**CST8921 – Cloud Industry Trends**

**Lab 9 – Serverless Computing Use Case**

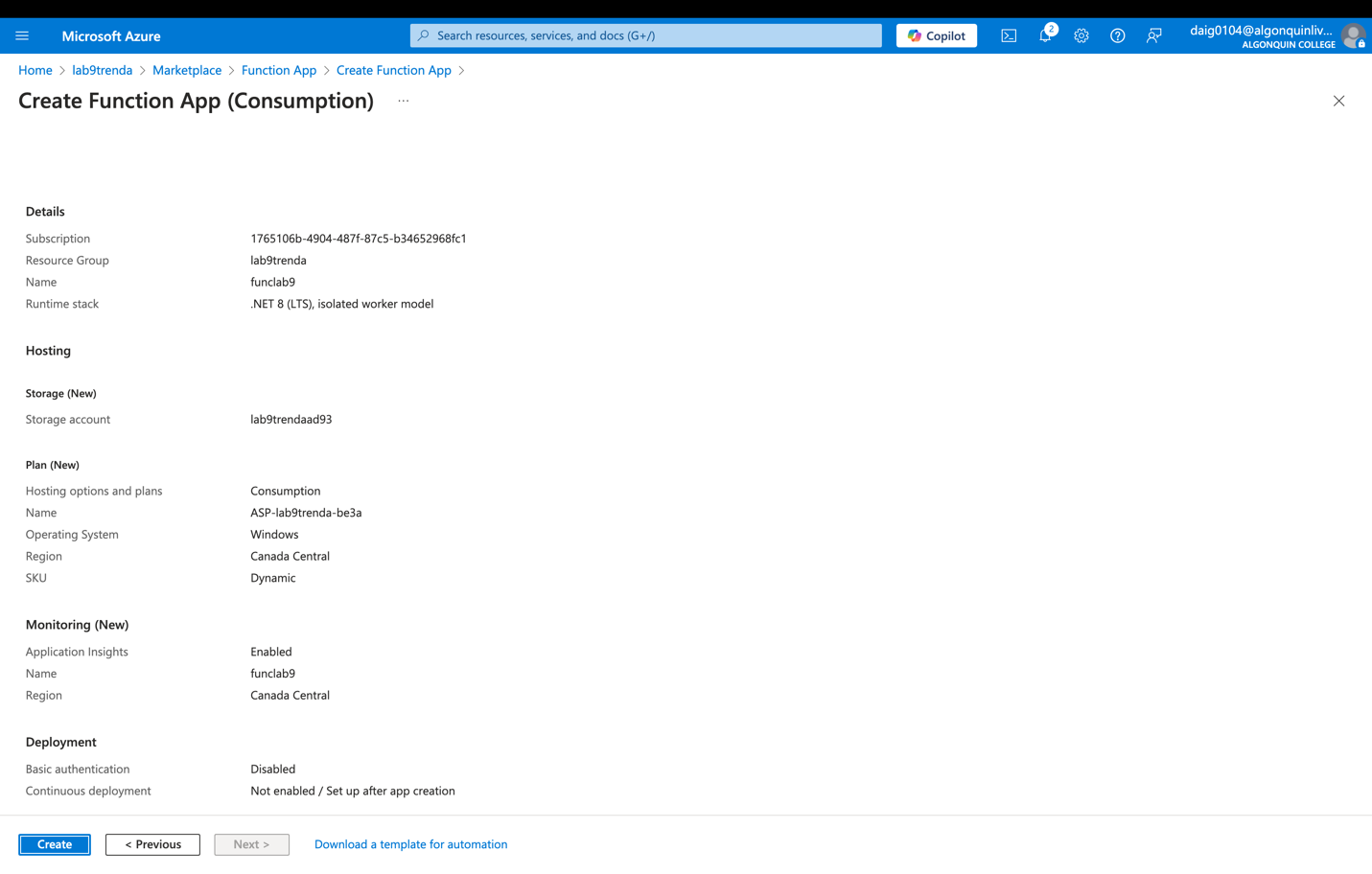
## Introduction

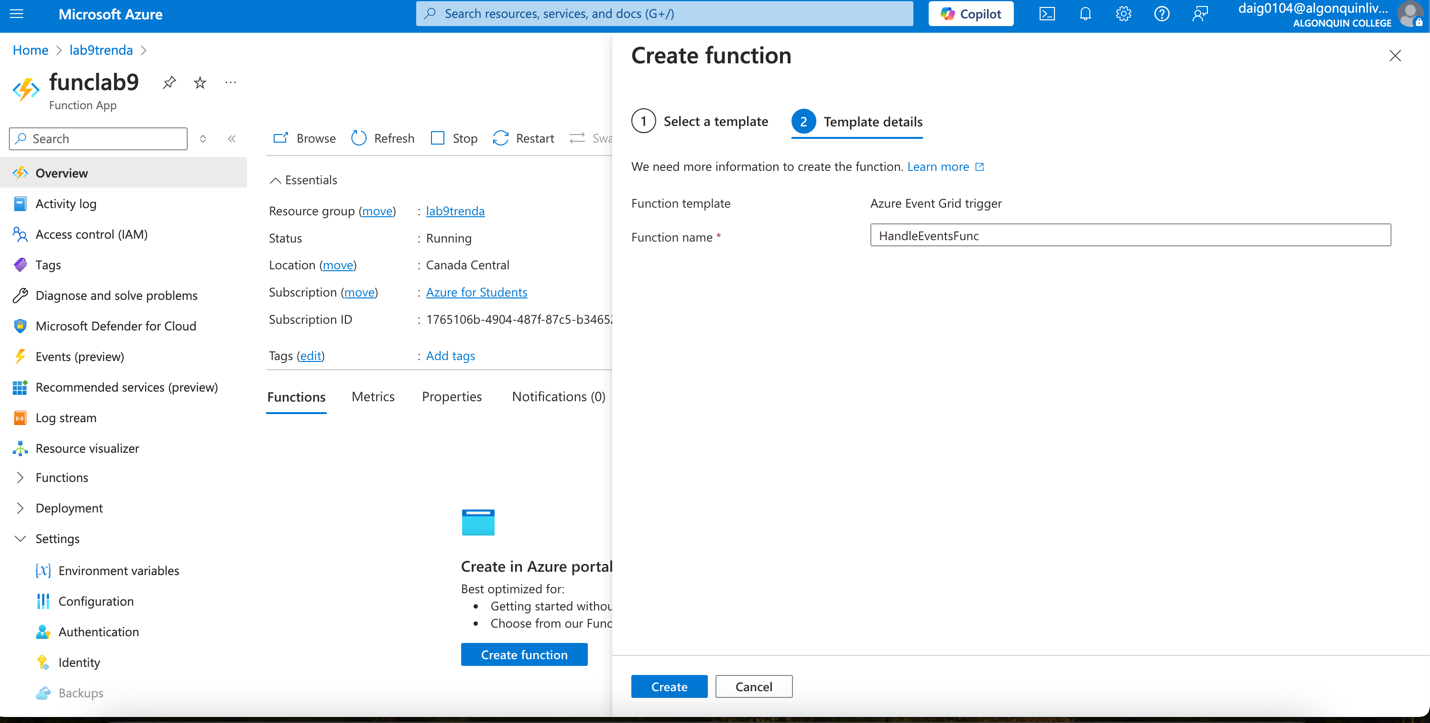
In this lab, I explore and discover serverless technologies such as Azure functions and Azure Event Grid. The lab demonstrates how Azure functions can work together with the Azure event grid monitor and handle when the event grid receives an event. The Azure function app within the lab uses a consumption plan; therefore, you only need to pay for what you use.

