|  |
| --- |
| Intel Sunny Point Project (Architectural/Requirement Specfication)  [User Story]  This spec describes the Intel Sunny Point Project being design and developed using the Azure IoT solution. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Property | Value | Property | | Value |
| TFS Feature ID | TBD | Doc Status | | Draft |
| Doc Owner | kevika | Milestone | |  |
| **Contacts** | | | | |
| **Role** | | | **Name/Alias** | |
| Program Manager | | | Kevin Kao / kevika | |
| Engineer/Developer | | | Walker Lin / Kevin Kao | |
| Test | | |  | |
| Content Publishing | | |  | |
| Security | | |  | |
| Planning | | |  | |
| Partner(s) | | | Intel | |

# Smart scenario(s)

This document defines the overall system architecture for Microsoft+Intel co-work project, Sunny Point, which is a classical automotive telematics IoT application by collecting driver’s driving records from filed and analyze those data on cloud to evaluate driving behavior.

Intel had developed a new module of OBD-II Dongle, Sunny Point, which embedded with GPS, WiFi, BLE and 3G/4G network connectivity out of box. Compare with existed OBD-II Dongle, driver no longer has to use his/her own mobile phone as GPS sensor and IoT field gateway, which quick often with battery consumption issue on mobile phone.

Accelerometer is another important sensor on Sunny Point Dongle, which offer triaxial accelerometer data, and allow client application (android application) to calculate for specific driving pattern on each case. By this, issuance company can design and offer different issuance package by driver’s driving mileage and behavior.

With Microsoft Azure IoT and Data platform, we are going to answer variable questions from this co-op project with phases approach.

# System architecture

Below diagram is current PoC phase system architecture with below assumption:

1. **Data scale**: There are only few OBD-II Dongle be plug-in on real car to get trip data. With limited timeline and device, we don’t expect to collect huge trip data at PoC phase. By plan, we are going to store couple k trip data by mixed with real trip and simulation data. Hence, we will not touch big data topic at current phase, instead, focus on end to end functionality around UBI scenario.
2. **Geo-IoT Hub**: Considering the solution will cover couple countries, service operators, and even for Intel and Microsoft different demo activities, the running architecture has to support Geo-IoT Hub design on supporting device provisioning and activation.
3. **Security**: To simplify task and integration effort, expect SSL/TLS data encryption between OBD-II Dongle and IoT Hub, by design, there is no extra authentication/authorization happened at PoC phase.
4. **Device Management**: Considering IoT Device Management still on preview version and lack of android SDK, so that, we will not implement fully device management features, but come with configuration management by trailer-made approach.

