CST8921 – Cloud Industry Trends

Title

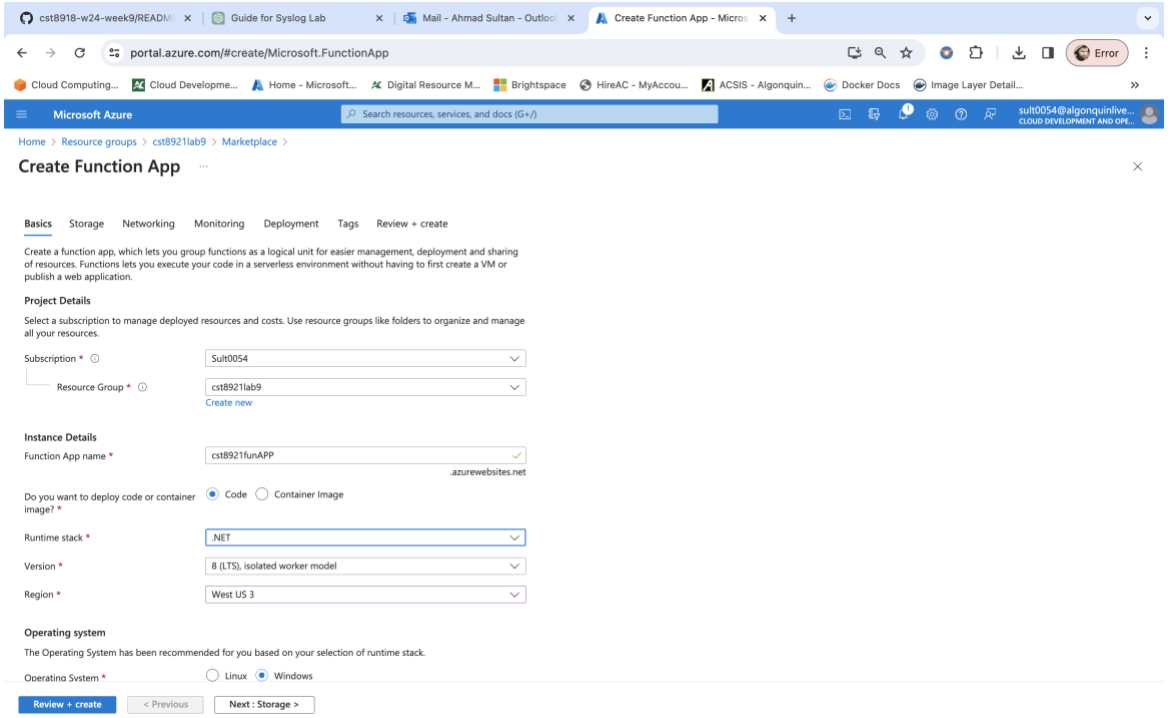
Exploring Serverless Computing in Cloud Environments

