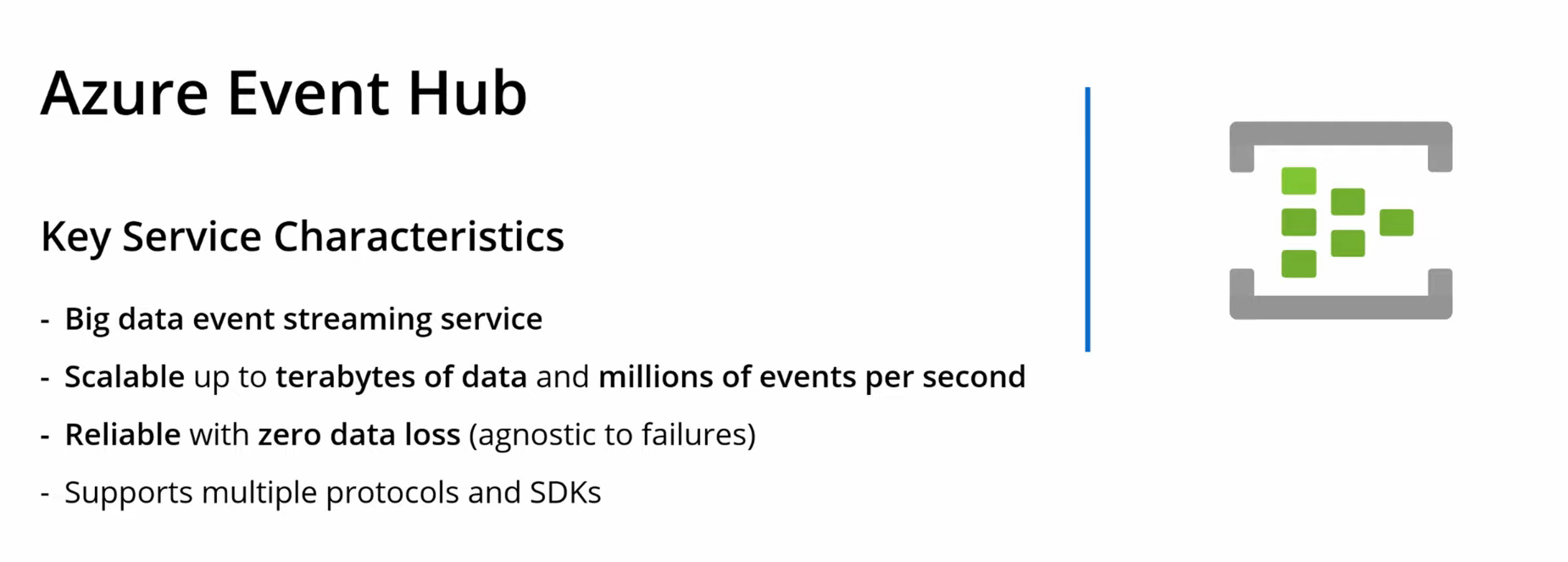
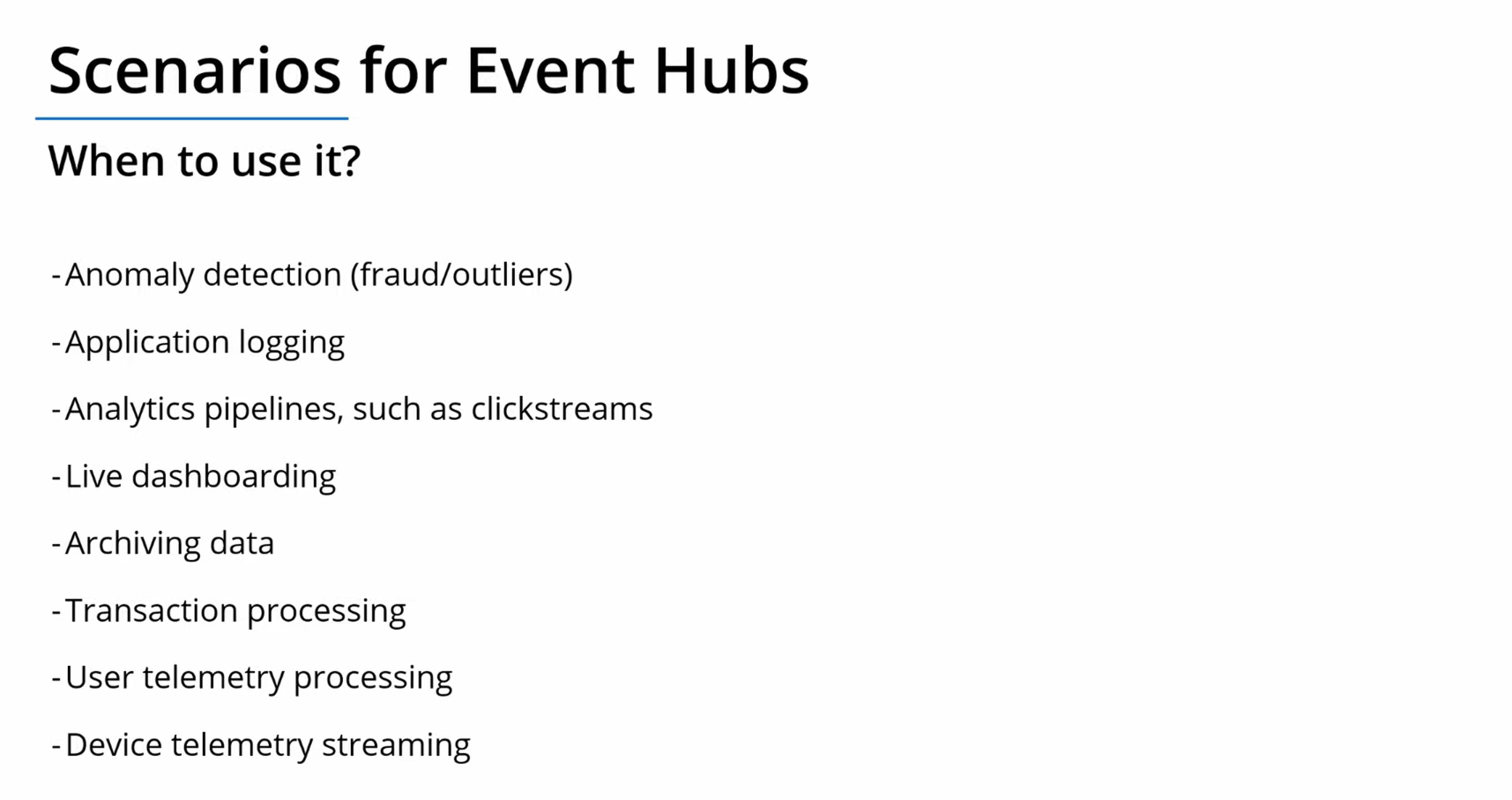
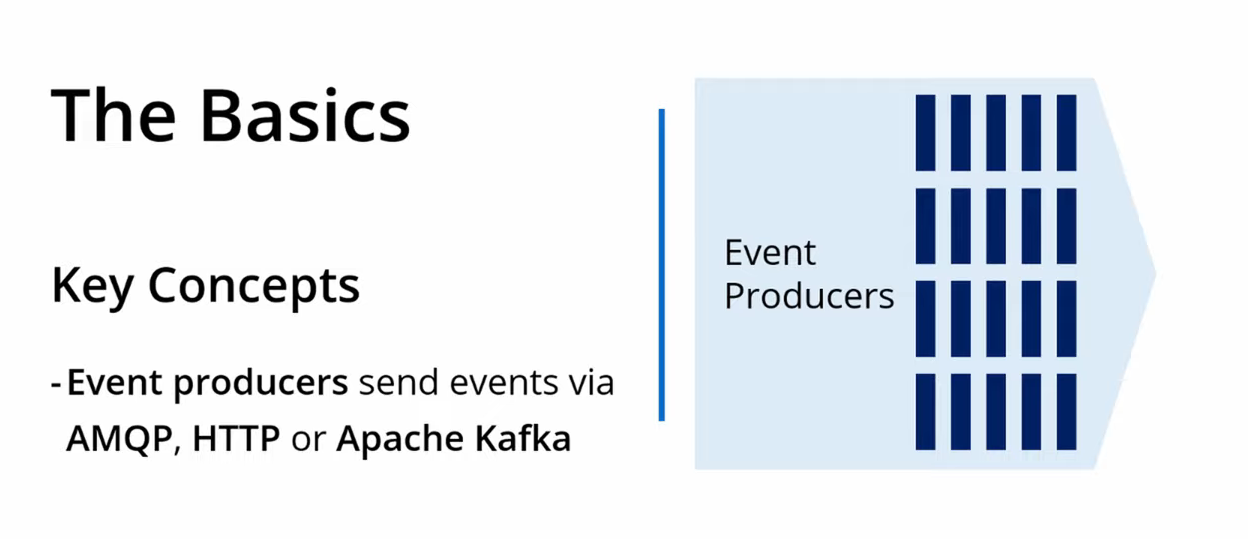
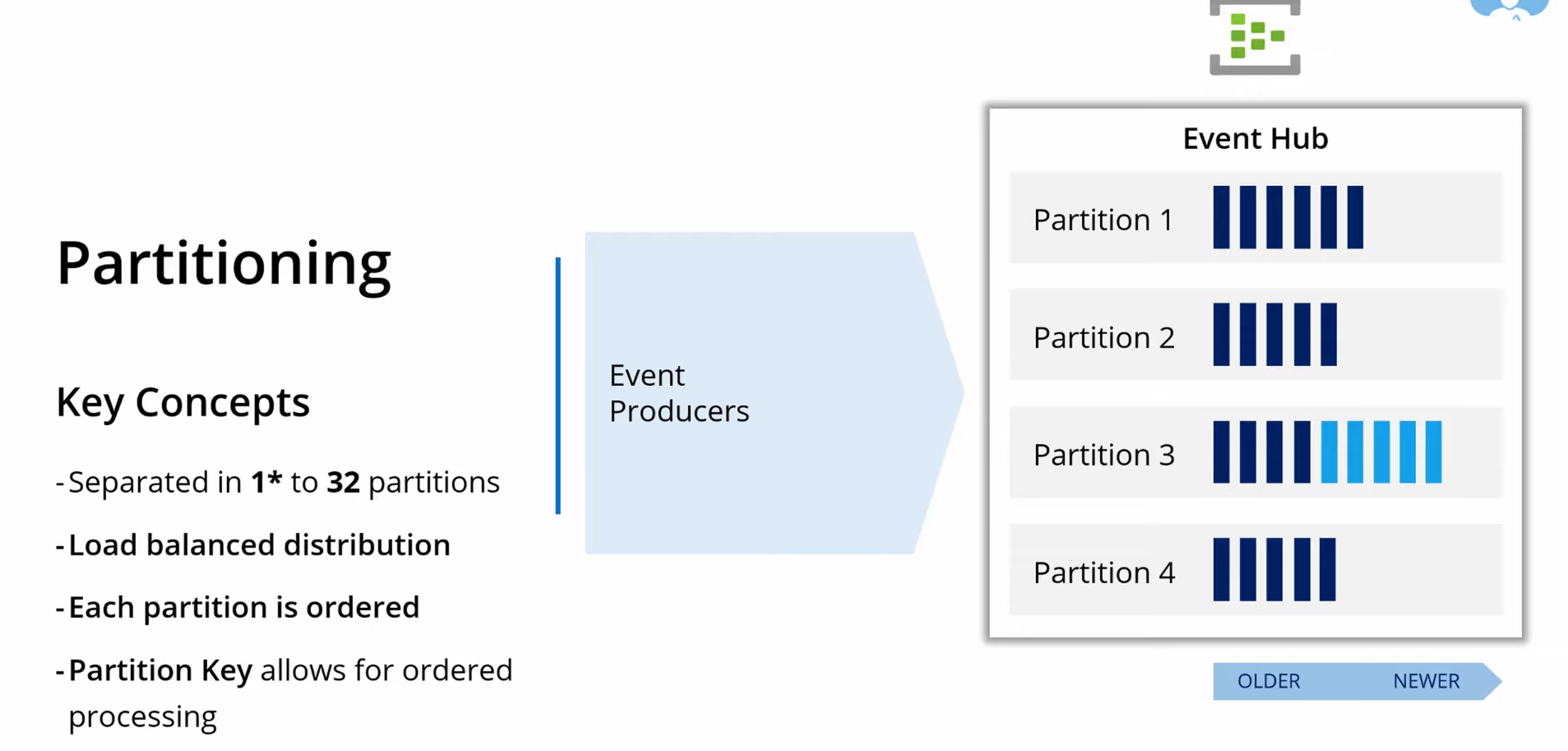
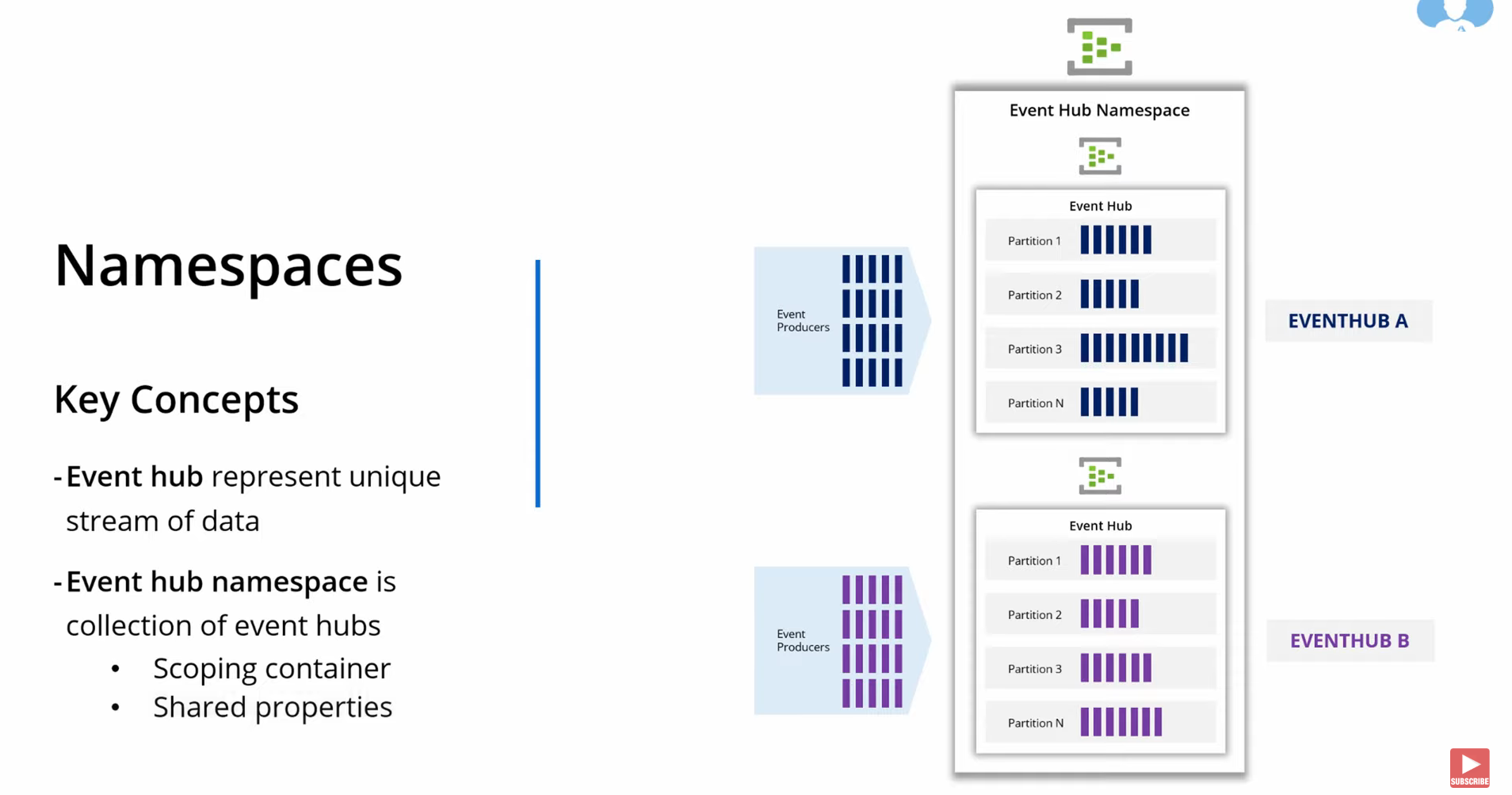
**Event Hub**







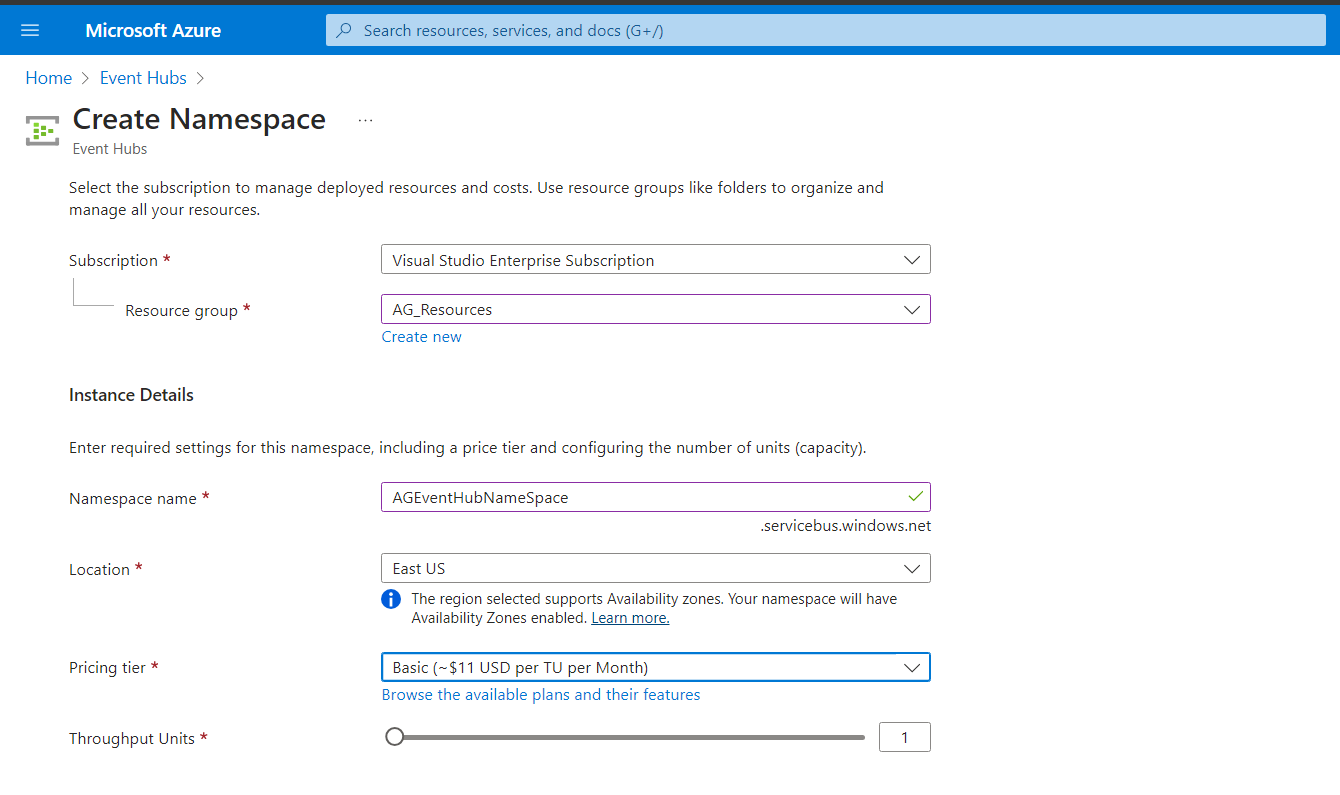




Namespace is actually the logical container for storing the eventhubs.

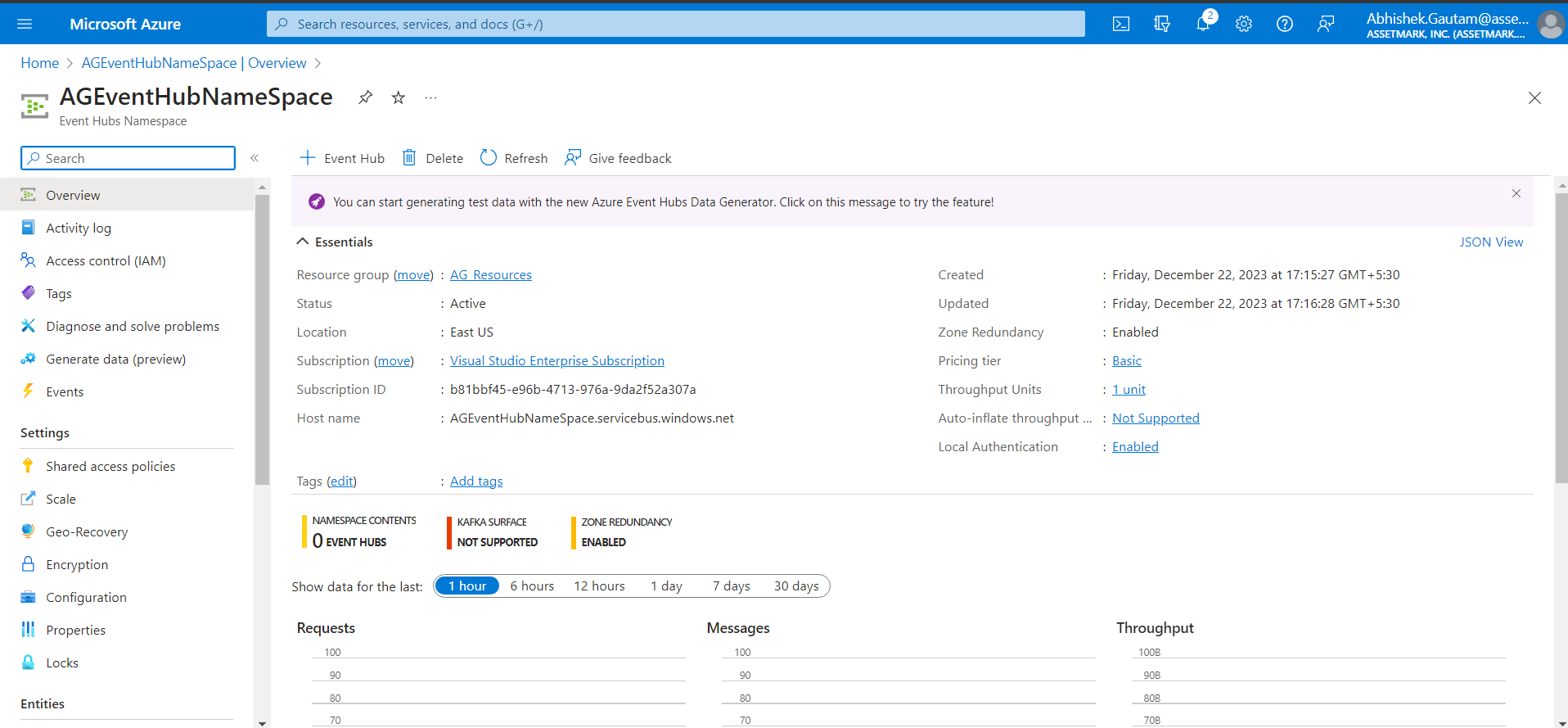
Go to Azure Portal and Search for Event Hub

When you press create you are actually creating a namespace first.

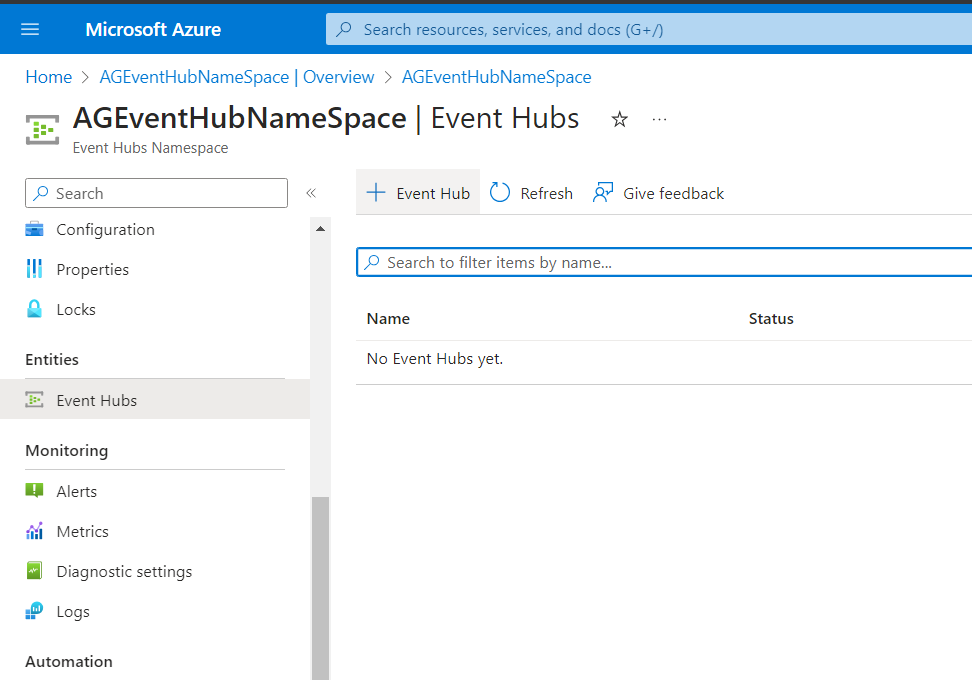


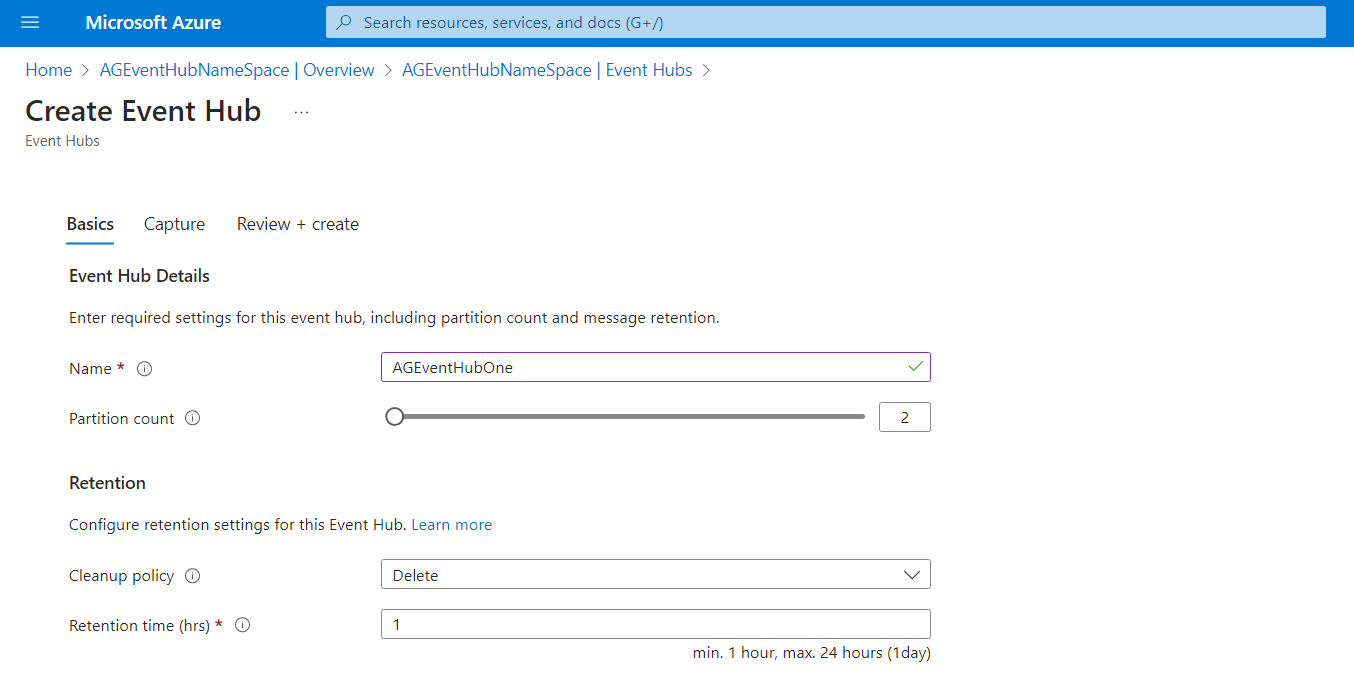
Throughput Units are actually the performance of our EventHub, you can give a value in-between 1 and 20.

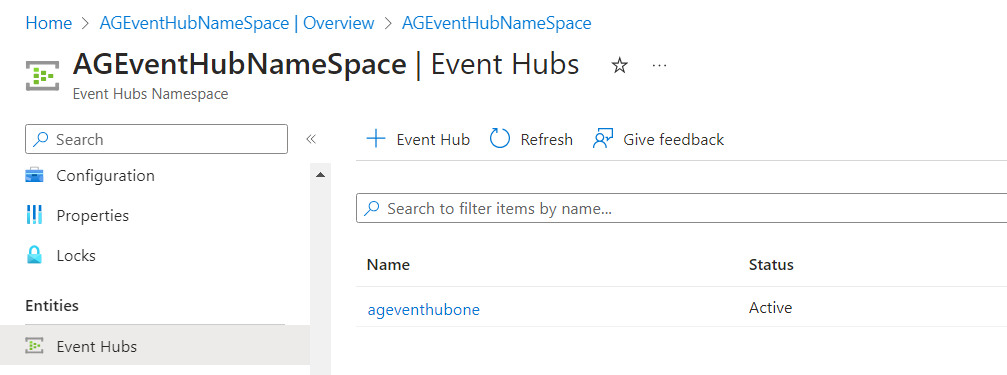
In few time our namespace is ready to use.



Now let’s create an Event Hub inside our namespace.







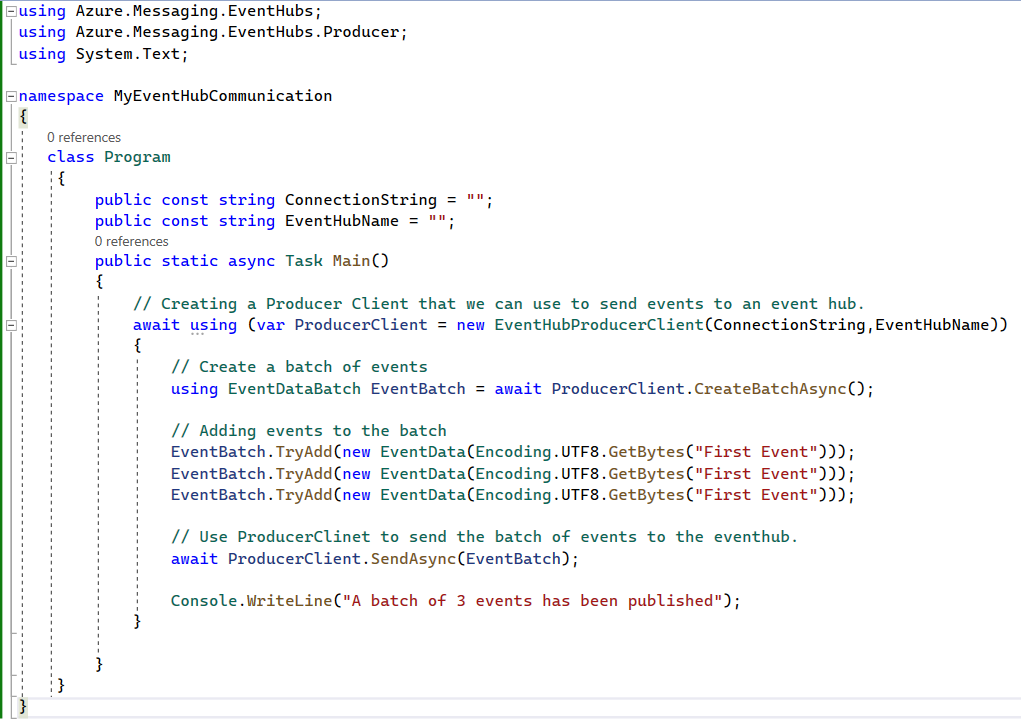
You can see our Event Hub inside our namespace, you can see that we can add more too.

Then we need a solution that will communicate with this event hub.

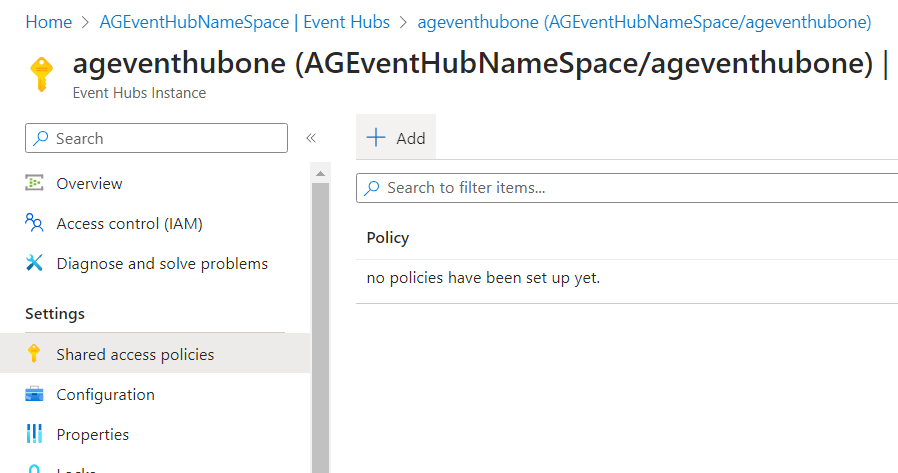
Add one console application inside your solution.

Add one packages in your solution:- Azure.Messaging.EventHubs

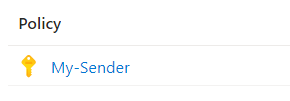
Add the following code in the Program.cs



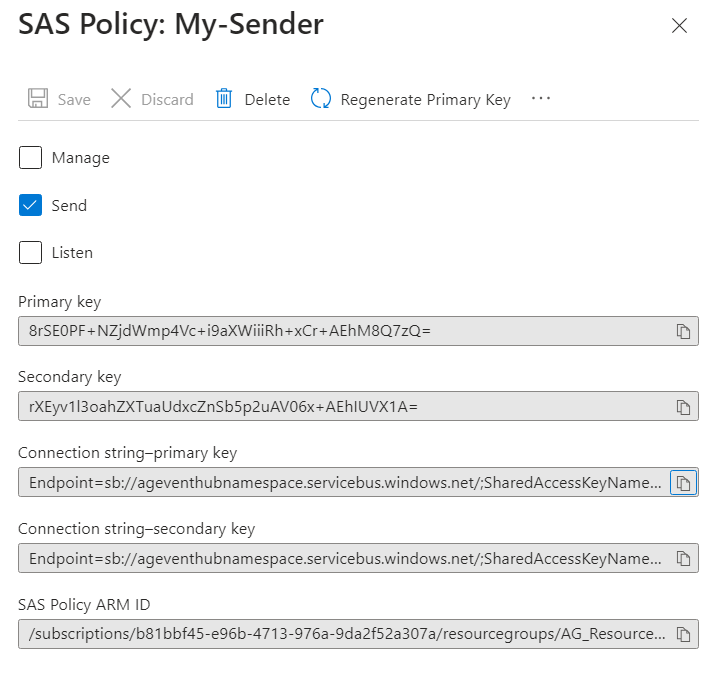
Inside the Connection String either you can add it from the Namespace’s Shared Access Policy or you can create your custom policy for your eventhub.



I have created one of Send type.

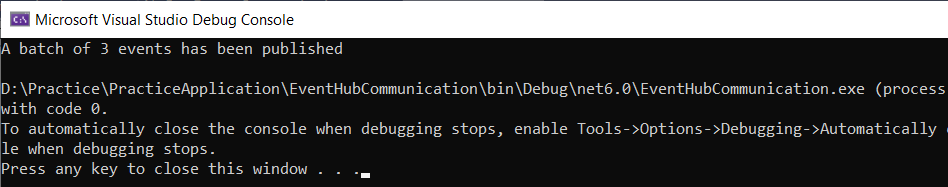


And when you click on this policy you can get the connection string



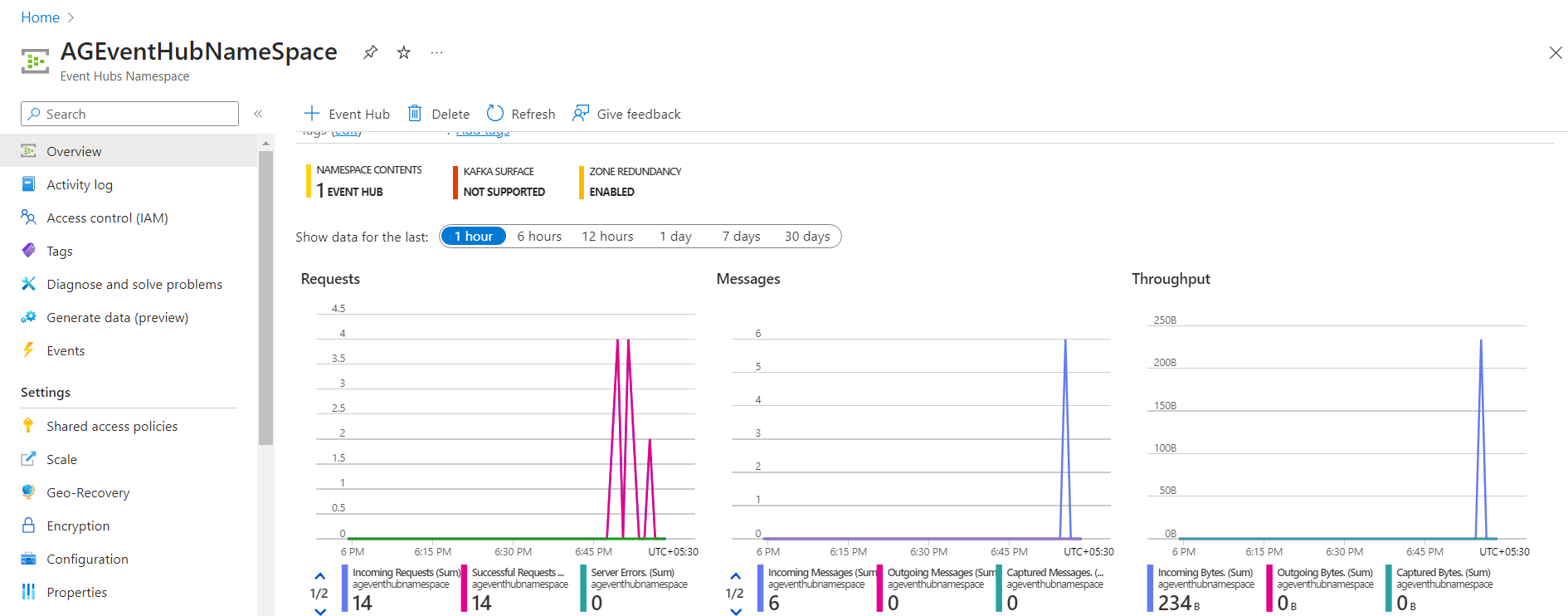
We can take the primary one.

Now save and build then run the application.

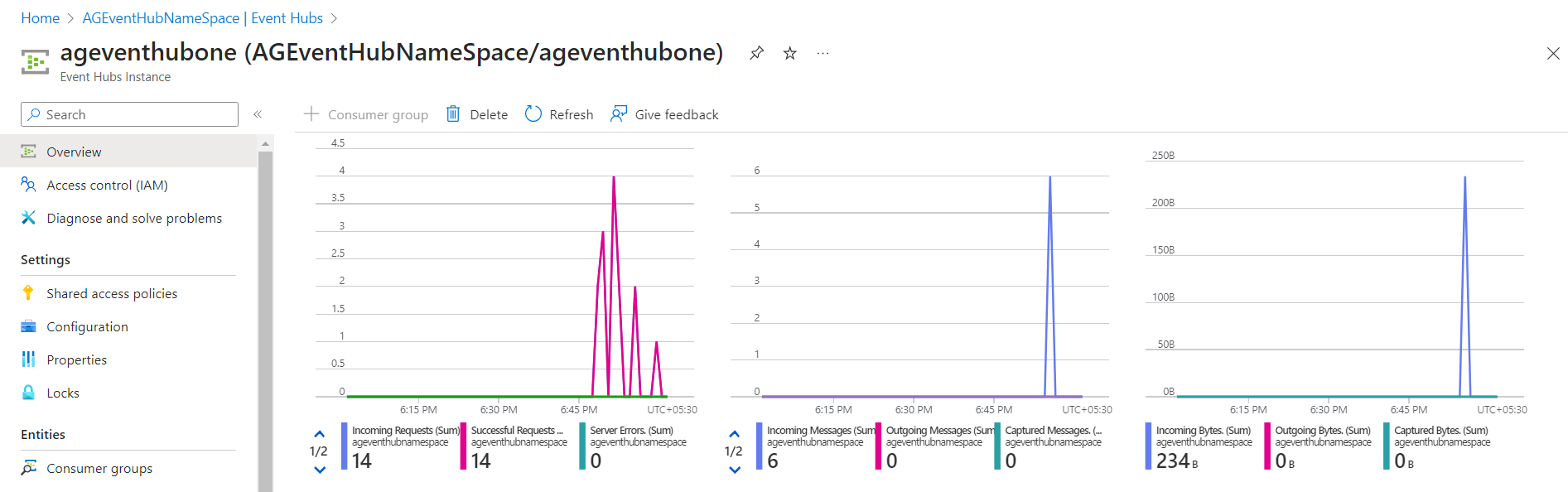


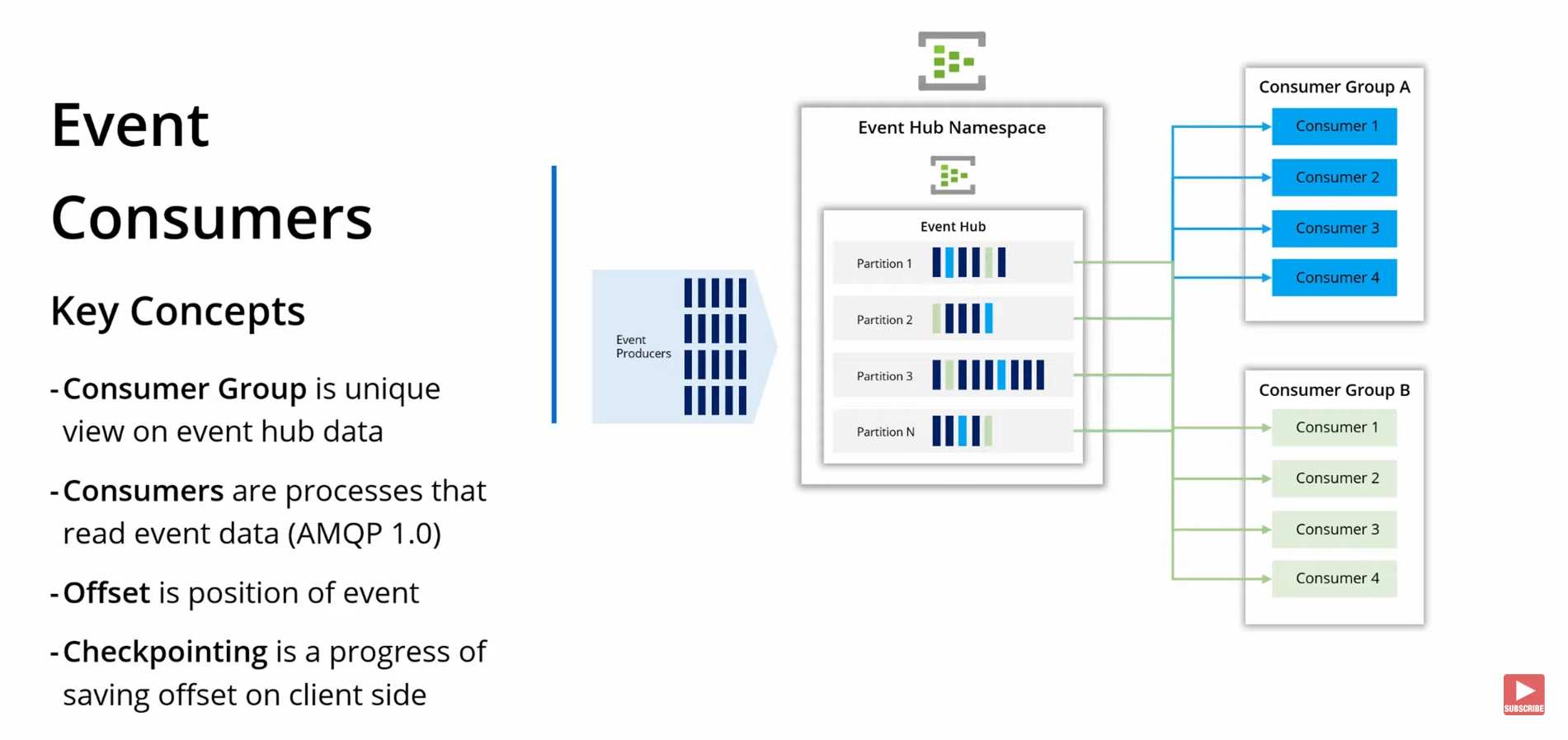
We can see that we have published the batch to our event hub.

On the namespace level if you press on overview you can see all the requests that we have pushed from the code.



We can also see the overview on the eventhubs window itself, this will filter the overview to the EventHub itself.





Offset is just the position where the consumer finishes the processing of the messages. If a consumer group dies, and restarts then it will not start from the beginning rather it will continue from where the offset is pointing.

We can have multiple consumer groups, and each consumer group will process the messages uniquely. All the consumer groups will have their offset.

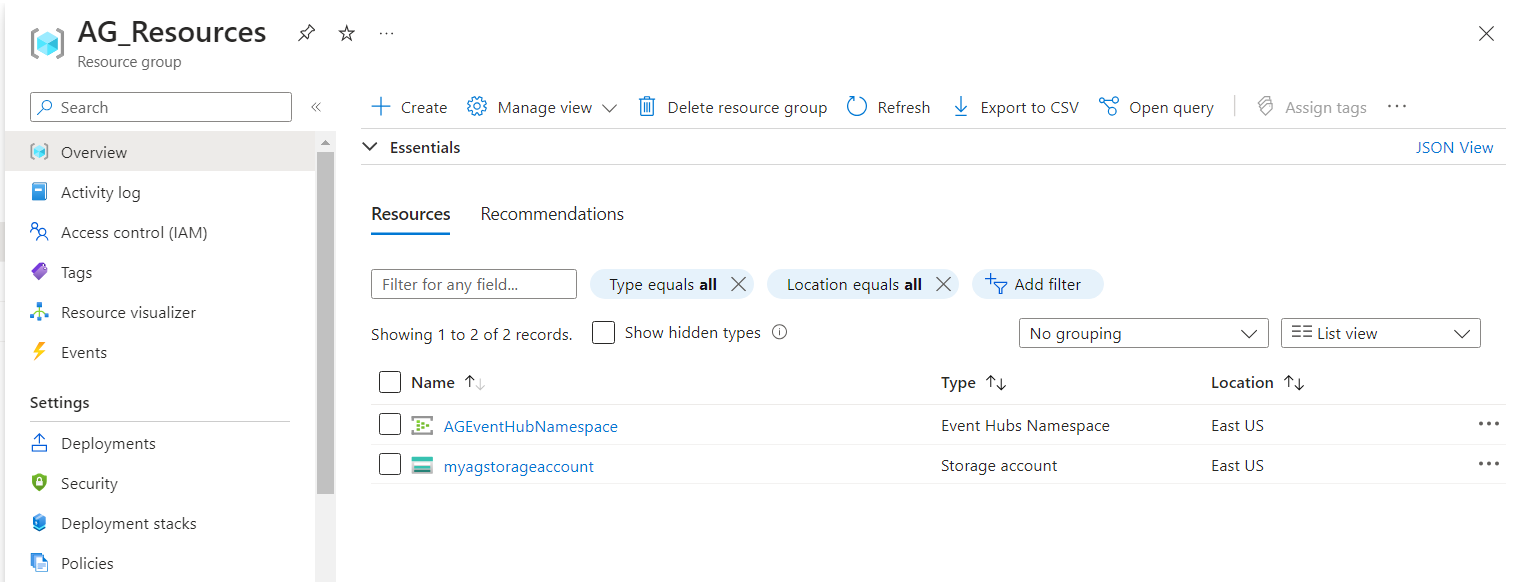
Now let’s create a new project inside our solution and name it as EventHubRecievingEvents, previous one was to Send the events.

Add the same nuget package inside it.

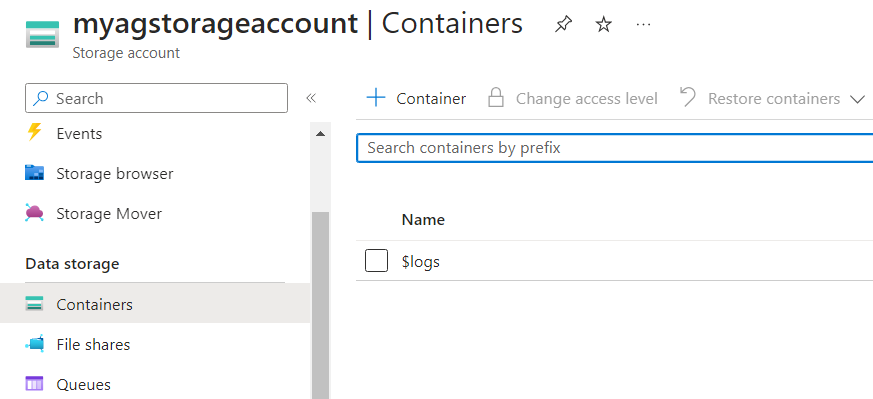
Here we also need to set the Checkpoint so we need to add one more nuget package.

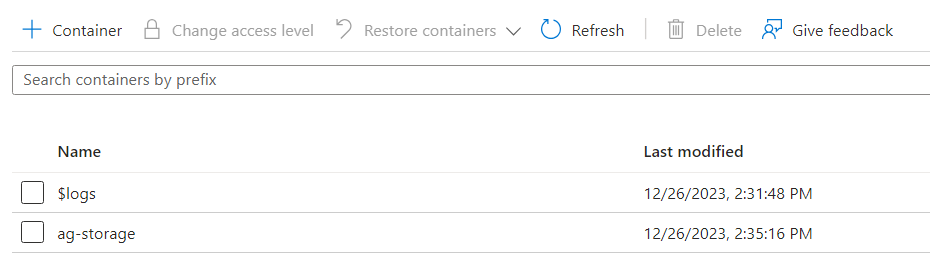


Create one storage account inside your Resource Group

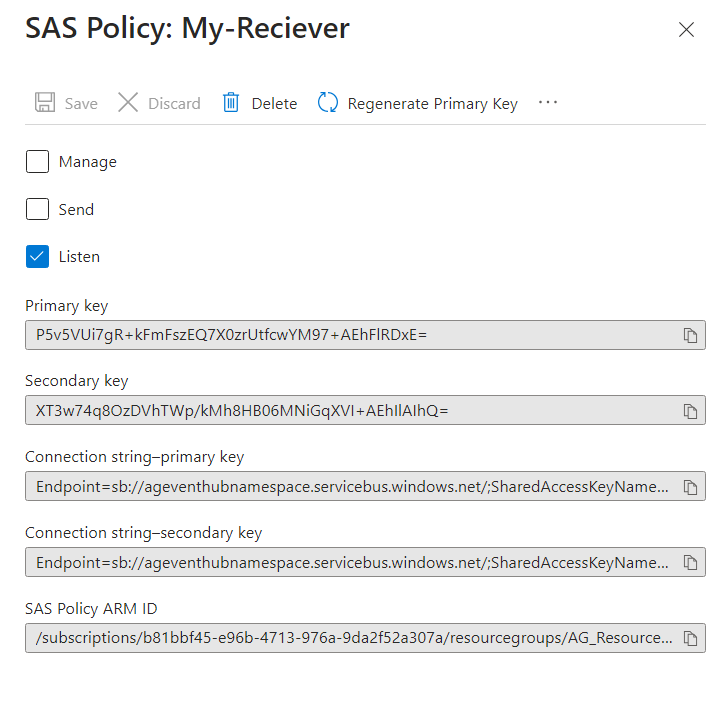


Then navigate to this Storage Account and then go to containers and then create one new container.

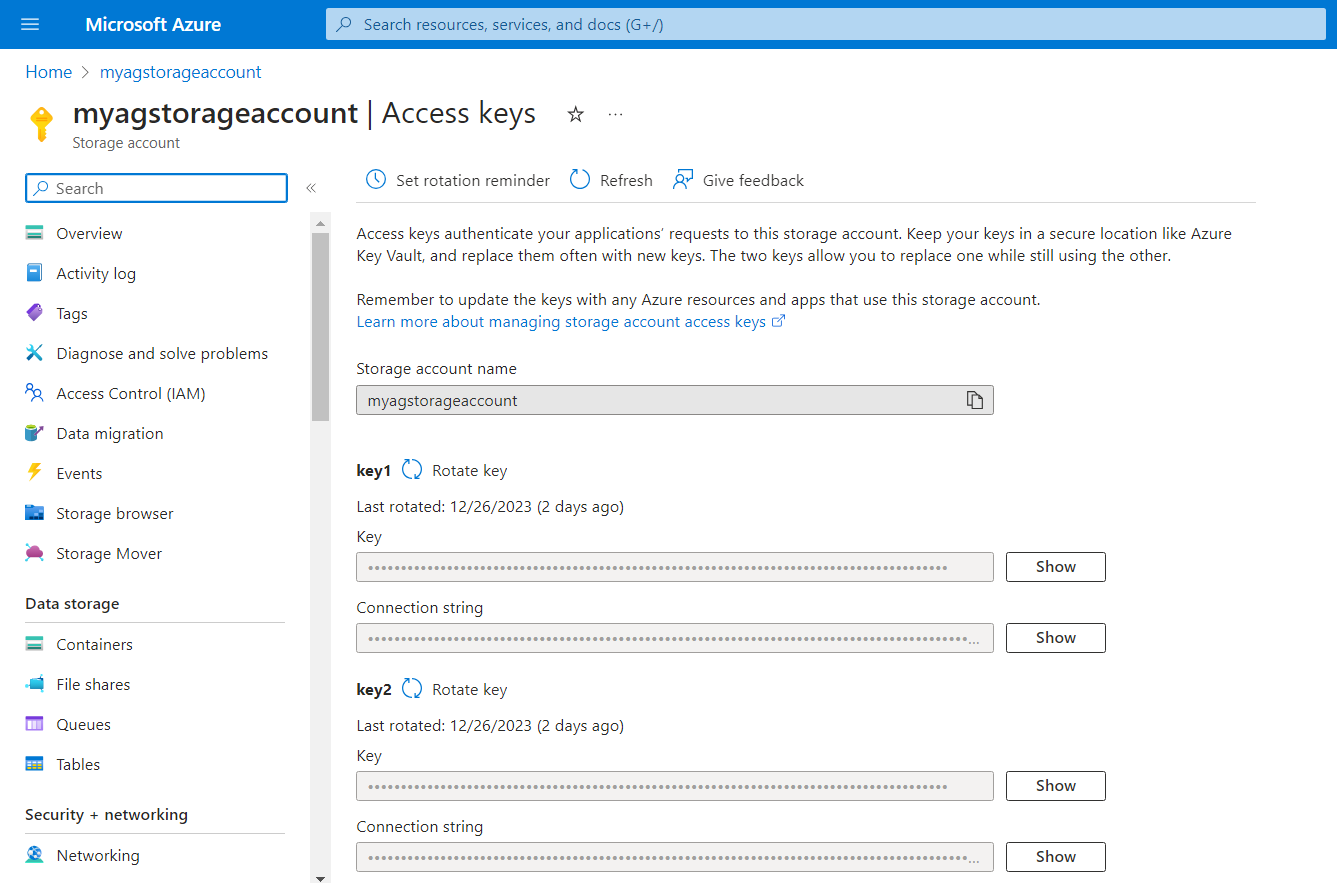




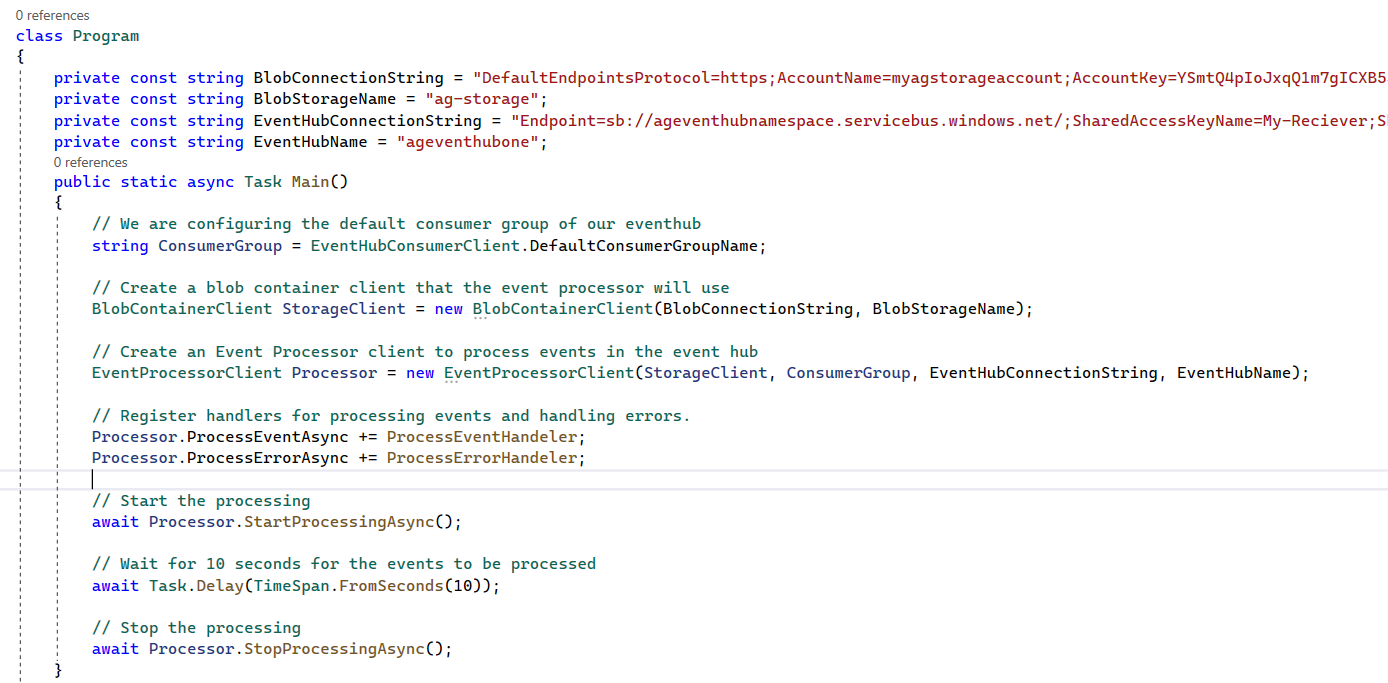
Inside your event hub add one more policy for listening

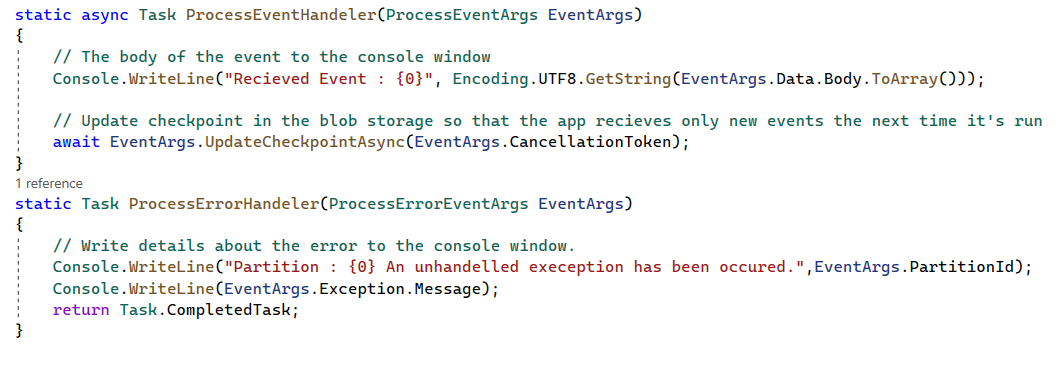


For the connection string of Blob storage, We need to give the connection string of the storage account and then we mention the name of the blob storage on the code side. In that way, it will connect to the respective blob storage inside our storage account.

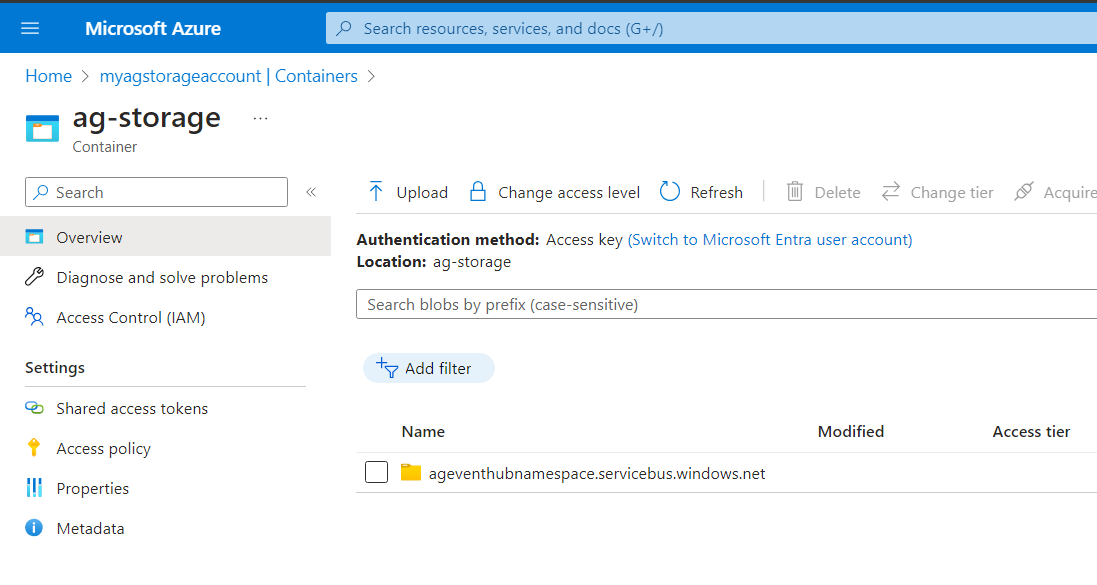


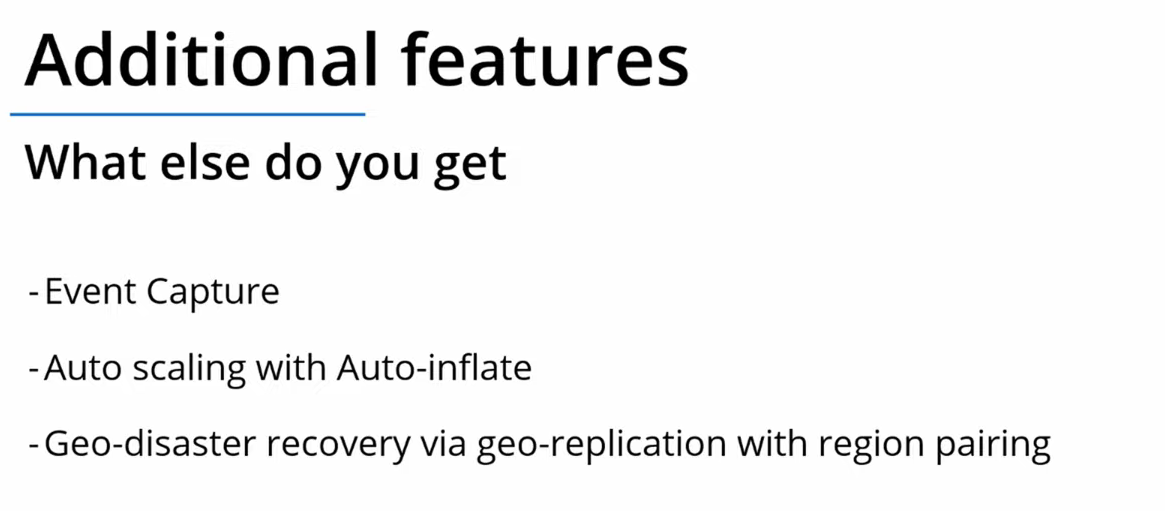
Now add all of these connections and names inside the C# Code for receiving the events.



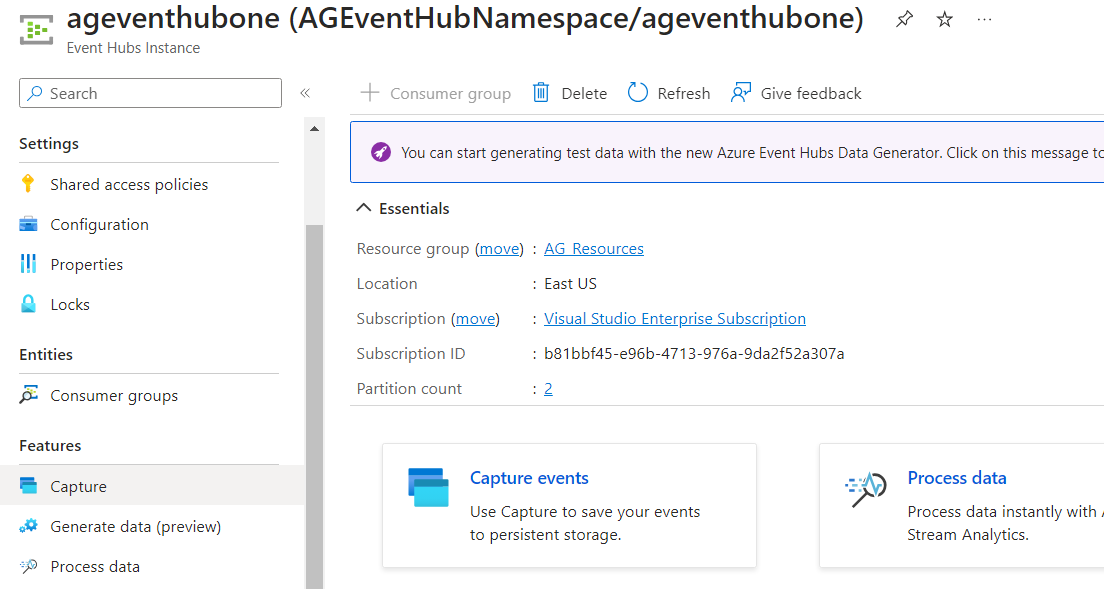


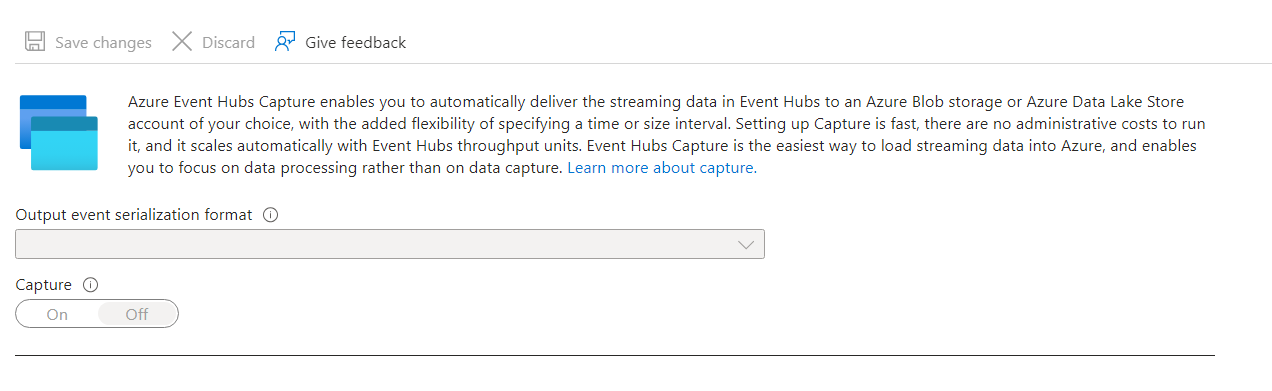
Now when you run this project (which is receiving the events) you will see a folder inside your respective blob storage.





Event Capturing - Go to your event hub, and on the left panel you can see the event capturing





Code :-

using Azure.Messaging.EventHubs;

using Azure.Messaging.EventHubs.Producer;

using System.Text;

// For sending the events

namespace MyEventHubCommunication

{

class Program

{

public const string ConnectionString = "Endpoint=sb://ageventhubnamespace.servicebus.windows.net/;SharedAccessKeyName=My-Sender;SharedAccessKey=1xiYEfPSSgoDZjs/4FrVo00JVzyHewP1A+AEhNjjOnM=;EntityPath=ageventhubone";

public const string EventHubName = "ageventhubone";

public static async Task Main()

{

// Creating a Producer Client that we can use to send events to an event hub.

await using (var ProducerClient = new EventHubProducerClient(ConnectionString,EventHubName))

{

// Create a batch of events

using EventDataBatch EventBatch = await ProducerClient.CreateBatchAsync();

// Adding events to the batch

EventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("First Event")));

EventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("Second Event")));

EventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("Third Event")));

// Use ProducerClinet to send the batch of events to the eventhub.

await ProducerClient.SendAsync(EventBatch);

Console.WriteLine("A batch of 3 events has been published");

}

}

}

}

using Azure.Messaging.EventHubs.Producer;

using Azure.Messaging.EventHubs;

using System.Text;

using Azure.Messaging.EventHubs.Processor;

using Azure.Messaging.EventHubs.Consumer;

using Azure.Storage.Blobs;

using Azure.Messaging.EventHubs.Primitives;

// For recieving the events

namespace EventHubRecievingEvents

{

class Program

{

private const string BlobConnectionString = "DefaultEndpointsProtocol=https;AccountName=myagstorageaccount;AccountKey=YSmtQ4pIoJxqQ1m7gICXB55wmnMq93JiLLn2WvvbtwTD63faQJQTtgM4xQ01hun72S34M5dBC8W5+AStNta/ZQ==;EndpointSuffix=core.windows.net";

private const string BlobStorageName = "ag-storage";

private const string EventHubConnectionString = "Endpoint=sb://ageventhubnamespace.servicebus.windows.net/;SharedAccessKeyName=My-Reciever;SharedAccessKey=P5v5VUi7gR+kFmFszEQ7X0zrUtfcwYM97+AEhFlRDxE=;EntityPath=ageventhubone";

private const string EventHubName = "ageventhubone";

public static async Task Main()

{

// We are configuring the default consumer group of our eventhub

string ConsumerGroup = EventHubConsumerClient.DefaultConsumerGroupName;

// Create a blob container client that the event processor will use

BlobContainerClient StorageClient = new BlobContainerClient(BlobConnectionString, BlobStorageName);

// Create an Event Processor client to process events in the event hub

EventProcessorClient Processor = new EventProcessorClient(StorageClient, ConsumerGroup, EventHubConnectionString, EventHubName);

// Register handlers for processing events and handling errors.

Processor.ProcessEventAsync += ProcessEventHandeler;

Processor.ProcessErrorAsync += ProcessErrorHandeler;

// Start the processing

await Processor.StartProcessingAsync();

// Wait for 10 seconds for the events to be processed

await Task.Delay(TimeSpan.FromSeconds(10));

// Stop the processing

await Processor.StopProcessingAsync();

}

static async Task ProcessEventHandeler(ProcessEventArgs EventArgs)

{

// The body of the event to the console window

Console.WriteLine("Recieved Event : {0}", Encoding.UTF8.GetString(EventArgs.Data.Body.ToArray()));

// Update checkpoint in the blob storage so that the app recieves only new events the next time it's run

await EventArgs.UpdateCheckpointAsync(EventArgs.CancellationToken);

}

static Task ProcessErrorHandeler(ProcessErrorEventArgs EventArgs)

{

// Write details about the error to the console window.

Console.WriteLine("Partition : {0} An unhandelled exeception has been occured.",EventArgs.PartitionId);

Console.WriteLine(EventArgs.Exception.Message);

return Task.CompletedTask;

}

}

}