Installation Guide - June 2016

12 May 2016

16:12

# Band on the Run Architecture

Overall Band on the Run Architecture

Machine generated alternative text:
An ics 
Web Job 

The [UWP](https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp) app connects via Bluetooth, using the [Microsoft Band SDK](http://developer.microsoftband.com/bandSDK), to the band to get real time sensor data.

When the app first connects to a band, it gets a special [Azure IoT Hub](https://azure.microsoft.com/en-us/services/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](https://azure.microsoft.com/en-us/services/stream-analytics/) job processes that telemetry data in real time, and outputs to three different destinations.

The hot path is wired to [Power BI](https://powerbi.microsoft.com/en-us/), and charts the data in a dashboard in real time.

The cold path writes the data to [Azure table storage](https://azure.microsoft.com/en-us/documentation/articles/storage-introduction/), 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](https://azure.microsoft.com/en-gb/services/service-bus/) queue.

An [Azure WebJob](https://azure.microsoft.com/en-us/documentation/articles/web-sites-create-web-jobs/) 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](http://www.asp.net/signalr) to send the live heart rate data down to bowsers, 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

There are a number of services in this solution that need storage, including web jobs and stream analytics, so create a storage account, and ensure that the data center region you select will be used throughout the project.

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

## IoT Hub

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

Article on managing IoT hubs through the Azure portal - <https://azure.microsoft.com/en-gb/documentation/articles/iot-hub-manage-through-portal/>

Machine generated alternative text:
IOT hub 
Name 
Name your ,'wb 
* Pricing and scale tier 
Sl - Standard 
* IOT Hub units O 
* partitions O 
4 partitions 
* Resource group 
New resource group name 
* Subscription 
Azure (davidgri@microsoffcom) 
Enable Device Management—PREVIEW O 
Pin to dashboard 
Create 
Choose your pricing and scale tier 
Fl Free 
messages/unit/day 
Device-to-cloud 
Cloud-to-device 
I unit 
0.00 
LED PER IOT HUB UNIT 
Select 
Sl Standard 
400k 
messages/unit/day 
Device-to-cloud 
Cloud-to-device 
200 units maximum 
50.00 
LED PER IOT HUB UNIT 
S2 Standard 
messages/unit/day 
Device-to-cloud 
Cloud-to-device 
200 units maximum 
500.00 
LED PER IOT HUB UNIT 

F1 free is good start for testing.

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

Machine generated alternative text:
iothu bowr,er 
service 
device 
registryRead 
reg ist ryReadWrite 
registry write, service connect, device connect 
service connect 
device connect 
registry read 
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".

Machine generated alternative text:
Consumer groups O 
SDefauIt 
p bistreamanalytics 
streamanalytics 

Create the stream analytics Query:

Machine generated alternative text:
Running 
Essentials 
Jab Topology 
Inputs 
1 
input 
Query 
Add tiles 
Outputs 
2 
sOqueueOutput 
TableOutput 

Enter the query:

Machine generated alternative text:
I 
3 
4 
7 
9 
12 
/ * log to table for debug and post event * / 
SELECT Deviceld, Heartrate, EventEnqueuedUtcTime as Time 
INTO TableOutput 
FROM input 
Timestamp by EventEnqueuedUtcTime 
/ * sustained high heart rate gets alert */ 
8 SELECT Deviceld, as 8161NT) 
as HeartRate 
max (Event Enq ueuedUtcT ime) 
as Time 
INTO sbqueueOutput 
FROM input 
Timestamp by EventEnqueuedUtcTime 
GROUP BY Deviceld, 
T umbIingWindow(second, 

Create the stream analytic Input:

Machine generated alternative text:
Inputs 
1 
input 
Query 
Add tiles 
Outputs 
input 
2 
sOqueueOutput 
TableOutput 
SOURCE TYPE 
Stream 
IOT hub 

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

Create the stream analytics Outputs:

Machine generated alternative text:
Jab Topology 
Inputs 
1 
input 
Query 
Outputs 
2 
sOqueueOutput 
TableOutput 
sbqueueOutput 
TableOutput 
Service bus Queue 
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

Machine generated alternative text:
Storage account O 
bandontherunstoragenu 
Storage account key 
* Table name O 
BandOnTheRun 
* Partition key O 
Deviceld 
* Row key O 
Time 
* Batch size O 

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)

Machine generated alternative text:
Service bus namespace O 
bando ntherunservicebus-ns 
Queue name O 
bandontherunqueue 
* Queue policy name O 
RootManageSharedAccessKey 
Queue policy key O 
* Event serialization formate 
'SON 
Encoding O 
UTF-8 
Format O 
Line separated 

Create the Power BI stream analytics job (optional)

Machine generated alternative text:
as 8161NT) 
2 
3 
4 
SELECT Deviceld, 
INTO Powera10utput 
FROM input 
Timestamp by EventEnqueuedUtcTime 
as HeartRate 
max (Event Enq ueuedUtcT ime) 
as Time 
GROUP BY Deviceld, 
Tumbling-window (second, 
le) 

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

Machine generated alternative text:
Running 
Essentials 
Jab Topology 
Inputs 
1 
input 
Query 
Add tiles 
Outputs 
1 
Power310utput 

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:

Machine generated alternative text:
service 
NAMESPACE 
bus 
CREATE A NAMESPACE 
Add a new namespace 
NAMESPACE NAME 
TYPE O 
MESSAGING 
EVENTHUB 
o 
.servicebus.windows.net 
NOTIFICATION HUB 
o 
MESSAGING TIER 
BASIC 
STANDARD 
REGION 
North Euro e 
PREMIUM 

Then create the queue in the new namespace:

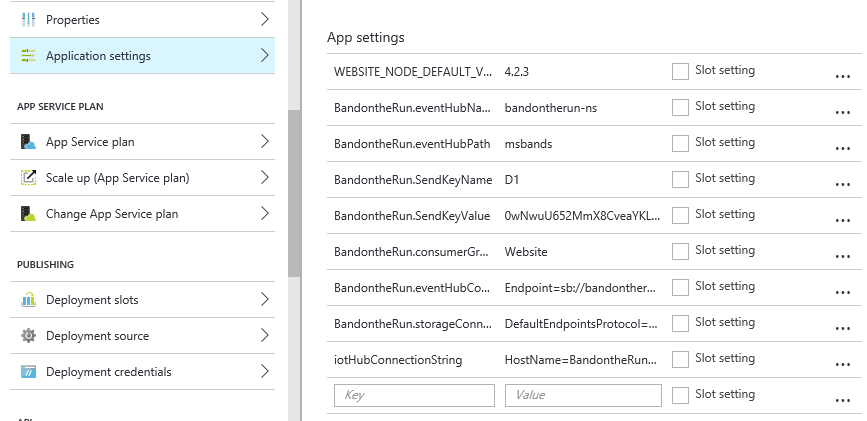
Machine generated alternative text:
QUEUE NAME 
QUICK CREATE 
CUSTOM CREATE 
REGION 
North Europe 
NAMESPACE 
bandontherunservicebus-ns 
.servicebus.windows.net 

## Azure Tracker Site and Azure Web Site

The solution requires two web sites to be present.

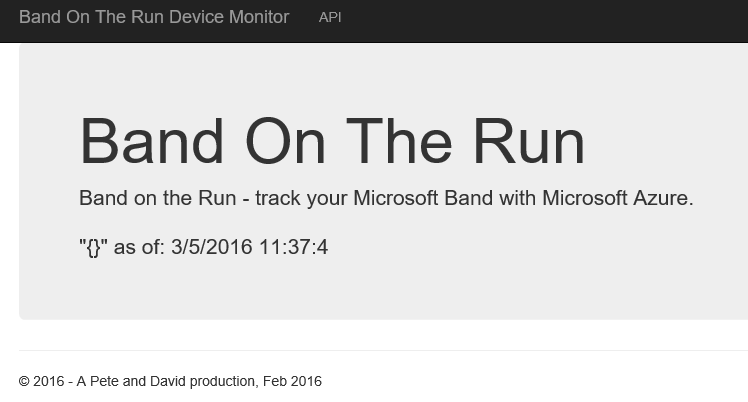
You will need to deploy your own versions of both web sites – don’t use the default URLs we do, set up your own.

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

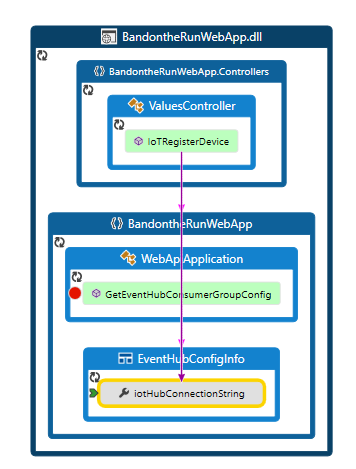
<https://bandontheruntracker.azurewebsites.net/>



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



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.

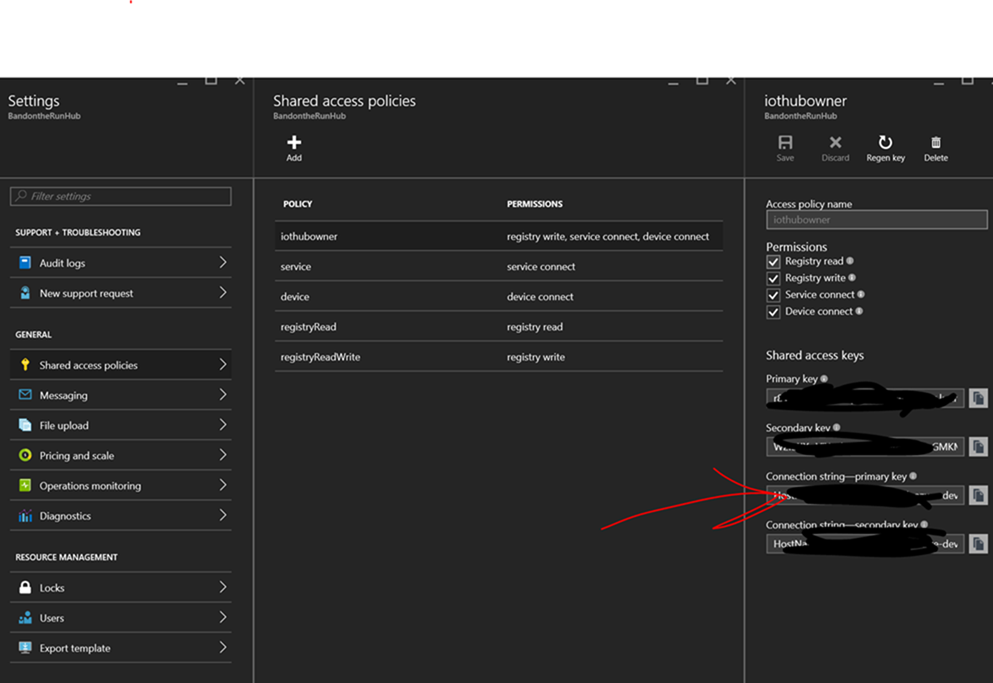
The IoT Hub connection string is needed to create the device key:

This is set in the portal:

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

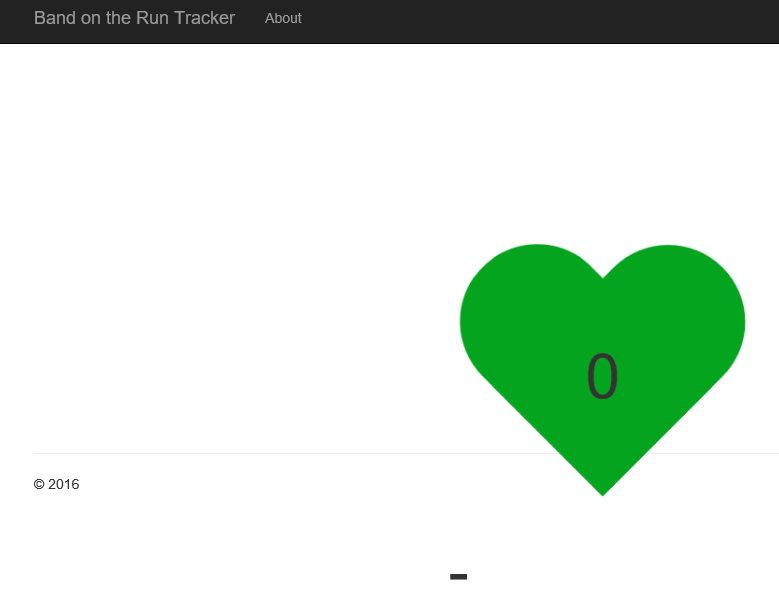
The connection string should be set in the Azure portal and of the form:

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

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

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

Machine generated alternative text:
Device Explorer 
Configuration Management Data 
Actions 
Create 
Devices 
Total: 17 
Id 
Refresh 
Messages To Device 
Update 
Delete 
Ashish's-aand-f4-- 
Chris's-8and-01-b 
Cormac's-aand- 
Dave-Band-af-43 
DavidGri-8and- 
Espen's-8and- 
Faister8and-14-a 
Fake-sand-I 
Fake-sand-2 
MSFT-Band-2-a 
MSFT-Band-e2-f9 
Create Device 
Device ID 
Primary Key 
Secondary Key 
Auto Generate ID 
Create 
SAS Tokem„ 
Auto Generate Keys 
Cancel 
Libezur€ 
bezure 
b azure 
b azure 
b azure 
b azure 
b azure 
b azure 
b azure 
b azure 
b azure 

A successful run should produce output similar to this:

Machine generated alternative text:
13/85/2816 
:8.8, "UVIndex" : 8, "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
42 : 38 
13/85/2816 
:ø.ø, "UVIndex" "Timestamp 
42 : 
42: 32 
13/85/2816 
:8.8, "UVIndex" : 8, "Timestamp 
42 : 
sending 
sending 
sending 
sending 
sending 
sending 
sending 
sending 
sending 
sending 
data 
data 
data 
data 
data 
data 
data 
data 
data 
data 
#8 
#9 
for 
for 
band 
band 
band 
band 
band 
band 
band 
band 
band 
band 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
SimulatedDevice1 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 71, "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 71 , "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 69 , "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 69 , "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 69 , "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 78, "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 69 , "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 78, "SkinTemp" 
{"Deviceld" : "SimulatedDevice1" , "HeartRate" : 78, "SkinTemp" 
"Deviceld " : "SimulatedDevice1" , "HeartRate" : 71, "SkinTemp" 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
" : "13/85/2816 
42: 18"} 
42 : 28"} 
42: 22"} 
42: 26"} 
42: 28"} 
42: 34"} 
Press 
to 
exit 

## Power BI (Optional)

(to follow)