Summary

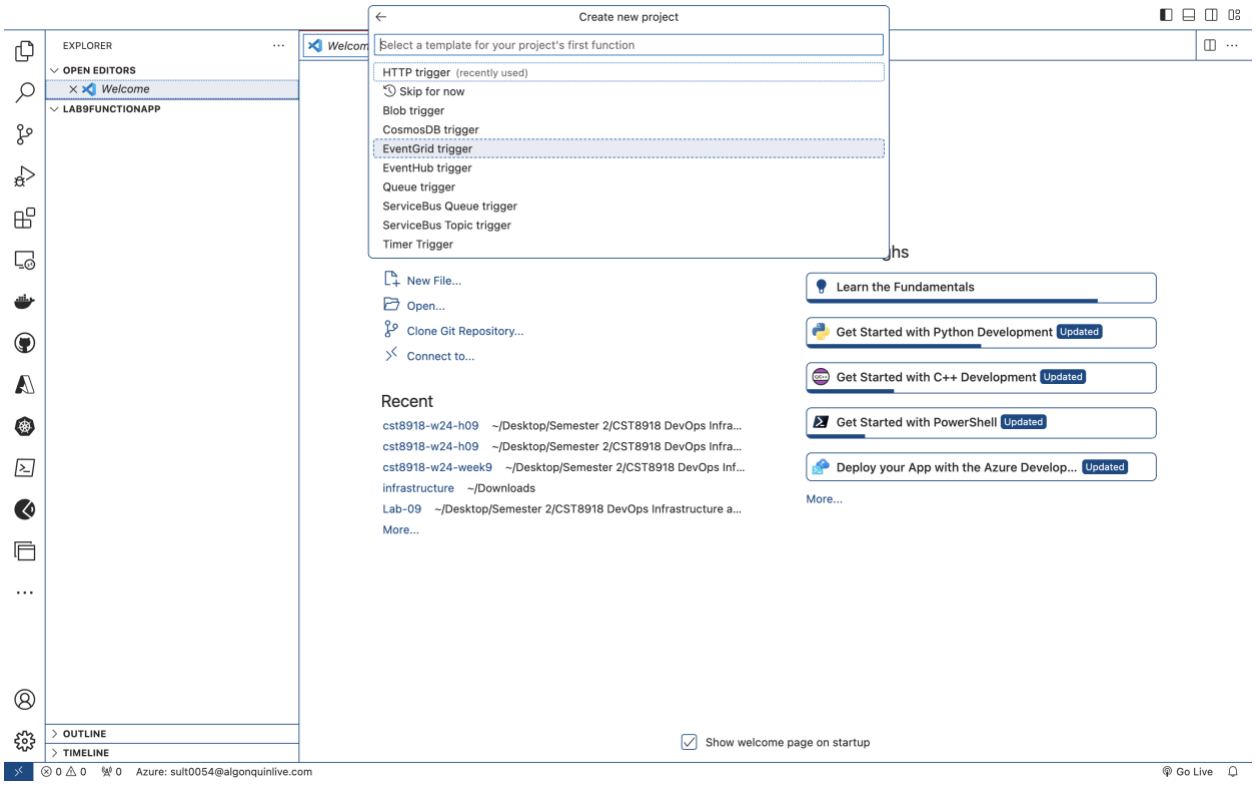
This lab introduced students to serverless computing services using Azure Functions and Azure Event Grid, emphasizing the elimination of infrastructure management tasks and focusing on code. The hands-on experience aimed to familiarize students with creating, testing, and monitoring serverless applications, specifically through the integration of Azure Functions with Azure Event Grid for event-driven architectures. The lab covered the entire process, from setting up a function app and event grid trigger to sending and receiving events, showcasing the agility and cost-effectiveness of serverless technologies.

Steps taken

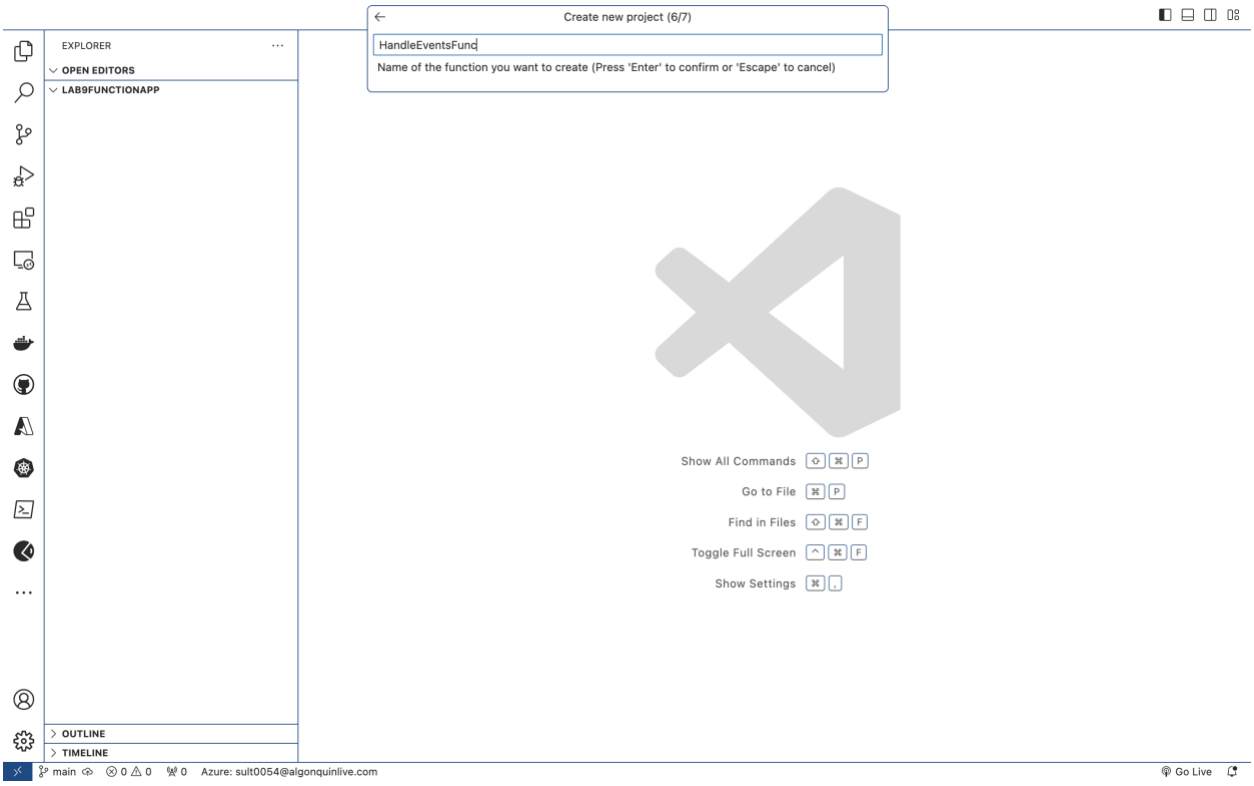
1. Create a function app
2. Choose .NET as runtime stack

