NXP – Innovation Project

Intro

Solution overview

Aim of proposed solution is to connect existing NXP Hardware with chips as reference architecture with Microsoft Cloud Services to enable the observation of location and type of detected objects. Basically 2 verticals are addressed: BlueBox reference scenario as box in a car to bridge car to cloud connectivity and a web serivce architecture that collects the stored information and displays it in an interactive map view. NXP chip technology is acquiring local data, act, transfer to the cloud, gain knowledge and receive data back.

Key technologies used

- Device: Windows IoT, device is developed by NXP
- Simulator Ingestion: IoT Hub, Web App
- Compute: Stream Analytics, SignalR
- Storage: Azure SQL Storage, Azure Blob Storage

Core Team (sorted by name)

- Daniel Heinze, Technical Evangelist
- Oliver Keller, Audience Evangelism Manager
- Ulrich Neidel, Technical Evangelist

Customer profile

NXP® Semiconductors enables secure connections and infrastructure for a smarter world, advancing solutions that make lives easier, better and safer. As the world leader in secure connectivity solutions for embedded applications, NXP is driving innovation in the secure connected vehicle, end-to-end security & privacy and smart connected solutions markets. Industry: Automotive, SmartHome, Vertical: High Tech and Electronics.

Problem statement

High competition in chip Market, customer needs to lead with innovative solutions aiming at being e.g. central innovation driver for Highly Automotive driving. Customer is bridging chip technology with cloud services to enable cross vertical eco system play and connect “dots” with connectivity and cognitive services to enable innovation in the eco system.

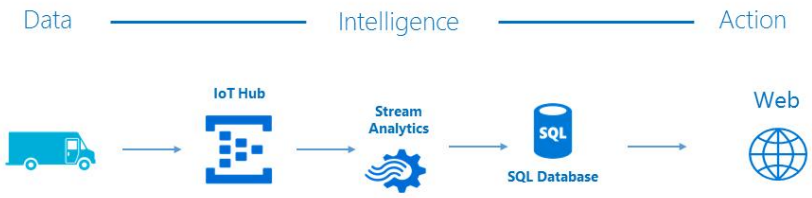
Solution and steps

Aim of proposed solution is to connect existing NXP Hardware with chips as reference architecture with Microsoft Cloud Services to enable connectivity and cognitive services. Basically 2 verticals are addressed: BlueBox reference scenario as box in a car to bridge car to cloud connectivity and speaker/microphone architecture for open SmartHome reference architecture with LUIS interface. Both scenarios have in common that NXP chip technology are acquiring local data, act, transfer to the cloud, gain knowledge and receive data back.

Architecture

The architecture of the pipeline is given below. It includes the sending device (car) which sends data to an IoTHub. Then, a StreamAnalytics job picks up this data and sends it to a SQL database. The Web Application, which displays the data then gets changes from the database and displays it in a Bing Maps view on the ASP.Net website.

This architecture was chosen to ensure two critical properties of the soultion: Every datapoint has to be stored in an easily accessible and structured way (SQL Server) and the solution should have a fast response time to changes in the device, which is ensured through IoT Hub and Stream Analytics. Azure IoT Hub is chosen instead of Event Hub to be able to send messages back to the device at any given point in the future without any changes to the architecture.




```
{
   "lat": "xxxx",
   "long": "xxxx",
   "sign": true
}
```

```
SELECT
   langitude, latitude, sign
INTO
   SQL_Database
FROM
   IoTHubDataInput
```

