LAB04 : Azure Cosmos DB Change Feed

In this lab you will use the Change Feed Processor Library and Azure Functions to implement three use cases for the Azure Cosmos DB Change Feed

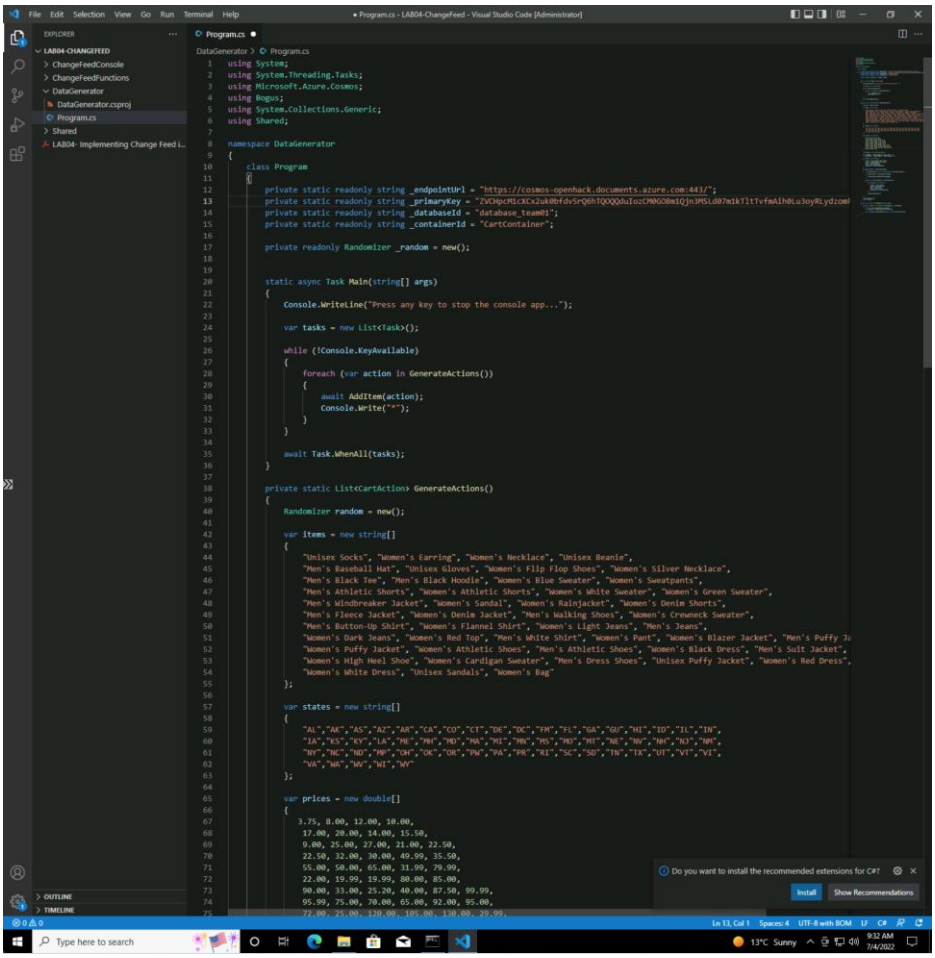
Build A .NET Console App to Generate Data

In order to simulate data flowing into our store, in the form of actions on an e-commerce website, we'll build a simple .NET Console App to generate and add documents to our Cosmos DB CartContainer

1. On your local machine, locate the Lab04 folder that will be used to contain the content of your .NET Core project. If you are completing this lab through Microsoft Hands-on Labs, the folder will be located at the path: **C:_COSMOSHACK_\labs\LAB04-ChangeFeed**
2. In the Lab04 folder, right-click the folder and select the **Open with Code** menu option. If you do not have this option in the context menu, you can run a command prompt in the lab04 directory directory and execute the code . command.

3. In the explorer pane on the left, locate the **DataGenerator** folder and expand it.

4. Select the program.cs link in the **Explorer** pane to open the file in the editor.



Log-in to the Azure Portal

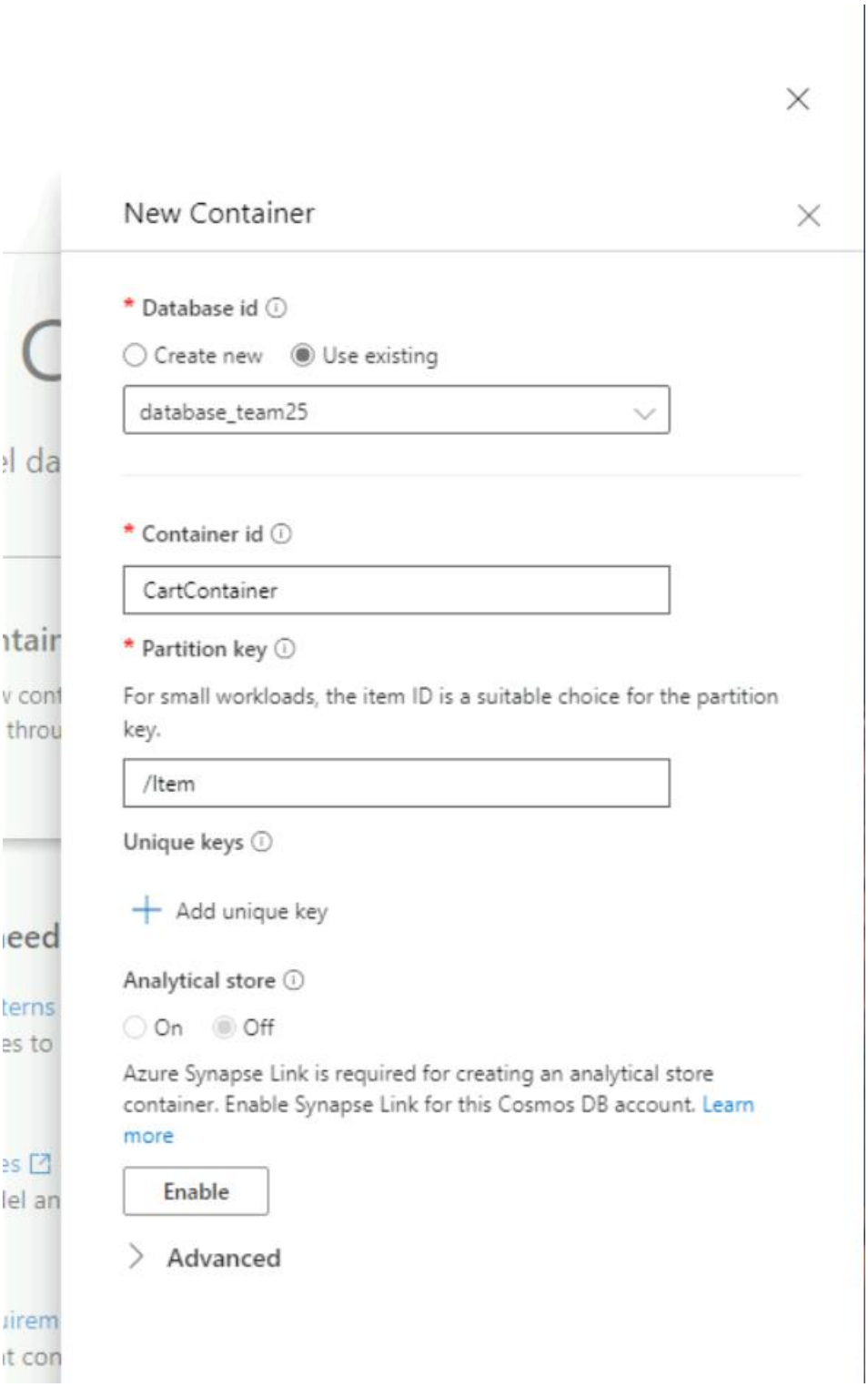
1. In a new window, sign in to the **Azure Portal** (https://portal.azure.com).
2. Once you have logged in, you may be prompted to start a tour of the Azure portal. You can safely skip this step.