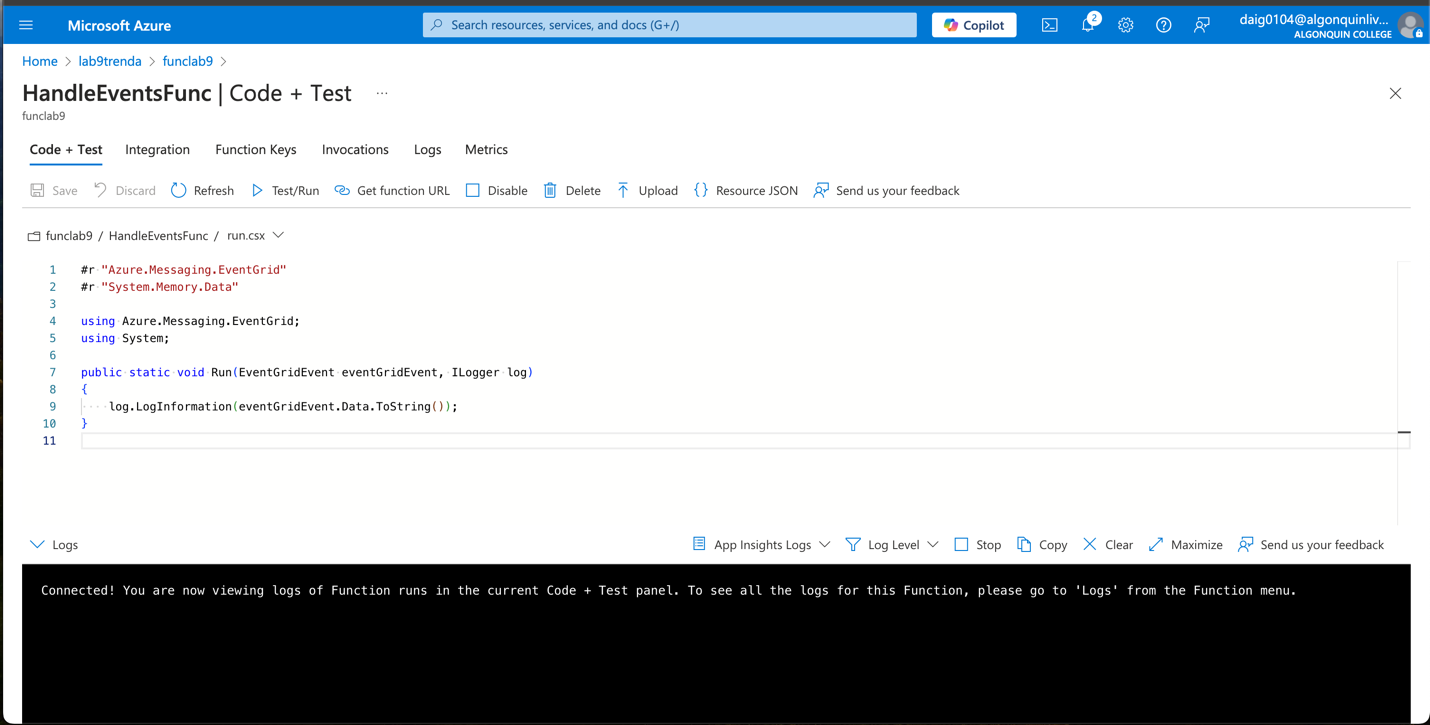
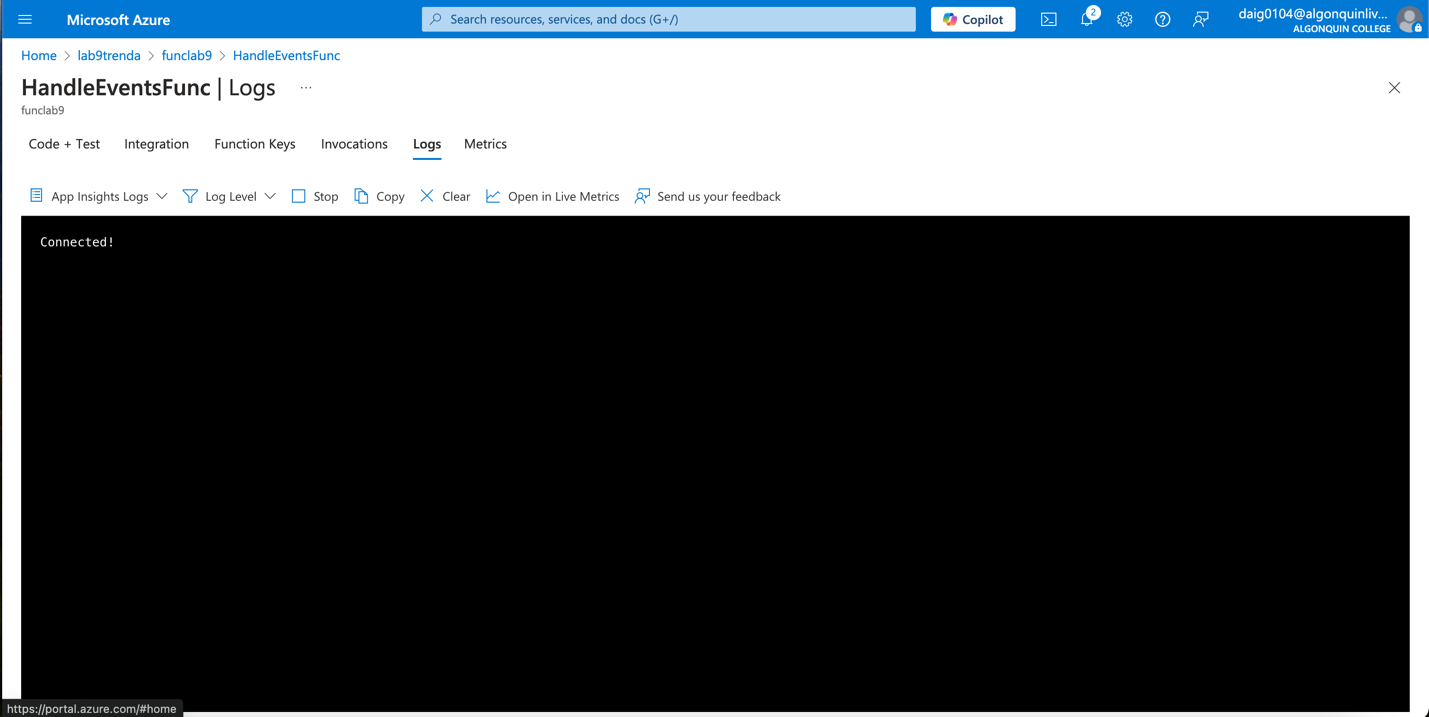
## Objective