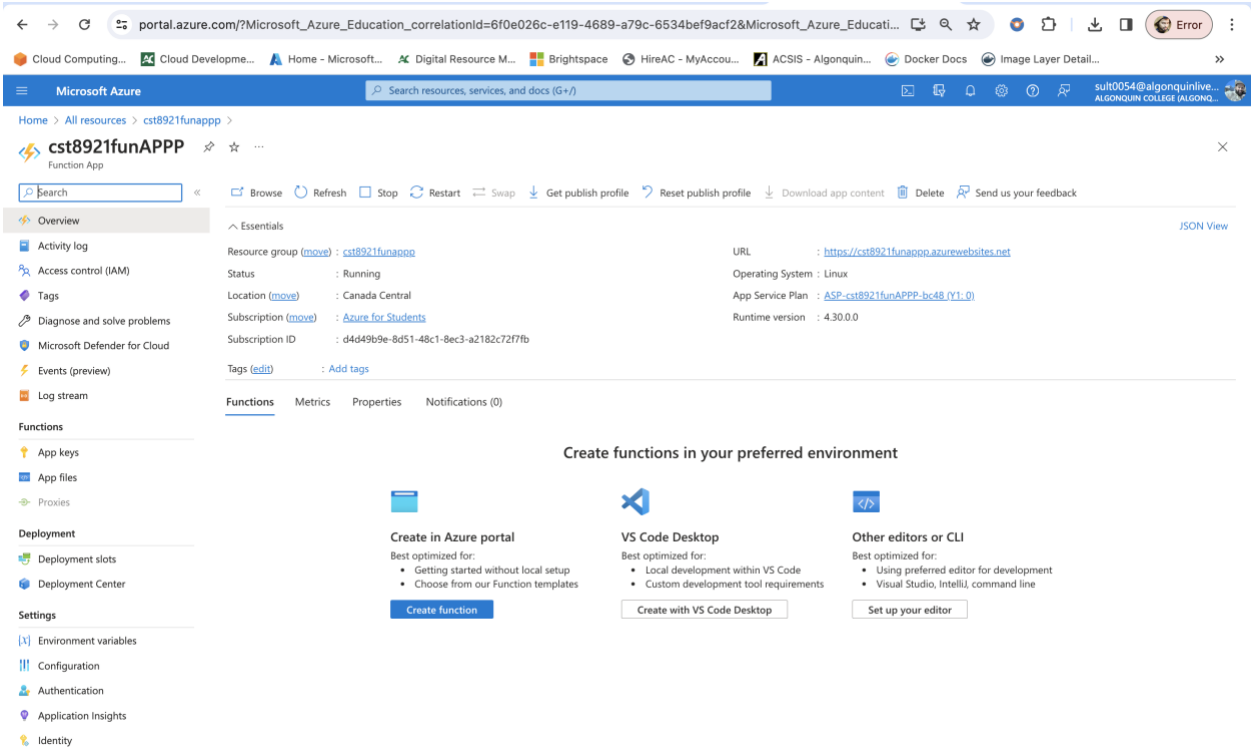


3. Create a function and in template section choose Azure event grid triger
   There was no option to create the function in the azure portal so I used VS Code



