ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Modern Data Management

Azure Streams Fundamentals & Business Intelligence

Academic Year 2022 - 2023
Full Time
Assignment 3

Dimitris Matsanganis, f2822212
Foteini Nefeli Nouskali, f2822213

BUSINESS ANALYTICS
Master of Science

Modern Data Management & Business Intelligence

Assignment 3

Description of the Case

You are going to use Azure Stream Analytics to process a data stream of ATM transactions and answer stream queries. The schema of the stream is: (ATMCode, CardNumber, Type, Amount)

1. Create a students account at: https://azure.microsoft.com/en-us/free/students/

2. Setup an Event Hub.

3. Generate a Security Access Signature (use a terminal with windows operationg system): https://github.com/sandrinodimattia/RedDog/releases

4. Edit Generator.html (open with a text editor, e.g.: Sublime or Notepad++) and update the CONFIG variables. Keep the "js" folder in the same folder as the Generator.html file.

5. Feed the Event Hub with the use of Generator.html (In order to start the Stream Generator, open the Generator.html with a web browser, e.g.: Chrome and press the "Send Data" button.)

6. Setup a Storage account.

7. Upload the Reference Data files to your storage account.

8. Setup a Stream Analytics Job.

9. Use the Event Hub + Reference Data Files as Input.

10. Create a Blob Storage Output.

SCENARIO

You have access to a data stream that's generated from ATMs. Each event contains data related to the transaction that took place. You are asked to create an Azure Analytics solution for the tasks listed in the "QUERIES" section.

REFERENCE DATA

GENERAL NOTES

- Use only the parts of the datasets that you need for the queries.

- Use Sublime or Notepad++ to have a proper view of the datasets.

AREA.json

- Geographical Information.

- Connects each area_code with a city and a country.

- The “area_code” of this dataset can be joined with the “area_code” of ATM.json

ATM.json

- Information about the ATM.

- Connects each ATM with an area.

- The “atm_code” of this dataset can be joined with the input’s “ATMCode” section.

Customer.json

- Information about each customer.

- Provides demographic information about each customer.

- The “card_number” of this dataset can be joined with the input’s “CardNumber” section.

DATA STREAM INPUT

GENERAL NOTES

Events generated have the following format:

{

"**ATMCode**": 13,

"**CardNumber**": 5610827137784218,

"**Type**": 0,

"**Amount**": 37

}

FIELDS DESCRIPTION

- **ATMCode**: Information about the type of the ATM. Can be joined with Atm.json

- **CardNumber**: The card number related to the transaction. Can be joined with Customer.json

- **Type**: The type of the transaction. There are two types of transactions, type “1” and type “0”.

- **Amount**: The amount of the transaction.

QUERIES

Query 1: Show the total "Amount" of "Type = 0" transactions at "ATM Code = 21" of the last 10 minutes. Repeat as new events keep flowing in (use a sliding window).

Query 2: Show the total "Amount" of "Type = 1" transactions at "ATM Code = 21" of the last hour. Repeat once every hour (use a tumbling window).

Query 3: Show the total "Amount" of "Type = 1" transactions at "ATM Code = 21" of the last hour. Repeat once every 30 minutes (use a hopping window).

Query 4: Show the total "Amount" of "Type = 1" transactions per "ATM Code" of the last one hour (use a sliding window).

Query 5: Show the total "Amount" of "Type = 1" transactions per "Area Code" of the last hour. Repeat once every hour (use a tumbling window).

Query 6: Show the total "Amount" per ATM's "City" and Customer's "Gender" of the last hour. Repeat once every hour (use a tumbling window).

Query 7: Alert (Do a simple SELECT "1") if a Customer has performed two transactions of "Type = 1" in a window of an hour (use a sliding window).

Query 8: Alert (Do a simple SELECT "1") if the "Area Code" of the ATM of the transaction is not the same as the "Area Code" of the "Card Number" (Customer's Area Code) - (use a sliding window)

DELIVERABLE

A .pdf file that documents (through screenshots) the process of setting up your account and executing the queries described above. Make sure that you start each job and you include screenshots of the output files.

Contents

Description of the Case 2
Creation of Academic - Students Account 7
Event Hub Setup Procedure 9
Creation of Namespace 10
Creation of Event Hub 12
Setting Up Event Hub Policies 14
Setting Event Hubs Signature Generator 17
Configuration of HTML Genarator 17
Send data with the HTML Genarator 18
Creation of Blob Storage Account 19
Creation of Container at Blob Storage Account 20
Upload Reference Data at Container 21
Creation of Stream Analytics Job 22
Add Inputs at Stream 23
Add Output at Stream 26
Sample data from input 27
Upload test data at Stream 27
Test Query and Results 28
Start Stream 28
Blob output of stream 29
Queries 30
Query 1 30
Input 30
Output 30
Query 2 31
Input 31
Output 31
Query 3 32
Input 32
Output 32
Query 4 33
Input 33
Output 33

Query 5 34
Input 34
Output 34
Query 6 35
Input 35
Output 35
Query 7 37
Input 37
Output 38
Query 8 39
Input 39
Output 39

Creation of Academic - Students Account

Following the provided link: https://azure.microsoft.com/en-us/free/students/ we are going to the Azure in order to create our academic profile. The steps in order to create successfully our academic account are provided below.

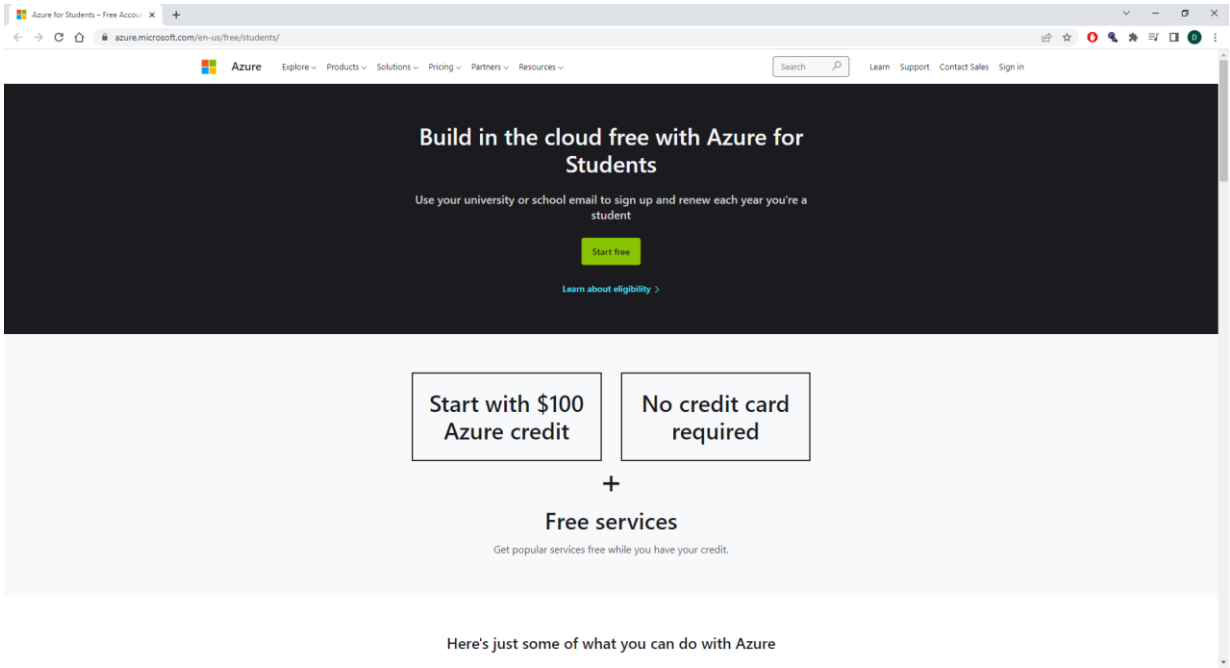


Figure 1: MS Azure Student Account webpage.

After by clicking the "Start free" option we login with our academic credentials (the AUEB ones) and we confirm through phone that we are a human and not a bot. Then we are able to create our profile.

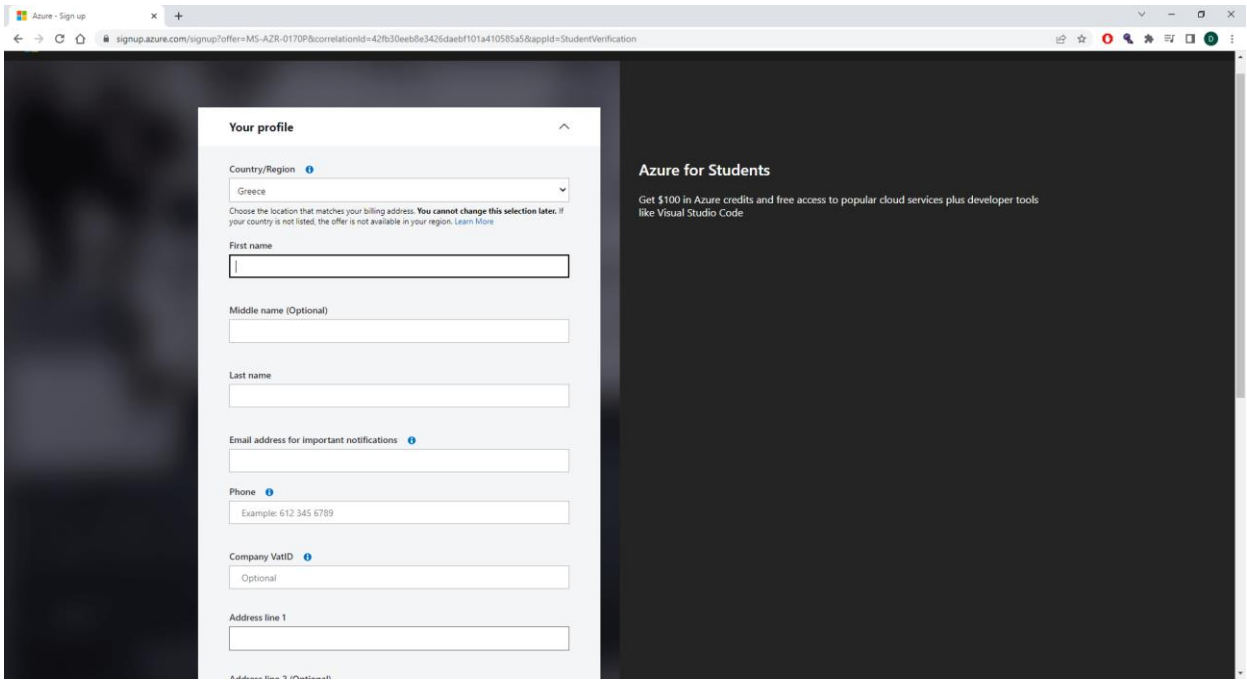


Figure 2: MS Azure Profile Set Up.

In the end we successfully create our academic profile, and we can move on to the set up of Event Hub procedure.

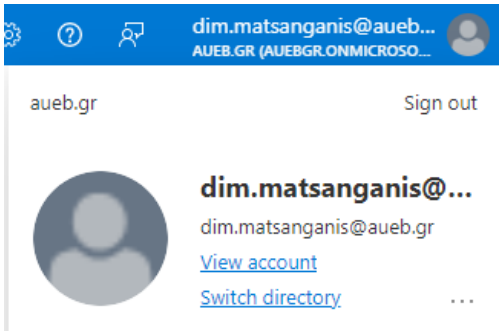


Figure 3: Successful creation of MS Azure Academic Profile 1/2.

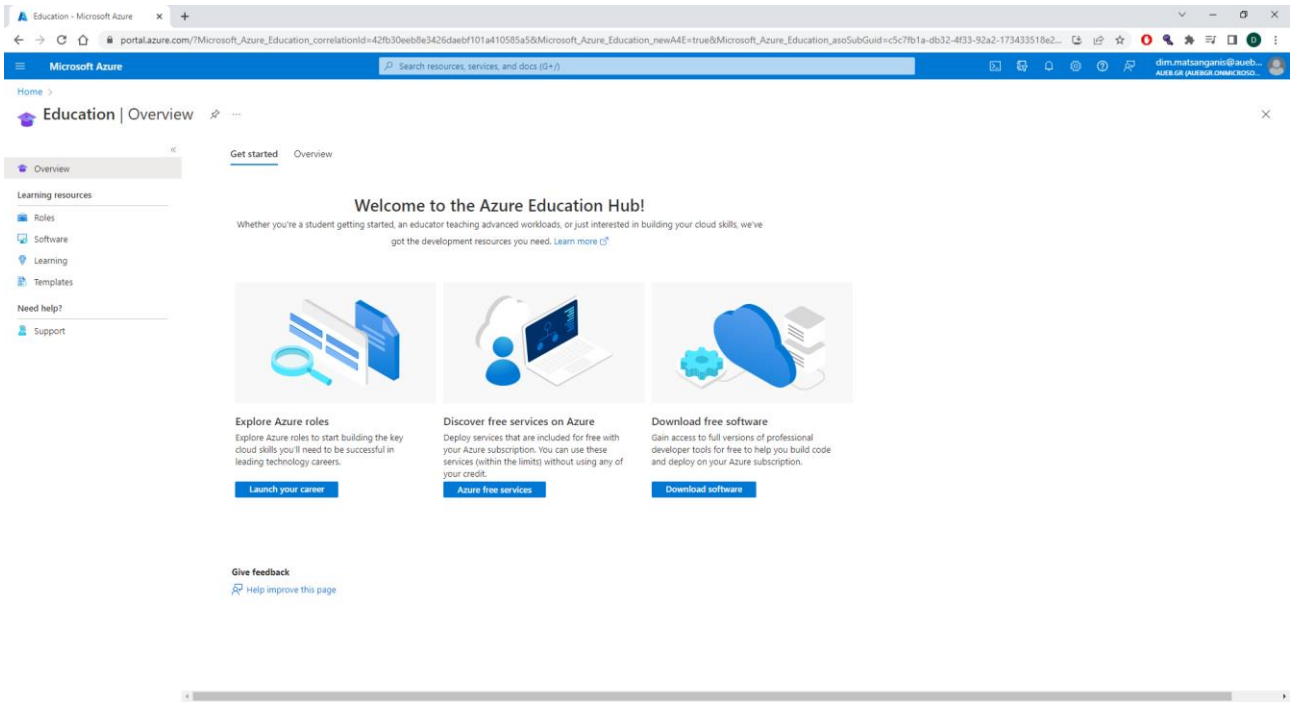


Figure 4: Successful creation of MS Azure Academic Profile 2/2.

Event Hub Setup Procedure

In the pictures below you can see the Event Hub Setup process that was followed in order to successfully create the Event Hub.

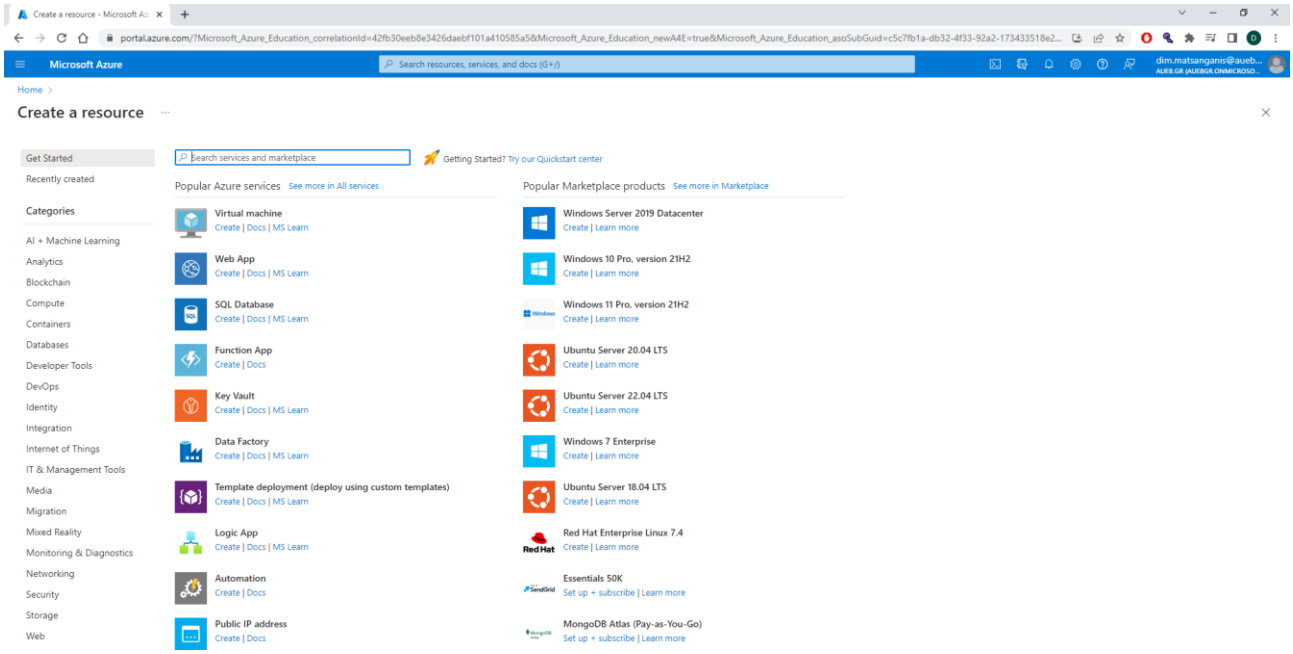


Figure 5: Event Hub Selection 1/2.

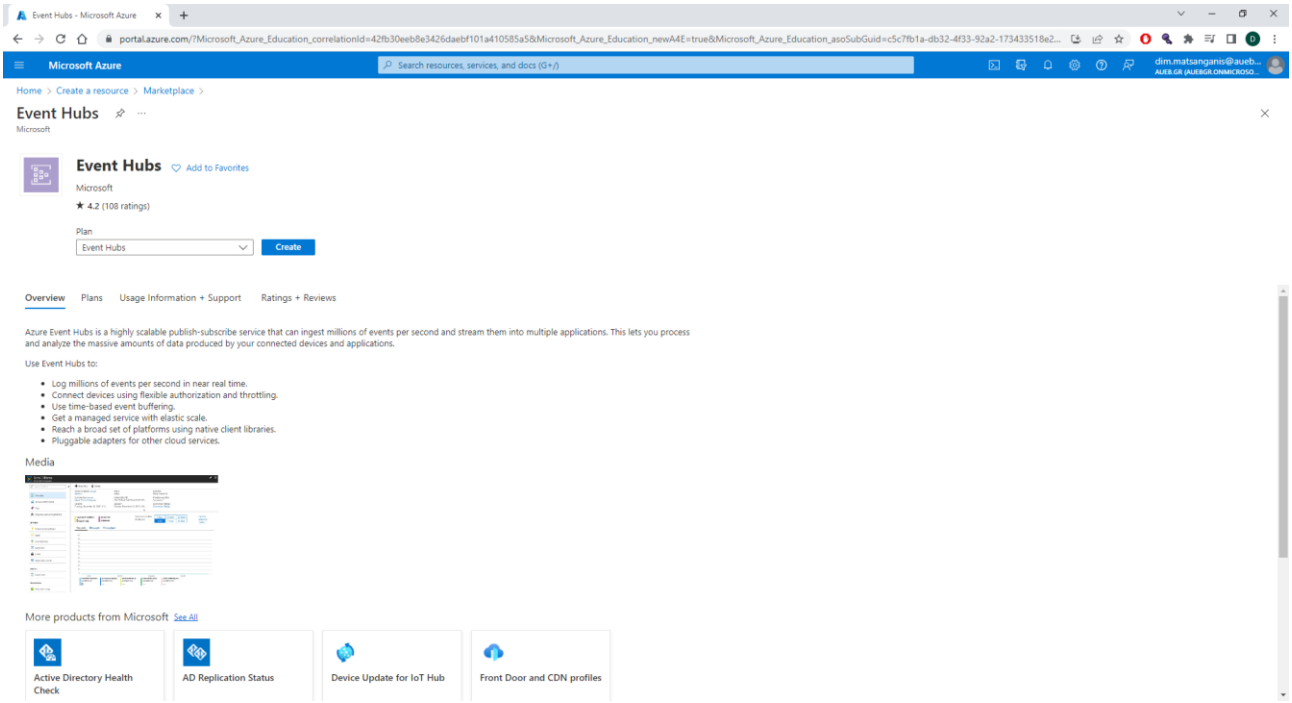


Figure 6: Event Hub Selection 2/2.

Creation of Namespace

In this section the creation process of the namespace of our Event Hub is going to be described below:

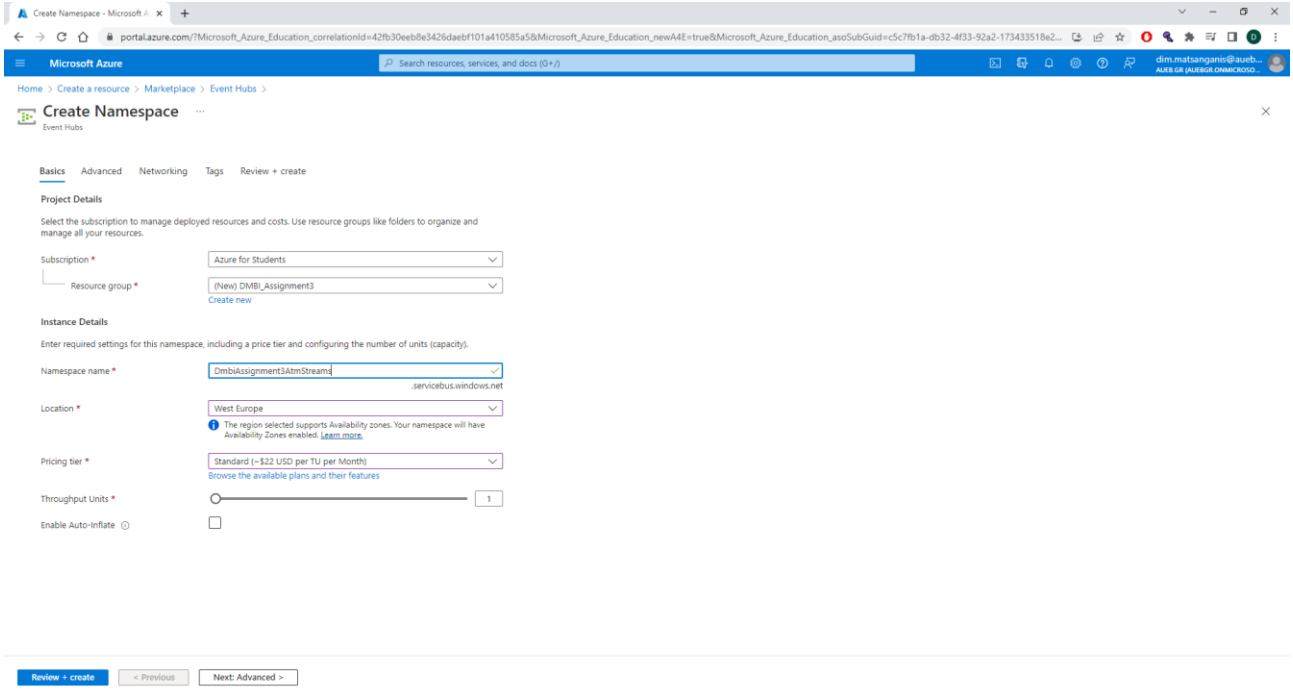


Figure 7: Setup Namespace constraints.

