**BIG DATA ENGINEERING PROJECT**

**IOT based solution for**

**Vehicular Traffic**

**using Stream Analytics**

**Team**

**Rutika Ranjolkar**

**Nehal Bhanushali**

**Jyoti Sharma**

**Amitha Murali**

**Table of Content**

1. Introduction
2. Research
3. Application architecture
4. Phase I: IOT simulation and Data preparation
   1. Installations and Setup
   2. Running the scripts and capturing output
5. Phase II: Stream Analytics
   1. Installations and Setup
   2. Create stream analytics job
   3. Event hub listener
   4. Executing SAJ job
6. Phase III: Power BI output Analysis
7. Use Cases
   1. Theft Detection
   2. Air pollution Analysis in major cities of USA
8. Challenges and Learnings
9. Future Scope
10. References

**Introduction**

Vast amounts of data are flowing at high velocity over the wire today. Organizations that can process and act on this streaming data in real time can dramatically improve efficiencies and differentiate themselves in the market. Scenarios of real-time streaming analytics can be found across all industries: personalized, real-time stock-trading analysis and alerts offered by financial services companies; real-time fraud detection; data and identity protection services; reliable ingestion and analysis of data generated by sensors and actuators embedded in physical objects (Internet of Things, or IoT); web clickstream analytics; and customer relationship management (CRM) applications issuing alerts when customer experience within a time frame is degraded. Businesses are looking for the most flexible, reliable and cost-effective way to do such real-time event-stream data analysis to succeed in the highly competitive modern business world.

The project aims at solving 2 important questions related to vehicular traffic:

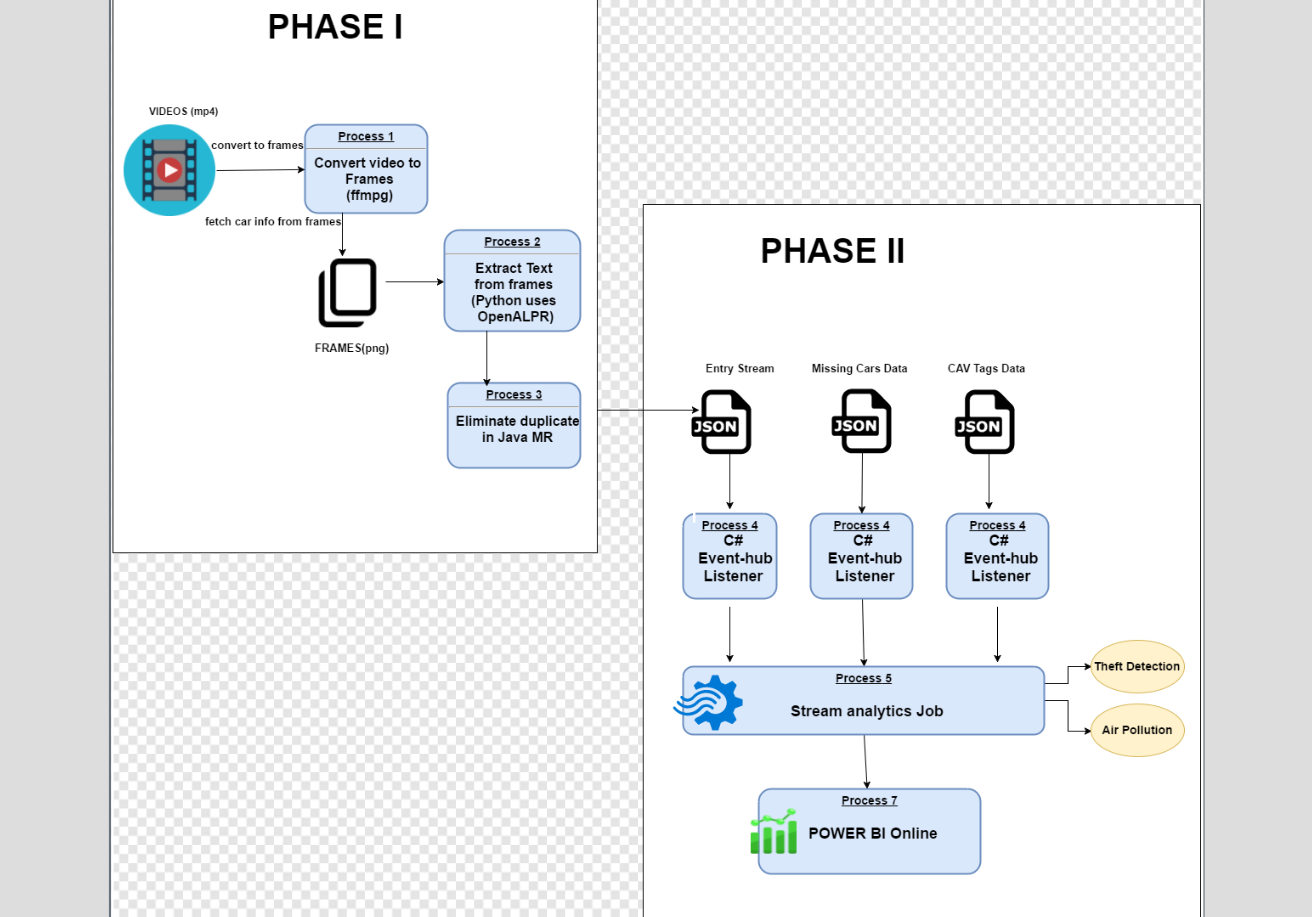
* Spotting and reporting stolen cars
* Calculating and projecting the environmental aftermath of vehicles at any given location at a specified time

**Research**

The research for the project was an ongoing process. It mainly consisted of two chapters:

* Research to zero-in on a project
* The team considered and evaluated a bunch of potential project topics related to sentiment analysis, convolutional neural networks and natural language processing. Finally, we landed on Stream Analytics. This concept sounded both interesting and challenging to the team.
* After looking at various applications of Stream Analytics, the team decided to choose *IoT based solution for Vehicular Traffic using Stream Analytics* as our project.
* Research while executing and implementing steps towards the project
* We identified individual components of the project and investigated suitable technologies to execute each of the outlined components.
* This was a continuous process throughout the project life cycle since we were on a constant lookout for the best match to perform each individual task.

**Project Overview:**



**Phase I: IOT simulation and Data Preparation**

**Approach**

A video consists of very fast moving images. The idea is to use a traffic cam video as an input and split it into component images using FFmpeg framework. Each image shall consist of vehicles whose license plates can be read using the openALPR (an open source Automatic License Plate Recognition library). The make, model and other details are extracted and sent over to Azure.

**Solution**

* Use FFmpeg to split video into images:
* FFmpeg is a free software framework that produces libraries and programs for handling multimedia data. FFmpeg is highly portable. It compiles and runs on various platforms such as Linux, Mac OS X, Microsoft Windows, Solaris etc. under a wide variety of build environments, machine architectures and configurations.
* FFmpeg splits the input video into individual frames that constitute the video. It is a codec library that handles video format conversions (binary to video data and vice-versa).
* Frame rate (expressed in frames per second or fps) is the rate at which consecutive images, called frames are displayed in an animated display. Ideally, films are shot at 24 frames per second.
* We consider the traffic video to consist of 24 frames per second for the project.
* The number of frames per second that the video is split can be controlled  via the shell file and can vary from 1 fps(frame per sec) to 30 fps.
* Use image names as text to input to the mapper. The actual image corresponding to the image name is extracted internally for processing.
* Use openALPR to capture license plate information from each image:
* OpenALPR is an open source Automatic License Plate Recognition library written in C++ with bindings in C#, Java, Node.js and Python.
* This library analyzes images to identify license plates. It also enables the users to select from a range of countries to help identify the country of issuance of the license plate.The output of this stage is a text representation of license plate characters.
* Hadoop streaming is used in the project. Both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output.
* As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input (STDIN) of the process.
* In the meantime, the mapper collects the line-oriented outputs from the standard output (STDOUT) of the process and converts each line into a key/value pair, which is collected as the output of the mapper.
* By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value.
* If there is no tab character in the line, then the entire line is considered as the key and the value is null. However, this can be customized, as per one need.

Prerequisites:

* Java:
* Hadoop
* Python
* FFmpeg
* openALPR

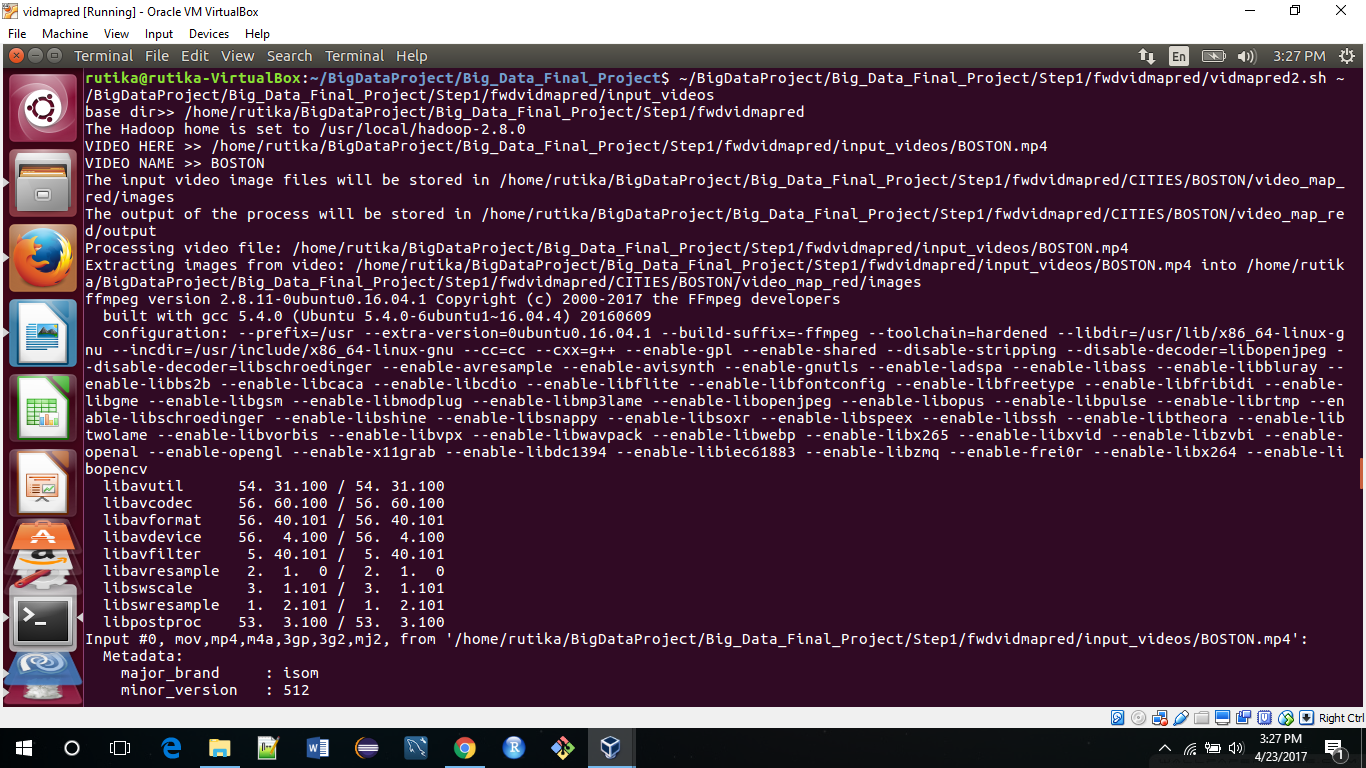
**Create a Map Reduce job**

**Mapreduce Step1**: To begin running the project, we run the shell file and the input video through the command line.

./vidmapred.sh <video file name>

vidmapred.sh is a shell file that contains all the commands that need to execute consecutively.  It contains commands to create input & output folders, run hadoop jobs  and provide updates.

The input to the project is in the form of a video generated by a cctv camera that monitors traffic on a busy city street.



This video is processed via FFmpeg software.

The syntax for running the ffmpeg framework is in the following format:

ffmpeg [global\_options] {[input\_file\_options] -i input\_url} ... {[output\_file\_options] output\_url}

The syntax used in the project is:

ffmpeg -i $VIDEO\_FILE -r 2/1 $MAPRED\_IMG\_DIR/%07d.png

‘ffmpeg’ reads from an arbitrary number of input "files” specified by the -i option, and writes to an arbitrary number of output "files", which are specified by a plain output url. Anything found on the command line which cannot be interpreted as an option is considered to be an output url.

The above syntax uses $VIDEO\_FILE as an input. 2/1 signifies number of frames per second. $MAPRED\_IMG\_DIR is the output folder. The individual images that are extracted from the video are stored in this folder.

The %07d signifies that the images that are extracted will be in the order of ‘0000000’ numbers. For example, the first image will be stored as 0000001.png. This format ensures that the images extracted will not run out of memory to be stored.

After the FFmpeg stage, the input video is converted into a bunch of images.

In the next stage, the processing begins on the images. The approach is to process all the images using mapreduce.

Mapreduce is suited for this project since the size of the images generated by the video processing is big data.

24 hours video = 24\*60\*60 = 86400 seconds video

24 frames per second\* 86400 seconds = 2,073,600 frames or images

1 image ~ 350 kb

2,073,600 \* 350 =725,760,000 kb ~ 725 gb of data per day

Also, since input to the mapper should be in the form of consecutive lines of text, all the image names are extracted and collected to form a text file. Each line in this file consists of one image name.

The Mapper is written using Python. After passing the above mentioned file to Hadoop, it splits the file into individual lines, each representing an image file name. These lines (image file names) are passed to the mapper one by one.

The mapper reads each line as a file name of one image from the video sequence. In other words, we have passed one image to the mapper (in the form of its file name). Next, the mapper passes this file name to the OpenALPR command, which it calls using the subprocess package. In other words, OpenALPR now processes a single image from the video sequence. Since the mappers can be parallelized, multiple mapper instances will each process a different image frame, and so the overall speed of processing is increased multiple times.

The library reads license plate numbers for each image passed as an input. The mapper result is the license plate number and confidence. Confidence is the probability with which the accuracy of license plate that is read is determined.