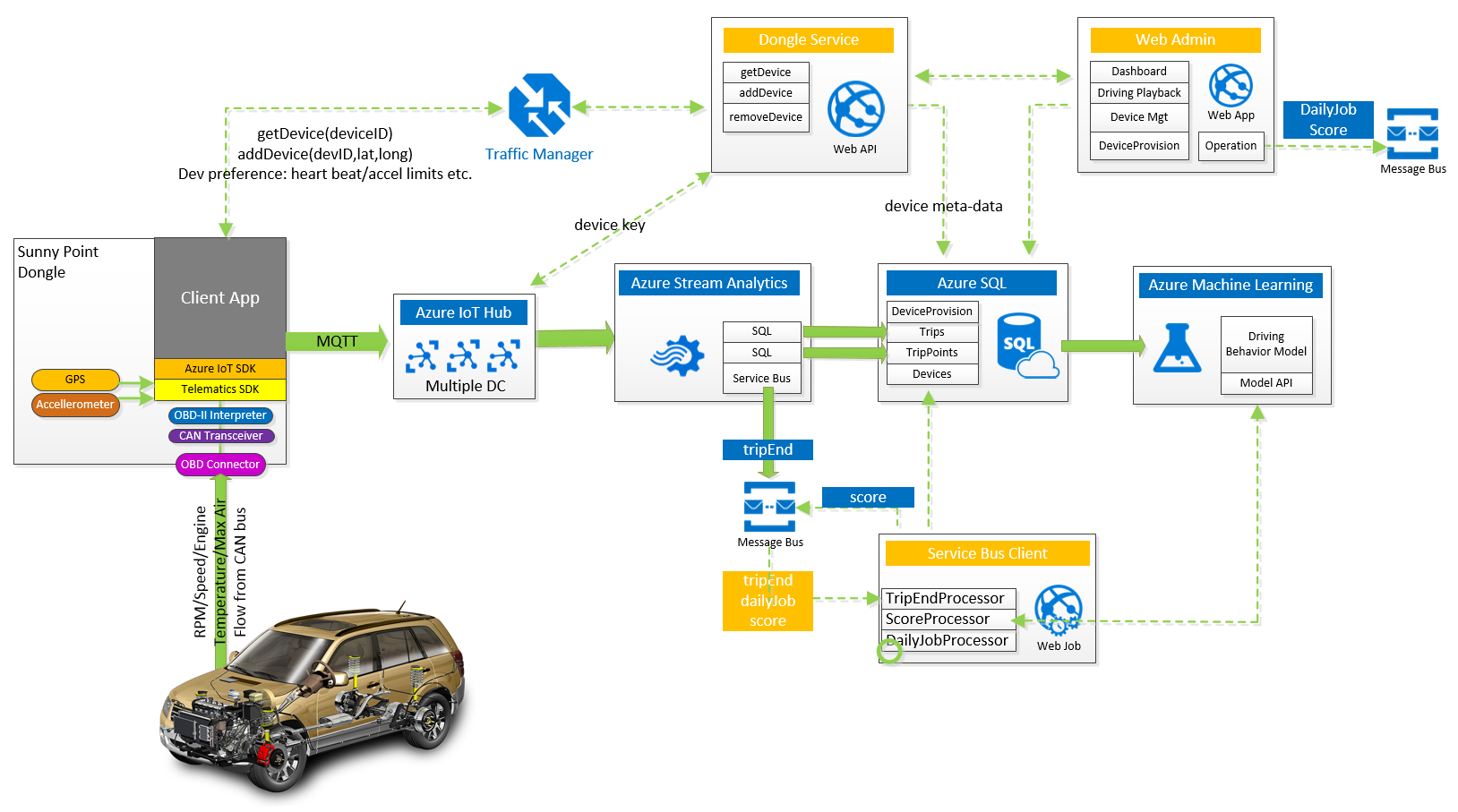


Figure 1: Intel Sunny Point Automotive Telematics System Architecture

## Architecture overview

1. **Car**: The car must come with OBD-II Socket. On-board diagnostics (OBD) is an automotive term referring to a vehicle's self-diagnostic and reporting capability. OBD systems give the vehicle owner or repair technician access to the status of the various vehicle subsystems. The amount of diagnostic information available via OBD has varied widely since its introduction in the early 1980s versions of on-board vehicle computers. Early versions of OBD would simply illuminate a malfunction indicator light or "idiot light" if a problem was detected but would not provide any information as to the nature of the problem. Modern OBD implementations use a standardized digital communications port to provide real-time data in addition to a standardized series of diagnostic trouble codes, or DTCs, which allow one to rapidly identify and remedy malfunctions within the vehicle.
2. **Sunny Point Dongle**: The Intel-based OBD-II dongle is a validated, highly refined platform specifically designed to collect data streams from an array of vehicle buses and sensors and more securely aggregate, analyze, filter, and report that information to drivers, fleet managers, insurance companies, and even private vehicle owners. This platform, built on an open Android system, can reduce the costly and lengthy hardware and development and certification phases, and begin software development almost immediately. The analytics that the dongle makes possible will open new worlds of opportunity for software

Developers and service designers. With this reference design, they will be able to more quickly create robust solutions for fleet management businesses of all types and sizes and for the mass market.

1. **Client Application** on Sunny Point Dongle: A customized android application with integrated with Intel Telematics SDK and Microsoft Azure IoT Device SDK. Not only upload trip message to IoT Hub, client application has to cover dongle activation, application running time configuration, message upload handling by batch and network disconnection, power management and etc.
2. **Traffic Manager**: Azure Traffic Manager plays global-DNS role to offer nearest Dongle Service by driver’s geo-location.
3. **Dongle Service** / Azure Web API: A customized restful Web API service facilitate on Dongle device activation, retrieve configuration of application and know where is right IoT Hub to connect with.
4. **Azure IoT Hub**: A standard Azure IoT Hub service with default configuration.
5. **Azure Stream Analytics**: With input data from Azure IoT Hub and output data to Azure SQL and Service Bus when received [*trip\_end*] message.
6. **Azure SQL**: To store and persistent data including driver profile, device provisioning, device, trip and trip points.
7. **Azure Message Bus**: Setup a message queue on receiving message from Stream Analytics and/or Web Admin console for variable operational cases, such as receive [*trip\_end*] message from IoT Hub, receive manual data clean job from Web Admin, receive driving scope re-calculate from Web Admin.
8. **Service Bus Client** / Web Job: A customized console application with Web Job pattern, which receive all operational messages from Azure Message Bus, and execute each tasks by threading. The tasks including TripEndProcessor, ScoreProcessor and DailyJobProcessor.
9. **Azure Machine Learning**: [To Be Add After Get Real Trip Data]
10. **Web Admin** / Web App: A customized Web application for showcase and demo purpose, including:

* PowerBI Dashboard
* Trip routing with each critical message event.
* Driver driving behavior evaluate and scope.
* Single dongle device provisioning and batch import

# Sunny Point Dongle

## Sunny Point Dongle

|  |  |
| --- | --- |
| Figure 2: Intel Sunny Point Dongle Hardware Architecture | Figure 3: Dongle Sample |

INTEL-BASED OBD-II DONGLE PRODUCT DETAILS

|  |  |
| --- | --- |
| **Security Features** | **Detail** |
| Processor | Intel AtomTM x3-C3205RK   * Intel AtomTM x3-C3200RK quad core processor 64-bit 1.2GHz CPU * A-GOLD 620 |
| Memory | eMMC 4.51 (8 GB) + LPDDR3 (8 Gb; 1200MT/s) |
| Boot time | Cold boot power on (<1min); warm boot power on (<2s) |
| OS | Android\* M, Linux\* (coming soon) |
| USB | USB 2.0 host |
| Security | Secure boot, secure download, secure VM, signing/key handling, anti software rollback |
| Debug | JTAG Header |
| OP Temp | -20 ~ 85C |
| Wireless | BT4.0 LE (HFP, GAT, SPP, etc)  Wi-Fi (802.11 b/g/n) support AP and stand-alone mode |
| Modem | HSPA+/UMTS band 1,2,5,8  GSM/GPRS/EDGE  Single micro SIM (no embedded SIM support) |
| Location | GPS/GLONASS A-GPS  Satellite acquisition after power on in < 20s |
| Accelerometer | Freescale MMA8653FC 3-axis, 10-bit digital accelerometer |
| OBDII | OBD-II interpreter (STN1170)  SAE J1962 connection  Fully compatible with ELM327 AT command set  All legislated OBD-II protocols:   * ISO 15765-4 (CAN), ISO 14230-4, ISO 9141-2, SAE J1850 VPW, SAE J1850 (PWM)   Nonlegislated OBD protocol:   * ISO 15765, ISO 118981 (raw), SW CAN (GMW3089), MS CAN   SAE J1939 (for heavy-duty vehicle) |