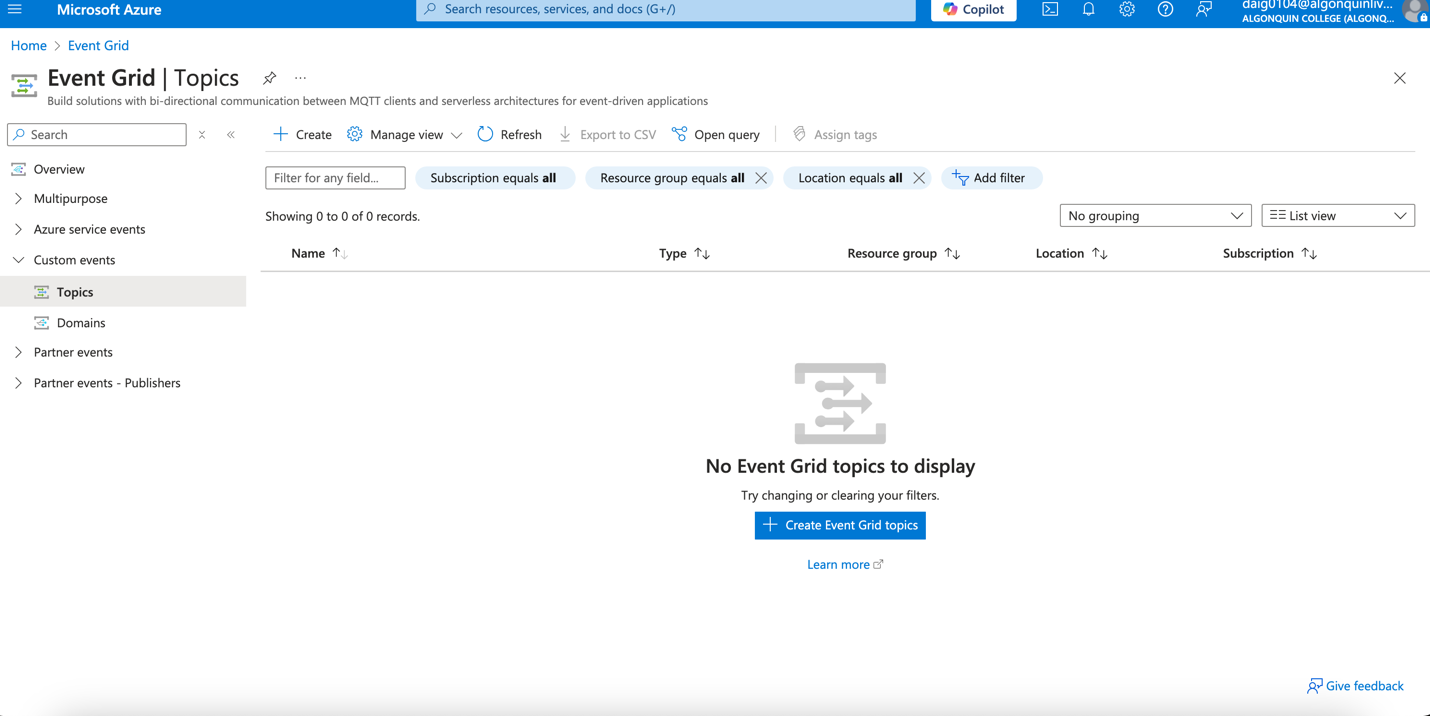
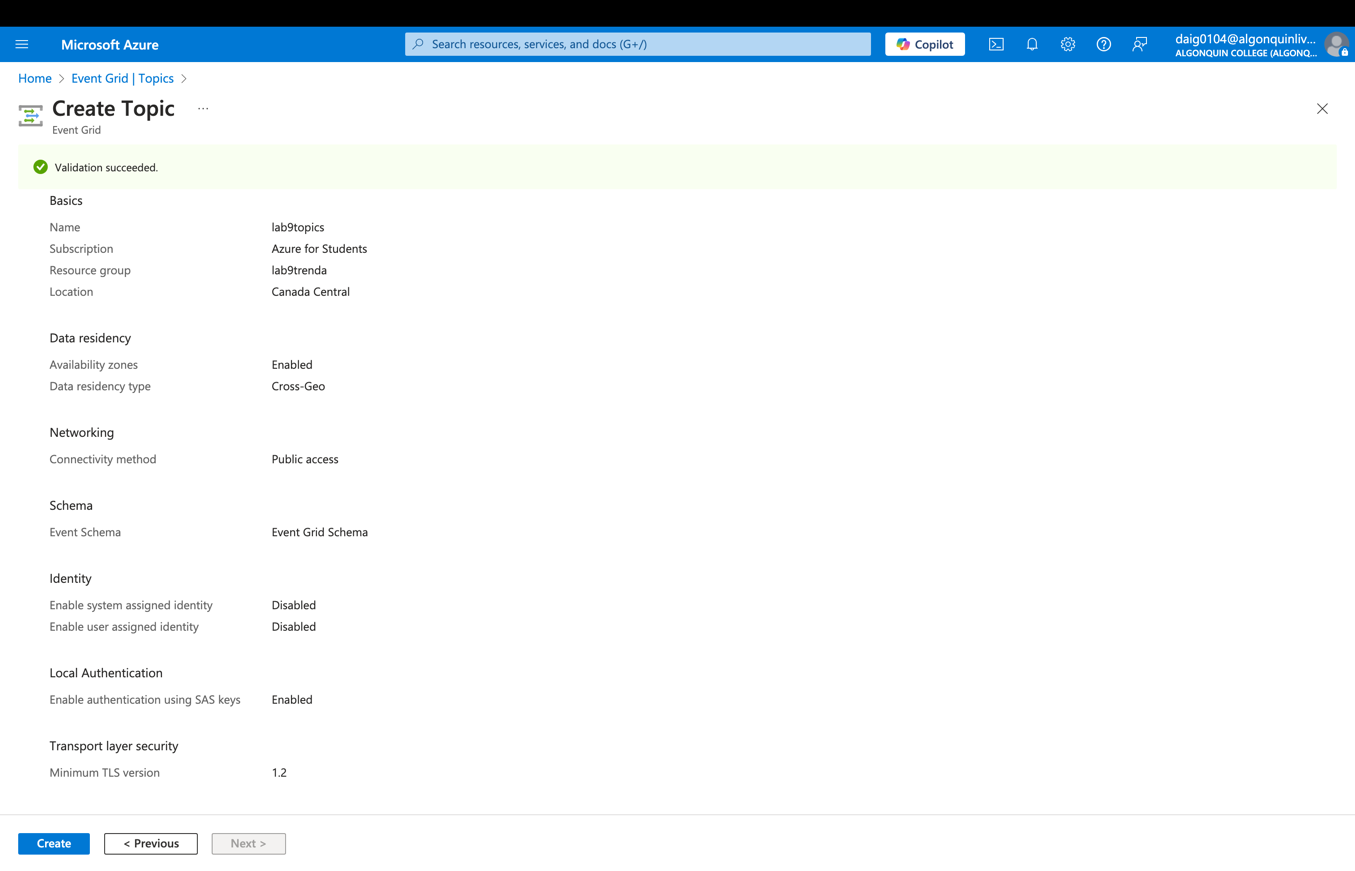
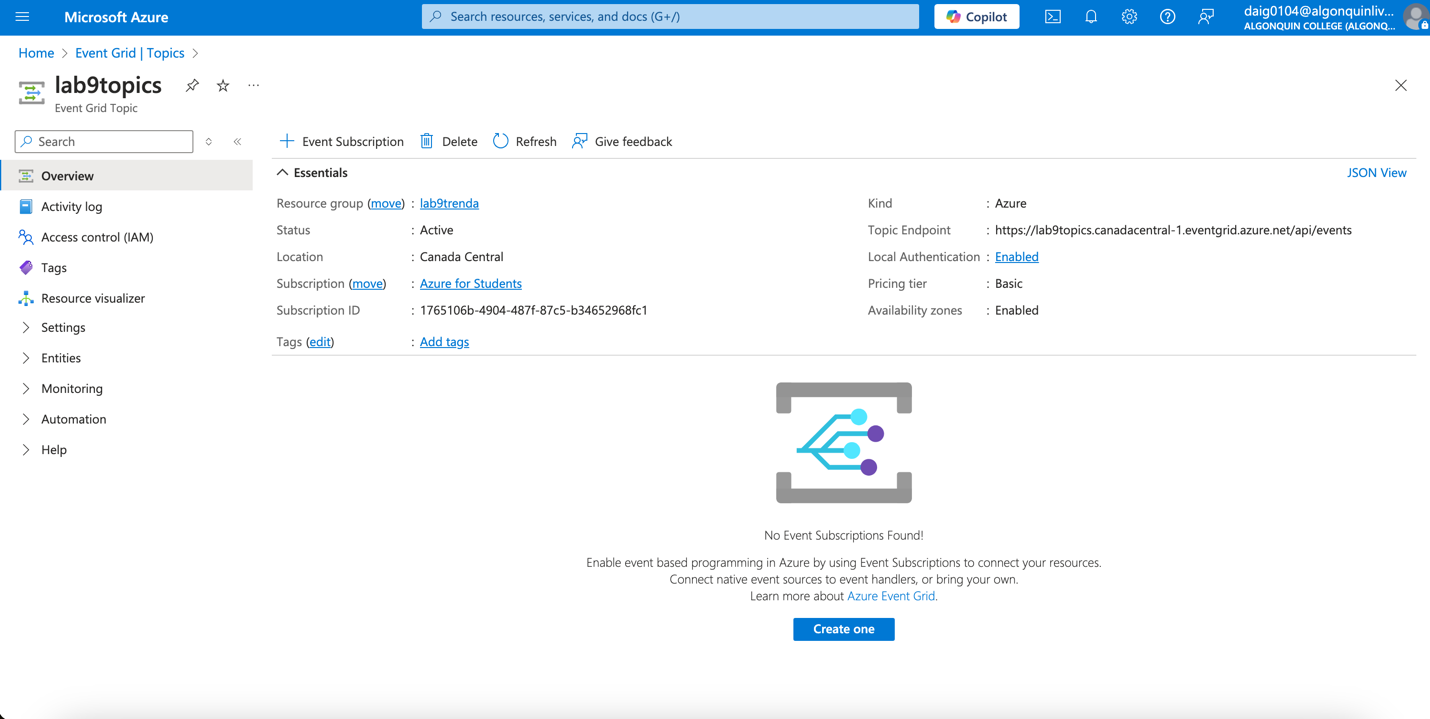
This lab activity aims to create an Azure function within Azure function apps to handle and connect to the Azure Event Grid topic.

