Title: C#, IoT Hub, ASA, PBI, Blob and SQL2016 for Simulating Vehicle Running LA<->NY  
  
Overview  
As a continuous journey for our dream summer trip. we updated the running vehicle simulator which simulates vehicles starting from one of 171 exit points on the route (mainly along Interstate 15, 70, 76 and 95) and moving in steps to east (direction=1) until reaching one of the ends of the route and then immediately changing the direction toward the other end. The device will send out a JSON message to an IoT hub every 10 seconds. A message contains the DeviceId, EventDateTime, Latitude, Longitude and engine Temperature, representing the vehicle telemetry.

This simulator has the following seven components (as shown in the Figure\_1):  
1. Four C# stand-alone apps:

* CreateDeviceIdentity: To register a device to the IoT hub. This app only use once for each device.
* SimulatedDevice: To simulate the vehicle movement and send the telemetry JSON message to the IoT hub.
* SimulatedDeviceReceiver: To wait for the coud-to-device message from IoT hub and acknowledge it upon receiving.
* SendCloudToDevice: Based on the last three telemetry datapoints’ average temperature decides if the vehicle engine is too hot. If so send a cloud-to-device message and wait for the acknowledge from the device.

2. Azure IoT Hub: A subscription is created to explore the following functionalities:

* Device registration
* Sending/Receiving device-to-cloud messages
* Sending/Receiving cloud-to-device messages

3. Azure Stream Analytics: A job is created to import the vehicle JSON messages from the Azure IoT Hub, and export the selected messages to Power BI for visualization and Azure Blob for long term storage.

4. Microsoft web Power BI: A dataset is created to consume the vehicle JSON messages received from the Azure Analytics Services and display them in a global map.

5. Azure Storage Account: A subscription is created to use its Blob storage for long term persistency of the telemetry messages.

6. Azure Logic App: A web logic app is created to hook up the IoT hub and an IaaS SQL 2016 database.

7. An IaaS SQL 2016 database: To hold the hot data and two stored procedures

* InsertRow: This SP is to insert the JSON string into a table dbo.raw\_input. The logic app calls this SP to add the device telemetry message into the db.
* CheckPoint: This SP is written in R for detection of engine temperature anomaly (defined as the average of the last three temperature measurement >= a threshold). The C# app SendCloudToDevice calls this SP every 3 seconds to seek if temperature anomaly was detected.



Figure \_1 Running Vehicle Simulator

About C# Apps CreateDeviceIdentity and SimulatedDevice  
The original code is from Dominic Betts’ blog, entitled “Get started with Azure IoT Hub for .NET” at <https://azure.microsoft.com/en-us/documentation/articles/iot-hub-csharp-csharp-getstarted/>.

As a new requirement in IoT hub, each device should have a unique ID and registered with the hub. Upon successful registration, a device key will be returned from the hub. The C# app CreateDeviceIdentity is for this purpose.

To read in CSV file on the exit points (town name, latitude, longitude, totally 171 such points from LA to NY roughly equal-distance distributed on the route), we installed package Microsoft.VisualBasic as well as Microsoft Azure Service Bus. The attached source code Program.cs contains four classes:  
  
• ExitPoint: Hold the information about each exit point.  
• LAtoNY: A list of exit point Device and a couple of convenience functions.  
• Device: One can image that each vehicle has been installed a telematics device which can send out signals (JSON) to a hub.   
• Program: Send out the device JSON message identified by the device key to Azure IoT Hub specified by IoT Hub Name and connectionString.  
  
A reasonable vehicle speed can be set by adjusting the parameters stepNeeded and thread sleep time. Between two adjacent exit point, the vehicle (aka device) location is a linear interpolation between these two points’ latitudes and longitudes based on the ratio of currentStep/stepNeeded. The simplest way is to set stepNeeded to 1. Which means in each simulation step, the vehicle will jump from one exit point to its adjacent one.

# About C# Apps SendCloudToDevice and SimulatedDeviceReceiver

The original code is from Elio Damaggio’s blog, entitled “Tutorial: How to send cloud-to-device messages with IoT Hub and .Net” at <https://azure.microsoft.com/en-us/documentation/articles/iot-hub-csharp-csharp-c2d/>.

C# app SendCloudToDevice calls SQL DB Store Procedure CheckPoint every 3 seconds to see if there is any engine temperature anomaly. If existed, then it creates and then sends out a cloud-to-device message and then waiting for the device acknowledging back.

C# app SimulatedDeviceReceiver is waiting for the cloud-to-device message from IoT hub and acknowledge it upon receiving.

About IoT Hub Setting  
From the ARM portal, [https://ms.portal.azure.com](https://ms.portal.azure.com/)¸ you can create an IoT hub and pick the free tier (limited to 5,000 messages daily) and use the $default consumer group.   
  
About ASA Setting  
Now you can create the Azure Stream Analytics Job from the ARM portal. After the creation, you need to configure the input, query and output.  
  
To configure the input, you need all the info from the IoT hub which you have been configured. After the input created, you may test the connectivity (to the hub).  
  
The query is for sampling the data. For now, we just take all the incoming JSON messages and thus the query is the following

With AllEvent AS (SELECT \* FROM jackiotinput)

SELECT \* INTO jackiotoutputpbi FROM AllEvent

SELECT \* INTO jackiotoutputblob FROM AllEvent

The output names are jackiotoutputpbi for Power BI output and jackiotoutputblob for Blob.  
  