## Telematics SDK

Intel Telematics SDK for Android (April 2016, Revision 0.85)

## IoT Device SDK

Microsoft Azure IoT device SDK for Java (2016-08-15 build)

## Client Application

[To Be Add]

# Azure IoT Services Design

## Dongle Device Provisioning

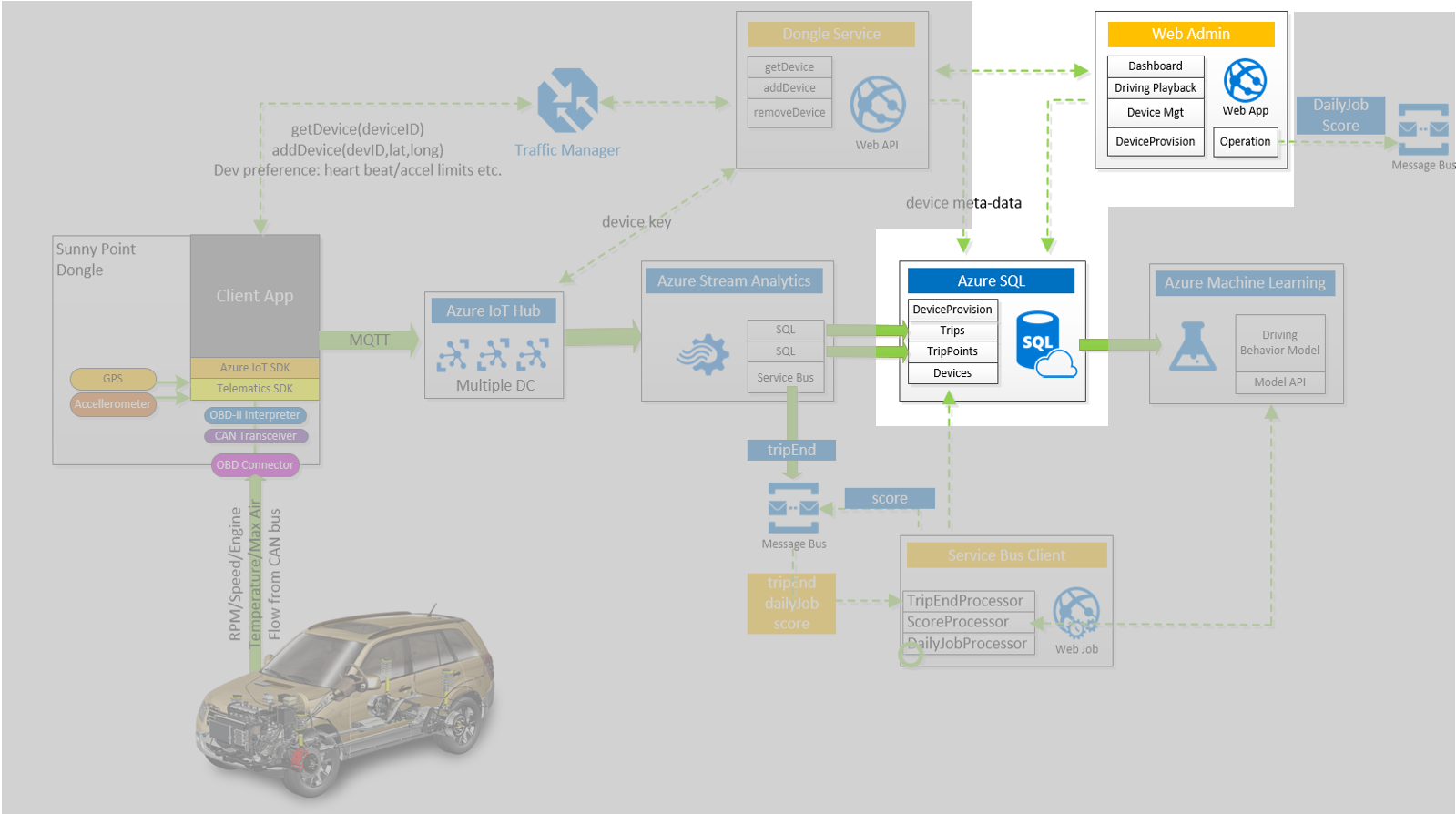


Figure 4: Data Flow on Provisioning batch of Dongle Devices

When a batch (say 10K piece) of OBD-II Dongle be manufactured from factory, there are 3 fundamental information have to uncover before them are able to provisioning to backend service by service operator:

1. Device ID: A unique id binds on each OBD-II Dongle.
2. IoT Hub Host: An Azure IoT hub host name, which the OBD-II Dongle belong to.
3. IoT Hub Protocol: To specify the network connection protocol between Dongle device and Azure IoT Hub, or empty to let it be default value.

Service operator, then, is able to upload a CVS file with above 10K OBD-II Dongle information on Web Admin; to provision 10K OBD-II Dongle devices.

The OBD-II Dongle provisioning data will be persisted on Azure SQL by below schema and sample data.