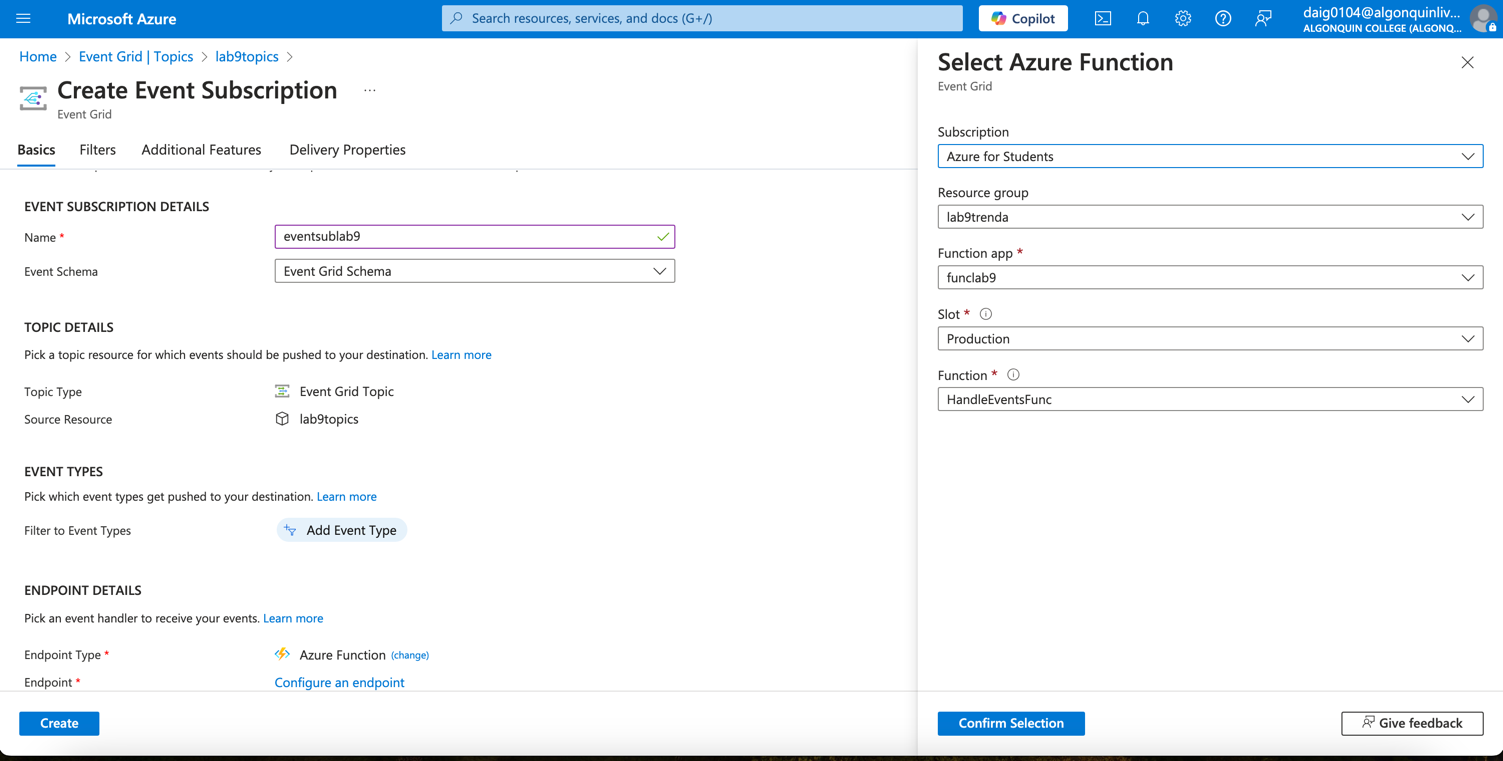
**Screenshots:**

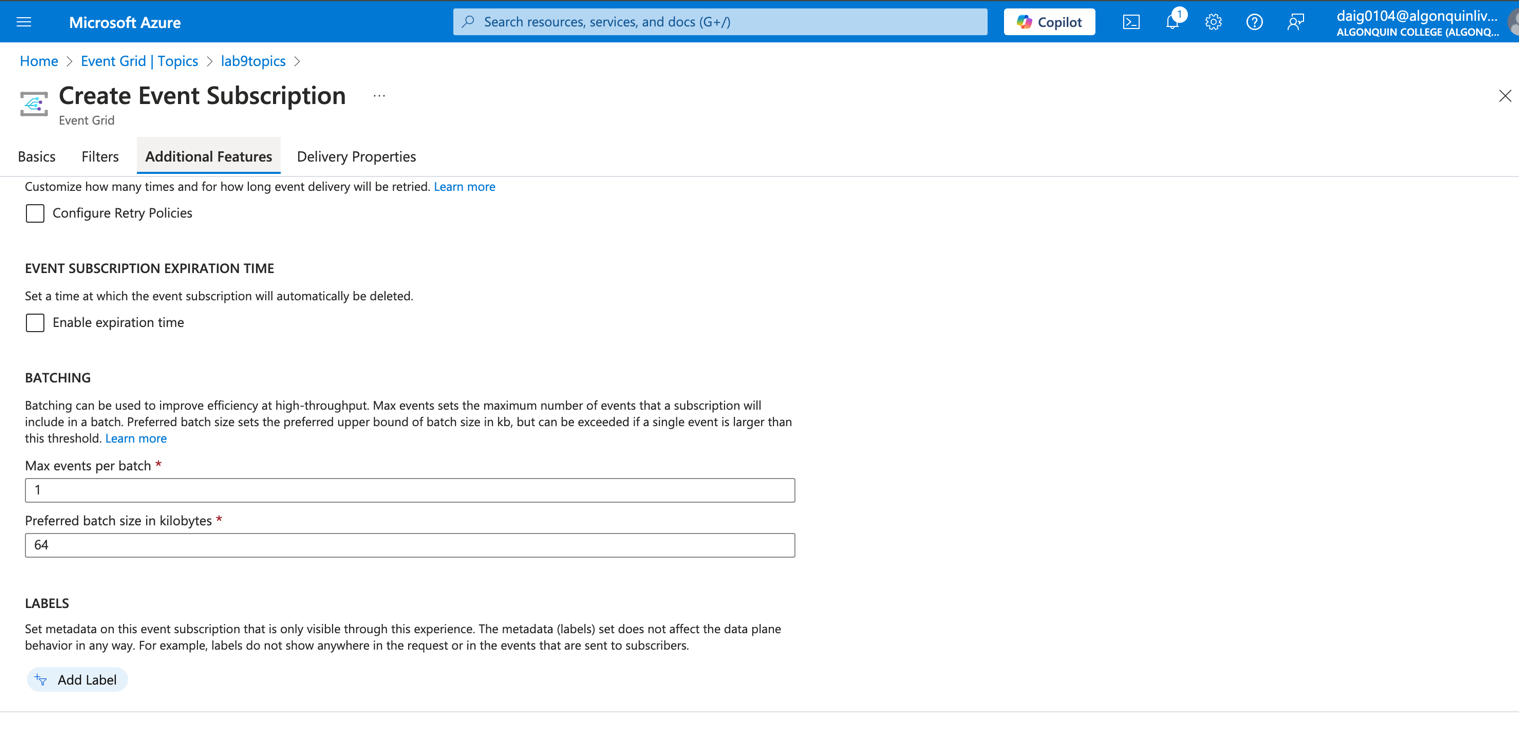
Step 1: Create a function app. This function app uses the Consumption plan and has application insights enabled for monitoring the function. .NET is runtime stack.  


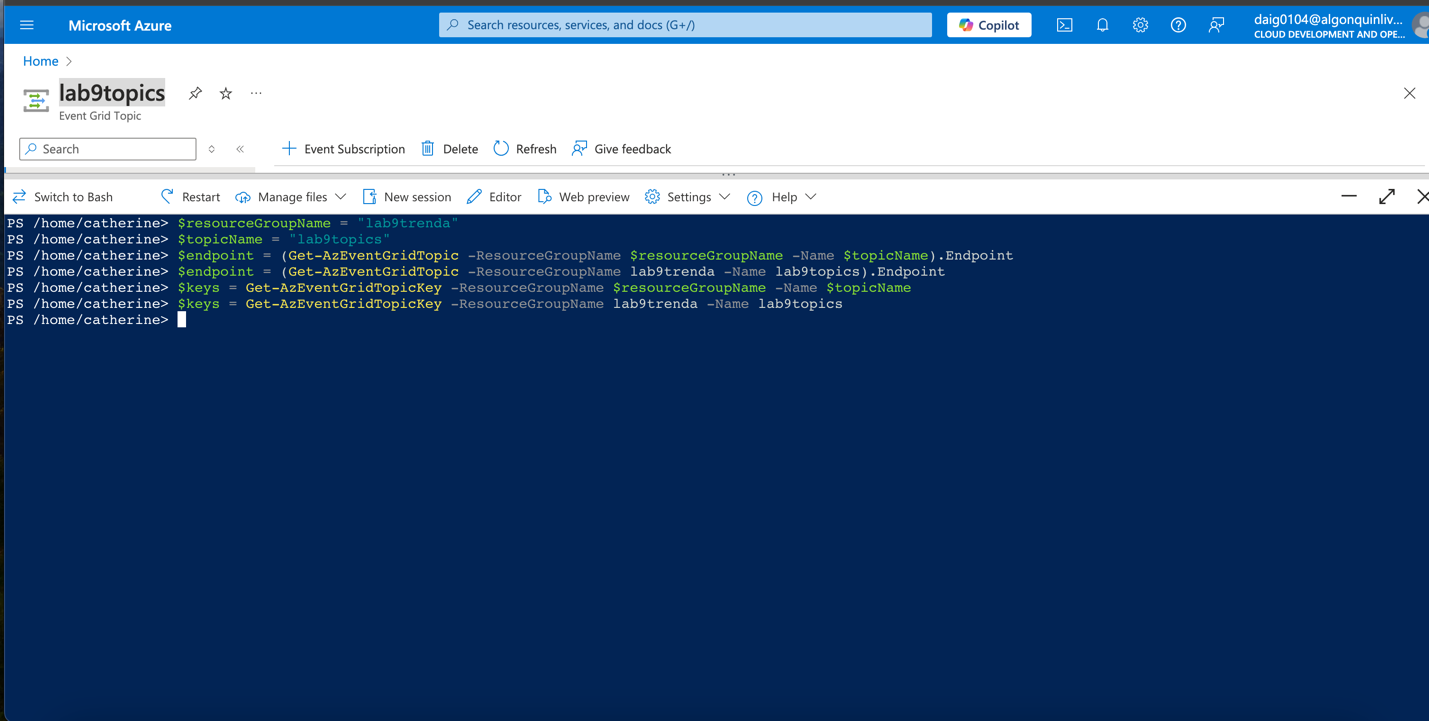
Step 2: Create a function and in the template section choose Azure event grid trigger. Note that to create a function within the Azure portal, WORKER\_RUNTIME = dotnet-isolated would need to be changed to WORKER\_RUNTIME = dotnet within the Azure app function environment settings.  


