**CST8921 – Cloud Industry Trends**

**Lab 7 – Serverless Computing**

## Introduction

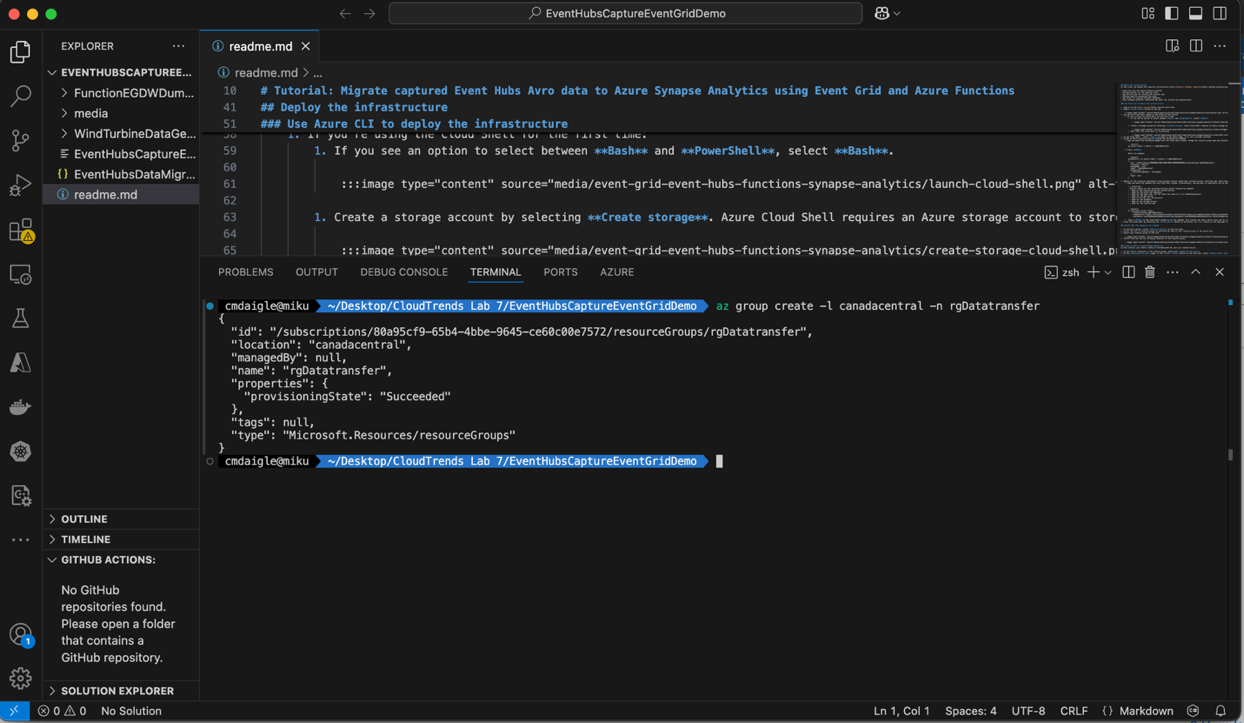
In this lab, I learned how to create, build and manage a serverless framework using Azure and VS Code. The goal of this lab is to take the sample project from [EventHubsCaptureEventGridDemo sample project](https://github.com/Azure/azure-event-hubs/tree/master/samples/e2e/EventHubsCaptureEventGridDemo) and use VS Code, as well as Azure CLI to deploy the required resources. From there, use a command to publish a local function, FunctionEGWDumper from VS code to Azure function app services. Data is broadcasted from a VS Code application to Azure Event hubs and is stored within a storage account. The recently published function triggers and migrates the data to a Dedicated SQL pool.

## Objective

The lab activity goal is to familiarize myself with Azure functions and serverless computing.

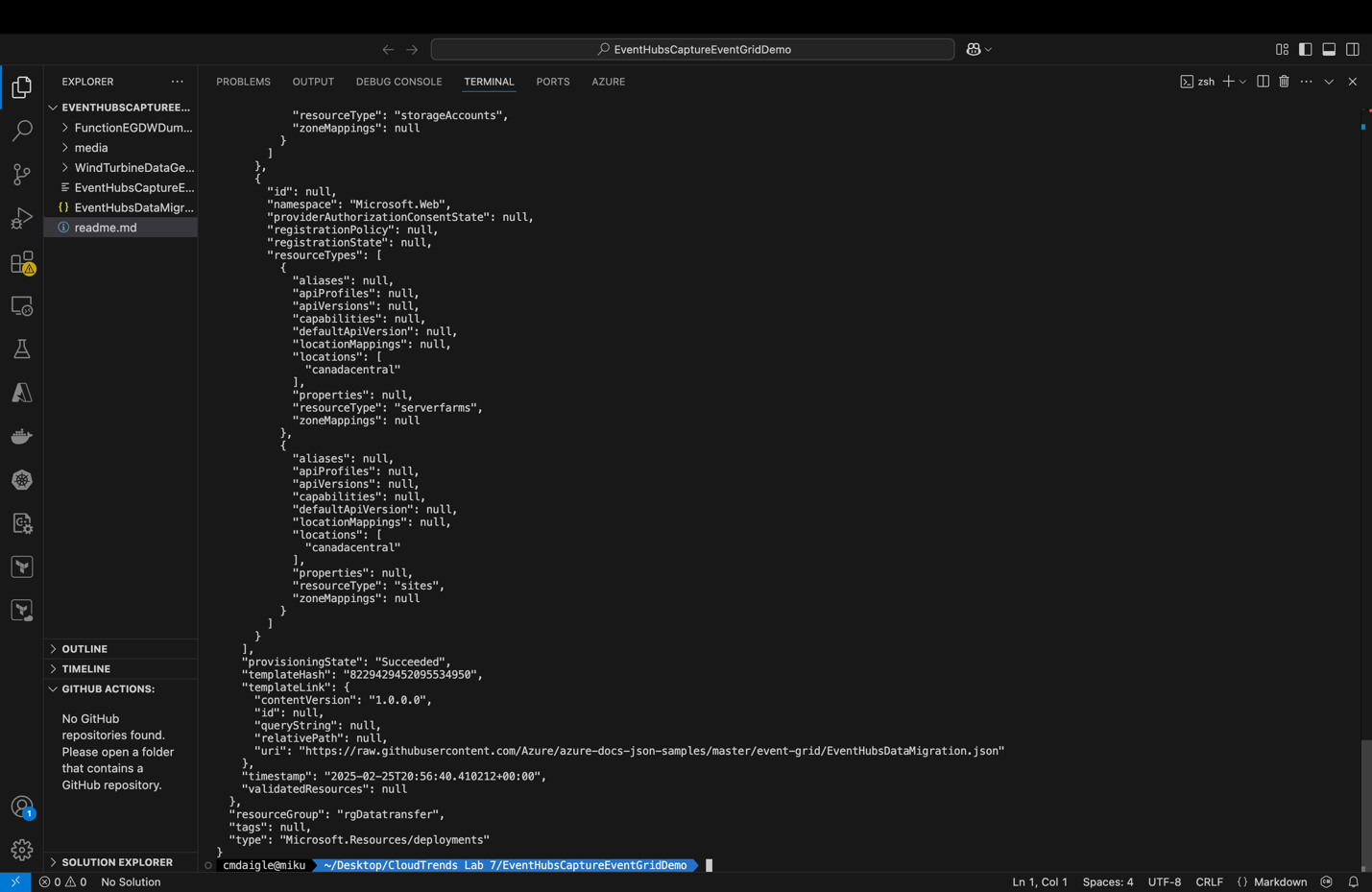
# Screenshots:

1. Deploy the infrastructure with the template- <https://github.com/Azure/azure-docs-json-samples/blob/master/event-grid/EventHubsDataMigration.json>

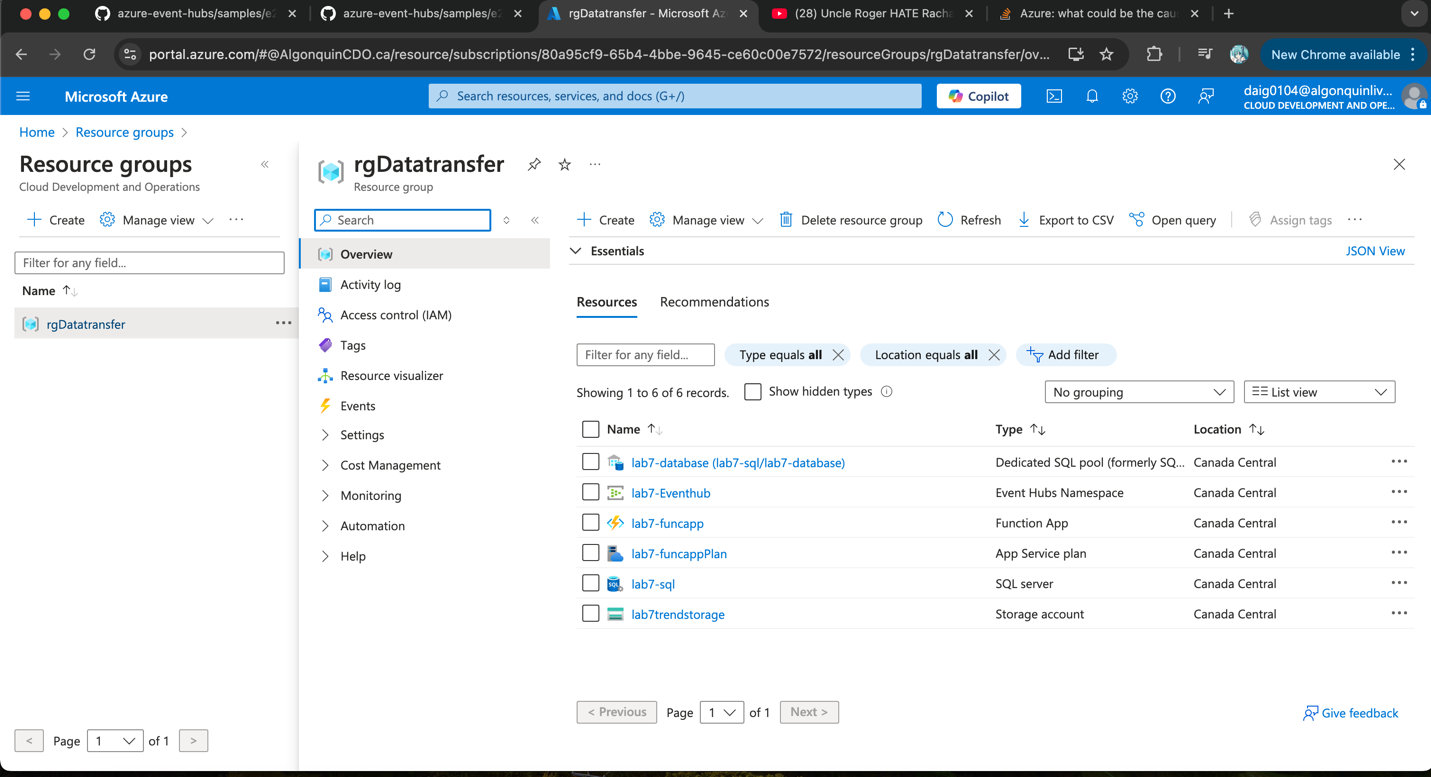
Resource group Creation: (I changed it to Canada Central because it gave me an error that I “exceeded a quota” for the US region.  
Deploying the Resources: (The error below is just because I didn’t make the password for the SQL database long and it needed a special character)



Successful deployment:



Resources made within Azure:



