

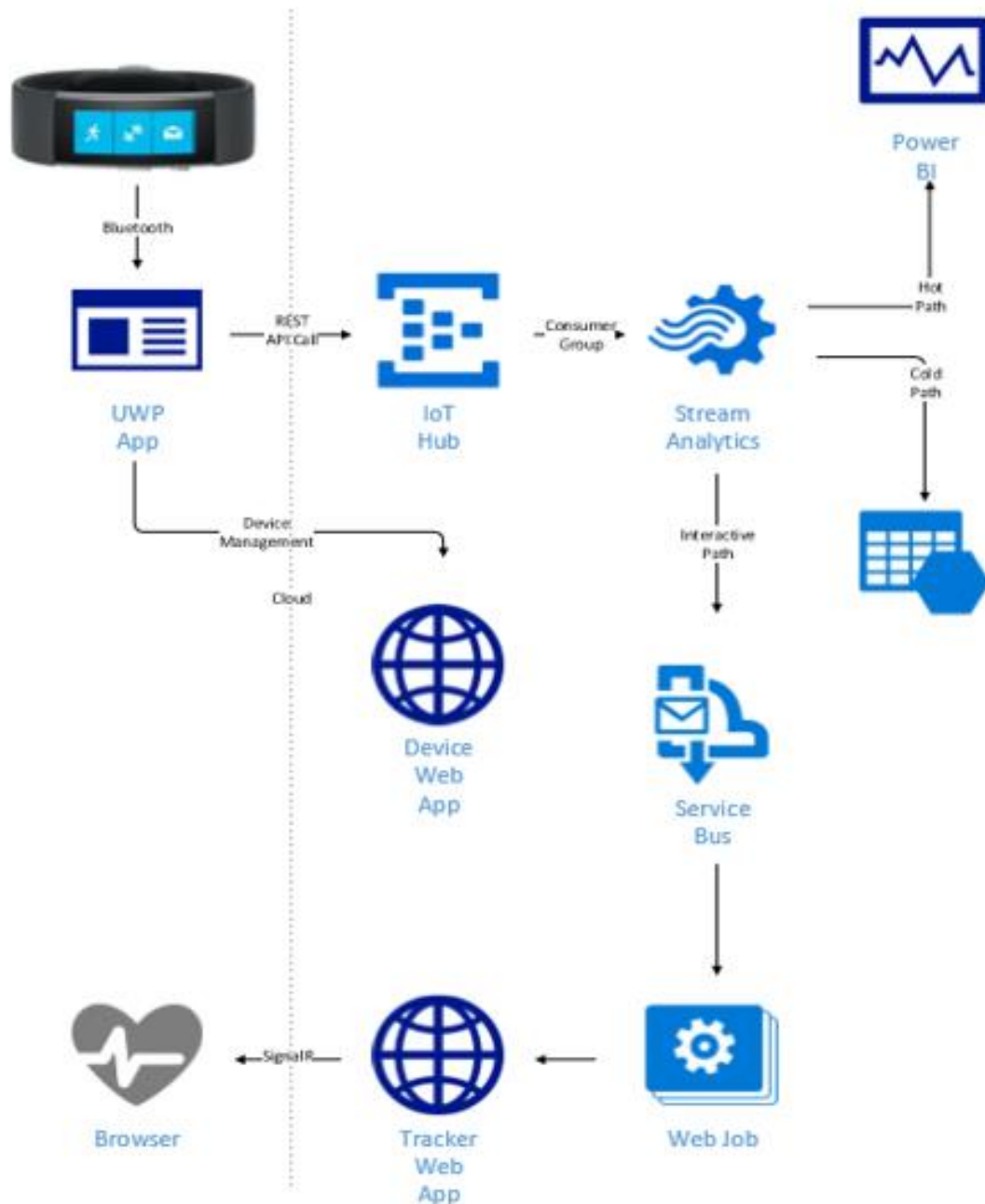
Installation Guide - June 2016

12 May 2016

16:12

Band on the Run Architecture

Overall Band on the Run Architecture



The [UWP](#) app connects via Bluetooth, using the [Microsoft Band SDK](#), to the band to get real time sensor data.

When the app first connects to a band, it gets a special [Azure IoT Hub](#) token from the device web app.

The band sensor telemetry is send via http to the Azure IoT Hub, using the token it obtained earlier.

The [Azure Stream Analytics](#) job processes that telemetry data in real time, and outputs to three different destinations.

The hot path is wired to [Power BI](#), and charts the data in a dashboard in real time.

The cold path writes the data to [Azure table storage](#), for historical analysis and processing.

The interactive path is trigger when the heart rate goes above a certain threshold, and is written to an [Azure Service Bus](#) queue.

An [Azure WebJob](#) pulls data off the service bus and pumps it to the tracker web site, via a REST API call.

The tracker web site uses [SignalR](#) to send the live heart rate data down to browsers, to be displayed in real time

Azure Configuration

This install process is not as automated as much as we would like, and we are working to add that. It's not hard to set the project up, and we have tried to adopt best practices wherever possible.

Going through the steps below will give you a good feel for the project architecture and all the moving parts, so when you come to update or debug the project, it will be much easier.

Pre-requisites

Visual Studio (at the time of writing, we are using Visual Studio 2015 - <https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx>)

Microsoft Azure SDK (at the time of writing, 2.9, installed via the Web Platform Installer at <https://azure.microsoft.com/en-us/downloads/>)

The Azure Cloud explorer is a useful tool to check the data landing in table storage (<https://azure.microsoft.com/en-gb/documentation/articles/vs-azure-tools-resources-managing-with-cloud-explorer/>)

IoT Hub SDK (useful to manage IoT Hub, especially the Device Explorer - <https://azure.microsoft.com/en-us/documentation/articles/iot-hub-sdks-summary/>)

Storage Account

If this is the first resource for the project you have created, create a new Resource Group, and add all future resources in the solution to this resource group

Go to the new Azure portal, and create a new IoT Hub

IoT hub

Microsoft

* Name

* Pricing and scale tier

S1 - Standard >

* IoT Hub units ⓘ

* Device-to-cloud partitions ⓘ

4 partitions ▾

* Resource group

+ New ▾

New resource group name

* Subscription

Azure (davidgri@microsoft.com) ▾

☐ Enable Device Management—PREVIEW ⓘ

☐ Pin to dashboard

Create

Choose your pricing and scale tier

F1 Free	S1 Standard	S2 Standard
8k messages/unit/day	400k messages/unit/day	6M messages/unit/day
<div>Device-to-cloud telemetry</div> <div>Cloud-to-device messaging</div> <div>1 unit</div>	<div>Device-to-cloud telemetry</div> <div>Cloud-to-device messaging</div> <div>200 units maximum</div>	<div>Device-to-cloud telemetry</div> <div>Cloud-to-device messaging</div> <div>200 units maximum</div>
0.00 USD PER IOT HUB UNIT	50.00 USD PER IOT HUB UNIT	500.00 USD PER IOT HUB UNIT

Select

See https://blogs.msdn.microsoft.com/david_gristwood/2016/04/12/choosing-the-right-azure-iot-hub-edition/ for more information for scaling IoT hub

Once created, you will need to return back to the "Shared access policies" tab to get and note the primary keys and connection strings to access the IoT hub from elsewhere in the solution

POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write

Azure Stream Analytics

Create the main streaming analytics job.

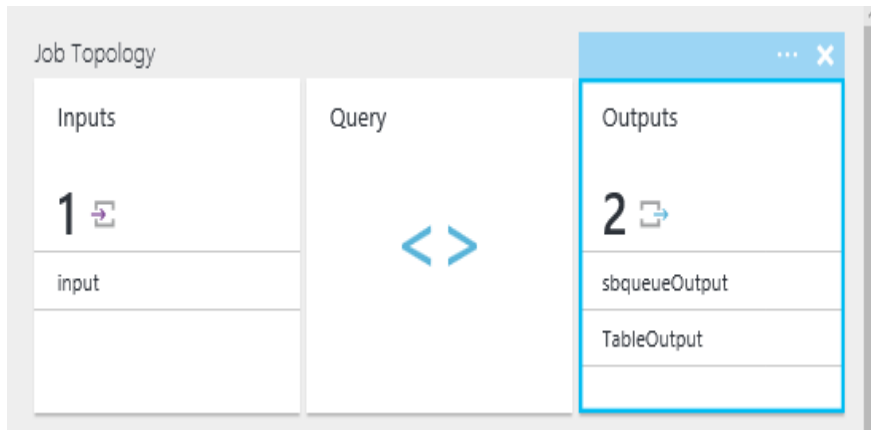
Under the Message tab, create a consumer group, which is an "end point" that you will need to hook to later, and use a name such as "streamanalytics".

Consumer groups ⓘ	
\$Default	
pbistreamanalytics	...
streamanalytics	...
<input type="text"/>	...

Create the stream analytics Query:

You will need to provide the connection information for your newly created IoT Hub:

Create the stream analytics Outputs:



NAME	SINK
sbqueueOutput	Service bus Queue
TableOutput	Table storage

TableOutput:

We output all input to table storage for long term "cold" storage queries and analysis. This is optional, but it is useful for debugging to check which data is flowing through the system, and lets you experiment with tools such as Power BI to view historic data

* Storage account ⓘ

Storage account key

* Table name ⓘ

* Partition key ⓘ

* Row key ⓘ

* Batch size ⓘ

sbqueueOutput:

We output an averaged heartrate data value to the service bus for interactive processing. The idea is to decouple any interactive processing from the core system, so these parts of the system can be evolved and updated without disrupting the message processing

This interactivity typically would only occur when the heart rate goes above a certain threshold, in which case we would add a conditional clause to the query:

HAVING (HeartRate >= 80) AND (HeartRate <= 200)

* Service bus namespace ⓘ

* Queue name ⓘ

* Queue policy name ⓘ

Queue policy key ⓘ

* Event serialization format ⓘ

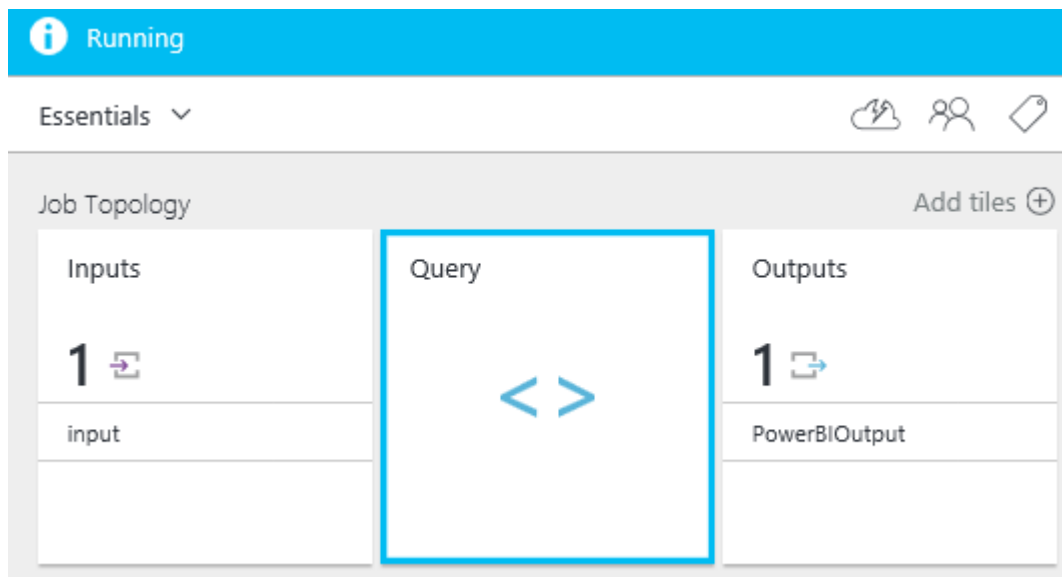
Encoding ⓘ

Format ⓘ

Create the Power BI stream analytics job (optional)

```
1 SELECT DeviceId, Cast(AVG(HeartRate) as BIGINT) as HeartRate, Max(EventEnqueuedUtcTime) as T
2 INTO PowerBIOutput
3 FROM input
4 Timestamp by EventEnqueuedUtcTime
5 GROUP BY DeviceId, Tumblingwindow(second, 10)
```

The Power BI job is optional, but it makes for a great interactive demo.



The input is the same input as before.

The output is to Power BI. See the later section on setting up Power BI, which you will need to have set up before creating PowerBIOutput.

Service Bus

A service bus is used to link the stream analytics flow and the interactive processing. We deliberately decouple the two, so they can evolve separately. We could use topic/subscription rather than a queue, if multiple listeners are needed.

First, use the old portal (at the time of writing) to create a service bus namespace (and append with "-ns")

Create namespace:

service bus

NAMESPACE NAME	STATUS	TYPE	LOCATION
----------------	--------	------	----------

bandontherunservicebus

CREATE A NAMESPACE

Add a new namespace

NAMESPACE NAME

!

.servicebus.windows.net

TYPE ?

MESSAGING

EVENTHUB

NOTIFICATION HUB

MESSAGING TIER ?

BASIC

STANDARD

PREMIUM

REGION

North Europe

▼

Then create the queue in the new namespace:

⚡ QUICK CREATE

📦

CUSTOM CREATE

QUEUE NAME

REGION

North Europe ▼

NAMESPACE

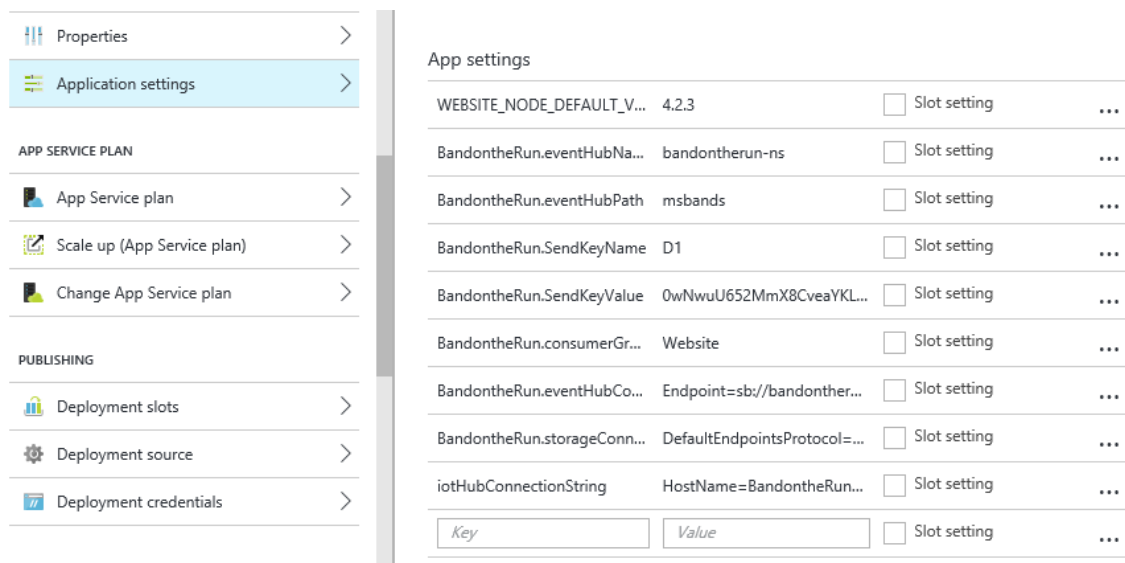
bandontherunservicebus-ns ▼
.servicebus.windows.net

Azure Tracker Site and Azure Web Site

The solution requires two web sites to be present.

You will need to deploy your own versions of both web sites – don't use the default URLs we do, set up your own.

We have a philosophy of keeping connection strings and keys out of web.config so they don't end up under source code control, and could be unintentionally exposed, and instead, place them in them in Azure web site portal:



The screenshot shows the Azure portal interface for managing application settings. The left-hand navigation pane includes sections for 'Properties', 'Application settings' (which is highlighted), 'APP SERVICE PLAN' (with options for App Service plan, Scale up, and Change App Service plan), and 'PUBLISHING' (with options for Deployment slots, Deployment source, and Deployment credentials). The main content area, titled 'App settings', contains a table of settings for the application.

Key	Value	Slot setting	
WEBSITE_NODE_DEFAULT_V...	4.2.3	<input type="checkbox"/>	...
BandontheRun.eventHubNa...	bandontherun-ns	<input type="checkbox"/>	...
BandontheRun.eventHubPath	msbands	<input type="checkbox"/>	...
BandontheRun.SendKeyName	D1	<input type="checkbox"/>	...
BandontheRun.SendKeyValue	0wNwuU652MmX8CveaYKL...	<input type="checkbox"/>	...
BandontheRun.consumerGr...	Website	<input type="checkbox"/>	...
BandontheRun.eventHubCo...	Endpoint=sb://bandonther...	<input type="checkbox"/>	...
BandontheRun.storageConn...	DefaultEndpointsProtocol=...	<input type="checkbox"/>	...
iotHubConnectionString	HostName=BandontheRun...	<input type="checkbox"/>	...
<input type="text" value="Key"/>	<input type="text" value="Value"/>	<input type="checkbox"/>	...

The first - Azure Web App - manage bands and hand out keys for use with IoT Hub. Longer term this will show more management information.

<https://bandontheruntracker.azurewebsites.net/>

Band On The Run

Band on the Run - track your Microsoft Band with Microsoft Azure.

"{" as of: 3/5/2016 11:37:4

© 2016 - A Pete and David production, Feb 2016

The main API function exposed by the site registers devices with IoT Hub

```
// IoT Hub device register code

[Route("api/IoTRegisterDevice/{deviceId}")]
[HttpGet]
0 references | davidgristwood, 98 days ago | 1 author, 3 changes | 0 exceptions
public async Task<string> IoTRegisterDevice(string deviceId)
{
    // register device with IoT Hub
    string connectionString = WebApiApplication.ehWebConsumerGroup.iotHubConnectionString;

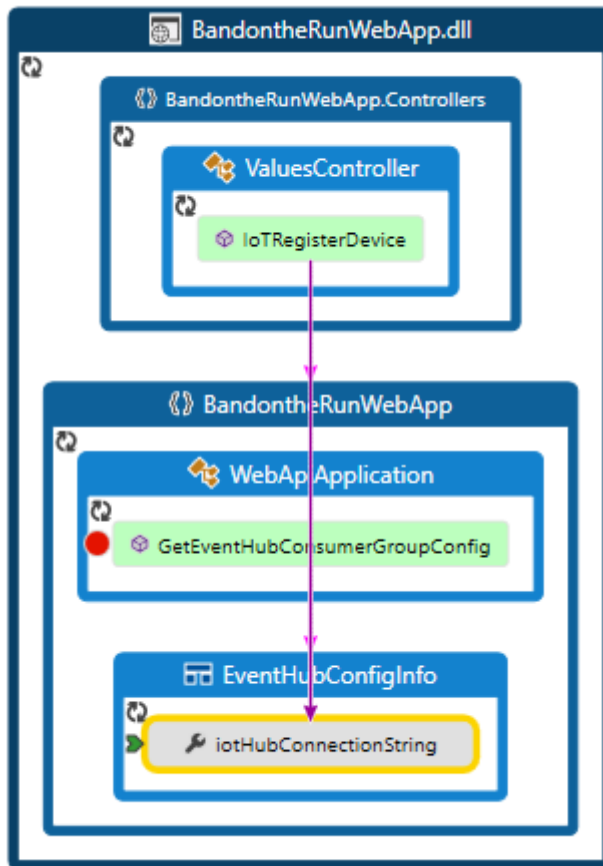
    RegistryManager registryManager = RegistryManager.CreateFromConnectionString(connectionString);
    string key = await AddDeviceAsync(registryManager, deviceId);

    // track list of devices for reporting
    _IoTRegisteredDevices[deviceId] = DateTime.Now.ToString();

    return key;
}
```

We chose this model to ensure that the IoT Hub keys are not baked into the Band on the Run app, but are issued at runtime. We don't implement any security model at the moment to validate the API calls, but it would be easy to plug in a suitable mechanism.

The IoT Hub connection string is needed to create the device key:



This is set in the portal:

```
eventHubSettings.iotHubConnectionString =  
System.Configuration.ConfigurationManager.AppSettings["iotHubConnectionString"];
```

The connection string should be set in the Azure portal and of the form:

```
HostName=MYHUB.azure-  
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=XXXXXXXXXXXXXXXXXXXX=
```

You obtain this from the Shared Access Policies in your IoT Hub:

POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write

iothubowner

Access policy name: iothubowner

Permissions:

- ☒ Registry read
- ☒ Registry write
- ☒ Service connect
- ☒ Device connect

Shared access keys:

Primary key: [Redacted]

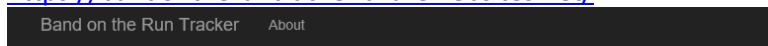
Secondary key: [Redacted]

Connection string—primary key: [Redacted]

Connection string—secondary key: [Redacted]

The second - Azure Band Tracker - visualises the heartrate information and sends it via SignalR to a browser

<https://bandontheruntracker.azurewebsites.net/>



Azure Web Job

We use tracker web app to host the web job that process messages from the service bus, but you can host it in any app service. You may find it useful to set the “Always On” to ensure the jobs are live and active, but this means using a Basic pricing tier, which has a cost implication.

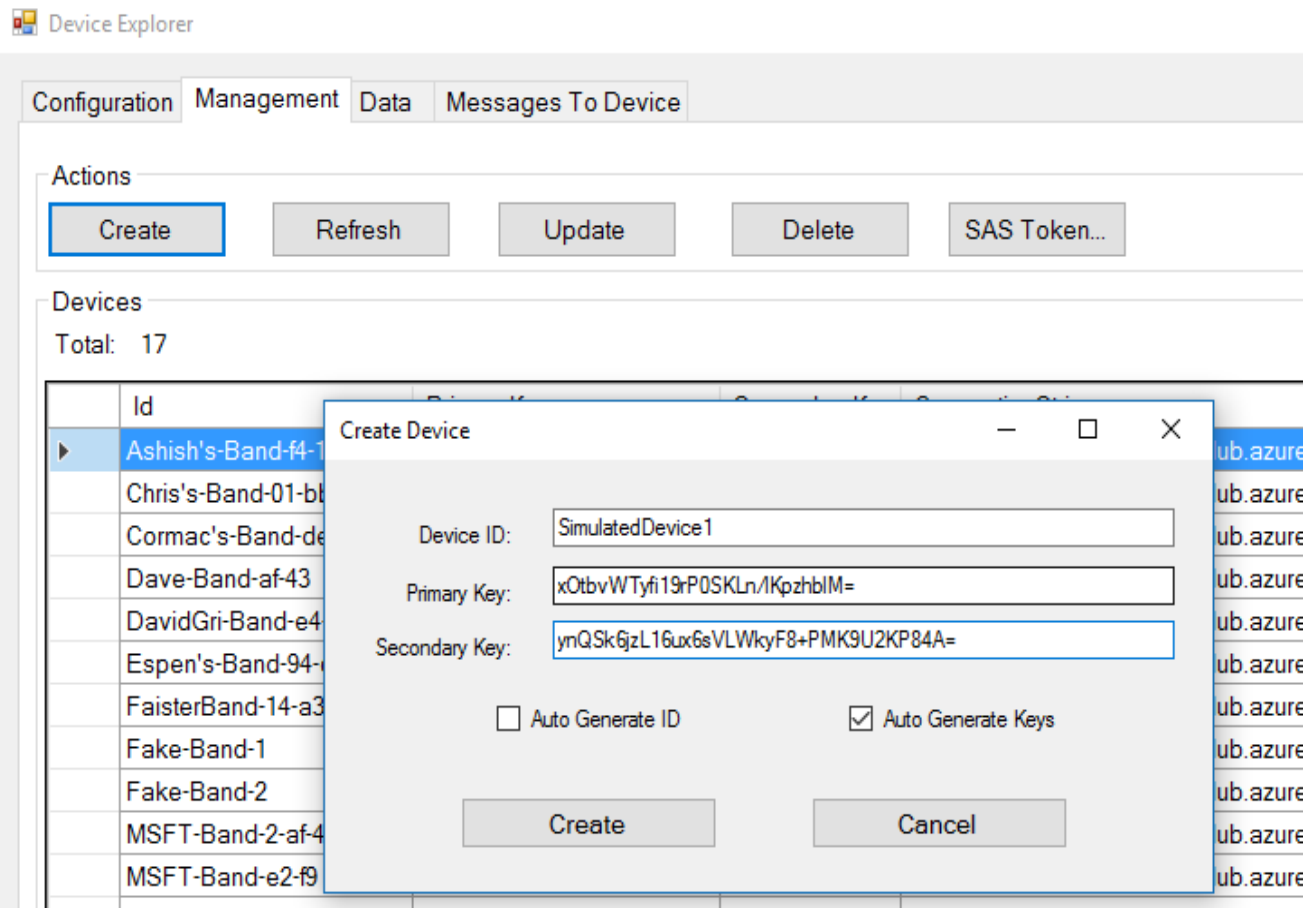
Note: we are looking to turn the web jobs in Azure Functions.

Simulated Band IoT Hub

This Visual Studio project "Simulated Band IoT Hub" is a test harness designed to simulate the data from the Band App. This can be very useful during Azure test and dev, as it pushes a small number of telemetry payload packages under controlled conditions to the IoT Hub, which you can then track and monitor to make sure the messages are flowing through the system correctly.

Ensure you set the name IoT Hub in the program, and run the program from command line, passing the device key on the command line.

The device key can be generated from Azure IoT SDK "Device Explorer". The program defaults to "SimulatedDevice1" for the device name



A successful run should produce output similar to this:

```
13/05/2016 12:42:16> sending data #0 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":71,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:16"}'
13/05/2016 12:42:18> sending data #1 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":71,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:18"}'
13/05/2016 12:42:20> sending data #2 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":69,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:20"}'
13/05/2016 12:42:22> sending data #3 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":69,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:22"}'
13/05/2016 12:42:24> sending data #4 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":69,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:24"}'
13/05/2016 12:42:26> sending data #5 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":70,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:26"}'
13/05/2016 12:42:28> sending data #6 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":69,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:28"}'
13/05/2016 12:42:30> sending data #7 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":70,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:30"}'
13/05/2016 12:42:32> sending data #8 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":70,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:32"}'
13/05/2016 12:42:34> sending data #9 for band SimulatedDevice1 = '{"DeviceId":"SimulatedDevice1","HeartRate":71,"SkinTemp":0.0,"UVIndex":0,"Timestamp":"13/05/2016 12:42:34"}'
Press <enter> to exit
```

Power BI (Optional)

(to follow)