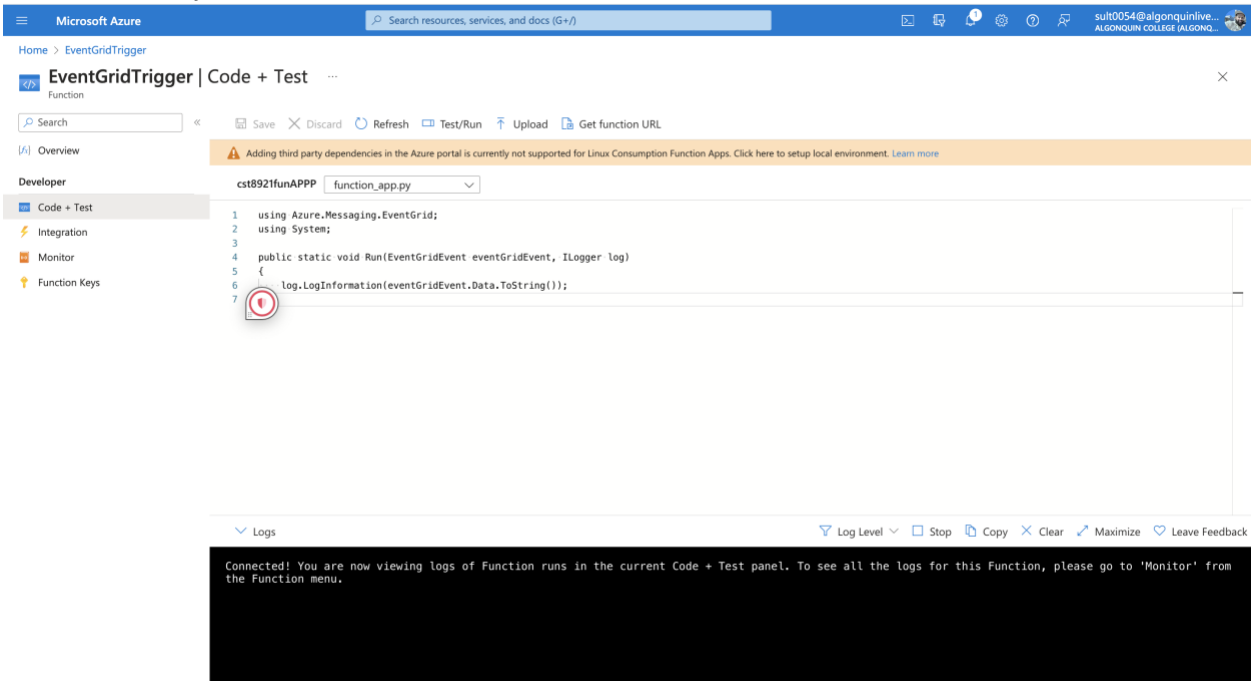
4. In the Template details section in the bottom pane, enter a name for the function. In this example, it's HandleEventsFun