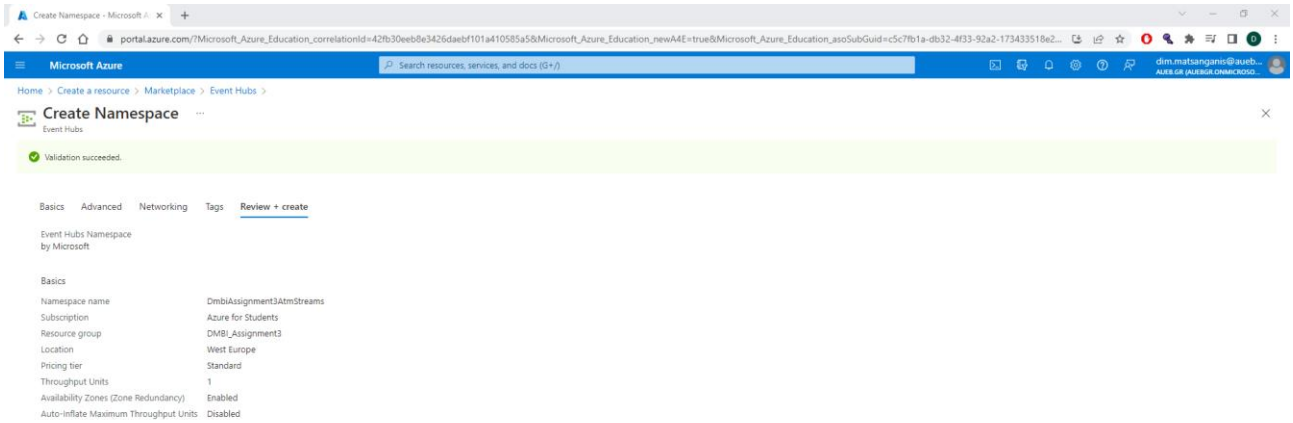


Figure 8: Validation of the Namespace.

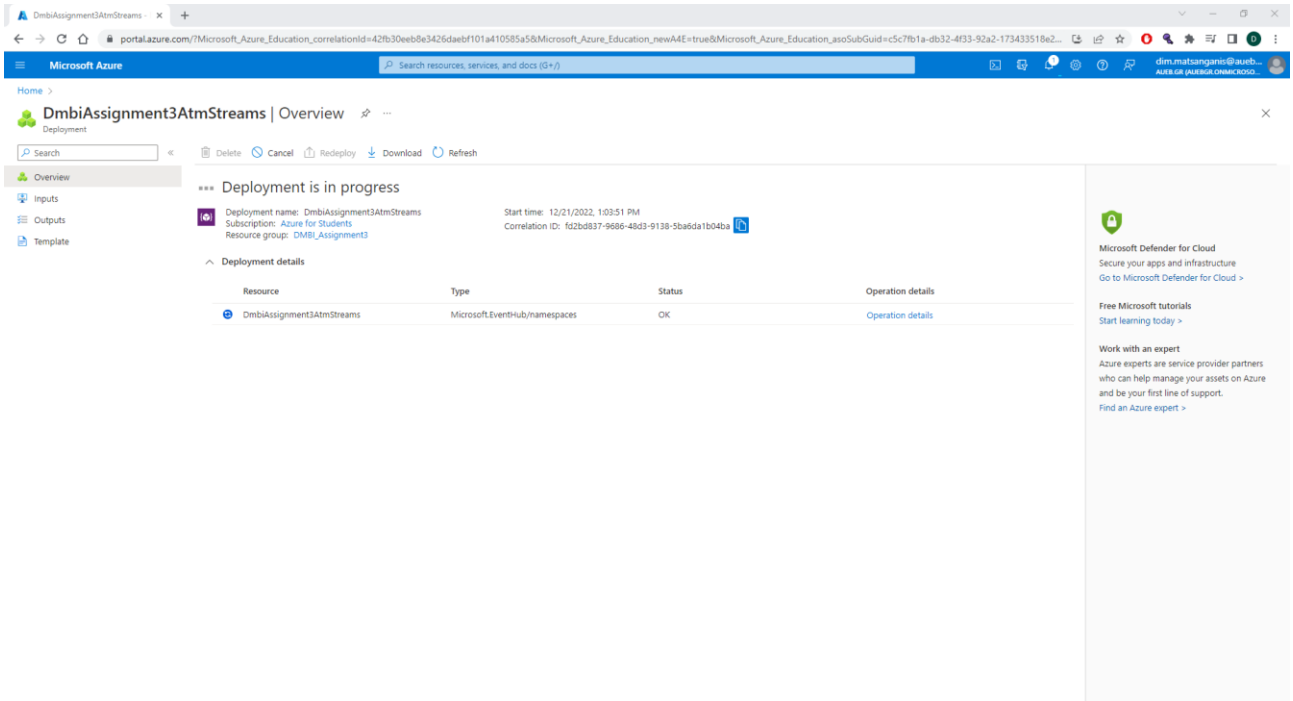


Figure 9: Deployment Process (in progress).

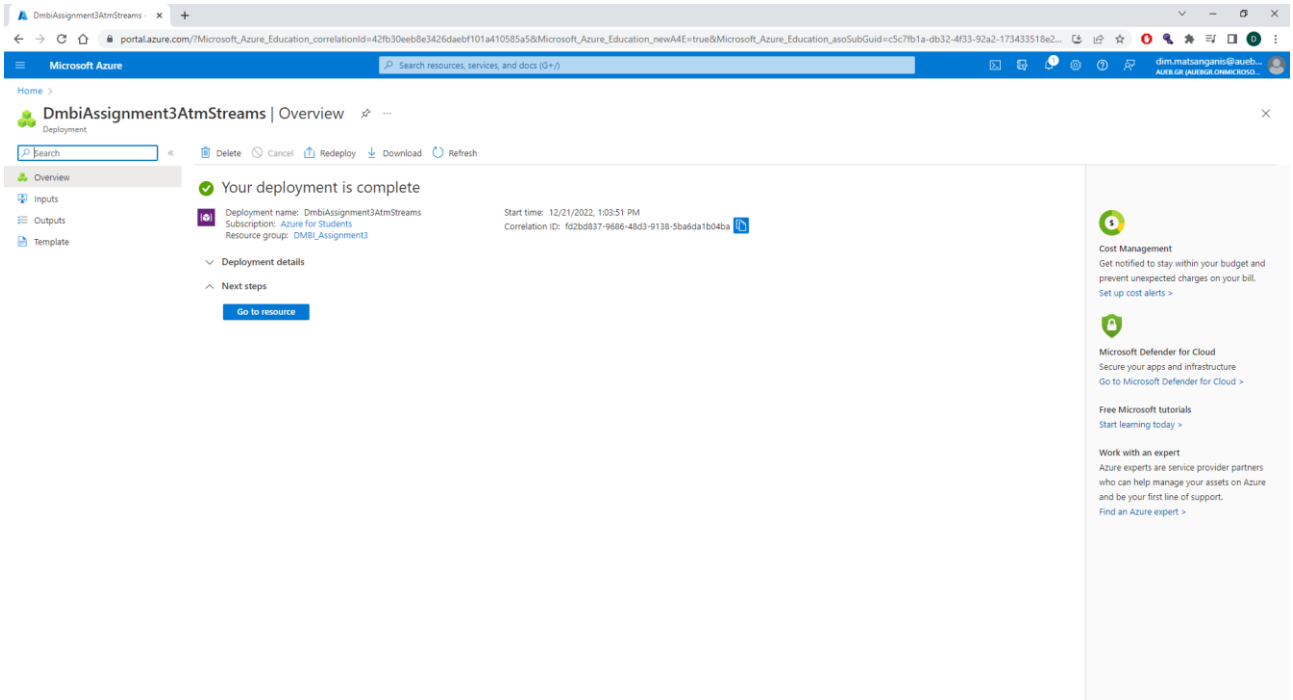


Figure 10: Deployment Process (completion).

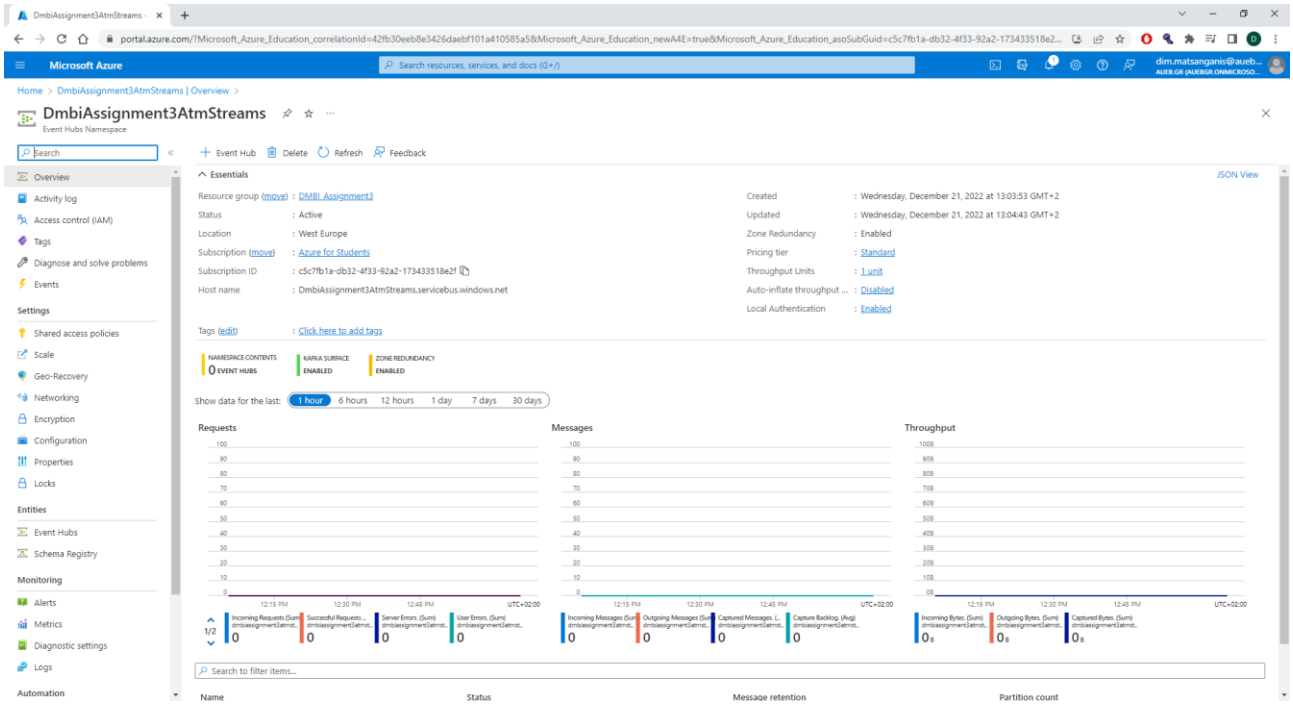


Figure 11: Event Hub Namespace Output.

Creation of Event Hub

Now we are able to create our Event Hub as you can see below:

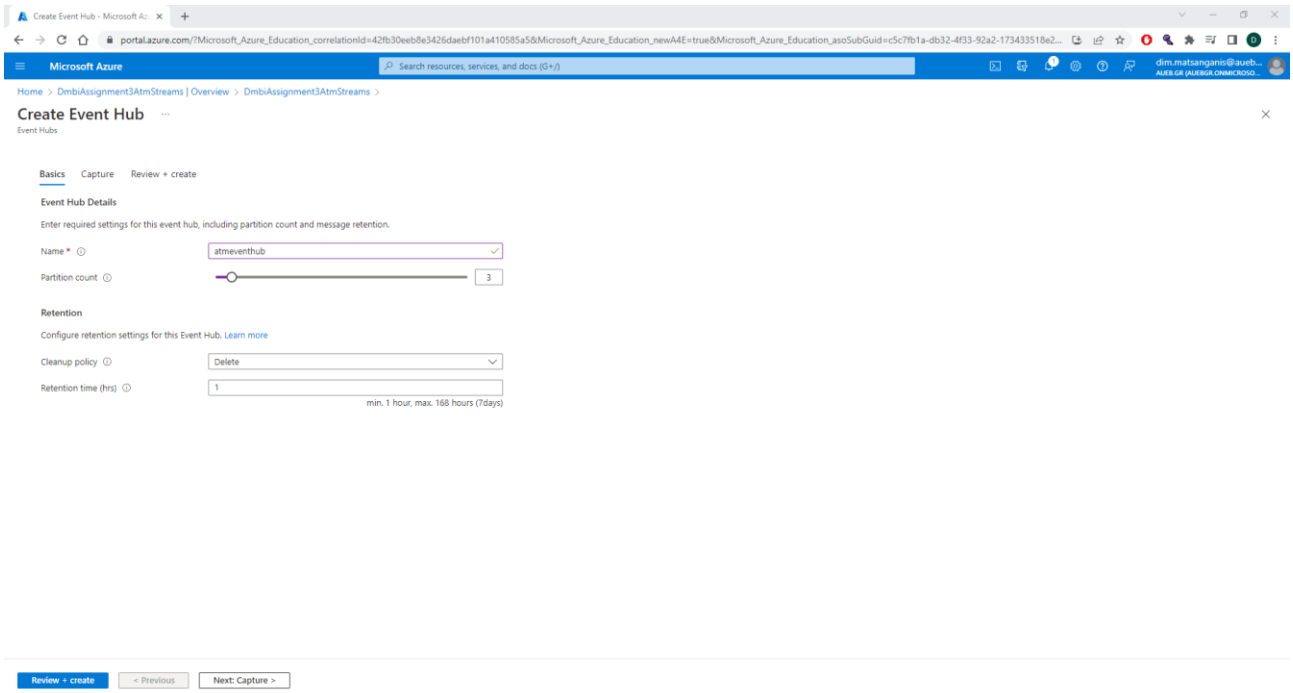


Figure 12: Setup Event Hub (1/3).

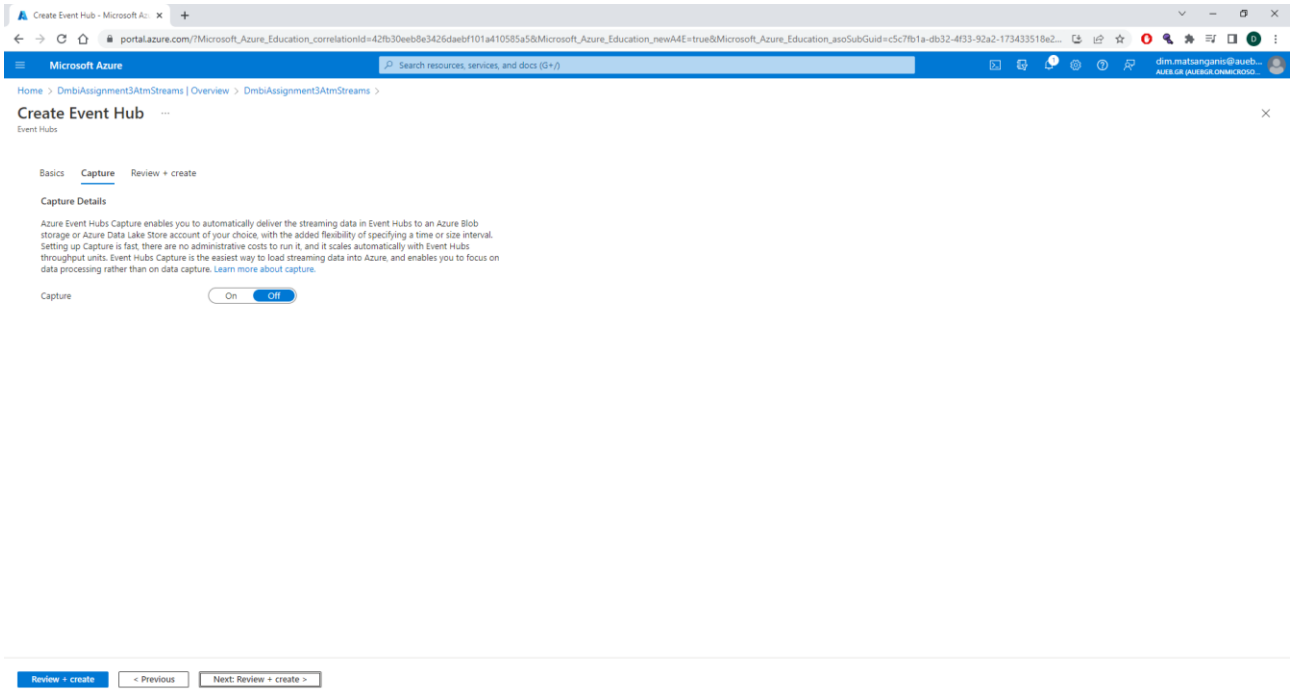


Figure 13: Setup Event Hub (2/3).

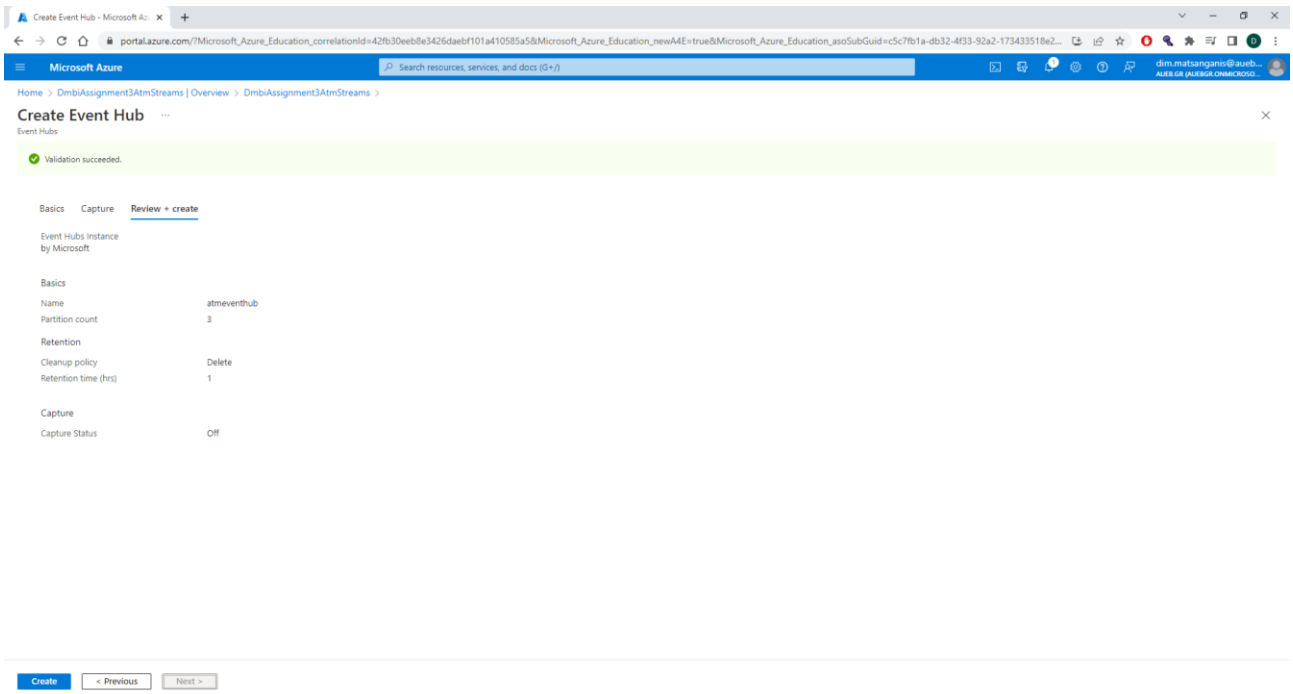


Figure 14: Setup Event Hub Validation (3/3).

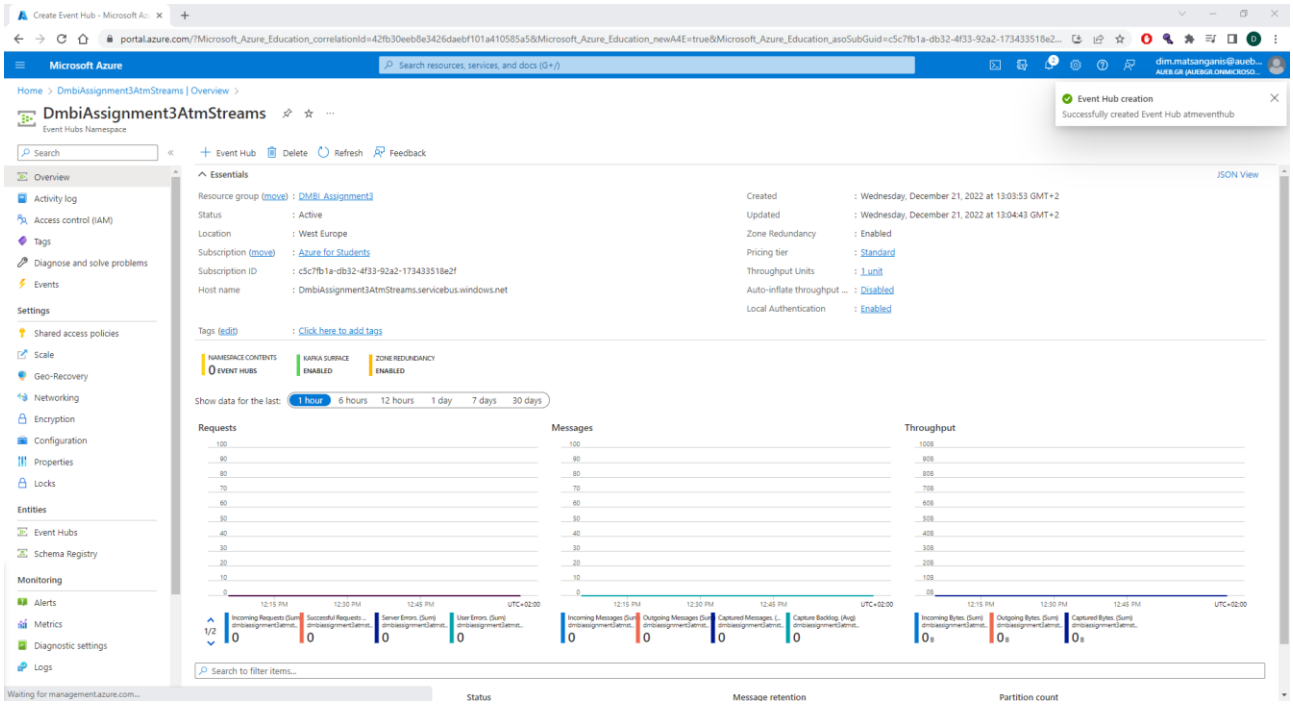


Figure 15: Event Hub Creation Success Message.

Setting Up Event Hub Policies

In this section we are going to create our Send and Listen Policies as you can see below:

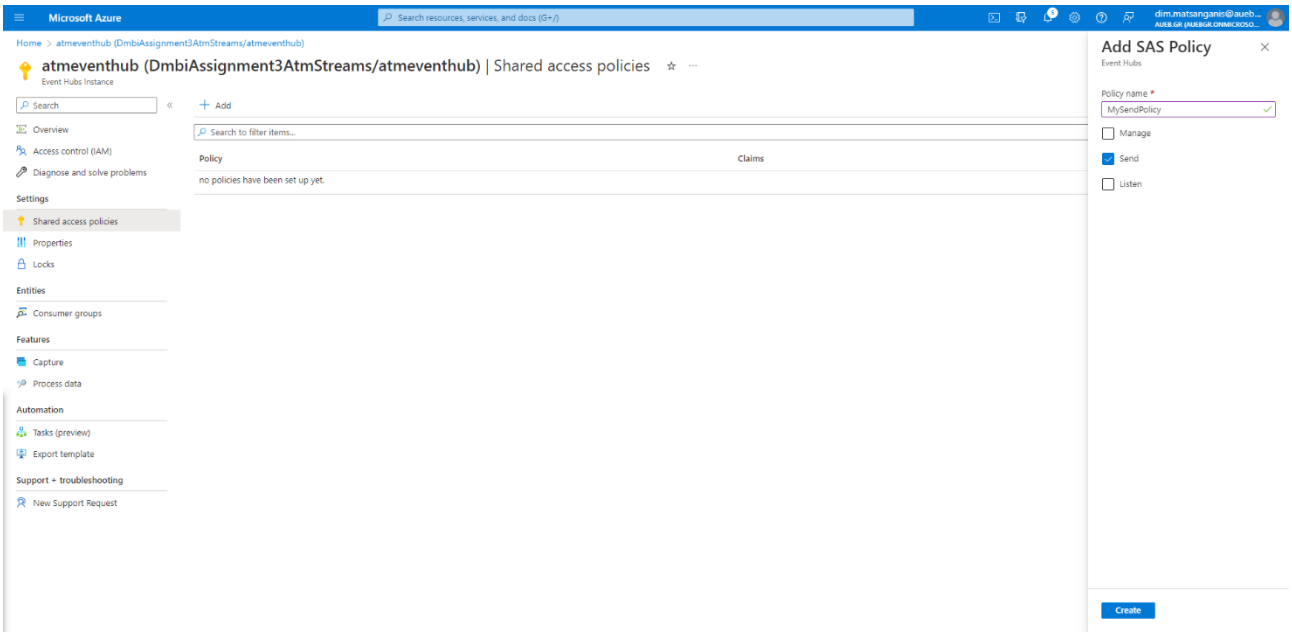


Figure 16: Send Policy Creation.

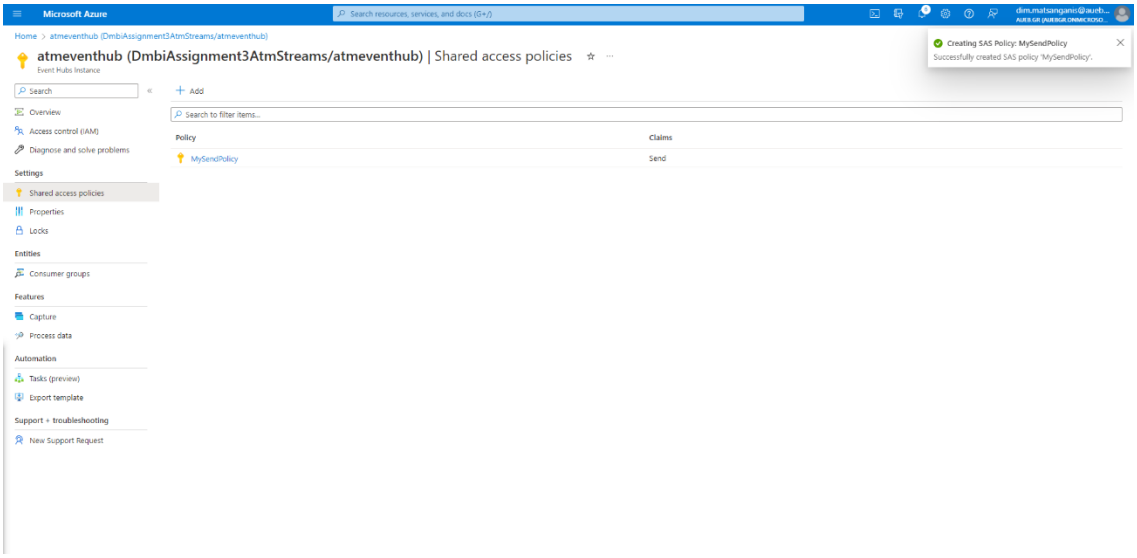


Figure 17: Send Policy Creation Success Message.

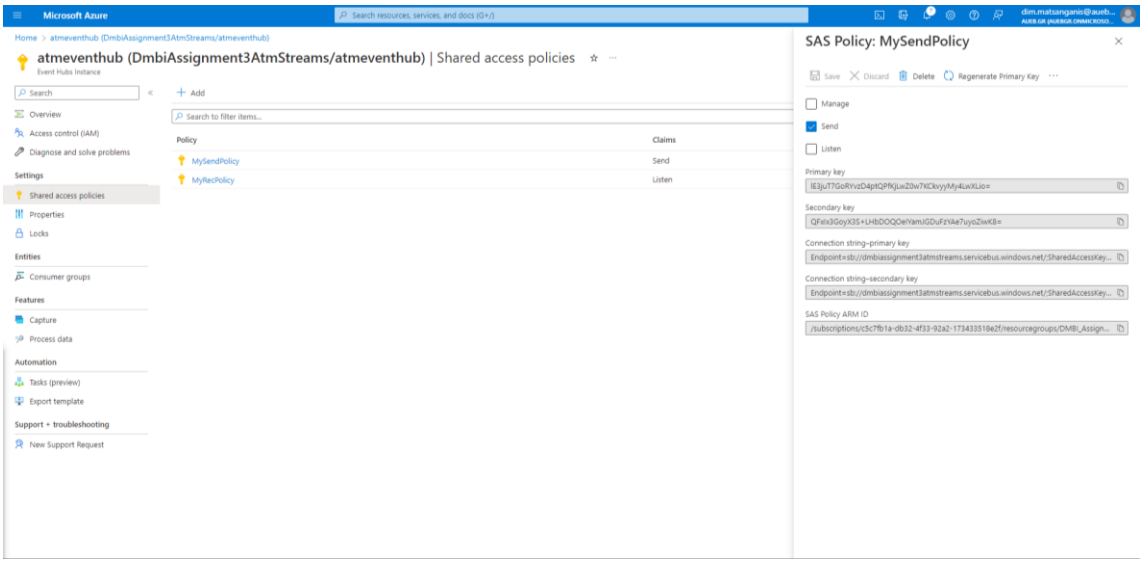


Figure 18: Send Policy.

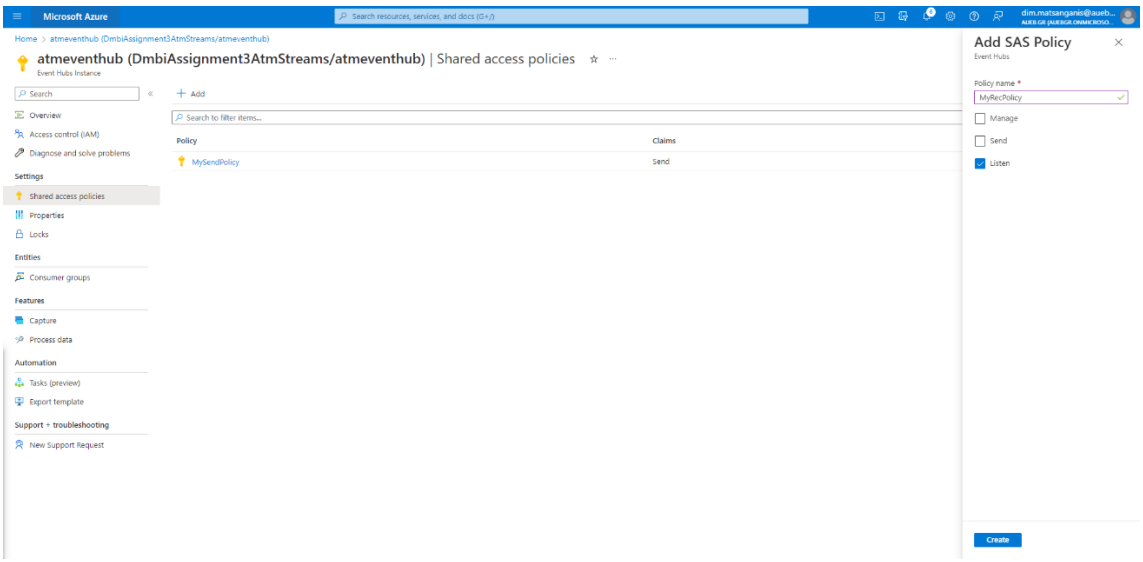


Figure 19: Rec Policy Creation.

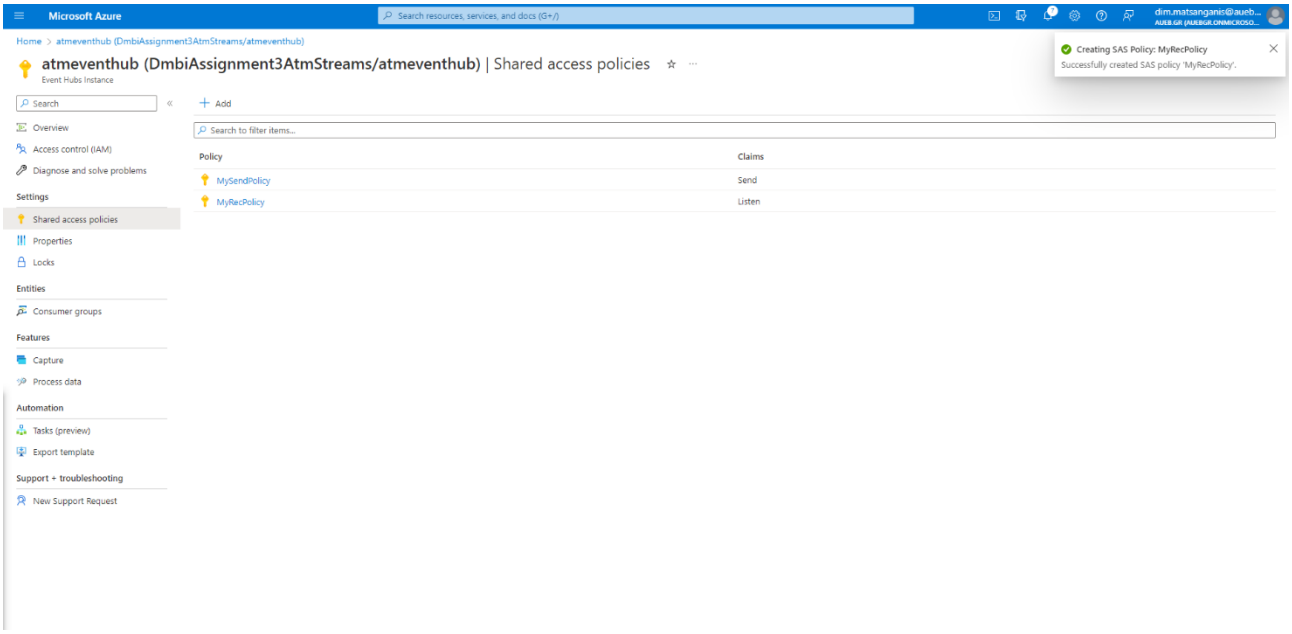


Figure 20: Rec Policy Creation Success Message.

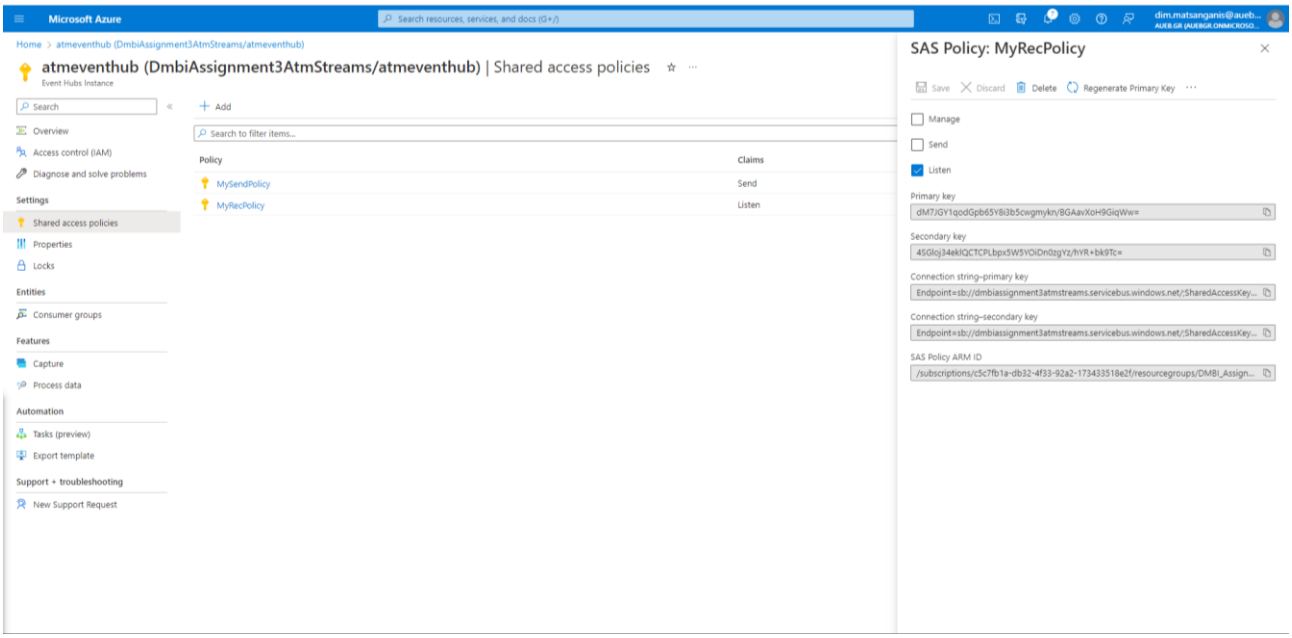


Figure 21: Rec Policy.

Setting Event Hubs Signature Generator

Now, we are going to edit the downloaded signature generator in order to be connected with our Event Hub and we fill the necessary fields to get the signature need it for the HTML Generator as you can see in the following picture:

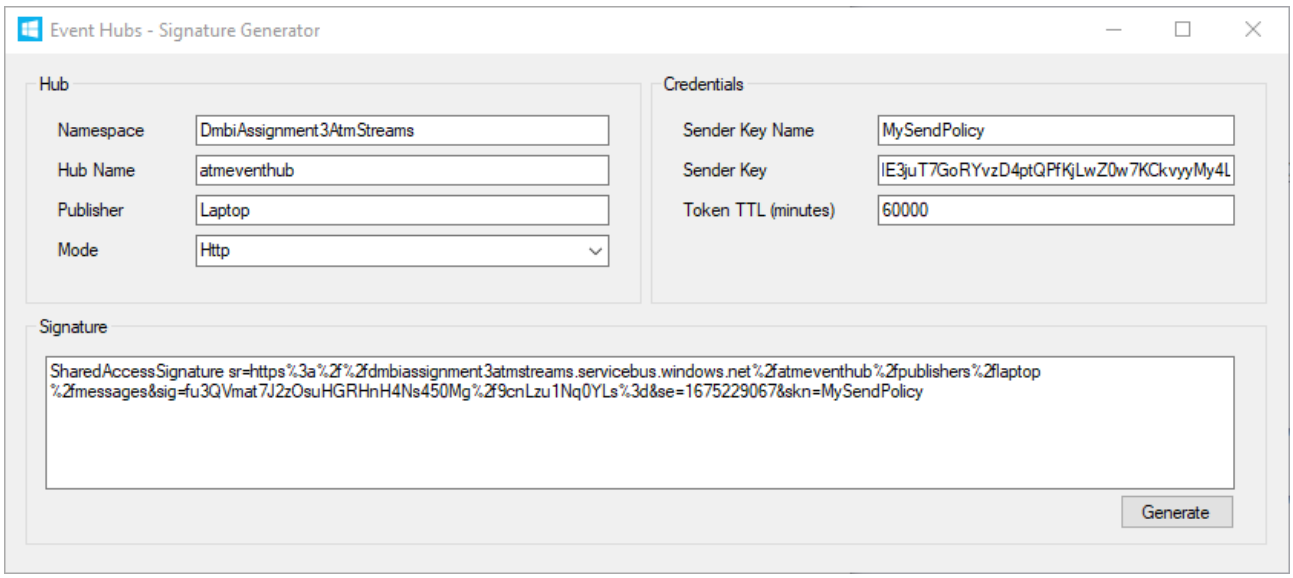
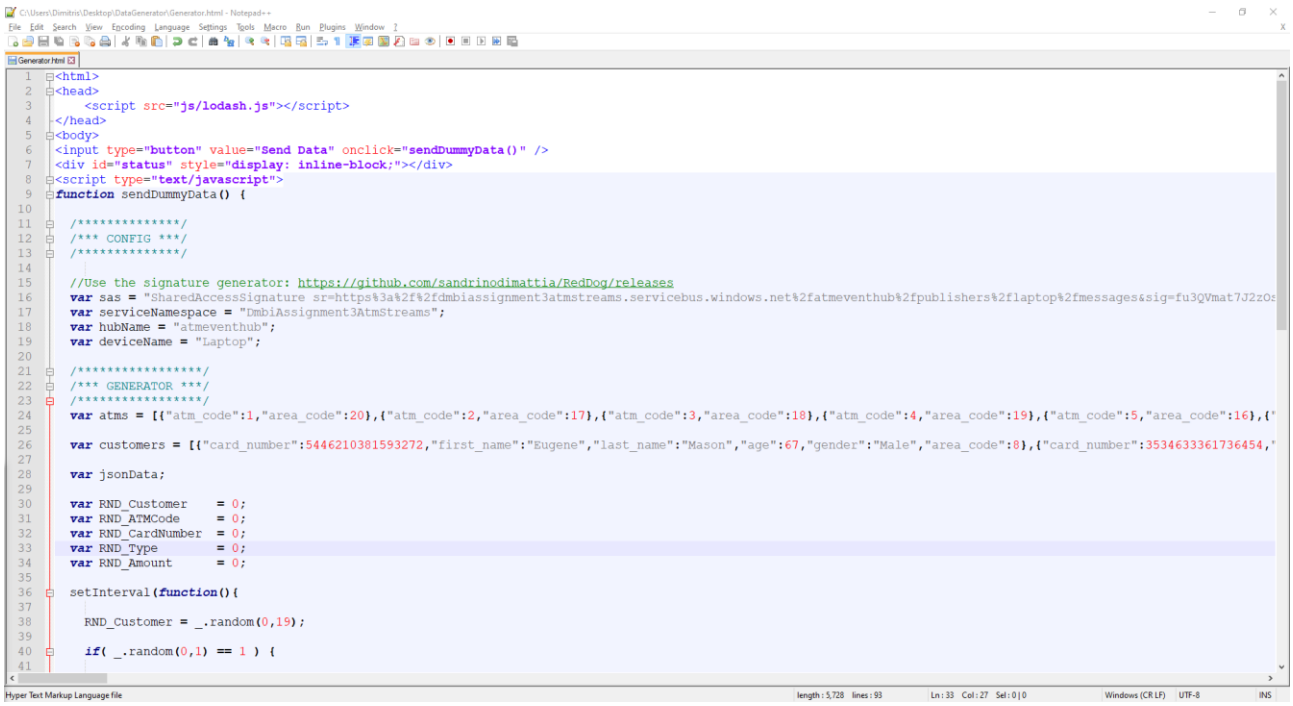


Figure 22: Event Hubs - Signature Generator.

Configuration of HTML Generator

Then, we need to edit our HTML Generator, in order to be connected with our Event Hub. We can do so if we replace the default credentials with the ones we got from the signature generator and our projects (namespace, event hub name and device name).



```
<html>
<head>
    <script src="js/lodash.js"></script>
</head>
<body>
<input type="button" value="Send Data" onclick="sendDummyData()" />
<div id="status" style="display: inline-block;"></div>
<script type="text/javascript">
function sendDummyData() {

  /**************/
  /*** CONFIG ***/
  /**************/

  //Use the signature generator: https://github.com/sandrinodimattia/RedDog/releases
  var sas = "SharedAccessSignature sr=https%3a%2f%2fdmbiassignment3atmstreams.servicebus.windows.net%2fatmeventhub%2fpublishers%2flaptop%2fmessages&sig=fu3QVmat7J2zO
  var serviceNamespace = "DmbiAssignment3AtmStreams";
  var hubName = "atmeventhub";
  var deviceName = "Laptop";

  /*****************/
  /*** GENERATOR ***/
  /*****************/
  var atms = [{"atm_code":1,"area_code":20},{"atm_code":2,"area_code":17},{"atm_code":3,"area_code":18},{"atm_code":4,"area_code":19},{"atm_code":5,"area_code":16},{"

  var customers = [{"card_number":5446210381593272,"first_name":"Eugene","last_name":"Mason","age":67,"gender":"Male","area_code":8},{"card_number":3534633361736454,"

  var jsonData;

  var RND_Customer    = 0;
  var RND_ATMCode     = 0;
  var RND_CardNumber  = 0;
  var RND_Type        = 0;
  var RND_Amount      = 0;

  setInterval(function(){

    RND_Customer = _.random(0,19);

    if( _.random(0,1) == 1 ) {
```

Figure 23: Configuration of HTML Generator through Notepad++.

Send data with the HTML Generator

After we configure our HTML generator we can send data to our event hub. We can evaluate this process through the Inspect page option and the Network tab.

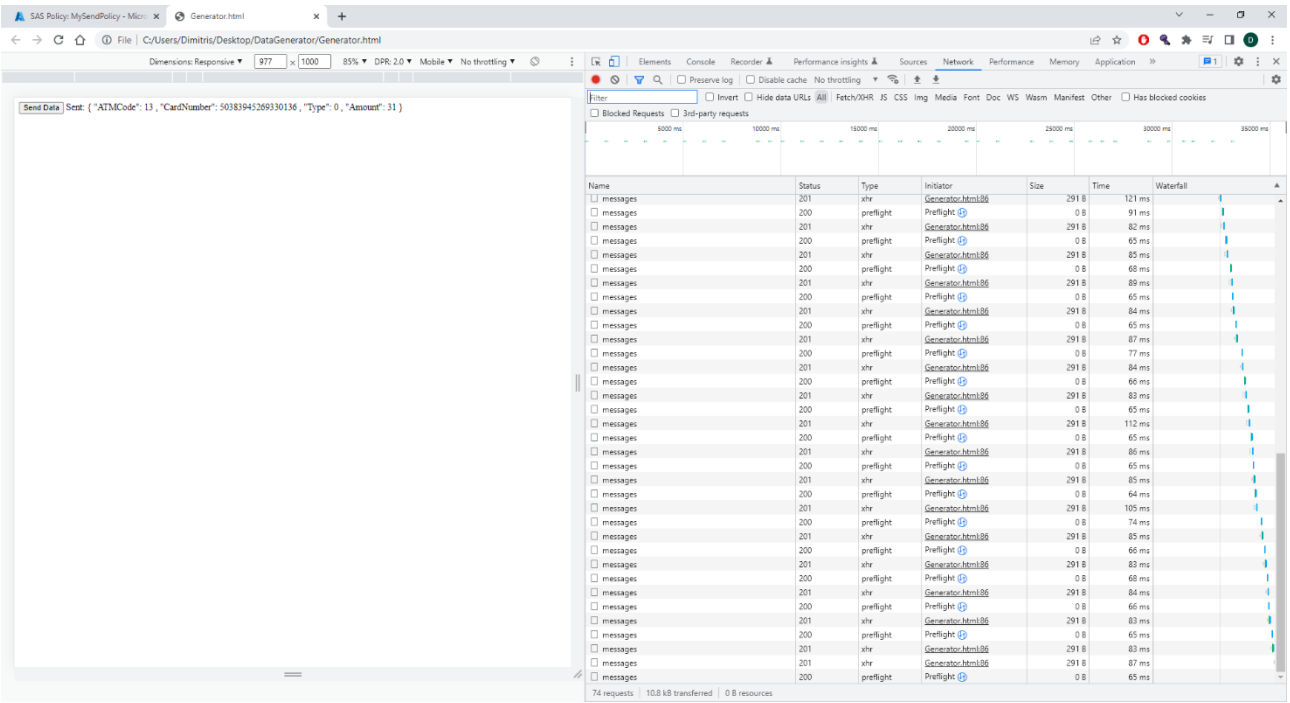


Figure 24: Send data with HTML Generator Example 1.

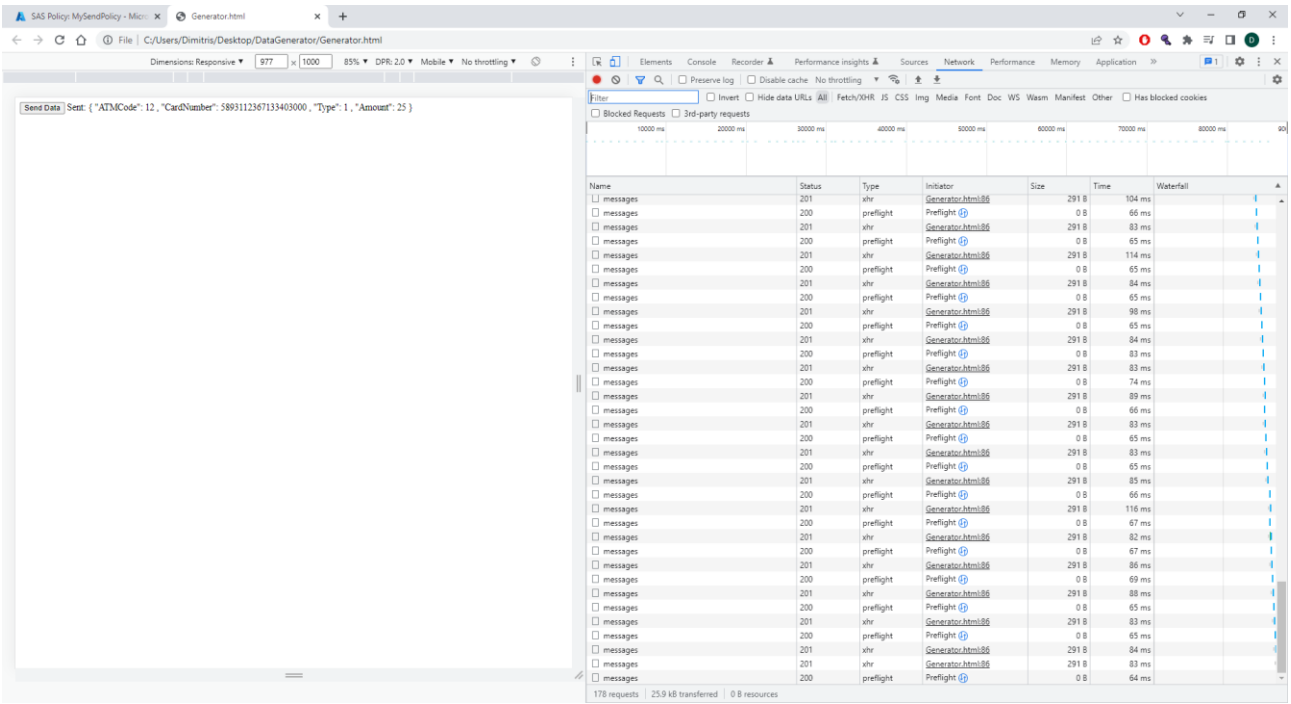


Figure 25: Send data with HTML Generator Example 2.

Creation of Blob Storage Account

Then, we need to create a Blob Storage Account. This process is described below:

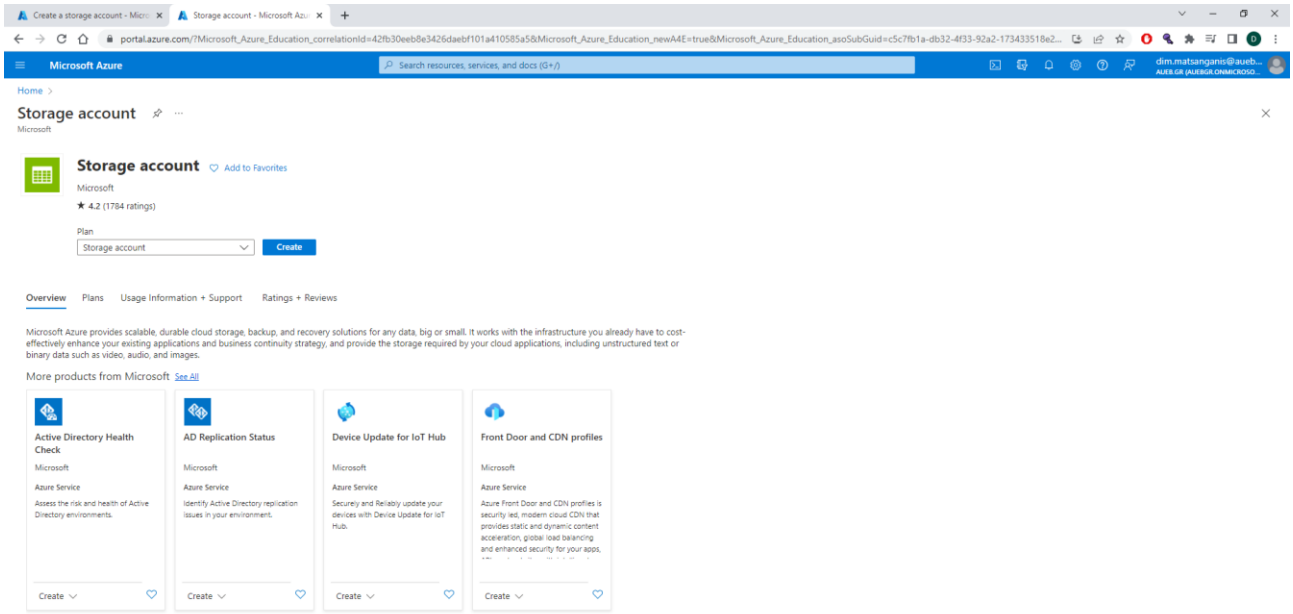


Figure 26: Creation of Storage Account (1/2).

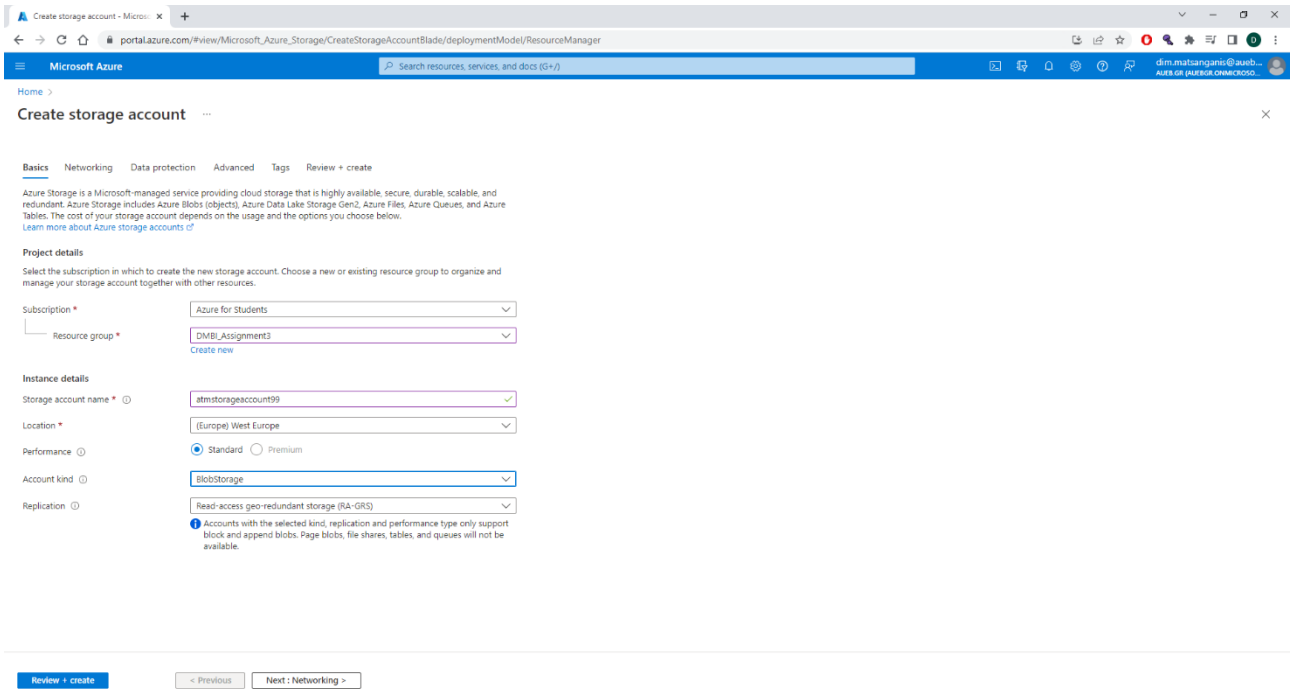


Figure 27: Creation of Blob Storage Account (2/2).

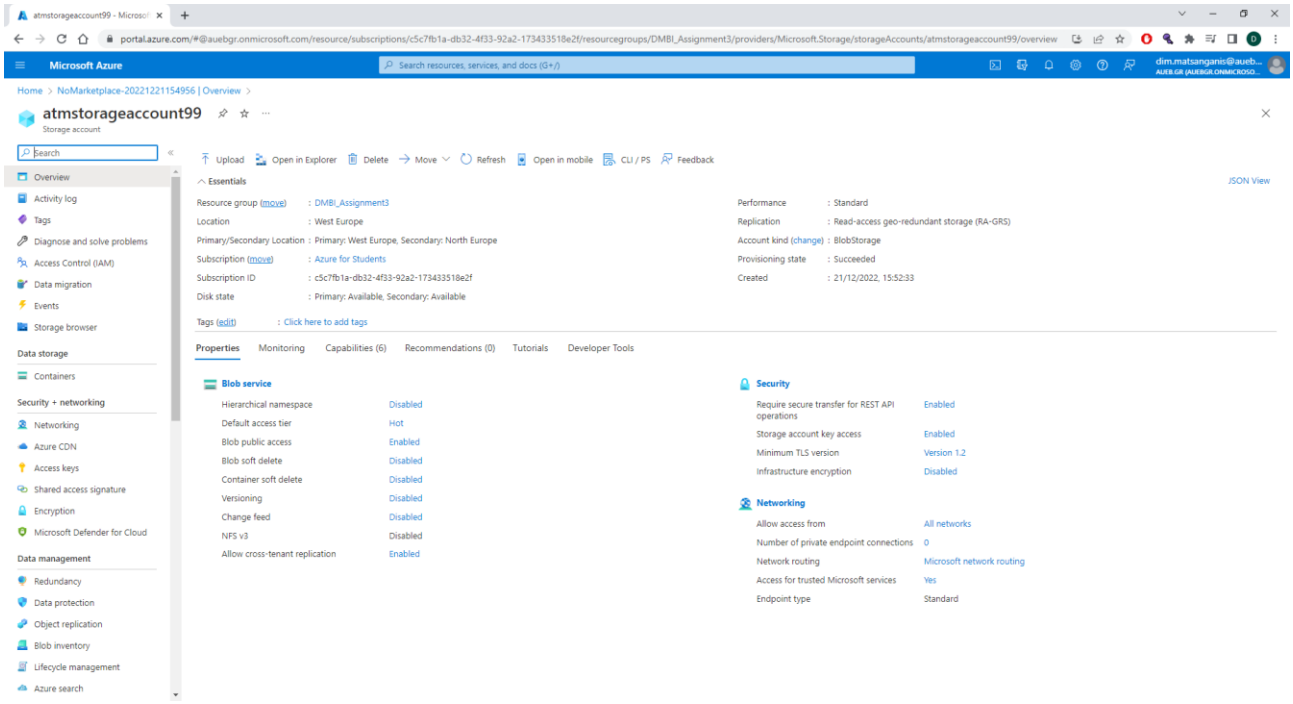


Figure 28: Successful Creation of Storage Account.

Creation of Container at Blob Storage Account

Now, we need to create our blob storage container for the output data (queries' output JSON files).

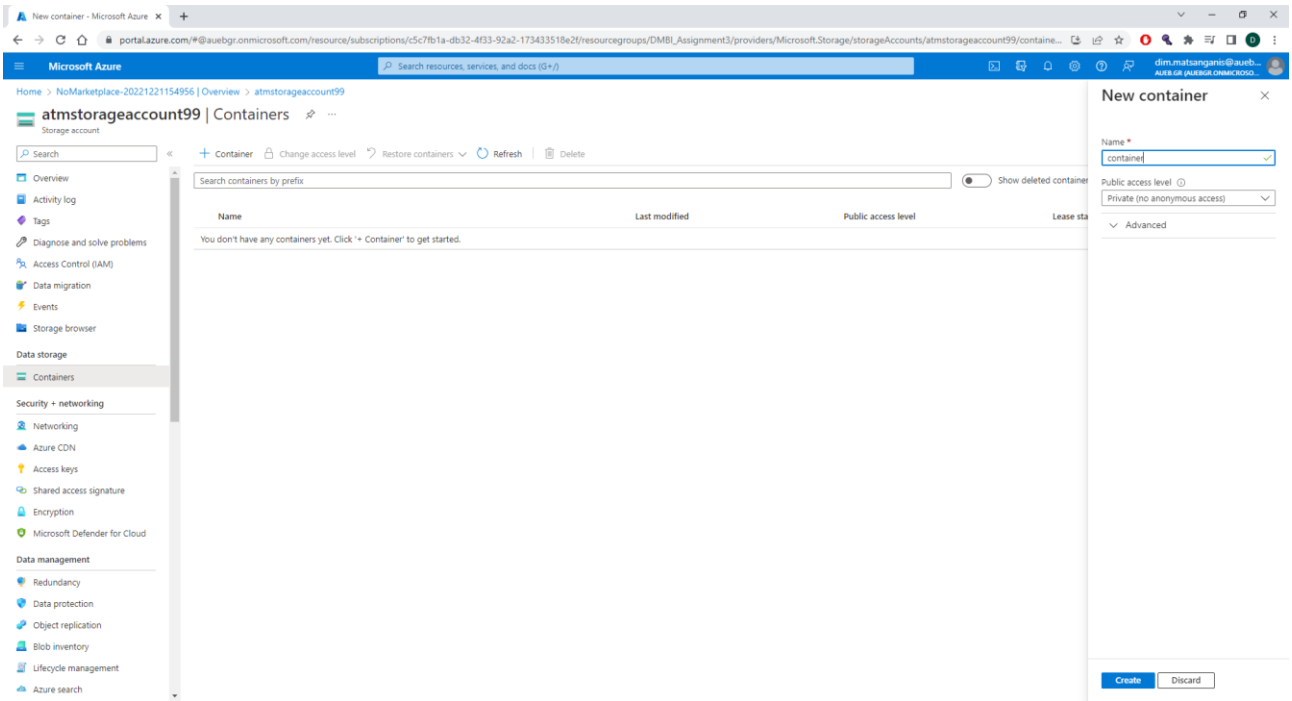


Figure 29: Creation of Container.

Upload Reference Data at Container

Then, we upload the given reference data to the previously created container.

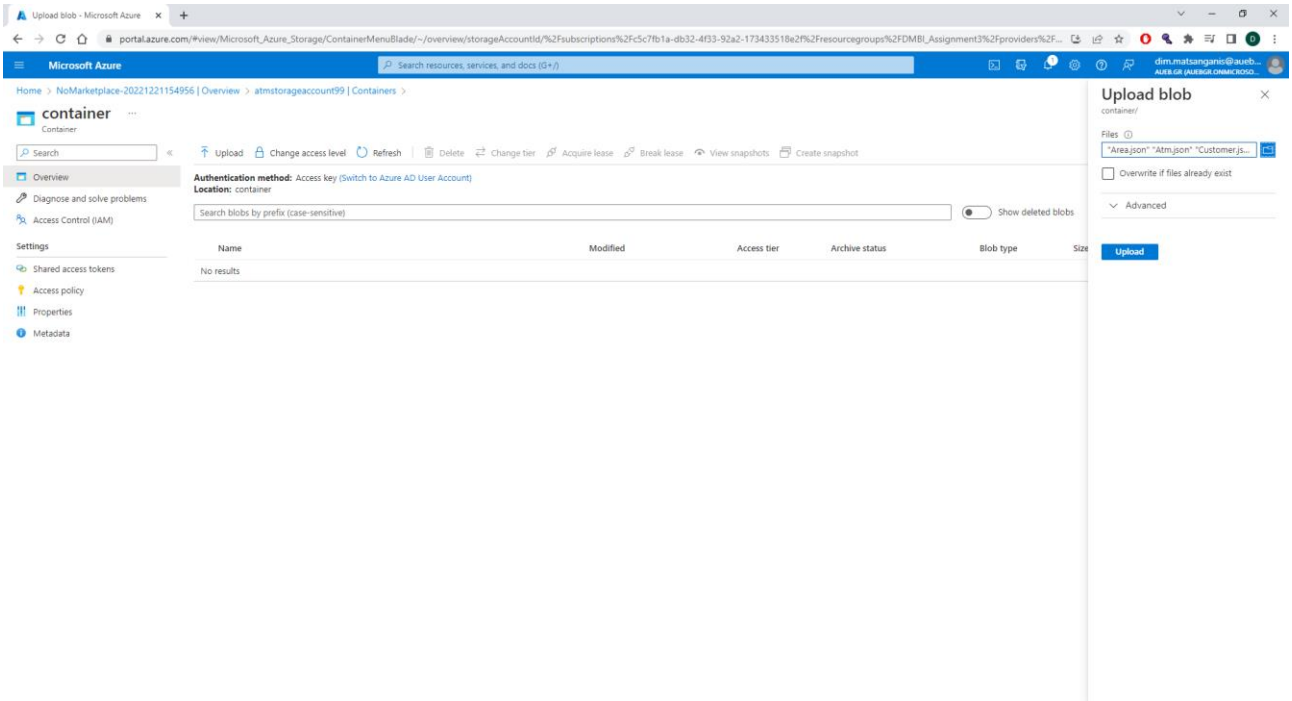


Figure 30: Upload of reference data files (JSON).

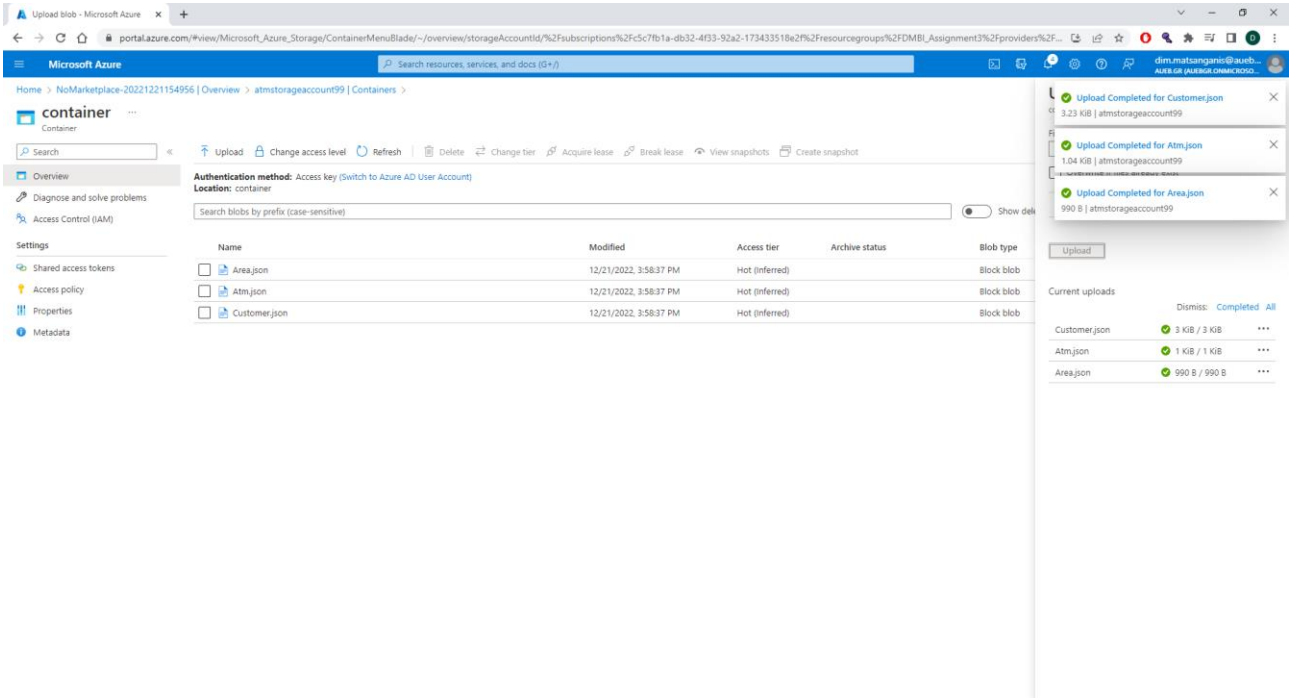


Figure 31: Successful upload of reference data files.

Creation of Stream Analytics Job

In this step, we need to create a Stream Jobs Analytics resource. The steps of this procedure are depicted below:

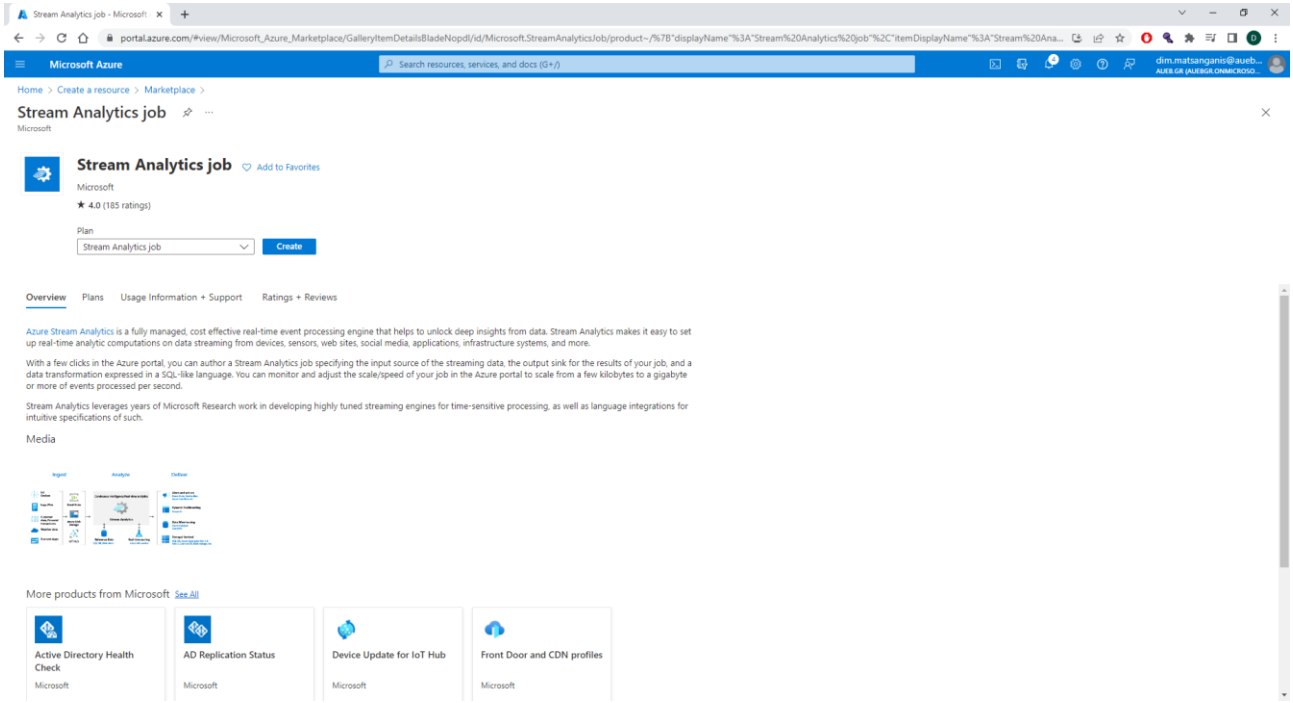


Figure 32: Creation of Stream Analytics Job (1/2).