Step 3: Replace the code in the function to log & event grid data:  
  
  
Step 4: Save and go to the log page to monitor function.  


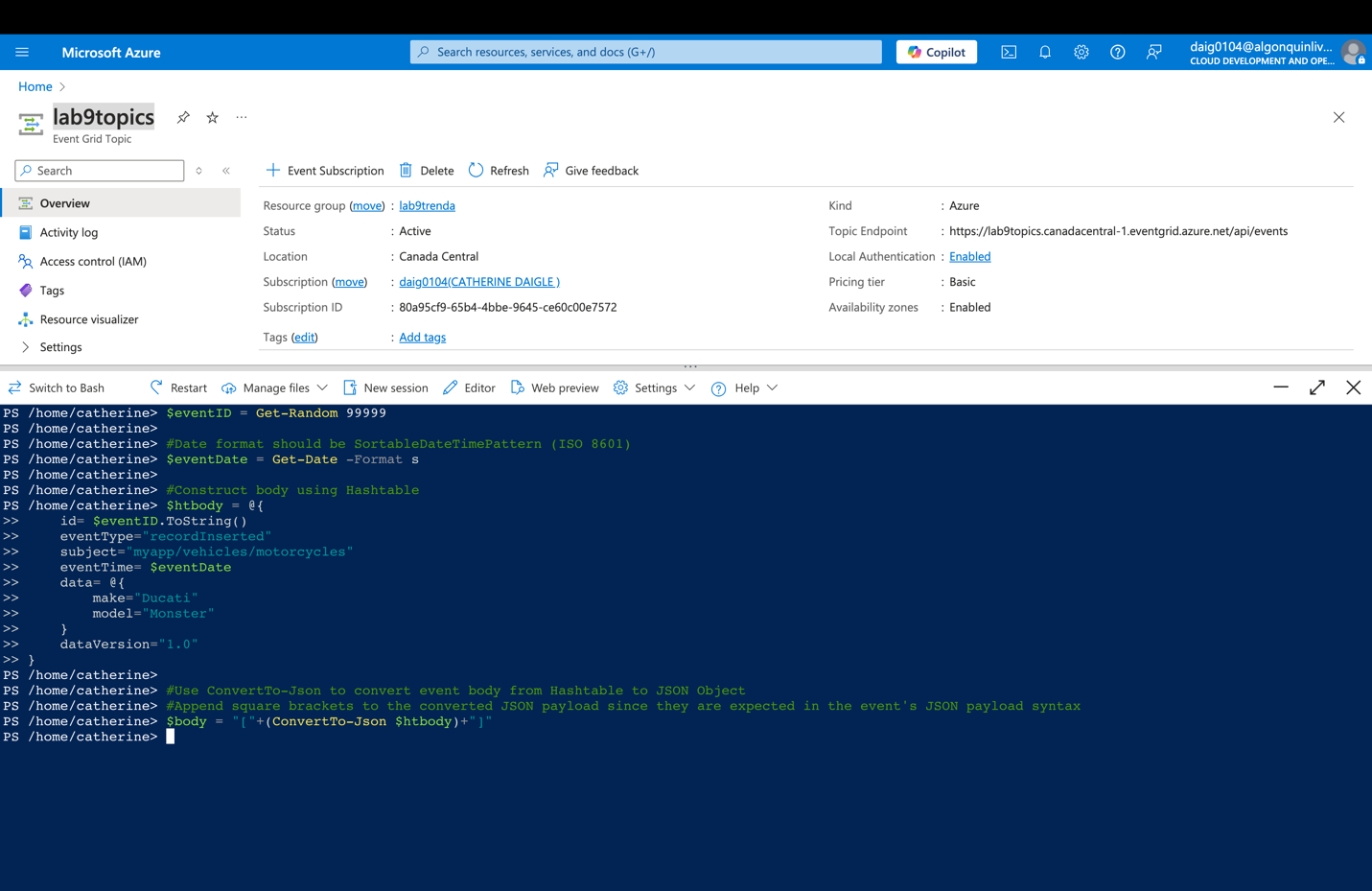
Step 4: Create a custom Event Grids topic. (It is mostly default.)  
  
  
  
Step 5: Create an Event Grid subscription.

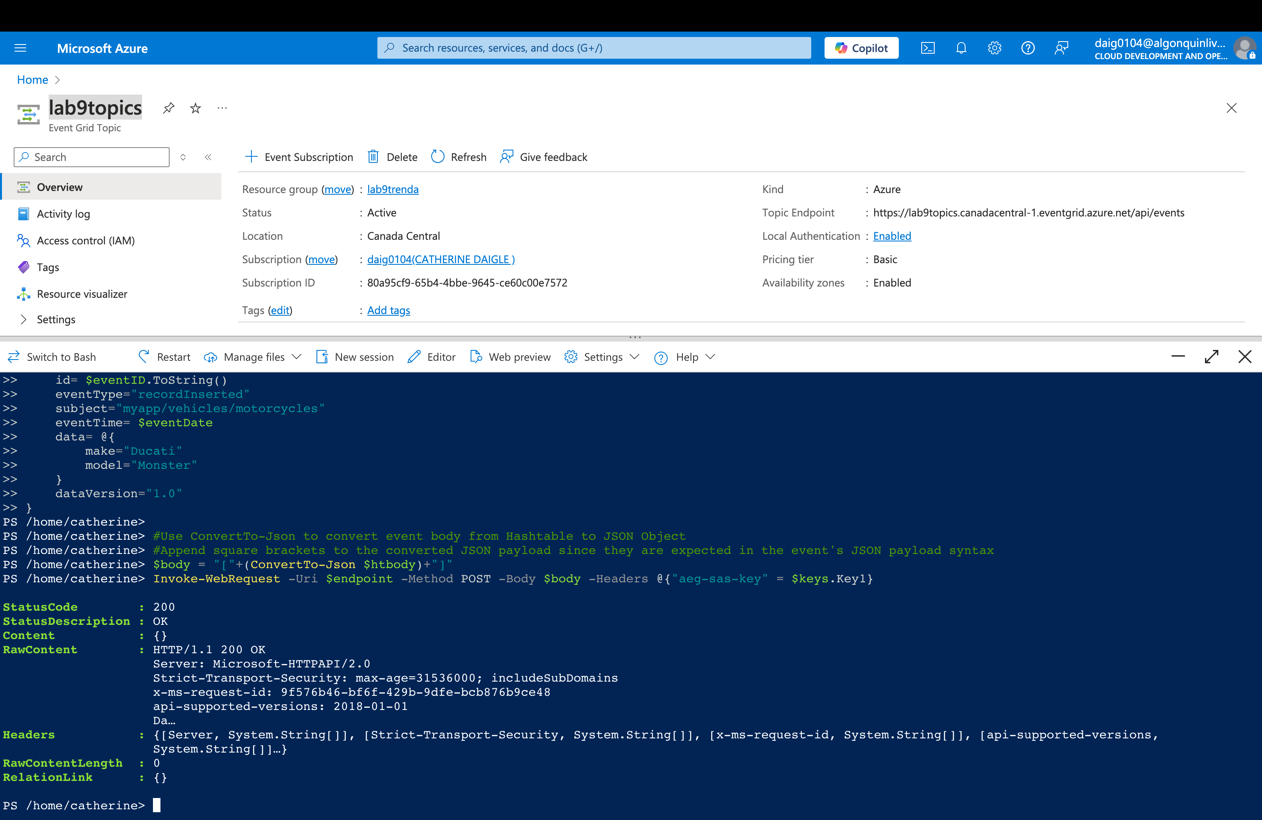
Choose the Azure function created previously as the function endpoint.  


