Optimización de la producción controlando la temperatura de las líneas

Oriol Pla Vallejo
Plan de Estudios del Estudiante
Big Data

José Luís Gómez García
Josep Curto Díaz

07/07/2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Optimización de la producción controlando la temperatura de las líneas*
Nombre del autor:	*Oriol Pla Vallejo*
Nombre del consultor/a:	*José Luís Gómez García*
Nombre del PRA:	*Josep Curto Díaz*
Fecha de entrega (mm/aaaa):	07/2017
Titulación:	*Máster en inteligencia de negocio y Big Data*
Área del Trabajo Final:	*Análisis de un conjunto de datos de un área de negocio concreta*
Idioma del trabajo:	*Español*
Palabras clave	*Real Time,* *Análisis de datos,* *Internet of things*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Todo el mundo Big Data está creciendo cada vez más y las empresas están viendo también las ventajas que puede aportarles este a la mejora de sus compañías. Sumando el hecho que con los servicios en la nube como es Azure, el coste de las arquitecturas se reduce, está suponiendo que cada vez se implementen más proyectos de este tipo.

Actualmente, existe en uno de nuestros clientes la necesidad de implementar un sistema Big Data para poder controlar todos los sensores de temperatura de las diferentes líneas de producción que tiene repartidas por sus plantas. Esta necesidad surge del hecho que un cambio de temperatura de 5 grados puede traducirse en una disminución del rendimiento de la producción, por lo que el control de las temperaturas y las acciones correctoras en el menor tiempo posible es vital para poder optimizar sus beneficios.

Por este motivo, utilizando la tecnología que nos proporciona Microsoft Azure, se implementará una PoC siguiendo una arquitectura Lambda, que permitirá a la empresa realizar la captura en tiempo real de los diferentes sensores y analizarlos al momento en sus informes desarrollados con Power BI.

Además de la ejecución de los datos en real time, se tratarán los mismos mediante MapReduce para poder así sacar las temperaturas medias por día.

Abstract (in English, 250 words or less):

The Big data world is growing more and more every day. Several companies are seeing the advantages that big data can provide them to improve their internal functions, sells previsions and procedures. Adding the fact that the cloud services as Azure reduces the cost of the architecture, this assumes that more and more projects of this type are being implemented.

Currently, one of our costumers requires to implement Big data system to control all the temperature sensors of the different production lines that are distributed by its plants. This need arises from the fact that a temperature change of 5 degrees could be translated as a decrease of the production rates, because of that the control of temperatures and corrective measures as close as possible is vital to optimize its benefits.

For this reason, using the technology provided by Microsoft Azure, a PoC will be implemented following a Lambda architecture, which will allow the company to capture in real time the different temperature sensors and analyse them in their reports developed with Power BI.

In addition to real time data execution, it will be treated using MapReduce to get extract the temperature average per day.

Índice

1. Introducción

1.1. Contexto y justificación del Trabajo

Des de la empresa donde estoy trabajando hay la necesidad de crear una PoC relacionada con el mundo Big Data

Esta iría dirigida a controlar los sensores de temperatura que una empresa real tiene repartidos por sus diferentes líneas (por temas de confidencialidad no diremos de que son las líneas ni a que se dedica la empresa, tampoco creo que sea necesario especificarlo para realizar el TFM) de cada una de las plantas.

El objetivo de la implementación del proyecto es poder controlar las temperaturas en tiempo real, para que estas no suban nunca de un cierto valor, puesto que si no el rendimiento de la máquina empieza a bajar de forma exponencial.

Así pues, lo que necesitaríamos crear es un flujo completo de trabajo utilizando todo el entorno de Microsoft Azure, para así poder realizar un seguimiento de los datos en tiempo real y por otro lado tratar los mismos, para ver las medias, máximas y mínimas diarias, y así poder realizar las tomas de decisiones que sean necesarias.

El esquema que se seguirá para la realización del proyecto es el siguiente:

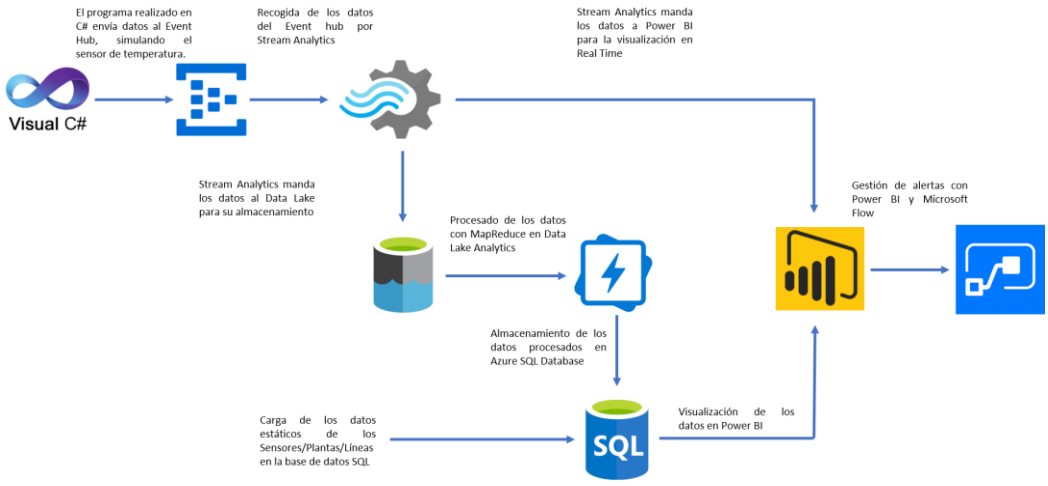


Figura 1.1- Diagrama del flujo de datos

1.2. Objetivos del Trabajo

- Controlar las temperaturas en real time utilizando Power BI
- Poder visualizar el histórico de temperaturas máximas, mínimas y medias diarias utilizando Power BI.

1.3. Enfoque y método seguido

Para la realización del proyecto se utilizarán los servicios de Microsoft Azure, puesto que es un proyecto real y somos partners de Microsoft.

1.4. Planificación del Trabajo

Para la realización del proyecto se hará uso de los siguientes recursos:

- Visual C#: para la creación de los datos.
- Azure Event hub: Para la obtención de los datos en tiempo real des de la nube.
- Azure Stream Analytics: Para la gestión de los datos en la nube en tiempo real. Este punto será el encargado de enviar los datos a Power BI para la visualización en tiempo real, o de enviarlos hacia la parte de gestión.
- Azure Data Lake: Almacenaje de los datos en su estructura original.
- Azure Data Factory: Capacidad de mover los datos dentro de Azure.
- Azure HDInsight: Creación de Map Reduce para calcular las máximas, mínimas y medias diarias.
- Azure SQL Database: Almacenaje tanto de los datos resultantes del Map Reduce como de los datos referentes a la empresa como las diferentes líneas, dispositivos, etc.
- Power BI: Creación de los informes necesarios para poder visualizar los datos.
- Microsoft Flow: Gestión de las alertas generadas en Power BI.

Las tareas a realizar en el proyecto son:

1- Propuesta inicial
2- Desarrollo del aplicativo de generación de datos en C#
3- Creación de Azure Event Hub
4- Creación y parametrización de Azure Stream Analytics
5- Creación de Azure Data Lake
6- Creación y parametrización de Azure Data Factory
7- Creación de Azure HDIsnight
8- Generación del código Map Reduce
9- Creación de la base de datos en Azure SQL Database
10- Creación de los informes en Power BI
11- Creación de las alertas en Microsoft Flow
12- Test del flujo
13- Validación del flujo
14- Corrección de los defectos detectados en el flujo
15- Creación de la estructura de la memoria
16- Creación de la memoria
17- Correcciones de la memoria

La entrega del proyecto final tiene como límite la fecha 07/07/2017, dividiéndose en las siguientes sub entregas:

- PEC 1: 03/04/2017
- PEC 2: 08/05/2017
- PEC 3: 05/06/2017

- PEC 4: 07/07/2017

Por lo que la planificación del proyecto la podemos observar en el siguiente diagrama de Gant:

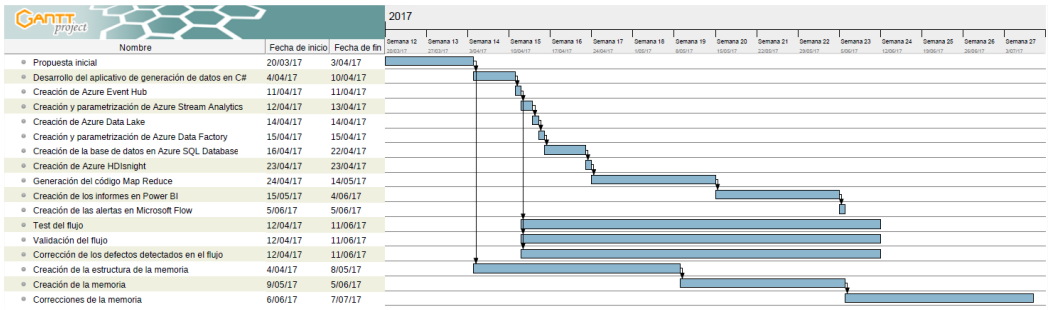


Figura 1.2- Diagrama de Gant del proyecto

1.5. Breve sumario de productos obtenidos

El producto final será un flujo de datos completo utilizando una arquitectura lambda con Microsoft Azure, que permitirá controlar las temperaturas de las líneas para tomar las acciones necesarias para optimizar la producción.

1.6. Breve descripción de los otros capítulos de la memoria

En los capítulos que siguen a la introducción, se hablará por un lado del Estado del Arte, analizando algunos proyectos similares a este.

Por otro lado, en el punto 3, se hablará de los puntos del proyecto, analizando los puntos principales de la arquitectura Lambda, así como de los diferentes pasos que forman el proyecto, des de la obtención de los datos hasta la presentación del informe final en Power BI, describiendo como se han realizado y explicando cada una de las tecnologías utilizadas.

A continuación, en el punto 4 se expone la estimación del presupuesto mensual y anual para la contratación de los servicios necesarios en Azure.

Se adjuntará también, en el punto 5, un apartado de conclusiones, con las observaciones más importantes del proyecto.

Se añadirá también el punto 6 con un glosario de los principales términos, y el punto 7 con la bibliografía.

Finalmente, se añadirá los anexos en el punto 8, en el que se incluirá el código desarrollado en cada uno de los puntos del proyecto.

2. Estado del arte

La instauración del Big Data en las empresas va creciendo día a día, entre otros factores, por la necesidad de las mismas de poder analizar y utilizar los diferentes datos generados lo más rápido posible.

De aquí nace la arquitectura Lambda, de mano de Nathan Marz, como capacidad del Big Data de gestionar los datos en Real Time. Este tipo de arquitectura cuenta con más de un proyecto que pueden servir como referencia, como pueden ser:

Xuejun Ding et al., [1] presentan un artículo en el junio del 2016 en el que proponen un algoritmo basado en los datos recogidos de los sensores inalámbricos para el análisis del riesgo de las operaciones industriales.

Muhammad Mazhar Ullah Rathore et al., [2] publican un artículo en el que presentan un modelo de análisis Big Data en real time para sensores remotos vía satélite.