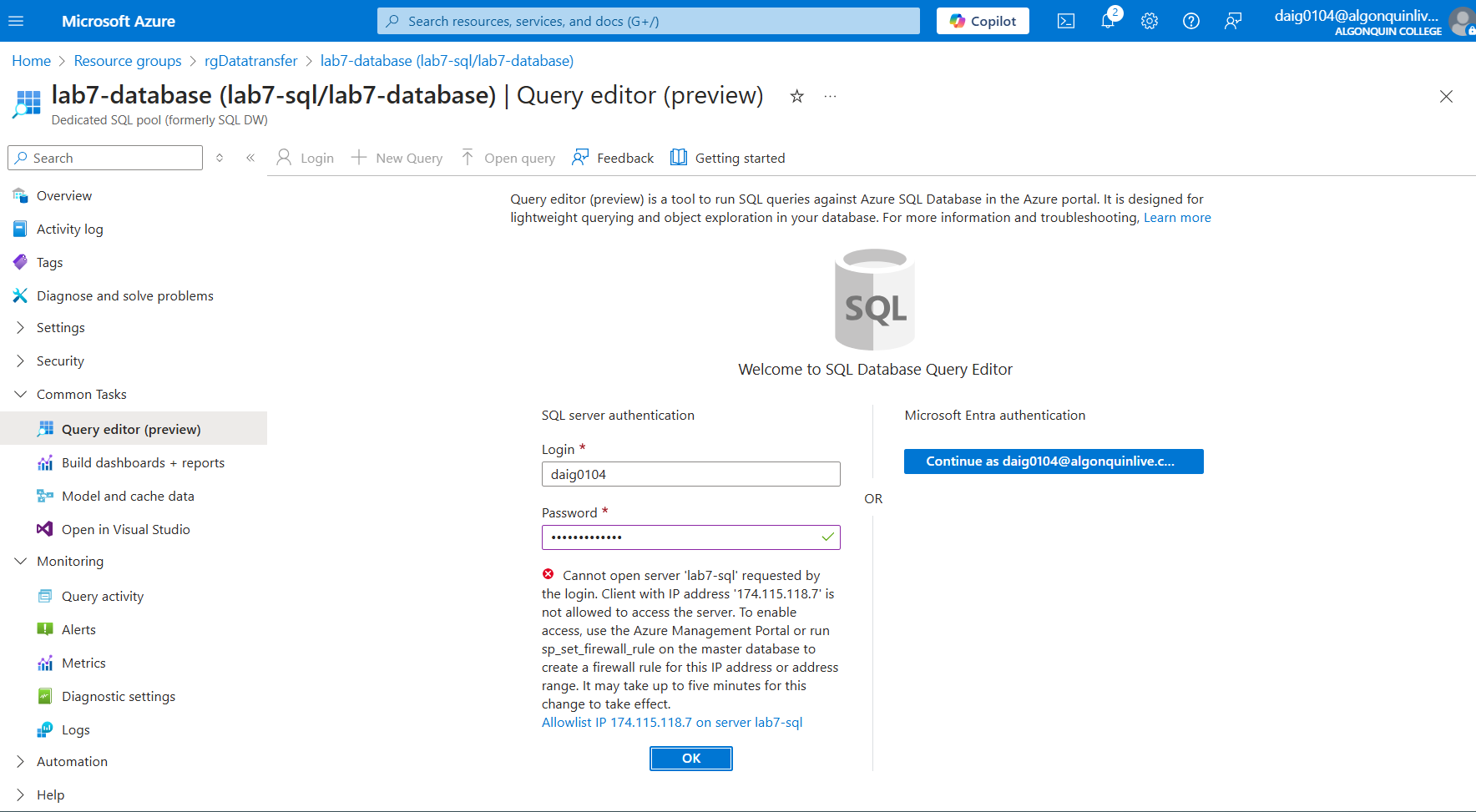
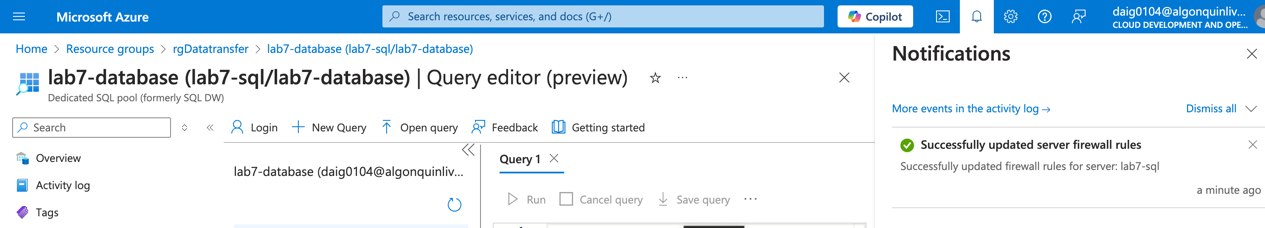
Here is the edited console code I used it only works in bash (later I had to switch to Windows and use a bash console)

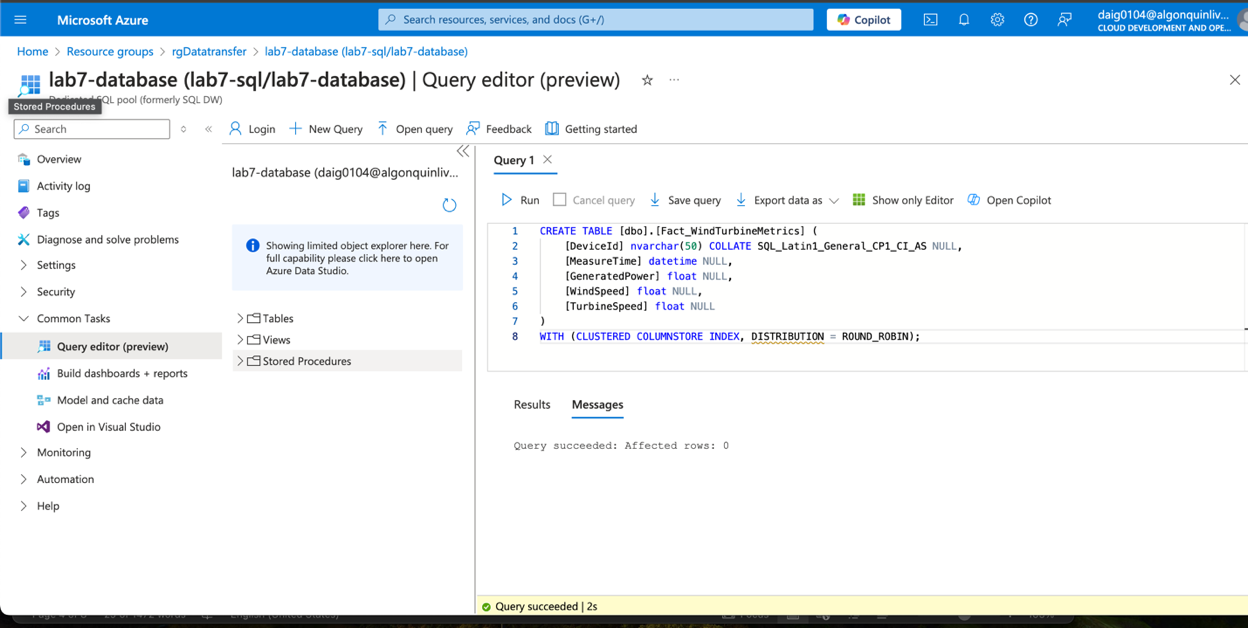
**az deployment group create \**

**--resource-group rgDatatransfer \**

**--template-uri** [**https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/event-grid/EventHubsDataMigration.json**](https://raw.githubusercontent.com/Azure/azure-docs-json-samples/master/event-grid/EventHubsDataMigration.json) **\**

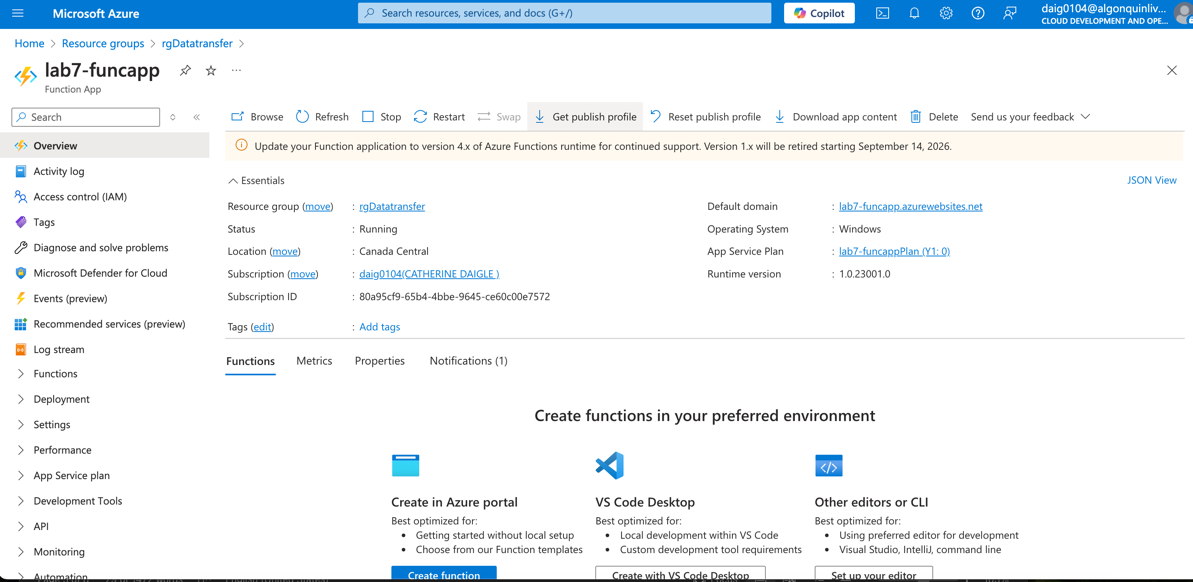
**--parameters eventHubNamespaceName=lab7-Eventhub eventHubName=hubdatamigration sqlServerName=lab7-sql sqlServerUserName=daig0104 sqlServerPassword=mtgl0ve4ev3r! sqlServerDatabaseName=lab7-database storageName=lab7trendstorage functionAppName=lab7-funcapp**