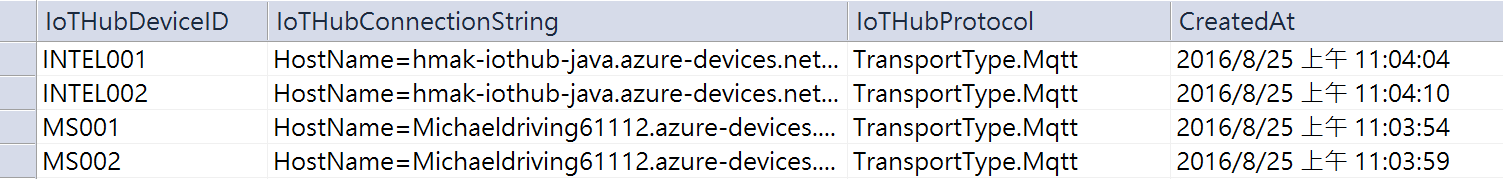


Figure 5: Example of Device Provisioning Data in Database

## Dongle Device Activation

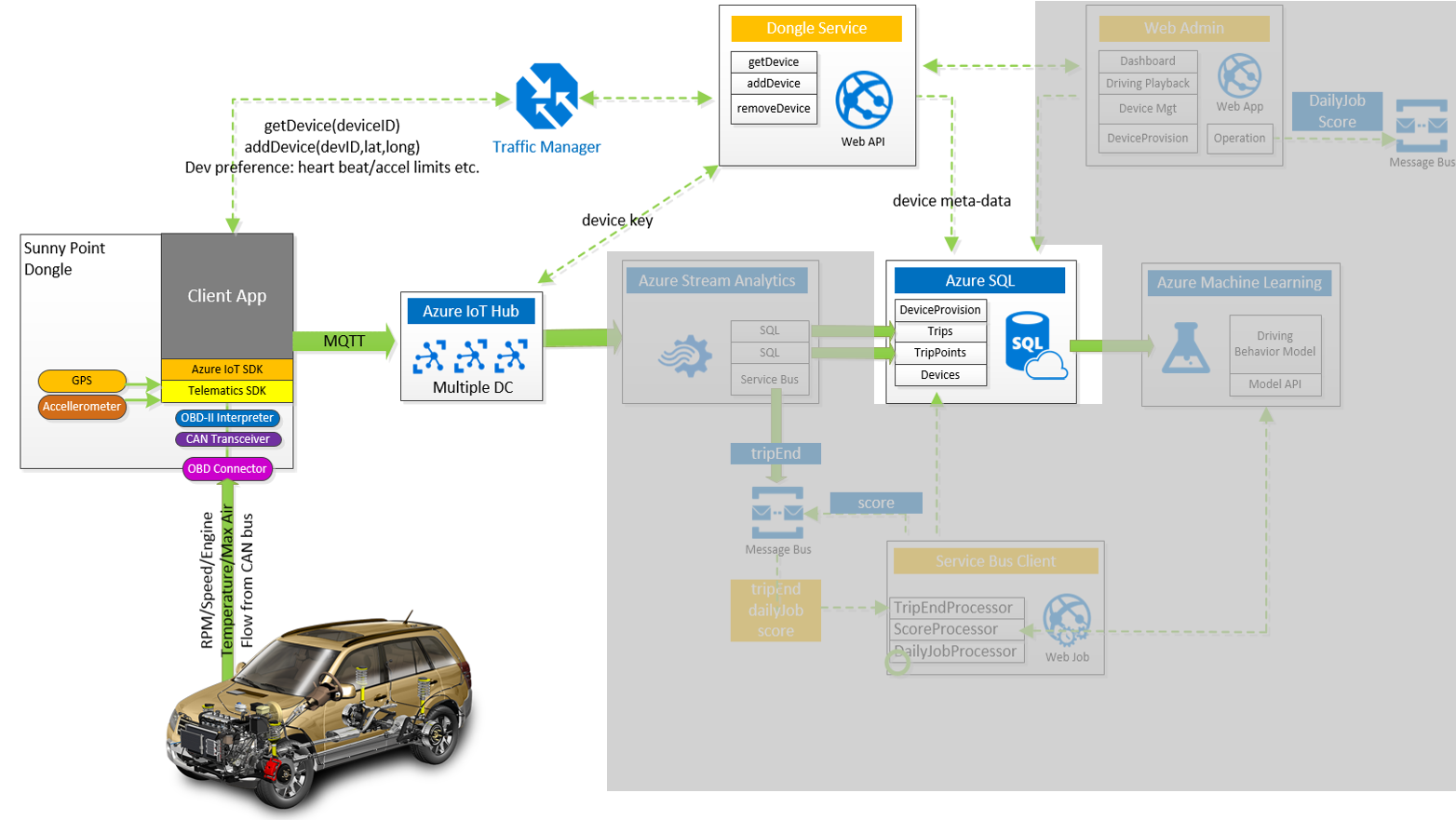


Figure 6: Data Flow on Activate a Dongle Device

When Dongle application be launch, apart from basic and fundamental initial processes, the application has to call getDevice() WebAPI (a URI on Dongle Service Host ) by passing its’ Device ID; of cause, this URI has to hardcode at Dongle application’s configuration file, and this is only one necessary hardcode configuration data.

At first time Dongle application shall get http-status:404 (Not Found) when call getDevice() WebAPI, which is because getDevice module can’t retrieve Device object from Azure IoT Hub.

HTTP-status:404 (Not Found) may also represent the coming Device ID was not be provisioning yet, or which was an invalid Device ID.

If this Device ID already existed in Azure IoT Hub (Of cause, it be provisioning already), the getDevice() WebAPI will return all necessary data around this Device ID.

Dongle application can call addDevice() WebAPI to activate itself on Azure IoT Hub with follow input parameters:

1. Device ID: a unique id binds on each OBD-II Dongle.
2. Latitude and Longitude: which backend service can calculate country from location.

addDevice() WebAPI then,

1. Get provisioning data from Azure SQL by Device ID.
2. Call pre-defined Azure IoT Hub to add this Device ID, and get back Device Key from Azure IoT Hub
3. Output Device ID, Device Key, IoT Hub Host Name, and other run-time configurations to Dongle application.

Of cause, if Device ID is not be provisioning in advance, addDevice() WebAPI will return http-status:404 (Not Found).