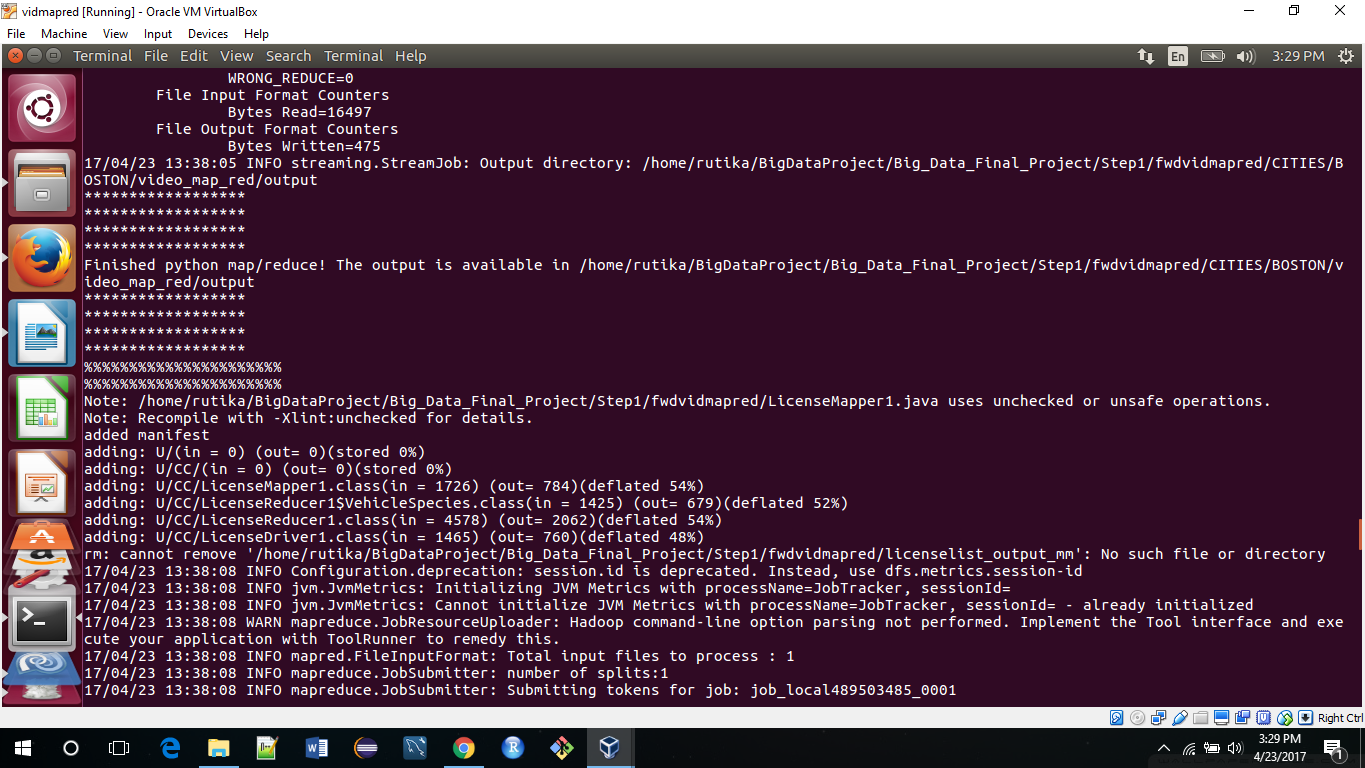
The Reducer takes the output from the mapper, processes it to only select license plates from the file, and provides uncluttered information.

This output is stored in the format ‘License\_plate\_number’ ‘Confidence:’ ‘Numerical\_value\_of\_Confidence’

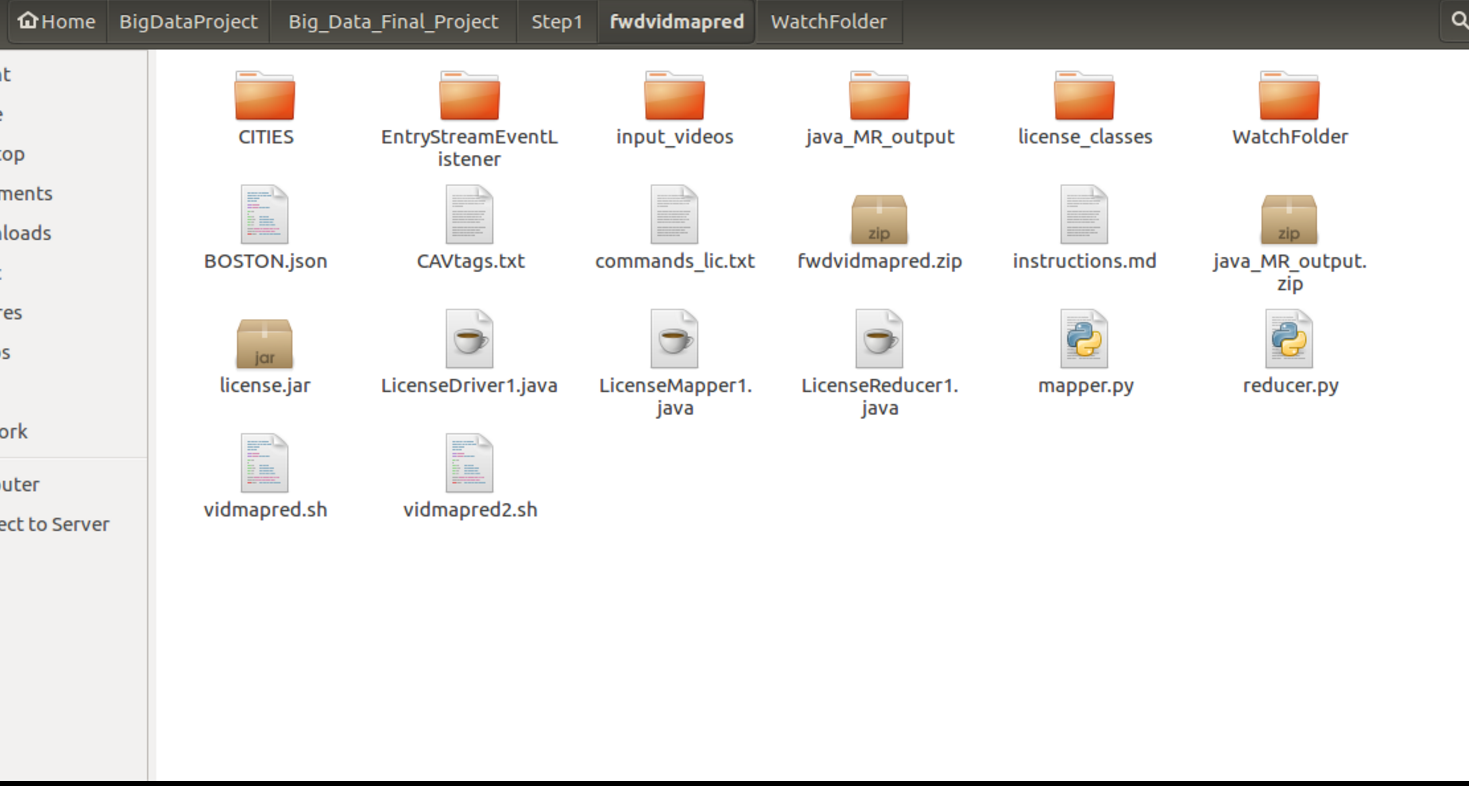
For example: -BD 792346     confidence: 86.0561

**Mapreduce stage 2:**

* The python Mapreduce extracts the car license info and outputs a text file (- <license-plate-number> confidence : 89>) which is sent to Java Mapreduce for obtaining Make, model and other info. (Ideally, an API would fetch make, model, but we have simulated that in our Java code. Also, Python is chosen to mapreduce the frame to text license number extraction as it is much lighter.)
* The Java Mapreduce eliminates the duplicate license numbers and fetches the car info. The output json is sent to the C# Event-hub listener that sends the files to the Azure stream analytics job.

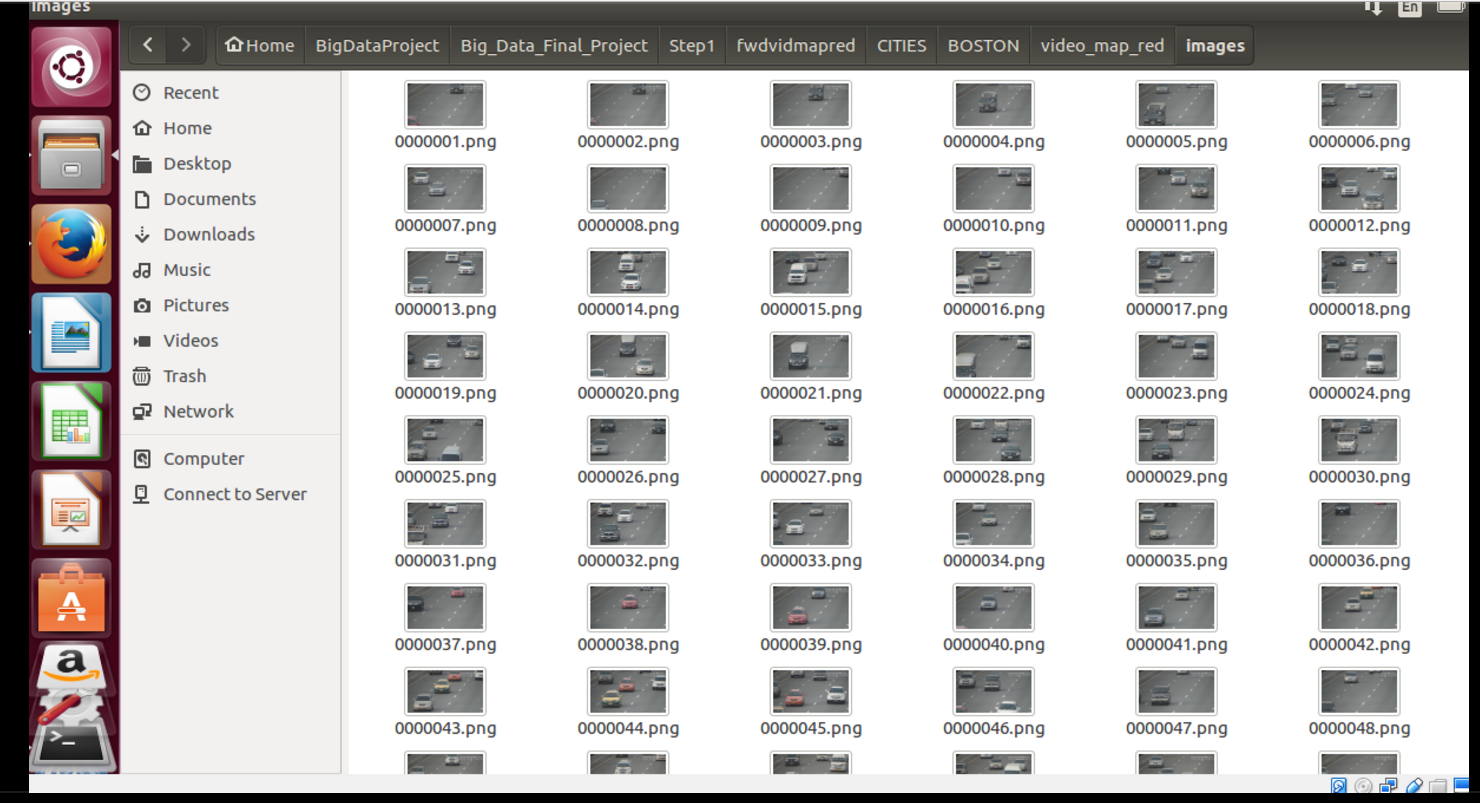


Folder structure for Phase1 with output files in Ubuntu :

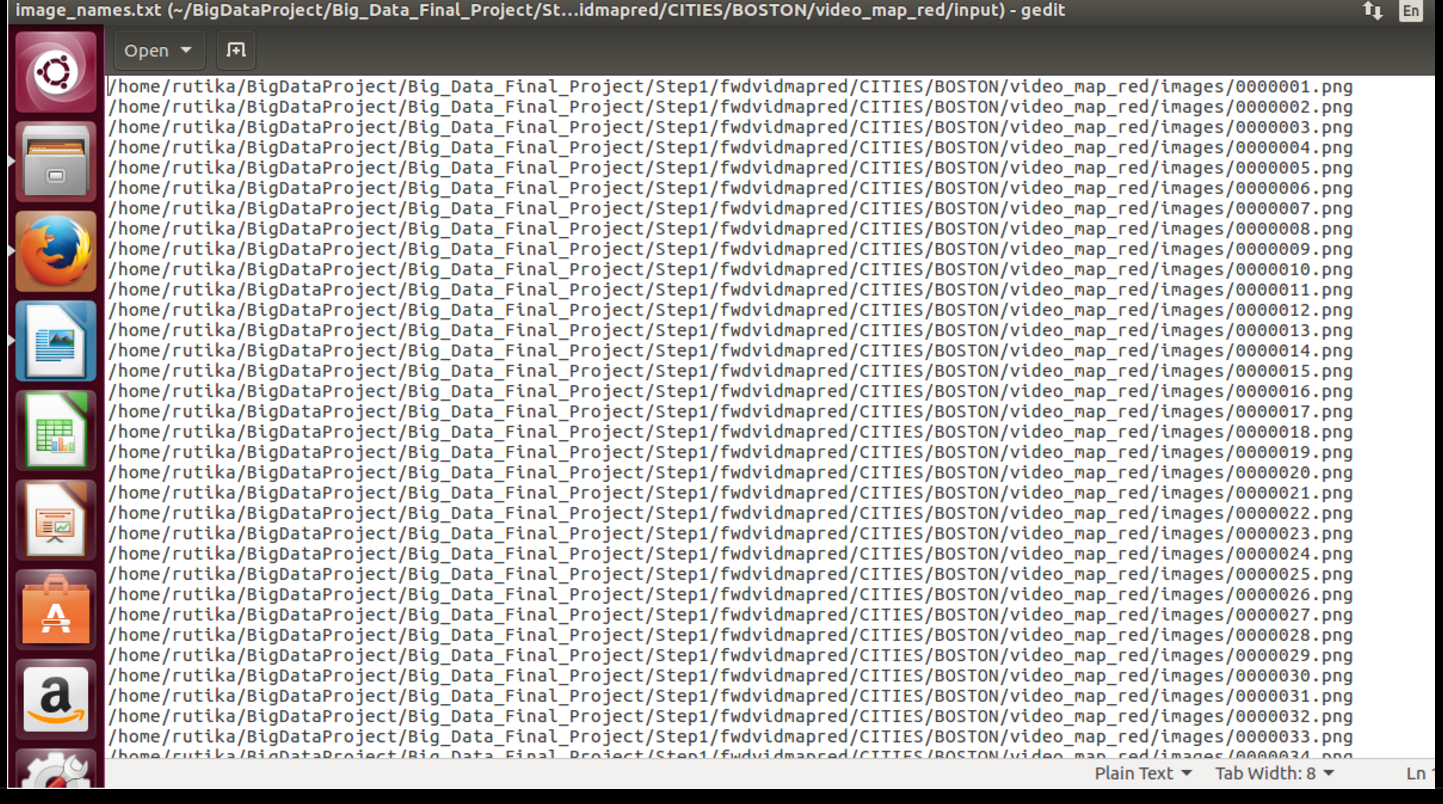


**Phase I output:**

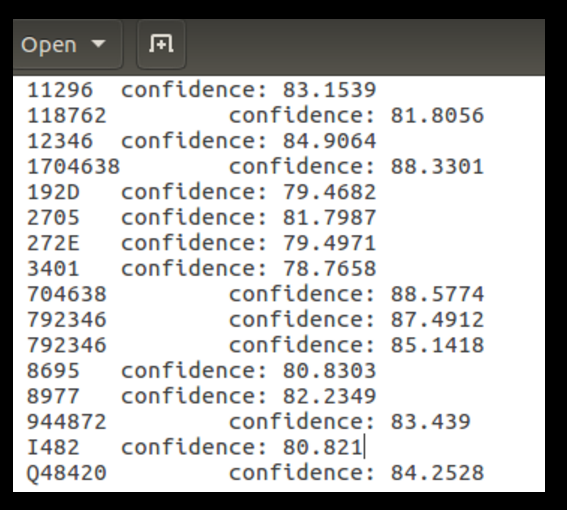
Phase 1 intermediate output of the images extracted from videos