Create a new container with following specification

Database id : database_teamxx (use your existing assigned DB number)

Container id : CartContainer

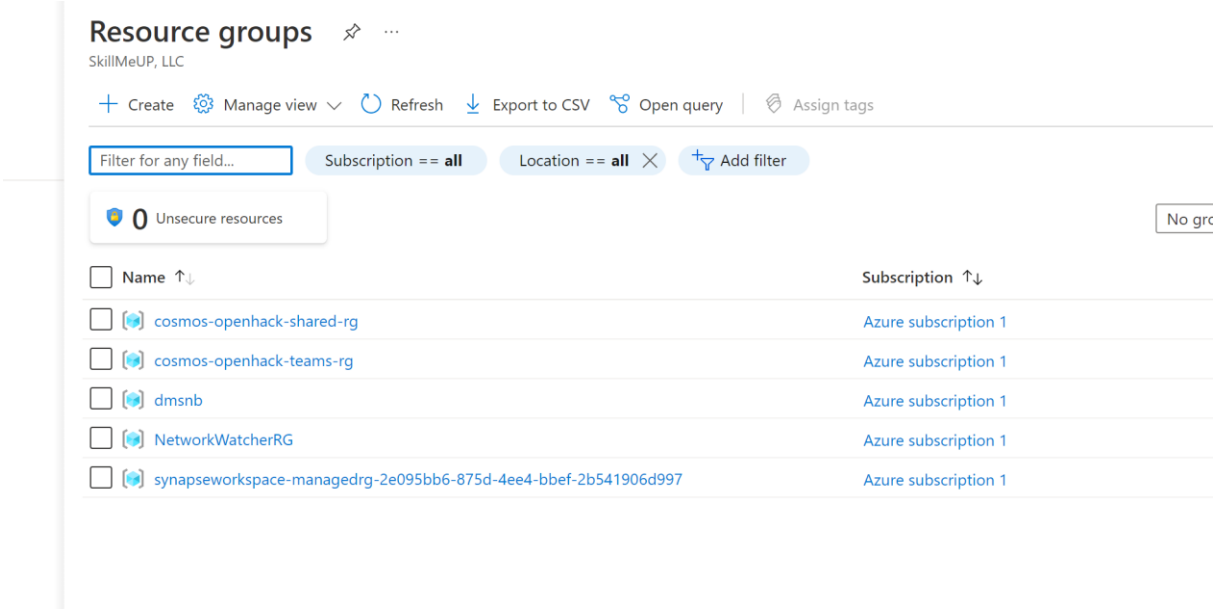
Partition key : Item



Retrieve Account Credentials

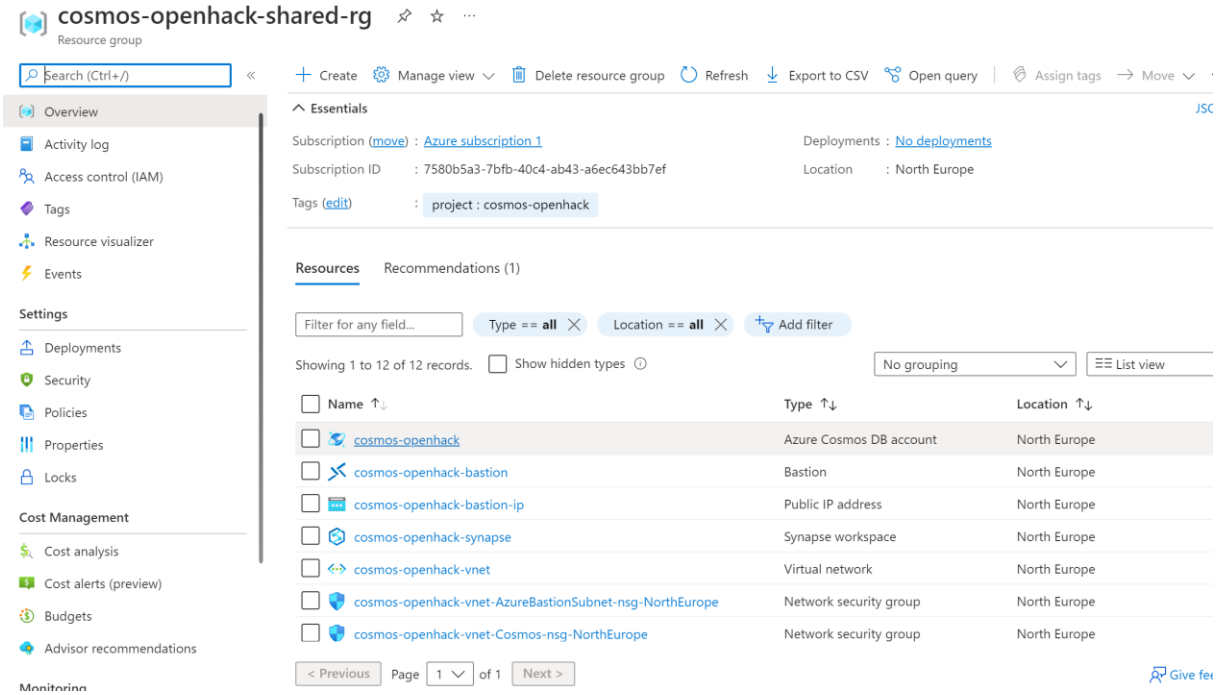
The .NET SDK requires credentials to connect to your Azure Cosmos DB account. You will collect and store these credentials for use throughout the lab.