## Vehicle Movement Profile

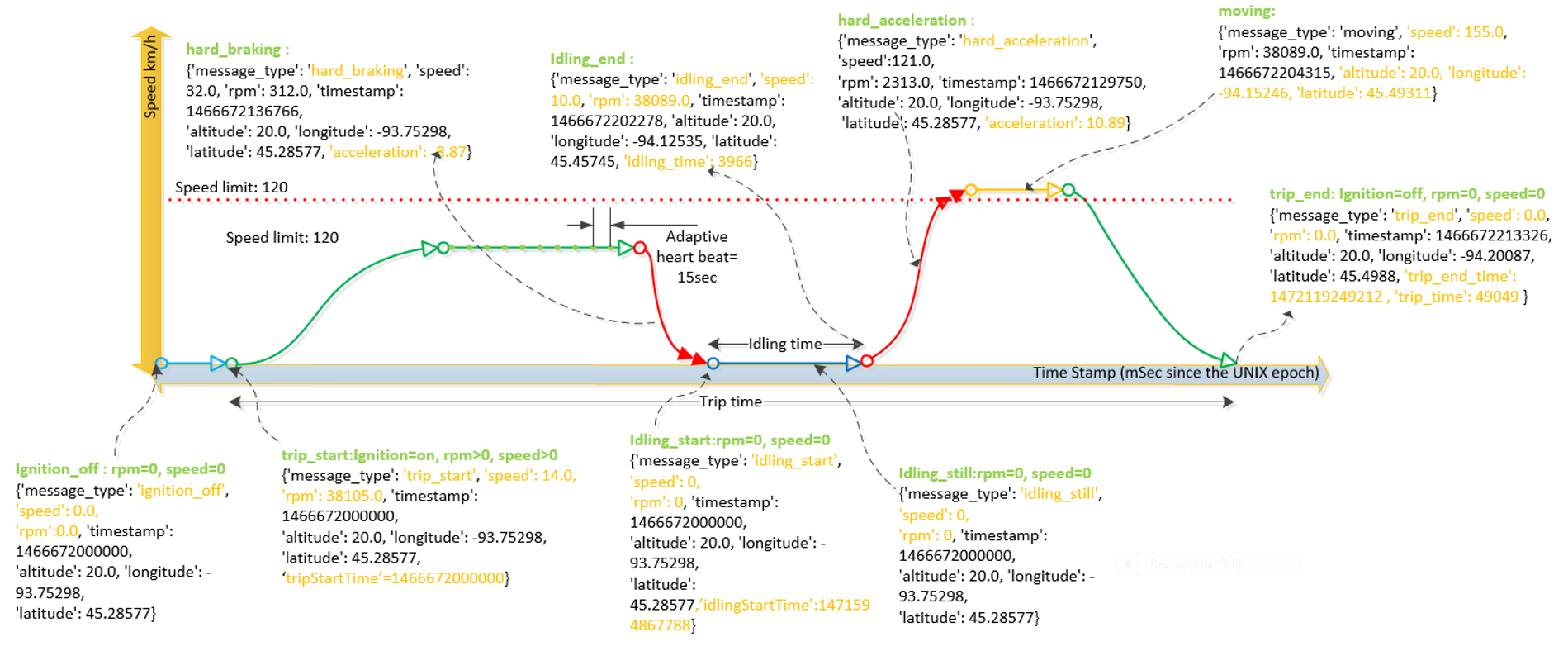


Figure 7: Vehicle Movement Profile

**Message Type**

There are 9 type of message are going to send to Azure IoT Hub from Dongle application, them are:

1. **ignition\_off**: This message either be filter out by Dongle application or Azure Stream Analytics, due to no tripID attached.
2. **trip\_start**: Start moving. Only one trip\_start message per Trip.
3. **hard\_acceleration**: By triaxial accelerometer data v.s. [*HardAccelerationValue*].
4. **hard\_breaking**: By triaxial accelerometer data v.s. [*HardBreakValue*].
5. **moving**: Triger by heartbeat, to track moving location.
6. **idling\_start**: Stop at one place with engine on.
7. **idling\_still**: Stop at one place still with heartbeat trigger.
8. **idling\_end**: Switch from stop status to moving.
9. **trip\_end**: Stop and turn engine off. Only one trip\_end message per Trip.

**Adaptive Heartbeat**

The value of heartbeat interval be retrieved from return of getDevice() WebAPI every time when Dongle application launched. Let us say the heartbeat interval is 15 seconds. Adaptive heartbeat will reset final count to 15 seconds once any message be trigger by event. This mechanism will keep uploading message as less as possible and without noise.

**Send Message Interval**

To optimize network bandwidth and connection efficiency, it is good idea to implement batch data uploading instead of real-time by event. The value of send message interval also be retrieved from return of getDevice() WebAPI.