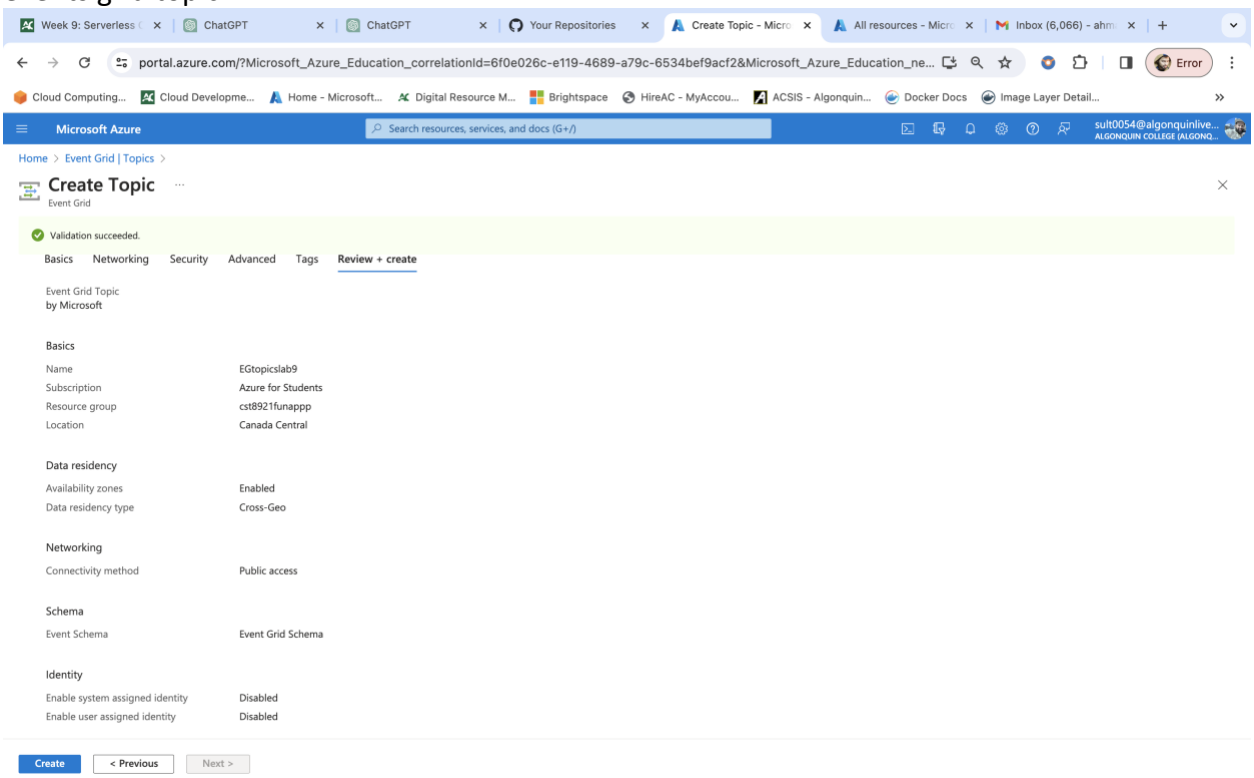


5. On the function page for handleeventsfunc, select code+test
6. Replace the code with the following code and test the function

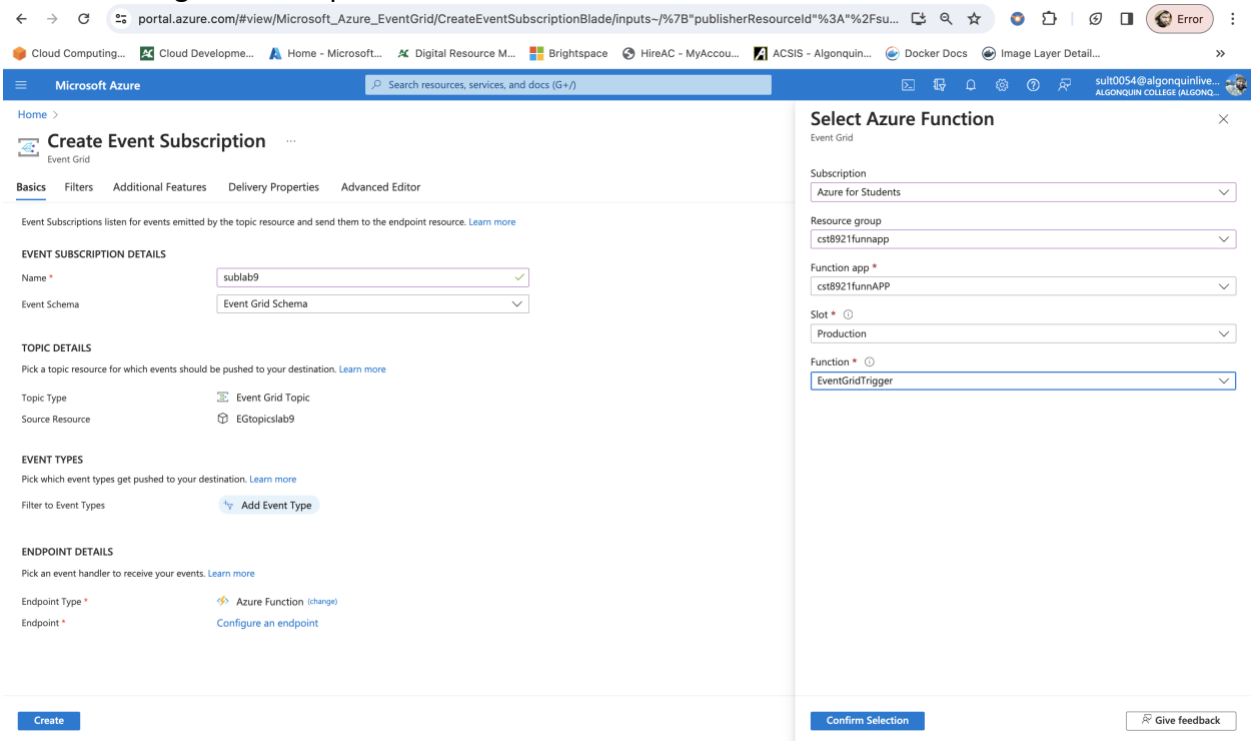
```
#r "Azure.Messaging.EventGrid"
#r "System.Memory.Data"
using Azure.Messaging.EventGrid;
using System;
public static void Run(EventGridEvent eventGridEvent, ILogger log)
{
   log.LogInformation(eventGridEvent.Data.ToString());
}
```



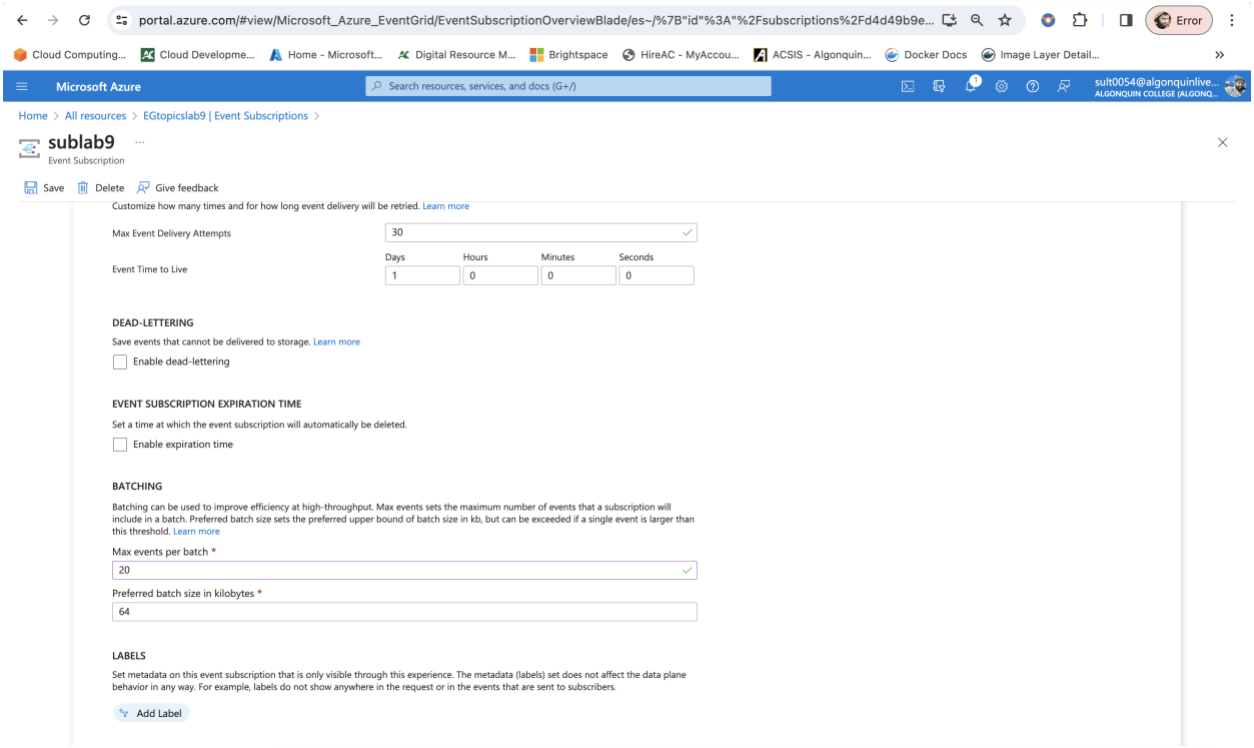
7. Select monitor to see the received event information. Create a custom topicin the events grid topic



8. Subscribe to the custom topic by selecting event subscription in event grid topic page
9. On the Create Event Subscription page, follow these steps:
10. Enter a name for the event subscription.
11. Select Azure Function for the Endpoint type.
12. Choose Configure an endpoint.



13. For the function endpoint, select the Azure Subscription and Resource Group your Function App is in and then select the Function App and function you created earlier. Select Confirm Selection.
14. On the Create Event Subscription page, switch to the Advanced Features tab, and set values for Max events per batch and Preferred batch size in kilobytes.
15. Batching can give you high-throughput. For Max events per batch, set maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kilo bytes, but can be exceeded if a single event is larger than this threshold.

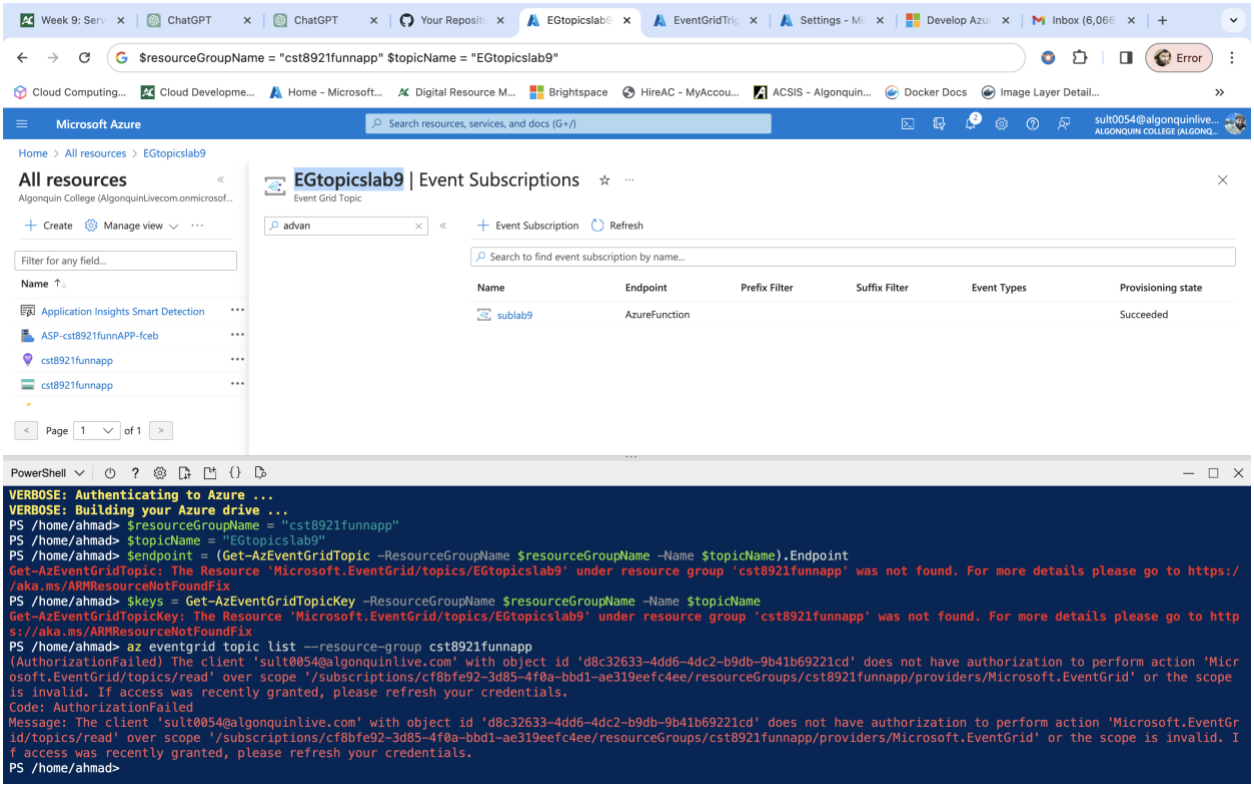


16. Send an event to your topic – Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. Use either Azure CLI or PowerShell to send a test event to your custom topic. Typically, an application or Azure service would send the event data.
17. In powershell, set the following variables and update the topic name and resource group name:

    $resourceGroupName = "cst8921funnapp"
    $topicName = "EGtopicslab9"

18. Run the following commands to get the endpoint and the keys for the topic:

    $endpoint = (Get-AzEventGridTopic -ResourceGroupName $resourceGroupName -Name $topicName).Endpoint
    $keys = Get-AzEventGridTopicKey -ResourceGroupName $resourceGroupName -Name $topicName



19. Prepare the event

```
$eventID = Get-Random 99999
#Date format should be SortableDateTimePattern (ISO 8601)
$eventDate = Get-Date -Format s
#Construct body using Hashtable
$htbody = @{
  id= $eventID
  eventType="recordInserted"
  subject="myapp/vehicles/motorcycles"
  eventTime= $eventDate
  data= @{
    make="Ducati"
    model="Monster"
  }
  dataVersion="1.0"
}
#Use ConvertTo-Json to convert event body from Hashtable to JSON Object
#Append square brackets to the converted JSON payload since they are expected in the event's JSON payload syntax
$body = "["+(ConvertTo-Json $htbody)+"]"
```

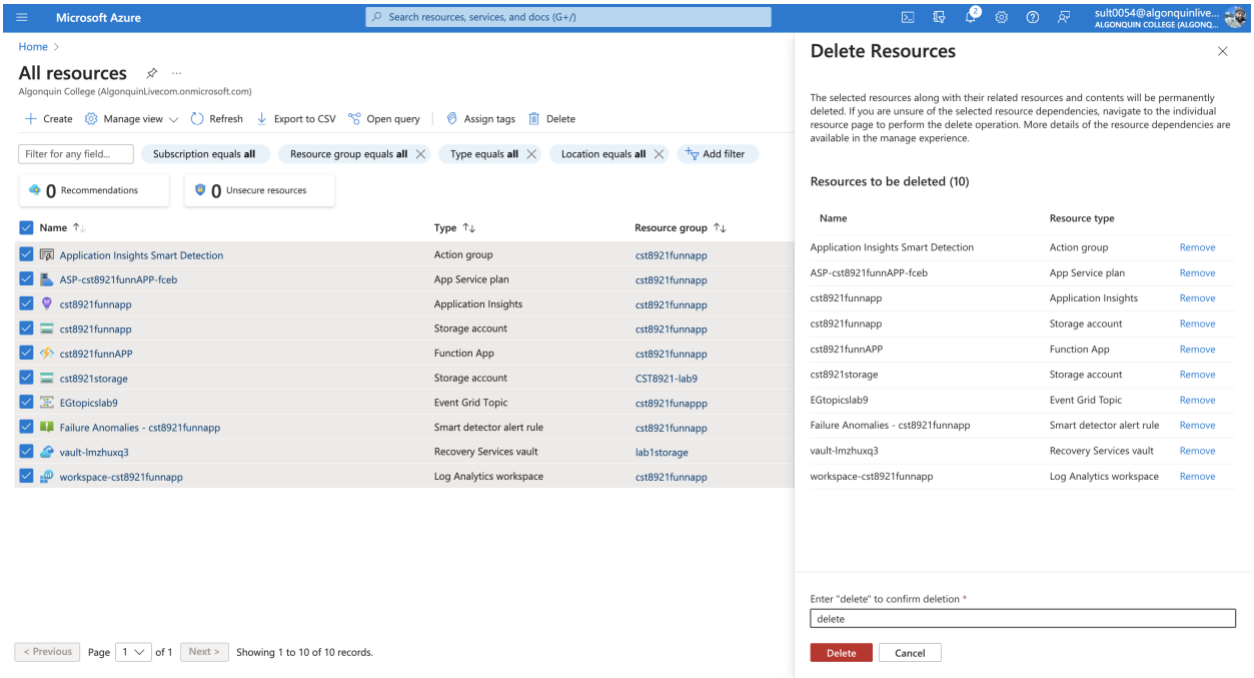
20. Use Invoke webrequest to send the event

    Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}

21. Verify that function received the event

    - On the monitor page of Azure function, see the invocation and details

22. Clean up the resources



Challenges

1. **Authorization Issues**: Encountering permission-related errors when attempting to access or manage Azure resources, highlighting the importance of proper access rights and subscription management. I tried many things and also tried changing the subscription but nothing worked so I think WE do not have enough permissions on our azure subscriptions. Could not complete last step.

2. **Resource Management**: Difficulties in locating or verifying the existence of specific Azure resources (e.g., Event Grid topics), which may stem from naming inconsistencies, resource deletion, or operating in the wrong Azure subscription context.

3. **Technical Configuration**: Complexity in configuring event subscriptions and endpoints, as well as crafting and sending events correctly, requiring careful attention to Azure documentation and syntax accuracy.

Conclusion

The lab effectively demonstrated the value of serverless computing in modern cloud architectures by allowing students to engage directly with Azure Functions and Event Grid. Despite facing challenges such as authorization issues and resource management complexities, students gained practical insights into developing event-driven applications without the overhead of server management. This hands-on experience not only solidified their understanding of serverless concepts but also prepared them for leveraging these technologies in real-world scenarios to enhance scalability and cost-efficiency.