1. On the left side of the portal, select the **Resource groups** link.

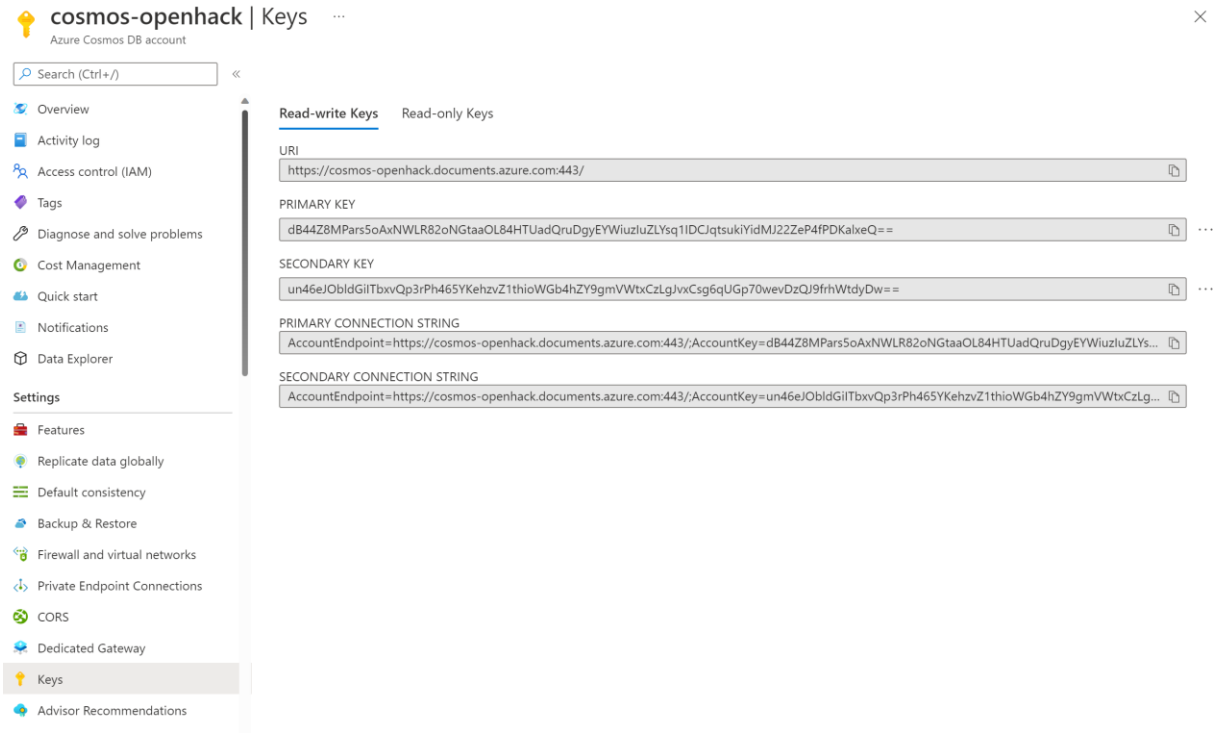


2. In the **Resource groups** blade, locate and select the **cosmos-openhack-shared-rg** *Resource Group*.

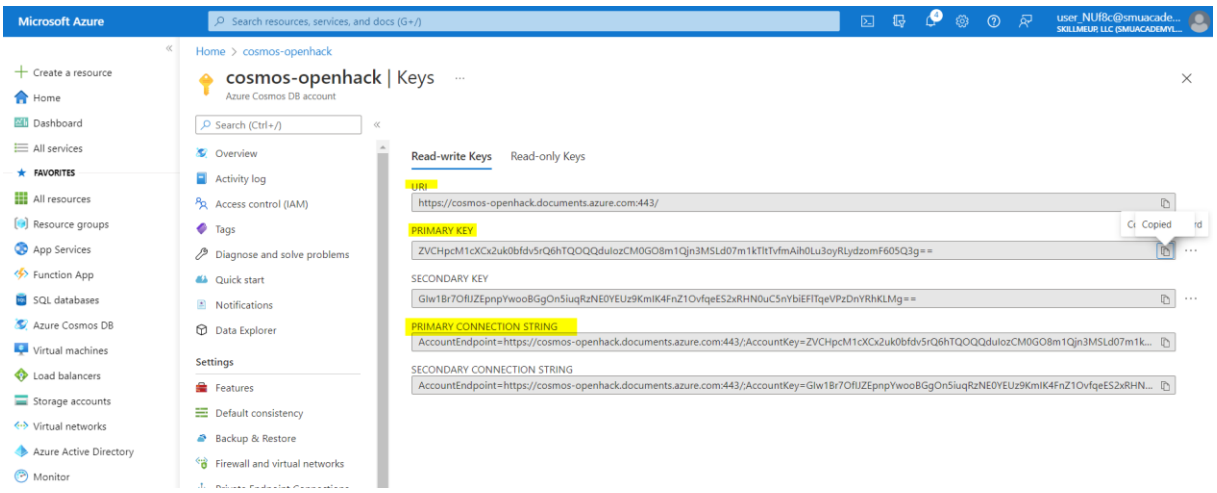
3. In the **cosmos-openhack** blade, select the **Azure Cosmos DB** account



4. In the **Azure Cosmos DB** blade, locate the **Settings** section and select the **Keys** link.



5. In the **Keys** pane, record the values in the **CONNECTION STRING**, **URI** and **PRIMARY KEY** fields. You will use these values later in this lab.