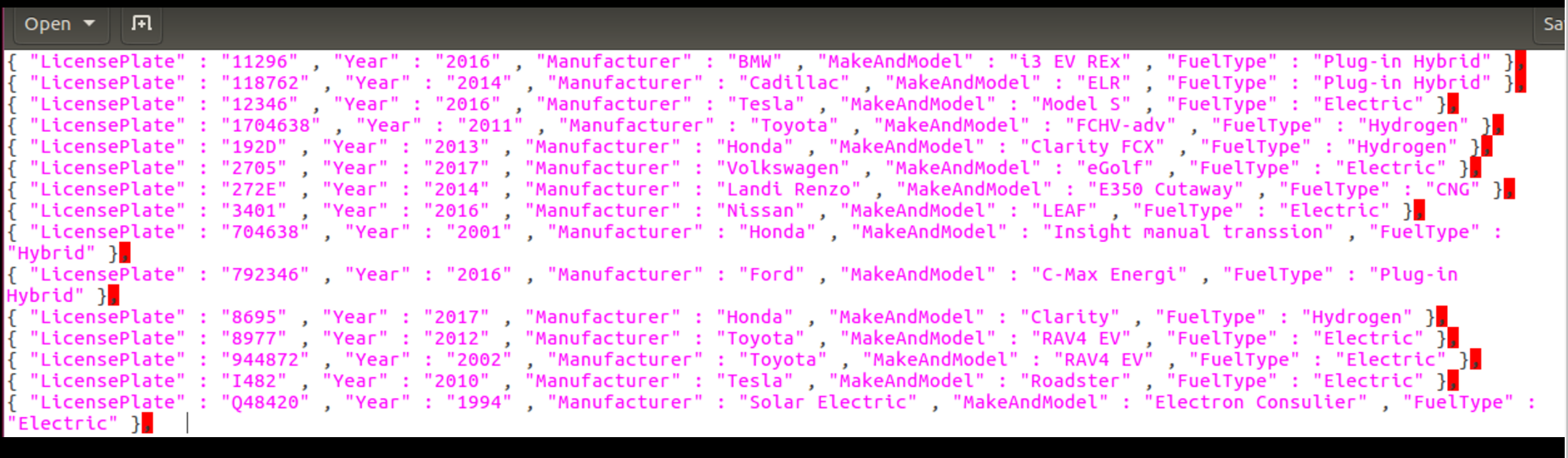
Phase 1 Output text file with image names as the values.



Phase 1 Output file with the license plate of US along woith their confidence score.



Phase 1 final output files as Json files

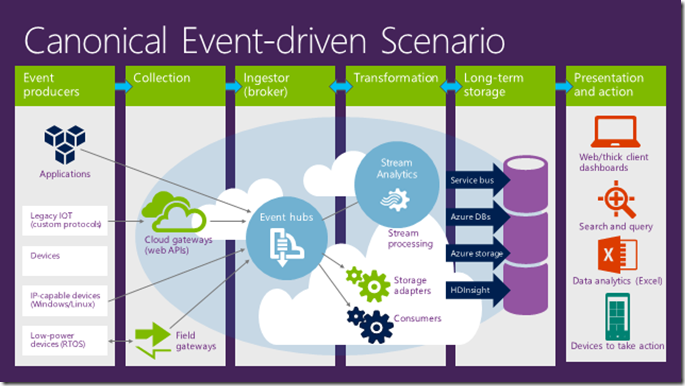


**Phase II: Stream Analytics**

Azure Stream Analytics is a fully managed, cost effective real-time event processing engine that helps to unlock deep insights from data. Stream Analytics makes it easy to set up real-time analytic computations on data streaming from devices, sensors, web sites, social media, applications, infrastructure systems, and more.

Developers can easily combine streams of data, such as click-streams, logs, and device-generated events, with historical records or reference data to derive business insights. As a fully managed, real-time stream computation service that's hosted in Microsoft Azure, Azure Stream Analytics provides built-in resiliency, low latency, and scalability to get you up and running in minutes.

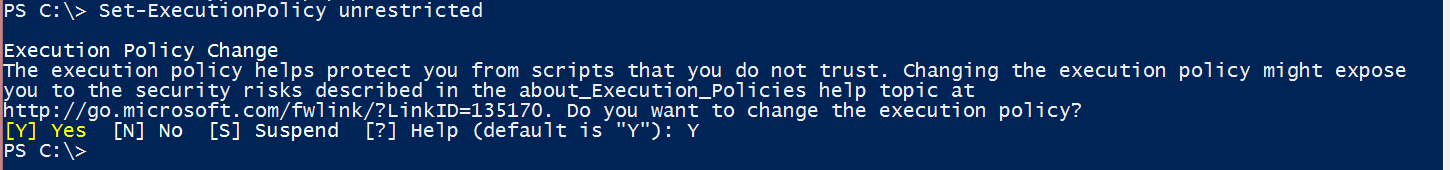
Here is an over view:



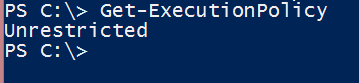
**Installations and Setup**

**Step1: Setting up environment**

To set up your Azure environment, open a **Microsoft Azure PowerShell** window as an administrator. Because Windows automatically blocks .ps1, .dll, and .exe files, you need to set the execution policy before you run the script. Make sure the Azure PowerShell window is running as an administrator. Run **Set-ExecutionPolicy unrestricted**. When prompted, type **Y**.



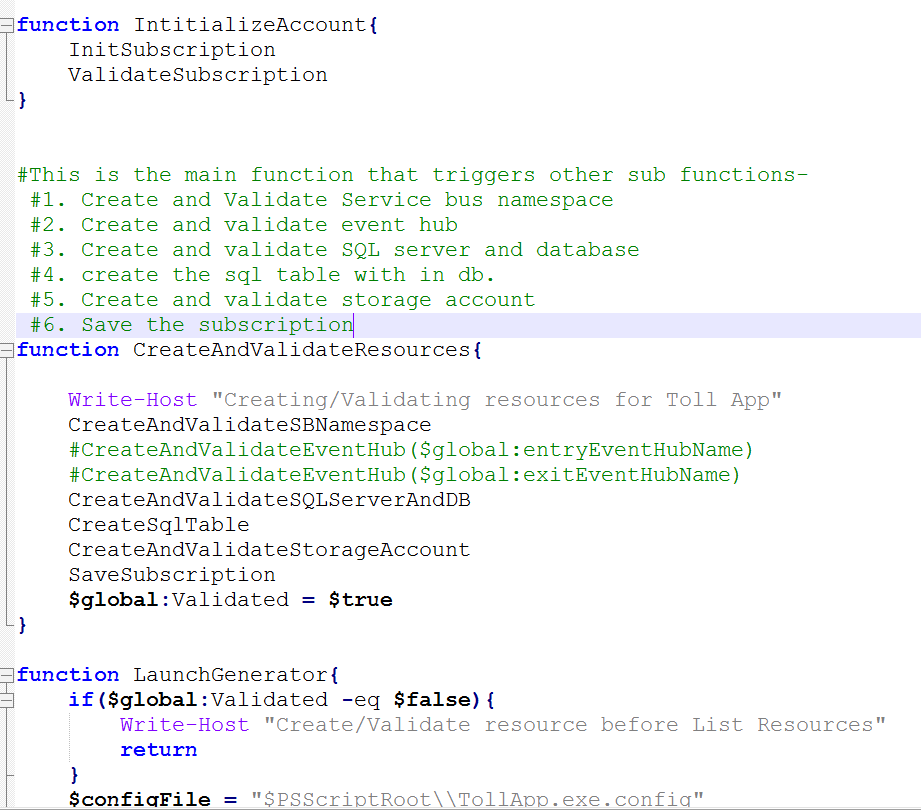
Run **Get-ExecutionPolicy** to make sure that the command worked.



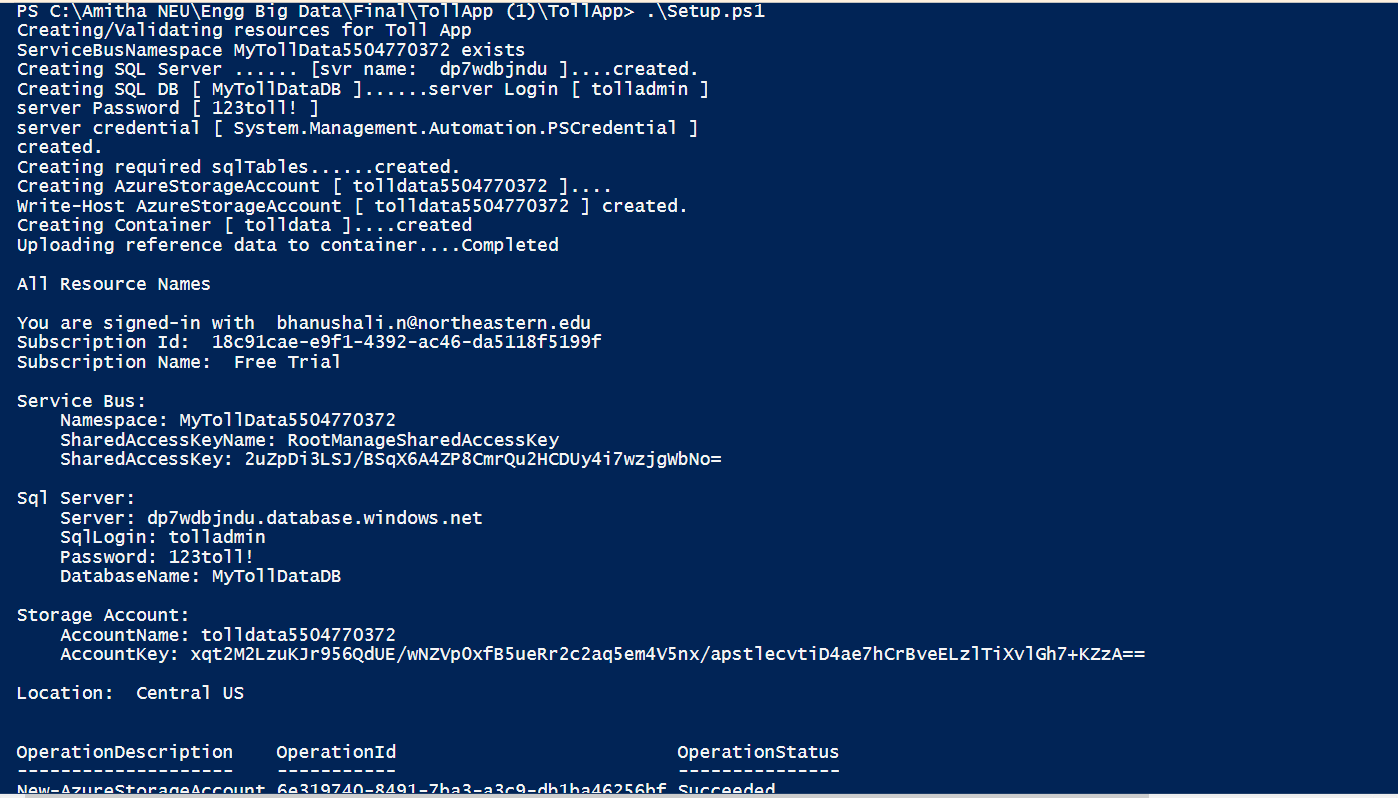
To set up your Azure account, create and configure all required resources, and start to generate events. The script randomly picks up a region to create your resources. To explicitly specify a region, you can pass the **-location** parameter as in the following example:

**.\Setup.ps1 -location “Central US”**

**else just type .\Setup.ps1**

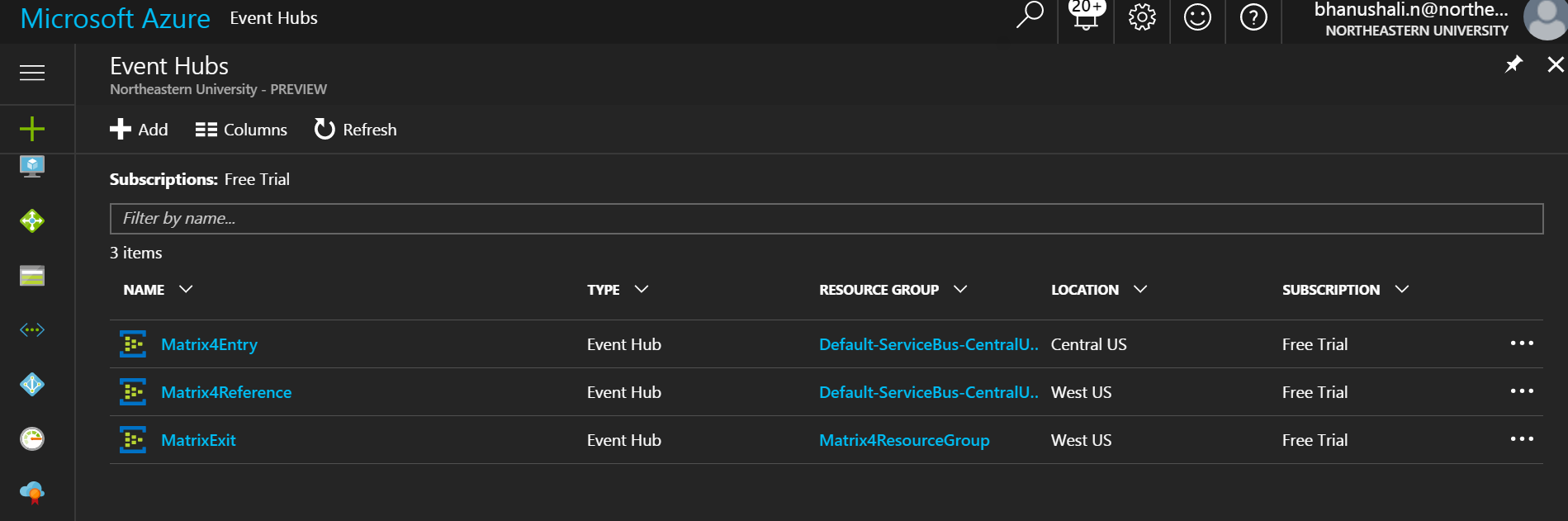


The script can take several minutes to run. After it finishes, the output should look like the following screenshot.