## Data Flow and Process Flow

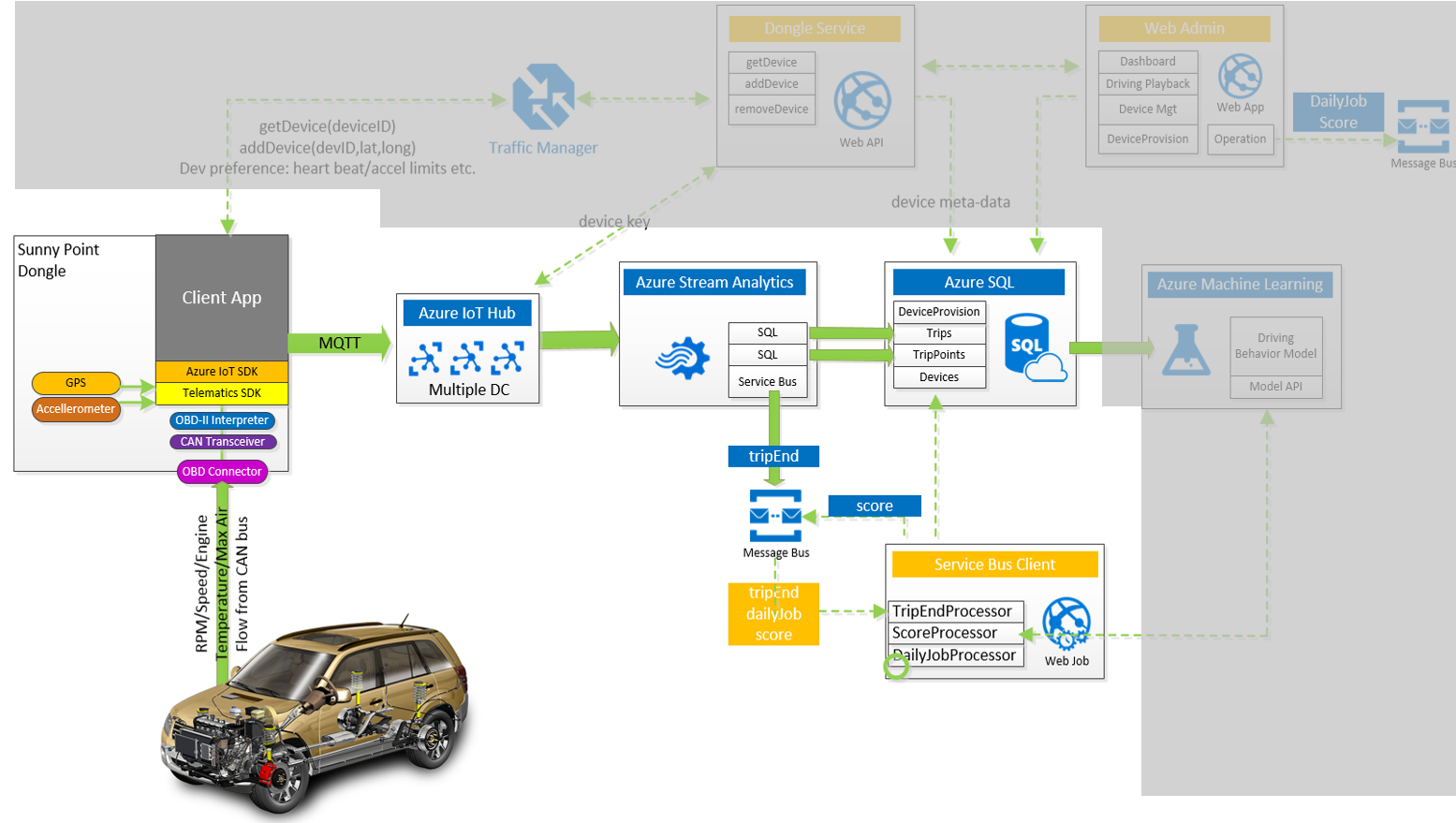


Figure 8: Data Flow and Process Flow for a Driving Message

**TripPoint Message Content**

TripPoint message be generated OBD-II Dongle device, and send over to Azure IoT Hub.

|  |
| --- |
| {  "tripID": "Dev12345",  "carID": "Car123",  "deviceID": "Device123",  "msgType": "trip\_start", /\* Refer to Message Type Definition \*/  "msgTS": 1409883453, /\* The message generated timestamp \*/  "alt": 50.08, /\* Altitude \*/  "lat": 25.3433490, /\* Latitude \*/  "lng": 120.8778934, /\* Longitude \*/  "speed": 76,  "rpm": 2500, /\* Engine Ring per minute \*/  "maf": 87,  "temp": 90, /\* Temperature \*/  "aclX": 1.1,  "aclY": 1.2,  "aclZ": 1.3,  "aclXYZ": 4.1,  "idlingSTime": 1409883453, /\* idling start timestamp \*/  "idlingETime": 1409883453, /\* idling end timestamp \*/  "idlinTime": 180, /\* idling duration in millisecond \*/  "tripSTime": 1409883453, /\* trip start timestamp \*/  "tripETime": 1409883453, /\* trip end timestamp \*/  "tripTime": 180 /\* trip duration in millisecond \*/  } |
| * trip\_start and trip\_end only occurs only one time for a trip. * Mandatory fields: tripID, carID, deviceID, msgType, msgTS, speed. * Location (alt, lat, lng) will be NULL if GPS unlock. * Idling related fields only occurs on message type equals to idling\_start, idling\_still and idling\_end. |

**Trip Data Schema**

Trip record was generated by Azure backend when received trip\_start message. Below is data schema.

|  |
| --- |
| [Id] NVARCHAR (128) NOT NULL,  [CarID] NVARCHAR (128) NULL,  [IoTHubDeviceID] NVARCHAR (128) NOT NULL,  [StartTimeStamp] BIGINT NOT NULL,  [EndTimeStamp] BIGINT NULL,  [StartCountry] NVARCHAR (20) NULL,  [EndCountry] NVARCHAR (20) NULL,  [isComplete] BIT DEFAULT ((0)) NULL,  [Distance] FLOAT (53) DEFAULT ((0)) NULL,  [HardBreaks] INT DEFAULT ((0)) NULL,  [HardAccelerations] INT DEFAULT ((0)) NULL,  [OverSpeeds] INT DEFAULT ((0)) NULL,  [MaxSpeed] FLOAT (53) DEFAULT ((0)) NULL,  [AverageSpeed] FLOAT (53) DEFAULT ((0)) NULL,  [MidNightDriveInSec] INT DEFAULT ((0)) NULL,  [DriveTimeInSec] INT DEFAULT ((0)) NULL,  [IdelingTimeInSec] INT NULL,  [ProcessCompleteFlag] BIT DEFAULT ((0)) NULL,  [Rating] INT DEFAULT ((0)) NULL,  [CenterLat] DECIMAL (12, 9) NULL,  [CenterLng] DECIMAL (12, 9) NULL, |
| These value of fields are come from Trip\_Start point:   * CarID, IoTHubDeviceID, StartTimeStamp   These value of fields are calculated after received Trip\_End message type of TripPoint:   * StartCountry and EndCountry: convert location of Trip\_Start and Trip\_End points to countries. * isComplete: True when received Trip\_End message. * Distance: Calculate whole trip distance by link with tripPoints 1 by 1. * HardBreaks and HardAccelerations: Calculate times from all tripPoints. * OverSpeeds, MaxSpeed, AverageSpeed: Calculate these value from all tripPoints. * MidNightDriveInSec: convert location and timestamp of tripPoint to local time, and identify if Midnight period. * DriveTimeInSec, IdelingTimeInSec: Calculate from all tripPoints. * ProcessCompleteFlag: True when all calculate values done. * CenterLat and CenterLng: Calculate center point from all tripPoints. This is for lockdown center location when load Bing Map. * Rating: Send over trip data to API of Machine Learning and get back score of this trip. |