5- For the _endpointUrl variable in the code file, replace the placeholder value with the **URI** value and for the _primaryKey variable, replace the placeholder value with the **PRIMARY KEY** value from your Azure Cosmos DB account

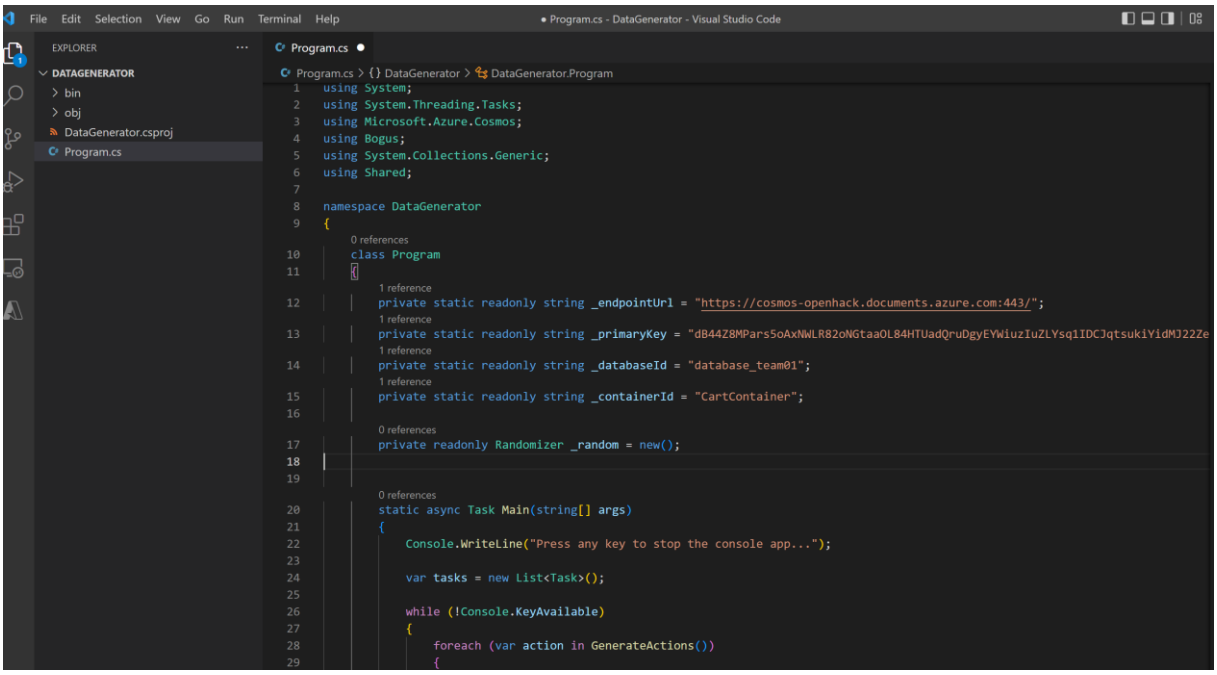
- For example, if your **url** is https://cosmosopenhack.documents.azure.com:443/, your new variable assignment will look like this:

private static readonly string _endpointUrl = "https:// cosmosopenhack.documents.azure.com:443/";

- For example, if your **primary key** is elzirrKCnXlacvh1CRAnQdYVbVLspmYHQyYrhx0PltHi8wn5lHVHFnd1Xm3ad5cn4TUcH4U0MSeHsVykkFPHpQ==, your new variable assignment will look like this:

```
private static readonly string _primaryKey =
"elzirrKCnXlacvh1CRAnQdYVbVLspmYHQyYrhx0PltHi8wn5lHVHFnd1Xm3ad5cn4TUcH4U0MSeHsVykkFPHpQ==";
```

Modifiy the connexion variables in the Program.cs file



Create Function to Add Documents to Cosmos DB

The key functionality of the console application is to add documents to our Cosmos DB to simulate activity on our e-commerce website. Here, you'll create a data definition for these documents and define a function to add them

1. Within the program.cs file in the **DataGenerator** folder, locate the AddItem() method. The purpose of this method is to add an instance of **CartAction** to our Cosmos DB Container.

```
private static async Task AddItem(CartAction item)
{
    using (var client = new CosmosClient(_endpointUrl, _primaryKey))
    {
        var db = client.GetDatabase(_databaseId);
        var container = db.GetContainer(_containerId);

        await container.CreateItemAsync(item, new PartitionKey(item.Item));
    }
}
```

Create a Function to Generate Random Shopping Data

1. Within the Program.cs file in the **DataGenerator** folder, locate the GenerateActions() method. The purpose of this method is to create randomized **CartAction** objects that you'll consume using the Cosmos DB change feed.

```
private static List<CartAction> GenerateActions()
{
    Randomizer random = new();

    var items = new string[]
    {
        "Unisex Socks", "Women's Earring", "Women's Necklace", "Unisex Beanie",
        "Men's Baseball Hat", "Unisex Gloves", "Women's Flip Flop Shoes", "Women's Silver Necklace",
        "Men's Black Tee", "Men's Black Hoodie", "Women's Blue Sweater", "Women's Sweatpants",
        "Men's Athletic Shorts", "Women's Athletic Shorts", "Women's White Sweater", "Women's Green Sweater",
        "Men's Windbreaker Jacket", "Women's Sandal", "Women's Rainjacket", "Women's Denim Shorts",
        "Men's Fleece Jacket", "Women's Denim Jacket", "Men's Walking Shoes", "Women's Crewneck Sweater",
        "Men's Button-Up Shirt", "Women's Flannel Shirt", "Women's Light Jeans", "Men's Jeans",
        "Women's Dark Jeans", "Women's Red Top", "Men's White Shirt", "Women's Pant", "Women's Blazer Jacket", "Men's Puffy Ja
        "Women's Puffy Jacket", "Women's Athletic Shoes", "Men's Athletic Shoes", "Women's Black Dress", "Men's Suit Jacket",
        "Women's High Heel Shoe", "Women's Cardigan Sweater", "Men's Dress Shoes", "Unisex Puffy Jacket", "Women's Red Dress",
        "Women's White Dress", "Unisex Sandals", "Women's Bag"
    };

    var states = new string[]
    {
        "AL","AK","AS","AZ","AR","CA","CO","CT","DE","DC","FM","FL","GA","GU","HI","ID","IL","IN",
        "IA","KS","KY","LA","ME","MH","MD","MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ","NM",
        "NY","NC","ND","MP","OH","OK","OR","PW","PA","PR","RI","SC","SD","TN","TX","UT","VT","VI",
        "VA","WA","WV","WI","WY"
    };

    var prices = new double[]
    {
       3.75, 8.00, 12.00, 10.00,
        17.00, 20.00, 14.00, 15.50,
        9.00, 25.00, 27.00, 21.00, 22.50,
        22.50, 32.00, 30.00, 49.99, 35.50,
        55.00, 50.00, 65.00, 31.99, 79.99,
        22.00, 19.99, 19.99, 80.00, 85.00,
        90.00, 33.00, 25.20, 40.00, 87.50, 99.99,
        95.99, 75.00, 70.00, 65.00, 92.00, 95.00,
        72.00, 25.00, 120.00, 105.00, 130.00, 29.99,
        84.99, 12.00, 37.50
    };
```

