Nombre: Luis Manuel García-Baquero Corredera **DNI**: 77818095-X

Prácticas de IARS

No Presenciales

Estudios: Máster en Sistemas Inteligentes en Energía y Transporte

Asignatura: Infraestructura Avanzada de Redes de Sensores

Realizado por : Luis Manuel García-Baquero Corredera <luikygaba@gmail.com>

Curso: 2018/2019

Índice

1. Introducción

En esta memoria se va a presentar la realización de las prácticas no presenciales de la asignatura IARS, en las que se va a montar una red formada por una Raspberry Pi 3B y un PC para efectuar la simulación de un sensor de temperatura, la comunicación entre los dispositivos, el control a distancia del sensor y la visualización de los datos recibidos.

Para ello se van a utilizar dos instancias de Node-RED, una instalada en la Raspberry Pi, que hace la función de servidor recogiendo los datos de los sensores, y otra en el ordenador, cuya función es la de estación base.

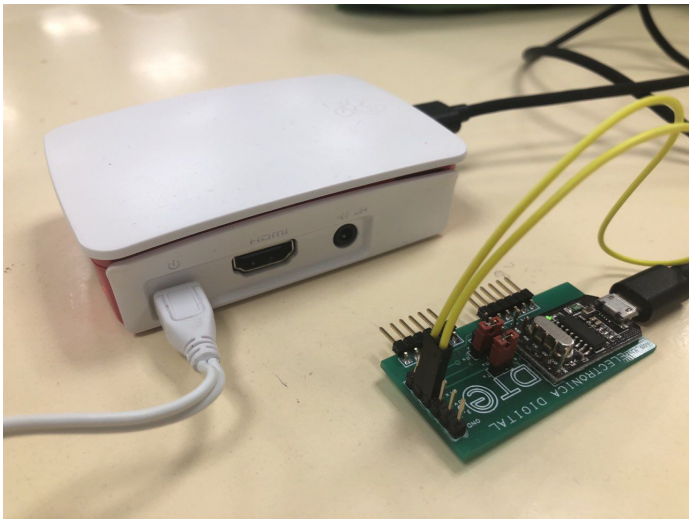
2. Práctica 1: Temperatura simulada

En esta primera práctica se pide simular el sensor de temperatura mediante la generación cíclica de un número aleatorio, el cual será enviado por el puerto serie. Para lograrlo, se ha implementado el siguiente flujo de Node-RED:

El primer nodo se encarga de inyectar de forma cíclica un valor (en este caso el timestamp del momento concreto, aunque no es de relevancia), de manera que se ejecute el flujo cada 5 segundos. El segundo nodo, al ser activado, genera un número aleatorio entre 30 y 15 con la librería Math.js:

```
// Genera el número aleatorio
var random_temp = Math.random() * (40-25)+25;

// Guarda el dato en el mensaje
msg.payload = {Temperatura: random_temp};
return msg;
```

Finalmente, el mensaje es emitido por el puerto serie. En este caso, se ha utilizado un circuito de conversión USB - Puerto serie, al que se le han cortocircuitado los pines de RX y TX para que la Raspberry Pi pueda enviar los datos por el puerto serie y leer los mismos en consola:

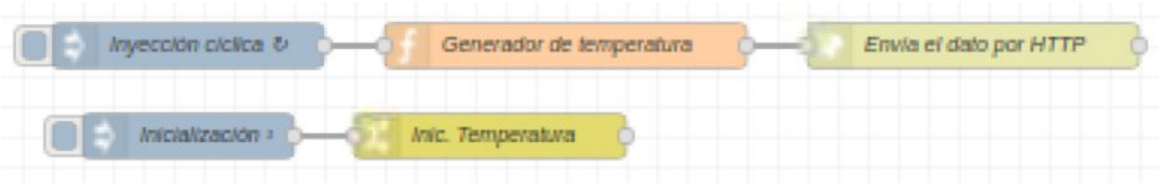


Así, se envían los datos en el último nodo a través del puerto serie /dev/ttyUSB0 con un baudaje de 57600. Si en la consola escribimos la línea de comandos 'cat -v /dev/ttyUSB0', podemos visualizar el puerto como si fuese un archivo, sobre el que se van escribiendo los datos, obteniendo:

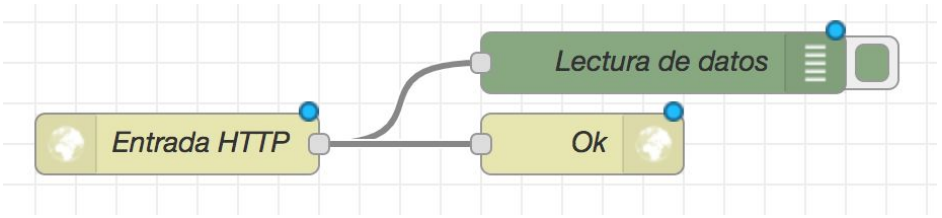
```
pi@raspberrypi:~ $ cat -v /dev/ttyUSB0
{"Temperatura":30.92588807251055}{"Temperatura":29.37910601947957}{"Temperatura"
:34.18988739685641}{"Temperatura":31.868849456587014}{"Temperatura":26.320182759
15034}{"Temperatura":38.14501778219215}{"Temperatura":26.535041453714165}{"Tempe
ratura":27.440363248179704}{"Temperatura":25.392190165578928}{"Temperatura":34.8
```

3. Práctica 2: Conexión HTTP PC <-> RPi

En esta práctica se pide crear una conexión HTTP entre la instancia Node-RED de la Raspberry Pi y una instancia nueva que debe crearse en el ordenador, de manera que se pueda enviar la temperatura al segundo en una trama HTTP y se pueda visualizar ésta misma en el sniffer Wireshark. Para ello, se ha implementado el siguiente flujo en el Node-RED de la Raspberry:



Y el siguiente en el ordenador:



En primer lugar, ahora en vez de un número aleatorio se pide que la temperatura sea un contador creciente. Además, se va añadir el dato de fecha y hora para poder seguir mejor los datos. El nodo de generación de temperatura en la RPi queda así:

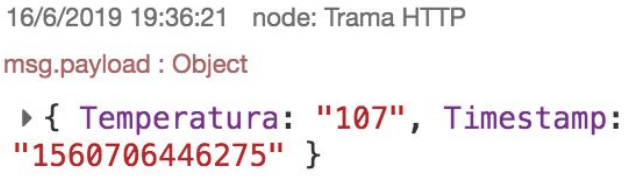
```
1  // Carga la variable de flujo Temperatura
2  var temp = flow.get('Temperatura');
3  // Aumenta el contador de la temperatura
4  temp += 0.5;
5  // Actualiza la variable de flujo
6  flow.set('Temperatura', temp);
7
8  // Guarda el dato en el mensaje
9  msg.payload = {Temperatura: temp,
10                Timestamp: msg.payload
11 };
12 return msg;
```

En el código puede verse que se ha creado la variable de flujo ‘Temperatura’, para poder ir aumentando el valor de la temperatura respecto al último. También se puede ver la rama de inicialización, en el que la variable ‘Temperatura’ es iniciada a 25ºC cada vez que arranca el flujo de este Node-RED.

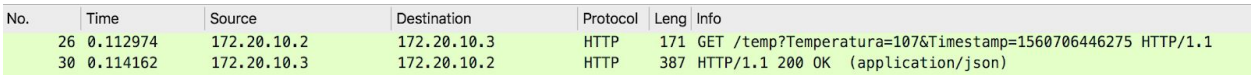
Finalmente, por el lado de la Raspberry Pi, se envían la temperatura y el timestamp mediante una petición HTTP con método GET a la dirección IP del ordenador, en el directorio ‘/temp’:

Una vez se ha enviado la trama, desde el ordenador se recibe por el nodo de entrada HTTP que comienza el flujo, configurado también con método GET:

El nodo con nombre 'Ok' envía de vuelta una respuesta HTTP a la Raspberry para verificar la recepción de los datos. Finalmente, en el nodo de debug podemos ver los datos recibidos:

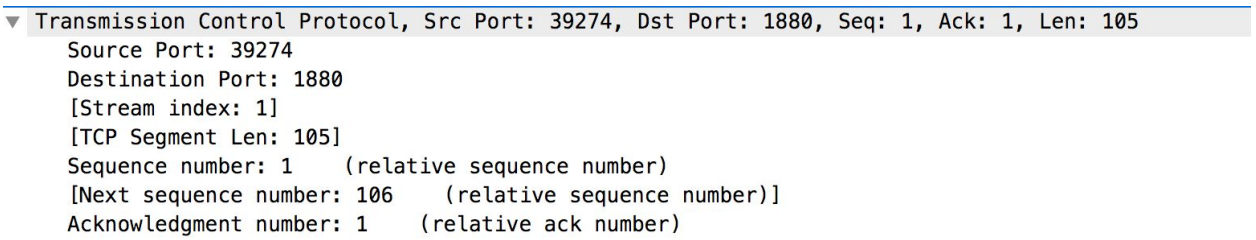


Con Wireshark se han capturado los paquetes enviados y recibidos, por lo que se puede buscar la trama concreta perteneciente a este envío. En esta captura se puede ver en primer lugar el paquete enviado con los datos de la Raspberry al ordenador, y después el status que se ha mandado de vuelta:



No.	Time	Source	Destination	Protocol	Leng	Info
26	0.112974	172.20.10.2	172.20.10.3	HTTP	171	GET /temp?Temperatura=107&Timestamp=1560706446275 HTTP/1.1
30	0.114162	172.20.10.3	172.20.10.2	HTTP	387	HTTP/1.1 200 OK (application/json)

Del primer paquete, el campo de Source indica la dirección IP de la Raspberry y el campo Destination la del PC. También podemos analizar la capa TCP de la trama enviada:



```
▼ Transmission Control Protocol, Src Port: 39274, Dst Port: 1880, Seq: 1, Ack: 1, Len: 105
    Source Port: 39274
    Destination Port: 1880
    [Stream index: 1]
    [TCP Segment Len: 105]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 106    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
```

Aquí el 'Source Port' es el puerto del PC por donde ha llegado la trama enviada desde la Raspberry, mientras que el 'Destination Port' es el puerto del PC al que se dirige la trama. En este caso, se ve que es el 1880, el mismo puerto donde está alojada la instancia de Node-RED.

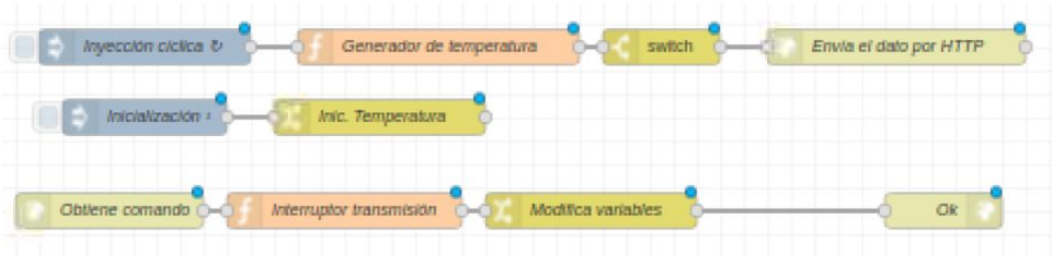
El 'Sequence Number' de esta trama vale 1, al igual que el 'Acknowledge Number', ya que se acaba de establecer la conexión TCP entre el cliente y el servidor y se ha enviado la flag de SYN. Si por el contrario se visualiza la trama HTTP enviada del PC a la Raspberry:

```
▼ Transmission Control Protocol, Src Port: 1880, Dst Port: 39274, Seq: 1, Ack: 106, Len: 321
    Source Port: 1880
    Destination Port: 39274
    [Stream index: 1]
    [TCP Segment Len: 321]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 322    (relative sequence number)]
    Acknowledgment number: 106    (relative ack number)
```

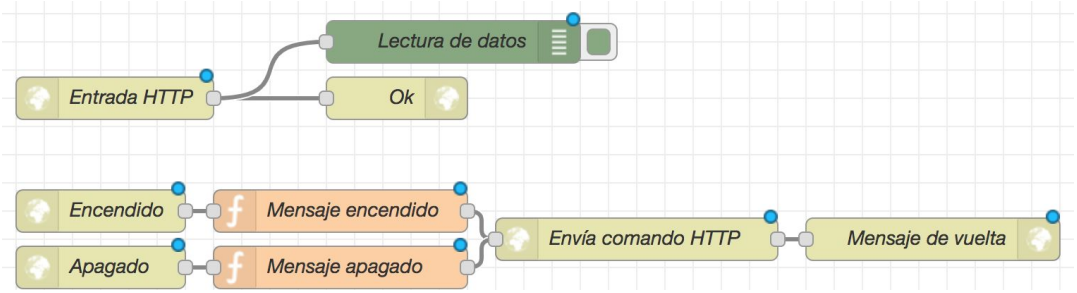
Aquí el 'Sequence number' sigue siendo 1 ya que el PC no había enviado todavía datos a la Raspberry, mientras que el 'Acknowledge Number' vale 106 ya que la Raspberry ya ha transmitido al PC un paquete de datos de longitud 105.