**Trip Data Calculation**

To accomplish above trip data calculation on trip\_end message, we have to send out an internal message from Azure Stream Analytics to Message Bus, and let service bus client application pick up the calculation task.

**Network Disconnect**

Network connection may be broken, especial when car driving into underground, which means trip\_end message will be lost for couple days till OBD-II Dongle re-connected network. Considering such issue, we have to implement store and forward mechanism on Dongle application to persistent message data on local if network disconnect.

**Trip\_End Message Lost**

It is possible to lost trip\_end message eventually by any reasons. In this case, we shall define a max waiting duration, and having a daily job to send out trip\_end message to IoT Hub to complete the trip. At PoC phase, the max waiting duration will be 24 hours.

If the real trip\_end coming after max waiting duration, we will treat this was duplicate trip\_end case, and ignore it.

**GPS Location Data Lost**

It is possible having missing GPS location message reach to IoT Hub, such as the car move at underground, move into tunnel.

To keep it simple, we will fill previous valid GPS location into these missing messages

If GPS location lost at trip\_start message, then, the value shall be fill with first valid GPS location point.

**Trip Rating**

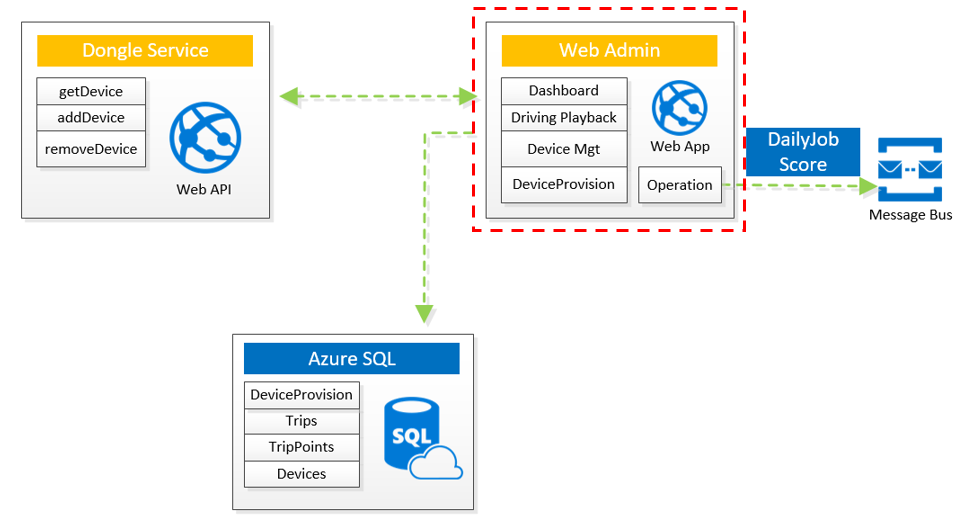
The final step for a trip is to give a rating score base on the rating model we created from machine learning and re-calculate driver’s score from all his/her trips records.

## Machine Learning on Driving Behavior

[To Be Add After Get Real Trip Data]