Como se puede observar, el IoT (internet of things) va muy atado a las implementaciones Big Data en Real Time, ya que existe la necesidad de recuperación de datos y realización de reajustes en la mayor brevedad posible. Se podría tener como ejemplo la realización de mantenimientos preventivos, para anticiparse a los problemas, donde varias empresas de fabricación de productos como las que se comentan a continuación ya lo han implementado con éxito:

- Rolls-Royce implementó Azure IoT para realizar el mantenimiento de los más de 13000 motores de aeronaves comerciales que tienen sus clientes en todo el mundo, obteniendo beneficios en el consumo de combustible y los mantenimientos.
- Jabil, realizó la implementación también con Azure para reducir los costes de mantenimiento no planificados y tiempos de inactividad, consiguiendo así un mayor porcentaje de servicio al cliente.
-

También podemos encontrar en el sector del deporte y las carreras un caso de obtención de datos en tiempo real:

- La conocida carrera de automovilismo Indianápolis 500 (también conocida como las 500 millas de Indianápolis), también implementaron un sistema con Azure, que permitía que los seguidores pudieran ver diferentes datos estadísticos en tiempo real y con un gran nivel de detalle.

En el sector de la salud, también se encuentra el siguiente caso:

- Liebherr realizó la implementación puesto que, al almacenar diferentes medicamentos y materiales sensibles en sus refrigeradores, necesitaban tanto la monitorización remota del mismo como un mantenimiento predictivo para anticiparse a los posibles problemas que pudieran tener los mismos.

Observando los diferentes casos de éxito que existen sobre las implementaciones Big Data en Real Time, se entenderá que el proyecto ha sido

un éxito siempre y cuando se puedan analizar los diferentes datos recibidos de los sensores en tiempo real.

En cualquier caso, al tratarse de una prueba de concepto, no se podrá calcular si realmente la implementación del proyecto ha conllevado una disminución de costes de mantenimiento, así como un aumento del servicio al cliente para la empresa. En este caso, para vender el proyecto, nos podemos basar en los casos de éxito ya contados, en los que se demuestra que las empresas que realizan estas implementaciones tienen beneficios tanto a nivel interno de empresa como para sus clientes.

Finalmente añadir que los casos de éxito que, de las diferentes compañías por sectores de Azure, se han obtenido de la misma web de Azure [3], donde se pueden encontrar también más casos de éxito: https://www.microsoft.com/es-es/internet-of-things/customer-stories

3. Arquitectura lambda

La arquitectura lambda, es una arquitectura de procesado de datos Big Data pensada por Nathan Marz y diseñada para poder gestionar datos tanto en tiempo real como en procesamiento en modo batch.

Esta arquitectura, divide el proceso en tres capas:

- Batch Layer
- Speed Layer
- Serving Layer

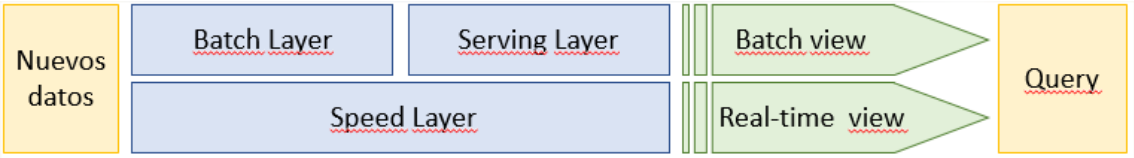


Figura 3.1- Diagrama genérico Arquitectura Lambda

Según la figura anterior, el esquema del proyecto sería el siguiente:

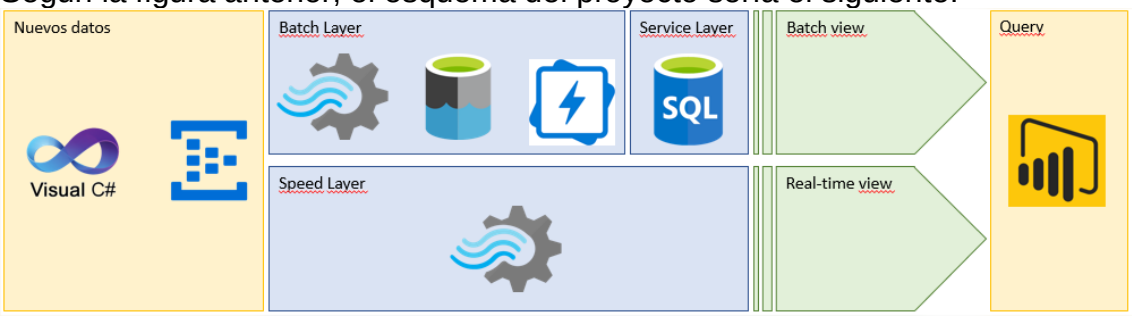


Figura 3.2- Diagrama de la arquitectura lambda del proyecto

3.1. Nuevos datos

Se recolectan los datos referentes a la temperatura, de los diferentes sensores que hay repartidos por cada una de las líneas de producción de las plantas.

Para la simulación de los datos de temperatura, se ha creado un aplicativo con C# que envía datos con formato JSON a Microsoft Azure (se puede visualizar el código de la aplicación que crea los datos en el Annexo 1 – Creación de datos en C#):

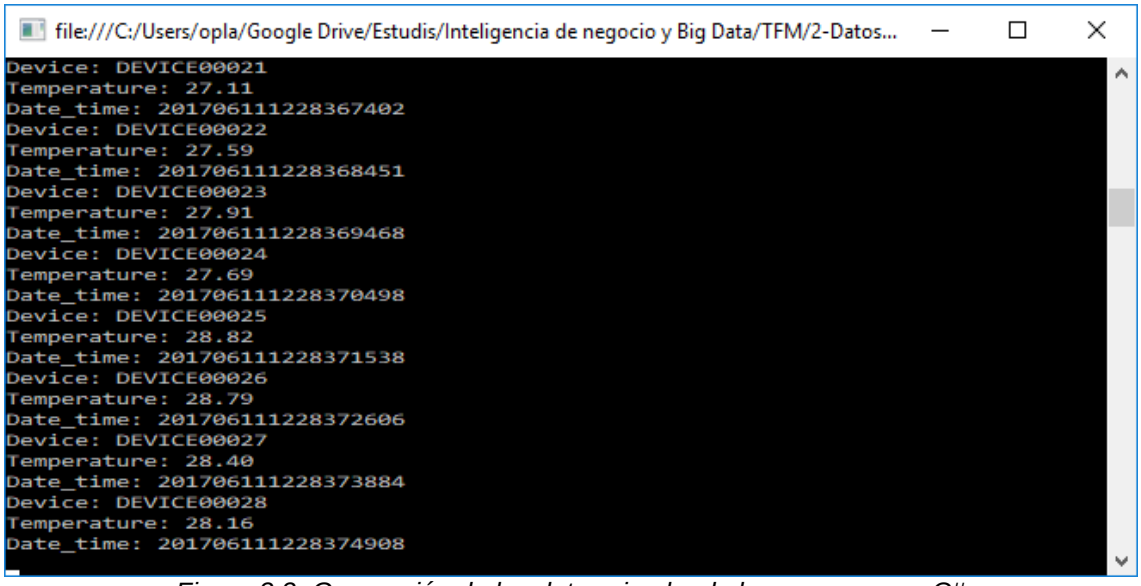


Figura 3.3- Generación de los datos simulando los sensores en C#

Los datos que se envian son los siguientes:

- Device: Id del dispositivo
- Temperature: Temperatura recogida por el dispositivo
- Date_time: Fecha y momento de recogida del dato en formato: yyyyMMddhhmmssffff

Des de Microsoft Azure, la recogida de datos se realiza mediante un servicio Event hub.

3.2. Speed Layer

La capa de velocidad se encarga de exponer los datos más recientes, sin pararse a procesarlos, indexarlos o gestionarlos.

Para la esta capa, se ha utilizado el servicio Microsoft, que permite desarrollar y ejecutar análisis paralelos en tiempo real en varios flujos de datos utilizando lenguaje SQL.

Stream Analytics, es el encargado de analizar los datos y enviarlos directamente a Power BI en tiempo real según la siguiente sentencia SQL:

```
SELECT
        Device,
        Temperature,
        SUBSTRING(Date_time,1,4) as Year,
        SUBSTRING(Date_time,5,2) as Month,
        SUBSTRING(Date_time,7,2) as Day,
        DATETIMEFROMPARTS(cast(SUBSTRING(Date_time,1,4)              as
        bigint),cast(SUBSTRING(Date_time,5,2)                        as
        bigint),cast(SUBSTRING(Date_time,7,2)                        as
        bigint),cast(SUBSTRING(Date_time,9,2)                        as
        bigint),cast(SUBSTRING(Date_time,11,2)                       as
        bigint),cast(SUBSTRING(Date_time,13,2) as bigint),00) as Date_time,
        dateadd(hh,-
        2,DATETIMEFROMPARTS(cast(SUBSTRING(Date_time,1,4)            as
        bigint),cast(SUBSTRING(Date_time,5,2)                        as
        bigint),cast(SUBSTRING(Date_time,7,2)                        as
        bigint),cast(SUBSTRING(Date_time,9,2)                        as
        bigint),cast(SUBSTRING(Date_time,11,2)                       as
        bigint),cast(SUBSTRING(Date_time,13,2) as bigint),00)) as UTC_time
INTO
        [oPowerBI]
FROM
        [iTemperaturedevices];
```

Donde iTemperaturedevices se ha configurado como la entrada del Event Hub:

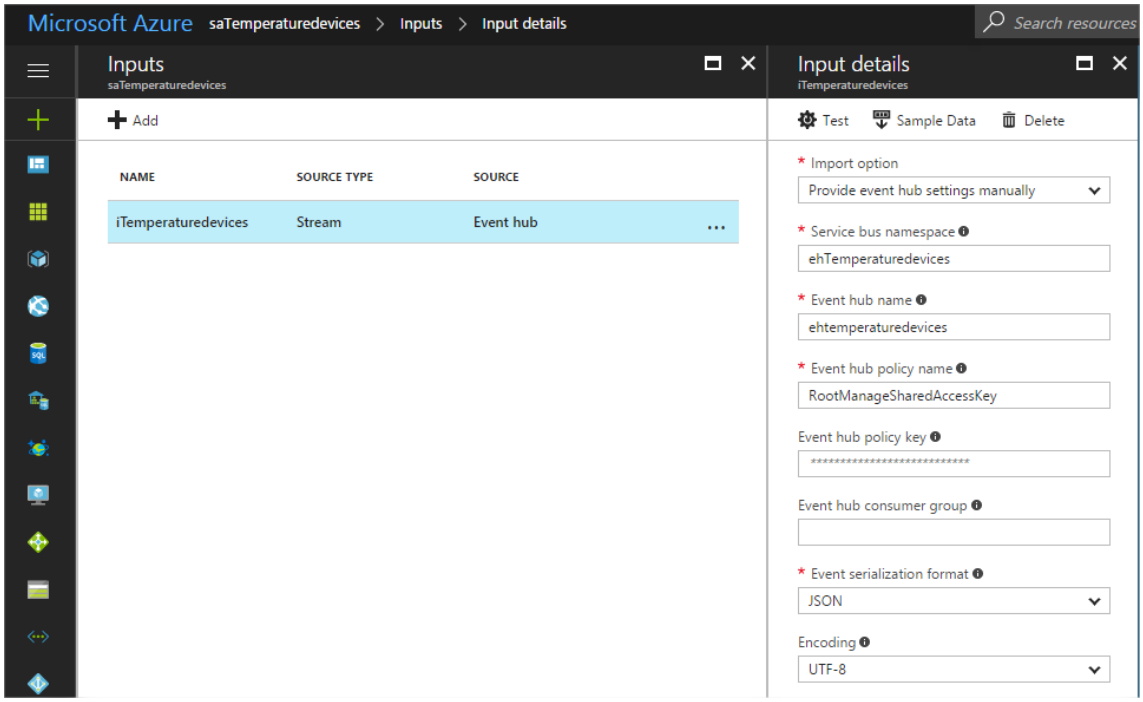


Figura 3.4- Creación del input del Event Hub en Azure Stream Analytics

I oPowerBI como la salida directa a Power BI:

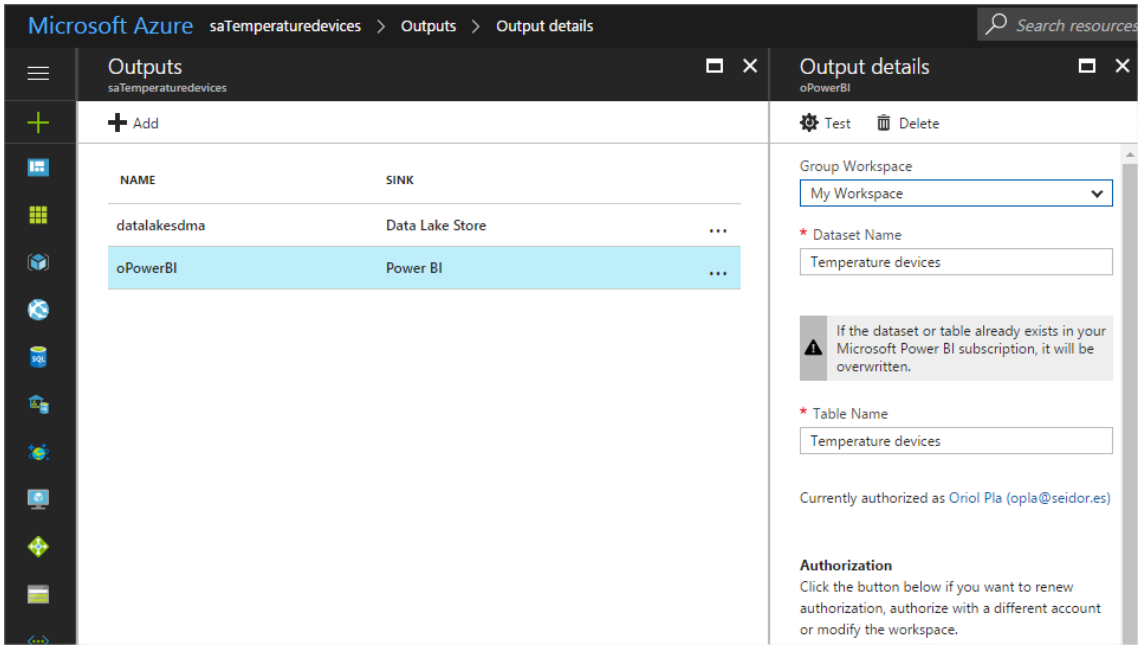


Figura 3.5- Creación del output a Power BI en Azure Stream Analytics

3.3. Batch Layer

Esta es la parte encargada de procesar los datos para así poder extraer, en este caso, las temperaturas máximas, mínimas y medias diarias por dispositivo.

Esta capa cuenta con tres partes:

1- Stream Analytics para la recogida de los datos capturados por el Event Hub.
2- Data Lake para el almacenamiento masivo de los datos.
3- Data Factory para el procesamiento de los datos, mediante Data Lake Analytics.

A continuación, se expone de forma más detallada cada una de las partes que forman esta capa.

3.3.1. Stream Analytics

Utilizando el mismo Stream Analytics que la capa de velocidad, se concatena otra consulta en SQL para el almacenamiento de los datos en un Data Lake, mediante la siguiente consulta:

```
SELECT
    *
INTO
    [datalakesdma]
FROM
    [iTemperaturedevices]
```

Donde iTemperaturedevices es la entrada del Event Hub i datalakesdma es la salida al Data Lake:

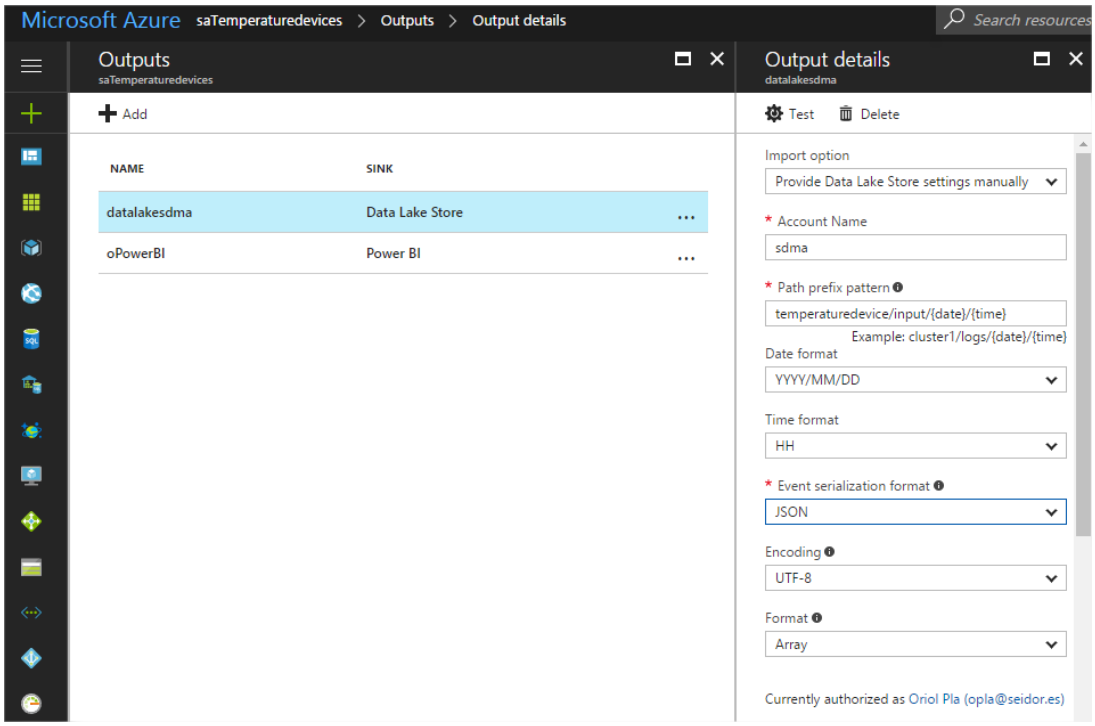


Figura 3.6- Creación del output a Data Lake Store en Azure Stream Analytics

3.3.2. Data Lake

Un Data Lake, es un repositorio donde se guardan tanto datos estructurados como no estructurados que aún no han sido procesados ni contienen ningún esquema, para ser analizados posteriormente.

El Data Lake, permite el escalado de forma masiva, así como una escritura rápida de los datos.

Se ha elegido el Data Lake por los siguientes motivos:

- Permite recoger los datos de los diferentes dispositivos con agilidad, y sin estructura.
- En caso de que la estructura de los datos recogidos cambie, se pueden seguir almacenando sin problema.
- En caso de querer recoger otras medidas, se podrían almacenar igualmente.
- La recogida de datos es en tiempo real, por lo que los mismos aumentaran en el tiempo de forma muy importante, y el Data Lake permite escalabilidad.

Se guardan los datos mediante archivos estructurados tipo JSON:

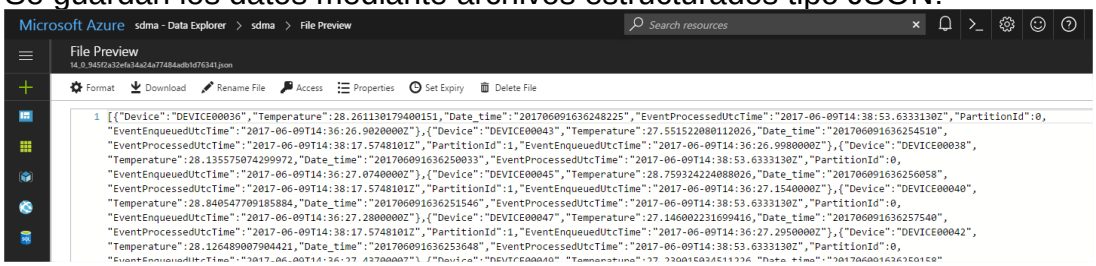


Figura 3.7- Archivo de salida con los datos de los sensores a Data Lake

Por otro lado, la estructura de carpetas utilizada sigue el formato “Input\YYYY\MM\DD\file.json”:

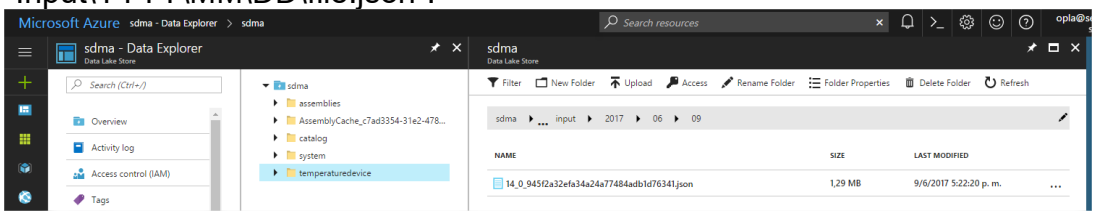


Figura 3.8- Estructura de las carpetas del Data Lake

3.3.3. Data Factory

El procesado de los datos, se ha realizado utilizando el servicio Data Factory y Data Lake Analytics.

Data Factory, es un servicio de procesamiento de datos de Azure, que permite el movimiento de los mismos y que tiene la capacidad de transformarlos, analizarlos, etc.

Las creaciones de los diferentes pasos del Data Factory se realizan mediante código escrito en formato json.

El flujo que siguen los datos dentro del Data Factory es el siguiente:

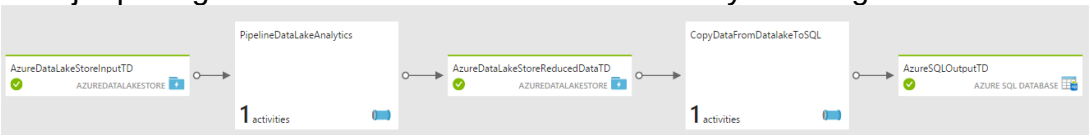


Figura 3.9- Diagrama del Data Factory

El código creado referente a los diferentes pasos, se puede visualizar en el Annexo 2 – Código en el Data Factory.

3.3.3.1. Recogida de los datos

En este primer paso, se recogen los datos del Data Lake. Para la recogida de datos, es necesario crear un código que haga referencia al servicio que contiene los datos, y posteriormente, un dataset que apunte al servicio y a la carpeta con los datos.

