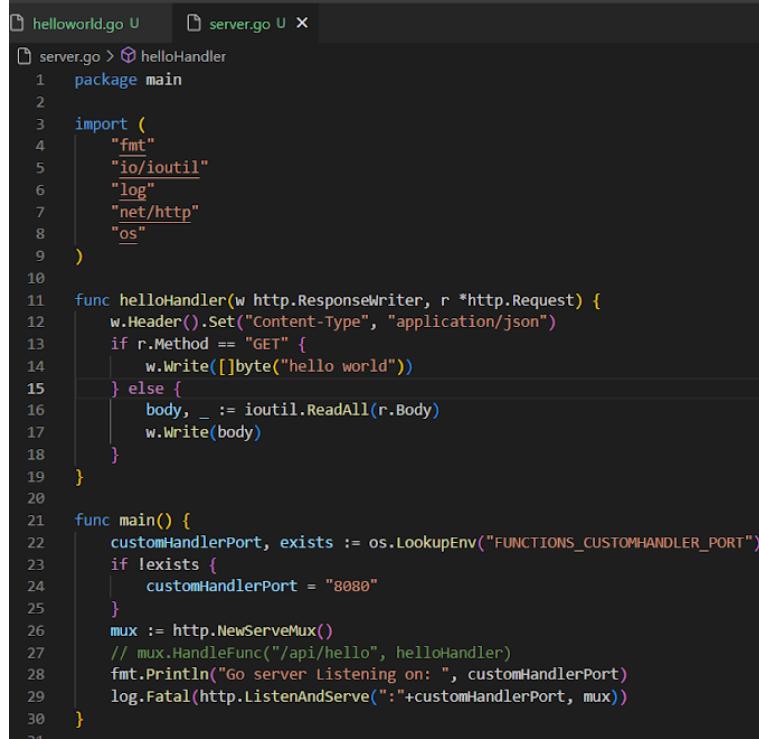
此模組重點強調開發人員可以使用任何熟悉的語言進行開發，只要語言支持 Http 工具，此模組使用 GoLang 為例子進行展示，並示範了另一個 Storage Account 的使用（Azurite，Local Environment 版本的 Storage Account）



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

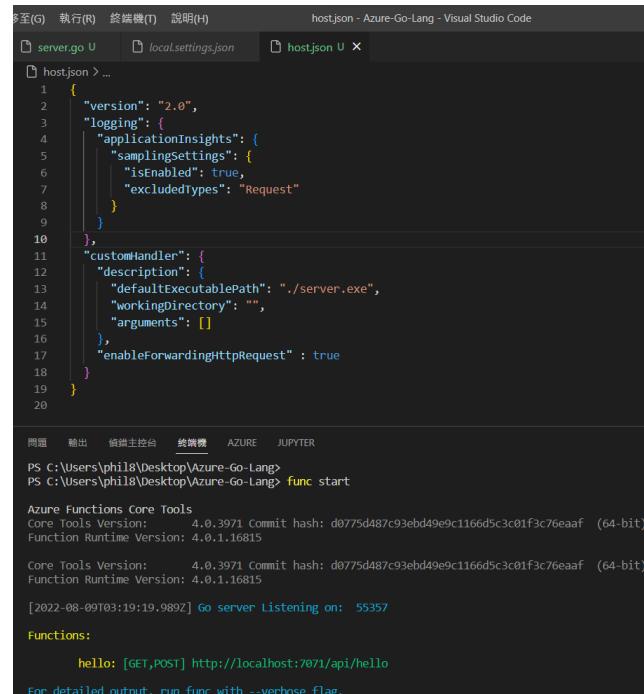
```
helloworld.go > main
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, World")
7 }
8
```

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 X
server.go > helloHandler
1 package main
2
3 import (
4     "fmt"
5     "io/ioutil"
6     "log"
7     "net/http"
8     "os"
9 )
10
11 func helloHandler(w http.ResponseWriter, r *http.Request) {
12     w.Header().Set("Content-Type", "application/json")
13     if r.Method == "GET" {
14         w.Write([]byte("hello world"))
15     } else {
16         body, _ := ioutil.ReadAll(r.Body)
17         w.Write(body)
18     }
19 }
20
21 func main() {
22     customHandlerPort, exists := os.LookupEnv("FUNCTIONS_CUSTOMHANDLER_PORT")
23     if !exists {
24         customHandlerPort = "8080"
25     }
26     mux := http.NewServeMux()
27     // mux.HandleFunc("/api/hello", helloHandler)
28     fmt.Println("go server Listening on: ", customHandlerPort)
29     log.Fatal(http.ListenAndServe(": "+customHandlerPort, mux))
30 }
```

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
host.json U localsettings.json X
host.json > ...
1 {
2     "version": "2.0",
3     "logging": {
4         "applicationInsights": {
5             "samplingSettings": {
6                 "isEnabled": true,
7                 "excludedTypes": "Request"
8             }
9         },
10    },
11   "customHandler": {
12       "description": {
13           "defaultExecutablePath": "./server.exe",
14           "workingDirectory": "",
15           "arguments": []
16       },
17       "enableForwardingHttpRequest" : true
18   }
19 }
20
```

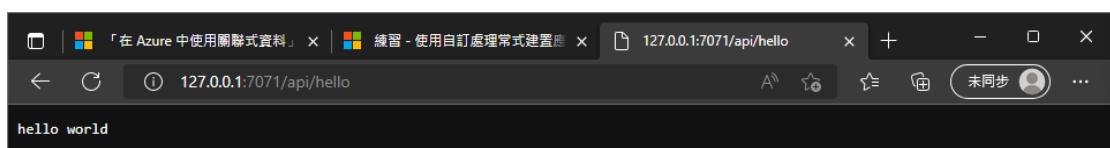
問題 輸出 值端主控台 編譯機 AZURE JUPYTER

```
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: d0775d487c93ebd49e9c1166d5c3c01f3c76aaaf (64-bit)
Function Runtime Version: 4.0.1.16815

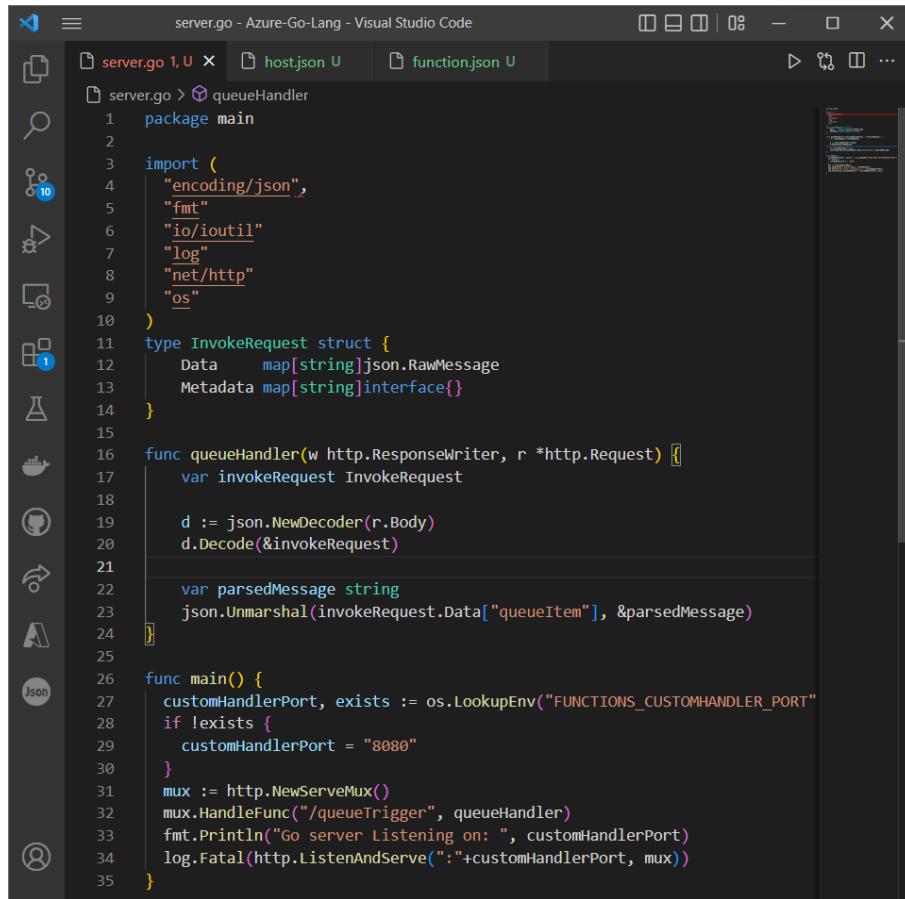
Core Tools Version:      4.0.3971 Commit hash: d0775d487c93ebd49e9c1166d5c3c01f3c76aaaf (64-bit)
Function Runtime Version: 4.0.1.16815

[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。



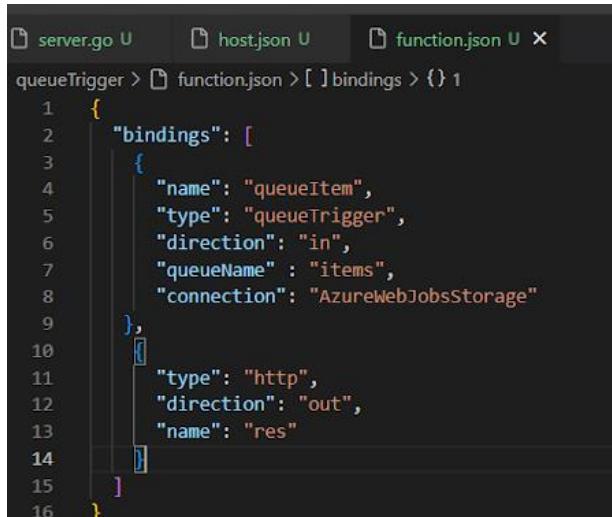
第二部分，再新增一個 Project，在語言的部分特別選定 Custom Handler，新增一個 Http Trigger，更換範例程式碼，加入 Http 服務，讀取 QueueStorage 資訊。