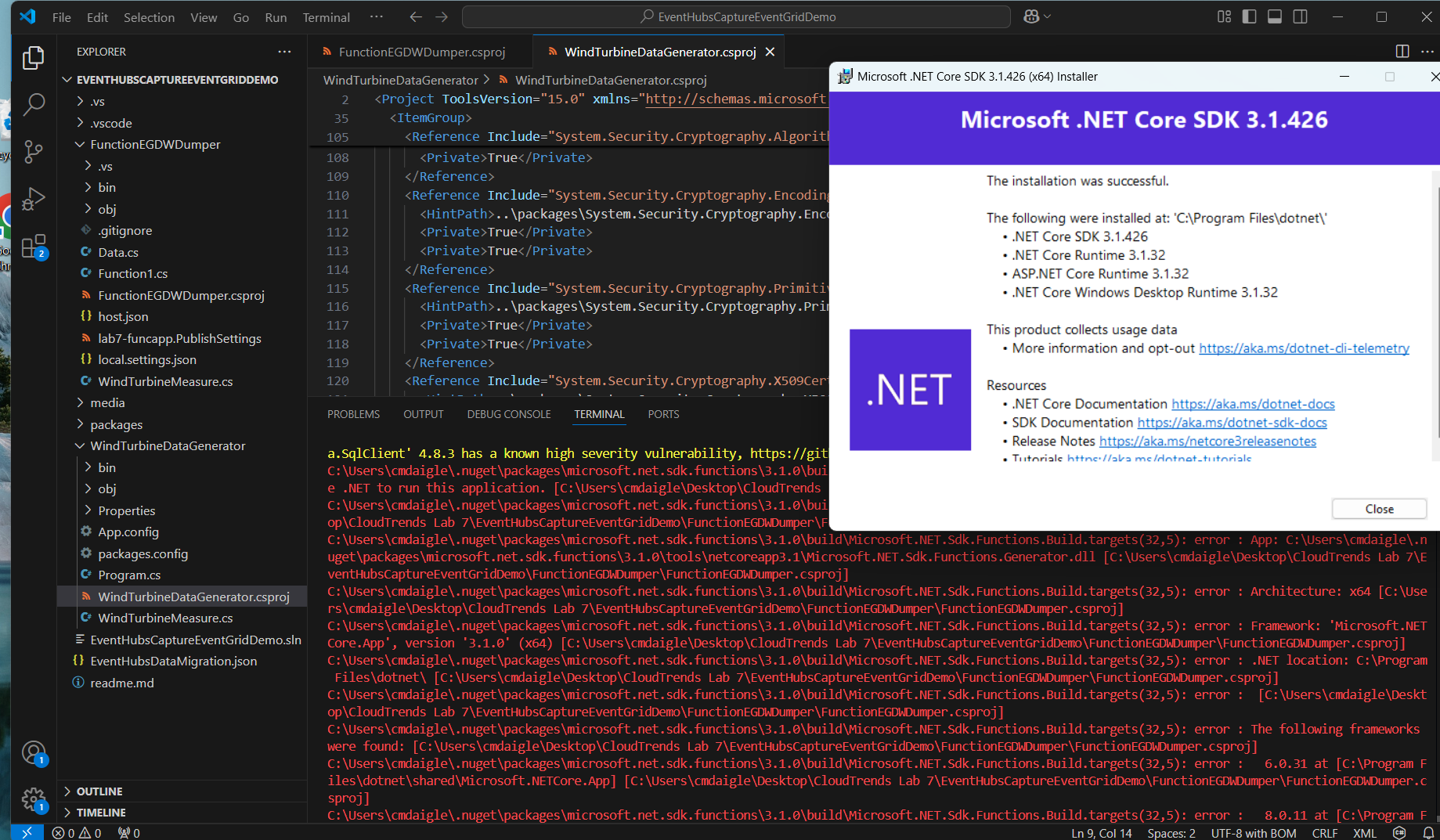
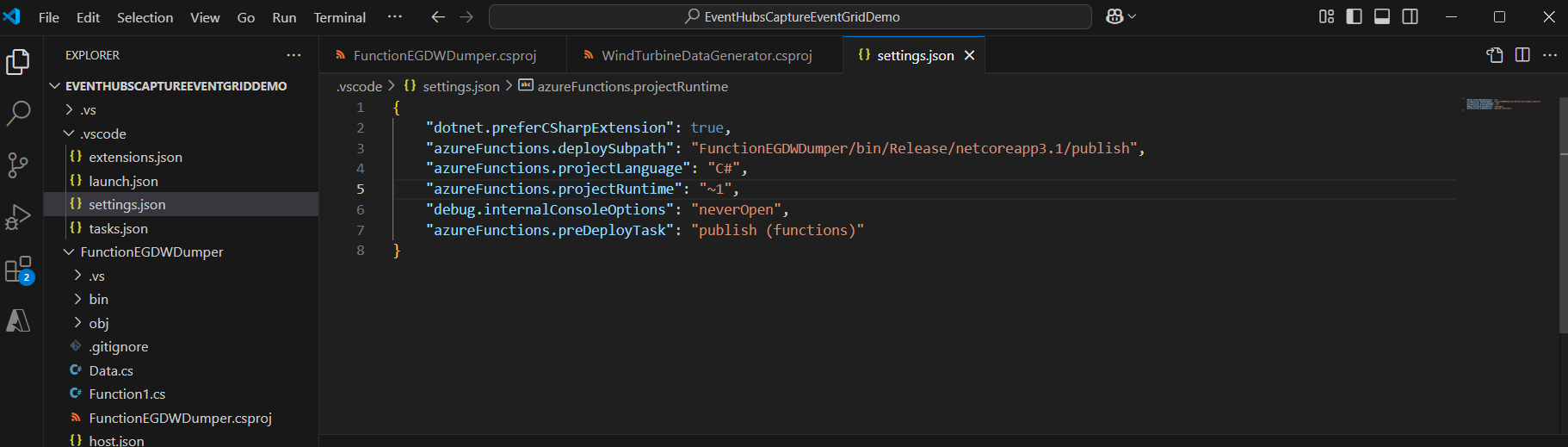
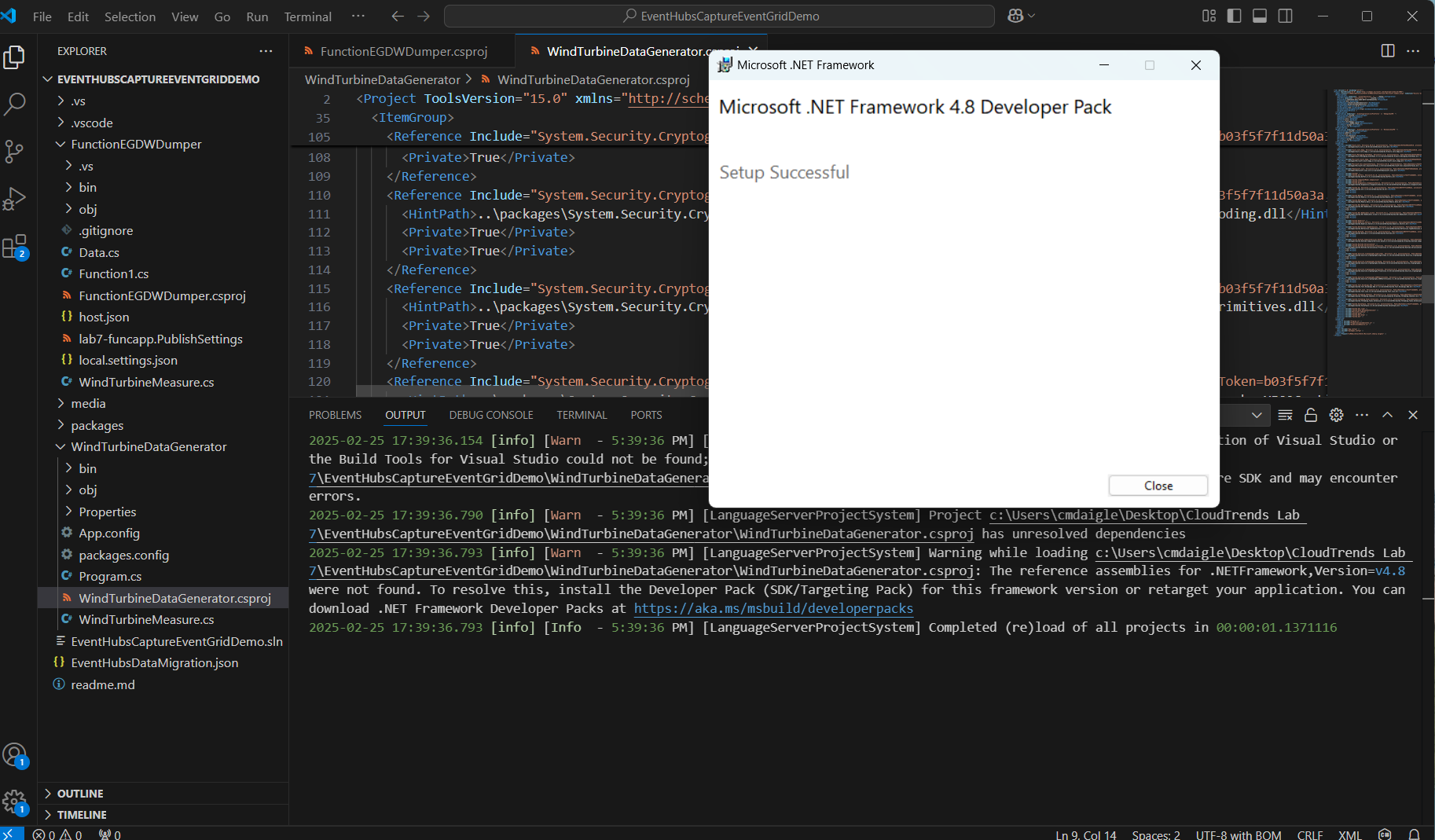
1. Within the dedicated SQL pool, select query editor (preview). Enter the name/password above. For the firewall, click the link allow IP and it adds the required firewall.  
     
   
2. Creating the table:



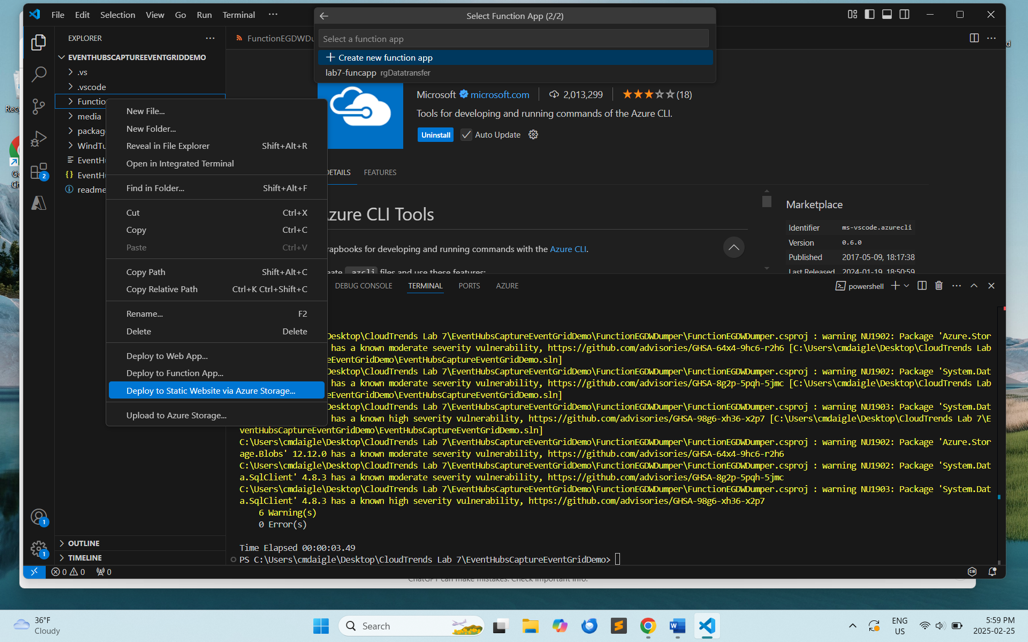
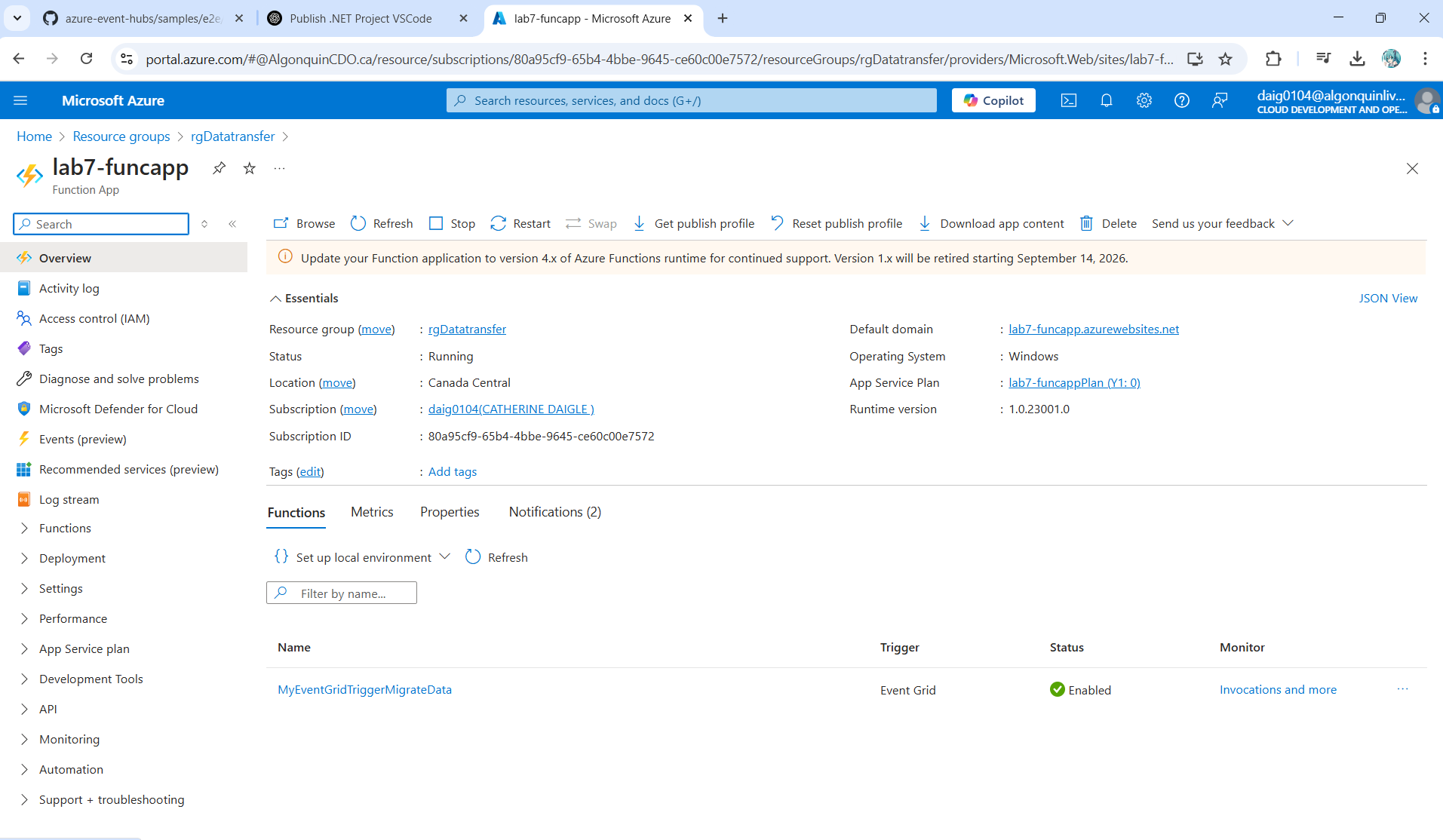
## Publish the Azure Functions app.