**Azure Event Hubs**

In the Azure portal, click More services on the bottom of the left management pane. Type Event hubs in the field provided and click Event hubs. This launches a new browser window to display the SERVICE BUS area in the classic portal. Here you can see the Event Hub created by the Setup.ps1 script.



Note: Matrix4Reference was later created manually.

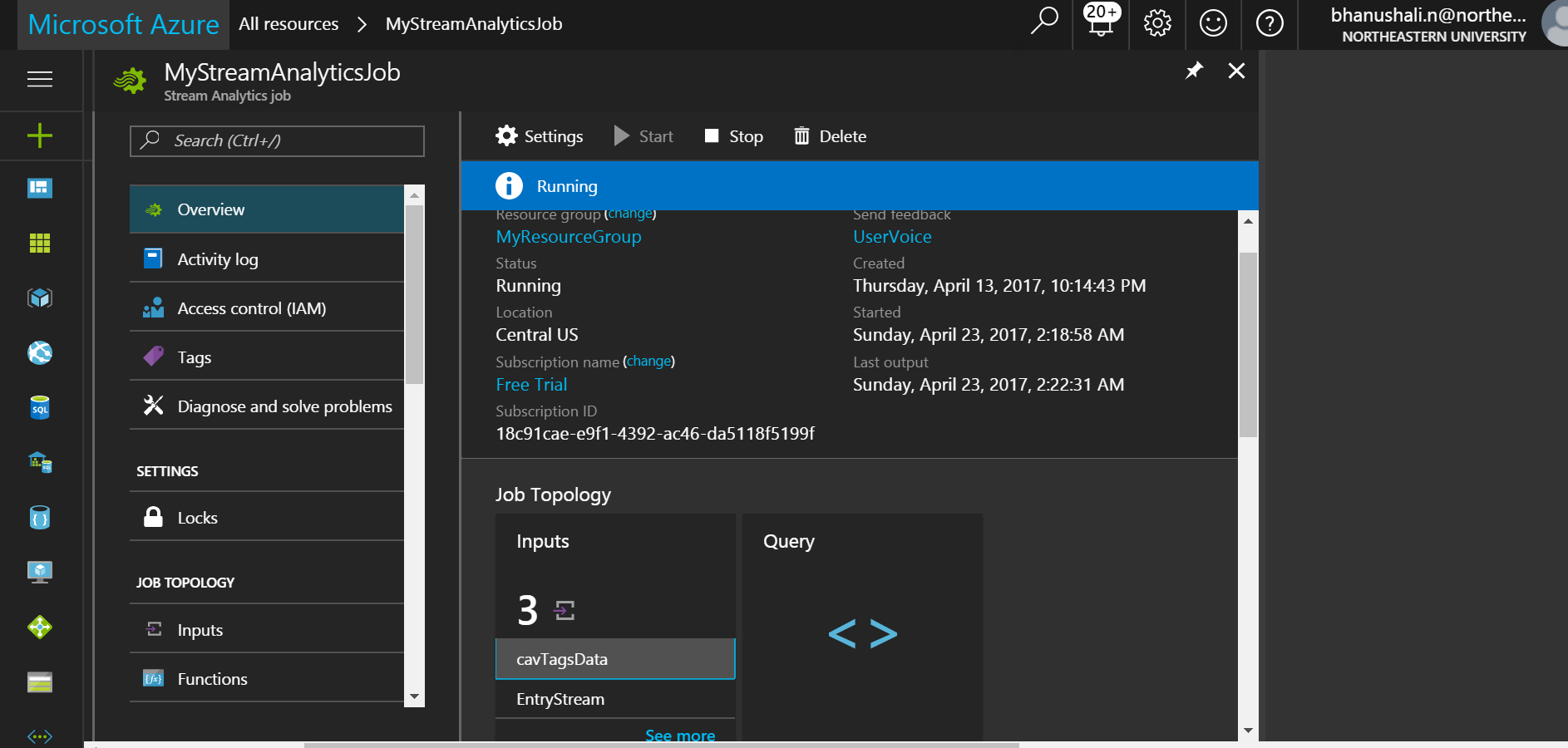
**Step2: Create a Stream Analytics job**

In the Azure portal, click the green plus sign in the top-left corner of the page to create a new Stream Analytics job. Select **Intelligence + Analytics** and then click **Stream Analytics job**.

Provide a job name, validate the subscription is correct and then create a new Resource group in the same region as the Event hub storage (default is South Central US for the script).Click Pin to dashboard and then CREATE at the bottom of the page.

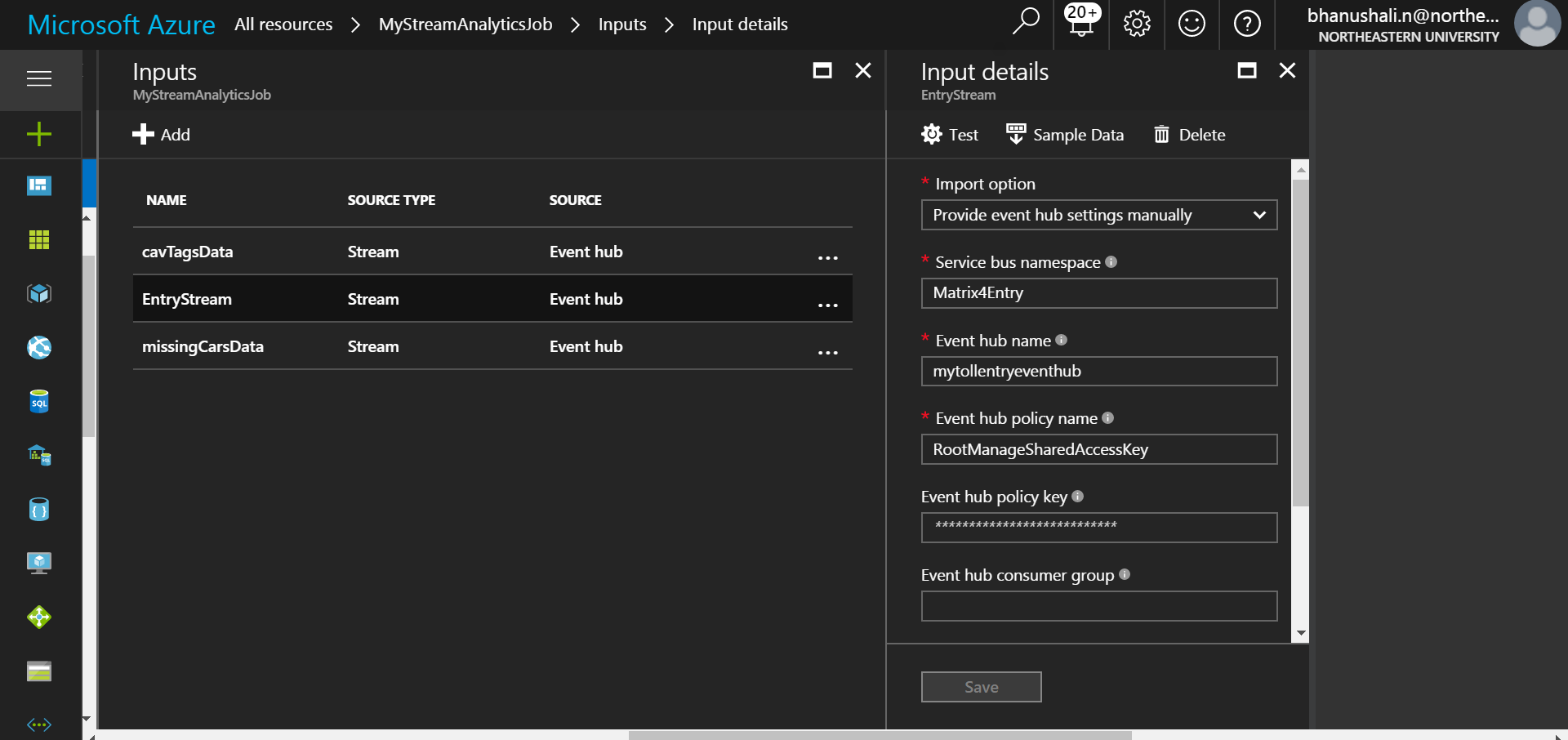
**Define Input Sources:**

The job will create and open the job page. Or you can click the created analytics job on the portal dashboard. Click the INPUTS tab to define the source data.

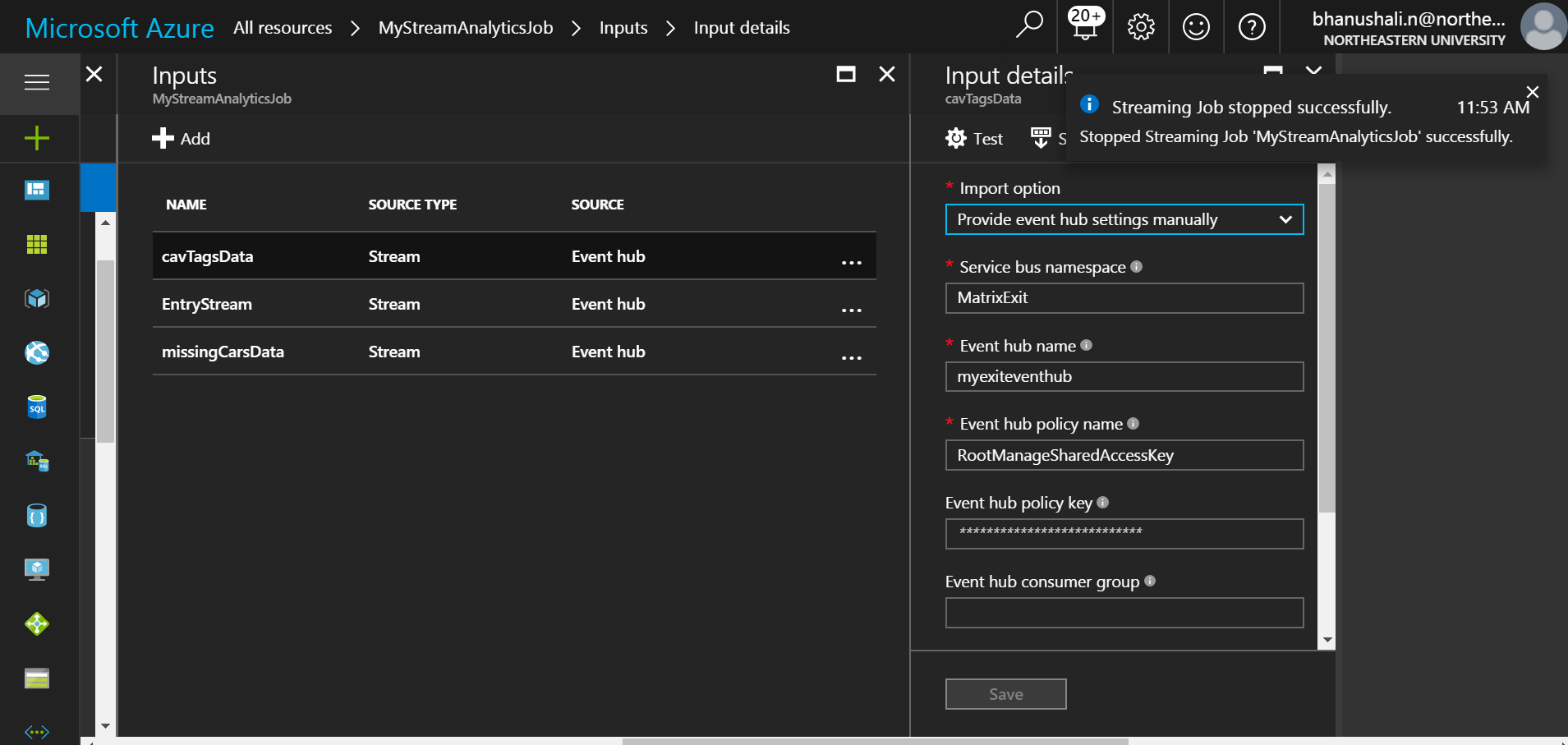


Now add new input (refer to the screen shots below):