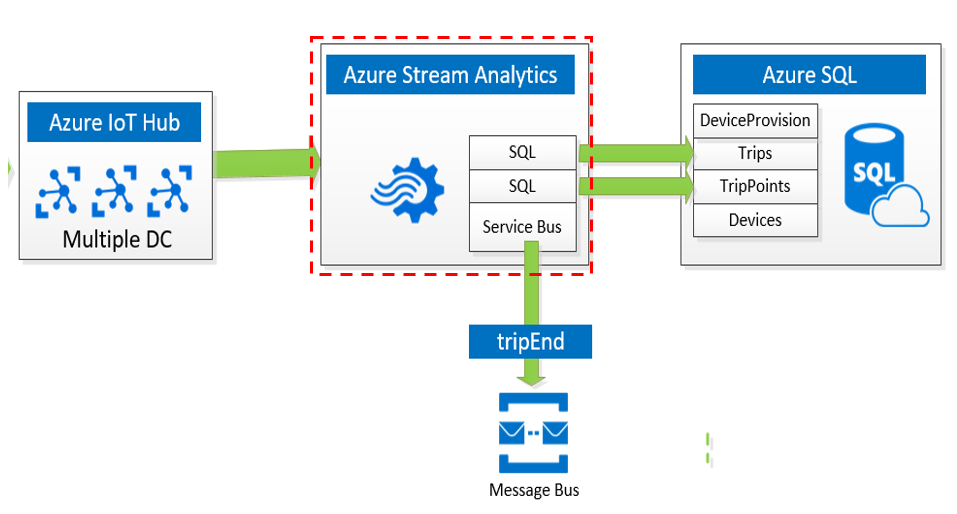
# Detailed feature requirements

## Functional requirements

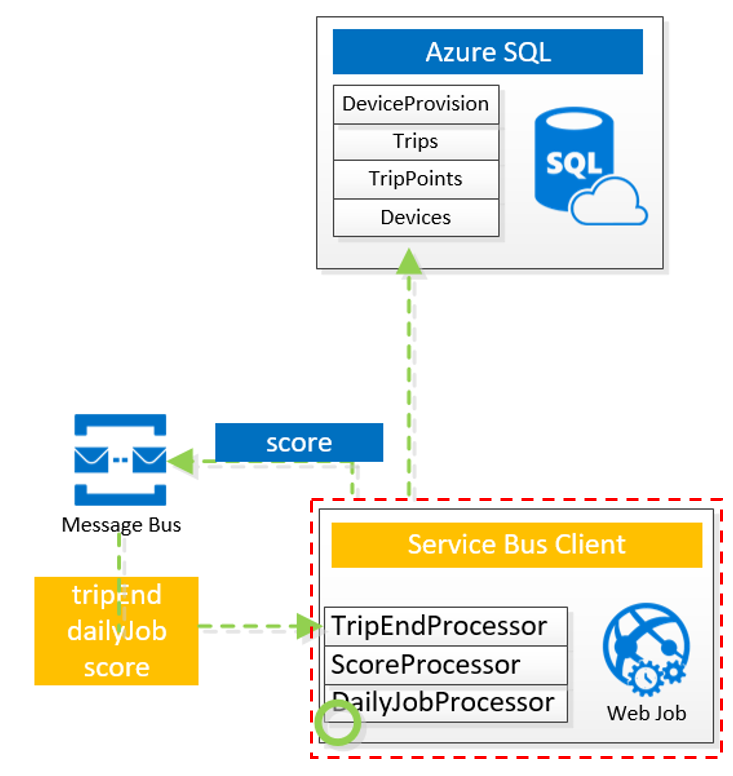


|  |
| --- |
| Web Admin Requirement |
| 1. [Dashboard] A web UI with PowerBI Dashboard to present statistic and analyze of overall driving data. |
| 1. [DrivingPlayback] A web UI with Bing Map component, offer driver/trip selection, driving trip playback and place message type of hardBreak, hardAcceleration, overSpeed and midNightDriving at point of map.   [DrivingPlayback] To present driver’s score at web page. |
| 1. [DeviceManagement] A web UI to setup OBD-II Dongle application run-time configuration, including:  * HardBreakValue * HardAccelerationValue * HeartBeatInterval * SendMessageInterval   [DeviceManagement] The configuration data persisted on Azure SQL. |
| 1. [DeviceProvision] A web UI for uploading a CVS file and provisioning OBD-II Dongle devices which defined at the file.   [DeviceProvision] The provisioning data persisted on Azure SQL. |
| 1. [Operation] A web UI for:  * Input Device ID, and call Dongle Service to remove the device. * Input Driver ID/Name, and send a re-scoring message into message bus. * Send a message to message bus for trigger a data-clean task. |

|  |
| --- |
| Dongle Service Requirement |
| 1. [getDevice] A restful WebAPI (GET /device/getDevice/{deviceID}, which can either get device information back or return 404 (not found) status.  * Retrieve device record from SQL DB/Table Devices. Throw 404 not found if the ID doesn’t exist. * Call IoT Hub (defined on Table Devices) to retrieve Device Key. Throw 404 not found if IoT Hub return nothing. * Return Device object. |
| 1. [addDevice] A restful WebAPI (POST /device/addDevice/{deviceID}, which can either add passed device into associated IoT Hub or return 404 (not found) status.  * Retrieve device record from SQL DB/Table DeviceProvision. Throw 404 not found if the ID doesn’t exist. * Call IoT Hub (defined on Table DeviceProvision) to add new Device. * Insert Device record into Table Devics. * Return Device object. |
| 1. [removeDevice] A restful WebAPI (DELETE /device/removeDevice/{deviceID}, which can either remove passed device from associated IoT Hub or return 404 (not found) status.  * Retrieve device record from SQL DB/Table Devices. Throw 404 not found if the ID doesn’t exist. * Call IoT Hub (defined on Table Devices) to remove this Device. Throw 404 not found if IoT Hub return nothing. * Return OK. |



|  |
| --- |
| Azure Stream Analytics Requirement |
| 1. A SAQL to add Trip record into Azure SQL when received trip\_start message. |
| 1. A SAQL to add TripPoint record into Azure SQL when received all message. |
| 1. A SAQL to add trip\_end message into Message Bus when received trip\_end message. |



|  |
| --- |
| Service Bus Client Requirement |
| 1. [TripEndProcessor] To calculate trip data when receive trip\_end message, which defined at **Trip Data Schema** section.   [TripEndProcessor] To handle GPS location lost cases. |
| 1. [ScoreProcessor] Send over trip data to API of Machine Learning, get back score of this trip, and update trip record of Azure SQL. |
| 1. [DailyJobProcessor] Daily Job on handle:  * Trip\_end message lost over max waiting duration. |

## Not Supported

|  |  |
| --- | --- |
| Non Goals | Mitigation |
|  | V2 candidate |
|  | V2 candidate |

## Architecture component

|  |  |  |
| --- | --- | --- |
| # | Component | Detail |
| 1 |  |  |
| 2. |  |  |
| 3. |  |  |

Table 3

# Future Extension

1. Add Azure Traffic Manager at front of Dongle Service, which allow multiple regions Dongle access with nearest Dongle Service Application.
2. Having primary and secondary Azure IoT Hub host for HA.
3. Add WebAPI access token on Dongle Service Application.

# Related Documents

# Change History

|  |  |  |
| --- | --- | --- |
| Date | Author | Changes |
| 08.30.16 | kevika | First Draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |