CSS 532 Internet of Things

HW2 Report

**Student**: *Dazhi* *Li*  **NetID**: *dazhili* **Student ID**: *2400330*

# 1.Impelemation

### 1.1Implementation hierarchy

The whole architecture of my cloud services and programs are shown in fig1.

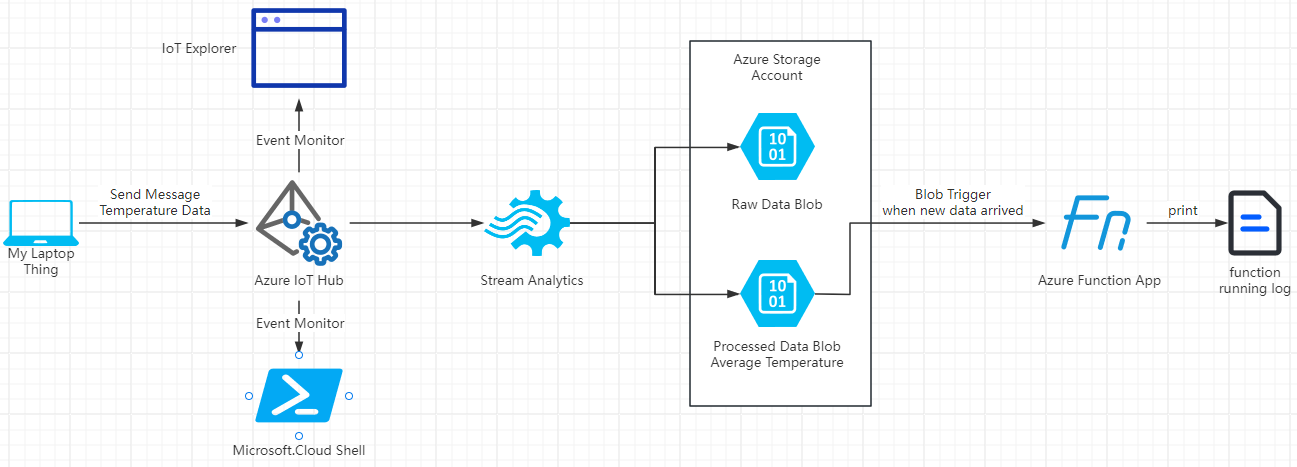
My laptop is the only thing in this IoT network, which is the start of the whole program. When I run my IoT client program on my laptop, it sends an array of temperature data to the Azure IoT Hub. Meanwhile, the Azure CLI (running on Microsoft Cloud Shell) and IoT Explorer are monitoring the event triggered by my laptop. Once the event is precepted by those, user could easily observe the temperature data sent from my laptop. After that, Azure system analytics is going to do data processing. That includes raw data passing to Azure Blob file and calculating the average temperature data from that array of temperature data mentioned before and then passing it to another Azure Blob file. Finally, An Azure function keeps monitoring the blob file which is used for storing processed data. Once the newly arrived average temperature data is documented in that blob file, my function will download the whole blob file and print the data into the log, which could also be accessed by the user.

Fig1. Cloud Service and Program Architecture

### 1.2Implementation Checklist

#### 1.2.1 Configure your laptop as an IoT device, which can communicate with Azure IoT Hub.

This step is easier than what we did in hw1 on AWS IoT Core. First of all, create a IoT Hub under a certain resource group. User could create a IoT Hub on webpage or by Azure CLI. My IoT Hub is shown in fig2.

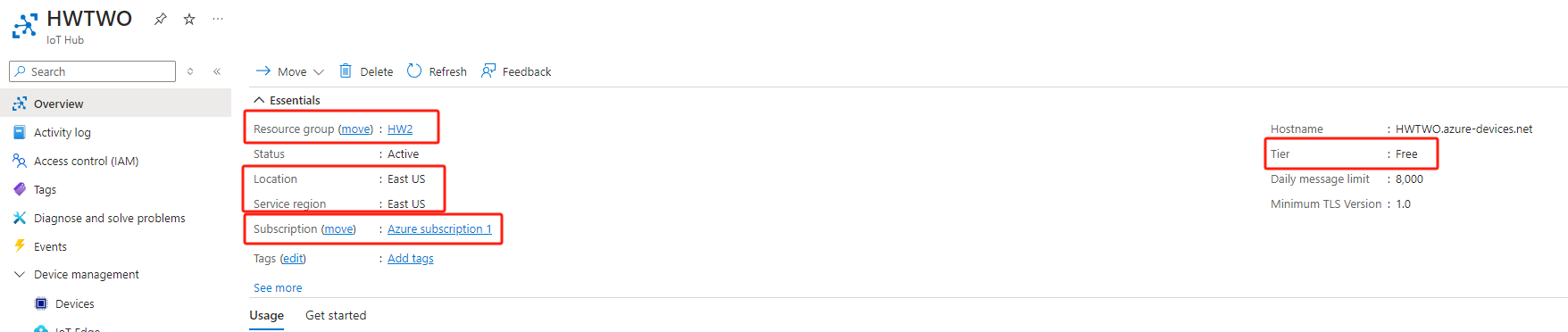


Fig2. My IoT Hub Configuration

#### After that, I created a device by the button, device, on the left column of the IoT Hub main page. By clicking on “Add Device” button and filling out a form, I got a virtual device information which contains credentials for further connection. My device configuration is shown in Fig3. Usually, the thing creator need to remember the connection string.

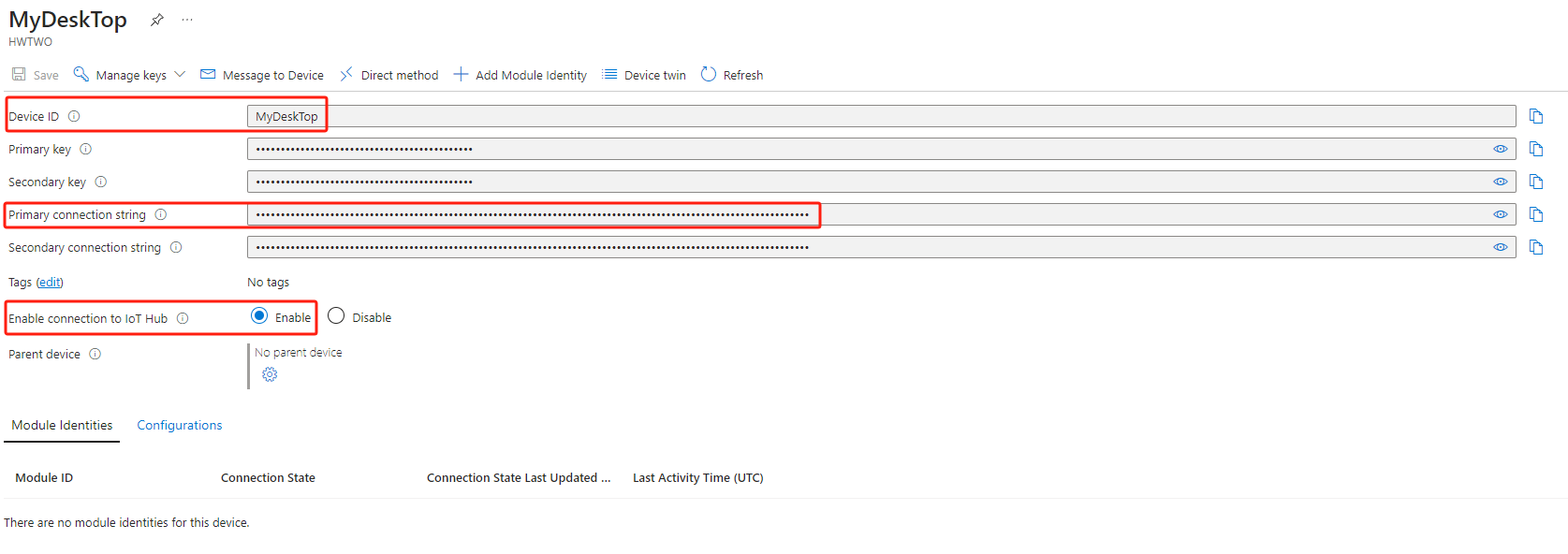
Then, I downloaded the quick start demo of Azure IoT Hub and running it on my own computer. The key of identifying my laptop to the virtual thing I mentioned above, is adding connection string into my laptop’s system variables. If programs run on some IDE like PyCharm, user is suggested to adding those system variables into their running configuration since the usage of virtual environment. My system environment was set as in Fig4.

Fig3. My Device Configuration

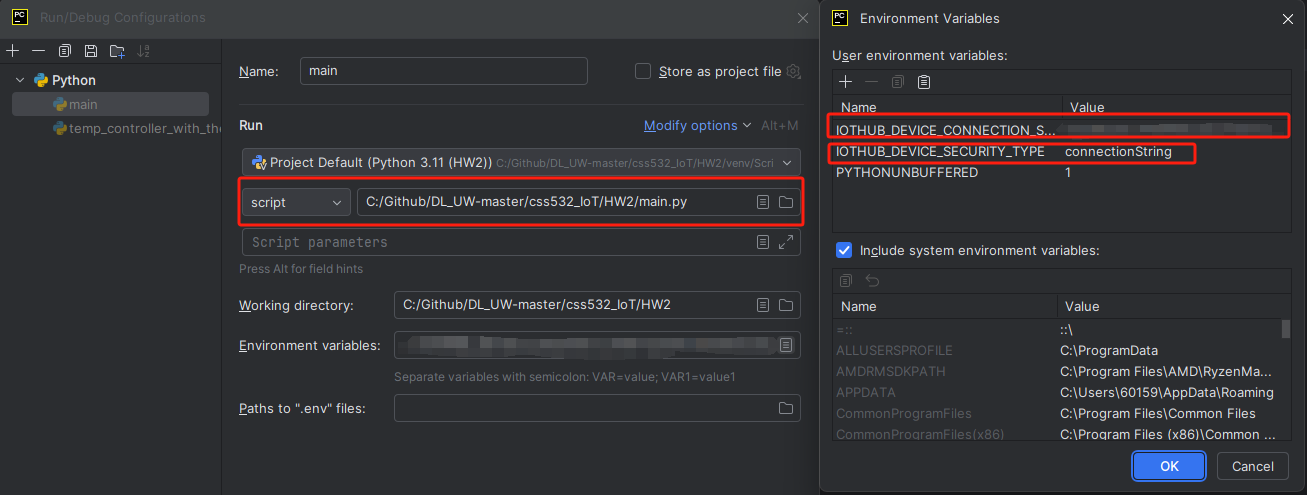
The last thing of making my laptop communicating with the IoT Hub is just connection establishment. I achieved this in my code by Azure IoT SDK.

Fig4. Environment Variables Configuration

device\_client = IoTHubDeviceClient.create\_from\_connection\_string(  
 conn\_str, product\_info=model\_id  
)

#### 1.2.2 Configure Azure IoT Hub to receive messages from your device (laptop) and show the result in Azure IoT Hub console.

#### By coding with Azure IoT SDK, I got a program which could send message to the IoT Hub. Codes could be accessed within the zip file. The principle is using device client to send temperature data. An array of 5 number, each randomly generated.

#### If user want to review the message from the CLI, they should run the following command on Microsoft Cloud Shell: az iot hub monitor-events --output table --device-id {user\_defined\_devicde\_id} --hub-name {user\_defined\_iot\_hub\_name}.

As a result, I could see my message as shown in fig5.

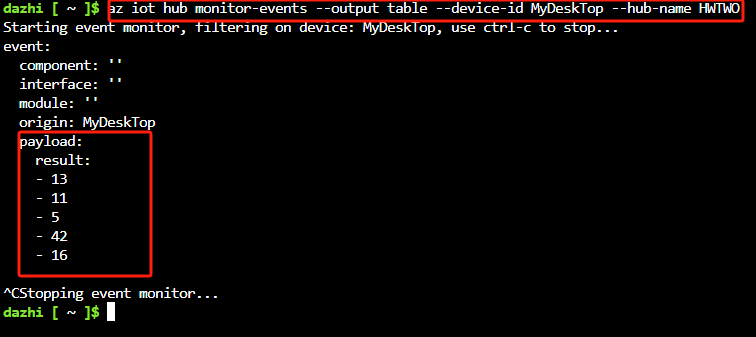


Fig5. CLI message result

If user want to see the message in IoT Explorer, a GUI tool for IoT Hub, user should obtain their IoT Hub connection string first and connect it via IoT Explorer. The connection details are shown in Fig6.