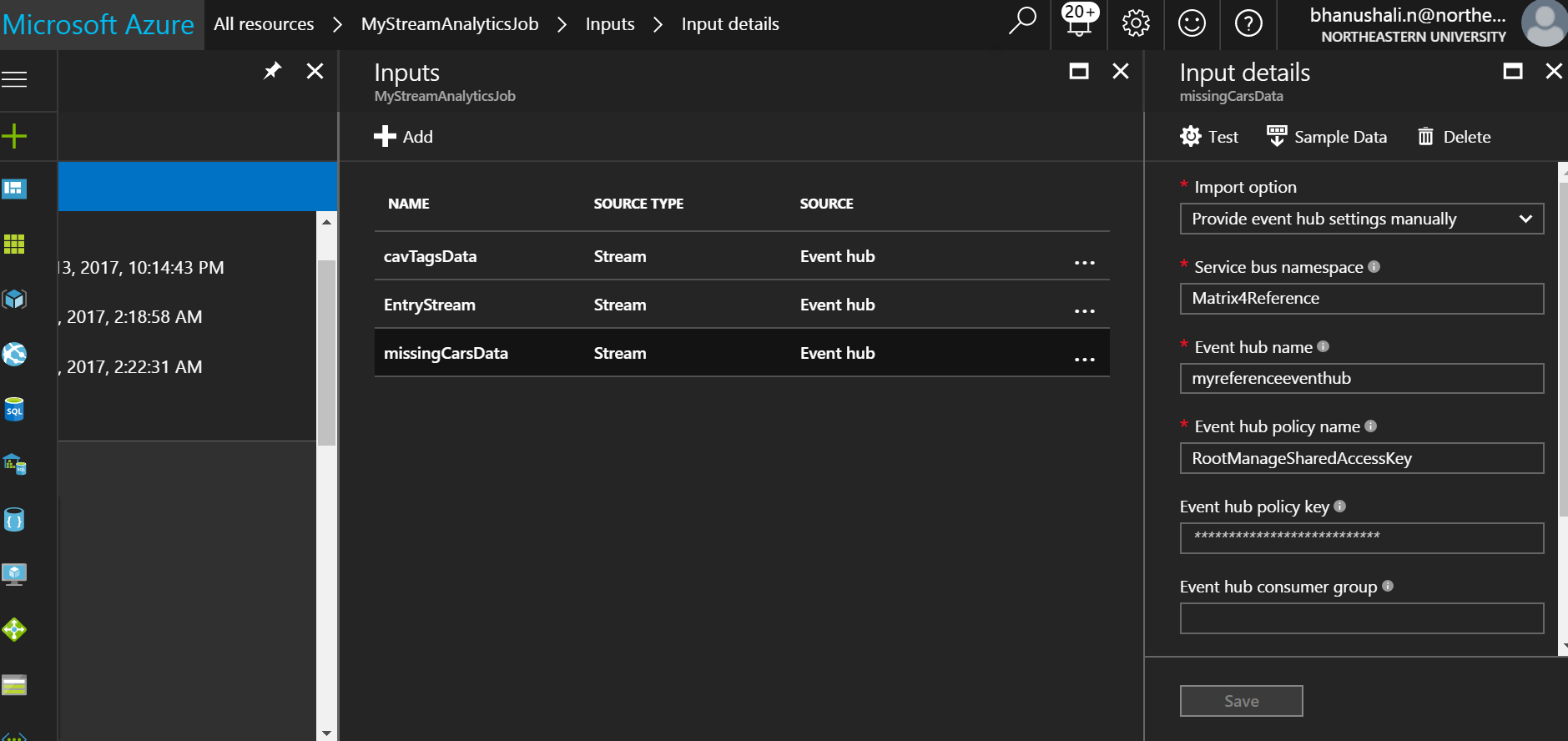
**Entry Stream**



**CAV tags input:**



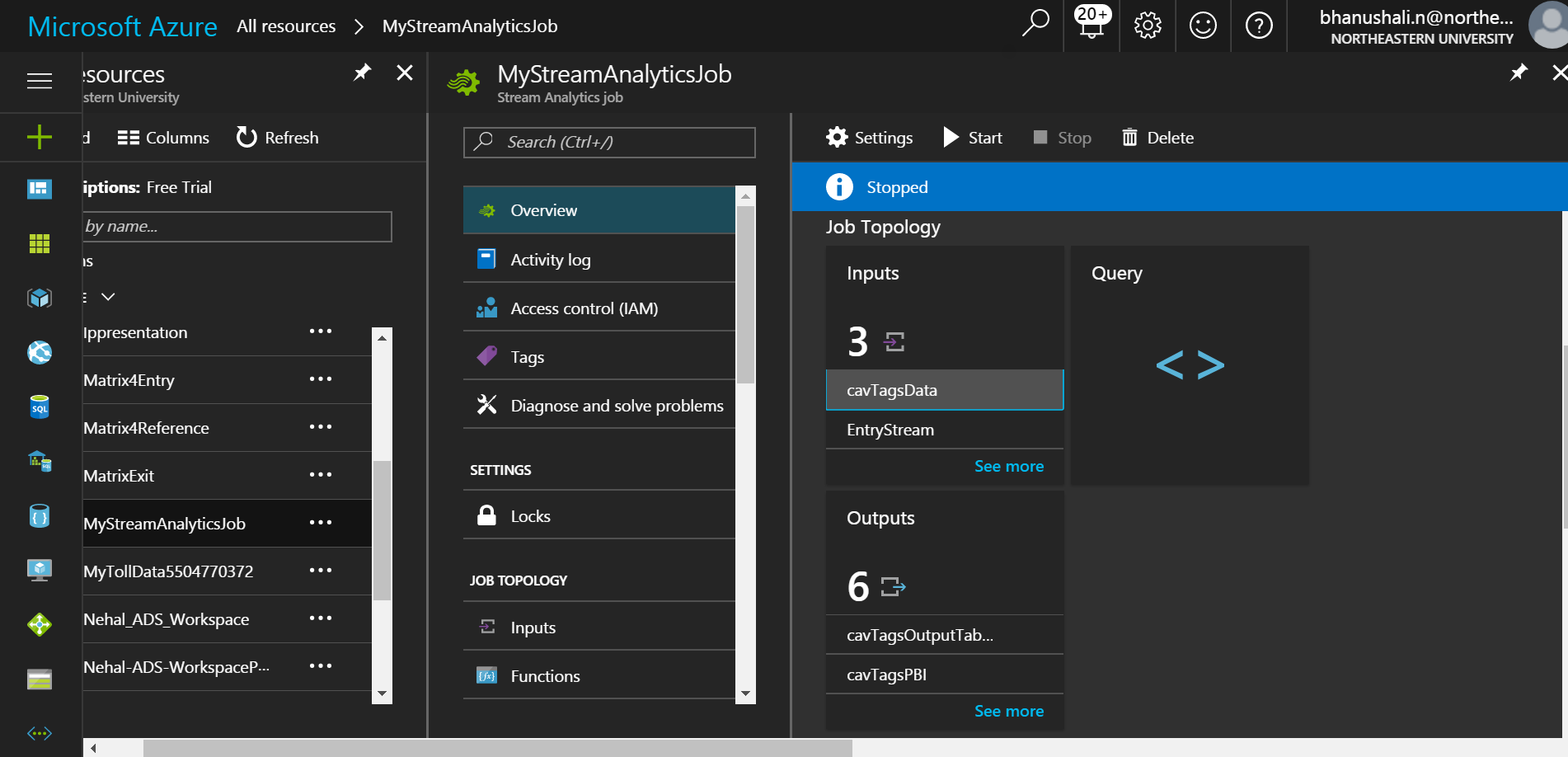
**Missing Cars input:**



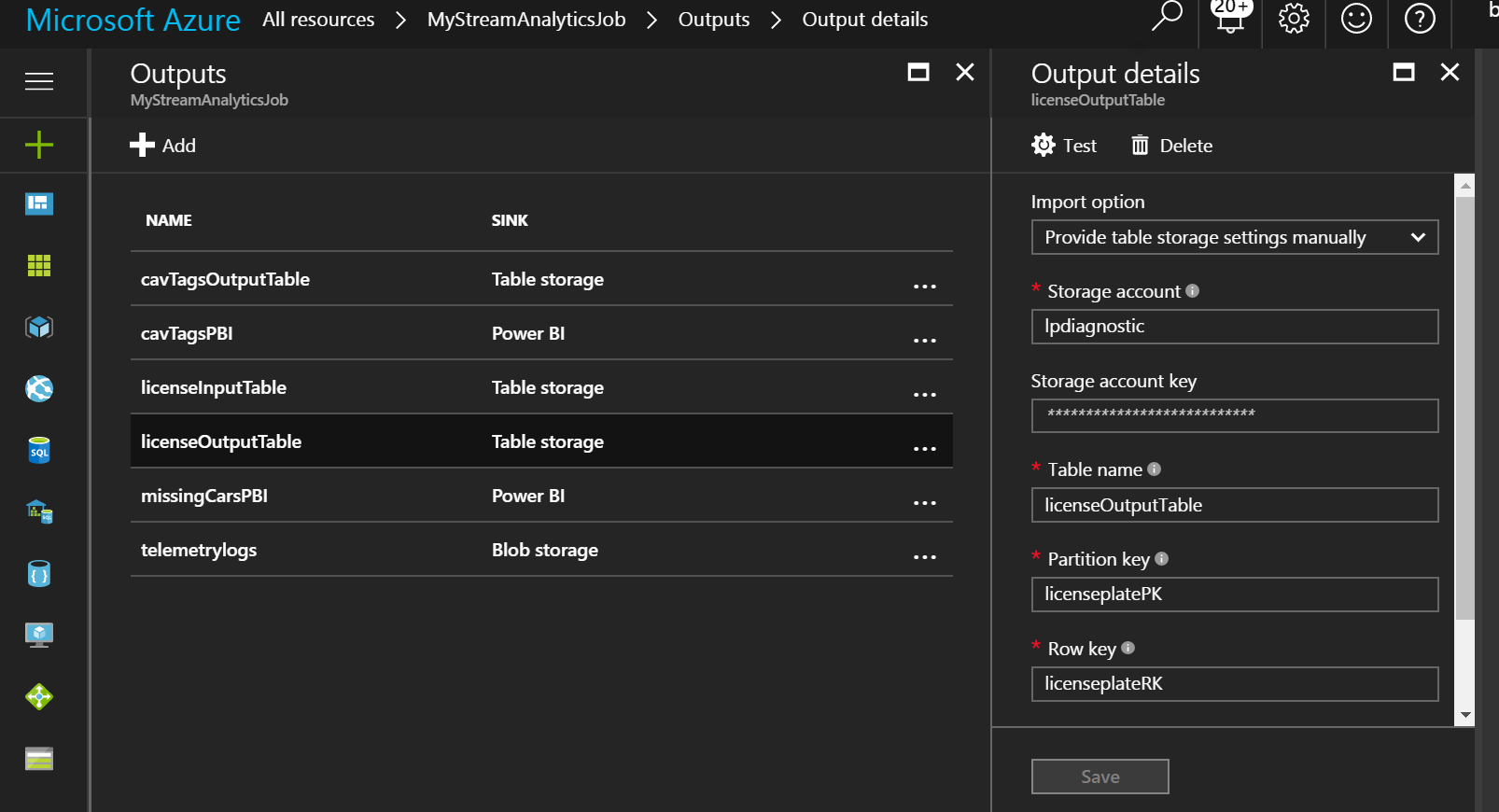
Note: Please make sure your service bus namespace and event hub names are unique for each of these streams. We would require them for connecting through event hub listener.

**Define Output:**

On the Stream Analytics job overview pane, select **OUTPUTS**.

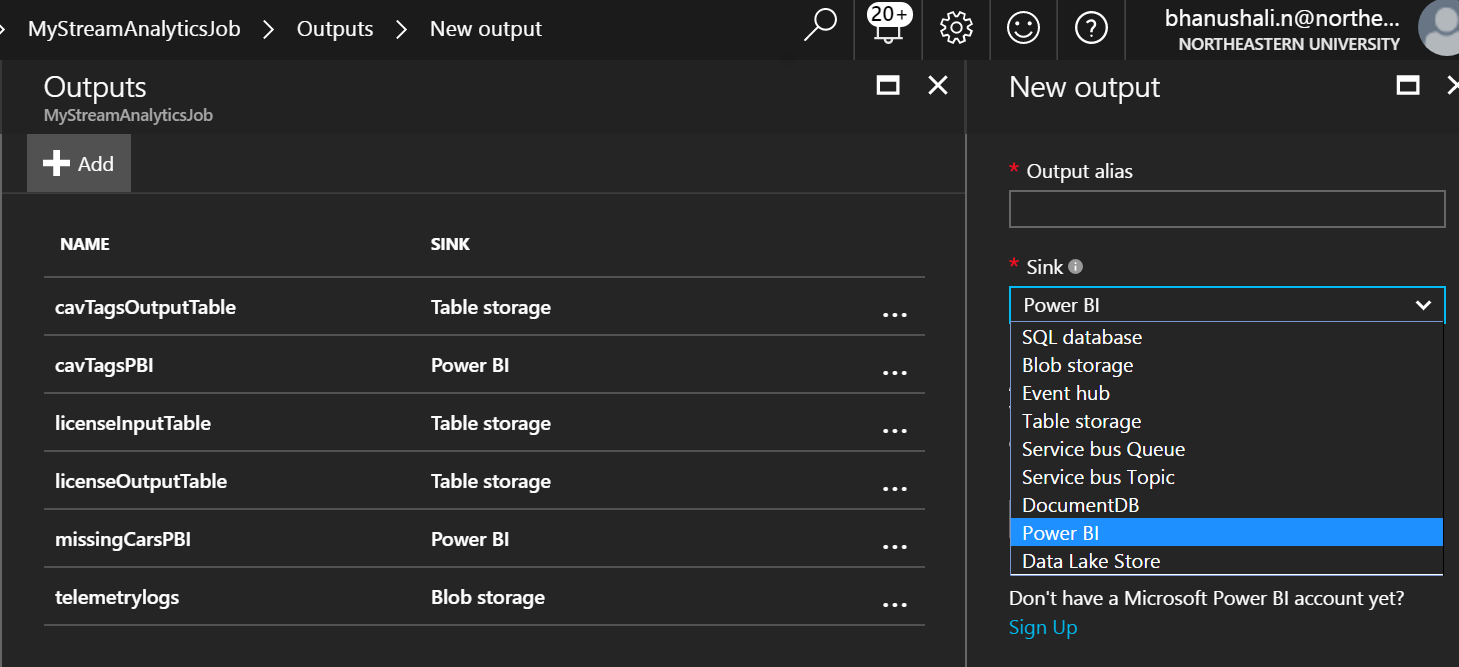


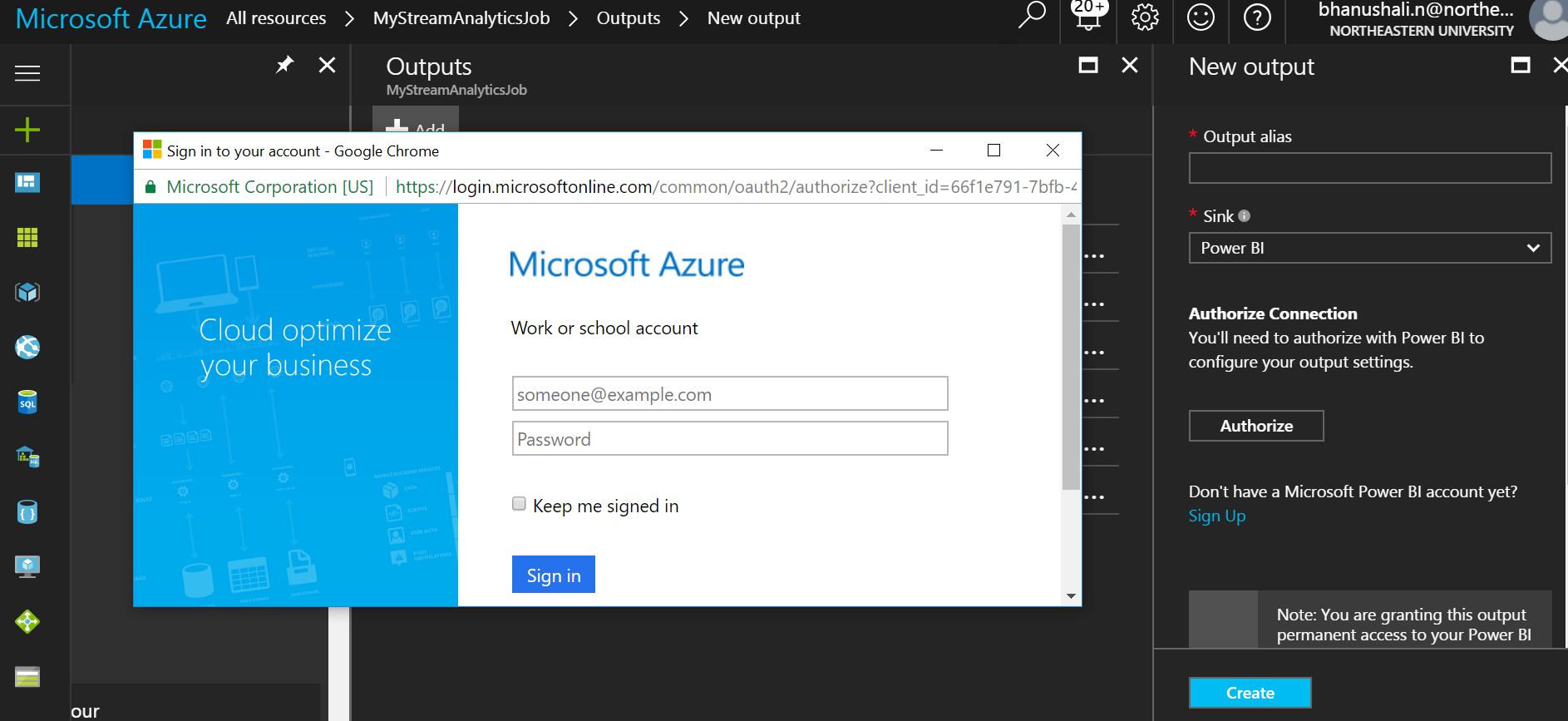
Output tables are table storage. We would need three of them for the three input streams.



**Power BI output:**

For power BI output, select ‘power BI’ as the option in ‘sink’. Now, authorize the power BI account to connect to Azure so that the data can flow directly.

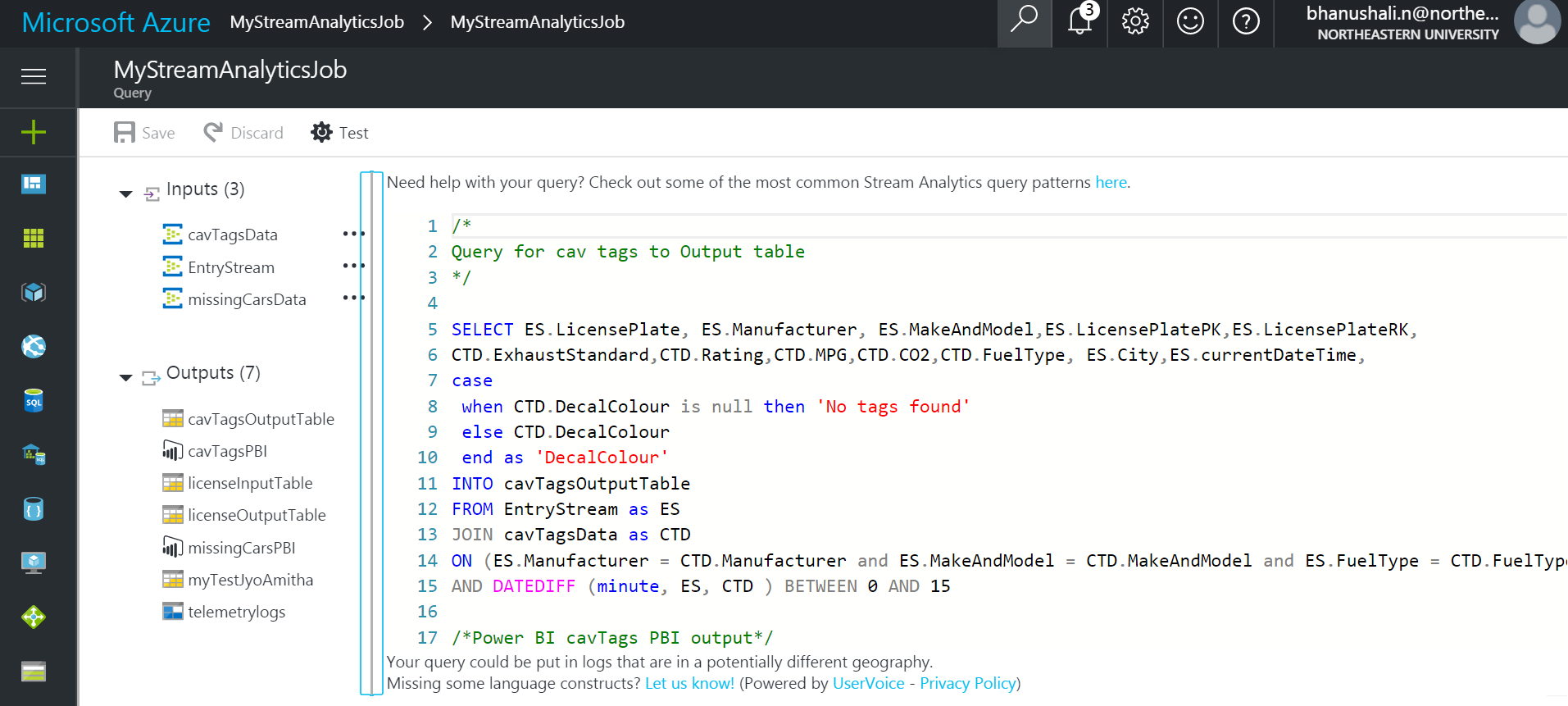




Note: Enter **tolladmin** in the **USERNAME** field, **123toll!** in the **PASSWORD** field, and **TollDataRefJoin** in the **TABLE** field.

**Azure Stream analytics query**

The stream-processing logic in ASA is expressed in a SQL-like query language with some added extensions such as windowing for performing temporal calculations.

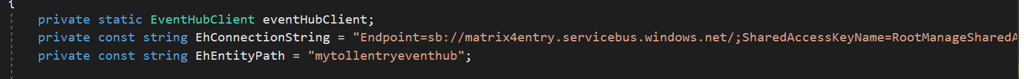


**Step 3: Event Hub Client Listener**

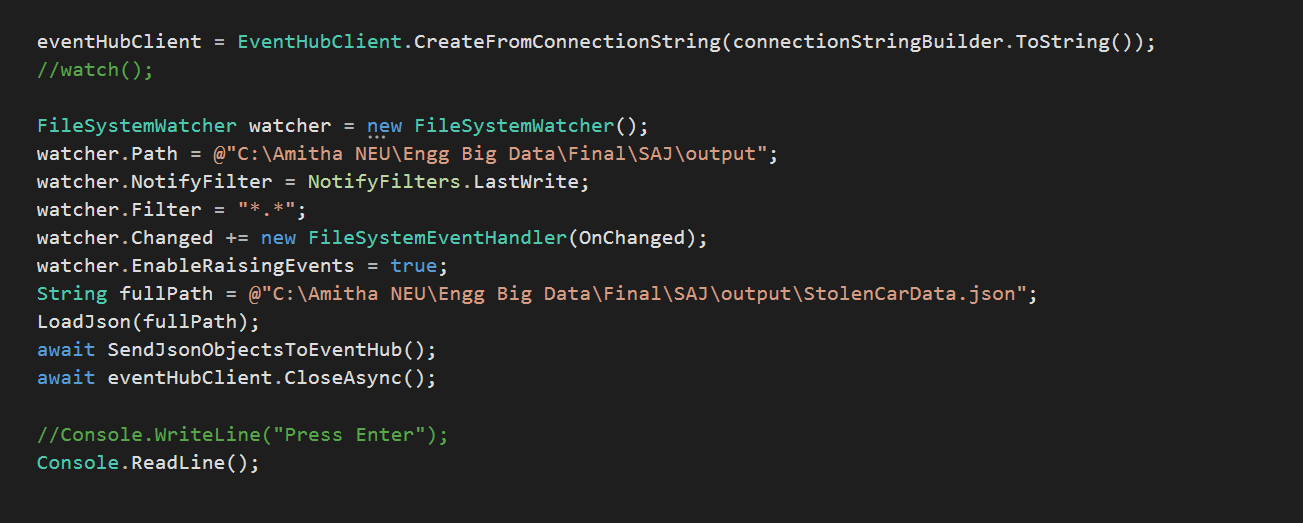
The primary class for interacting with Event Hubs is Microsoft.ServiceBus.Messaging.EventHubClient. This class provides both sender and receiver capabilities.

var client = EventHubClient.Create(description.Path);

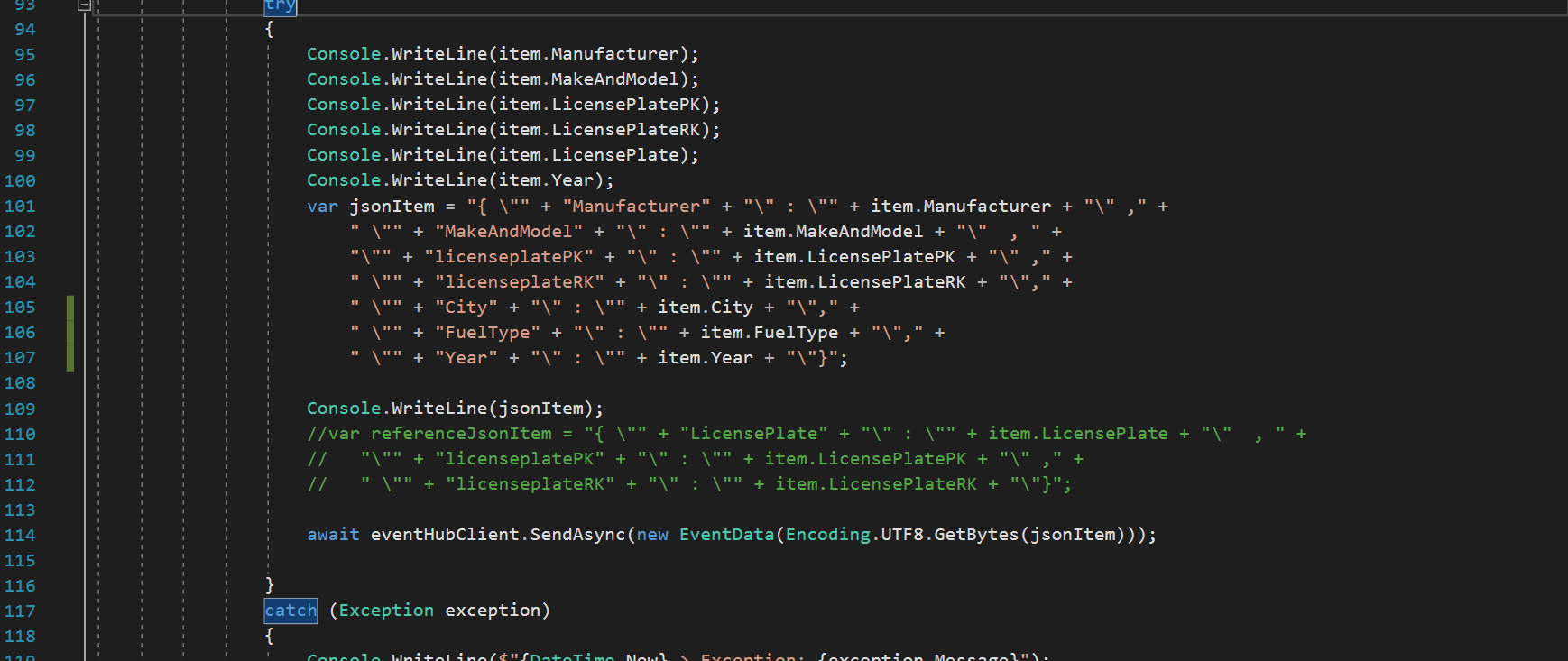
* Create a console application and add Microsoft.Azure.EventHubs package.
* Use await eventHubClient.SendAsync method to send the output.json file created in phase I.
* Add the ‘connection string’ from the event hub you are trying to connect to and then add the event hub name in ‘entity path’ as shown below in the screen shot:



We have used ‘windows file watcher’ to watch the folder. Every time phase I emits a new output file, it triggers an event and the file watcher send the json file path to onchange event.



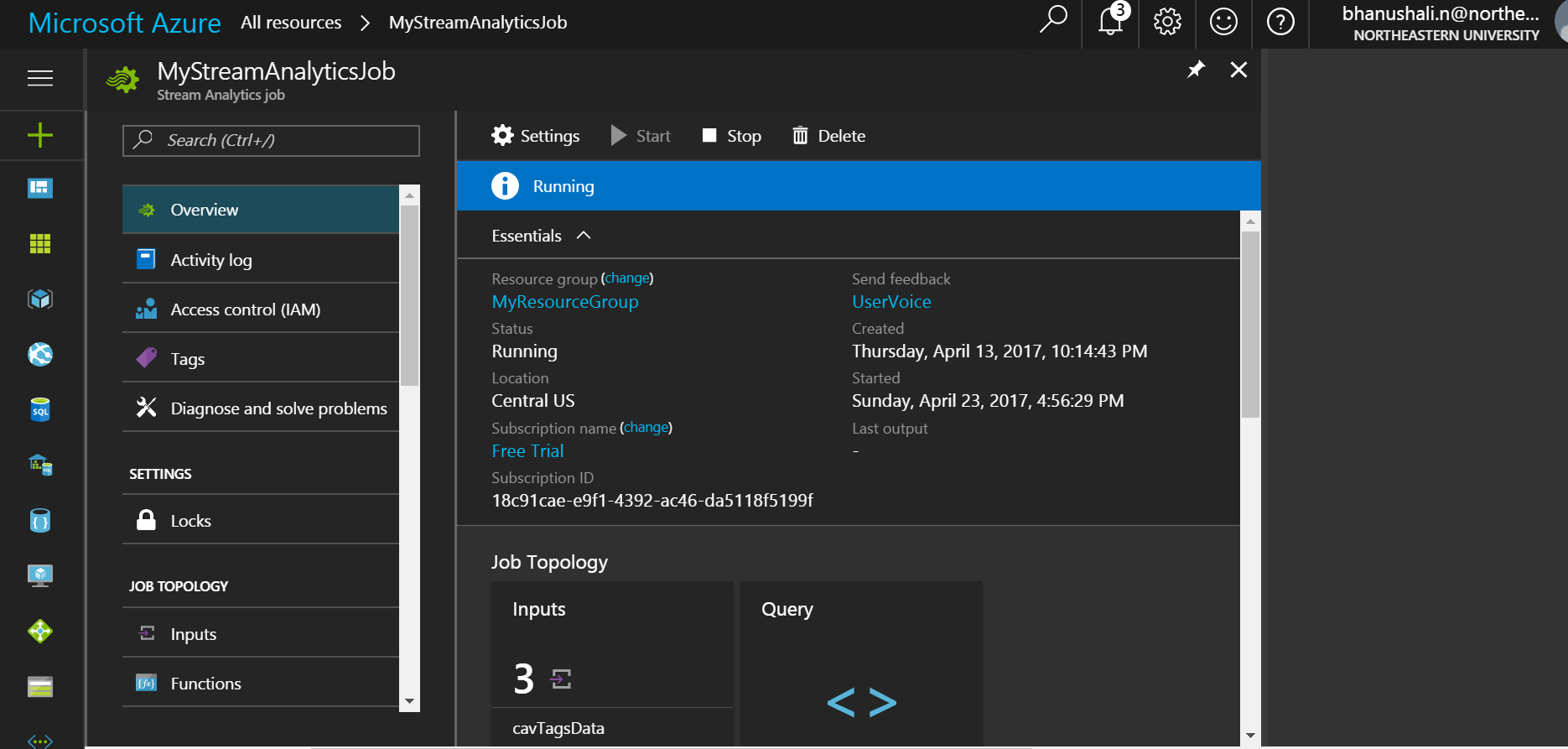
Parse json and send them to event hub on Azure.



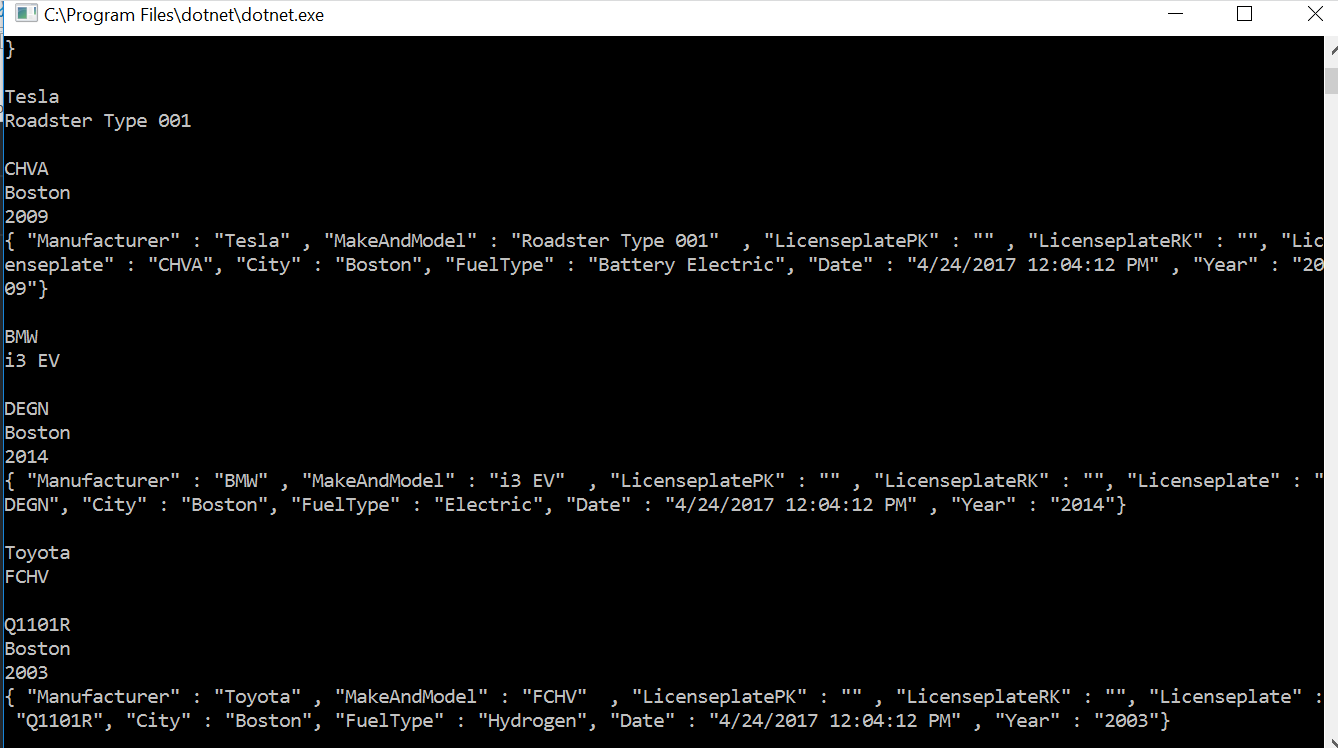
To compile this code in Ubuntu, we used MONO and MonoDevelop. Below is the screen shot for entry stream handler running in mono.



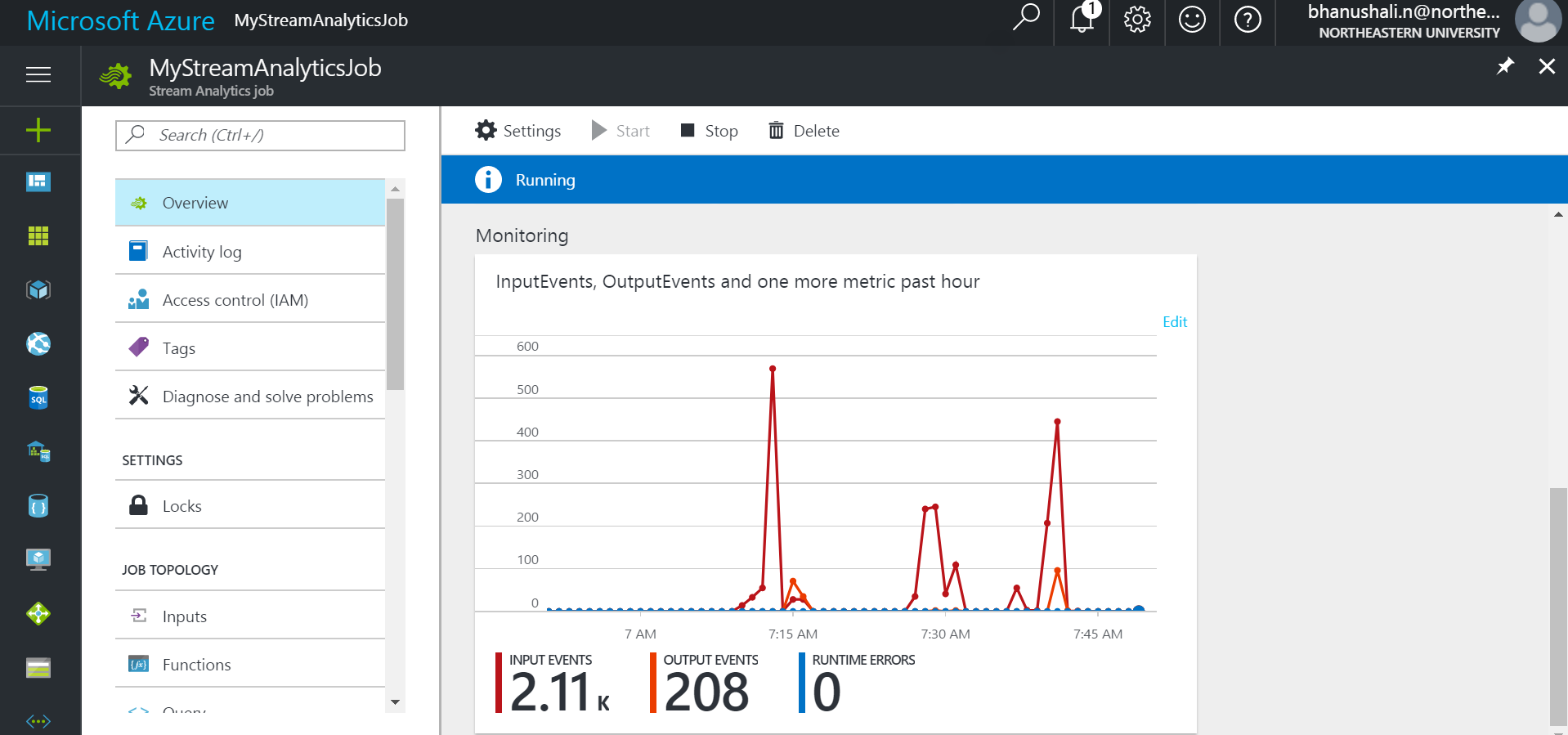
**Step 4: Executing SAJ job**

To start the stream analytics job, select it in the azure portal and click ‘start’

Once the job is running, the event hubs listener can rely the data to the hub. This is shown in the below screenshot.

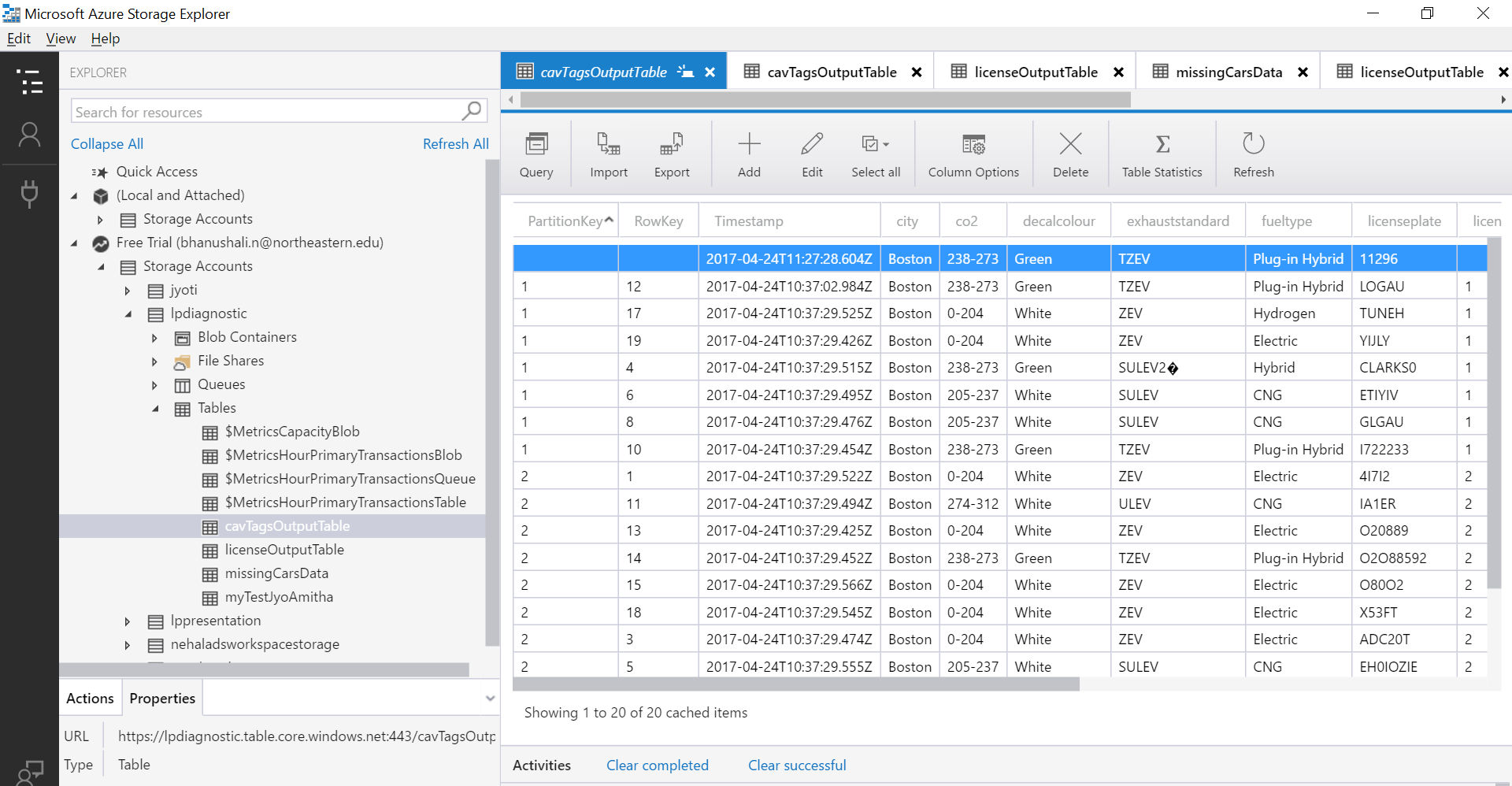


After a couple of mins, we can see a visible spike in the stream analytics monitoring graph.