Run the Console App and Verify Functionality

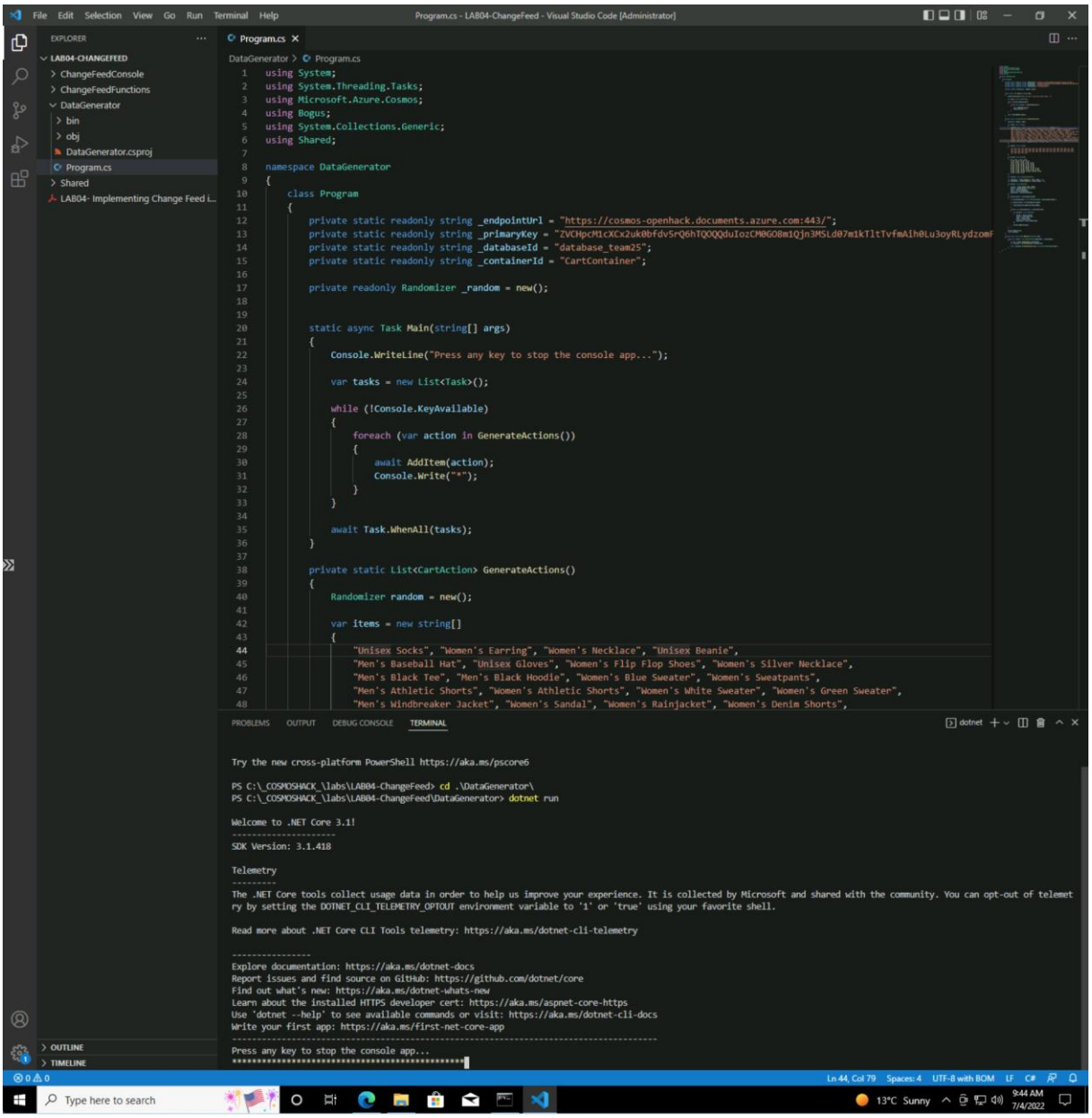
You're ready to run the console app, and in this step you'll take a look at your Cosmos DB account to ensure test data is being written as expected.

1. Open a terminal window

2. In the terminal pane, enter and execute the following command to run your console app:

3. cd DataGenerator
4. dotnet run

5. After a brief build process, you should begin to see the asterisks being printed as data is being generated and written to Cosmos DB.

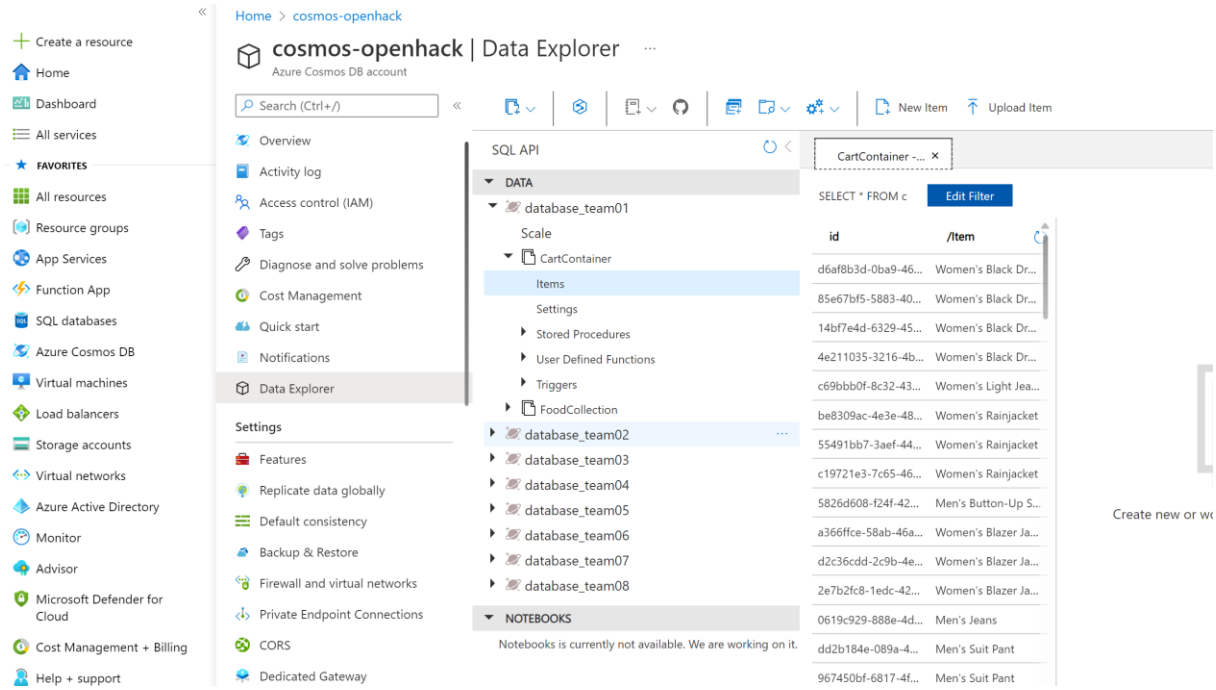


6. Let the console app run for a minute or two and then stop it by pressing any key in the console.

7. Switch to the Azure Portal and your Cosmos DB Account.

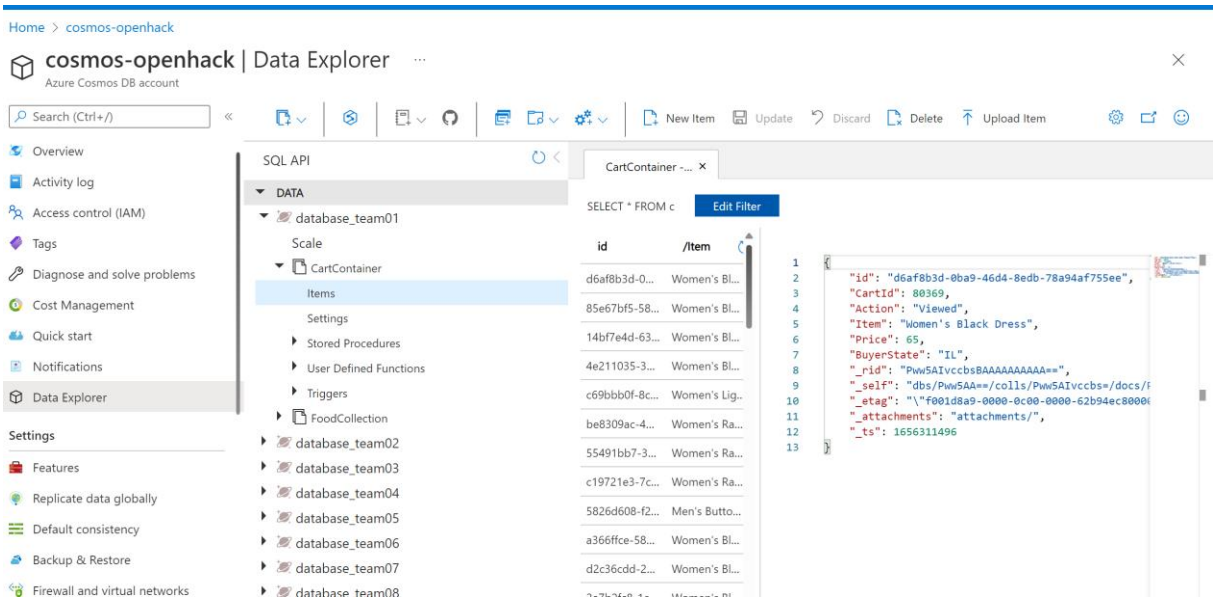
8. From within the **Azure Cosmos DB** blade, select the **Data Explorer** tab on the left.

cd



9. Expand the **database_teamXX** database, then the **CartContainer** and select **Items**. You should see something like the following screenshot.

   Note your data will be slightly different since it is random, the important thing is that there is data here at all



Consume Cosmos DB Change Feed via the Change Feed Processor

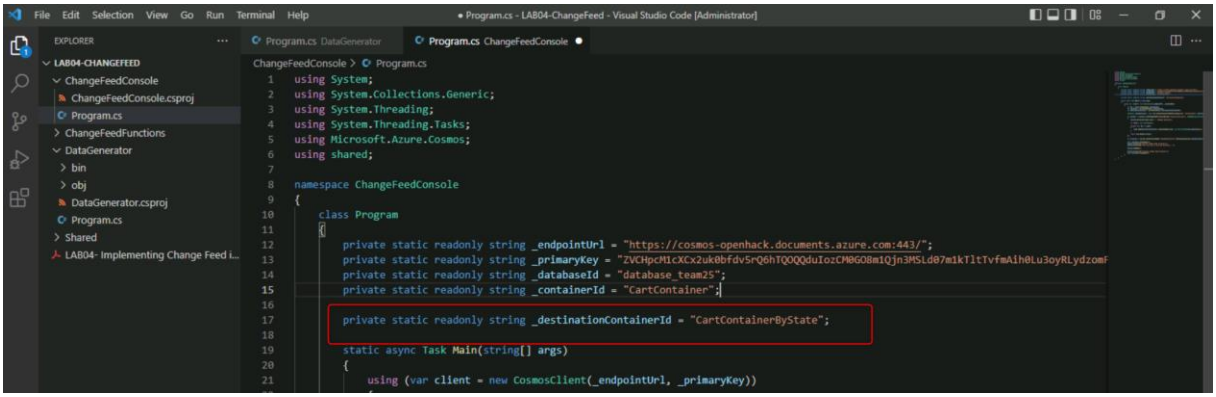
The two main options for consuming the Cosmos DB change feed are Azure Functions and the Change Feed Processor library. We'll start with the Change Feed Processor via a simple console application

Connect to the Cosmos DB Change Feed

The first use case we'll explore for Cosmos DB Change Feed is Live Migration. A common concern when designing a Cosmos DB container is proper selection of a partition key. You'll recall that we created our CartContainer with a partition key of /Item. What if we find out later this key is wrong? Or what if writes work better with /Item while reads work better with /BuyerState as the partition key? We can avoid analysis paralysis by using Cosmos DB Change Feed to migrate our data in real time to a second container with a different partition key!