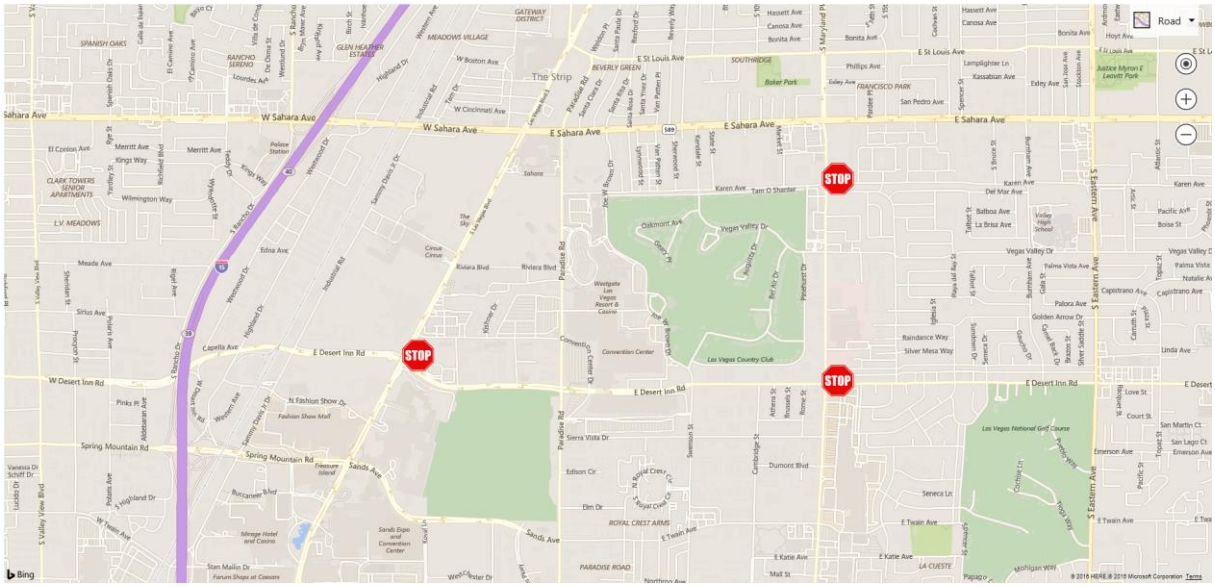
Contents

The repository features 2 applications:

1. A data generator that creates random data files in the following JSON format. This is send via the IoTHub, over a Stream Analytics Service to a SQL Database. The frequency of sending is 10Hz. The package size per send is below 5KB.

```
{
    "DeviceID":"BlueBox1",
    "Latitude": 30.272164093811060,
    "Longitude": -97.739044053646921,
    "StreetSign": "STOP"
}
```

7. An ASP.Net website, which displays a interactive Bing Maps UI, which displays the latest STOP signs stored in the SQL database.



Getting started

Prerequisites

To run the application, you need to build the IoTHub, StreamAnalytics job and SQL database yourself. You can find guidance on how to do this here:

- IoTHub
- Stream Analytics

  The Stream Analytics query can look like this:

  ```
  SELECT
      DeviceID, Latitude, Longitude, StreetSign,
      EventEnqueuedUtcTime AS [TimeStamp]
  INTO
      SQLOutput
  FROM
      IoTHubInput
  ```

- SQL Database

  A SQL database query can look like this. The table called locationdata should therefor contain a table with 5 columns: DeviceID, Latitude, Longitude, StreetSign and TimeStamp.

  ```
  SELECT TOP (1000) [DeviceID]
      ,[Latitude]
      ,[Longitude]
      ,[StreetSign]
      ,[TimeStamp]
  FROM [dbo].[locationdata]
  ```

Keys and Settings

After you created the pipeline, you need to ensure the following settings:

1. Set your SQL credentials in the Web.config file of the SignalRService. Replace the connectionString value with valid SQL credentials.

```
<connectionStrings>
    <add name="DbConnection"
         connectionString="ADD YOUR SQL CREDENTIALS HERE"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
```

6. Add a credentials file to your DataGenerator, that looks like the class below. Replace the IoTHub Uri und Device Key to connect to your pipeline.

```
namespace DataGenerator.Objects
{
    public static class Credentials
    {
        public static string IOT_HUB_URI = "IoTHub URI";
        public static string DEVICE_KEY = "DEVICE KEY";
    }
}
```

14. A blob storage can be added to display a car driving a pre-defined road, a a fail-safe. The received blob data will be send via SignalR to every listening device. Here is the code to retreive the blob data:

```
private string GetCarJson()
{
    // Retrieve storage account from connection string.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
$"DefaultEndpointsProtocol=https;AccountName={Credentials.BLOB_NAME};AccountKey={Credentials.BLOB_KEY}");

    // Create the blob client.
    CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

    // Retrieve reference to a previously created container.
    CloudBlobContainer container =
blobClient.GetContainerReference("locationcontainer");

    // Retrieve reference to a blob named "myblob.txt"
    CloudBlockBlob blockBlob = container.GetBlockBlobReference("positiondata.json");

    string text;
    using (var memoryStream = new MemoryStream())
    {
        blockBlob.DownloadToStream(memoryStream);
        text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());
    }

    return text;
}
```

Conclusion

This section will briefly summarize the technical story of the case:

- The PoC showcases that custom hardware can be easily connected to the Azure IoT Hub and any information transmitted can simply be displayed in a Azure web application.
- Additionally, the PoC can be scaled with ease to include more devices and different web interfaces and applications
- In the PoC, one car was used to recognize road signs. This data was send to the afforementioned pipeline to display the new information inside the Azure web application

Additional resources

In this section, include a list of links to resources that complement your story, including (but not limited to) the following:

- IoTHub

- Stream Analytics
- SQL Database
- SignalR