To configure the Power BI as output, you need to have a Power BI subscription first. After entering the Power BI subscription in this step the connectivity (to PBI) will be tested before you can go any further. You need also to name the dataset for PBI.

To configure the Blob as output, you need first have a Azure Storage Account. Then you may create the container, and the path /sensor/{date}/{time}. Currently the pattern for data and time are fixed to YYYY/MM/DD and HH. Therefore in the Blob container you will see the JSON files are saved in a holder structure, such as /sensor/2016/08/09/20/0\_95cfce069c3c460c8089a66ce06fbb10\_1.JSON

About PBI Setting  
After you start ASA and Event Hub, and then the Sender.exe, you should see the JSON message flow through the system. You may first observe them in Event Hub Dashboard, and then you can sample the JSON message in ASA input (a sample file attached). After you verified the ASA input sample, you bring up web Power BI from [https://powerbi.microsoft.com](https://powerbi.microsoft.com/). Once you signed in, you should see the dataset. Then you can filter it and graph it with Global Map, and then save it in a report and finally pin the report to a Dashboard. Here is an example (see Figure 2).

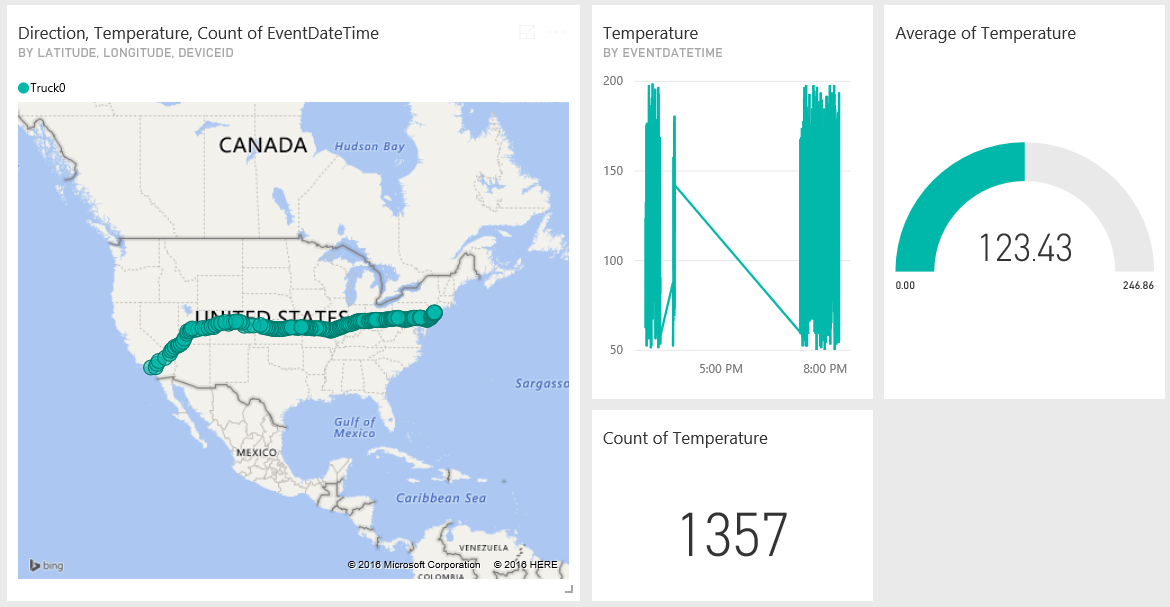


Figure Power BI Dashboard

About Logic App/App Service/App Service Plan Setting

From the ARM portal, you may first create an App Service Plan with the Basic tire price. Then you may create an App Service in this plan. The app service is for Event Hub with external URL of <https://github.com/logicappsio/EventHubAPI>. This app service’s URL will be used by the logic app.

Now you may create the logic app with a trigger with trigging URL from the app service and a step of Execute Stored Procedure. The db credential and SP name are needed.

About IaaS SQL 2016 DB Setting

The IaaS SQL 2016 was created from the portal image. Afterward, newer version JRE was installed. R-Service are turned on. A two-column table dbo.input\_row was create to hold the device JSON strings.

The stored procedure InsertRow is to insert the JSON string into a table dbo.raw\_input. The logic app calls this SP to add the device telemetry message into the db.

Additionally, a view dbo.VehicleTelemetry was created to showcase new JSON functionality within SQL Server 2016. The logical view separates the DeviceId, EventDateTime, Temperature, Latitude, Longitude and Directrion from the raw JSON message stored in the ‘raw’ column, within the dbo.raw\_input table. This view can be used by other database objects so that JSON parsing functions do not have to be continuously re-used.

The stored procedure CheckPoint first get the latest three entries (for a device) then using the newly provided SQL DB functionality of JSON parsing to extract the needed fields from the JSON string and then use R-language to detect the engine temperature anomaly (defined as the average of the last three temperature measurement >= a threshold). The C# app SendCloudToDevice calls this SP every 3 seconds to seek if temperature anomaly was detected.

Final Remarks  
We have taken the advantage of IoT Hub rich functions, SQL 2016 new functions, and dynamic update of Power BI dashboard, a function provided by ASA and PBI integration. However, we feel that the C# app SendCloudToDevice should be replaced by implementation of some Azure services. We also noticed that IoT hub has another function: pushing file to device, which we have not touched.

KC Munnings contributed whole logic app/app service/app service plan part as well as db with stored procedures. Help and guidance from Joe Carver, KC Munnings, and Bunty Ranu are greatly appreciated.