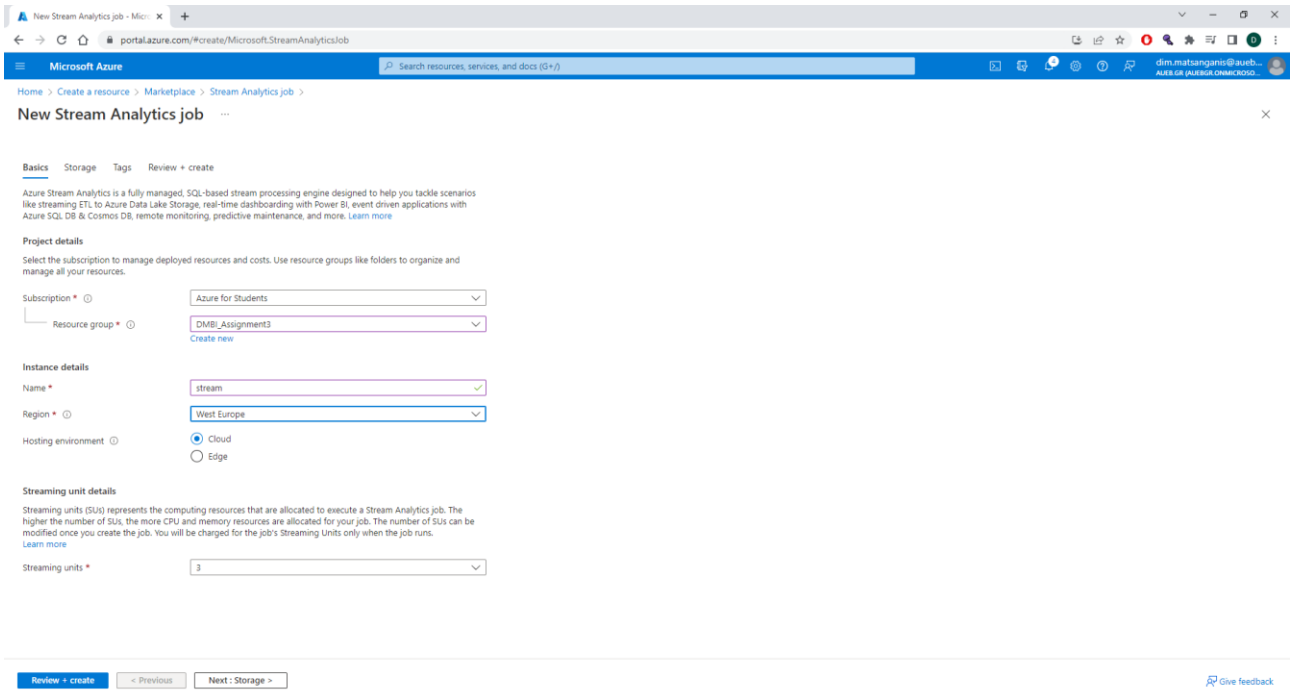


Figure 33: Creation of Stream Analytics Job (2/2).

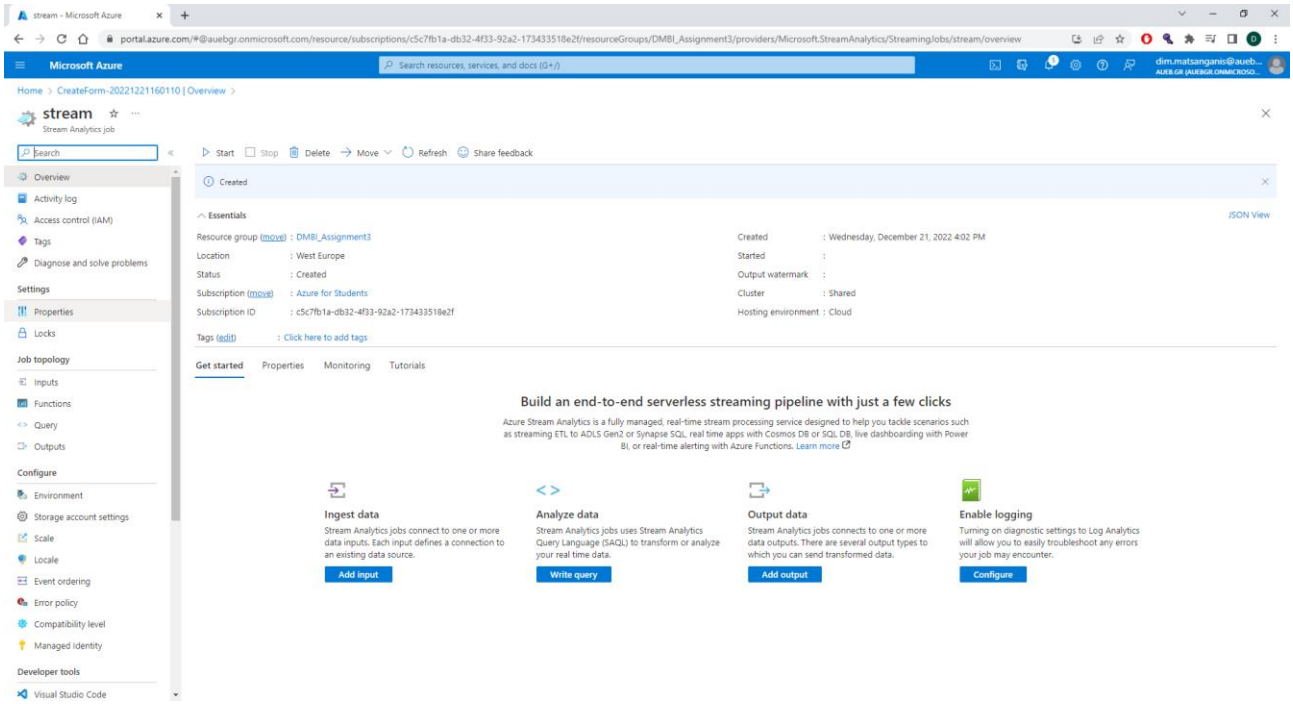


Figure 34: : Successful creation of Stream Analytics Job.

Add Inputs at Stream

Now we are going to create the input by combining the reference data files and the event hub.

- Event Hub Input

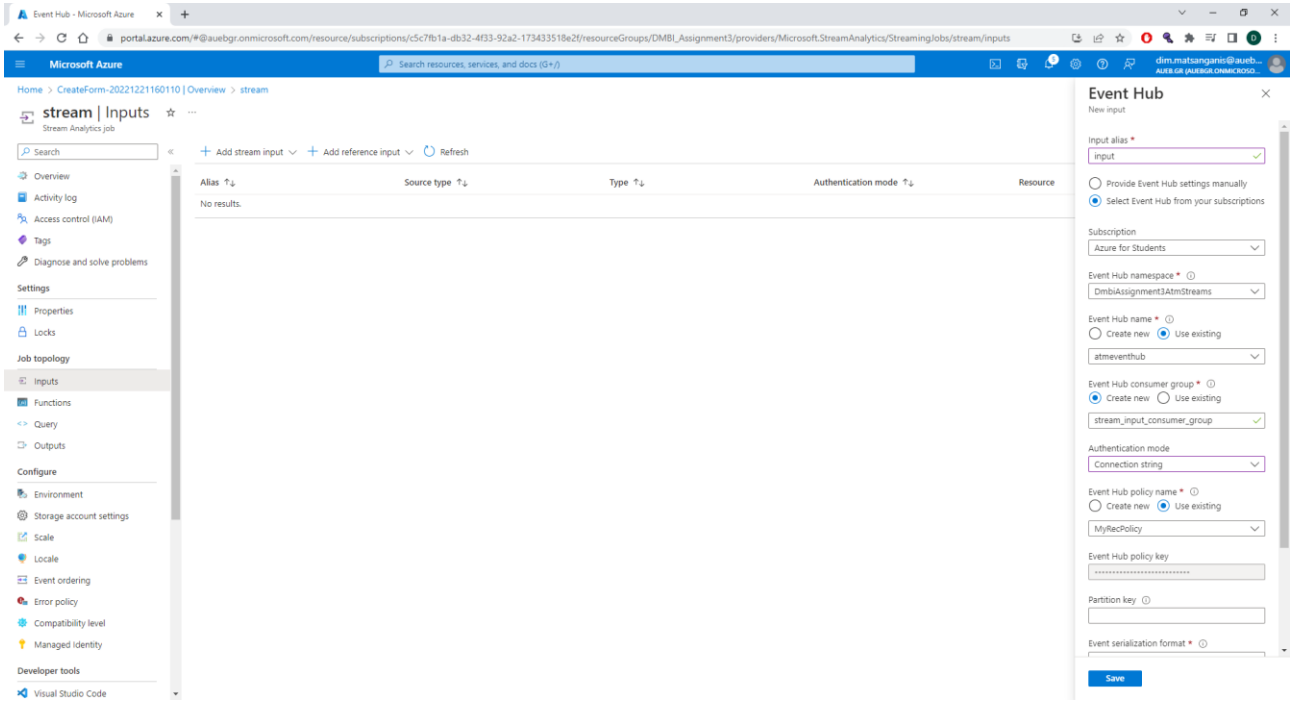


Figure 35: Input Stream Creation.

- Reference Data : Area

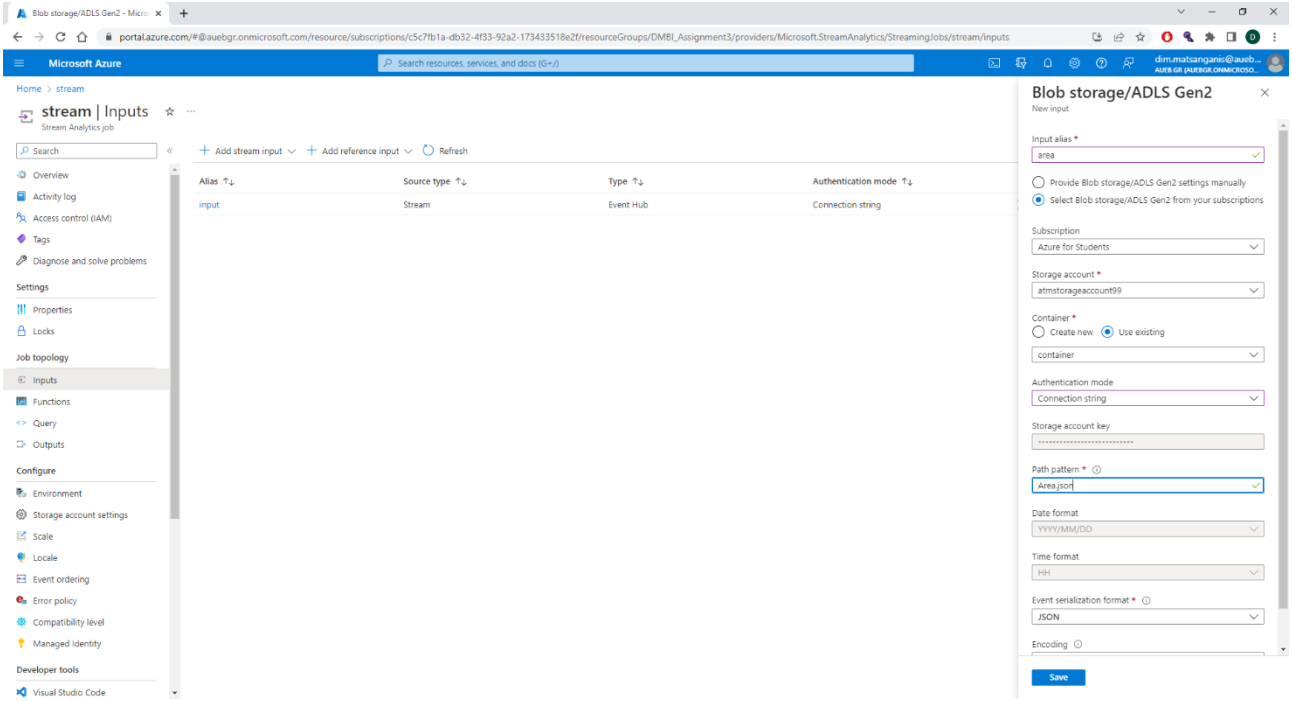


Figure 36: Reference Data Area.

- Reference Data : Atm

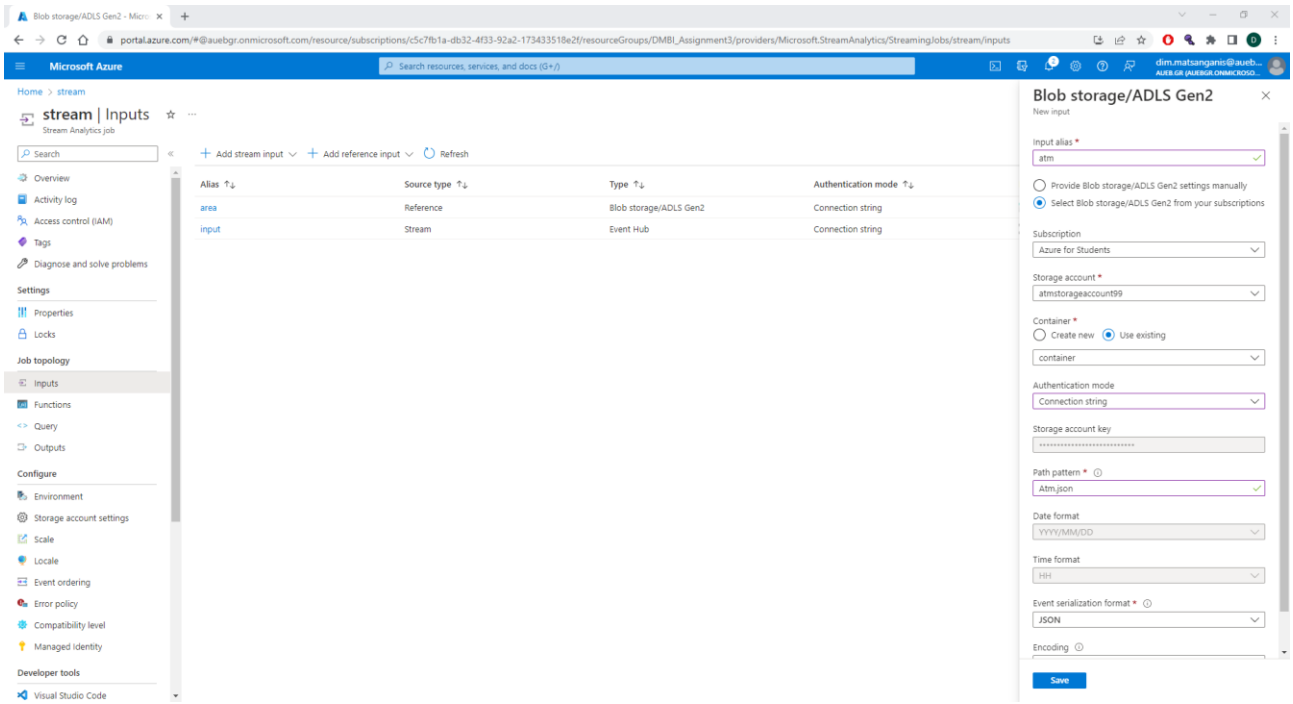


Figure 37: Reference Data Atm.

- Reference Data : Customer

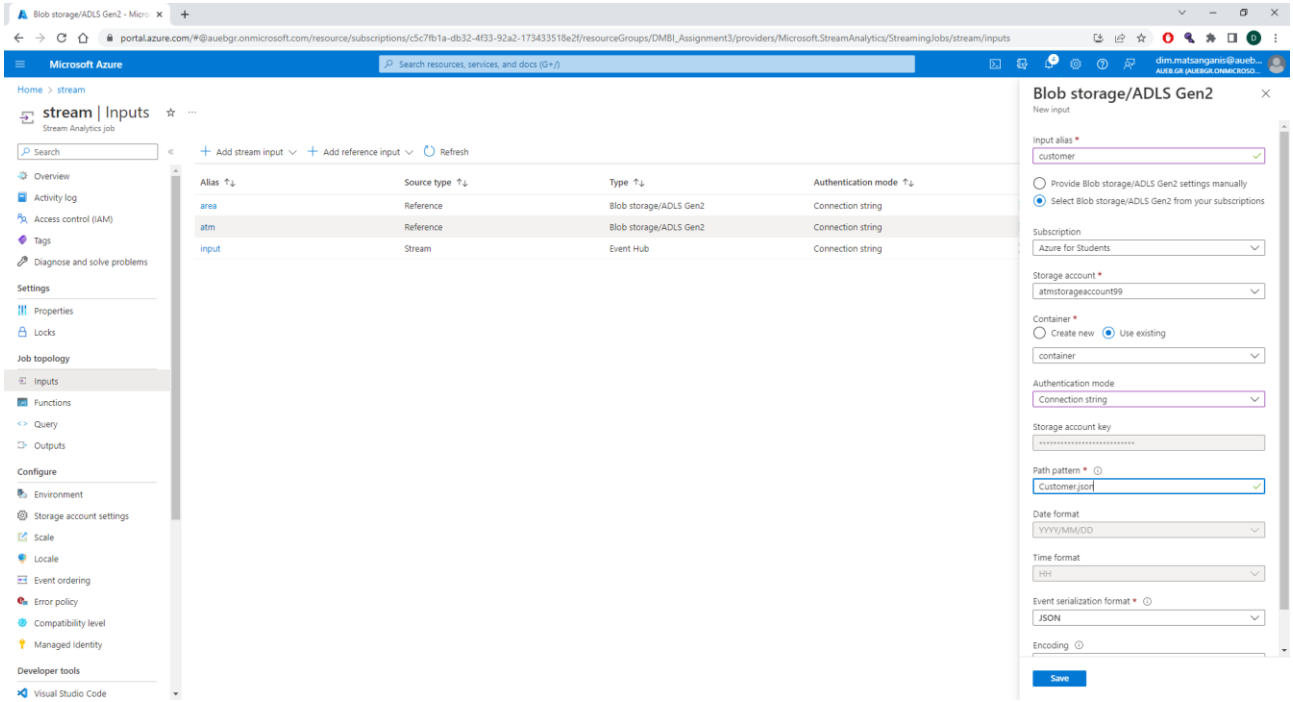


Figure 38: Reference Data Customer.

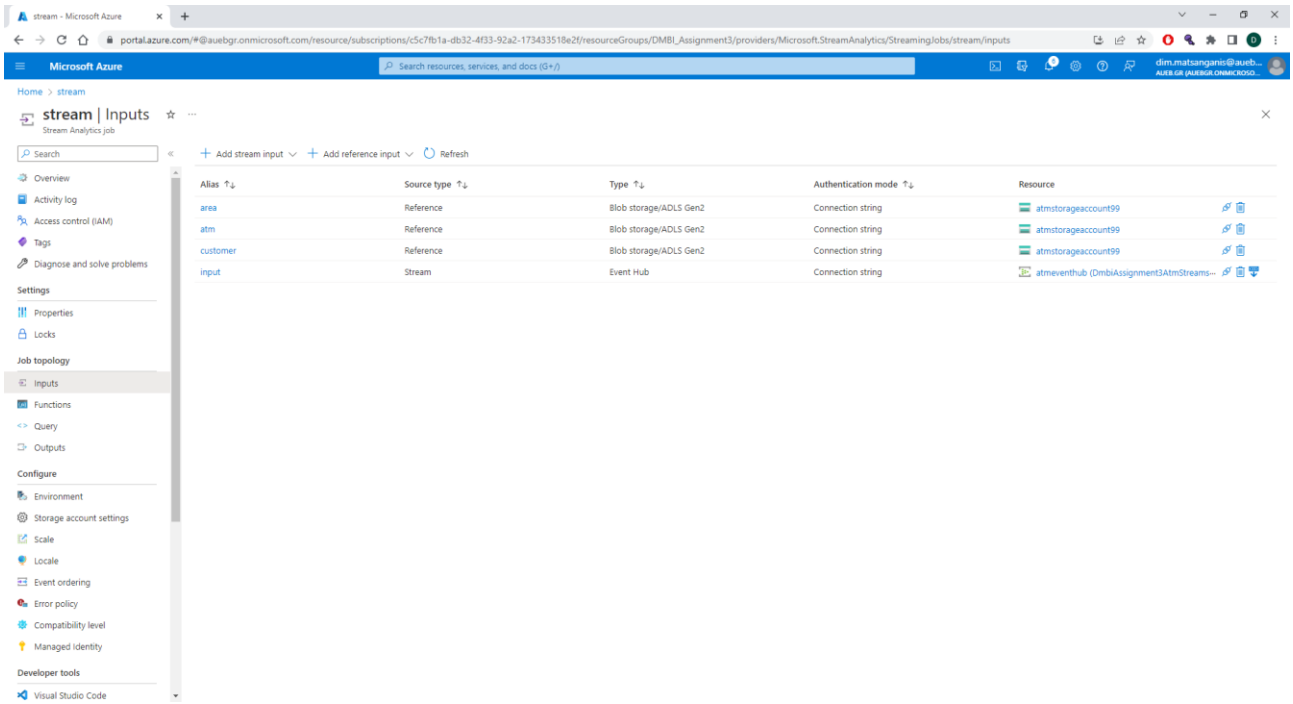


Figure 39: Successful creation of inputs and reference data.

Add Output at Stream

In addition to the previous steps we need to create an output for the stream.

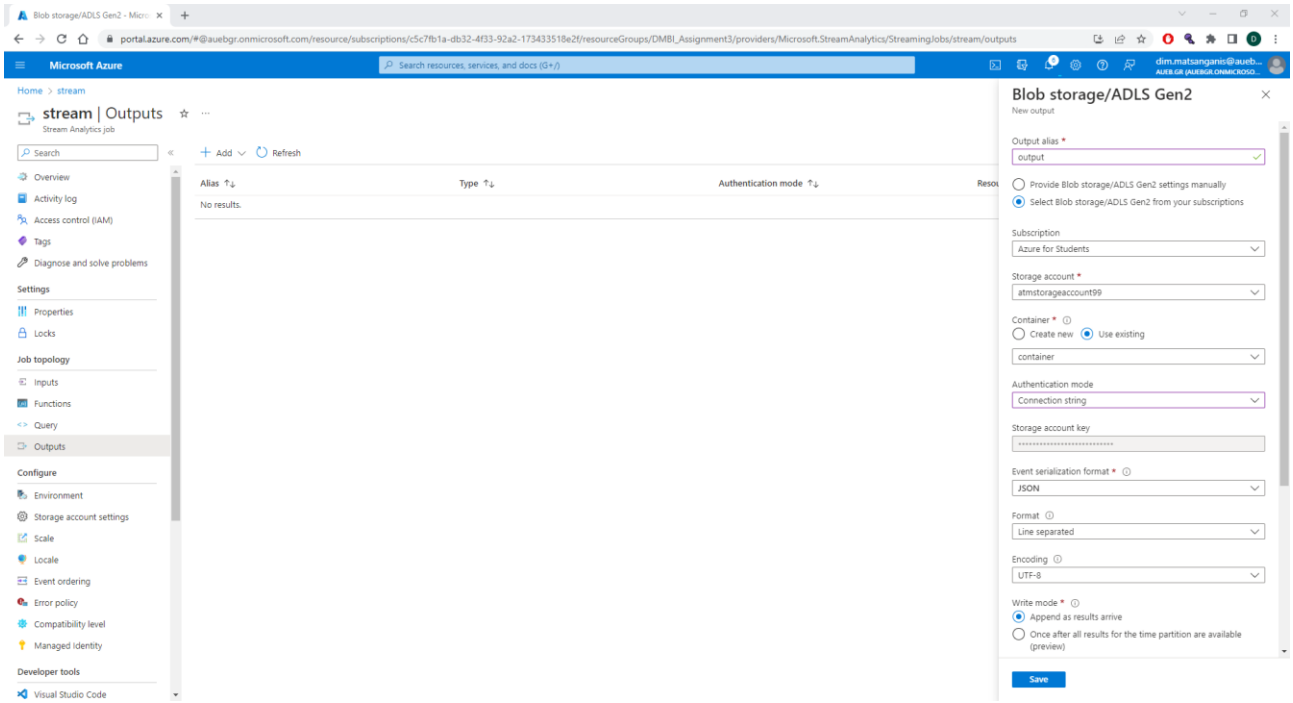


Figure 40: Stream's Output Creation.

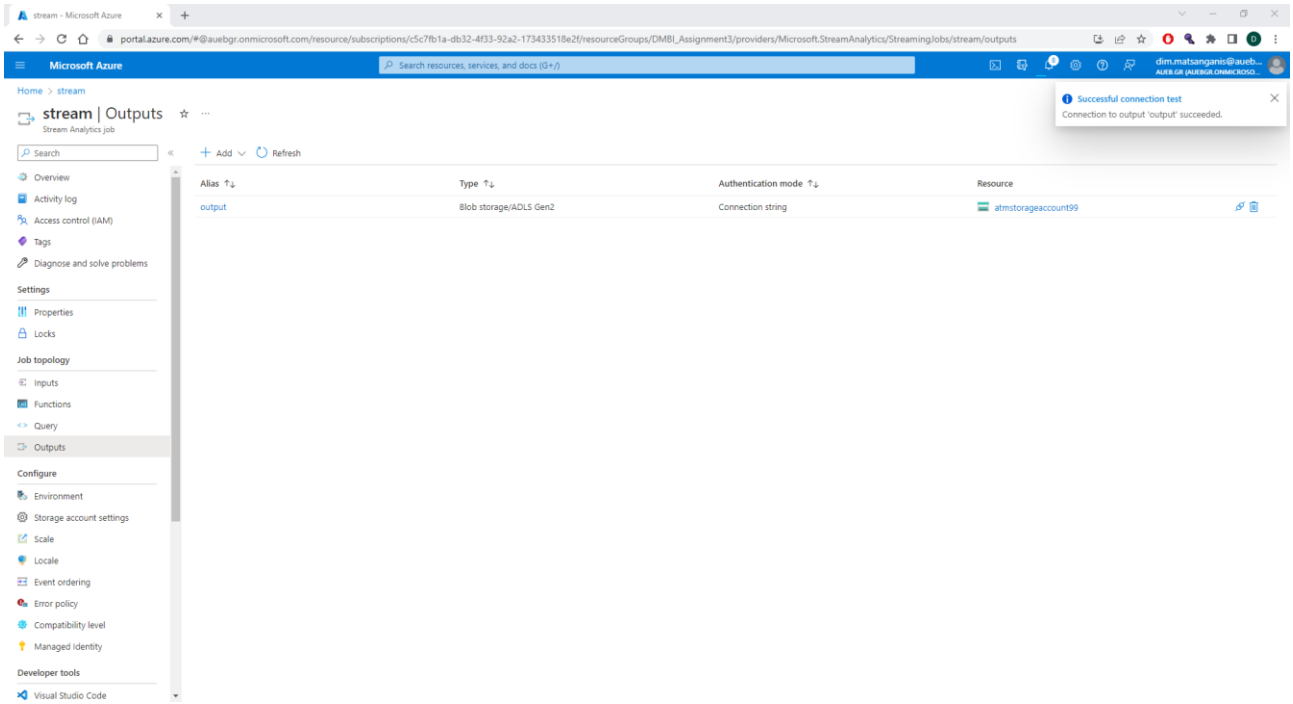
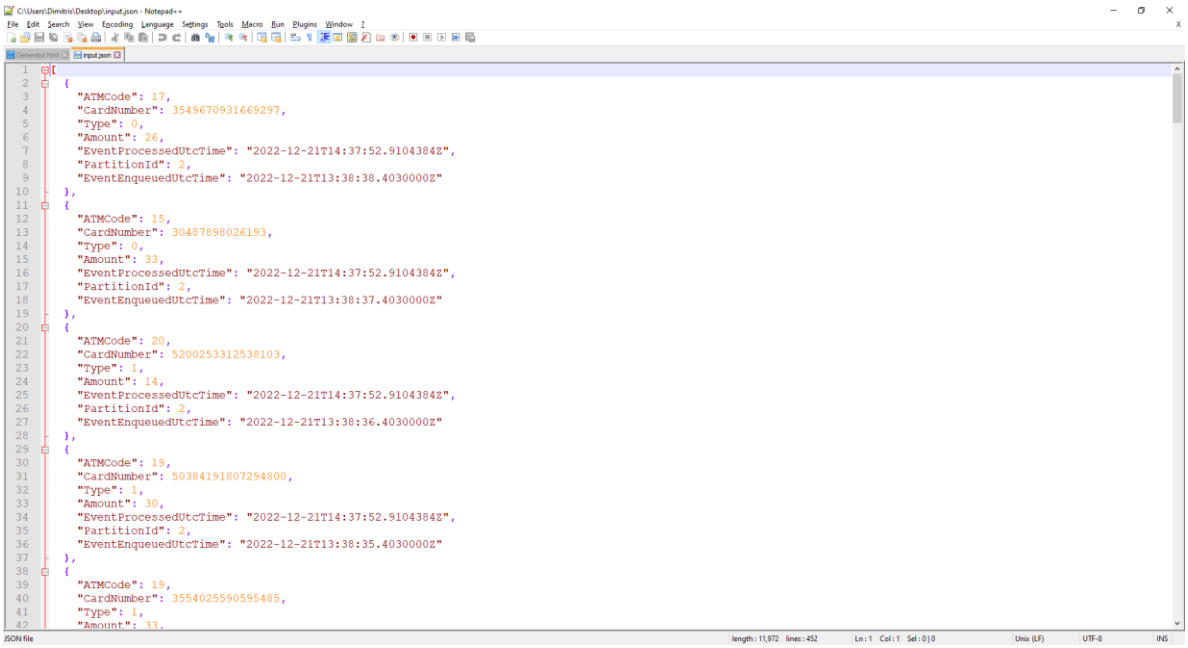


Figure 41: Stream's Output Successful Creation.

Sample data from input

The following picture depicts a sample portion of the data that have being collected from the input.




```
[
  {
    "ATMCode": 17,
    "CardNumber": 3549670931669297,
    "Type": 0,
    "Amount": 26,
    "EventProcessedUtcTime": "2022-12-21T14:37:52.9104384Z",
    "PartitionId": 2,
    "EventEnqueuedUtcTime": "2022-12-21T13:38:38.4030000Z"
  },
  {
    "ATMCode": 15,
    "CardNumber": 30487898026193,
    "Type": 0,
    "Amount": 33,
    "EventProcessedUtcTime": "2022-12-21T14:37:52.9104384Z",
    "PartitionId": 2,
    "EventEnqueuedUtcTime": "2022-12-21T13:38:37.4030000Z"
  },
  {
    "ATMCode": 20,
    "CardNumber": 5200253312538103,
    "Type": 1,
    "Amount": 14,
    "EventProcessedUtcTime": "2022-12-21T14:37:52.9104384Z",
    "PartitionId": 2,
    "EventEnqueuedUtcTime": "2022-12-21T13:38:36.4030000Z"
  },
  {
    "ATMCode": 19,
    "CardNumber": 5038419180729480 0,
    "Type": 1,
    "Amount": 30,
    "EventProcessedUtcTime": "2022-12-21T14:37:52.9104384Z",
    "PartitionId": 2,
    "EventEnqueuedUtcTime": "2022-12-21T13:38:35.4030000Z"
  },
  {
    "ATMCode": 19,
    "CardNumber": 3554025590595485,
    "Type": 1,
    "Amount": 33,
```


Figure 42: Sample Input's Data (Downloaded from Azure Stream).

Upload test data at Stream

Regarding the reference data we upload the test sample data and by doing so we can see the following results.

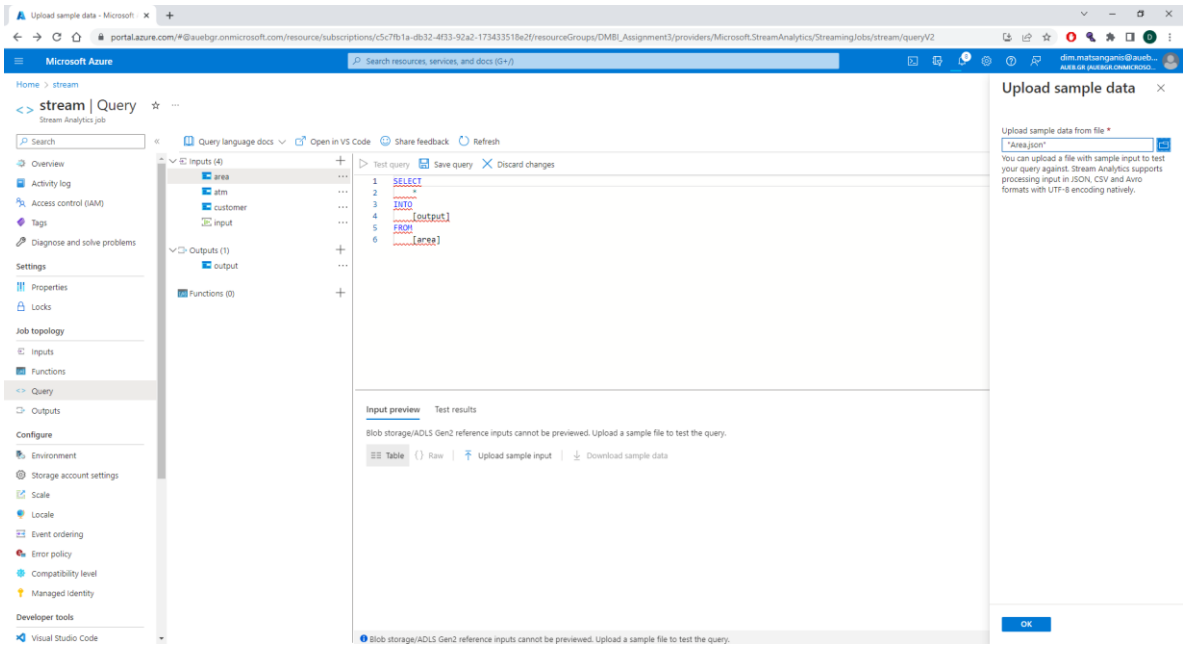


Figure 43: Upload test data for the reference data inputs.

Test Query and Results

In this section some test queries performed in order to explore the features and capabilities of the steam's query option offered by MS Azure. One of them is presented below:

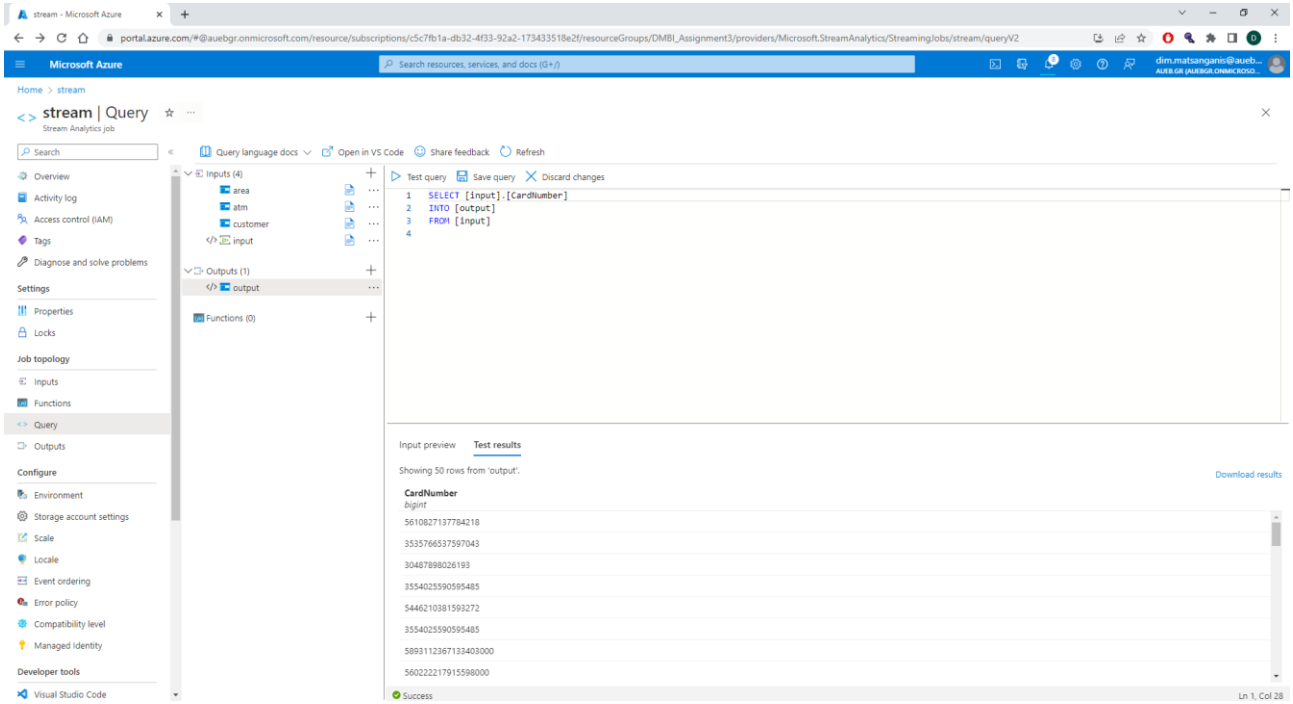


Figure 44: Test Query.

Start Stream

Now we select to start the stream in order to check the above tested query's results.

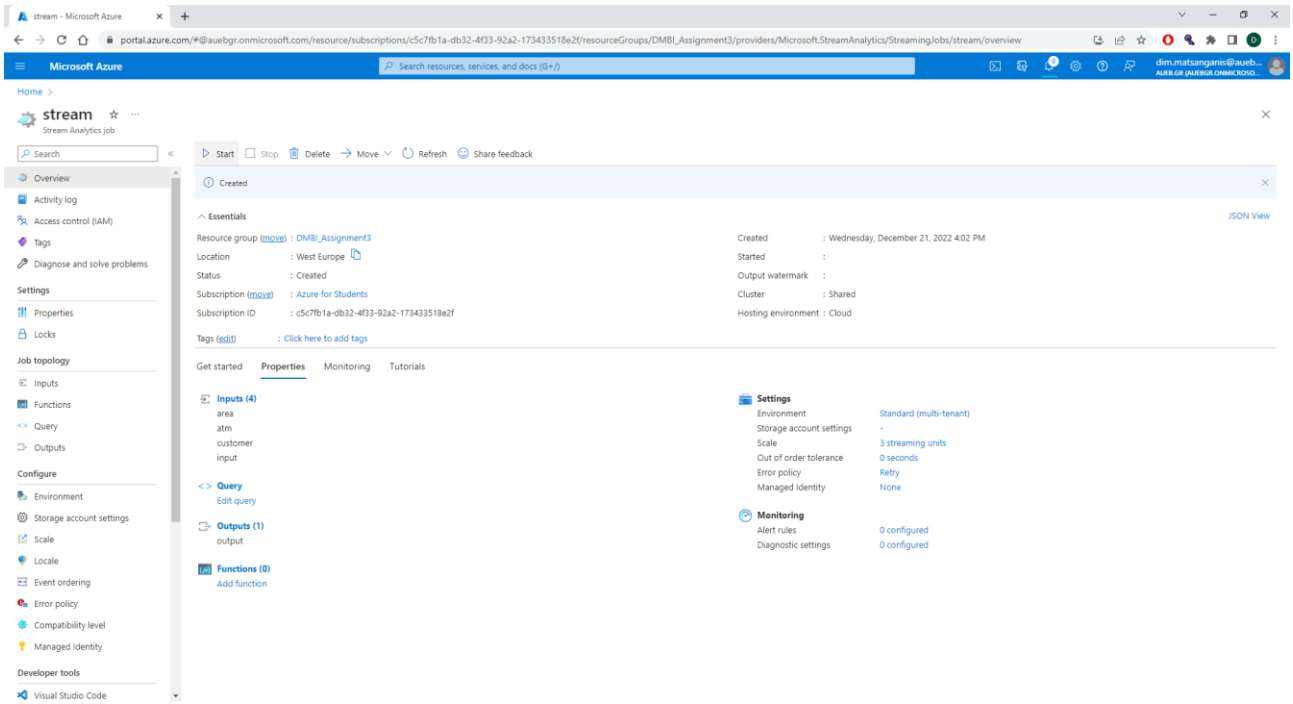


Figure 45: Start Stream.

Blob output of stream

The image below depicts the output of the aforementioned query.

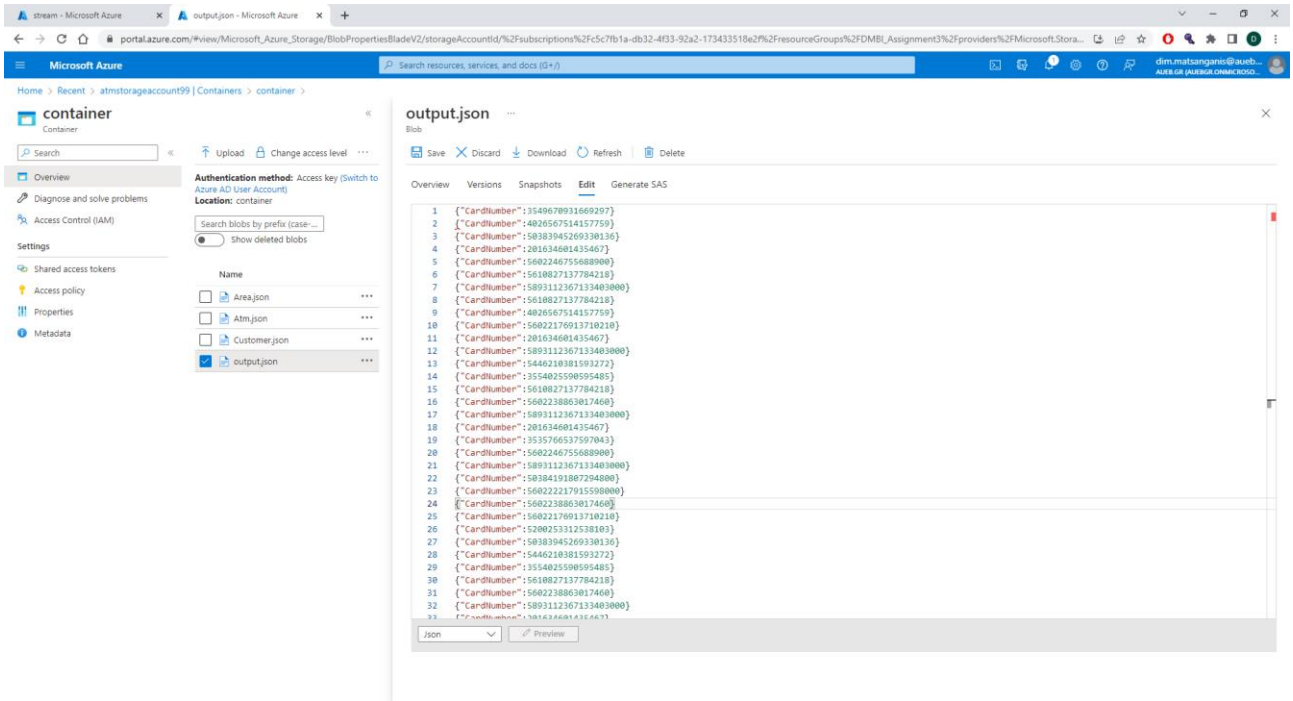


Figure 46: Test Query's Output

Queries

In this part of our assignment, we are going to describe the input and the output of all the wanted queries.

Query 1

Show the total “Amount” of “Type = 0” transactions at “ATM Code = 21” of the last 10 minutes. Repeat as new events keep flowing in (use a sliding window).

Input

```
SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY Type, ATMCode, SlidingWindow(minute,10)
HAVING Type=0 AND ATMCode=21
```

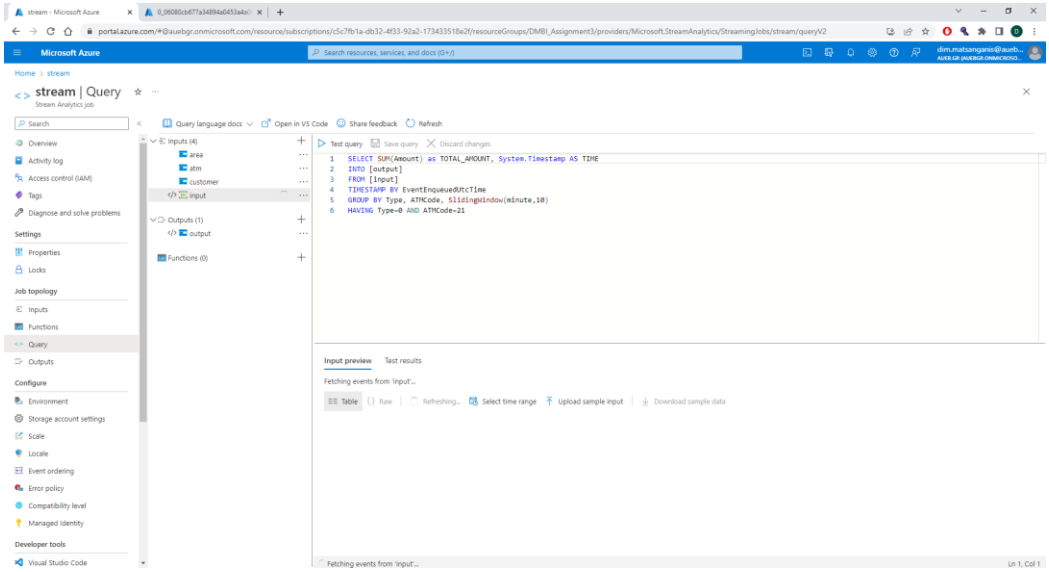


Figure 47: Query 1.

Output

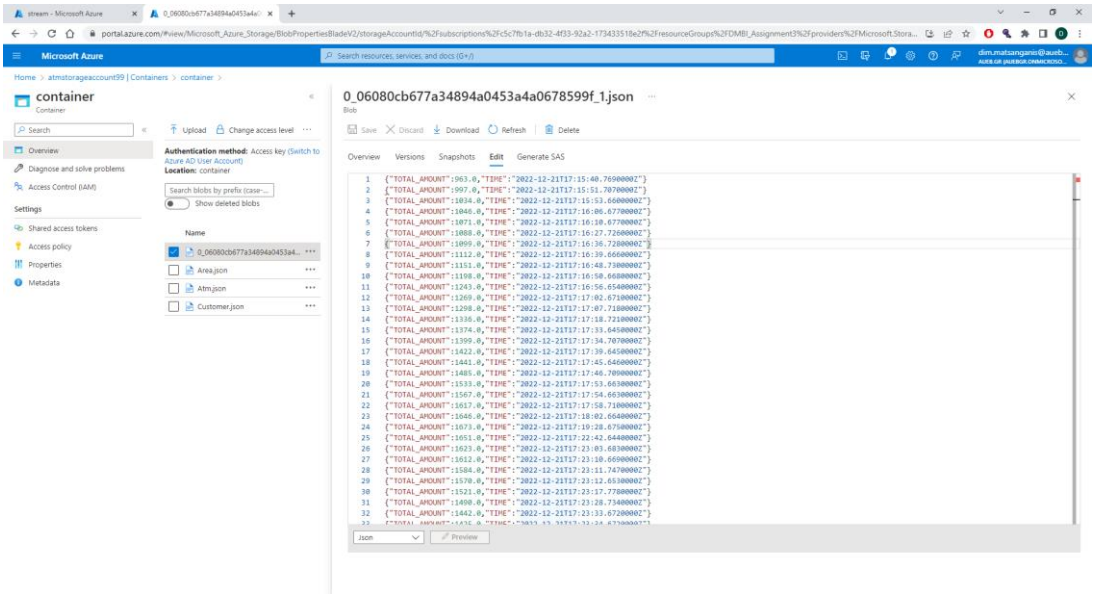


Figure 48: Query 1 Output.

Query 2

Show the total “Amount” of “Type = 1” transactions at “ATM Code = 21” of the last hour. Repeat once every hour (use a tumbling window).

Input

```
1.  SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
2.  INTO [output]
3.  FROM [input]
4.  TIMESTAMP BY EventEnqueuedUtcTime
5.  GROUP BY Type, ATMCode, TumblingWindow(hour,1)
6.  HAVING Type=1 AND ATMCode=21
```

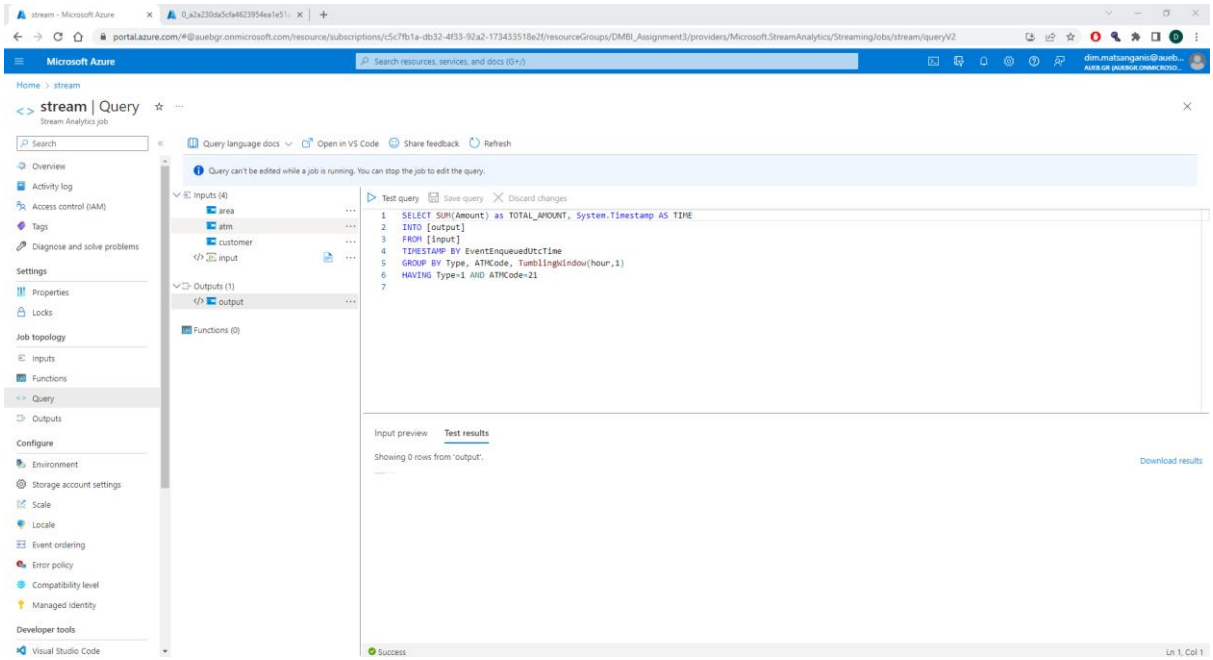


Figure 49: Query 2.

Output

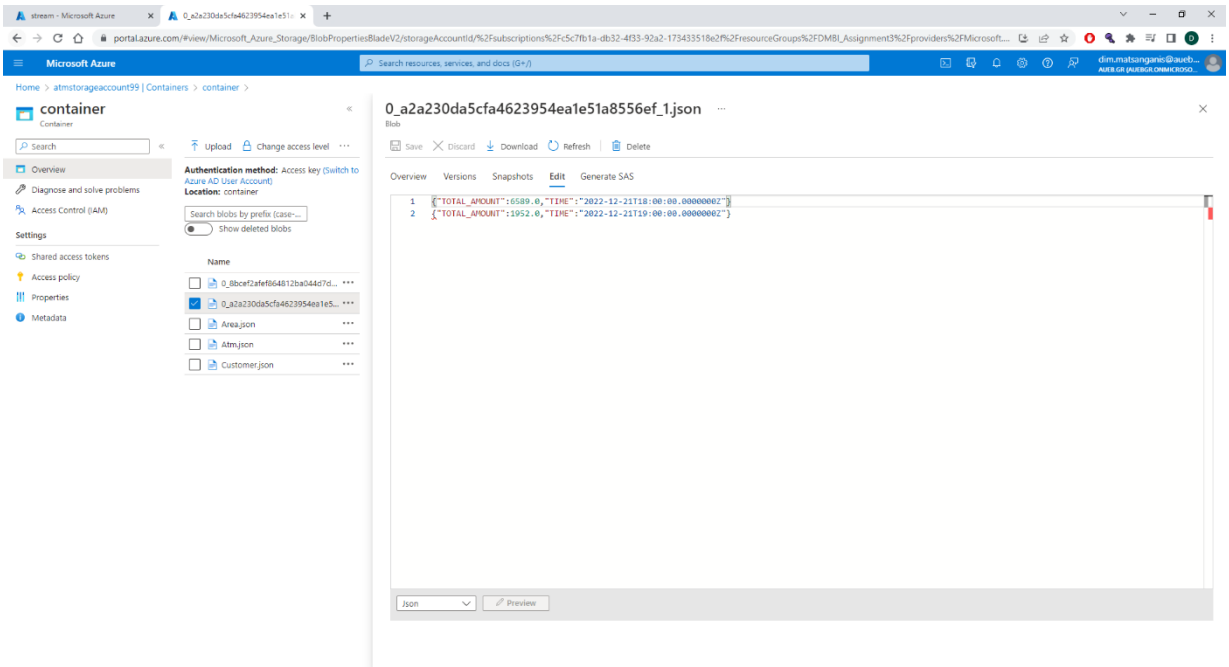


Figure 50: Query 2 Output.

Query 3

Show the total "Amount" of "Type = 1" transactions at "ATM Code = 21" of the last hour. Repeat once every 30 minutes (use a hopping window).

Input

```
SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
WHERE Type=1 AND ATMCode=21
GROUP BY ATMCode, HoppingWindow(minute,60,30)
```

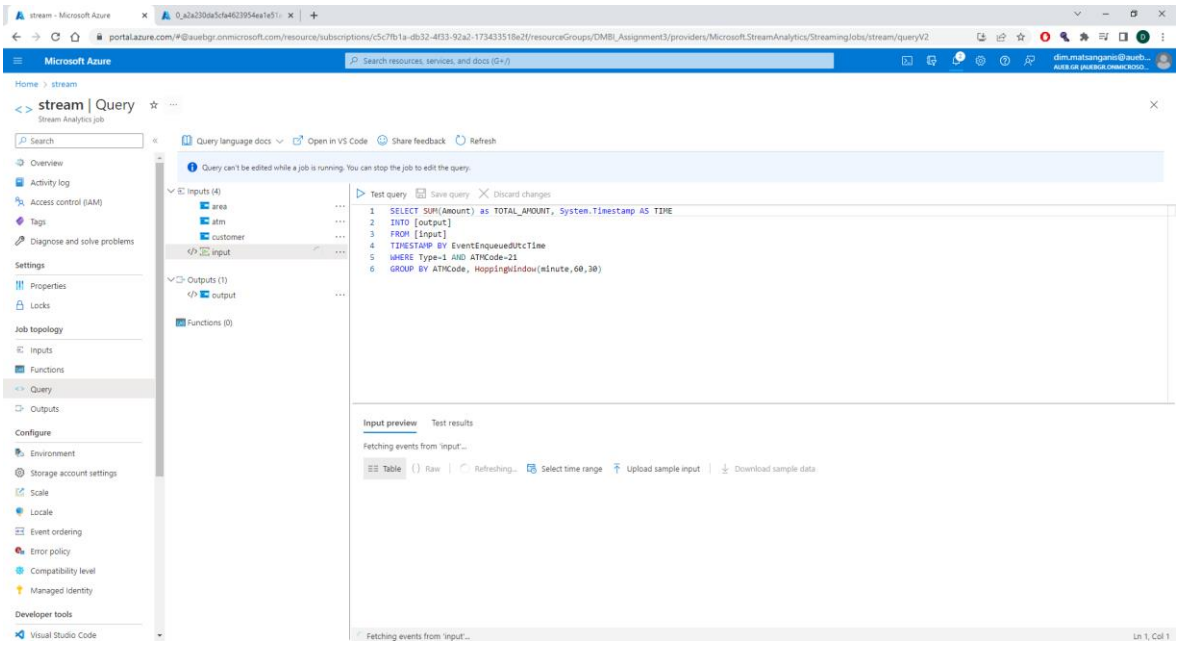


Figure 51: Query 3.

Output

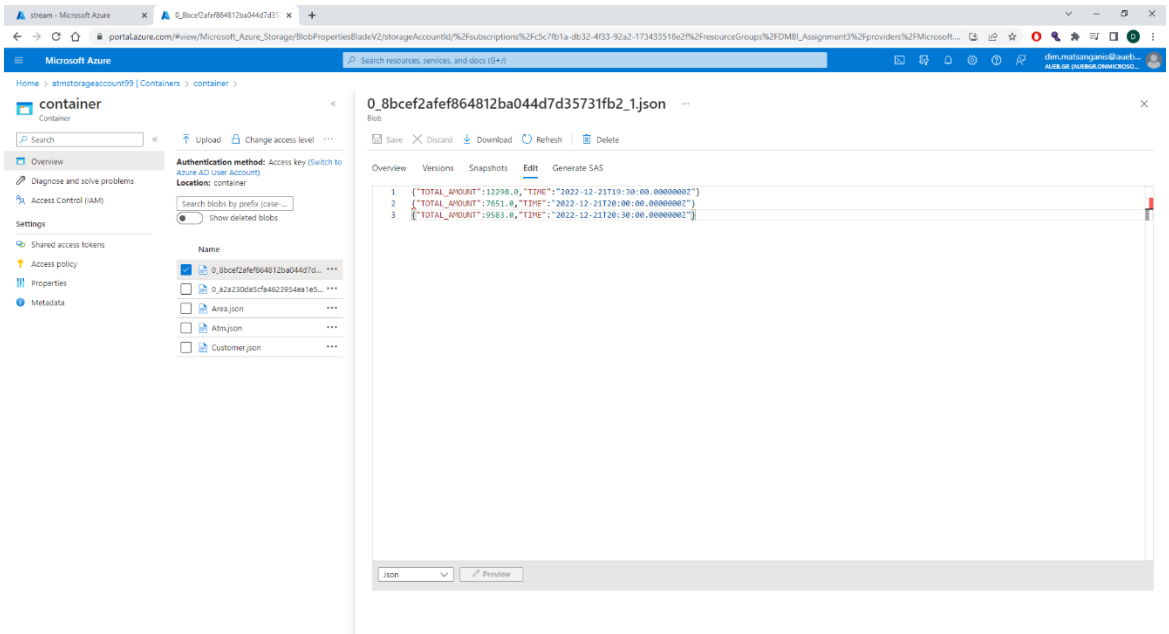


Figure 52: Query 3 Output.

Query 4

Show the total "Amount" of "Type = 1" transactions per "ATM Code" of the last one hour (use a sliding window).

Input

```
SELECT SUM(Amount) as TOTAL_AMOUNT, ATMCode, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
WHERE Type =1
GROUP BY ATMCode, SlidingWindow(minute,60)
```

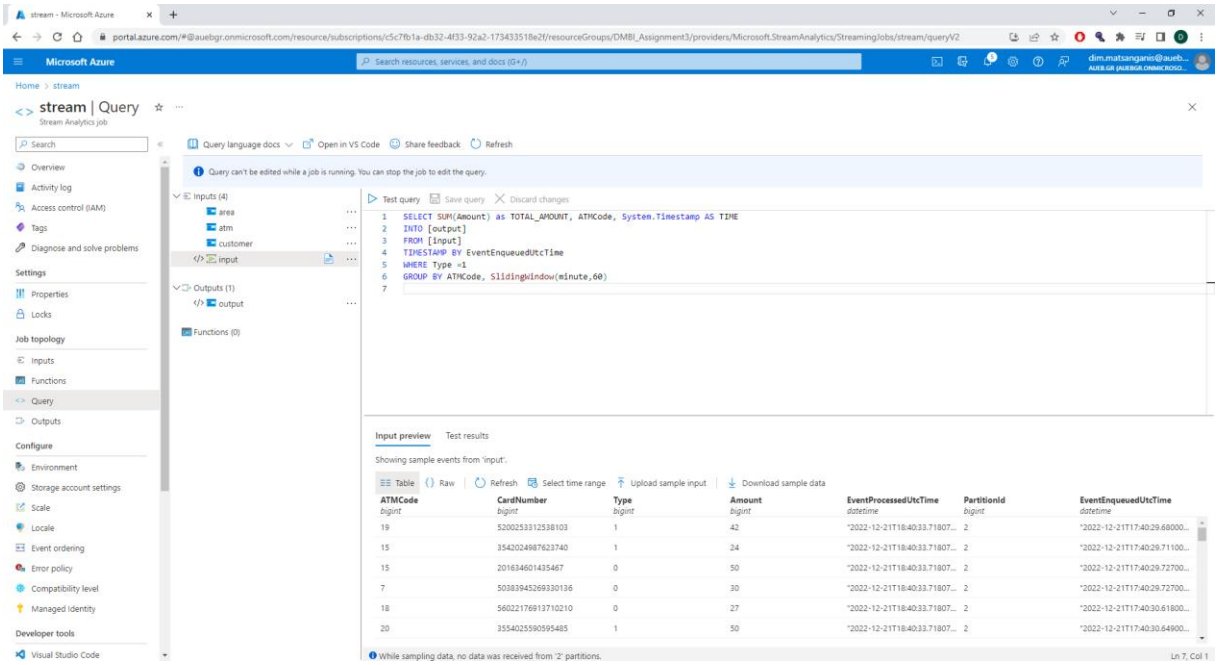


Figure 53: Query 4.

Output

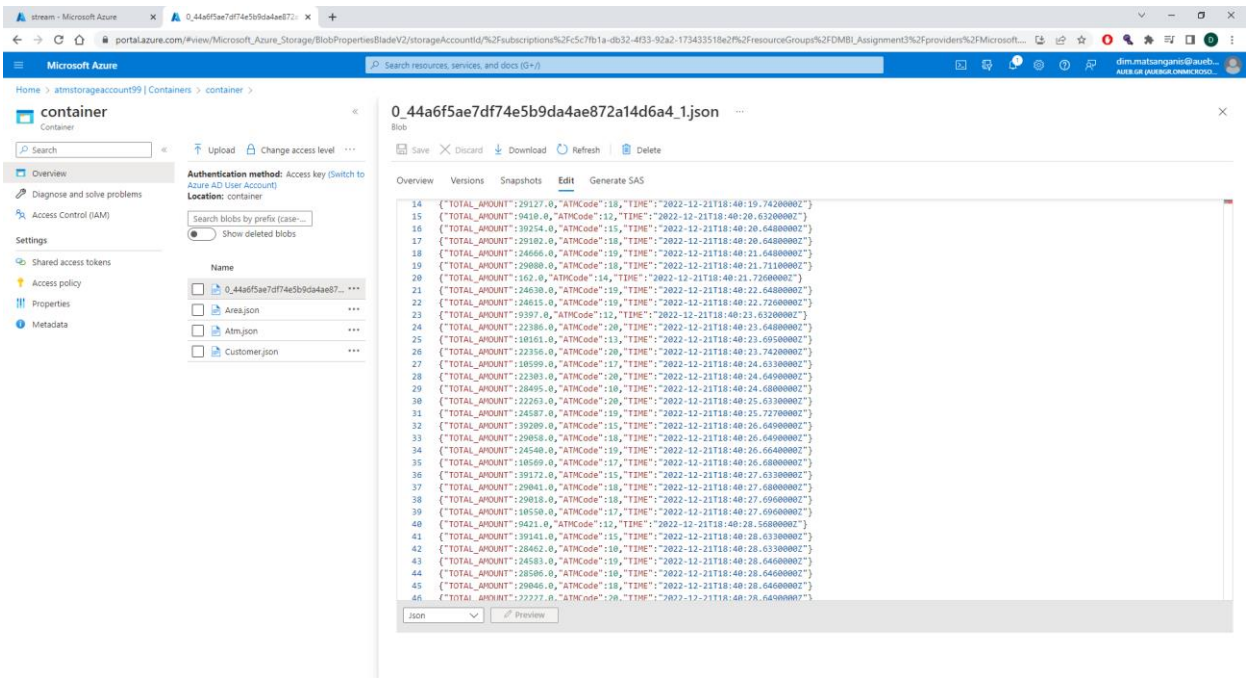


Figure 54: Query 4 Output.

Query 5

Show the total "Amount" of "Type = 1" transactions per "Area Code" of the last hour. Repeat once every hour (use a tumbling window).

Input

```
SELECT SUM([input].[Amount]) AS TOTAL_AMOUNT, [atm].[area_code], System.Timestamp
AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
INNER JOIN [atm]
ON [input].[ATMCode]=[atm].[atm_code]
WHERE [input].[Type]=1
GROUP BY [atm].[area_code], TumblingWindow(hour,1)
```

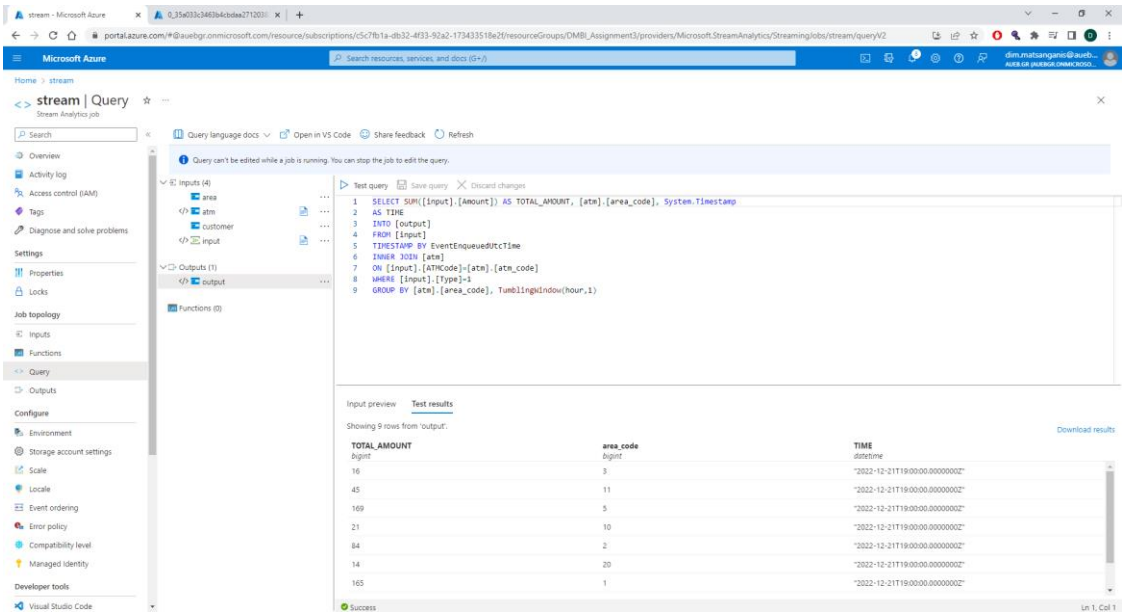


Figure 55: Query 5.

Output

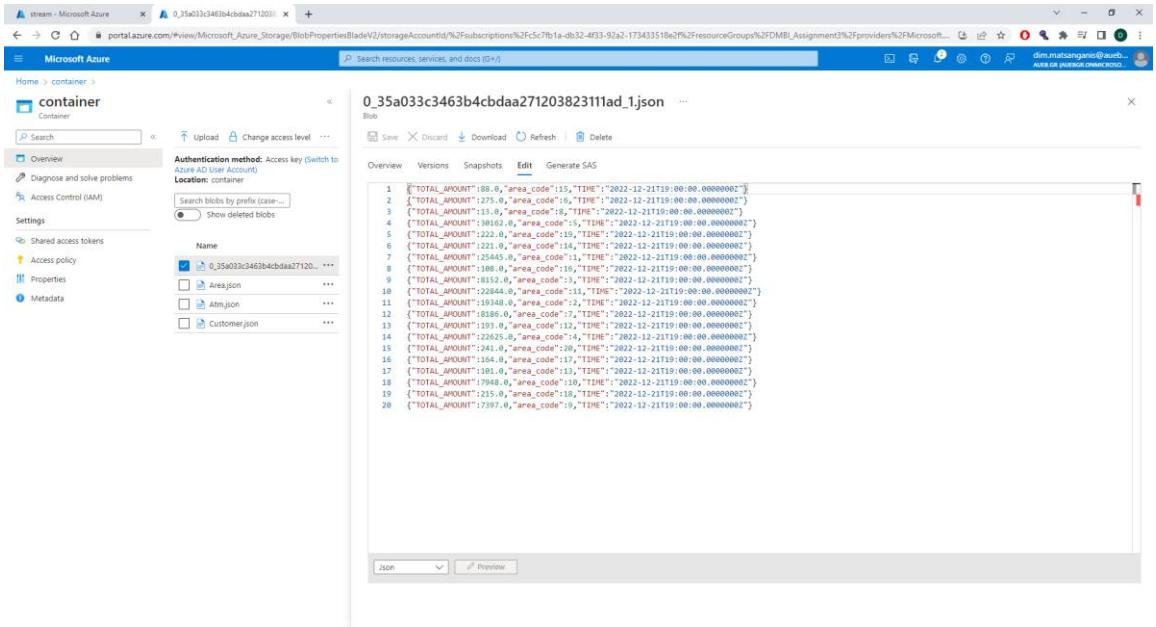


Figure 56: Query 5 Output.

Query 6

Show the total "Amount" per ATM's "City" and Customer's "Gender" of the last hour. Repeat once every hour (use a tumbling window).

Input

```
SELECT SUM([input].[Amount]) AS TOTAL_AMOUNT, [area].[area_city] AS CITY,
[customer].[gender] AS GENDER, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
INNER JOIN [customer]
ON [input].[CardNumber]=[customer].[card_number]
INNER JOIN [atm]
ON [input].[ATMCode]=[atm].[atm_code]
INNER JOIN [area]
ON [atm].[area_code]=[area].[area_code]
GROUP BY [customer].[gender], [area].[area_city], TumblingWindow(hour,1)
```

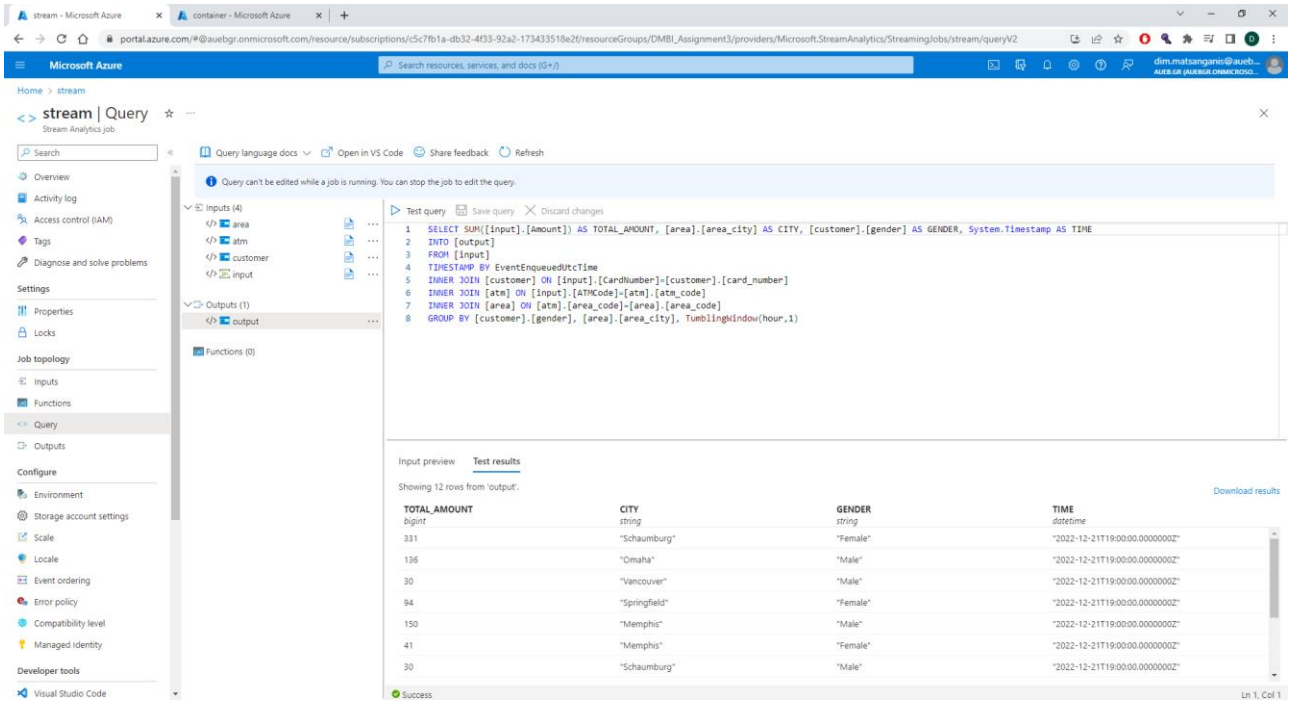


Figure 57: Query 6.

Output

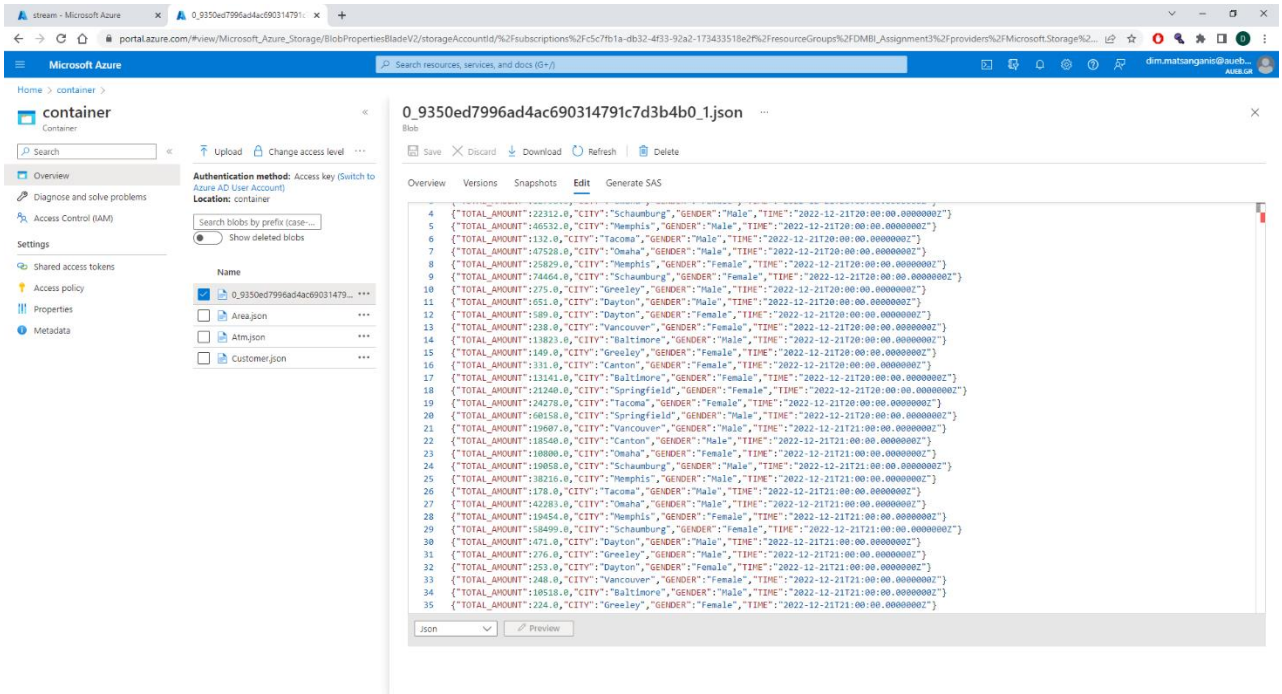


Figure 58: Query 6 Output.

Query 7

Alert (Do a simple SELECT "1") if a Customer has performed two transactions of "Type = 1" in a window of an hour (use a sliding window).

Input

```
SELECT 1 AS ALERT,[customer].[card_number], [customer].[first_name],[customer].[last_name],
COUNT(*) AS TRANSACTIONS, System.Timestamp AS TIME
INTO [output]
FROM [input]
INNER JOIN [customer] ON [input].[CardNumber]=[customer].[card_number]
WHERE [input].[Type]=1
GROUP BY [customer].[card_number],[customer].[first_name],
[customer].[last_name],SlidingWindow(hour,1)
HAVING TRANSACTIONS=2
```

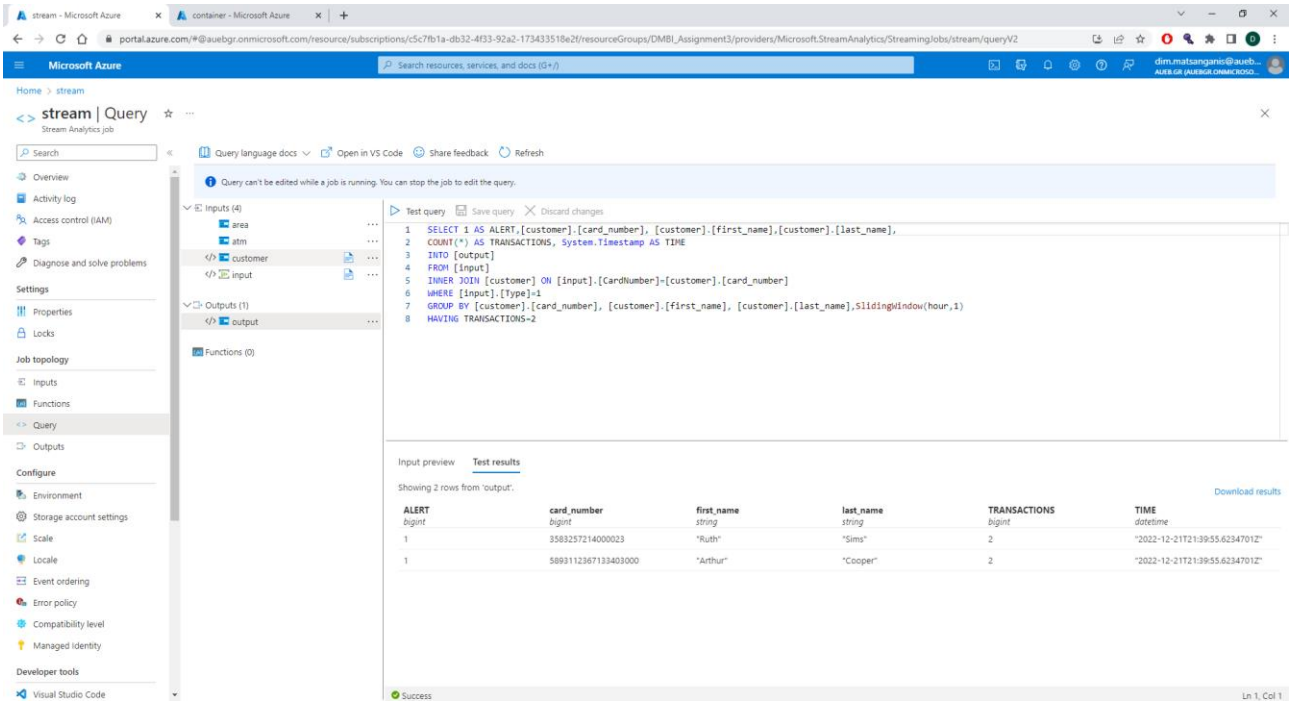


Figure 59: Query 7.

Output

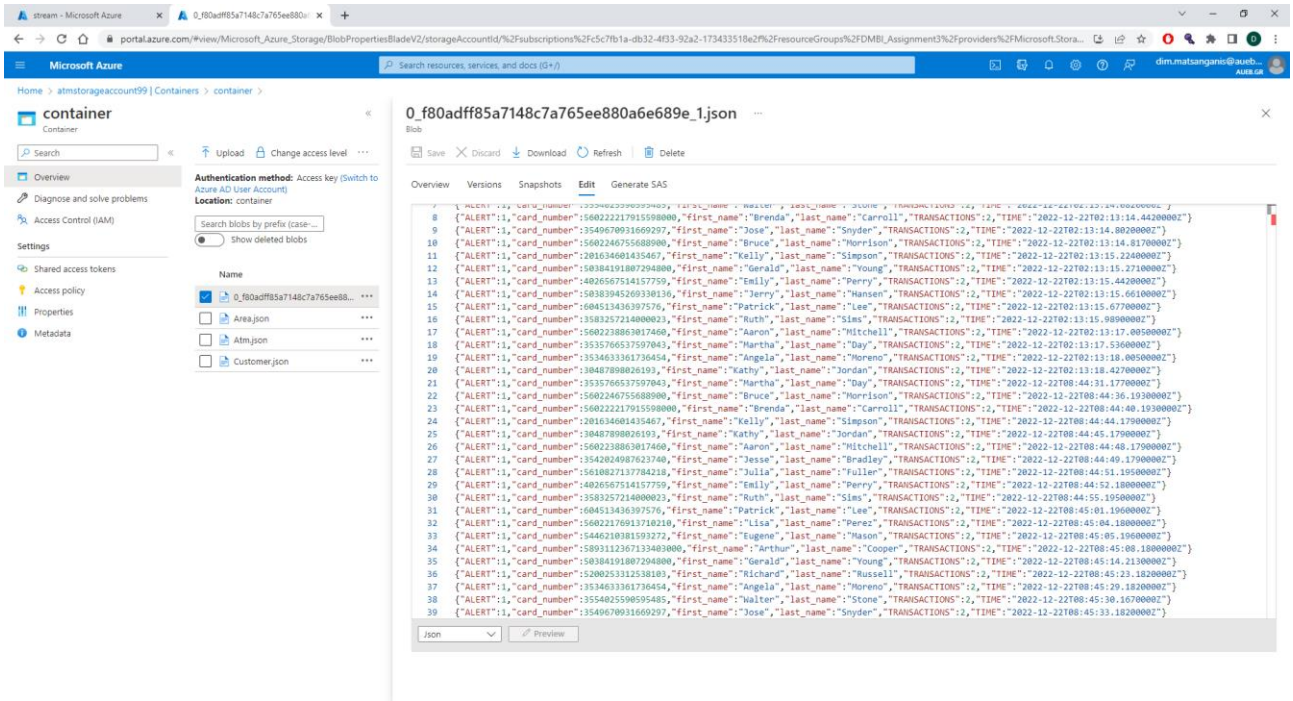


Figure 60: Query 7 Output.

Query 8

Alert (Do a simple SELECT "1") if the "Area Code" of the ATM of the transaction is not the same as the "Area Code" of the "Card Number" (Customer's Area Code) - (use a sliding window) .

Input

```
SELECT 1 AS ALERT,[customer].[card_number], [customer].[area_code] AS
CUSTOMER_AREA,[atm].[area_code] AS ATM_AREA, COUNT(*) AS TRANSACTIONS,
System.Timestamp AS TIME
INTO [output]
FROM [input]
INNER JOIN [customer] ON [input].[CardNumber]=[customer].[card_number]
INNER JOIN [atm] ON [input].[ATMCode]=[atm].[atm_code]
WHERE [customer].[area_code]!=[atm].[area_code]
GROUP BY [customer].[card_number], [customer].[area_code], [atm].[area_code],
SlidingWindow(hour,1)
```

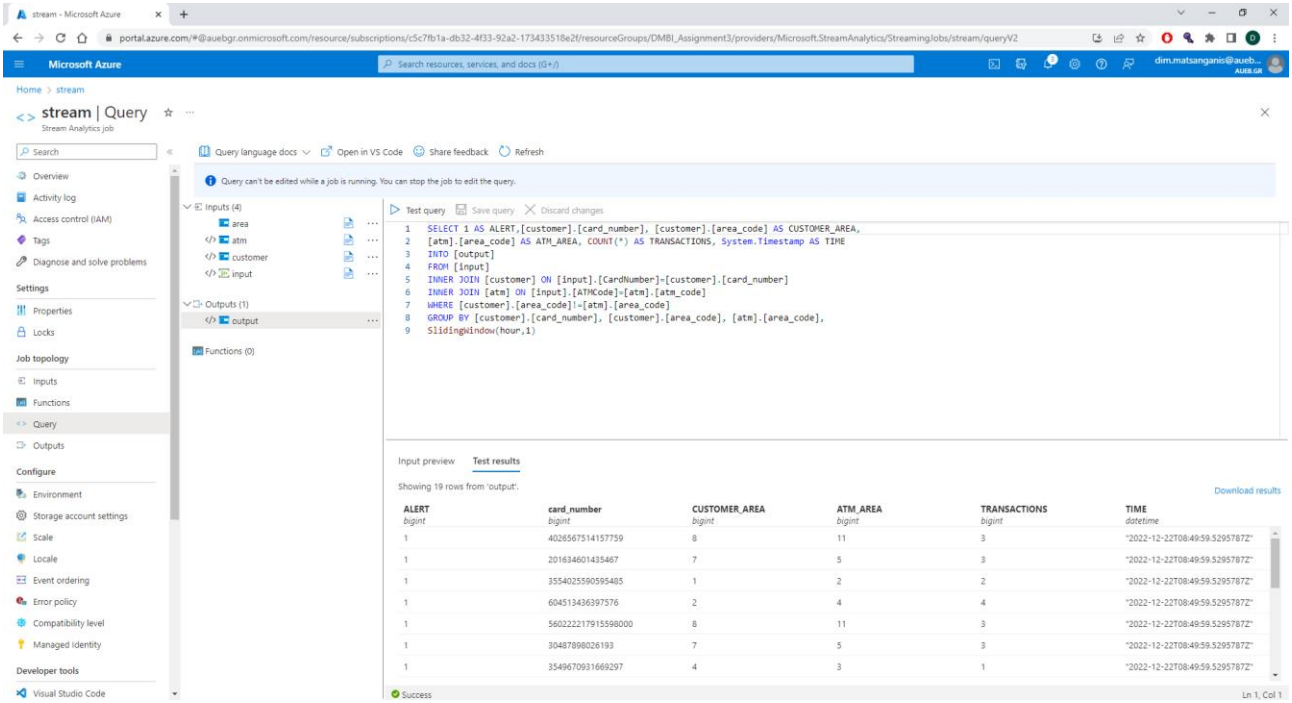


Figure 61: Query 8.

Output

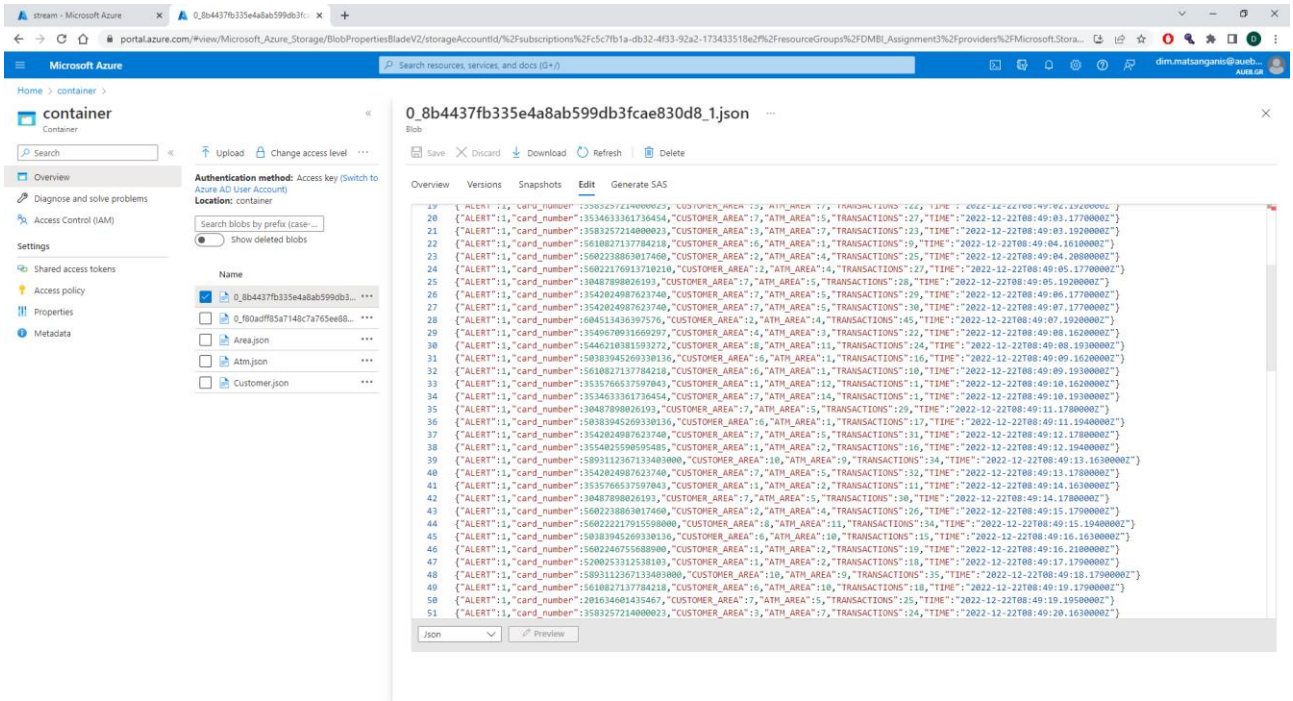


Figure 62: Query 8 Output (1/2).

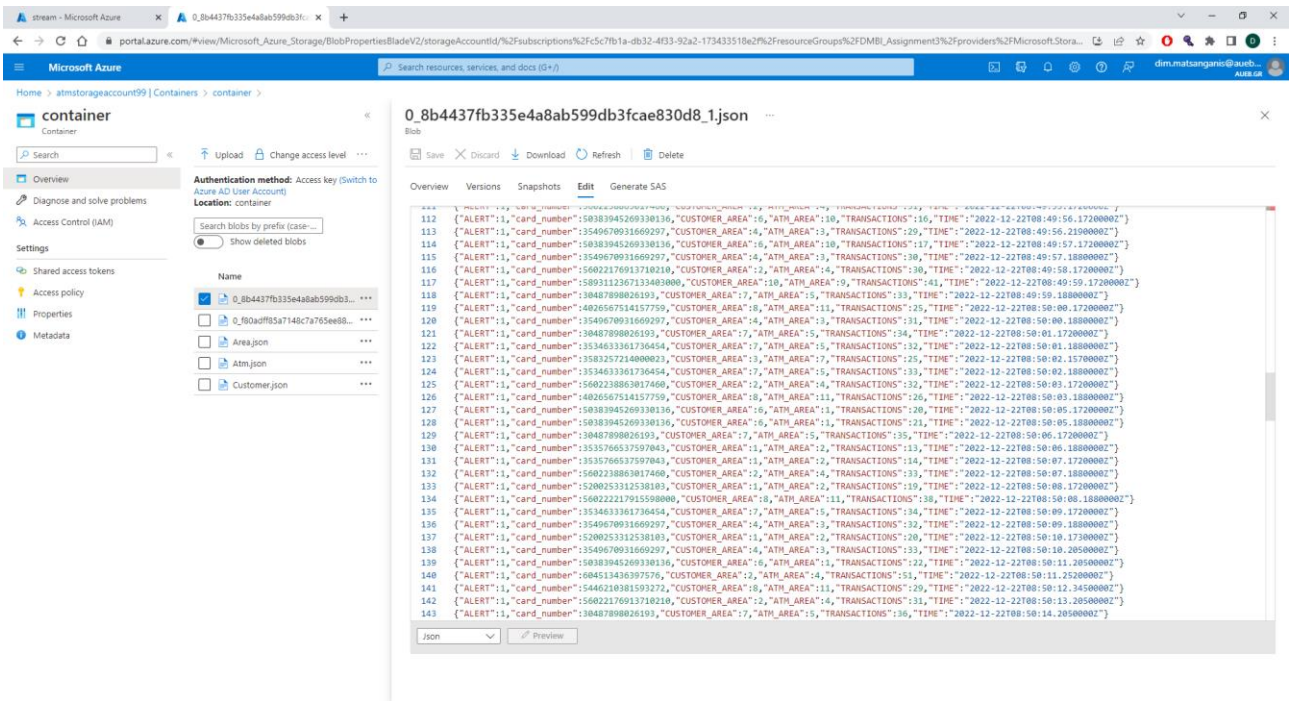


Figure 63: Query 8 Output (2/2).

SQL - Queries' Code

The SQL code for the aforementioned queries is presented below:

```
--Query 1
--Show the total “Amount” of “Type = 0” transactions at “ATM Code = 21” of the last 10 minutes.
--Repeat as new events keep flowing in (use a sliding window).

SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY Type, ATMCode, SlidingWindow(minute,10)
HAVING Type=0 AND ATMCode=21

--Query 2
--Show the total “Amount” of “Type = 1” transactions at “ATM Code = 21” of the last hour.
--Repeat once every hour (use a tumbling window).

SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY Type, ATMCode, TumblingWindow(hour,1)
HAVING Type=1 AND ATMCode=21

-- Query 3:
--Show the total “Amount” of “Type = 1” transactions at “ATM Code = 21” of the last hour.
--Repeat once every 30 minutes (use a hopping window). */

SELECT SUM(Amount) as TOTAL_AMOUNT, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
WHERE Type=1 AND ATMCode=21
GROUP BY ATMCode, HoppingWindow(minute,60,30)

--Query 4
--Show the total “Amount” of “Type = 1” transactions per “ATM Code” of the last one hour (use a sliding window).

SELECT SUM(Amount) as TOTAL_AMOUNT, ATMCode, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY Type, ATMCode, SlidingWindow(minute,60)
HAVING Type=1
```

```
--Query 5
--Show the total “Amount” of “Type = 1” transactions per “Area Code” of the last hour.
--Repeat once every hour (use a tumbling window).

SELECT SUM([input].[Amount]) AS TOTAL_AMOUNT, [atm].[area_code], System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
INNER JOIN [atm] ON [input].[ATMCode]=[atm].[atm_code]
WHERE [input].[Type]=1
GROUP BY [atm].[area_code], TumblingWindow(hour,1)

--Query 6
--Show the total “Amount” per ATM’s “City” and Customer’s “Gender” of the last hour.
--Repeat once every hour (use a tumbling window).

SELECT SUM([input].[Amount]) AS TOTAL_AMOUNT, [area].[area_city] AS CITY,
[customer].[gender] AS GENDER, System.Timestamp AS TIME
INTO [output]
FROM [input]
TIMESTAMP BY EventEnqueuedUtcTime
INNER JOIN [customer] ON [input].[CardNumber]=[customer].[card_number]
INNER JOIN [atm] ON [input].[ATMCode]=[atm].[atm_code]
INNER JOIN [area] ON [atm].[area_code]=[area].[area_code]
GROUP BY [customer].[gender], [area].[area_city], TumblingWindow(hour,1)

--Query 7
--Alert (Do a simple SELECT “1”) if a Customer has performed two transactions of “Type = 1” in a
--window of an hour (use a sliding window).

SELECT 1 AS ALERT,[customer].[card_number], [customer].[first_name],
[customer].[last_name], COUNT(*) AS TRANSACTIONS, System.Timestamp AS TIME
INTO [output]
FROM [input]
INNER JOIN [customer] ON [input].[CardNumber]=[customer].[card_number]
WHERE [input].[Type]=1
GROUP BY [customer].[card_number], [customer].[first_name], [customer].[last_name], SlidingWindow(hour,1)
HAVING TRANSACTIONS=2

--Query 8
--Alert (Do a simple SELECT “1”) if the “Area Code” of the ATM of the transaction is not the same
--as the “Area Code” of the “Card Number” (Customer’s Area Code) - (use a sliding window)

SELECT 1 AS ALERT,[customer].[card_number], [customer].[area_code] AS CUSTOMER_AREA,
[atm].[area_code] AS ATM_AREA, COUNT(*) AS TRANSACTIONS, System.Timestamp AS TIME
INTO [output]
```

```
FROM [input]
INNER JOIN [customer] ON [input].[CardNumber]=[customer].[card_number]
INNER JOIN [atm] ON [input].[ATMCode]=[atm].[atm_code]
WHERE [customer].[area_code]!=[atm].[area_code]
GROUP BY [customer].[card_number], [customer].[area_code], [atm].[area_code],
SlidingWindow(hour,1)
```