```
server.go 1, U host.json U function.json U
server.go > queueHandler
package main

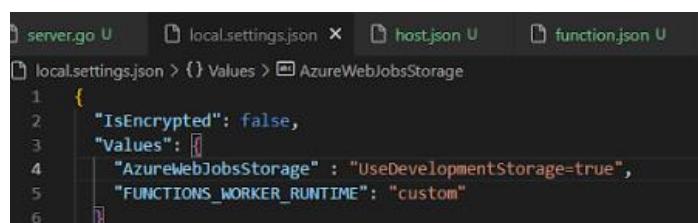
import (
    "encoding/json",
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)
type InvokeRequest struct {
    Data map[string]json.RawMessage
    Metadata map[string]interface{}
}
func queueHandler(w http.ResponseWriter, r *http.Request) {
    var invokeRequest InvokeRequest
    d := json.NewDecoder(r.Body)
    d.Decode(&invokeRequest)
    var parsedMessage string
    json.Unmarshal(invokeRequest.Data["queueItem"], &parsedMessage)
}
func main() {
    customHandlerPort, exists := os.LookupEnv("FUNCTIONS_CUSTOMHANDLER_PORT")
    if !exists {
        customHandlerPort = "8080"
    }
    mux := http.NewServeMux()
    mux.HandleFunc("/queueTrigger", queueHandler)
    fmt.Println("Go server Listening on: ", customHandlerPort)
    log.Fatal(http.ListenAndServe(": "+customHandlerPort, mux))
}
```

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



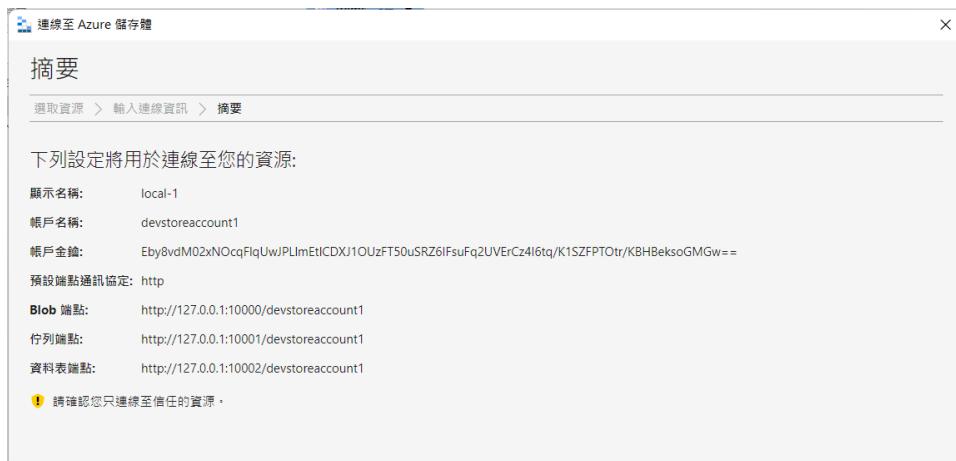
```
server.go U host.json U function.json U
queueTrigger > function.json > [ ] bindings > {} 1
{
    "bindings": [
        {
            "name": "queueItem",
            "type": "queueTrigger",
            "direction": "in",
            "queueName": "items",
            "connection": "AzureWebJobsStorage"
        },
        {
            "type": "http",
            "direction": "out",
            "name": "res"
        }
    ]
}
```

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

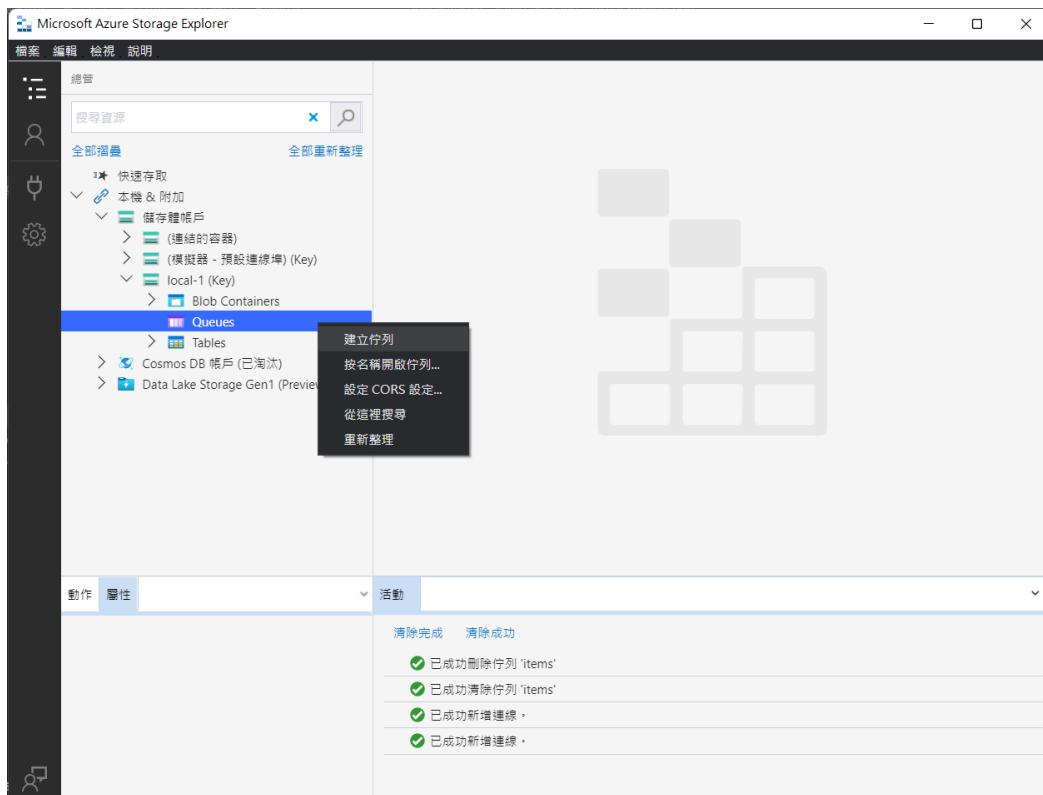


```
server.go U local.settings.json X host.json U function.json U
local.settings.json > {} Values > AzureWebJobsStorage
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "UseDevelopmentStorage=true",
        "FUNCTIONS_WORKER_RUNTIME": "custom"
    }
}
```

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。

```
PS C:\Users\phil8\Desktop\Azure-Go-Lang> conda deactivate pytorch
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

[2022-08-09T03:54:04.884Z] Go server Listening on: 55995

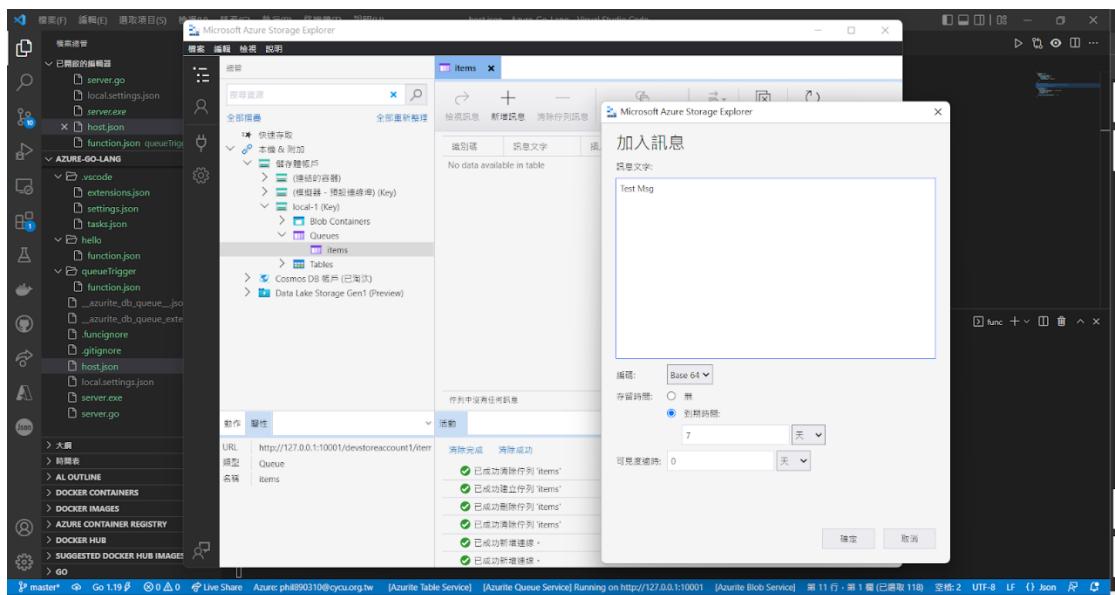
Functions:

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

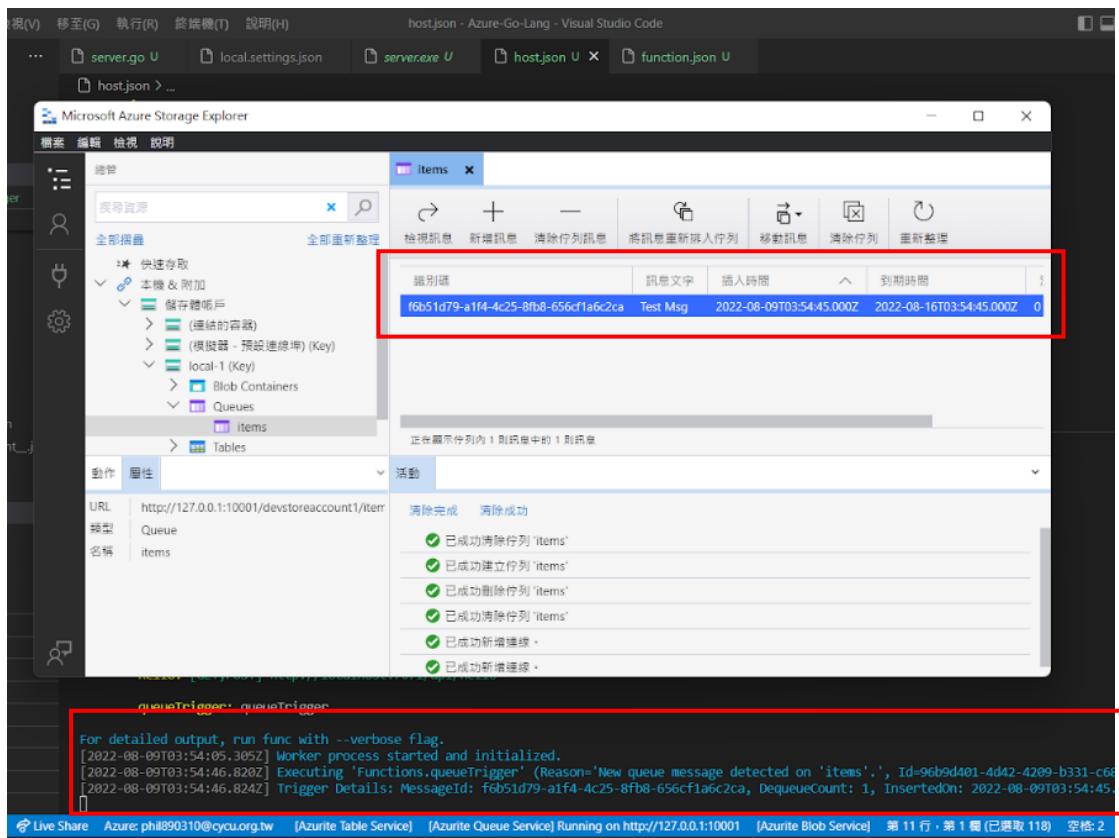
    queueTrigger: queueTrigger

For detailed output, run func with --verbose flag.
[2022-08-09T03:54:05.305Z] Worker process started and initialized.
```

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

The screenshot shows a user profile for Yan-Hui Lin (huihui0310tt@gmail.com). The profile includes a blue circular icon, the user's name, email, and a digital trophy icon. Below the profile are statistics: 63 Badges, 10 Trophies, 0 Reputation points, 0 Accepted answers, 0 Following, and 0 Followers. A progress bar indicates Level 9 at 86,850/106,299 XP. The main content area is divided into sections: Activity, Achievements, Training, Certifications, Learning Paths, Modules, Courses & More, Q&A, Achievements (selected), Collections, and Transcript. The Achievements section lists nine trophies, with one specifically for 'Create serverless applications' highlighted by a red box.

Achievement	Description	Completed On
TROPHY	Process and Translate Speech with Azure Cognitive Speech Services	Completed on 9/20/2022
TROPHY	Microsoft Azure AI Fundamentals: Explore natural language processing	Completed on 9/20/2022
TROPHY	Microsoft Azure AI Fundamentals: Explore visual tools for machine learning	Completed on 9/19/2022
<b>TROPHY</b>	<b>Create serverless applications</b>	<b>Completed on 9/19/2022</b>
TROPHY	Work with relational data in Azure	Completed on 9/16/2022
TROPHY	Store data in Azure	Completed on 9/15/2022
TROPHY	Deploy a website to Azure with Azure App Service	Completed on 8/23/2022
TROPHY	Process and translate text with Azure Cognitive Services	Completed on 8/15/2022
TROPHY	Create computer vision solutions with Azure Cognitive Services	

## 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 可以把章節切成兩個部分。

## Reference

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