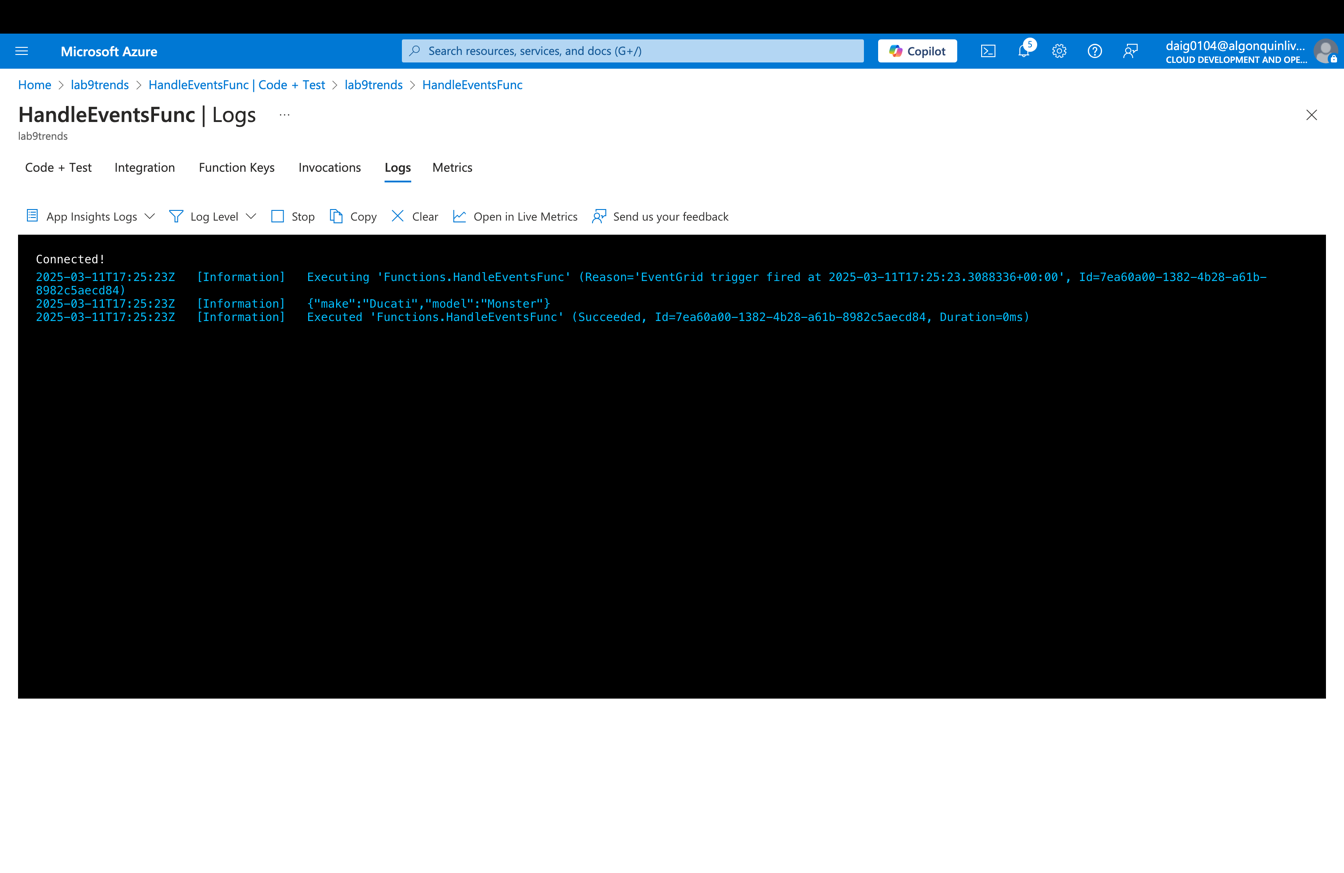
On the Create Event Subscription page, within the Advanced Features tab, set values for Max events per batch and Preferred batch size in kilobytes. It is the default batch size.  


Step 6&7: within powershell, set the variables to the topic name and resource group (it is not a typo; the resource group is called lab9trenda). Additionally, get the end points and keys for the topic as shown below.  


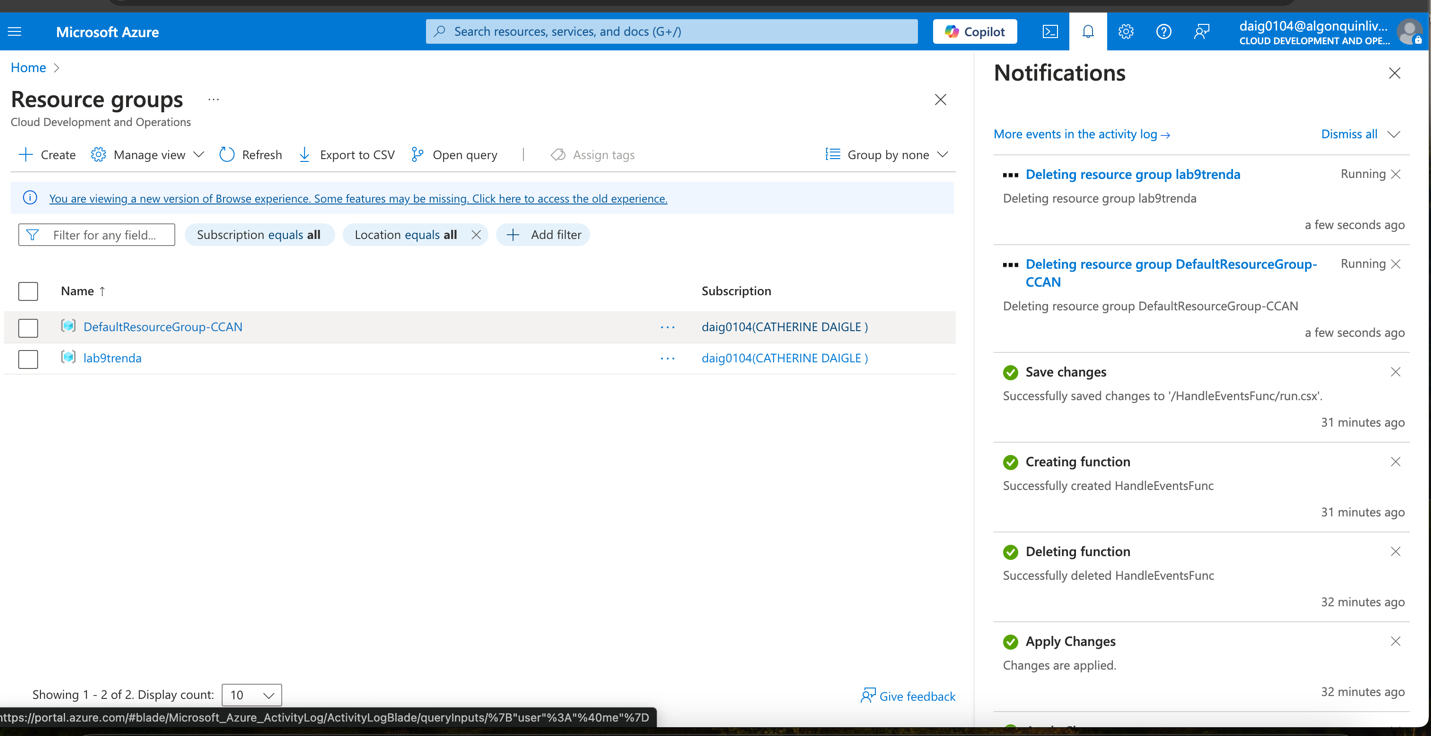
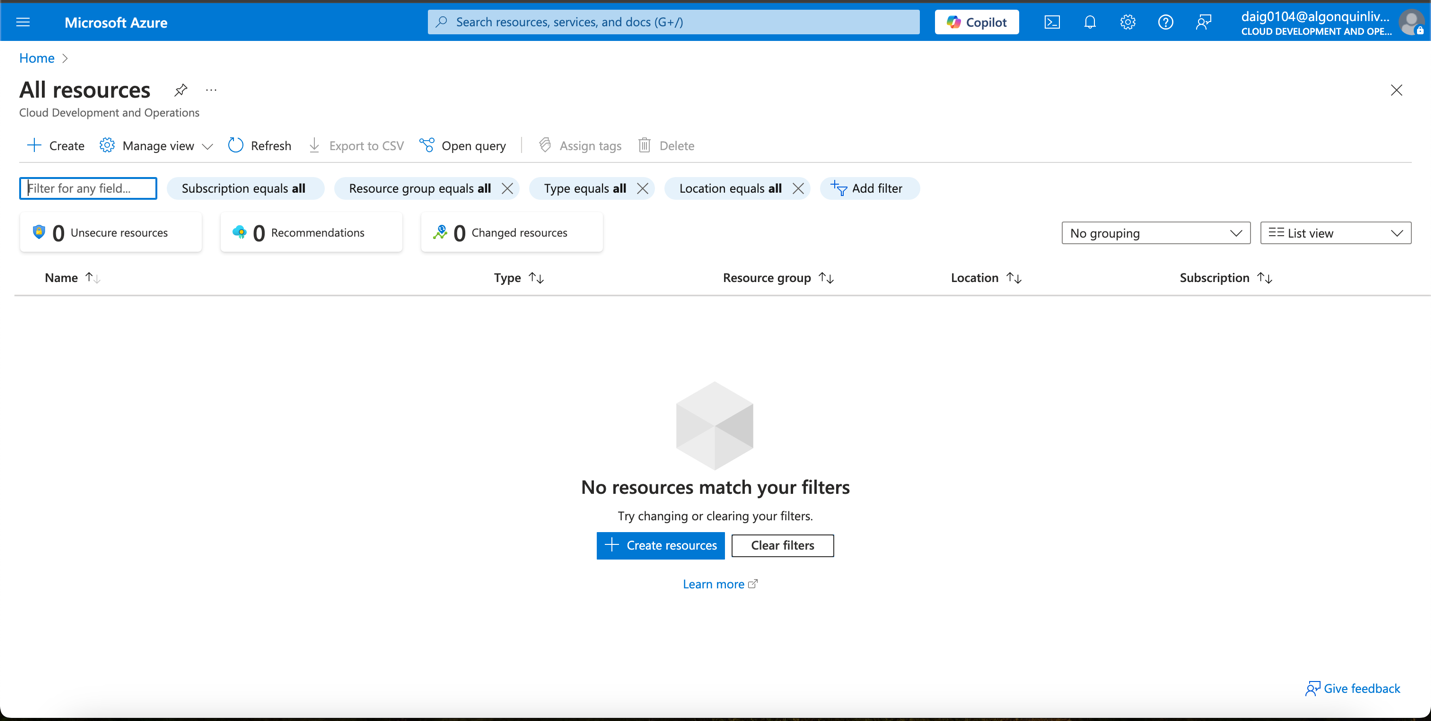
Step 8: Prepare the event. Note that there was an error due to the id not being a string; it was solved by adding.ToString() to reformat the ID to a string.



Step 8: Use Invoke web request to send the event and test the function.  
  
  
Step 9: Verify that the function received the event. Below, you can see the motorcycle has been processed by the Event grid handler function.



Step 10: Clean up the resources. Before:

  
After:  


## Result:

The lab's result went much more smoothly than expected, with minimal errors. It has output a successful log and handles Azure app functions, as well as the function outputs a motorcycle’s make and model.

## Conclusion:

In conclusion, I had very few errors within this lab however, I have learned some new interesting tricks and things that may smoothen future lab processes. I found that the Bash script causes fewer errors than the PowerShell equivalent. For example, within Step 8. I had to convert the ID value to a string with the .ToString() method to avoid an error when handling the ID. Within bash, that conversion was not required. Another error I encountered is that the button to create a function is missing when first creating a .NET app function. An easy way I found to resolve that issue is to go within the Function app > Settings > Environment variables and set WORKER\_RUNTIME = dotnet-isolated to WORKER\_RUNTIME = dotnet, and the button will appear again.

Overall, I have achieved the objective as I was able to successfully trigger the function when the Event Grid topic receives a message onto which the function outputs a log of the motorcycle’s make and model, which was “Ducati” for the make and “monster” for the model. With Azure serverless functions, it can allow you to keep track of, alert and handle messages when the Event Grid topic receives them.