El Linked Service, que se ha creado es el siguiente:

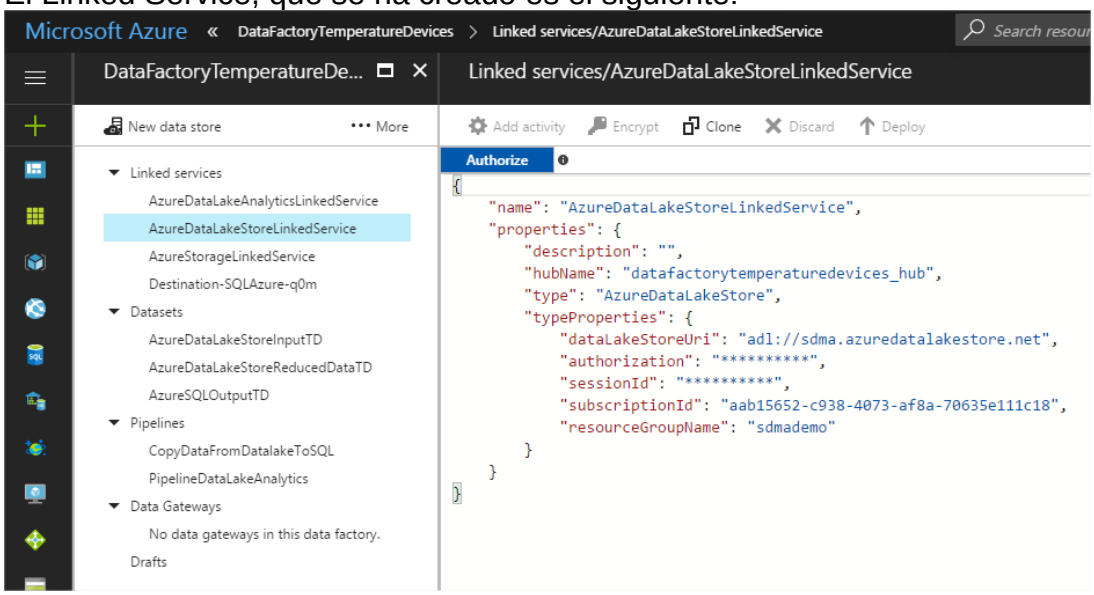


Figura 3.10- Servicio de enlace con Azure Data Lake

Por otro lado, se ha creado también el dataset:

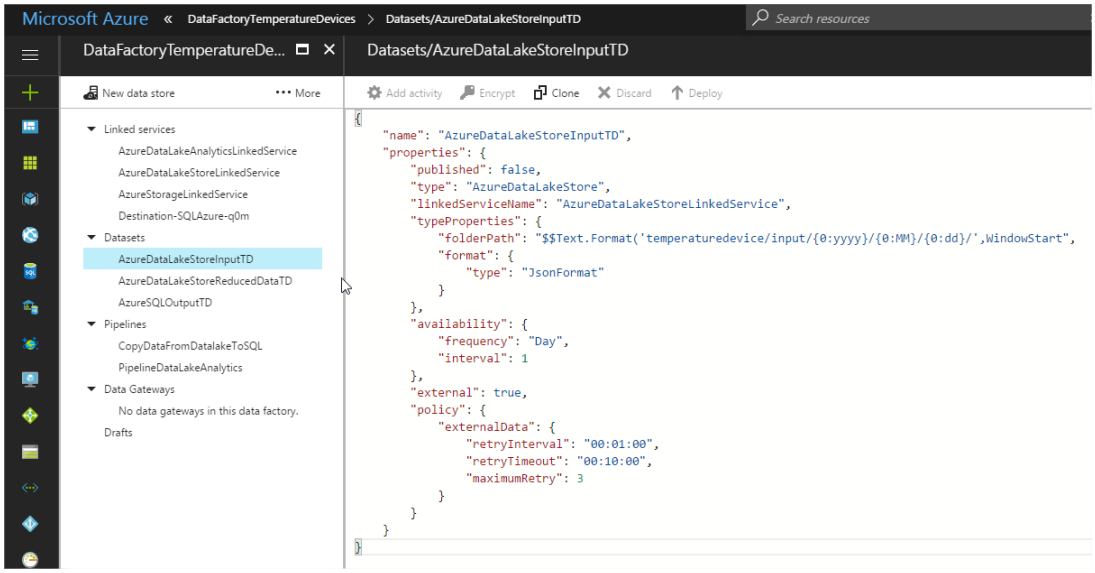


Figura 3.11- Creación en lenguaje json del Data Set en el Data Lake

3.3.3.2. Procesado con Data Lake Analytics

Data Lake Analytics, es un servicio de Azure que permite realizar procesados y transformaciones de datos mediante lenguajes U-SQL, R, Python y .NET.

Este análisi no necesita tener una infraestructura detrás, pudiendo analizar los datos a petición, permitiendo escalar las unidades de análisi de forma instantanea y pagando sólo por el tiempo que el servicio esté funcionando.

La sentencia U-SQL utilizada se ha guardado en un archivo dentro de un Azure Storage con el siguiente código:

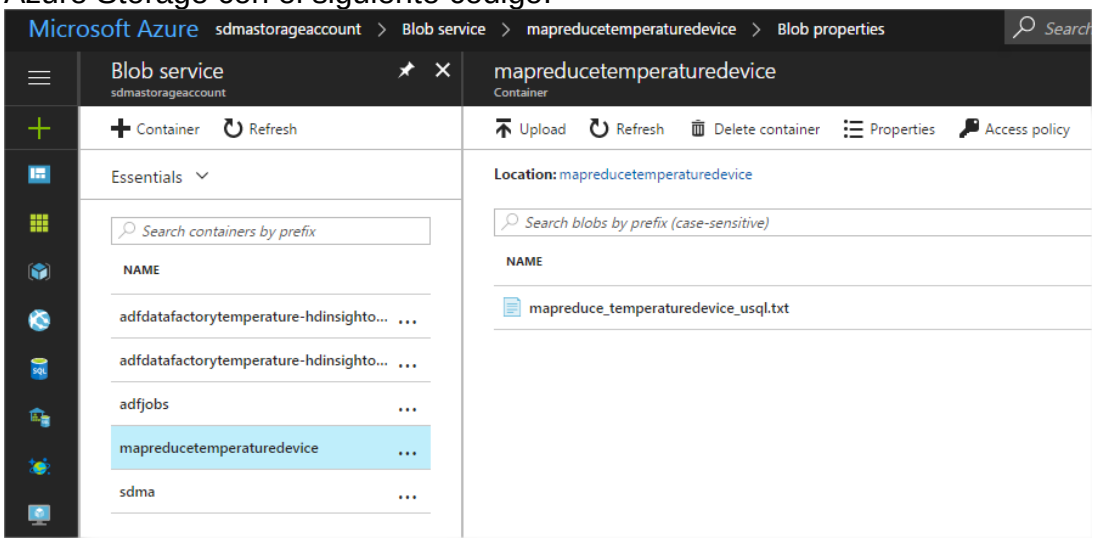


Figura 3.12- Archivo con la sentencia en USQL para ejecutar el Data Lake Analytics

La sentencia U-SQL se puede visualizar en el Annexo 2 – Código en el Data Factory, apartado DataLake Analytics – Sentencia U-SQL.

3.3.3.3. Almacenamiento de los datos en Data Lake

Des del Data Lake Analytics, el almacenamiento no se ha podido realizar directamente en la base de datos de Azure SQL, por lo que se ha tenido que guardar otra vez en el Data Lake:

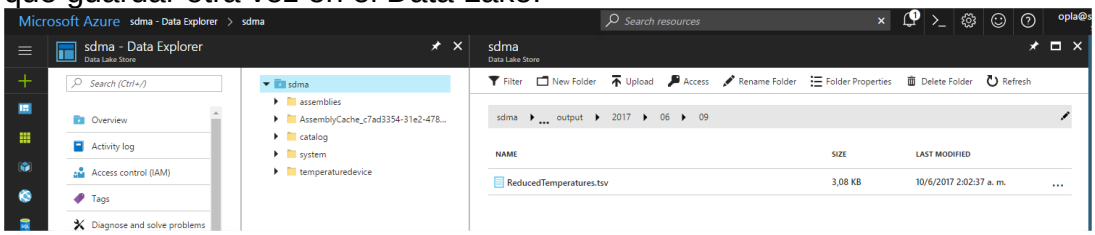
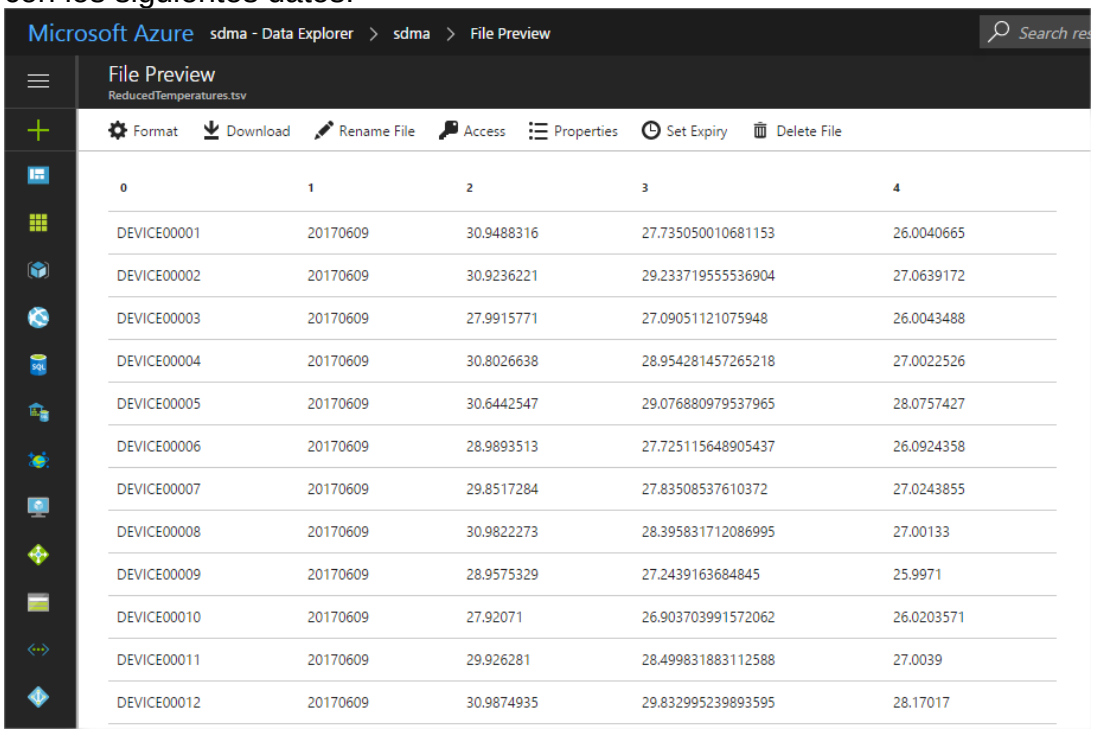


Figura 3.13- Archivo resultante de la ejecución del Azure Data Lake Analytics

Como resultado del procesado, se crea en la carpeta un archivo tipo .tsv con los siguientes datos:



0	1	2	3	4
DEVICE00001	20170609	30.9488316	27.735050010681153	26.0040665
DEVICE00002	20170609	30.9236221	29.233719555536904	27.0639172
DEVICE00003	20170609	27.9915771	27.09051121075948	26.0043488
DEVICE00004	20170609	30.8026638	28.954281457265218	27.0022526
DEVICE00005	20170609	30.6442547	29.076880979537965	28.0757427
DEVICE00006	20170609	28.9893513	27.725115648905437	26.0924358
DEVICE00007	20170609	29.8517284	27.83508537610372	27.0243855
DEVICE00008	20170609	30.9822273	28.395831712086995	27.00133
DEVICE00009	20170609	28.9575329	27.2439163684845	25.9971
DEVICE00010	20170609	27.92071	26.903703991572062	26.0203571
DEVICE00011	20170609	29.926281	28.499831883112588	27.0039
DEVICE00012	20170609	30.9874935	29.832995239893595	28.17017

Figura 3.14- Datos resultantes de la ejecución del Azure Data Lake Analytics

3.3.3.4. Copia de los datos

En este paso, se recogen los datos guardados en el Data Lake con formato .tsv, y se envian al Azure SQL.

3.3.3.5. Almacenamiento de los datos en Azure SQL Database

Finalmente, la copia de datos realizada por el Azure Data Factory se almacena en la tabla de Azure SQL Database:

Results | Messages

	Device	Date	TemperatureMax	TemperatureAvg	TemperatureMin
1	DEVICE00001	20170604	30.9488316	28.942214124343000	27.1604214
2	DEVICE00001	20170605	30.9488316	26.676403093537500	25.0190887
3	DEVICE00001	20170606	30.9488316	29.633596869076000	28.0561314
4	DEVICE00001	20170607	30.9488316	27.912876884892300	26.0273590
5	DEVICE00001	20170608	26.9728260	25.594269013942600	25.0190887
6	DEVICE00001	20170609	30.9488316	27.735050010681200	26.0040665
7	DEVICE00002	20170604	29.9195328	28.984440018149000	28.0371914
8	DEVICE00002	20170605	30.9236221	28.688485458184100	27.0503445
9	DEVICE00002	20170606	29.9195328	28.842676723704600	28.0073566
10	DEVICE00002	20170607	30.9236221	29.474179879674400	27.0978000
11	DEVICE00002	20170608	30.7825928	28.253793398539200	27.0503445
12	DEVICE00002	20170609	30.9236221	29.233719555536900	27.0639172
13	DEVICE00003	20170604	27.9820919	27.241448963389700	26.0043488
14	DEVICE00003	20170605	30.0307732	27.463199830055200	26.0043488

Figura 3.15- Datos analizados con Data Lake Analytics almacenados en SQL Database

3.4. Serving Layer

Esta capa guarda los datos procesados e indexados, pudiendo ser recuperados mediante una consulta.

Para ello, se ha utilizado un servicio Azure SQL Database, creando la siguiente base de datos:

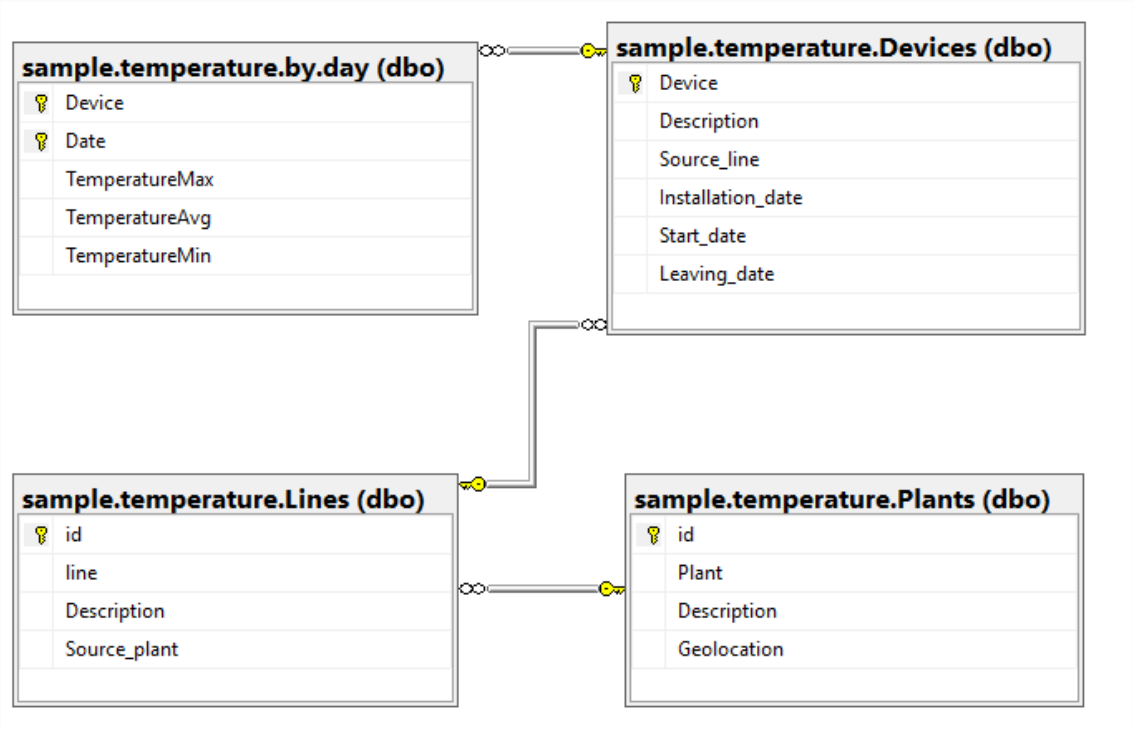


Figura 3.16- Diagrama de la base de datos

En ella se encuentra las siguientes tablas:

- Sample.temperature.by.day: datos reducidos procedentes de la capa batch.

  A continuación se exponen los tipos de datos:

  - Device: Aunque es aconsejable que la primary key sea del tipo int/bigint para facilitar la búsqueda registros, se ha optado por el tipo nvarchar para facilitar la subida de datos.
    - Primary Key
    - nvarchar(50) not null
  - Date:
    - Primary Key
    - Int not null
  - TemperatureMax:
    - Decimal
  - TemperatureAvg:
    - Decimal
  - TemperatureMin:
    - Decimal

  Se puede observar una muestra de los datos:

	Device	Date	TemperatureMax	TemperatureAvg	TemperatureMin
1	DEVICE00001	20170604	30.9488316	28.942214124343000	27.1604214
2	DEVICE00001	20170605	30.9488316	26.676403093537500	25.0190887
3	DEVICE00001	20170606	30.9488316	29.633596869076000	28.0561314
4	DEVICE00001	20170607	30.9488316	27.912876884892300	26.0273590
5	DEVICE00001	20170608	26.9728260	25.594269013942600	25.0190887
6	DEVICE00001	20170609	30.9488316	27.735050010681200	26.0040665
7	DEVICE00002	20170604	29.9195328	28.984440018149000	28.0371914
8	DEVICE00002	20170605	30.9236221	28.688485458184100	27.0503445
9	DEVICE00002	20170606	29.9195328	28.842676723704600	28.0073566
10	DEVICE00002	20170607	30.9236221	29.474179879674400	27.0978000
11	DEVICE00002	20170608	30.7825928	28.253793398539200	27.0503445
12	DEVICE00002	20170609	30.9236221	29.233719555536900	27.0639172
13	DEVICE00003	20170604	27.9820919	27.241448963389700	26.0043488
14	DEVICE00003	20170605	30.0307732	27.463199830055200	26.0043488
15	DEVICE00003	20170606	27.9892666	27.268254222891800	26.0043488

Figura 3.17- Muestra de datos de la tabla sample.temperature.by.day

- Sample.temperature.Devices: datos referentes a cada uno de los dispositivos. Los datos se entran puntualmente en caso de que se añandan o se quiten dispositivos.
  Tipos de datos:
  - Device:
    - Primary key
    - Nvarchar(50)
  - Description:
    - Nvarchar(50)
  - Source line:
    - Int
  - Installation_date:
    - Date
  - Start_date:
    - Date
  - Leaving_date:
    - Date

  Se expone a continuación la muestra de datos:

	Device	Description	Source_line	Installation_date	Start_date	Leaving_date
1	DEVICE00001	Dispositivo de temperatura	1	2017-03-19	2017-03-19	NULL
2	DEVICE00002	Dispositivo de temperatura	1	2017-03-19	2017-03-19	NULL
3	DEVICE00003	Dispositivo de temperatura	1	2017-03-19	2017-03-19	NULL
4	DEVICE00004	Dispositivo de temperatura	1	2017-03-19	2017-03-19	NULL
5	DEVICE00005	Dispositivo de temperatura	2	2017-03-19	2017-03-19	NULL
6	DEVICE00006	Dispositivo de temperatura	2	2017-03-19	2017-03-19	NULL
7	DEVICE00007	Dispositivo de temperatura	2	2017-03-19	2017-03-19	NULL
8	DEVICE00008	Dispositivo de temperatura	2	2017-03-19	2017-03-19	NULL
9	DEVICE00009	Dispositivo de temperatura	3	2017-03-19	2017-03-19	NULL
10	DEVICE00010	Dispositivo de temperatura	3	2017-03-19	2017-03-19	NULL
11	DEVICE00011	Dispositivo de temperatura	3	2017-03-19	2017-03-19	NULL
12	DEVICE00012	Dispositivo de temperatura	3	2017-03-19	2017-03-19	NULL
13	DEVICE00013	Dispositivo de temperatura	4	2017-03-19	2017-03-19	NULL
14	DEVICE00014	Dispositivo de temperatura	4	2017-03-19	2017-03-19	NULL
15	DEVICE00015	Dispositivo de temperatura	4	2017-03-19	2017-03-19	NULL

Figura 3.18- Muestra de datos de la tabla sample.temperature.devices

- Sample.temperature.Lines: datos referentes a las líneas de producción. Igual que en los dispositivos, los datos se entran puntualmente.
  Los tipos de datos de cada columna son:
  - Id:
    - Primary Key
    - Int
  - Line:
    - Nvarchar(50)
  - Description:
    - Nvarchar(50)
  - Source_plant:
    - Int

  La muestra de datos es:

	id	line	Description	Source_plant
1	1	Line0001	Tren de laminación redondos	1
2	2	Line0002	Tren de laminación Pomini	1
3	3	Line0003	Tren de laminación de perfiles estructurales	1
4	4	Line0004	Tren de laminación redondos	2
5	5	Line0005	Tren de laminación Pomini	2
6	6	Line0006	Tren de laminación de perfiles estructurales	2
7	7	Line0007	Tren de laminación redondos	3
8	8	Line0008	Tren de laminación Pomini	3
9	9	Line0009	Tren de laminación de perfiles estructurales	3
10	10	Line0010	Tren de laminación redondos	4
11	11	Line0011	Tren de laminación Pomini	4
12	12	Line0012	Tren de laminación de perfiles estructurales	4
13	13	Line0013	Tren de laminación redondos	5
14	14	Line0014	Tren de laminación Pomini	5

Figura 3.19- Muestra de datos de la tabla sample.temperature.Lines

- Sample.temperature.Plants: datos referentes a las diferentes plantas industriales. Los datos se entran de forma puntual.

  Los tipos de datos de las columnas son:
  - Id:
    - Primary key
    - Int
  - Plant:
    - Nvarchar(50)
  - Description:
    - Nvarchar(250)
  - Geolocation:
    - Geography

  Se expone a continuación la muestra de datos:

	id	Plant	Description	Geolocation
1	1	P001	Planta de Martorell - Barcelona	0xE6100000010C57F3774A62BD44400B88FF2A76CAFE3F
2	2	P002	Planta del Polígon Industrial Santiga - Barcelona	0xE6100000010C9ACE4E0647C34440D3EF671DFA330140
3	3	P003	Planta del Polígon Industrial Camí dels Frares - ...	0xE6100000010CBEDBBC7152CE4440BE1248895D9BE53F
4	4	P004	Planta del Polígon Industrial Els Massets - Tarr...	0xE6100000010C8DE66D11BD9D444028EDB204BEC7F83F
5	5	P005	Planta del Polígon Industrial Puigtió - Girona	0xE6100000010C2CE5C63E12E54440C04EA03EB8D10540

Figura 3.20- Muestra de datos de la tabla sample.temperature.Plants

En el Annexo 3 – Código Azure SQL Database, se puede ver el código de creación de la base de datos.

3.5. Visualización de los datos con Power BI:

La visualización de los datos se ha realizado con la herramienta de análisis Power BI.

3.5.1. Datasets

Los datasets, son los conjuntos de datos con los que se va a crear un informe. Cada informe puede contener datos de un solo dataset, por lo que para poder realizar la visualización tanto en tiempo real como de los datos procesados se ha creado dos conjuntos de datos:

- Devices: Datos que provienen de Azure SQL Database
- Temperature devices: Datos que provienen de Stream Analytics.

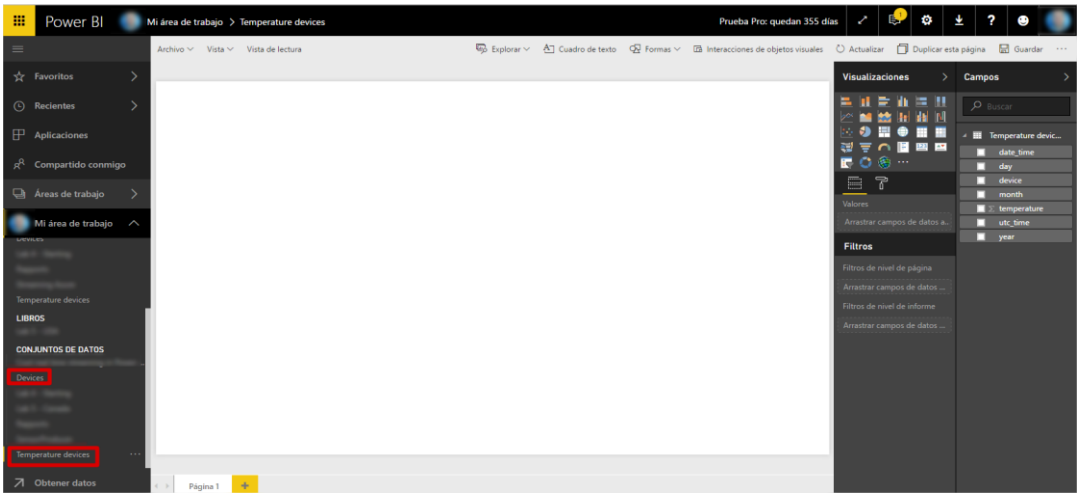


Figura 3.21- Datasets en Power BI

3.5.2. Informes

Se ha creado un informe con los datos de Azure SQL Database, que nos permite ver los datos de temperaturas por dispositivo, así como el evolutivo, la situación geográfica de los dispositivos, el recuento de dispositivos que tiene cada tipo de línea en un gráfico circular, y el recuento de dispositivos por planta en un gráfico treemap:

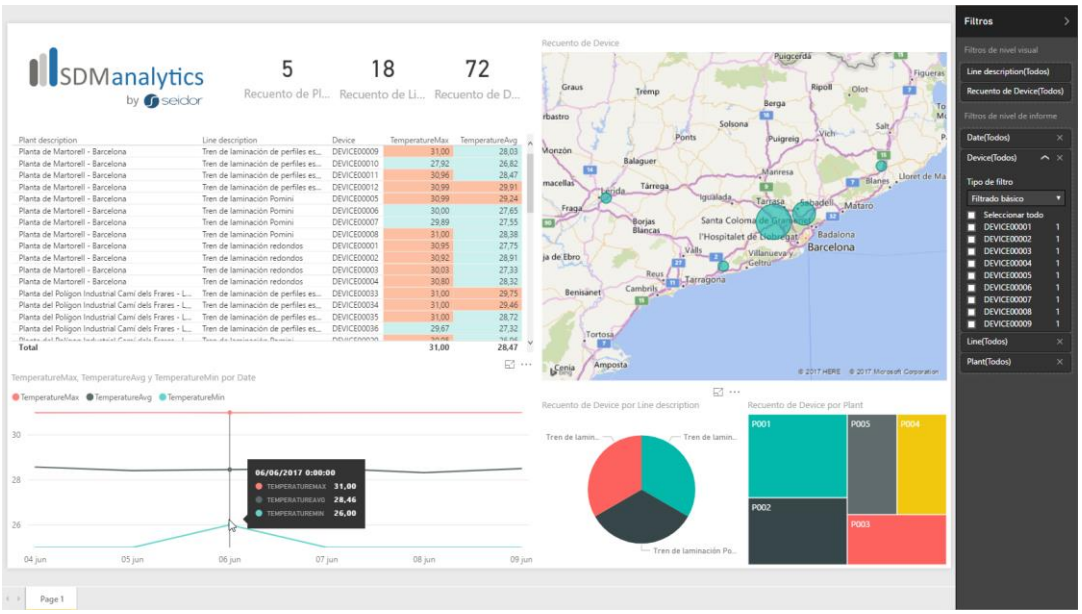


Figura 3.22- Informes en Power BI

Este informe, se puede filtrar por fecha, dispositivo, línea y planta:

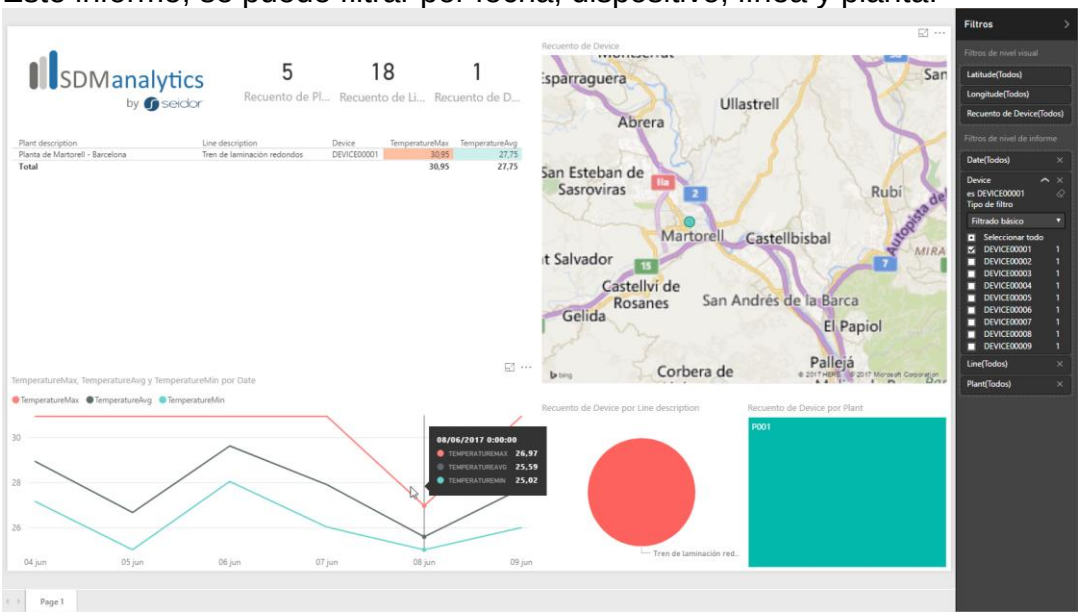


Figura 3.23- Informes filtados en Power BI

3.5.3. Paneles

Los paneles pueden estar formados por más de un informe distinto. Esto nos permite mezclar los datos que vienen de la parte Real-time con los que vienen procesados.

En el panel creado, se puede visualizar por un lado los siguientes datos en tiempo real:

- Tabla con los valores máximos, medios y mínimos de cada uno de los dispositivos en los últimos 10 minutos.
- Gráfico evolutivo del último minuto del dispositivo DEVICE0001
- La media de temperatura del último minuto para el dispositivo DEVICE0001.

También se puede visualizar los siguientes datos de la base de datos Azure SQL Database:

- Mapa anclado del informe explicado en el punto anterior.
- Gráfico tipo treemap anclado del informe anterior.

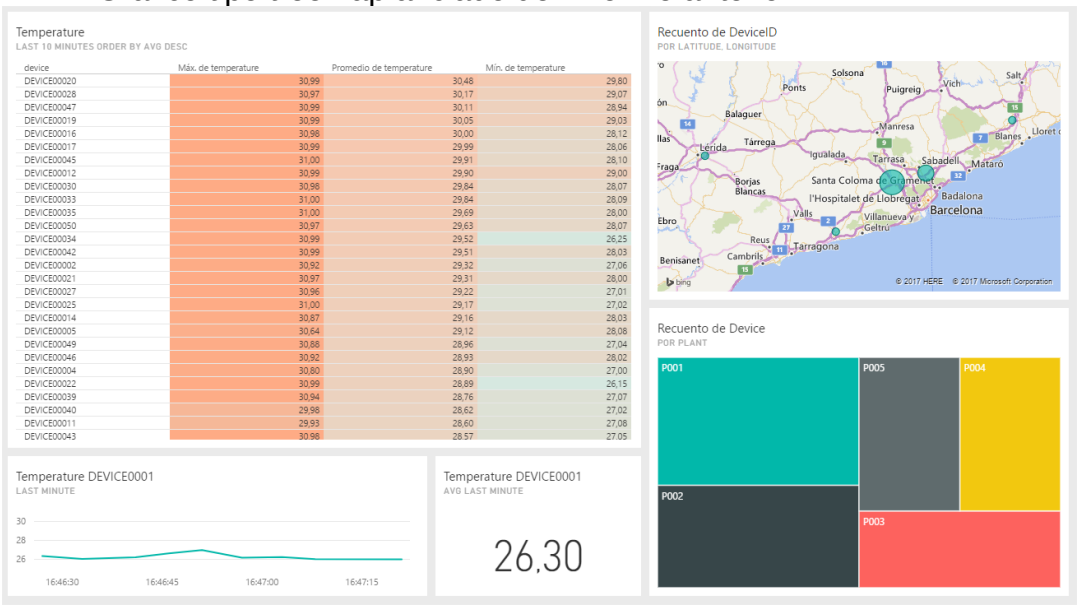


Figura 3.24- Dashboard en Real Time en Power BI

Este panel, permitiria ver en tiempo real, aquellos dispositivos con las temperaturas más elevadas, para detectar posibles problemas y poder realizar las acciones preventivas correspondientes.

4. Pressupuesto Microsoft Azure

Se ha calculado un presupuesto de 287,73€ mensuales (3452,72€ anuales) al que se tendrá que hacer frente por los servicios contratados en Microsoft Azure.

El presupuesto detallado es el siguiente:

Service type	Region	Description	Estimated Cost
Event Hubs	North Europe	Basic tier: 0 million Ingress events, 2 Throughput unit(s) x 744 Hours	€18,82
Stream Analytics	North Europe	2 unit(s), 744 Hours	€150,58
Data Lake Store	North Europe	Pago por uso: 1 TB Storage, 1300 Read Transactions, 1300 Write Transactions	€92,88
Data Factory	North Europe	Cloud: 1 low frequency and 0 high frequency activities, 0 re-run activities, 0 data movement hour(s) On-Premises: 0 low frequency and 0 high frequency activities, 0 re-run activities, 0 data movement hour(s)	€0,00
Data Lake Analytics	North Europe	Pago por uso type, 30 analytic unit(s), 0.25 hour(s)	€12,65
SQL Database	North Europe	Base de datos única, Estándar tier, S0 level, 10 DTUs, 250 GB storage per DB	€12,67
Storage	North Europe	Archivo type, GRS redundancy	€0,12
Support		Support	€0,00
		Monthly Total	€287,73
		Annual Total	€3.452,72

Disclaimer

All prices shown are in Euro (€). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/

Tabla 4.1- Presupuesto de los servicios contratados en Microsoft Azure

5. Conclusiones

Este proyecto se ha centrado en implementar una arquitectura lambda con la finalidad de analizar las temperaturas de los diferentes sensores que hay en las diferentes líneas de producción, para así poder optimizar la producción, puesto que las altas temperaturas disminuyen el rendimiento de las líneas.

Aunque no podemos concluir si esto ha llegado a aumentar el rendimiento y la producción puesto que se trata de un proyecto piloto, si que podemos realizar una serie de observaciones:

- Se ha conseguido analizar las temperaturas de los diferentes dispositivos en tiempo real, con lo que podemos detectar al momento aquellas líneas que tienen una temperatura más elevada y realizar las acciones preventivas necesarias.
- Se procesan los datos obtenidos para ver las medias, máximas y mínimas, por lo que sería posible cruzar los datos con la producción y ver cómo afecta la temperatura a la misma, así como analizar si realmente el hecho de realizar el seguimiento en tiempo real mejora la producción con el tiempo.
- Se ha implementado una estructura de análisis y procesado de datos básica con mucho potencial, puesto que se podría, por ejemplo, añadir un procesado con Azure Machine Learning que permita analizar patrones para detectar posibles fallos en las líneas y así poder realizar las acciones preventivas pertinentes que permitan reducir los paros y aumentar la producción. Es posible añadir los datos de otros tipos de sensores (por ejemplo, de presión) al sistema, para mejor así la capacidad de detectar problemas y aumentar la producción.

Un posible problema en la implementación del mismo podría ser el coste de mantenimiento de los servicios contratados en Microsoft Azure, por lo que sería necesario saber si compensa el precio a pagar con las posibles ganancias de implementación.

6. Glosario

C#: lenguaje de programación orientado a objetos. Fue desarrollado y estandarizado por Microsoft formando parte de su plataforma .NET.

Dashboard: representación gráfica de los principales KPI o métricas de un informe.

Data Lake: repositorio donde se guardan tanto datos estructurados como no estructurados que aún no han sido procesados ni contienen ningún esquema, para ser analizados posteriormente.

Dataset: conjunto de datos recuperado de un proveedor de datos.

JSON (JavaScript Object Notation): formato de archivo para el intercambio de datos.

Nube (Cloud): paradigma que permite ofrecer servicios computación en internet.

PoC (Proof of Concept): es una demostración que tiene como finalidad verificar que un proyecto, en este caso, tiene el potencial de ser aplicado en el mundo real con éxito.

Python: es un lenguaje de programación interpretado, multiplataforma y orientado a objetos.

R: entorno y lenguaje de programación orientado a la estadística.

SQL (Structured Query Language): lenguaje de programación que permite realizar la extracción de información de una base de datos.

7. Bibliografía

[1] Xuejun Ding et al., *A Real-Time Big Data Gathering Algorithm Based on Indoor Wireless Sensor Networks for Risk Analysis of Industrial Operations*. Transactions on Industrial Informatics (1551-3203). Volume 12 – Nº 3, June 2016.

[2] Muhammad Mazhar Ullah Rathore et al., *Real-Time Big Data Analytical Architecture for Remote Sensing Application*. Journal of Selected Topics in Applied Earth Observations and Remote Sensing (1939-1404). Volume 8 – Nº 10, May 2015.

[3] https://www.microsoft.com/es-es/internet-of-things/customer-stories, May 2017.

[4] https://azure.microsoft.com/es-es/pricing/calculator/preview/, May 2017.

[5] https://azure.microsoft.com/es-ES/, May 2017.

8. Anexos

8.1. Anexo 1 – Creación de datos en C#

El programa de creación de datos se ha desarrollado en lenguaje de programación C# y está divido en dos archivos:

1- Program.cs: contiene el código del programa
2- Sensor.cs: contiene la creación de la clase tipo Sensor

A continuación se expone el código de cada uno de los archivos (las claves de acceso se han modificado para no hacerlas públicas):

1- Program.cs:

```
using System;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using Microsoft.ServiceBus.Messaging;
using Newtonsoft.Json;

namespace SensorProducer
{
    class Program
    {
        static int _Total_devices = 72;
        static int _Temperatura_minima = 25;
        static int _Temperatura_maxima = 31;
        static double _Temperatura_inicial = 28;
        static double _Temperatura_increase_percentage = 0.02;


        static Random _next = new Random(212323);
        static double[] _LastValue = Enumerable.Repeat(_Temperatura_inicial,
_Total_devices).ToArray();
        static int[] _MinValue = new int [_Total_devices];
        static int[] _MaxValue = new int [_Total_devices];
        static string connectionString =
"Endpoint=sb://ehTemperaturedevices.servicebus.windows.net/;SharedAcce
ssKeyName=RootManageSharedAccessKey;SharedAccessKey=W7XA8jSq
GM1mnp63n3wjOiXXXXXXXXX/J2EzoXXXXXX=;EntityPath=ehTemperatur
edevices";

        static void Main(string[] args)
        {
            Console.WriteLine("Press Enter to start now");
            Console.ReadLine();

            var tokenSource = new CancellationTokenSource();
```

```
            Console.WriteLine("Press any key to stop the sender process");
            var task = new TaskFactory()
                .StartNew(() => SendingRandomMessages(tokenSource.Token));
            Console.ReadLine();

            tokenSource.Cancel();
            task.Wait();
        }
        //Creamos una clase para devolver la fecha de creación del registro en
formato TimeStamp.
        public static String GetTimestamp(DateTime value)
        {
            return value.ToString("yyyyMMddHHmmssfffff");
        }
        static void SendingRandomMessages(CancellationToken
cancellationToken)
        {
            var hub =
EventHubClient.CreateFromConnectionString(connectionString);
            while (true)
            {
                for (int i = 1; i <= _Total_devices; i++)
                {
                    try
                    {
                        //Si se ha presionado alguna tecla, finalizar el proceso.
                        if (cancellationToken.IsCancellationRequested)
                        {
                            return;
                        }

                        //Creamos los valores de temperatura. Se ha realizado toda
esta fórmula para que no haya saltos grandes en la temperatura resultante.
                        _MinValue[i - 1] = Convert.ToInt32(_LastValue[i - 1] -
(_LastValue[i - 1] * _Temperatura_increase_percentage));
                        if (_MinValue[i - 1] < _Temperatura_minima) { _MinValue[i -
1] = _Temperatura_minima; };
                        _MaxValue[i - 1] = Convert.ToInt32(_LastValue[i - 1] +
(_LastValue[i - 1] * _Temperatura_increase_percentage));

                        if (_MaxValue[i - 1] <= _MinValue[i - 1]) { _MaxValue[i - 1] =
_MinValue[i - 1] + 1; };
                        if (_MaxValue[i - 1] > _Temperatura_maxima) { _MaxValue[i -
1] = _Temperatura_maxima; _MinValue[i - 1] = _MaxValue[i - 1] - 1; };

                        //Insertamos los valores a la clase Sensor.
                        var sensor = new Sensor()
                        {
                            Date_time = GetTimestamp(DateTime.Now),
```

```
                    Temperature = _next.Next(_MinValue[i - 1], _MaxValue[i -
1]) + _next.NextDouble(),
                    Device = "DEVICE" + i.ToString("D5")
                };

                //Guardamos el último valor de la temperatura para poder
calcular el siguiente sin tener un salto muy grande entre los dos.
                _LastValue[i - 1] = sensor.Temperature;

                //Mostramos por pantalla los valores registrados.
                Console.WriteLine("Date_time: "+sensor.Date_time);
                Console.WriteLine("Device: "+sensor.Device.ToString());
                NumberFormatInfo nfi = new CultureInfo("en-US",
false).NumberFormat;
                Console.WriteLine("Temperature:
"+sensor.Temperature.ToString("N", nfi));

                //Enviamos los datos a nuestro Eventhub en Azure.
                hub.SendAsync(new
EventData(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(sensor)))
).Wait();
            }
            //En caso de error muestra un mensaje en rojo.
            catch (Exception exception)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("{0} > Exception: {1}", DateTime.Now,
exception.Message);
                Console.ResetColor();
            }
        }

        Thread.Sleep(200);
    }
  }
}
```

2- Sensor.cs:

```
namespace SensorProducer
{
  public class Sensor
  {
    public string Device { get; set; }

    public double Temperature { get; set; }

    //public double Pressure { get; set; }

    public string Date_time { get; set; }
```

```
    }
}
```

8.2. Anexo 2 – Código en el Data Factory

8.2.1. Linked Service – AzureDataLakeAnalyticsLinkedService


```
{
    "name": "AzureDataLakeAnalyticsLinkedService",
    "properties": {
        "description": "",
        "hubName": "datafactorytemperaturedevices_hub",
        "type": "AzureDataLakeAnalytics",
        "typeProperties": {
            "accountName": "dlanalytemperaturedevice",
            "authorization": "**********",
            "sessionId": "**********",
            "subscriptionId": "aab15652-c938-4073-af8a-70635e111c18",
            "resourceGroupName": "sdmademo"
        }
    }
}
```


8.2.2. Linked Service – AzureDataLakeStoreLinkedService


```
{
    "name": "AzureDataLakeStoreLinkedService",
    "properties": {
        "description": "",
        "hubName": "datafactorytemperaturedevices_hub",
        "type": "AzureDataLakeStore",
        "typeProperties": {
            "dataLakeStoreUri": "adl://sdma.azuredatalakestore.net",
            "authorization": "**********",
            "sessionId": "**********",
            "subscriptionId": "aab15652-c938-4073-af8a-70635e111c18",
            "resourceGroupName": "sdmademo"
        }
    }
}
```


8.2.3. Linked Service – AzureStorageLinkedService


```
{
    "name": "AzureStorageLinkedService",
    "properties": {
        "description": "",
        "hubName": "datafactorytemperaturedevices_hub",
        "type": "AzureStorage",
        "typeProperties": {
            "connectionString":
"DefaultEndpointsProtocol=https;AccountName=sdmastorageaccount;Accou
ntKey=**********"
        }
    }
}
```

8.2.4. Linked Service – Destination-SQLAzure-q0m


```
{
    "name": "Destination-SQLAzure-q0m",
    "properties": {
        "hubName": "datafactorytemperaturedevices_hub",
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data
Source=sdmasqlserver.database.windows.net;Initial
Catalog=sdmasqldatabase;Integrated Security=False;User
ID=****;Password=**********;Connect Timeout=30;Encrypt=True"
        }
    }
}
```


8.2.5. Datasets – AzureDataLakeStoreInputTD


```
{
    "name": "AzureDataLakeStoreInputTD",
    "properties": {
        "published": false,
        "type": "AzureDataLakeStore",
        "linkedServiceName": "AzureDataLakeStoreLinkedService",
        "typeProperties": {
            "folderPath":
"$$Text.Format('temperaturedevice/input/{0:yyyy}/{0:MM}/{0:dd}/',WindowSt
art",
            "format": {
                "type": "JsonFormat"
            }
        },
        "availability": {
            "frequency": "Day",
            "interval": 1
        },
        "external": true,
        "policy": {
            "externalData": {
                "retryInterval": "00:01:00",
                "retryTimeout": "00:10:00",
                "maximumRetry": 3
            }
        }
    }
}
```


8.2.6. Datasets – AzureDataLakeStoreReducedDataTD


```
{
    "name": "AzureDataLakeStoreReducedDataTD",
    "properties": {
        "structure": [
```

```
        {
            "name": "Prop_0",
            "type": "String"
        },
        {
            "name": "Prop_1",
            "type": "Int64"
        },
        {
            "name": "Prop_2",
            "type": "Double"
        },
        {
            "name": "Prop_3",
            "type": "Double"
        },
        {
            "name": "Prop_4",
            "type": "Double"
        }
    ],
    "published": false,
    "type": "AzureDataLakeStore",
    "linkedServiceName": "AzureDataLakeStoreLinkedService",
    "typeProperties": {
        "fileName": "ReducedTemperatures.tsv",
        "folderPath": "temperaturedevice/output/{year}/{month}/{day}",
        "format": {
            "type": "TextFormat",
            "columnDelimiter": "\t"
        },
        "partitionedBy": [
            {
                "name": "year",
                "value": {
                    "type": "DateTime",
                    "date": "WindowStart",
                    "format": "yyyy"
                }
            },
            {
                "name": "month",
                "value": {
                    "type": "DateTime",
                    "date": "WindowStart",
                    "format": "MM"
                }
            },
            {
                "name": "day",
```

```
                "value": {
                    "type": "DateTime",
                    "date": "WindowStart",
                    "format": "dd"
                }
            }
        ]
    },
    "availability": {
        "frequency": "Day",
        "interval": 1
    }
  }
}
```

8.2.7. Datasets – AzureSQLOutputTD

```
{
    "name": "AzureSQLOutputTD",
    "properties": {
        "structure": [
            {
                "name": "Device",
                "type": "String"
            },
            {
                "name": "Date",
                "type": "Int32"
            },
            {
                "name": "TemperatureMax",
                "type": "Decimal"
            },
            {
                "name": "TemperatureAvg",
                "type": "Decimal"
            },
            {
                "name": "TemperatureMin",
                "type": "Decimal"
            }
        ],
        "published": false,
        "type": "AzureSqlTable",
        "linkedServiceName": "Destination-SQLAzure-q0m",
        "typeProperties": {
            "tableName": "[dbo].[sample.temperature.by.day]"
        },
        "availability": {
            "frequency": "Day",
            "interval": 1
```

```
        },
        "external": false,
        "policy": {}
    }
}
```

8.2.8. Pipelines – CopyDataFromDatalakeToSQL

```
{
    "name": "CopyDataFromDatalakeToSQL",
    "properties": {
        "description": "Copy reduced data from Datalake to SQL",
        "activities": [
            {
                "type": "Copy",
                "typeProperties": {
                    "source": {
                        "type": "AzureDataLakeStoreSource",
                        "recursive": false
                    },
                    "sink": {
                        "type": "SqlSink",
                        "writeBatchSize": 0,
                        "writeBatchTimeout": "00:00:00"
                    },
                    "translator": {
                        "type": "TabularTranslator",
                        "columnMappings":
"Prop_0:Device,Prop_1:Date,Prop_2:TemperatureMax,Prop_3:Temperature
Avg,Prop_4:TemperatureMin"
                    }
                },
                "inputs": [
                    {
                        "name": "AzureDataLakeStoreReducedDataTD"
                    }
                ],
                "outputs": [
                    {
                        "name": "AzureSQLOutputTD"
                    }
                ],
                "policy": {
                    "timeout": "1.00:00:00",
                    "concurrency": 1,
                    "executionPriorityOrder": "NewestFirst",
                    "style": "StartOfInterval",
                    "retry": 1,
                    "longRetry": 0,
                    "longRetryInterval": "00:00:00"
                },
```

```
            "scheduler": {
                "frequency": "Day",
                "interval": 1
            },
            "name": "Copy data from Data Lake to SQL"
        }
    ],
    "start": "2017-06-04T02:00:00Z",
    "end": "2017-06-10T15:00:00Z",
    "isPaused": false,
    "hubName": "datafactorytemperaturedevices_hub",
    "pipelineMode": "Scheduled"
  }
}
```

8.2.9. Pipelines – PipelineDataLakeAnalytics

```
{
    "name": "PipelineDataLakeAnalytics",
    "properties": {
        "description": "Max, min, avg temperature devices",
        "activities": [
            {
                "type": "DataLakeAnalyticsU-SQL",
                "typeProperties": {
                    "scriptPath":
"mapreducetemperaturedevice\\mapreduce_temperaturedevice_usql.txt",
                    "scriptLinkedService": "AzureStorageLinkedService",
                    "degreeOfParallelism": 1,
                    "priority": 1,
                    "parameters": {
                        "in":
"$$Text.Format('/temperaturedevice/input/{0:yyyy}/{0:MM}/{0:dd}/{1}.json',Wi
ndowStart,'{*}')",
                        "out":
"$$Text.Format('/temperaturedevice/output/{0:yyyy}/{0:MM}/{0:dd}/Reduced
Temperatures.tsv', WindowStart)"
                    }
                },
                "inputs": [
                    {
                        "name": "AzureDataLakeStoreInputTD"
                    }
                ],
                "outputs": [
                    {
                        "name": "AzureDataLakeStoreReducedDataTD"
                    }
                ],
                "policy": {
                    "timeout": "01:00:00",
```

```
            "concurrency": 1,
            "retry": 1
        },
        "scheduler": {
            "frequency": "Day",
            "interval": 1
        },
        "name": "DataLakeAnalyticsUSqlActivityTemplate",
        "linkedServiceName": "AzureDataLakeAnalyticsLinkedService"
      }
    ],
    "start": "2017-06-04T01:00:00Z",
    "end": "2017-06-10T15:00:00Z",
    "isPaused": false,
    "hubName": "datafactorytemperaturedevices_hub",
    "pipelineMode": "Scheduled"
  }
}
```

8.2.10. DataLake Analytics – Sentencia U-SQL

```
REFERENCE ASSEMBLY [Analytics Account].[Newtonsoft.Json];
REFERENCE ASSEMBLY [Analytics
Account].[Microsoft.Analytics.Samples.Formats];

//Define schema of file, must map all columns
 @TemperatureDeviceExtract =
   EXTRACT
      Device string,
            Temperature float,
            Date_time string,
            EventProcessedUtcTime string,
            PartitionId int,
            EventEnqueuedUtcTime string
      FROM @in
   USING new Microsoft.Analytics.Samples.Formats.Json.JsonExtractor();

@TransformDate =
      SELECT
            Device,
            Date_time.Substring(0,8) AS Temperature_date,
            Temperature
      FROM @TemperatureDeviceExtract;

@FinalData =
      SELECT
            Device,
            Temperature_date,
            MAX(Temperature)          AS Temperature_max,
            AVG(Temperature)          AS Temperature_avg,
            MIN(Temperature)          AS Temperature_mMin
```

```
        FROM @TransformDate
        GROUP BY Device, Temperature_date;

OUTPUT @FinalData
    TO @out
      USING Outputters.Tsv(quoting:false, dateTimeFormat:null);
```

8.3. Anexo 3 – Código Azure SQL Database

```
USE [sdmasqldatabase]
GO
/****** Object: Table [dbo].[sample.temperature.by.day] Script Date: 11/06/2017 14:00:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[sample.temperature.by.day](
	[Device] [nvarchar](50) NOT NULL,
	[Date] [int] NOT NULL,
	[TemperatureMax] [decimal](9, 7) NULL,
	[TemperatureAvg] [decimal](17, 15) NULL,
	[TemperatureMin] [decimal](9, 7) NULL,
 CONSTRAINT [PK_sample.temperature.by.day] PRIMARY KEY CLUSTERED
(
	[Device] ASC,
	[Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)

GO
/****** Object: Table [dbo].[sample.temperature.Devices] Script Date: 11/06/2017 14:00:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[sample.temperature.Devices](
	[Device] [nvarchar](50) NOT NULL,
	[Description] [nvarchar](250) NULL,
	[Source_line] [int] NOT NULL,
	[Installation_date] [date] NULL,
	[Start_date] [date] NULL,
	[Leaving_date] [date] NULL,
 CONSTRAINT [PK_sample.temperature.Devices] PRIMARY KEY CLUSTERED
(
	[Device] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)

GO
/****** Object: Table [dbo].[sample.temperature.Lines] Script Date: 11/06/2017 14:00:55 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[sample.temperature.Lines](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[line] [nvarchar](50) NOT NULL,
	[Description] [nvarchar](250) NULL,
	[Source_plant] [int] NOT NULL,
 CONSTRAINT [PK_sample.temperature.Lines] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON)
)

GO
/****** Object: Table [dbo].[sample.temperature.Plants] Script Date:
11/06/2017 14:00:55 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[sample.temperature.Plants](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[Plant] [nvarchar](50) NOT NULL,
	[Description] [nvarchar](250) NULL,
	[Geolocation] [geography] NULL,
 CONSTRAINT [PK_sample.temperature.Plants] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON)
)

GO
ALTER TABLE [dbo].[sample.temperature.by.day] WITH CHECK ADD
CONSTRAINT [FK_sample.temperature.by.day_sample.temperature.Devices]
FOREIGN KEY([Device])
REFERENCES [dbo].[sample.temperature.Devices] ([Device])
GO
ALTER TABLE [dbo].[sample.temperature.by.day] CHECK CONSTRAINT
[FK_sample.temperature.by.day_sample.temperature.Devices]
GO
ALTER TABLE [dbo].[sample.temperature.Devices] WITH CHECK ADD
CONSTRAINT [FK_sample.temperature.Devices_sample.temperature.Lines]
FOREIGN KEY([Source_line])
REFERENCES [dbo].[sample.temperature.Lines] ([id])
```

```
GO
ALTER TABLE [dbo].[sample.temperature.Devices] CHECK CONSTRAINT [FK_sample.temperature.Devices_sample.temperature.Lines]
GO
ALTER TABLE [dbo].[sample.temperature.Lines] WITH CHECK ADD CONSTRAINT [FK_sample.temperature.Lines_sample.temperature.Plants] FOREIGN KEY([Source_plant])
REFERENCES [dbo].[sample.temperature.Plants] ([id])
GO
ALTER TABLE [dbo].[sample.temperature.Lines] CHECK CONSTRAINT [FK_sample.temperature.Lines_sample.temperature.Plants]
GO
```