**Using Power BI Embedded and Machine Learning for displaying Sound Data Monitoring within a Smart Stadium**

**Table of Contents**

[Executive Summary 2](#_Toc482360775)

[Noise Pollution 3](#_Toc482360776)

[Solution, steps and delivery 4](#_Toc482360777)

[Step 1: Solution envisioning 4](#_Toc482360778)

[Step 2: Create the IoT Hub 5](#_Toc482360779)

[Step 3: Create the Stream Analytics Job 6](#_Toc482360780)

[Step 4: Create the SQL Database 7](#_Toc482360781)

[Step 5: Create the Machine Learning Model and Service 8](#_Toc482360782)

[Step 6: Create the Web API RESTful Service 9](#_Toc482360783)

[Step 7: Create the Web Dashboard 12](#_Toc482360784)

[Step 8: Create the Power BI Workspace collection and Provision 13](#_Toc482360785)

[Engagement Approach and Team 17](#_Toc482360786)

**Table of Figures**

[Figure 1 - Solution Architecture 4](#_Toc482360787)

[Figure 2 - Azure IoT Hub 5](#_Toc482360788)

[Figure 3 - SQL Database Tables 7](#_Toc482360789)

[Figure 4 - Machine Learning Services 8](#_Toc482360790)

[Figure 5 - Web API Application Architecture 9](#_Toc482360791)

[Figure 6 - Web Dashboard 12](#_Toc482360792)

[Figure 7 - Power BI Access Keys 13](#_Toc482360793)

[Figure 8 - Power BI Workspace 14](#_Toc482360794)

[Figure 9 - Power BI Report for Sound 16](#_Toc482360795)

## Executive Summary

The Gaelic Athletic Association (GAA) is Ireland’s largest sporting organisation with 2,500 clubs in Ireland and 500,000 members. It is celebrated as one of the great amateur sporting associations in the world. It is part of the Irish consciousness and plays an influential role in Irish society that extends far beyond the basic aim of promoting Gaelic games. Every summer the inter-county All-Ireland Championships in hurling and football capture the attention of the Irish public, and regional towns heave with the arrival of large numbers of supporters and the colour, noise and excitement that they bring. In the region of 1.5 million people attend the GAA Championships from May to September in the National Stadium Croke Park.

However, by far the two biggest days in the GAA calendar are the All-Ireland finals in hurling and football. A sell out attendance of 82,300 is guaranteed in Croke Park. The finals are broadcast around the world.

The Croke Park Stadium plays host to iconic moments in Irish sport & history and to major cultural and international event as well as home to one of the world’s leading internet of things (IoT) programmes, with Croke Park the test bed for everything from pitch management and crowd control to micro-weather wind circulation. The information coming from this project is support other smart stadium around the world like Arizona, ASU is redeveloping its Sun Devils Stadium leveraging the learnings from this smart stadium project.

The Croke Park Power BI Reporting project is a collaboration between GAA, Dublin City University (DCU) and Microsoft to leverage the innovations that were achieved by the GAA Smart Stadium project and present the data in a meaningful manner to the end user. One of the issues around the initial solution was in that the Sound Data that was coming into the Azure Platform included both internal and external sound data. Within this project, a Machine Learning model was developed that stripped out the noise pollution and presented more accurate data in the form of Power BI embedded reports. These reports were embedded within an Azure App Service and accessed it’s data via a Web API and a Machine Learning API.

**Authors:**

Mike Myers, Principal Technical Evangelist, Microsoft Ireland

Suzanne Little, Dublin City University

## Noise Pollution

 An important step in building strong community relations is ensuring that Croke Park is a good neighbour in that its events must have a minimal impact on those who live nearby. That starts with environmental sound monitoring. When it comes to decibels, Croke Park must stay within the parameters established by Dublin City Council. Before the Smart Stadium initiative, an independent third party would record the noise levels and let the Stadium know after the fact whether it was in compliance. Now this monitoring is done in real time.

An automated solution to this problem solves a number of problems for the GAA:

1. Reduced overhead in sound monitoring: The current solution is very manual and requires significant effort throughout a concert to record results.
2. Sound data can be disseminated through multiple channels, e.g. a website, a publicly accessible app or a dashboard accessible by key personnel.
3. Remove the noise pollution from external forces that could be mixed in with the internal sounds.

The Smart Stadium Project is an ongoing collaborative project between GAA, Dublin City University (DCU) and Microsoft. This document outlines the next phase within the Smart Stadium project by leveraging Power BI Embedded and Machine Learning for displaying Sound Data Monitoring within a Smart Stadium. After completion of the initial project an opportunity was discovered to build a new solution leveraging sound data that was coming into the Azure Platform included both internal and external sound data. Within this new project, a Machine Learning model will be developed that will remove out the noise pollution and presented an extended and more accurate data visualization in the form of Power BI embedded reports. These reports will be embedded within an Azure App Service and will access its data via a Web API and a Machine Learning API.

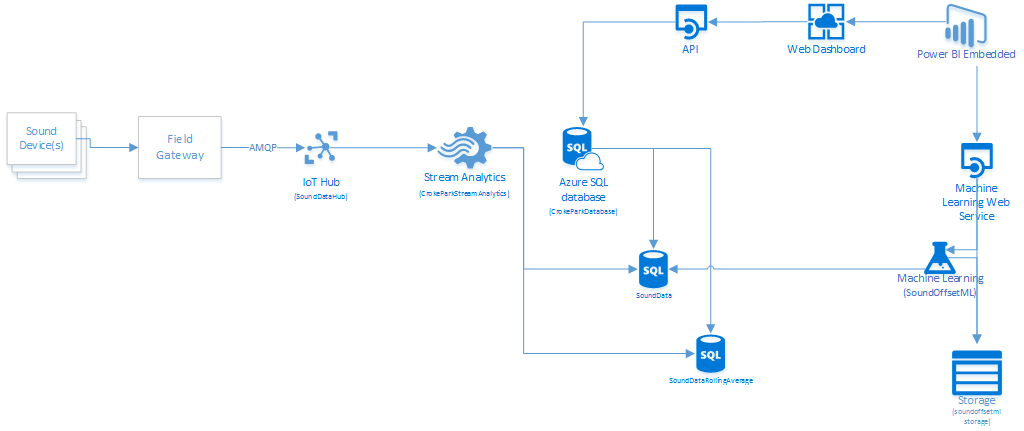
Another key requirement for this solution was the importance of strategically positioning the microphones within the stadium. To understand noise levels for both crowd cheer and pollution level microphones must be located both within the stadium bowl and externally. For this reason, 4 microphones were deployed, 2 on the East side of the stadium and two on the west side. Each side had one microphone inside the stadium and one outside.

## Solution, steps and delivery

### Step 1: Solution envisioning

The web application (Dashboard) and its associated services all run within the Azure platform as an App Service. The App Service connects to its data source via an REST API, which in turn connects to a SQL Database that using several tables to hold the received data that came from a Stream Analytics job.

The solution is initially composed of several sound recording devices that sends sound data to a Field Gateway that in turn pushes an aggregated message up to an Azure IoT Hub via the AMQP protocol. A Stream Analytics job then performs a number of queries on this data and pushes the data into the required table – either a rolling average table or just standard data that is envisioned using a Azure SQL Database. This is mainly used for the reporting needs of the solution and is consumed using several Power BI Reports that are embedded within a Web App running as an Azure App Services.



**Figure 1 - Solution Architecture**

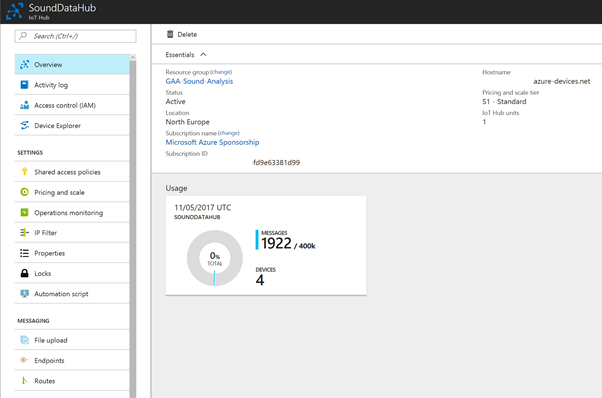
The Power BI Embedded services uses this database as its data source. The following steps were identified for the technical implementation.

* Ingest sound data in its raw format
* Store the data in SQL DB Tables and include a rolling table (accumulated figures)
* Create a Machine Learning Model to stripe out the external noises from the data resulting in just the internal noise data.
* Create a REST API to provide GET operations in the Power BI Embedded workspace collection to retrieve the data for consumption
* Set up the use of **DirectQuery** with Azure SQL Database so that the data can be displayed by using the Power BI Embedded REST API functionality
* Create .pbix reports using Power BI Desktop.
* Upload .pbix reports to the server
* Create the Web Dashboard.
* Include the reports in the Web App for consumption.

### Step 2: Create the IoT Hub

The initial step involves creating an Azure IoT Hub. We went with a S1 Standard pricing tier as it is expected that the number of message will grow to over 8000 per day and we could have several Gateways (Devices) that would be sending telemetry data to the IoT hub. It was also agreed that 1 Unit would suffice for this scenario.

While this scenario is solely about pushing sound telemetry data into the Azure Platform, it could be argued that we could have used an Event Hub instead. We went with an IoT hub as we wanted to use the additional features that this brings including Device Identity and Management and the ability to do Cloud to Device messaging as this will be using in future version of the solution.



**Figure 2 - Azure IoT Hub**

### Step 3: Create the Stream Analytics Job

Once data has been ingested into the IoT Hub, Stream Analytics is used to analyse the data and create a stream of data for each use case. Each stream of data is then outputted to a table in the SQL database. There are two stream analytics jobs, one to handle weather data and another to handle sound. Each job has the IoT Hub as input and defines a number of outputs. The table below outlines each stream analytics query, their inputs, queries and outputs:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stream Analytics Job | Input | Outputs | Queries | Description |
| CrokeParkStreamAnalytics | IoTHub | sql | SELECT  d.Gateway. audio data. LAF MAX as L Amax,  d.Gateway. audio data.LAge as LAg0,  d. Gateway. audio data. LAeq as LAEQ,  d.Gateway. audio_data. LAIe as LAIØ,  System. Timestamp as Time,  d. Gateway. audio_data . deviceid as Deviceld,  ConnectionDeviceId as 10THubDeviceId  INTO  [sql]  ClotHub]  TIMESTAMP by EventEnqueuedUtcTime  where d. Gateway. audio_data . LAF MAX  e | Sends all sound data points to a sql database table. The LAMax values are used to evaluate intensity of crowd cheer. |
|  | IoTHub | sqlrolling | / * send averaged L EQ values over sliding window of 15 minutes  / * need to do for avg lamax  SELECT  Avg(d.Gateway.audio data.LAeq) as AVGLEQ,  avg( Power( 18, d. Gateway. audio_data. LAeq/1e  MAX(d.Gateway.audio data. LAF MAX) as MAXLARAX,  d . Gateway. audio data. deviceid as Deviceld,  as AVGPressureSquared,  System. Timestamp as Universal Time,  max(EventEnqueuedUtcTime) as Time  INTO  sqlrolling  FROM  IotHub  TIMESTAMP BY EventEnqueuedUtcTime  where d.Gateway.audio data. LAeq  gm)up by SlidingWindow(minute, 15  0  d. Gateway. audio_data. deviceid | Retrieve a rolling average of LAEQ values to determine a normalised view of sound data for noise pollution. The average is calculated as a logarithmic average. |
|  | IoTHub | soundblob | select  into soundblob  from IotHub  TIMESTAMP by EventEnqueuedUtcTime  where d. gateway. audio_data. timestamp | All data points are sent to blob storage for diagnosis. |
| CrokeParkWeatherStreamAnalytics | IoTHub | Sql | WITH WeatherData AS  select  from iothub  TIMESTAMP by EventEnqueuedUtcTime  where d. gateway. weather data. "Wind Speed kph"  select  System. Timestamp as Time,  'weatherstation' as Deviceld,  ConnectionDeviceId as 10THubDeviceId,  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  d. gateway. weather  into sql  from WeatherData  data . "Barometric Trend" as BarometricTrend,  data . "Pressure - hPa" as Pressure  data. "Weather Forecast" as WeatherForecast,  data  "Stadium Temperature C" as StadiumTemperature,  data. "Skywalk Temperature C" as SkywalkTemperature,  data. "Wind Speed kph" as WindSpeedkph,  data . "Wind lemin Average kph" as WindAveragekph,  data. "Wind Direction" as WindDirection,  data . "Humidity  as Humidity  data  "Current Rain Rate" as CurrentRainRate,  data. "Daily Rain Rate" as DailyRainRate,  data  "Sunrise"  data . Sunset,  data. "Battery Status" as BatteryStatus  where d. gateway. weather data. "Wind Speed kph" | Raw weather data is sent to a sql table |

### Step 4: Create the SQL Database

An Azure SQL database is used to store relevant data for analysis or display. The SQL Database is a fully managed relational database service hosted on Azure and offers 99.99% SLA for availability which helps this solution scale to a fully production system. The database also supports 100 database transaction units but can scale to 4000 units when required. The DTU is a blended measure of performance that can be used to get predictable performance.

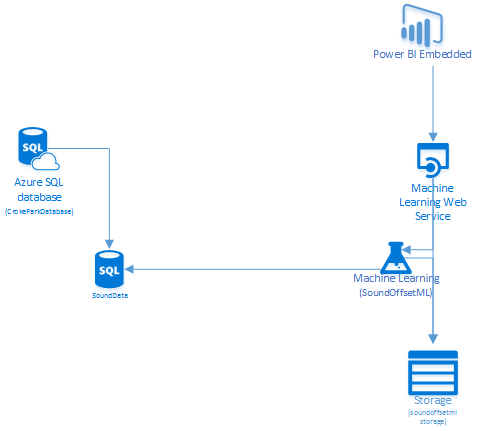
The database tables are populated by the stream analytics jobs described above.

5 = コ N MO = 4 
ad 、 12 帑 0 
•W 一 帑 帑 p 
•W 一 帑 帑 p 
2 」 n 帑 」 adLualLun 一 
aweN ULUnIOD 
2 」 n 帑 」 2dE21 2 」 S 
工 d 262 」 2 》 vpu 一 
pua 」 1 」 帑 WO 」 28 
2U2 」 0 亠 」 2 工 帑 
P -2U520qn 工 10 - 
」 !OPL'!M 
一 ヒ 25 」 2 ~ un 
2 帑 u 一 2 、 T20 
sn 帑 、 」 2 た 28 
2E2N2 製 》 20 
LIO こ d ご H20 
言 一 Lun 工 
P - 写 20 
2 」 n552 」 d 
Esuns 
pue4S 
一 1 
5 = コ N MO = 4 
SIInN MO = 4 
ロ ロ 国 ロ 国 国 国 国 国 国 国 国 ロ 
ad 、 12 帑 0 
ad 、 1 
oqp ・ u 三 0 eopunos 
お L) 」 2 モ u 
•W 一 帑 帑 p 
•W 一 帑 帑 p 
•W 一 帑 帑 p 
•W 一 帑 帑 p 
aweN ULUnIOD 
pa 」 enbsa 」 n552 」 d9 > 4 
aweN ULUnIOD 
oqp Qeopunos 
P - 写 aoqn 工 10 - 
一 ヒ 25 」 2 ~ un 
一 ヒ 25 」 2 ~ un 
2E2N2 》 20 
2E2N2 製 》 20 
XYLAV 受 VLA 
LIO こ d ご U520 
P - 写 20 
P - 2 》 20 
0 山 コ 9 > 4 
pue4S 
9 鶩 
一 1 
一 1 

**Figure 3 - SQL Database Tables**

### Step 5: Create the Machine Learning Model and Service

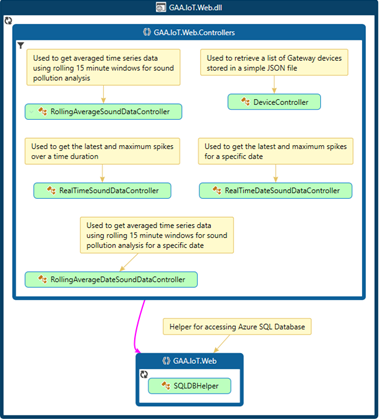
The team created an Azure Machine learning model around help them removing the unwanted noise pollution from the code datasets. This was achieved by receiving data from several IoT devices (Gateways) with some been internal to the Stadium and some been external. The Model got its datasets from the SQL Database as two streams and then the removal of the external forces occurred. The resulting data was then stored in a storage account that was then consumed by a Power BI report. This can be seen here:



**Figure 4 - Machine Learning Services**

### Step 6: Create the Web API RESTful Service

The team decided to create an API to allow access to the stored data within the SQL Database. This would be in the form of a RESTful service and was developed as an ASP.NET Web API using Visual Studio 2015. This contains several controllers that allow access to the data that will be consumed by the front-end dashboards of the solution.



**Figure 5 - Web API Application Architecture**

#### Controllers

public class RealTimeSoundDataController : ApiController

{

[SwaggerOperation("GetLatestSoundData", OperationId = "GetLatestSoundData")]

public IEnumerable<DeviceRealTimeSoundDataModel> Get( int minutes )

{

return MvcApplication.SqlDBHelper.GetLatestRealtimeSoundData(minutes );

}

}

public class RealTimeDateSoundDataController : ApiController

{

[SwaggerOperation("GetSoundDataForDate", OperationId = "GetSoundDataForDate")]

public IEnumerable<DeviceRealTimeSoundDataModel> Get(DateTime date)

{

return MvcApplication.SqlDBHelper.GetSoundDataByDate( date );

}

}

public class RollingAverageSoundDataController : ApiController

{

[SwaggerOperation("GetRollingAverageSoundData")]

public IEnumerable<DeviceRollingAverageSoundDataModel> Get(int minutes)

{

return MvcApplication.SqlDBHelper.GetRollingAverages(minutes);

}

}

public class RollingAverageDateSoundDataController : ApiController

{

[SwaggerOperation("GetRollingAverageSoundDataByDate")]

public IEnumerable<DeviceRollingAverageSoundDataModel> Get( DateTime date )

{

return MvcApplication.SqlDBHelper.GetRollingAveragesByDate( date );

}

}

#### SQLDBHelper

public class SQLDBHelper

{

public SQLDBHelper()

{

}

/// <summary>

/// Returns the latest information for each device over a certain time period in minutes

/// This method looks at Realtime LAMax figures and gets the largest LAMax figure for the time period

/// as well as the most recent LAMax data. Max(LAMax) should be used to monitor crowd cheer etc

/// </summary>

/// <param name="minutes"></param>

/// <returns></returns>

public List<DeviceRealTimeSoundDataModel> GetLatestRealtimeSoundData(int minutes )

{

var dbContext = new AzureSQLDBDataContext();

var latestTime = DateTime.Now.ToUniversalTime().AddMinutes(-minutes);

var data = from d in dbContext.SoundDatas

where d.Time >= latestTime && d.LAMax != null && d.LAMax > 0

group d by d.DeviceId into g

select new DeviceRealTimeSoundDataModel

{

DeviceId = g.Key,

MaximumSoundData = (from x in g orderby x.LAMax descending select x).FirstOrDefault(),

LatestSoundData = (from y in g orderby y.Time descending select y).FirstOrDefault()

};

var list = data.ToList();

return list.OrderBy(x => x.Device.SortIndex).ToList();

}

public List<DeviceRealTimeSoundDataModel> GetSoundDataByDate( DateTime date )

{

var dbContext = new AzureSQLDBDataContext();

var data = from d in dbContext.SoundDatas

where d.Time.Date.Equals(date.Date) && d.LAMax > 0

group d by d.DeviceId into g

select new DeviceRealTimeSoundDataModel

{

DeviceId = g.Key,

MaximumSoundData = (from x in g orderby x.LAMax descending select x).FirstOrDefault(),

MinimumSoundData = (from y in g orderby y.LAMax ascending select y).FirstOrDefault()

};

return data.ToList().OrderBy(x => x.Device.SortIndex).ToList();

}

/// <summary>

/// Returns the data from a sliding window of 15 minutes for the time window specified in minutes

/// </summary>

/// <param name="minutes"></param>

/// <returns></returns>

public List<DeviceRollingAverageSoundDataModel> GetRollingAverages(int minutes)

{

var dbContext = new AzureSQLDBDataContext();

var devices = DeviceFactory.Instance.Devices;

var latestTime = DateTime.Now.ToUniversalTime().AddMinutes(-minutes);

var data = from d in dbContext.SoundDataRollingAverageModels

where d.Time >= latestTime && d.AVGLEQ != null && d.AVGLEQ > 0

group d by d.DeviceId into g

select new DeviceRollingAverageSoundDataModel

{

DeviceId = g.Key,

SoundData = g.OrderBy( l=>l.Time).ToList()

};

// order data by device sort order

return data.ToList().OrderBy( x=>x.Device.SortIndex).ToList();

}

/// <summary>

/// Returns the data from a sliding window of 15 minutes for the time window specified in minutes

/// </summary>

/// <param name="minutes"></param>

/// <returns></returns>

public List<DeviceRollingAverageSoundDataModel> GetRollingAveragesByDate( DateTime date )

{

var dbContext = new AzureSQLDBDataContext();

var devices = DeviceFactory.Instance.Devices;

var data = from d in dbContext.SoundDataRollingAverageModels

where d.Time.Value.Date.Equals(date.Date) && d.AVGLEQ != null && d.AVGLEQ > 0

group d by d.DeviceId into g

select new DeviceRollingAverageSoundDataModel

{

DeviceId = g.Key,

SoundData = g.OrderBy(l => l.Time).ToList()

};

// order data by device sort order

return data.ToList().OrderBy(x => x.Device.SortIndex).ToList();

}

}

### Step 7: Create the Web Dashboard

A simple web dashboard was created to present the data online. The dashboard was built using the MVC framework within Visual Studio 2015 and used Power BI embedded to embed the Power BI dashboards. The Power BI dashboards were built using Power BI Desktop and then uploaded to Microsoft Azure to make them available to the web dashboard.

Two Power BI reports were created and uploaded to an Azure Power BI Workspace. Both reports connect directly to the Azure SQL Database described earlier and uses several views to present the following information:

* 15-minute rolling average sound data and maximum spikes for the last 20 minutes, 60 minutes, 2 hours, 1 day and any specific date to review historical data. This data is represented in a single Power BI data set as tabs, see diagram below.
* Most recent wind speed data, average wind speed data for the current day and a time series graph of wind speed for the current day. This graph helps indicate the likelihood of a sky line tour taking place.

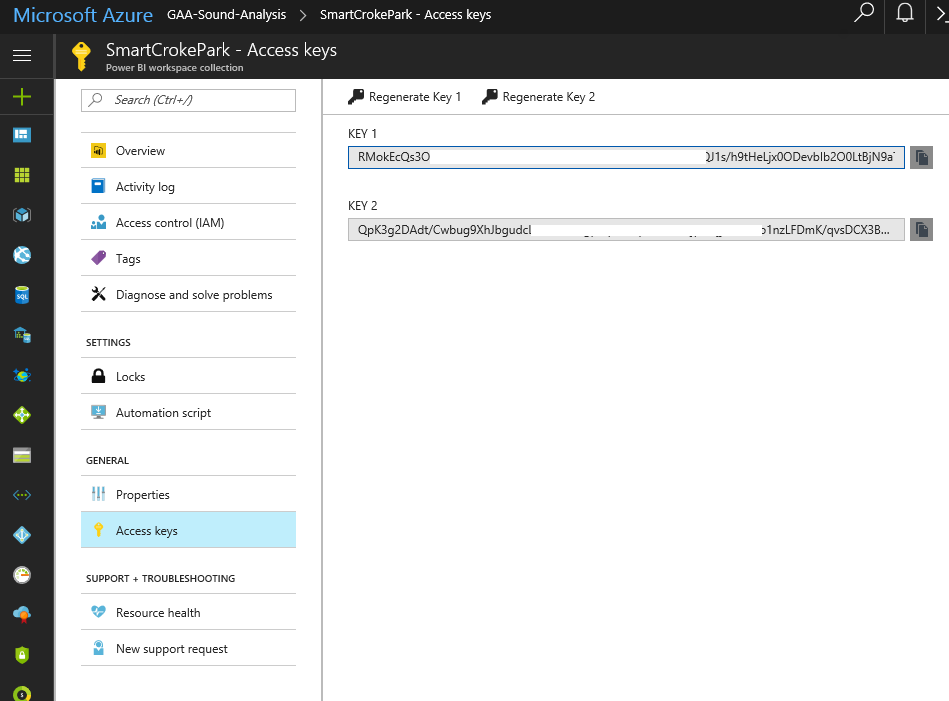
Diagrams below show the dashboard interfaces:

Smart Croke Park 
smartcrokeparkazurewebsites 'Report? reportld=577cfeee -d 7fc-4d36-ba2e- effc80f8ba93 
Reports > Wind Speed Monitoring > Windspeed 
Showing Data for: 28/09/2016 19:59:44 
Latest Wind Speed (KPH) 
Wind Speed (KPH) by Time 
Average Windspeed 
13:00 
Winds peed 
@ 201 6 Copyright. 
Report Data 
Refresh Data 
Page Navigation 
Previous Page 
Next Page 
Settings 
Z] Show Navigation 
D Show Filter Pane 
15:03 
17:03 
2.1 t 

**Figure 6 - Web Dashboard**

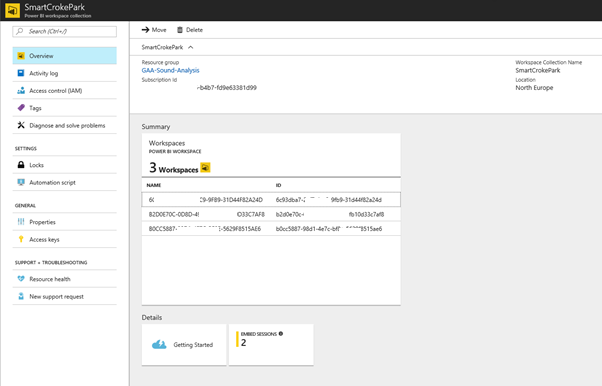
### Step 8: Create the Power BI Workspace collection and Provision

This step involved creating a Power BI workspace within the Azure Portal. Each workspace of Power BI Embedded is the workspace for each customer (tenant), and we can add many workspaces in each workspace collection. The same access key is used in each workspace collection. In-effect, the workspace collection is the security boundary for Power BI Embedded. We get out Access Key under the general area as can be seen here:



**Figure 7 - Power BI Access Keys**

Now that the provisioning is all done and we have our Access Key, we now need to create a workspace in the workspace collection via REST APIs. As you can see in our figure we have 3 workspaces in our workspace collection



**Figure 8 - Power BI Workspace**

The following HTTP POST Request (REST) was used to create a new workspace in our existing workspace collection. This is the POST Workspace API. We just set the access key, which we previously copied, as AppKey. It’s very simple authentication!

****HTTP Request****

POST https://api.powerbi.com/v1.0/collections/SmartCrokePark/workspaces

Authorization: AppKey dgdu2277db...

****HTTP Response****

HTTP/1.1 201 Created

Content-Type: application/json; odata.metadata=minimal; odata.streaming=true

Location: https://wabi-us-east2-redirect.analysis.windows.net/v1.0/collections/mypbiapp/workspaces

RequestId: 6665385-2yy3-406b-8501-4ehdfgh5f6da

{

"@odata.context": "http://wabi-eu-north-redirect.analysis.windows.net/v1.0/collections/mypbiapp/$metadata#workspaces/$entity",

"workspaceId": "55560a09-66566-4658-a8bb-9hfghghbb9d",

"workspaceCollectionName": "SmartCrokePark"

}

The returned **workspaceId** is then used for all subsequent API calls. Our solution will retain this value.

The next step in this process is to create the data connection and the reports that will be embedded within out app service. For this task, there’s no programming or code. We just use Power BI Desktop.

With Power BI Desktop, we connect to the data sources that we need within the SQL Database and then shape the data using a model to create the report, which will be consumed by the front-end dashboard. Saving the reports as a .pbix file we then upload them onto the Power BI Service. Using the Power BI Desktop service helped us in streamlining the process of designing and creating the repositories and reports.

Now we need to import our previously saved .pbix reports into the Power BI workspace. Each report in a workspace corresponds to a single Power BI Desktop file with a dataset (including datasource settings). We import our .pbix file to the workspace using the Power BI API using our previous saved workspaceid. An example of this is seen here:

POST https://api.powerbi.com/v1.0/collections/mypbiapp/workspaces/32960a09-6366-4208-a8bb-9e0678cdbb9d/imports?datasetDisplayName=mydataset01

Authorization: AppKey dgdu2277db...

Content-Type: multipart/form-data; boundary="ABCtest"

-- ABCtest

Content-Disposition: form-data

{the content (binary) of .pbix file}

-- ABCtest --

This import task take a while to run and when complete returns an import id in the form of a GUID.

Embedding the reports into the dashboard we need to secure the authorization header value as we do not want to be using our access key for security reasons; as such, we use an OAuth Json Web Token which consists of the claims and the computed digital signature.

For embedding our report, we must get the embed URL and report **id** using the following REST API.

****HTTP Request****

GET https://api.powerbi.com/v1.0/collections/SmartCrokePark/workspaces/55560a09-66566-4658-a8bb-9hfghghbb9d/reports

Authorization: AppKey dgdu2277db...

****HTTP Response****

HTTP/1.1 200 OK

Content-Type: application/json; odata.metadata=minimal; odata.streaming=true

RequestId: d4099022-405b-49d3-b3b7-3c60cf675958

{

"@odata.context": "http://wabi-us-east2-redirect.analysis.windows.net/v1.0/collections/SmartCrokePark /workspaces/55560a09-66566-4658-a8bb-9hfghghbb9d/$metadata#reports",

"value": [

{

"id": "2027efc6-a308-4632-a775-rrtrt65656",

"name": "mydataset01",

"webUrl": "https://app.powerbi.com/reports/5545efc6-a308-4642-a775-b9a9186f087c",

"embedUrl": "https://embedded.powerbi.com/appTokenReportEmbed?reportId=4555efc6-a308-4632-a775-b9a9186f087c",

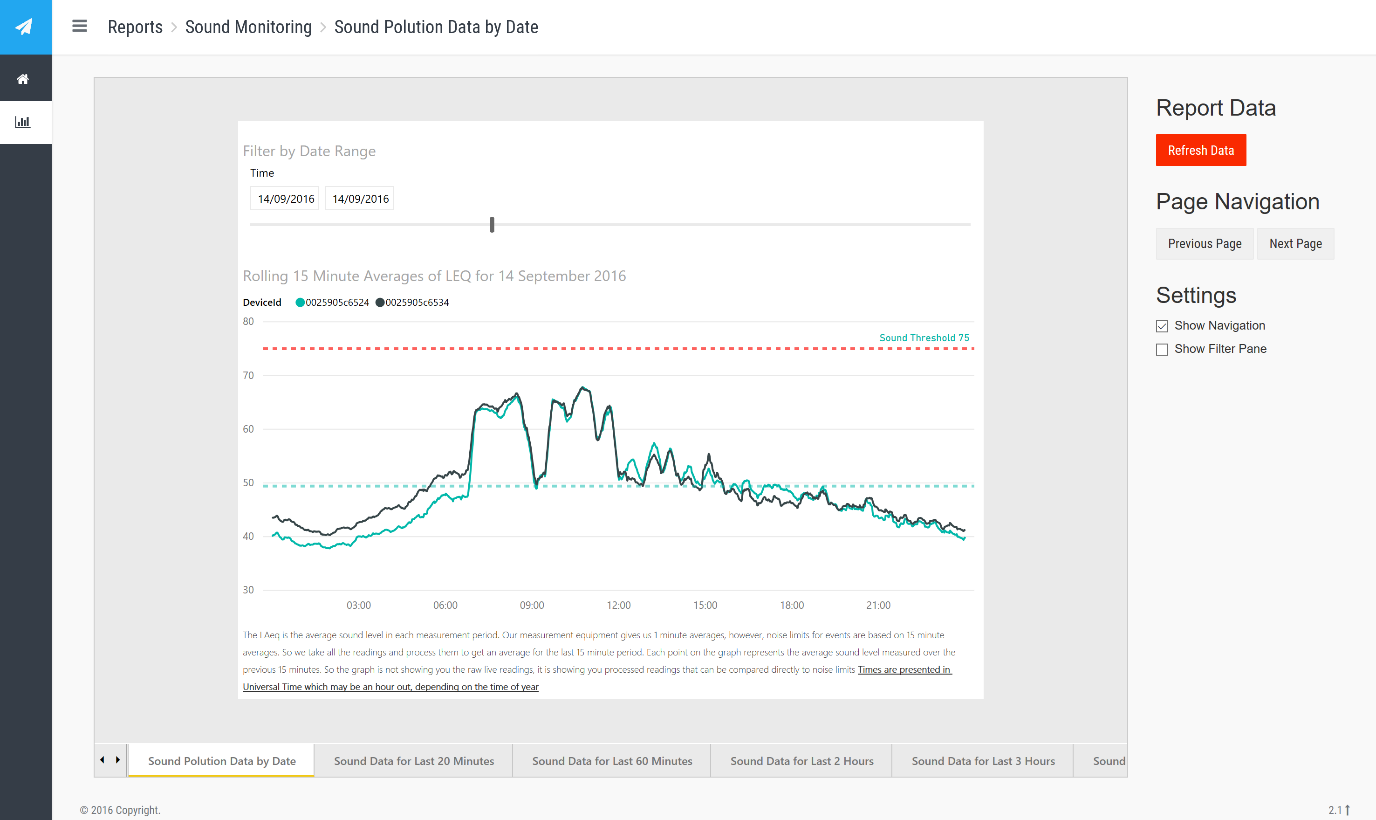
"isFromPbix": false

}

]

}

We can now embed the report in our web app using the previous app token and here’s the results with each report running in an IFrame container.



**Figure 9 - Power BI Report for Sound**

## Engagement Approach and Team

One of the challenges in working on the smart stadium project was the various skills sets required, including:

1. Sound and weather monitoring specialists. This activity was primarily carried out by Croke park staff and Sonitus a specialist sound monitoring organisation.
2. Gateway management, including: deployment, networking and development. Intel deployed and managed all gateways within the stadium with GAA IT staff providing connectivity to the network and internet, where required.
3. Stadium staff for access control and health and safety monitoring.
4. Cloud specialists to handle the ingestion and analysis of collected data. Cloud capabilities were provided by Microsoft.
5. Business intelligence and UX experts to develop dashboards and user interfaces for displaying the data in effective ways. All BI dashboarding was provided by Microsoft.
6. Data scientists to analyse data and develop predictive models to proactively act on intelligence extracted from historical data. All data science work was carried out by a team of research scientists at Dublin City University (DCU).

Bringing all of these resources together and successfully managing the delivery of each use case was challenging and required a governance model managed by two core teams:

1. A central governance team responsible for agreeing use cases and alignment between all parties. This team managed the budget for all delivery and provided the direction for prioritising the delivery of specific use cases.
2. A core technical team responsible for designing and implementing solutions for each use case.

As this project grows in the future with more phases added to the overall solution it is envisioned that the engagement team will grow and this will result in multiple teams including operational teams. These learnings can then be applied to other Smart Stadium / City projects.

It is worth noting that this project along with several other projects that are planned, are all part of a broader solution around moving these all into Production. As such this project is now moving into a production ready phase.