1. Switch back to Visual Studio Code

2. Select the Program.cs link under the **ChangeFeedConsole** folder in the **Explorer** pane to open the file in the editor.
3. For the _endpointUrl variable, replace the placeholder value with the **URI** value and for the _primaryKey variable, replace the placeholder value with the **PRIMARY KEY** value from your Azure Cosmos DB account and **_databasId** with your database name
4. Notice the container configuration value at the top of the program.cs file, for the name of the destination container, following _containerId:
   private static readonly string _destinationContainerId = "CartContainerByState";



   In this case we are going to migrate our data to another container within the same database. The same ideas apply even if we wanted to migrate our data to another database entirely.

5. In order to consume the change feed we make use of a **Lease Container**. Add the following lines of code in place of //todo: Add lab code here to create the lease container:

```
8   namespace ChangeFeedConsole
9   {
10      class Program
11      {
12          private static readonly string _endpointUrl = "https://cosmos-openhack.documents.azure.com:443/";
13          private static readonly string _primaryKey = "primarykey";
14          private static readonly string _databaseId = "database_teamxx";
15          private static readonly string _containerId = "CartContainer";
16
17          private static readonly string _destinationContainerId = "CartContainerByState";
18
19         static async Task Main(string[] args)
20          {
21              using (var client = new CosmosClient(_endpointUrl, _primaryKey))
22              {
23                  var db = client.GetDatabase(_databaseId);
24                  Container container = db.GetContainer(_containerId);
25                  Container destContainer = db.GetContainer(_destinationContainerId);
26
27                 //TODO
28
29
30                  Console.WriteLine("Started Change Feed Processor");
31                  Console.WriteLine("Press any key to stop the processor...");
32
33                  Console.ReadKey();
34
35                  Console.WriteLine("Stopping Change Feed Processor");
36                  //TODO
37              }
38          }
39      }
40  }
```

```
ContainerProperties leaseContainerProperties = new ContainerProperties("consoleLeases", "/id");

Container leaseContainer = await  db.CreateContainerIfNotExistsAsync(leaseContainerProperties);

Container destinationContainer = await db.CreateContainerIfNotExistsAsync(id:
"CartContainerByState", partitionKeyPath: "/BuyerState");
```

The **Lease Container** stores information to allow for parallel processing of the change feed, and acts as a bookmark for where we last processed changes from the feed.

6. Now, add the following lines of code directly after the **leaseContainer** definition in order to get an instance of the change processor:

```
var builder = container.GetChangeFeedProcessorBuilder("migrationProcessor",
(IReadOnlyCollection<object> input, CancellationToken cancellationToken) => {
   Console.WriteLine(input.Count + " Changes Received");
   //todo: Add processor code here
});

var processor = builder
            .WithInstanceName("changeFeedConsole")
            .WithLeaseContainer(leaseContainer)
                        .Build();
```

Each time a set of changes is received, the Func<T> defined in CreateChangeFeedProcessorBuilder will be called. We're skipping the handling of those changes for the moment.

7. In order for our processor to run, we have to start it. Following the definition of **processor** add the following line of code:

```
await processor.StartAsync();
```

8. Finally, when a key is pressed to terminate the processor we need to end it. Locate the //todo: Add stop code here line and replace it with this code:
   `await processor.StopAsync();`

Complete the Live Data Migration

1. Within the program.cs file in the **ChangeFeedConsole** folder, locate the todo we left ourselves //todo: Add processor code here
2. Modify the signature of the Func<T> in the GetChangeFeedProcessorBuilder replacing object with CartAction as follows:

```
var builder = container.GetChangeFeedProcessorBuilder("migrationProcessor",
  (IReadOnlyCollection<CartAction> input, CancellationToken cancellationToken) =>
  {
    Console.WriteLine(input.Count + " Changes Received");
    //todo: Add processor code here
          });
```

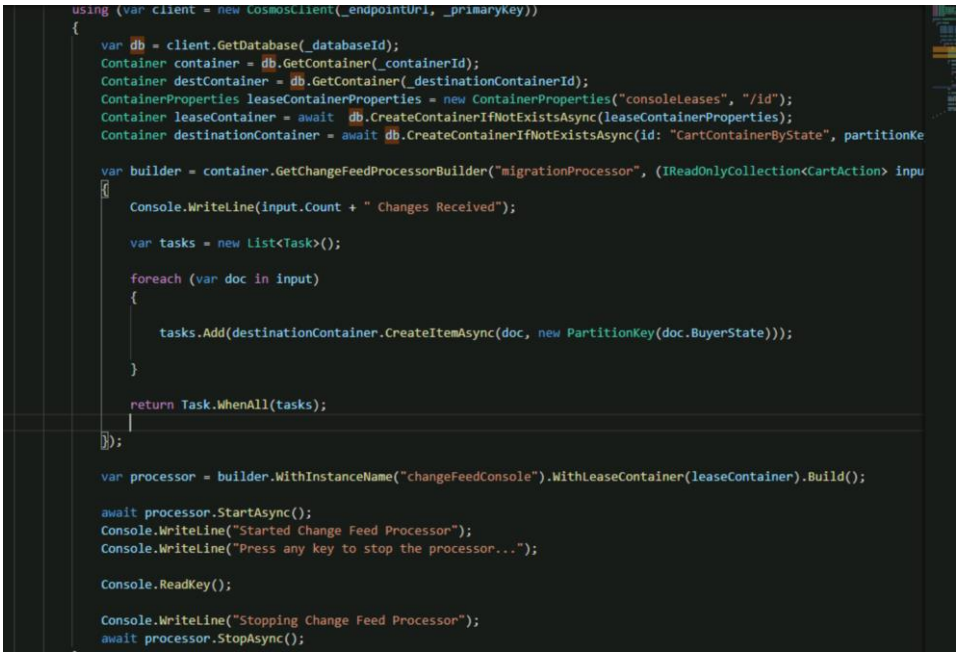
3. The **input** is a collection of **CartAction** documents that have changed. To migrate them, we'll simply loop through them and write them out to our destination container. Replace the //todo: Add processor code here with the following code:

```
var tasks = new List<Task>();

foreach (var doc in input)
{
  tasks.Add(destinationContainer.CreateItemAsync(doc, new PartitionKey(doc.BuyerState)));
}

return Task.WhenAll(tasks);
```

You can check the solution in the C:_COSMOSHACK_\labs\LAB04-ChangeFeed\ChangeFeedConsole\Program.cs.complete.txt



```
using (var client = new CosmosClient(_endpointUri, _primaryKey))
{
    var db = client.GetDatabase(_databaseId);
    Container container = db.GetContainer(_containerId);
    Container destContainer = db.GetContainer(_destinationContainerId);
    ContainerProperties leaseContainerProperties = new ContainerProperties("consoleLeases", "/id");
    Container leaseContainer = await  db.CreateContainerIfNotExistsAsync(leaseContainerProperties);
    Container destinationContainer = await db.CreateContainerIfNotExistsAsync(id: "CartContainerByState", partitionKe

    var builder = container.GetChangeFeedProcessorBuilder("migrationProcessor", (IReadOnlyCollection<CartAction> inpu
    {
        Console.WriteLine(input.Count + " Changes Received");

        var tasks = new List<Task>();

        foreach (var doc in input)
        {

            tasks.Add(destinationContainer.CreateItemAsync(doc, new PartitionKey(doc.BuyerState)));

        }

        return Task.WhenAll(tasks);

    });

    var processor = builder.WithInstanceName("changeFeedConsole").WithLeaseContainer(leaseContainer).Build();

    await processor.StartAsync();
    Console.WriteLine("Started Change Feed Processor");
    Console.WriteLine("Press any key to stop the processor...");

    Console.ReadKey();

    Console.WriteLine("Stopping Change Feed Processor");
    await processor.StopAsync();
```

Test to Confirm the Change Feed Function Works

Now that we have our first Change Feed consumer, we're ready to run a test and confirm that it works

1. Open a **second** terminal window and navigate to the **ChangeFeedConsole** folder

2. Start up your console app by running the following commands in the **second** terminal window:

3. cd ChangeFeedConsole
4.
   dotnet run

5. Once the function starts running you'll see the following messages in your console:

6. Started Change Feed Processor
   Press any key to stop the processor...

   Because this is the first we've run this consumer, there will be no data to consume. We'll start the data generator in order to start receiving changes.

7. In the **first** terminal window, navigate to the **DataGenerator** folder

8. Start the **DataGenerator** again by running the following command in the **first** terminal window

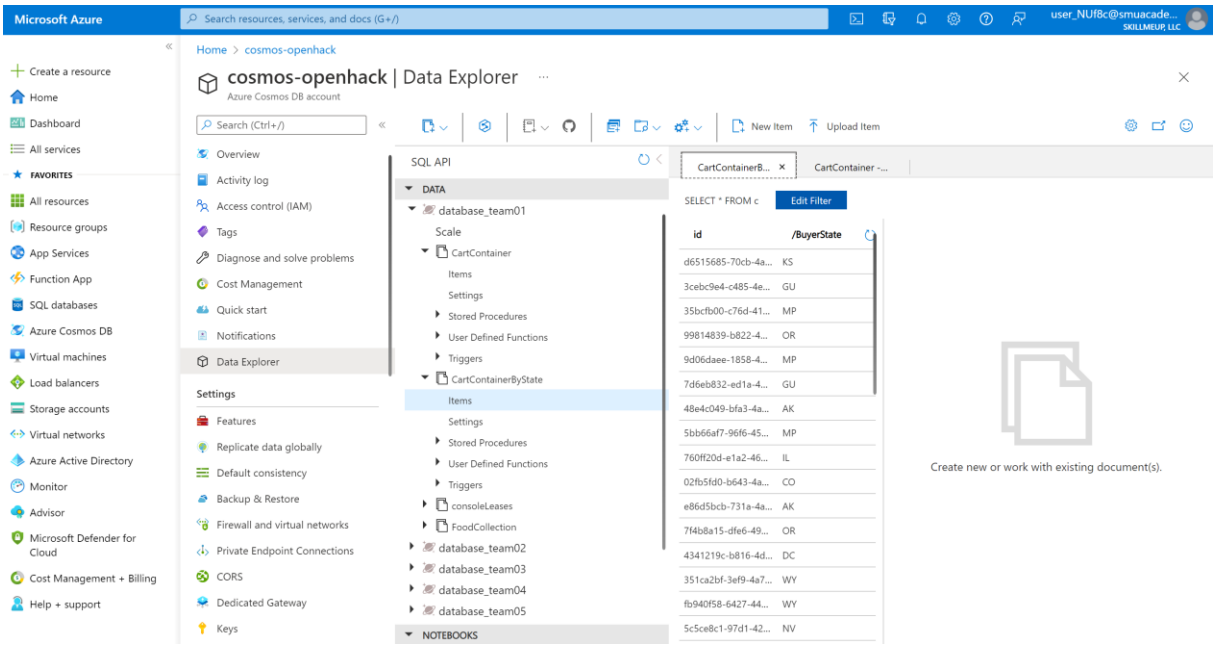
   dotnet run

9. You should see the asterisks start to appear again as the data is being written.

10. Soon after data starts being written, you'll start to see the following output in the **second** terminal window:

11. 100 Changes Received
12. 100 Changes Received
13. 3 Changes Received
    ...

14. After a few minutes, navigate to the c**osmos-openhack** Data Explorer and expand **database_teamXX** then **CartContainerByState** and select **Items**. You should see items populating there, and note that the Partition Key this time is /BuyerState.



15. Press any key in the **first** terminal to stop data generation

16. Let the **ChangeFeedConsole** finish running (it shouldn't take very long). You'll know it's done when it stops writing new log messages. Stop the function by pressing any key in the **second** terminal window.

You've now written your first Cosmos DB Change Feed consumer, which writes live data to a new collection. Congrats! In the next steps we'll take a look at using Azure Functions to consume Cosmos DB change feed for two additional use cases.