1. Publishing the functions app starts to get complex first: I get the publish profile off the function app that’s at the top bar and save it to the function folder:

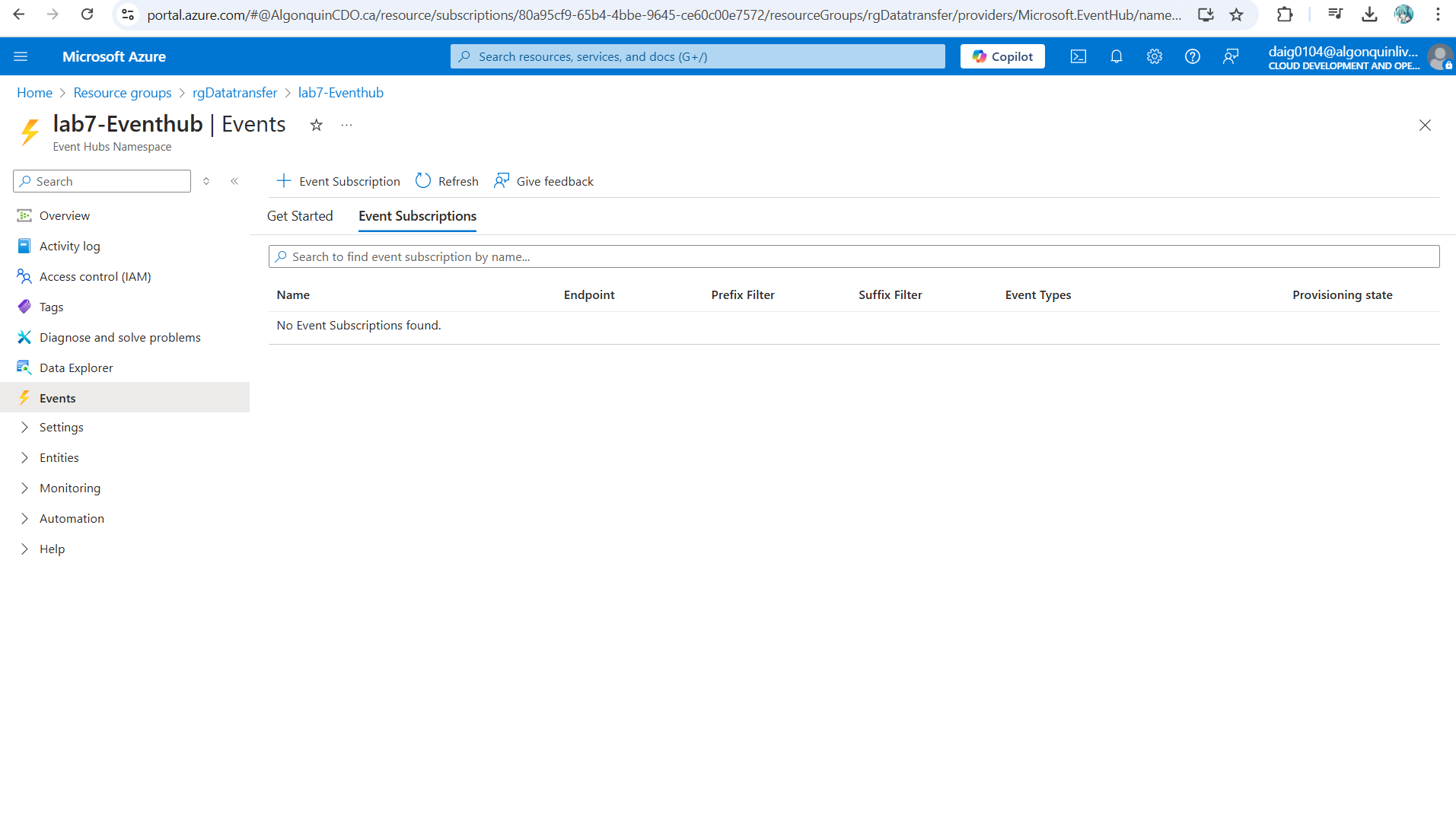
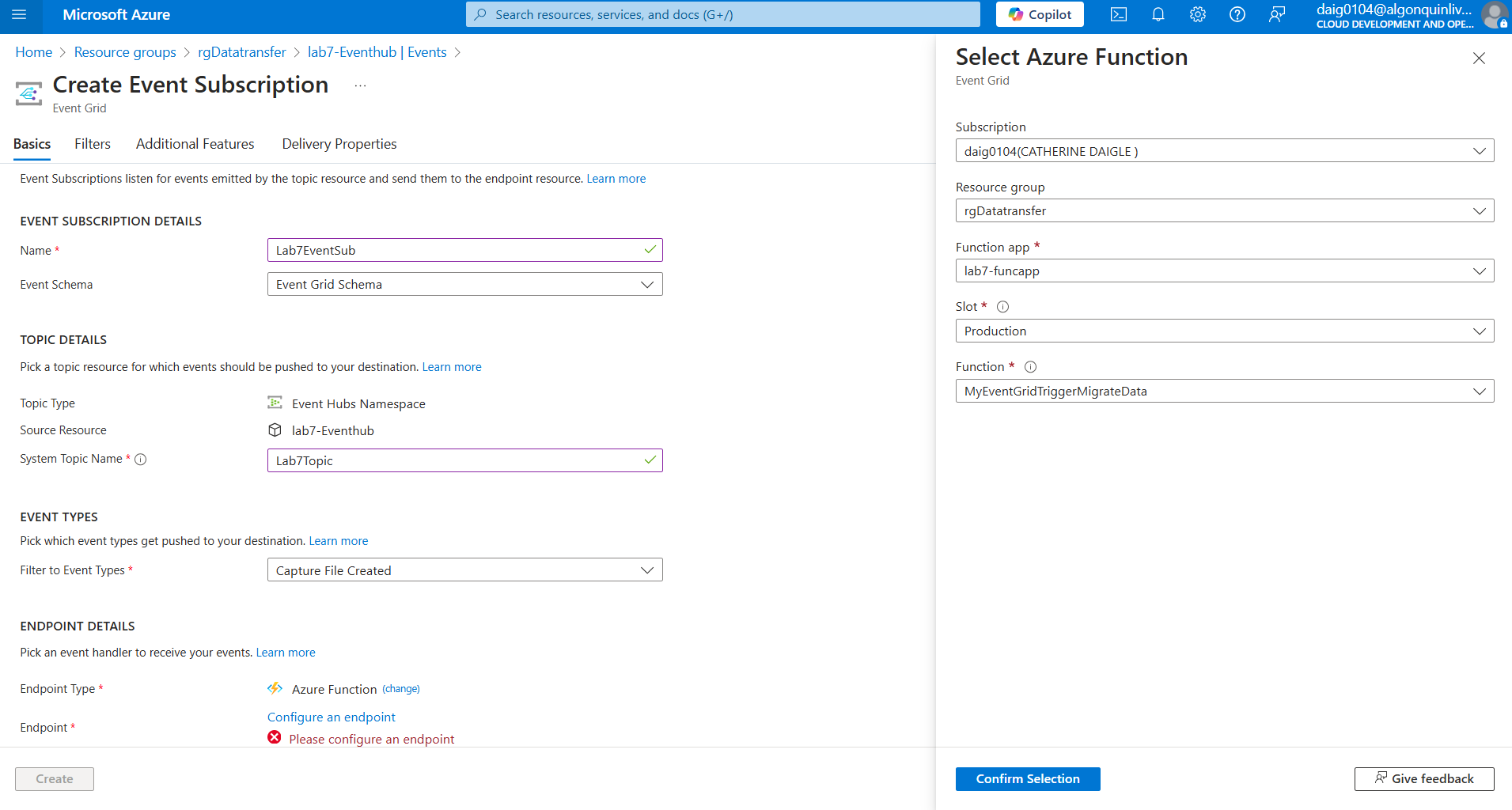
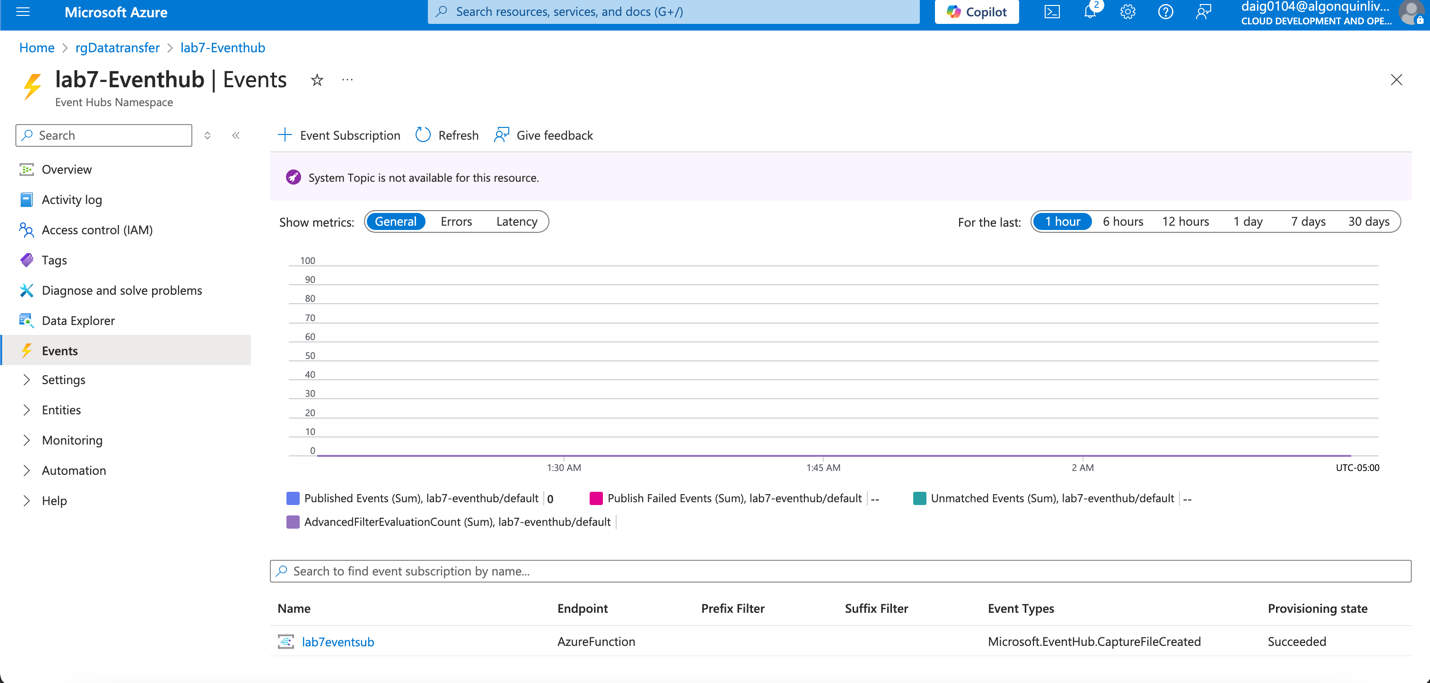
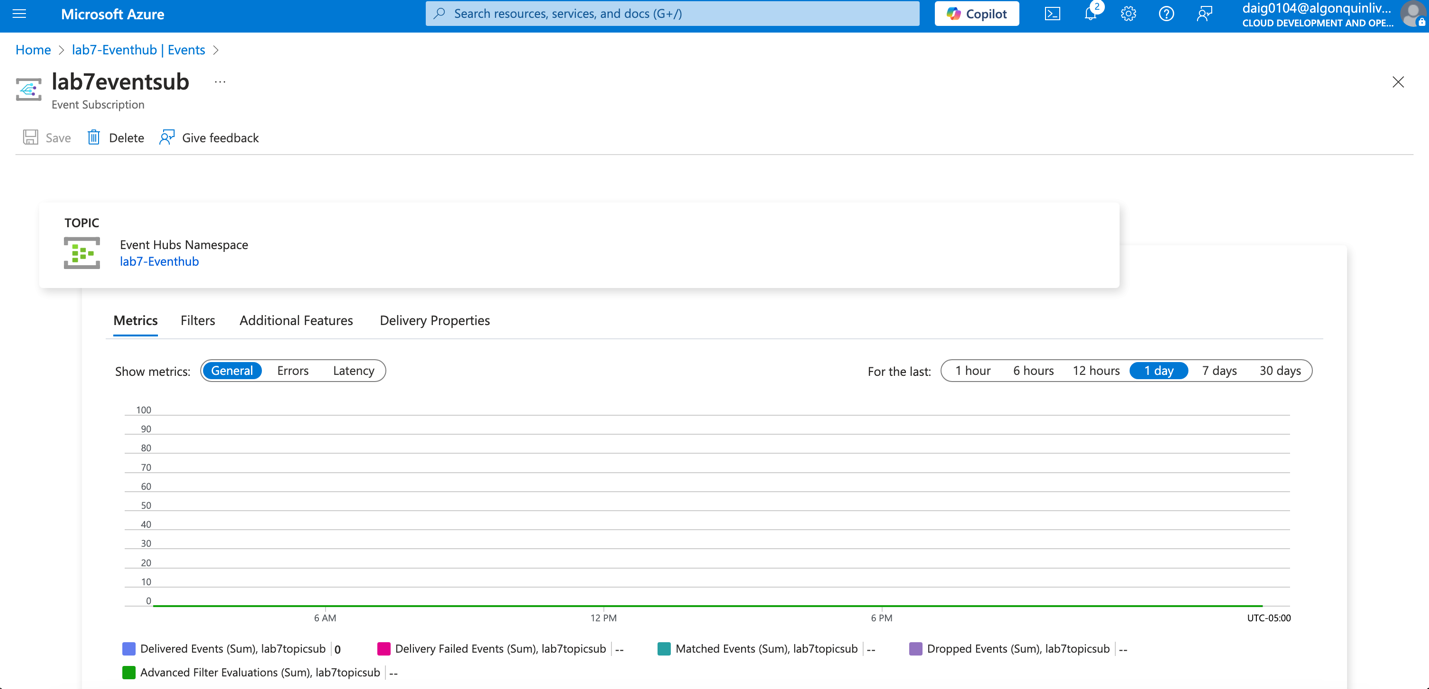


1. Put the file, lab7-funcapp.PublishSettings into FunctionEGDWDumper. Then to build the function I used dotnet build. Unfortunately, this is the tricky part, and it was trying to find the correct dotnet version as the code was extremely depreciated and older versions.  
   
2. Another fix Before publishing, I had to switch Azurefunctionproject Runtime from ~4 to ~1 or the function would not appear within the Azure portal:  
   
3. As shown above, I had to install .NET 3.1 SDK and the developer pack for 4.8, unfortunately, this made me switch to Windows as these versions do not support Mac due to their age.  
   
4. I used the C# output to debug what .Net version I needed. The application could not be updated to use more recent versions of .Net since the code requires the developer version of .Net and would error out otherwise. (that’s when I tried changing <TargetFramework>

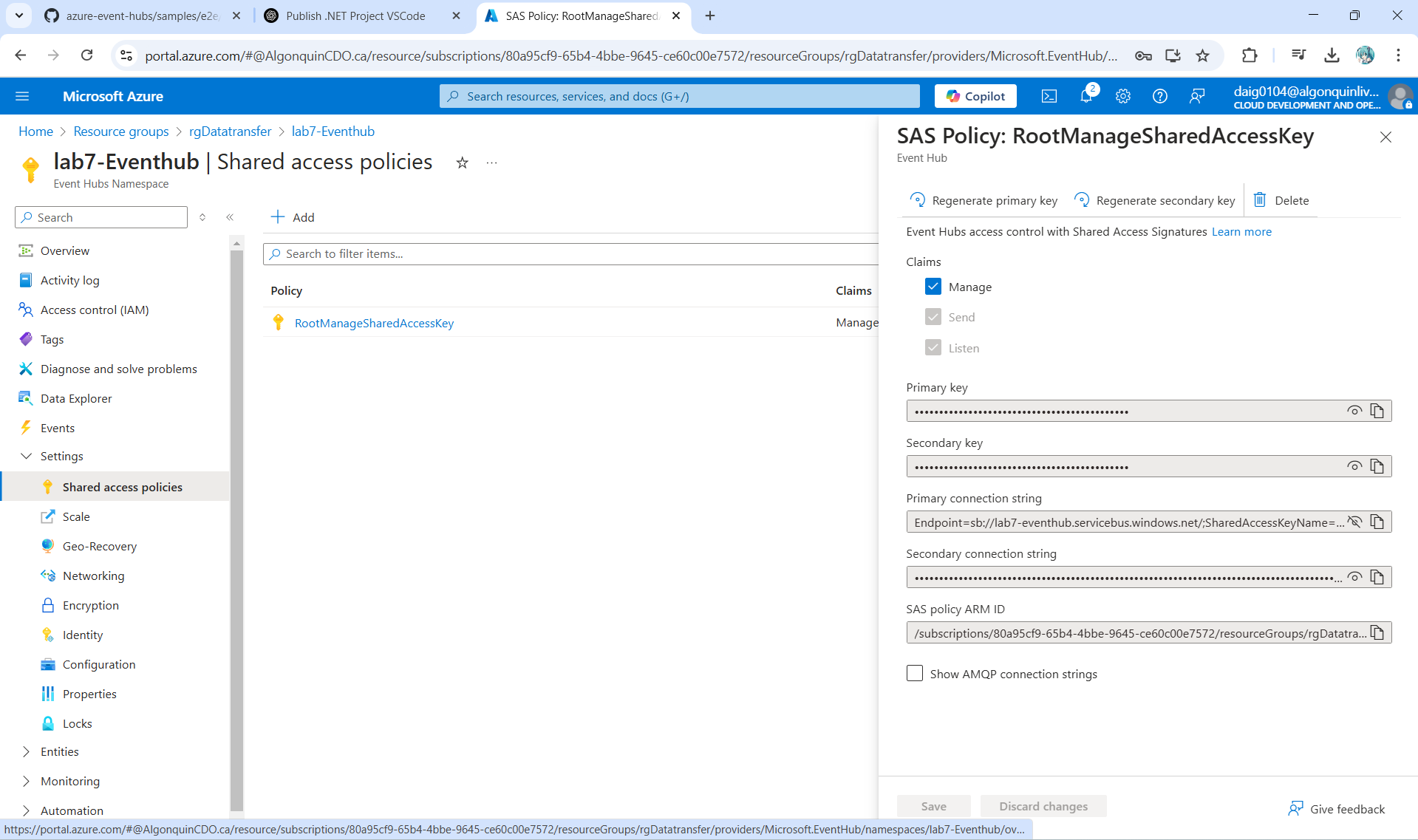
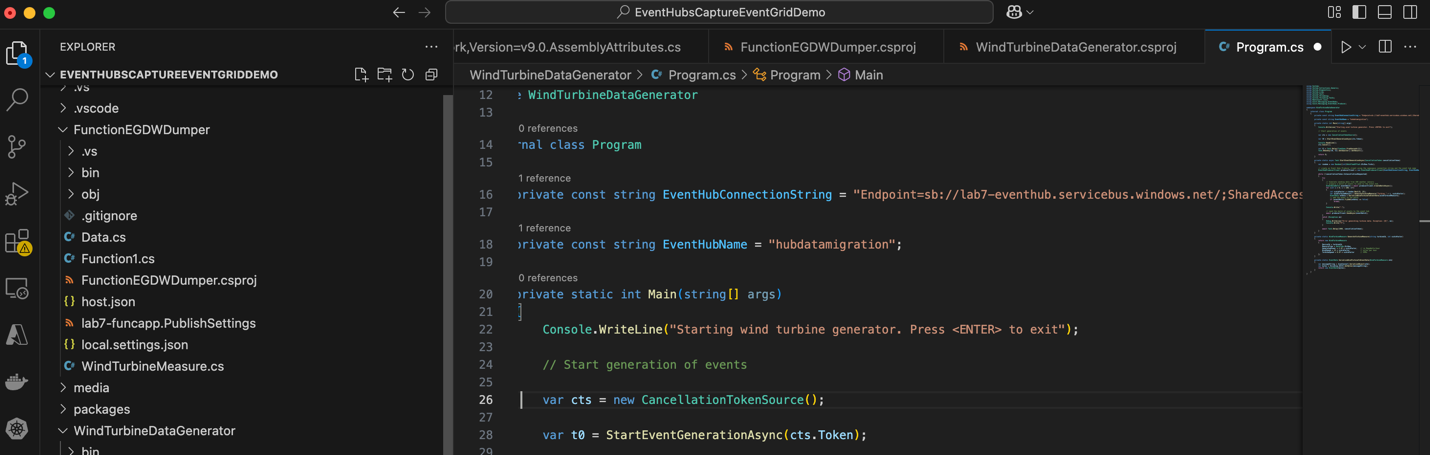
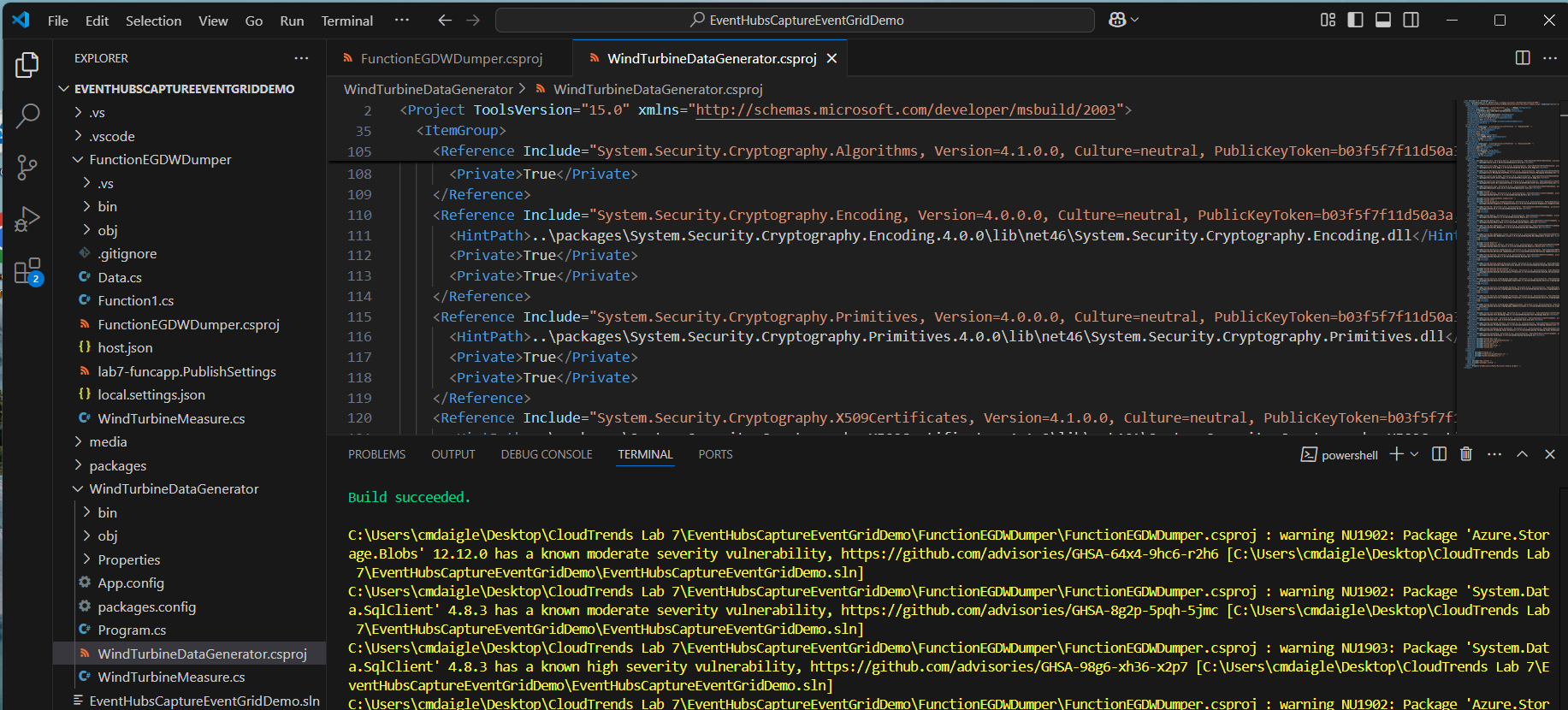
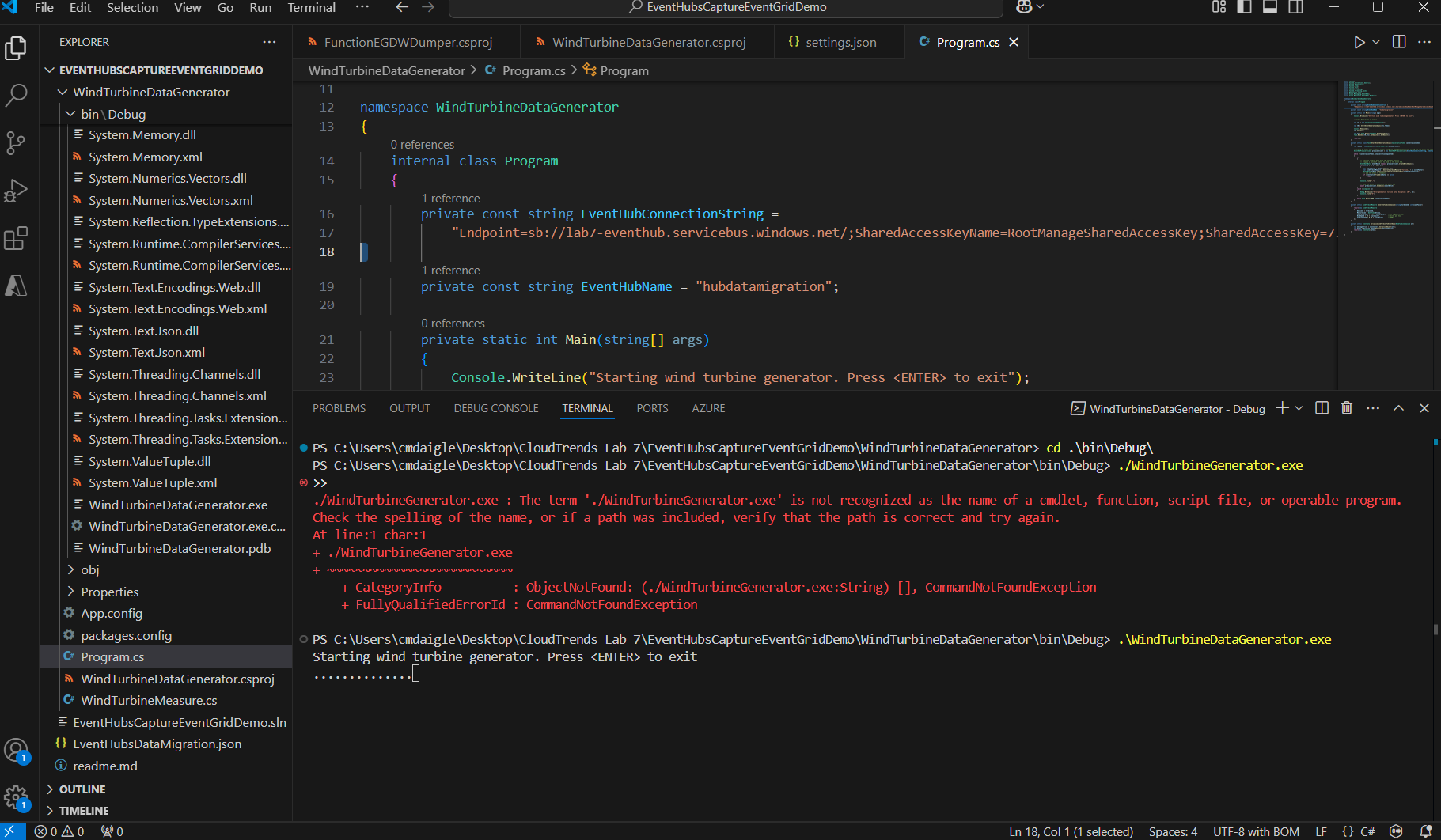
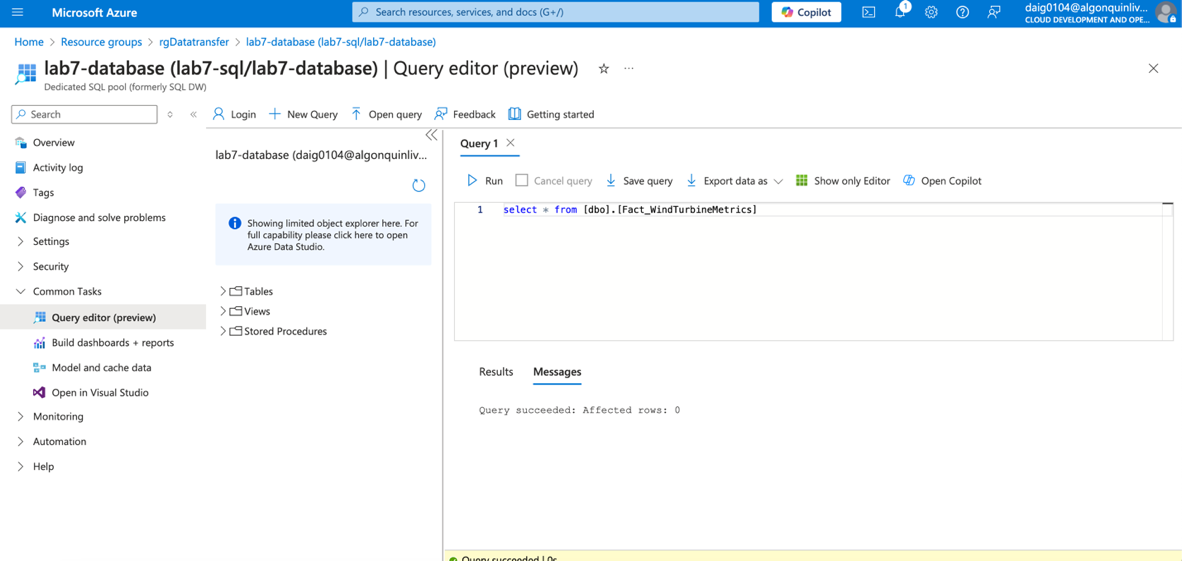
### **Use the publish profile to publish the Functions app**

1. The method of deployment is also extremely depreciated and has to be adjusted. Deployment required Visual Studio which is not supported and therefore, had to apply to Visual Studio code to do this, I had to use the Azure CLI or the Azure functions extension in VS Code:  
     
   
2. (I later used Azure CLI as well as overnight some features were updated and my CDO subscription was not able to be accessed in the above way This was an attempt to fix the bigger function problem later on):  
   
3. Once Published, I check it within the Azure app functions tab You can see it as MyEventGridTriggerMigrateData:  
   

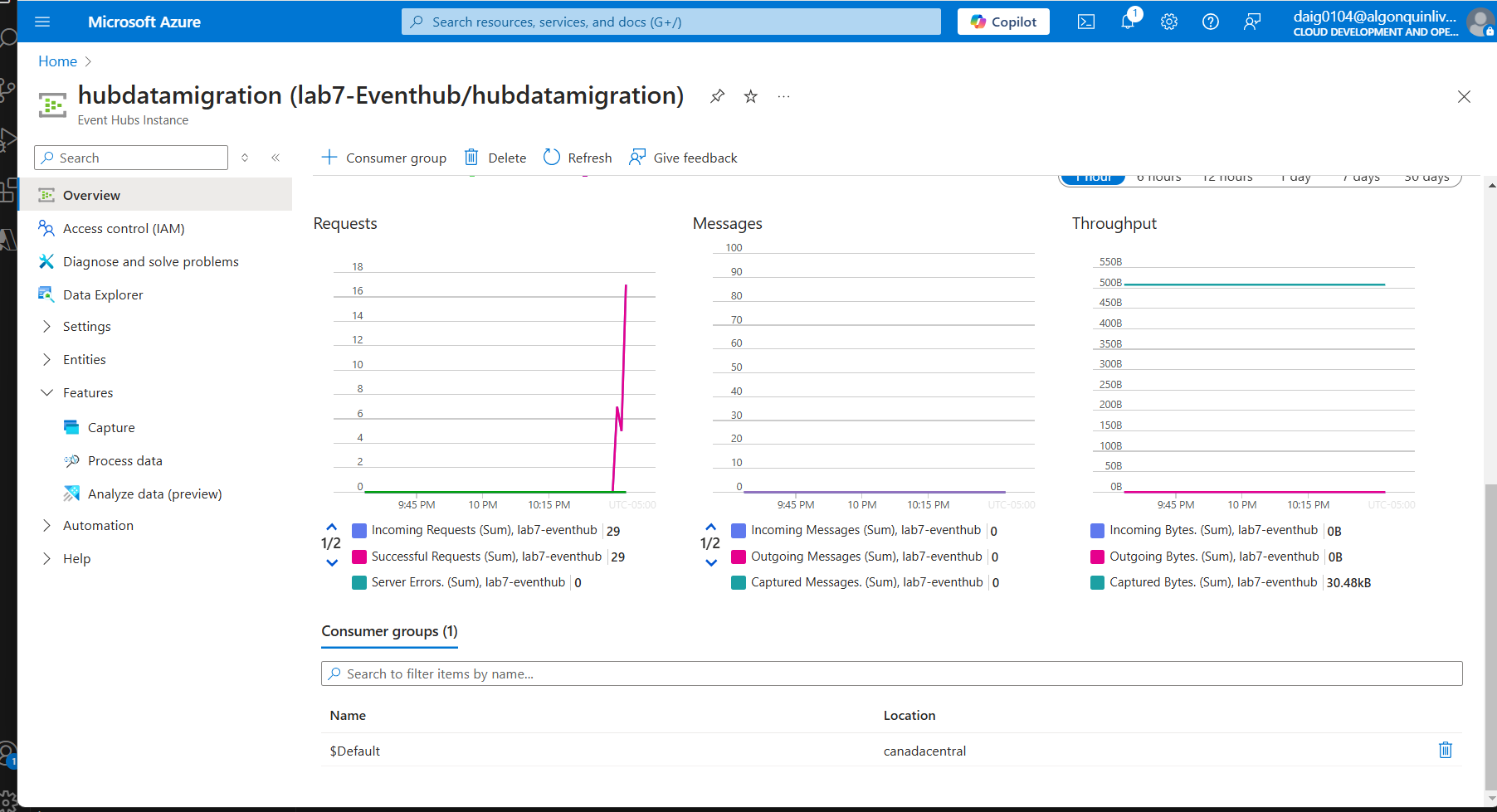
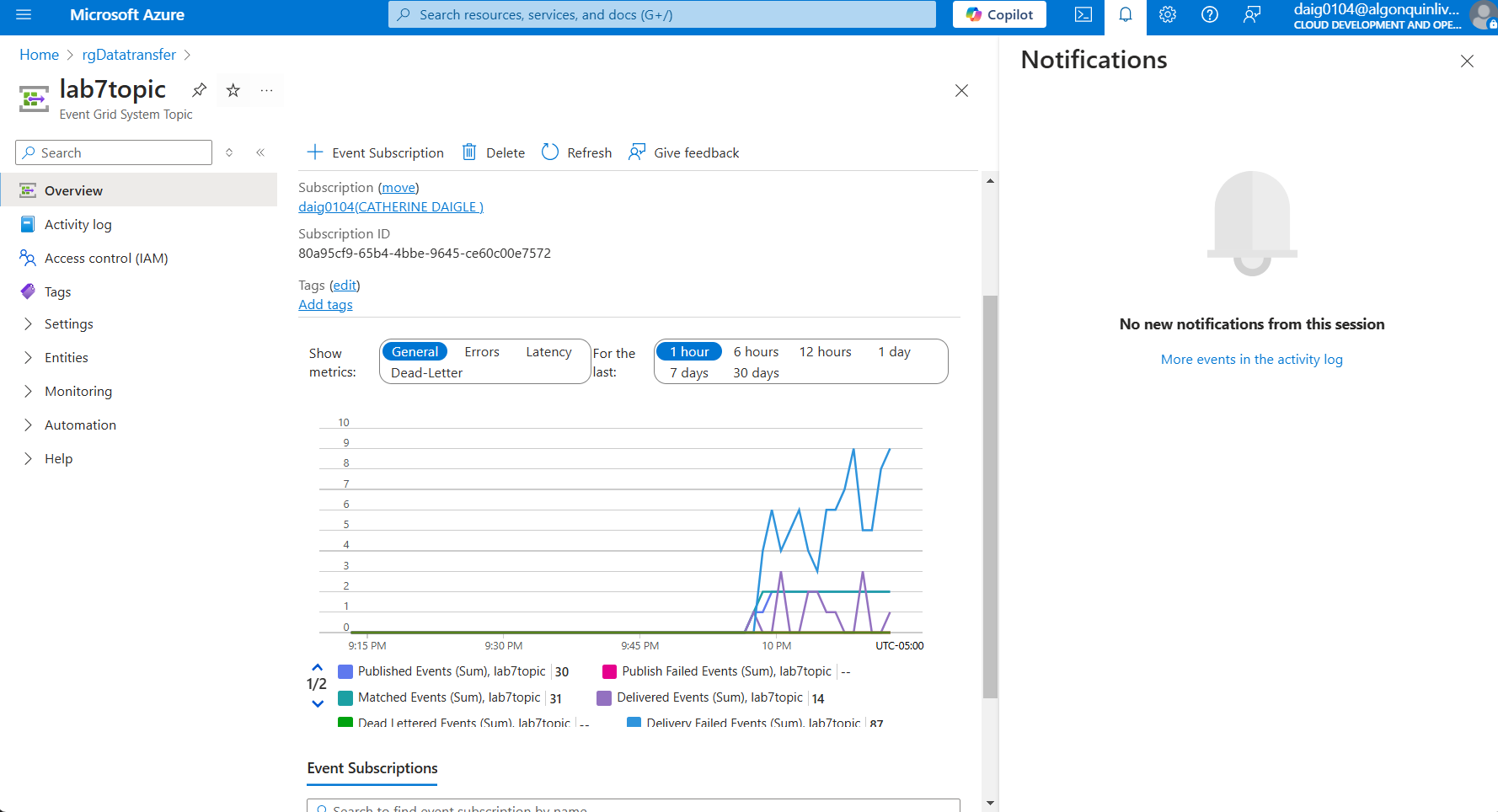
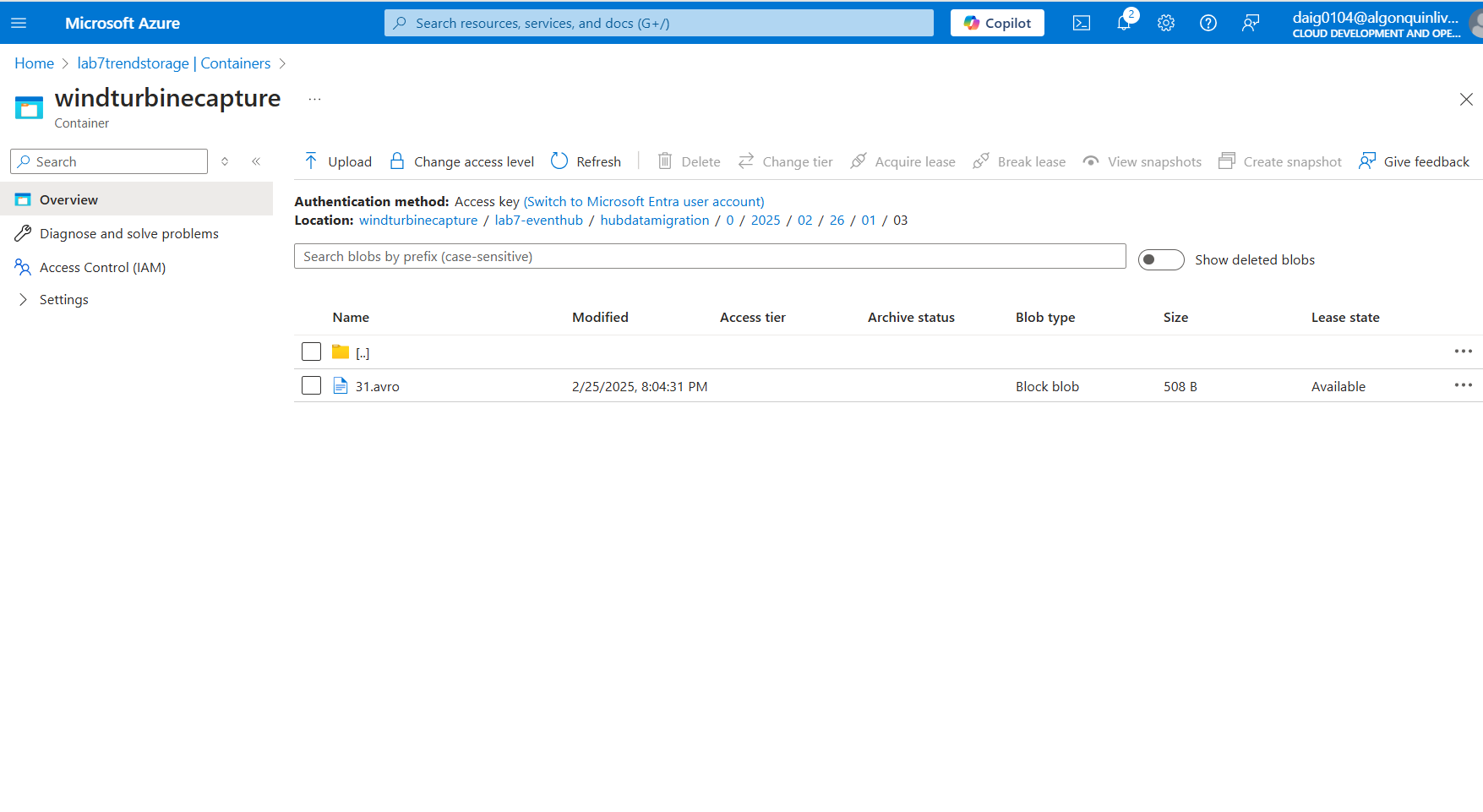
**Subscribe to events**

1. On the Event Hubs Namespace page, select Events on the left menu, and then select + Event Subscription on the toolbar.  
   
2. Creating an event subscription and assigning it the Azure function that was newly deployed as the endpoint:  
   
3. Verify that the event subscription is created. Switch to the Event Subscriptions tab on the Events page for the Event Hubs namespace. Below is the event hub subscription:  
     
   Event hub topic:  
   

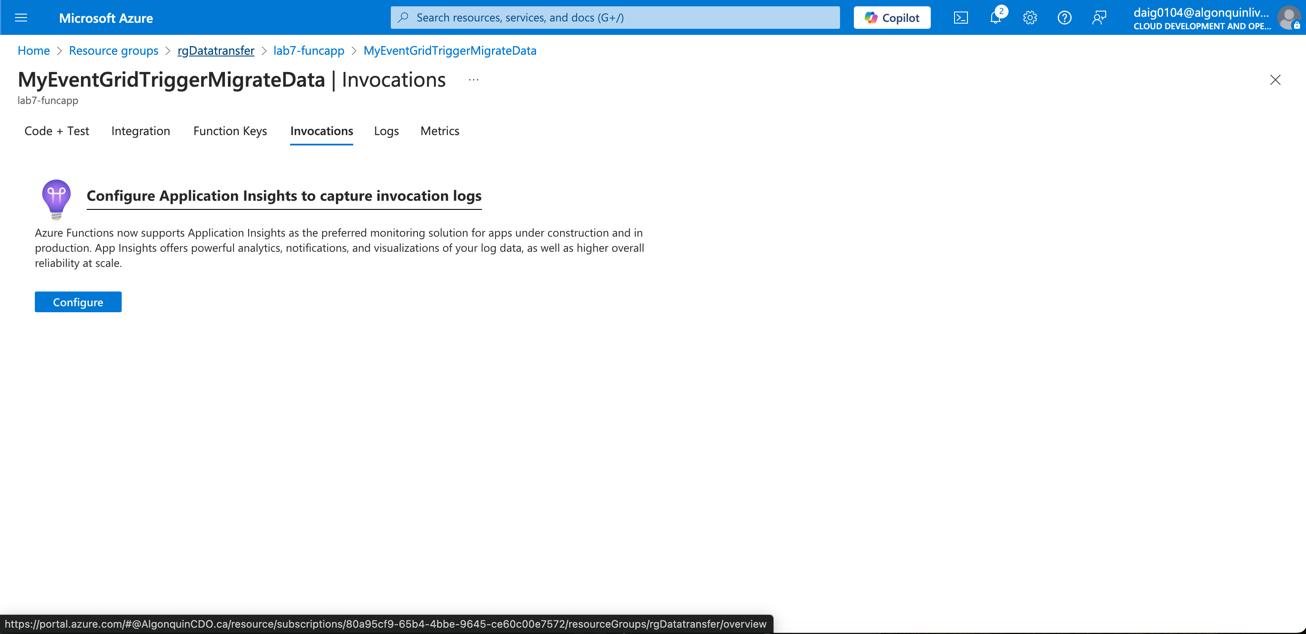
## Run the app to generate data.

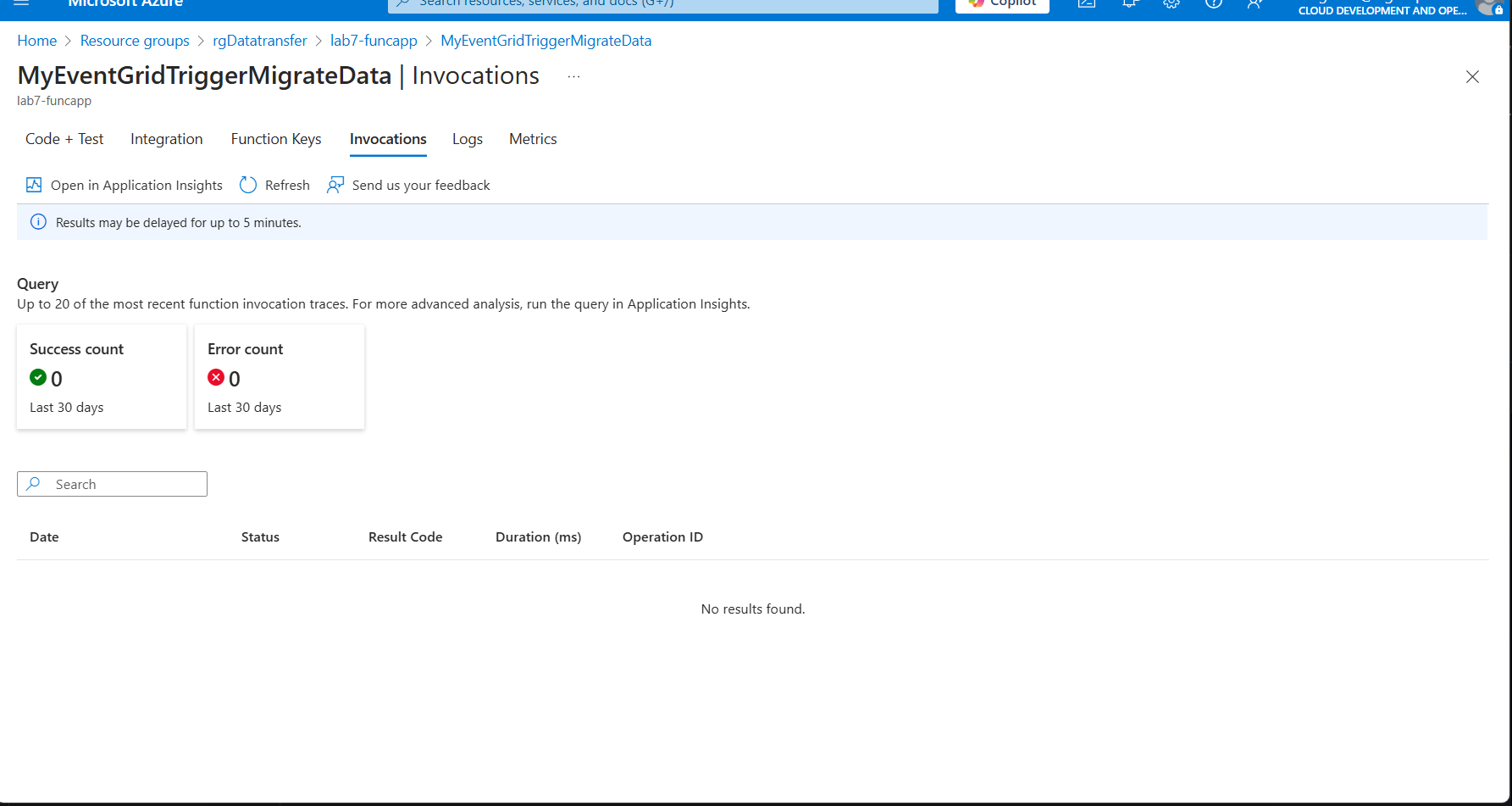
1. In the Event Hubs Namespace page, select Shared Access Policies on the left menu. Select RootManageSharedAccessKey in the list of policies. And copy the primary connection string.   
   
2. Replace <EVENT HUBS NAMESPACE CONNECTION STRING> with the connection string you copied from the portal within Program.cs.  
     
   
3. Build the solution. Run the WindTurbineGenerator.exe application.  
   I used dotnet build as shown below. This also can cause some dotnet errors if the correct version was not installed but the yellow is just vulnerability warnings.  
   
4. Running the application. (The error is just because I was not in the right folder I had to cd into /bin/debug. You will see the app running at the bottom with the …)  
   
5. After a few minutes, in the other browser tab where the query window is open, query the table in your data warehouse for the migrated data. Unfortunately, I could not get the migrated data as it did not appear with the query but I will now demonstrate my debugging process and the reason behind why:  
   
6. Within Tables when expanded it is apparent there is no data within the SQL therefore, the SQL database did not receive any data.

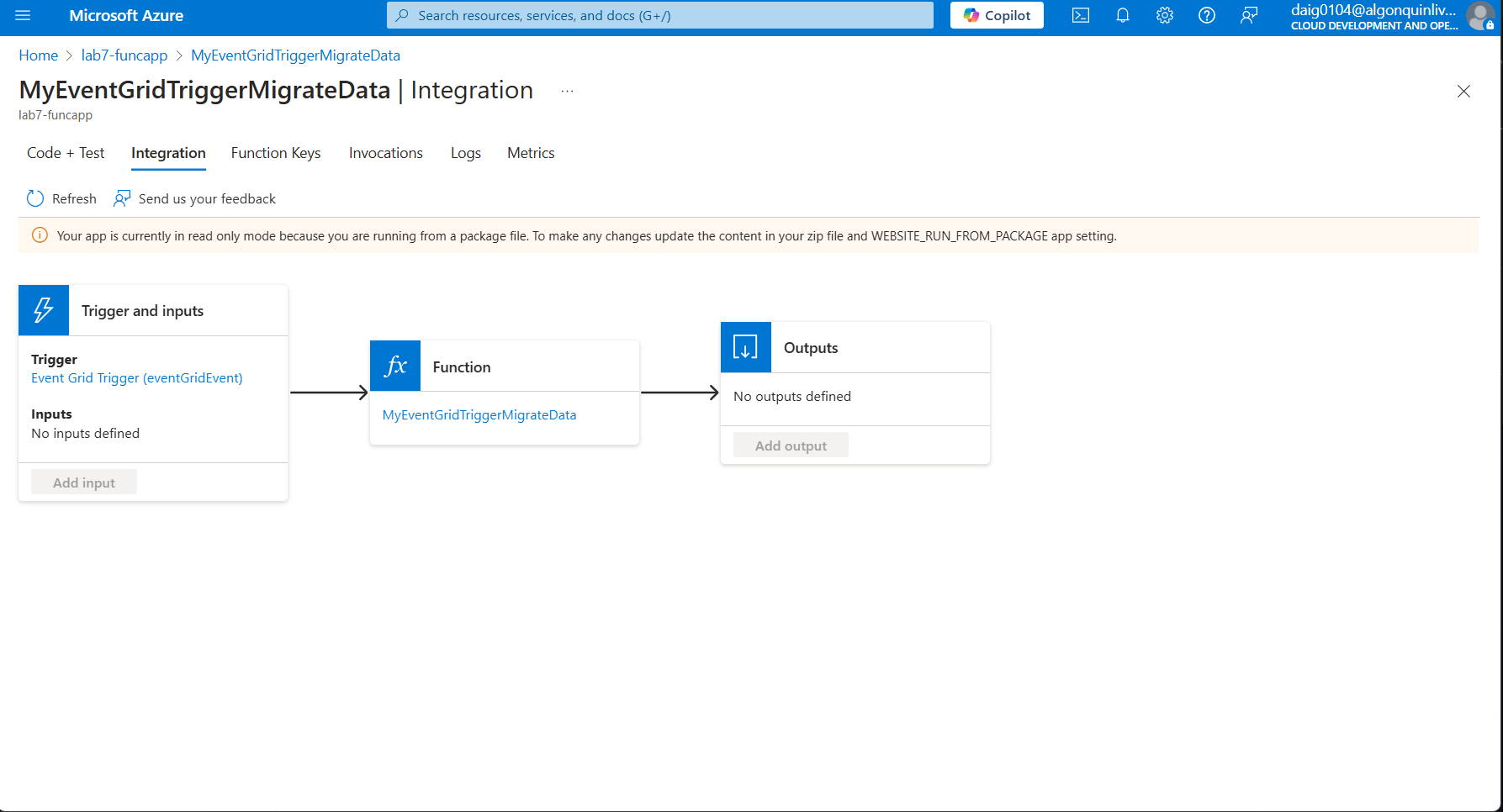
**Monitor the solution.**

1. I wanted to see if the Event hub was capturing data:  
     
   
2. These two slides above show the events published to the lab7topic and the successful requests to the Azure Events hub. This means that the application is indeed sending data to the Events hub, the next question is whether the data is stored within the storage account.
3. View captured data in the storage account, by drilling through the wind turbine capture container we can see the 31. avro container which is indeed the output expected from the application. This shot proves that the storage account is storing the files.  
   

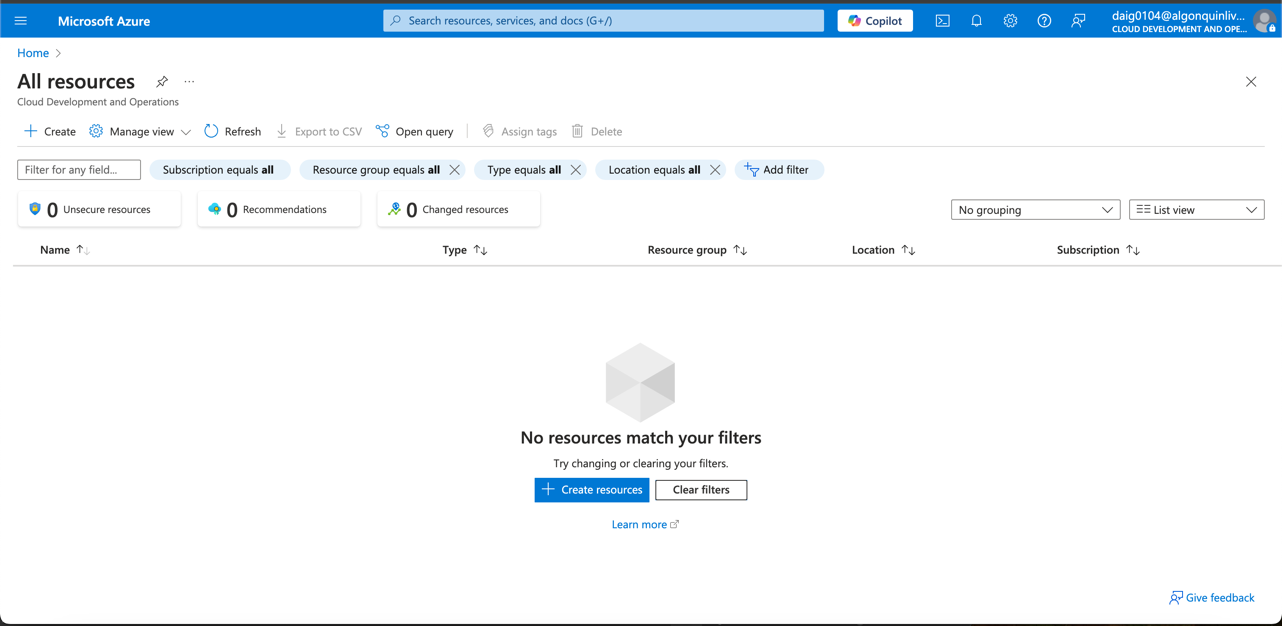
**Verify the Event grid trigger invoked the function**

1. Select Configure to configure application insights to capture invocation logs. The next thing to test I found is whether or not the function triggers, to do this I need application insights. The configure button automatically configures the insights to monitor.  
   
2. Here I look at the function Invocations and found that the function is indeed not triggering whilst the application is sending events to further confirm I further explore:



1. I do find that the function itself does not have an input or output defined. When looked at the function code within the portal I was only able to look at the function.json file. This could be that the function links to GitHub or that essential files are missing.  
   

**Deleting the Resources:**

1. Resource Deleted After:  
   

## Results:

The theory I had is that I would have trouble, as I expected with the previous lab dealing with .NET SDK, and as soon as I saw that it involved Azure Synapse, it would cost a bit more than usual. What I did not expect was the resulting overall cost and the function not triggering despite receiving data. The data could not migrate from the storage/event hub to the SQL database.  
  
In summary, Data flows successfully from VS Code to Azure Event Hubs. Data flows successfully from Event hubs to Storage, and the function does not trigger therefore, the data does not reach the SQL Pool. The resulting cost of error debugging for Azure Synapse is $19.00. (This is spent after a good chunk of my day)

# Conclusion:

In conclusion, I learned that the biggest issue, as with most programs and especially this lab, is that programs depreciate over time, and old resources should not be used but rather updated. Unfortunately, I found that .NET is rather restrictive, and programs under .NET 7.0 tend to be unsupported by most computers and applications today, especially when dealing with migrations from local to cloud. In my case, my Mac could not support .NET 4.8 developer SDK as well, and it could not support Visual Studio as Visual Studio for Mac became unsupported in 2024. Despite that, I switched to Visual Studio code and deployed the function that way. The screenshots above also show my debugging and error process although there was a lot more as I was trying and updating many different versions of .NET SDKs to satisfy the out-of-date program. For the function, I show my process with Screenshots within the section: “Monitoring the Solution” on how I figured out that the function is not triggering. I have two theories and one I have not tested is the cost of the hours/redeployment (I redeployed three times trying to fix the error) was getting too high.

1. The most likely error was that the function was not triggering because the function was not deployed properly from VS Code. This could also go with the same as the function app. Through investigation, I realized the function app .NET SDK does not exactly match the function SDK thus, this mismatch could cause the error. This could be why the function does not have any Input or Output within its integration. A fix ensuring the .NET dependencies are all aligned when deployed within the code.
2. The deployment itself uses Visual Studio Code and not Visual Studio. I do not have Visual Studio installed on my Windows; however, the incorrect deployment and function error could be solved if the function was deployed within the correct version of Visual Studio. Unfortunately, the version I own on my Mac is depreciated and does not load the project at the time. Installing the correct Visual Studio on my Windows machine would take too much time and running Azure Synapse through the debugging process costs much more than I expected.
3. Permissions Error (Highly unlikely) there could be a chance the function could not migrate data due to subscription permissions as with the previous lab.

Overall, I learned my costs might have been reduced if I turned off Azure Synapse when debugging the function, as it was not triggering in the first place. It cost me around $20 to spend a day to do and debug this lab. Additionally, I had to test and switch both from Windows and Mac to build the solution and run the WindTurbine.exe I had to install older versions of .NET SDKs, and those are unsupported by Mac. In short, I slowly started disliking .NET as a dependency because .NET seems highly incompatible with later versions, and many programs require a specific version or multiple specific versions of .NET. The correct .NET SDK should come within the program environments/requirements so that the process of installing SDKs is automated and fewer headaches are caused by searching for the correct ones.