**Phase II output:**

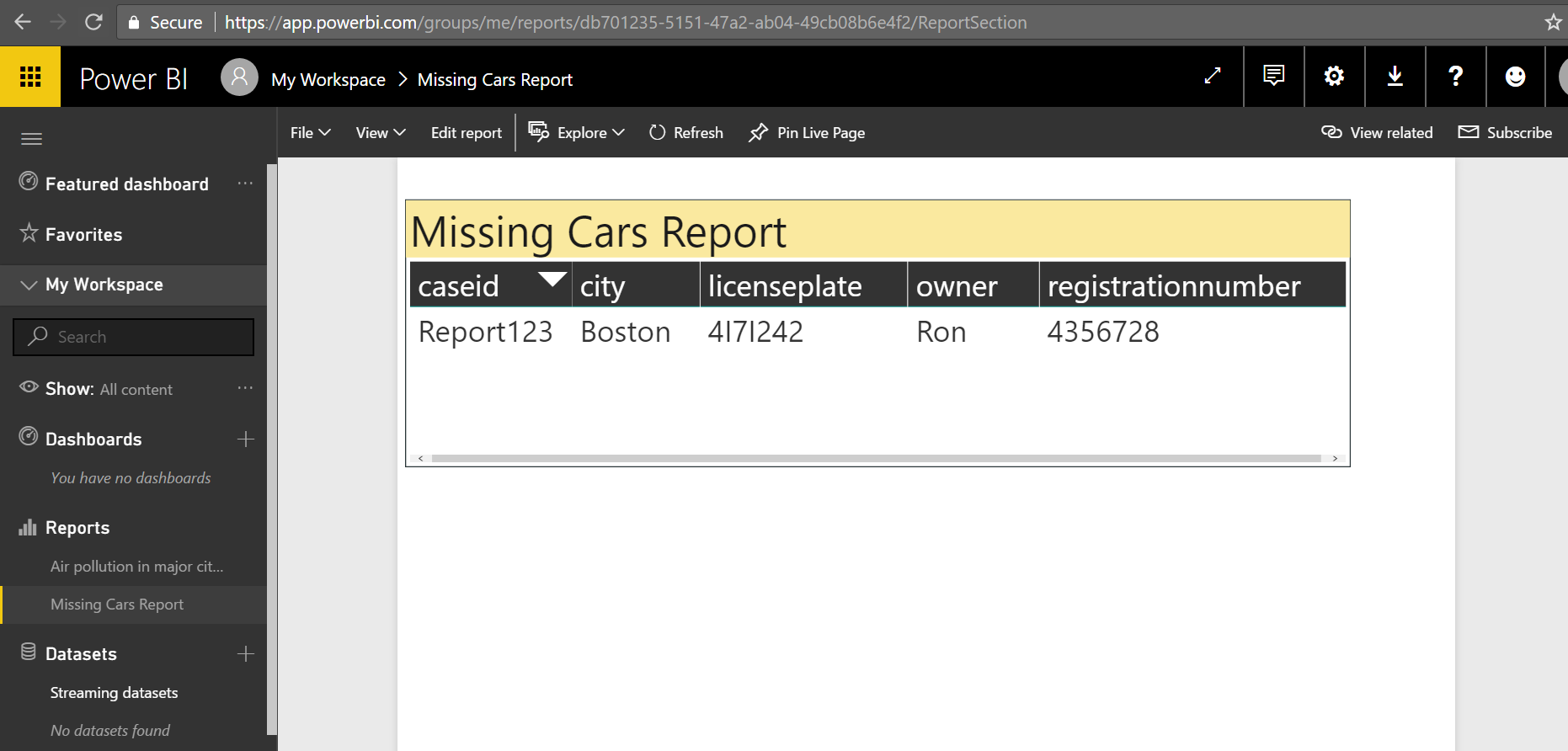
To view the data in Azure, we use Azure storage explore. Microsoft Azure Storage Explorer (Preview) is a standalone app from Microsoft that allows you to easily work with Azure Storage data on Windows, macOS and Linux.



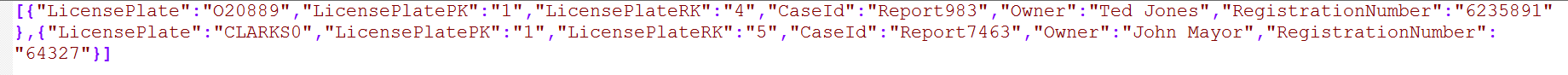
Power BI output files have been included below:

**Use Case 1 – Car Theft Detection**

In this scenario, we have another stream of data (can be kept as reference data depending on frequency of change). When the entry stream is received, we check if any of the license numbers are spotted in the reference file of stolen cars and if yes, then authorities are notified.



This is a screenshot of our missing cars:



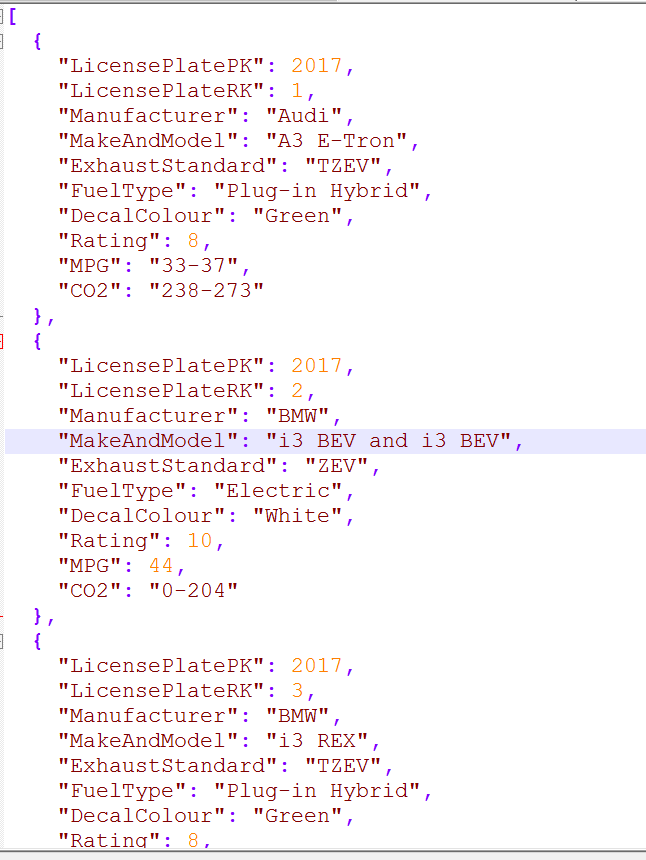
**Use Case 2: Air pollution in major cities of USA**

A vehicle that meets specified emissions standards may be issued CAV decals that allow the vehicle to be operated by a single occupant in the HOV (carpool or diamond) lanes of California's freeways.

White CAV decals are issued according to the following criteria:

* A vehicle that meets California's **super** ultra-low emission vehicle (SULEV) standard for exhaust emissions **and** the federal inherently low-emission vehicle (ILEV) evaporative emission standard. This includes certain **zero-**emission vehicles (ZEVs). Vehicles that meet these requirements are certified pure zero emission vehicles (100% battery electric and hydrogen fuel cell), liquefied petroleum gas (LPG) and compressed natural gas (CNG).
* A 2004 model-year or older vehicle that meets the California ultra-low emission vehicle (ULEV) standard for exhaust emissions **and** the federal ILEV standard.

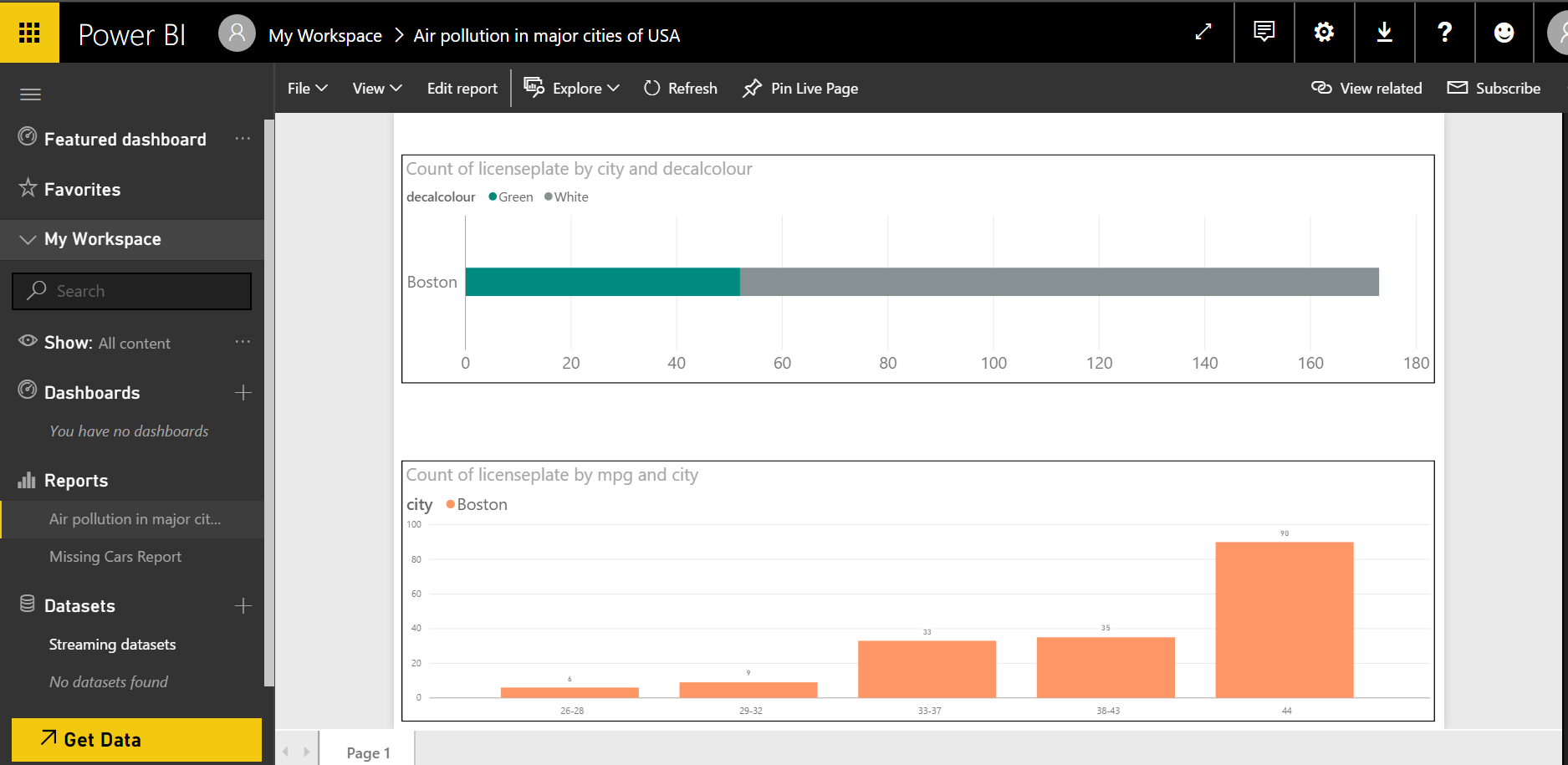
Green CAV - A vehicle that meets California’s Enhanced Advanced Technology Partial Zero-Emissions (Enhanced AT PZEV) or Transitional Zero-Emission (TZEV) standards for exhaust emissions.  Vehicles that meet these requirements are usually plug-in Hybrids.



United states Environmental Protection Agency has provided the below table representing the CO2 and MPG levels and their respective ratings.We have used the combination of these two tables:



The below screenshot depicts the cities and their decal tag ratios. Also, the count of vehicles emitting CO2 and MPG.



**Challenges and Learning outcomes**

* Converting video to individual component images.
* Parallelization of images to make them compatible with hadoop
* Since the mapper in hadoop accepts a text file and reads each line at one time, the options/ways to input images in the mapper was discussed at length.
* Capturing license plate data using openALPR
* The challenges faced in this task were pertinent to the quantity and quality of images generated from the video.
* Better  image quality was required to facilitate recognition of license plate information  with a higher confidence.
* In order to achieve the above, the number of frames per second were to be increased to create a larger pool of images for the openALPR library to choose from, and consequently increase the confidence.
* This created additional time and data for hadoop to process.
* Arriving at an ideal fps (frames per second) number and still maintaining reasonable processing times was a fairly challenging task.
* Data cleansing and processing
* To scale up map reduce performance, we tried setting up clusters on Azure HD insight. But our phython code make it hard to run on Ubuntu.
* Significance of Partition Key and Row key were tricky and hard to figure out.

**Future Scope**

* Scaling up application to process real time stream data emitted from CCTV video cameras.
* This processed data can be also be used for detection and prevention of traffic congestion, based on traffic movement patterns.
* Automation of E-Zpass toll collector and recommendation of appropriate number of toll booths to reduce traffic bottleneck situations.

**Presentation Link**

<https://prezi.com/xusx-vdbosup/stream-analytics-on-vehicular-traffic/?utm_campaign=share&utm_medium=copy>

**Reference**

1. Stream analytics job setup:

<https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-build-an-iot-solution-using-stream-analytics#introduction>

1. Event hub listener:

<https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-dotnet-standard-getstarted-send>

1. FFMpeg :

<https://www.ffmpeg.org/ffmpeg-formats.html>

1. Open ALPR :

<http://doc.openalpr.com/>

1. CAV DECAL :

<https://www.dmv.ca.gov/portal/dmv/detail/vr/decal>

1. Green House Gas and smog Ratings

https://www.epa.gov/greenvehicles/greenhouse-gas-and-smog-air-pollution-ratings-required-smartway-certification