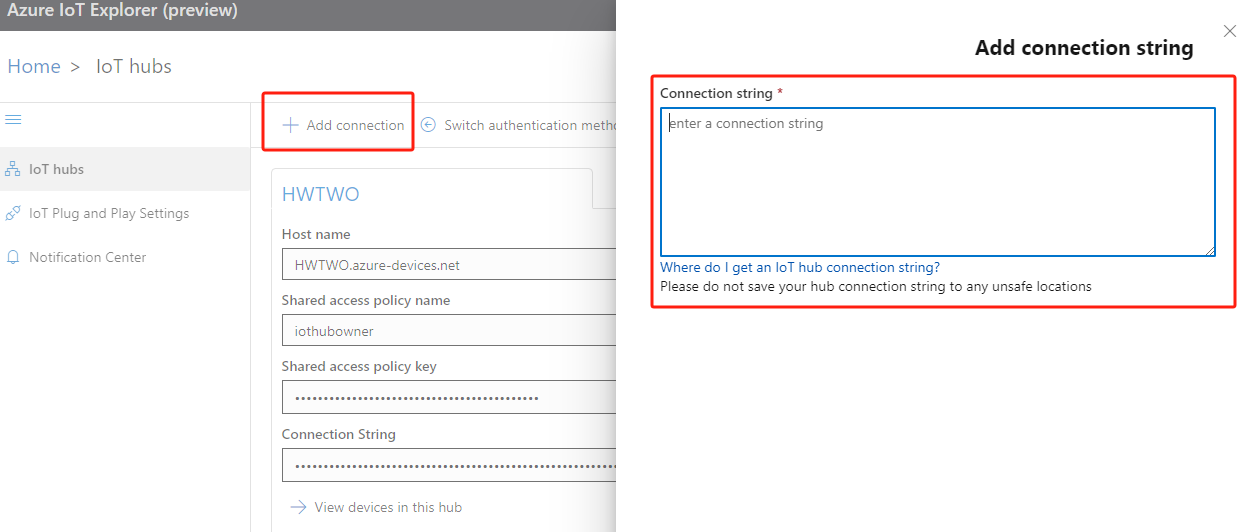


Fig6. Connecting IoT Hub via IoT Explorer

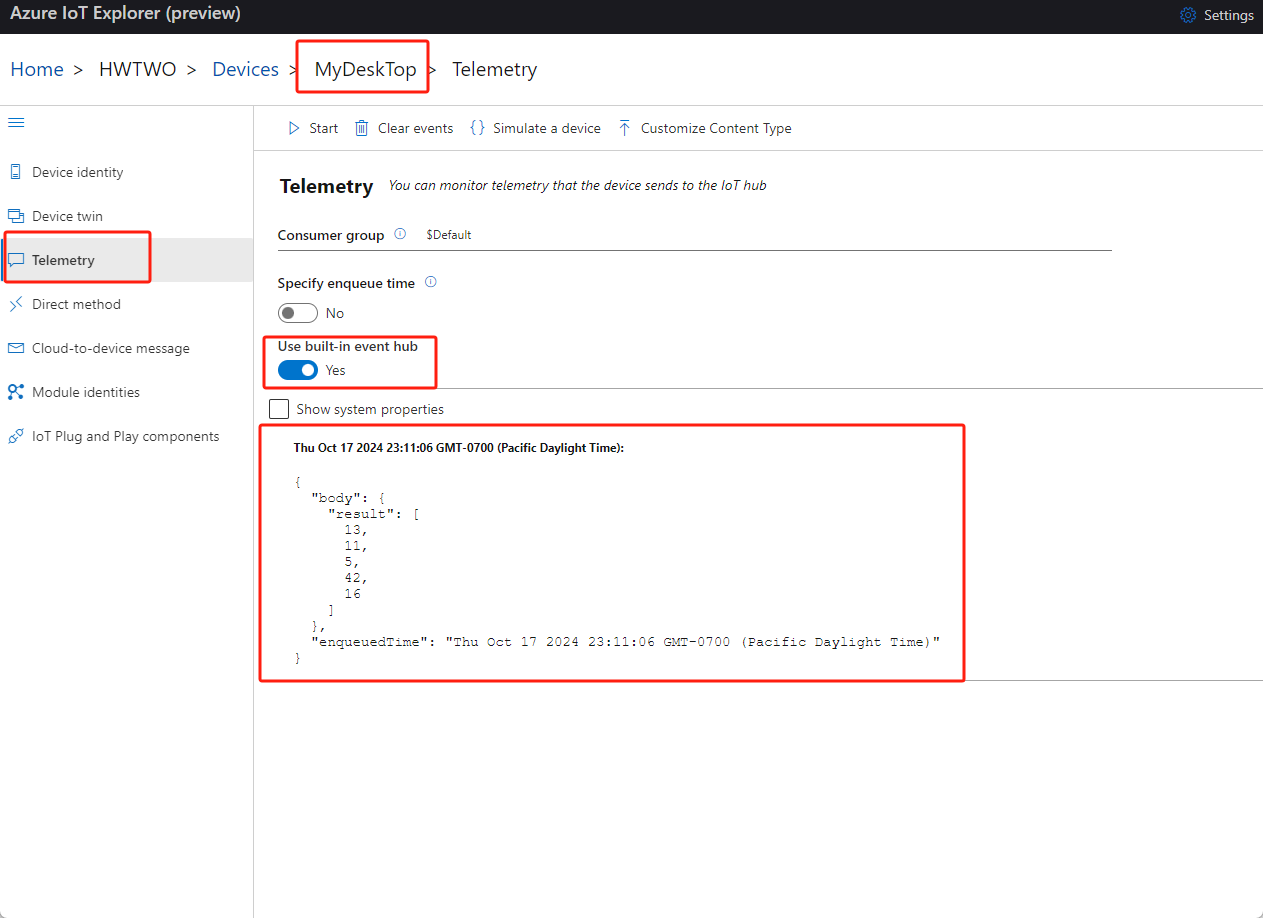
If a connection is established, user could select device/thing which is created before in the IoT Explorer. After getting into the device information, users are suggested to click telemetry on the left column. And once clicking start, the IoT Explorer will monitor event triggered by the specific device. If a message is sent by device, the message will be shown on the IoT Explorer as shown in fig7.

Fig.7 IoT Explorer Message Result

#### 1.2.3 Connect Azure Stream Analytics to your Azure IoT Hub to process received messages from devices and save the raw data (the data shall be extracted from received messages) and the processed data (you should use Stream Analytics to process the data) into Azure Blob.

The key feature here is how to create a stream analytics flow and how to write a proper SQL codes by using the featured windows provided by Azure Strem Analytics.

The very first thing here is creating an Azure Storage Account. This Storage Account provides Blob Storage service. And users are supposed to create a container in an account first. This is very important for generating the output. The successful result should be like fig.8.

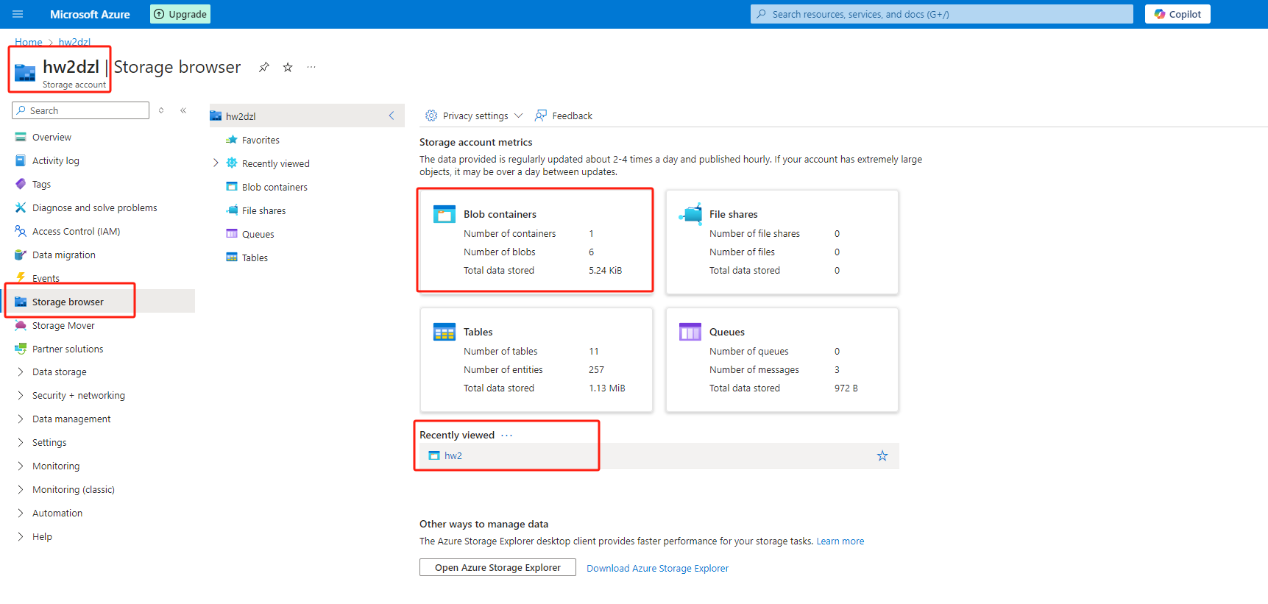
The second thing is creating stream analytics for sure. Its input should be IoT Hub which exactly matches with the one we used before. However, I set two different outputs here. One for saving data to the raw data blob. The other one for storing data to the processed data blob.

Fig8. Azure Storage Account

Then, I wrote SQL codes as shown in Fig9. Codes could also be accessed within the zip file. The first select sentence extract data from the message and store them into a blob file, which is the output named hw2. The second select sentence aggregate the data and calculate the average data and storing them into the other blob file, which is the output named hw2-avg.

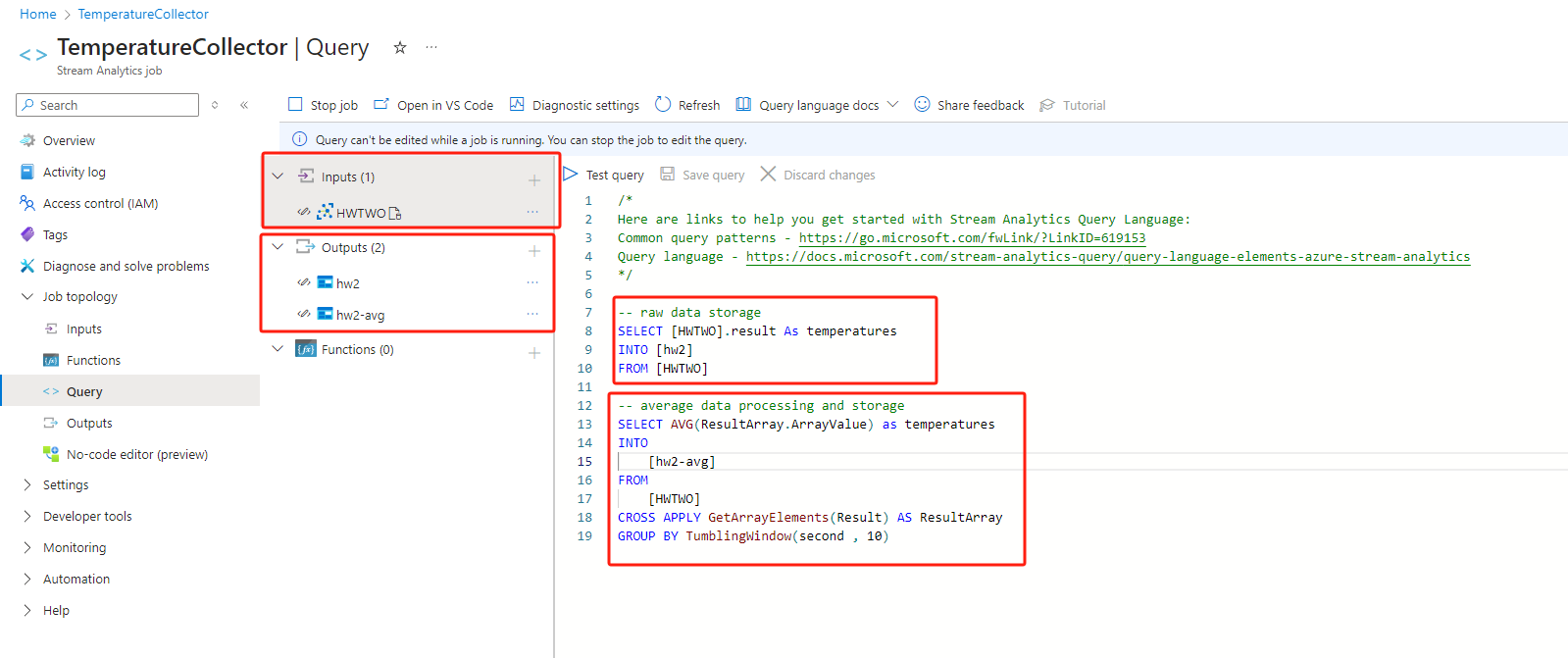


Fig9. Stream Analytics Configuration

I differentiate the output by assigning results to different path, as shown in fig10.

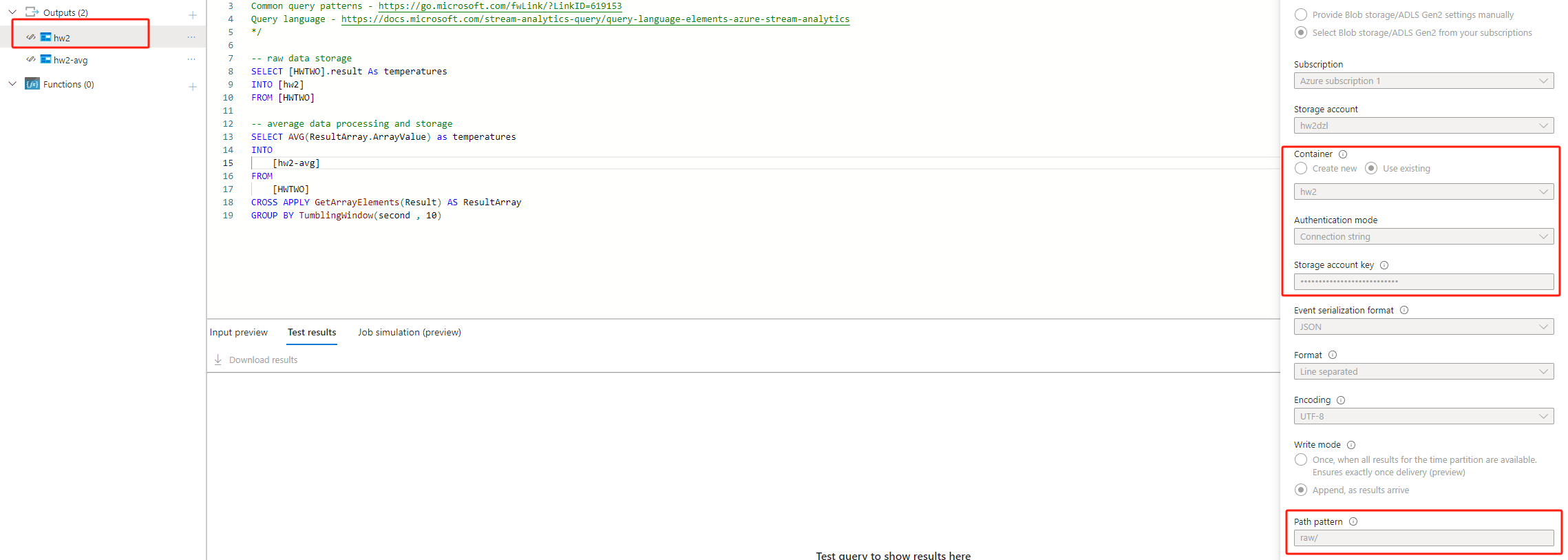


Fig10. Output Configuration

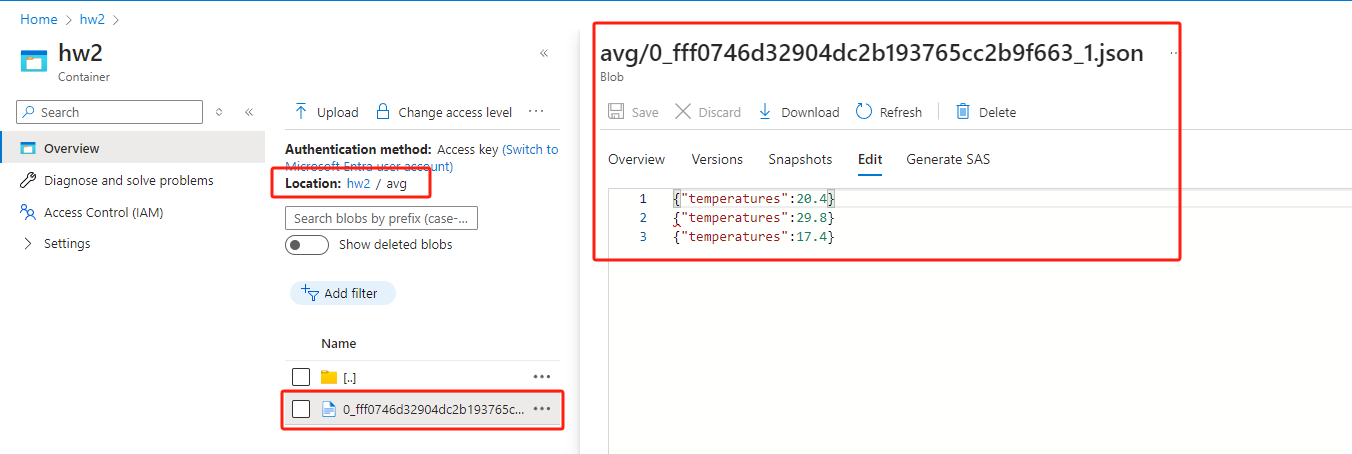
After starting this job, I could send message to IoT Hub again. Under the influence of Stream Analytics, I should see two separate files in my Storage Account. As shown in Fig11. and Fig12. One thing I have to mention that is my output setting is appending new message to the previous line. So, we could see multiple lines of data generated from my computer and stored into the blob.

Fig11. Average Temperature Data in Blob

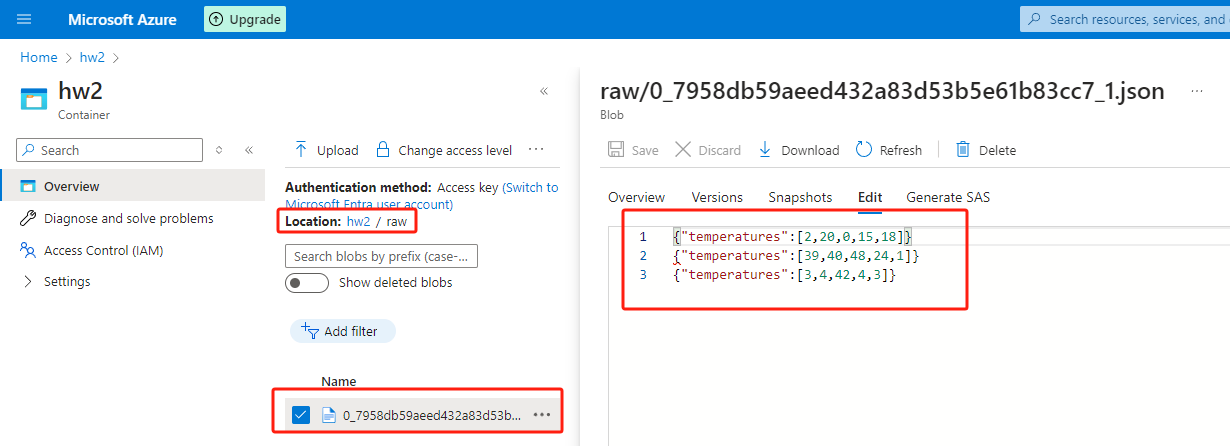


Fig12. Raw Temperature Data in Blob

As a result, we could see two files in the Azure Blob.

#### 1.2.4Connect Azure Functions to respond to Azure Blob events when new "processed data" is added. Your Azure Function may simply print out the content of newly added data, which is the "processed data" obtained in Step 3.

This step is a little bit harder than what we did in previous HW1 AWS lambda function. I think the difficulty comes from the totally different code editing ways.

In Azure, I do not know why I cannot get an online editor and finish those job. I have to download Visual Studio Code and modify my code on my local IDE and deploy them later on Azure Function App.

The first thing here is downloading different kind of prerequisites like azure function extensions and relating libraries for local debugging. The Azure function extension should be like in Fig13.

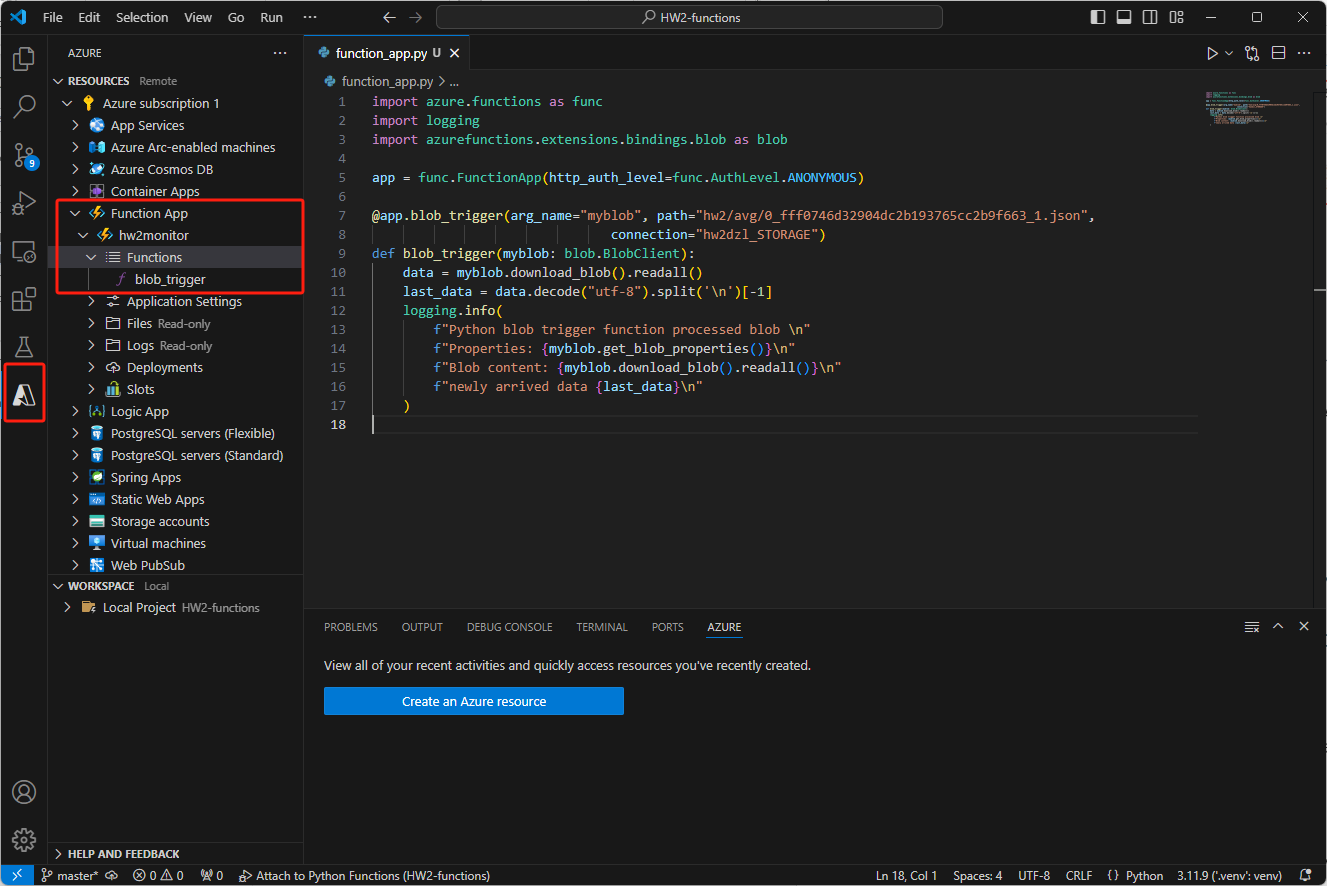


Fig13. Azure Function Extension

#### When those coding are finished and tested locally, I utilize the Azure Function Extension to deploy my program in the cloud, which is responsible for printing the newly arrived data. Also, the source code of this program is accessible within the zip file too. However, as I mentioned that AWS lambda and Azure Function App are quite different. In AWS, they would usually make often-used dependencies like s3 SDK integrated. I.e. user do not have to download or manage such kind of SDK on their own. However, in Azure Function App, there is a very important but easily ignored file named requirement.txt. In our case, if we want to use blob client to read data in our file, we have to add dependency name in it. The requirement.txt file is shown in Fig14.

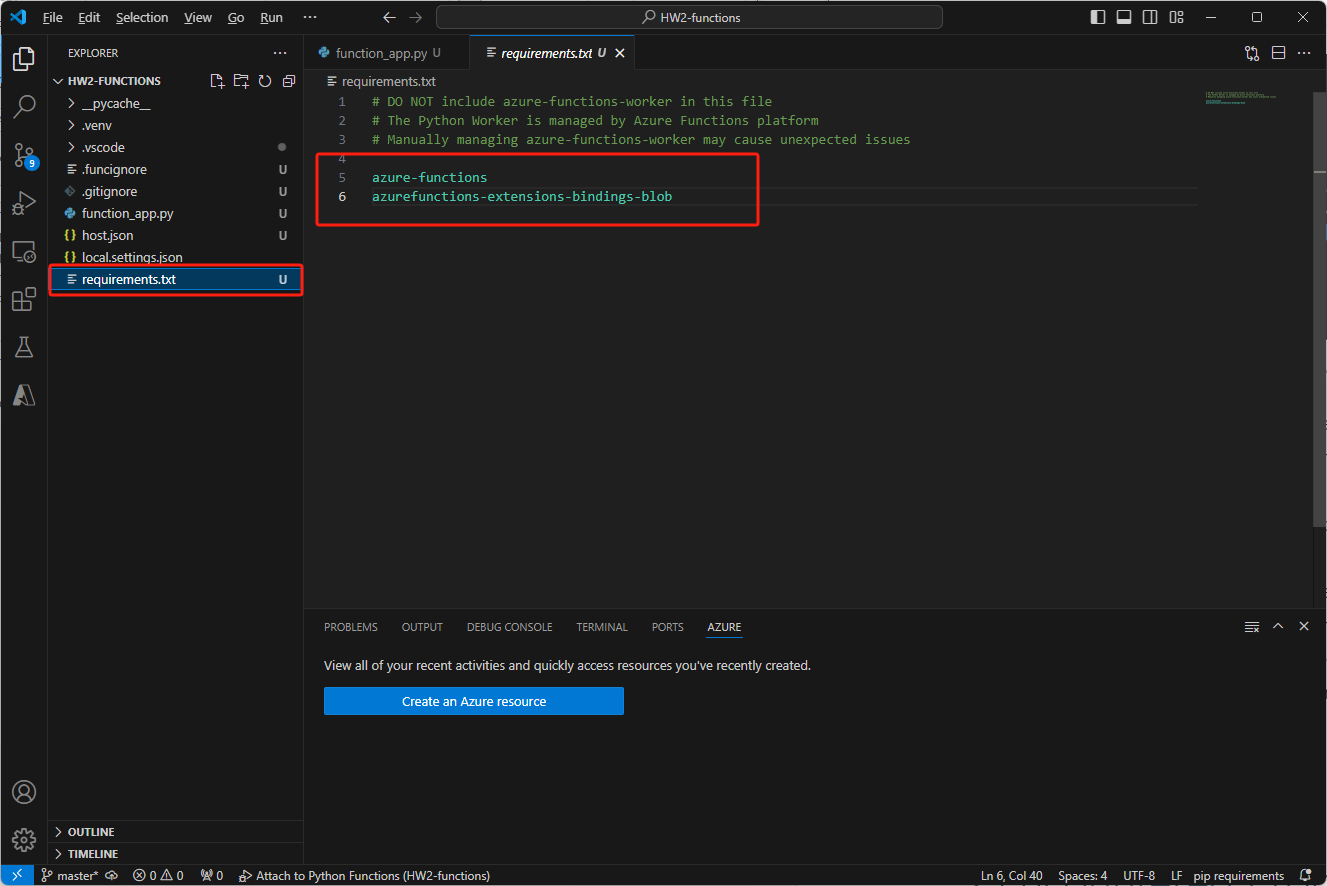


Fig14. Azure Function App Dependency File

If this detail is missed, this program could be deployed on the cloud but not runnable. I will mention that later.

Finally, once the code is ready and I start the Azure Function App, I could send message to IoT Hub via my computer again. This time, I receive log information which prints out the newly arrived data. As I mentioned above, the way that I am adding output to the blob file is appending write. So, there are a whole bunch of data and the latest data in my log printed dedicatedly. The printing result is showed in Fig15.

This part is different from AWS two. In AWS lambda, we just select trigger event by cloud service and we will get an event for further data processing. However, in Azure Function App, I put the trigger into my code in a python decorator.

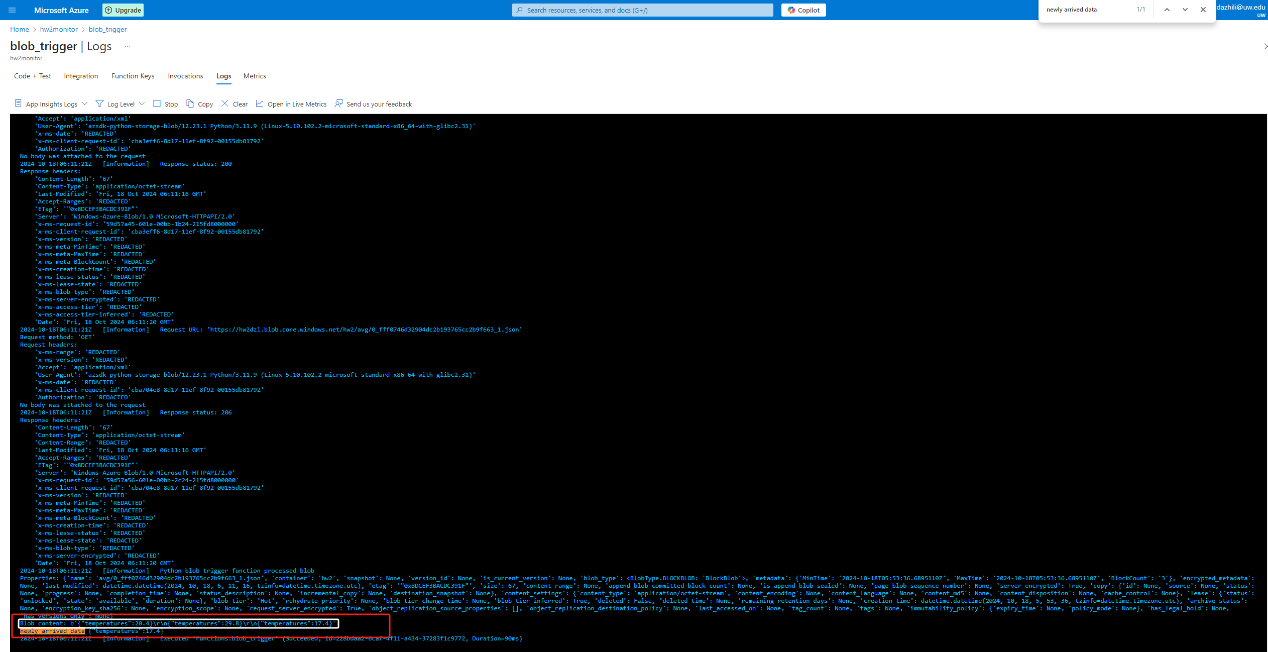


Fig15. Azure Function App Log Printed Result

# 2.Encountered Problems

### 2.1 How to use the IoT SDK and hide my credentials

This part is difficult at finding the device demo provided by Azure. The device demo could be fluently downloaded in AWS by following the starter guide. However, things in Azure IoT are quite different. In Azure, the guideline does not provide any example code on how to use its SDK. I found the demo in a document which is hiding very deep. And it could not be runnable if user just downloads their demo. User are required to set system variables to use their demo. If a user starts with an IDE like PyCharm, which will provide virtual environment for different project, that used will spent a lot of time on searching why the demo could not be executed. This is exactly what I experienced in this homework. I finally overcome it with finding the problems first. I read all the code and find that connection\_method is not specified, which I did specify in my computer environment variables. After thinking for a while, an idea came to my mind that I could add explicit environment variables at the run/debug configuration so that my program could 100% read that variable. Fortunately, this method works and I found I could hide my credentials like connection string into my system variables.

### 2.2 SQL grammar is different in Stream Analysis

Extracting raw data from the IoT-Hub and storing it into Azure Blob is quite simple. However, when I tried to calculate the average data from a table of temperature data, which seems quite simple too, I failed all the times. No matter how I adjust the AVG() function provided by Stream Analytics, the online code editor always reported syntax error. This error is quite confusing that they did not tell exact where the problem is. I took a lot of time searching and finding documents on Azure. Finally I found that, AVG() function in Stream Analytics is just totally a different thing from what I learnt in database. Every group by clause should be followed by a window or timestamp. On the contrary, element is optional. The answer could be shown in fig16.

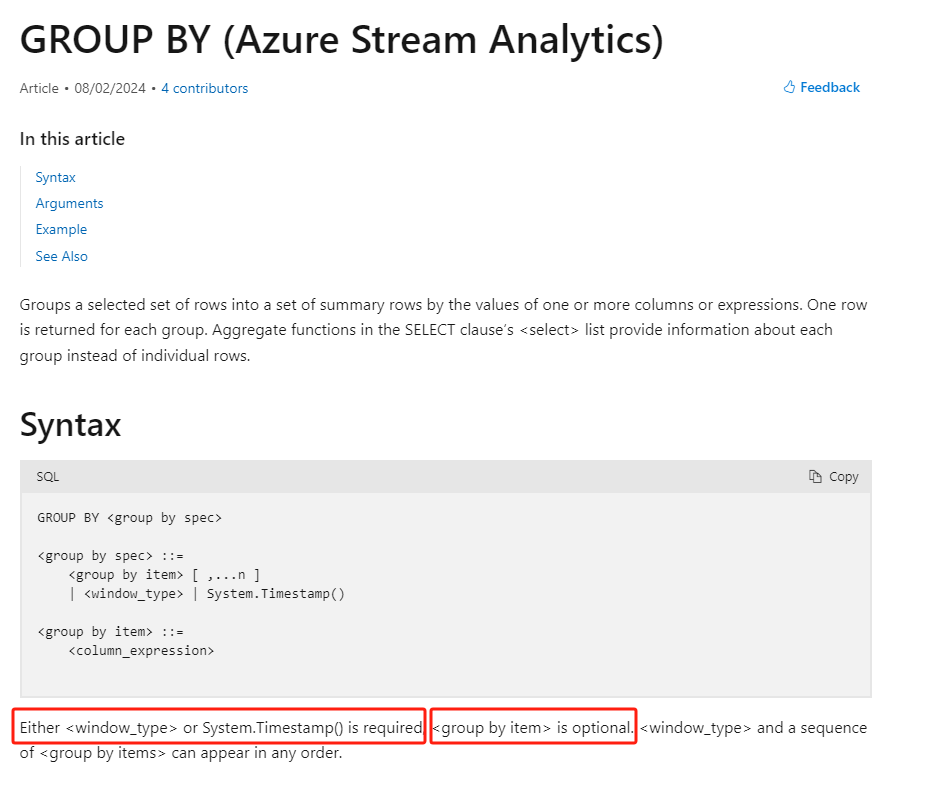


Fig16. Azure Stream Analytics Group By Clause Usage Rules

### 2.3 Azure Function App deployment

The worst thing of serverless services is that they sometimes do not give enough debugging details. This time, I encountered a huge problem in Azure Function App. I attempted to run all the codes locally, and everything works totally correct. But every time I tried to deploy my code on the cloud service, it showed successful deployed. However, my program is not visible on the webpage and not working at all. I spent a lot of time finding why my program is missing. I tried to stop the function app. And then, I could only see a sentence of error which is also confusing saying that internal error. By searching that little information on the website, I gained nothing. Then I tried different codes which is only printing data incoming message not reading data from blob files. Unexpectedly, it worked. Then I tried to compare those two codes and did a bunch of testes. At the time when I did not import blob client, a dependency, the whole program worked correctly on the website. This clue gave me enough information on troubleshooting. I just kept going on Stream Analytics dependencies questions. Finally, I found the solution at requirements.txt. The documentation on Stream Analytics which is showing how to set a blob trigger did not mention it at all. So, when doing a project, try different ways to figure out what is missing.

### 2.4 Azure Function App local setting

When I tried to debug locally, errors just showed up every time. The error said I was missing the connection string, which is exactly lying in my local.setting.json file. I spent a lot of time on finding the problem. Finally, I saw a blog which is discussing the same problem. All the problem came from blob client, which is not reading connecting string except variable “AzureWebJobsStorage”. The only thing I did was copying my connection string and pasting it to the value. Then locally debugging worked greatly. Although it did cost me a lot of time on finding the solution, it is worth to save a lot of time when the program does not go into my expectations.

# 3 Time Spent

I estimated this homework will cost me like 8 hours including report writing. However, I spent totally about 12 hours on this homework due to unfamiliar with Azure cloud services. That’s what happens in doing a project that unexpected situations take most of time.