4. Práctica 3: Interruptor para el sensor

En esta práctica se pide simular un interruptor de encendido y apagado para la recepción de datos de temperatura, accionado mediante una trama HTTP enviada desde el ordenador. El flujo implementado en la Raspberry es:



Mientras que en el ordenador es:



En el flujo del ordenador se colocan dos nodos de entrada HTTP: 'Encendido' y 'Apagado'. Estos funcionan de manera que si ponemos en el navegador las direcciones localhost:1880/on o localhost:1880/off, estaremos encendiendo o apagando la recepción de datos en la Raspberry.

Los dos nodos de función almacenan en el payload del mensaje el status de 1 o 0 en función de si se ha encendido o apagado, y se envía mediante una petición HTTP a la Raspberry, en '172.20.10.2:1880/status'.

En el flujo de la Raspberry, con un nodo HTTP de entrada se recibe el comando enviado por el ordenador y se mete a una función que actúa como semáforo, haciendo las siguientes asignaciones en función del comando y el estado actual de la Raspberry:

```
// Carga la variable de flujo de estado y el comando recibido del ordenador
var status = flow.get('st');
var status_pay = msg.payload['Status'];

// Flujo para determinar el estado siguiente y devolver la respuesta
if (status=='1' & status_pay=='1'){
    msg.payload['Resp'] = 'Ya estaba encendido';
} else if (status=='1' & status_pay=='0'){
    msg.payload['Resp'] = 'Se ha apagado';
    msg.payload['Status'] = '0';
} else if (status=='0' & status_pay=='0'){
    msg.payload['Resp'] = 'Ya estaba apagado';
} else if (status=='0' & status_pay=='1'){
    msg.payload['Resp'] = 'Se ha encendido';
    msg.payload['Status'] = '1';
}
return msg;
```

Básicamente, modifica la variable 'Status' y almacena un mensaje que se va a mandar de vuelta al ordenador para indicar cómo ha resultado el accionamiento. Después este mensaje pasa a un nodo change, donde actualiza la variable de flujo 'St', que indica el estado de encendido o apagado con 1 o 0 en la raspberry y es inicializado a 1 al comienzo, y pasa el mensaje informativo al nodo de respuesta HTTP. Finalmente, en la parte superior del flujo se ha introducido un nodo de switch para que, si la variable de flujo vale 1, los mensajes se envíen a la estación base y si por el contrario vale 0, se corte la transmisión.

5. Práctica 4: Almacenamiento en SQLite

En esta práctica se pide almacenar los datos recibidos por la estación base en una base de datos SQLite local del ordenador. Para ello, en primer lugar, se ha instalado SQLite a través de la consola de comandos y se ha creado una base de datos con el comando 'sqlite3 IARS.db'.

Para esta práctica se va a aumentar el número de datos enviados en cada paquete de la Raspberry al ordenador, para enriquecer la estructura de información del sistema. Estas nuevas variables son las unidades físicas (ºC), el identificador de sensor (que se ha fijado a 31), un identificador único de lectura compuesto por el timestamp y el identificador de sensor, y la magnitud física, que en este caso es la temperatura:

```
msg.payload = {Temperature: temp,
    Timestamp: d,
    Units: 'ºC',
    Sensor_Id: sensor_id,
    reading_Id: msg.payload+sensor_id,
    magnitud: 'Temperatura'
};
```

De cara a alojar todos estos datos en la base de datos, se han creado tres tablas en IARS.db. En primer lugar está la tabla 'sensor_data', que tiene la siguiente forma con una consulta de ejemplo:

```
CREATE TABLE sensor_data (
reading_id INTEGER PRIMARY KEY,
timestamp TEXT NOT NULL,
value FLOAT NOT NULL,
units TEXT NOT NULL,
sensor_id INTEGER NOT NULL
);
```

```
sqlite> SELECT * FROM SENSOR_DATA LIMIT 3;
1560363470547|2019-06-12T18:17:50.516Z|36.0|ºC|31
1560363471563|2019-06-12T18:17:51.532Z|29.0|ºC|31
1560363472579|2019-06-12T18:17:52.548Z|38.0|ºC|31
```

Ésta es la tabla principal de la base de datos, la cual almacena los datos de lectura de sensor recibidos en la Raspberry. Las otras dos tablas son informativas y aportan datos sobre las magnitudes físicas y los sensores que forman la red. La primera se llama 'sensors_info' y sus queries de creación y consulta de ejemplo de las columnas son:

```
CREATE TABLE sensors_info (
sensor_id INTEGER PRIMARY KEY,
type TEXT NOT NULL,
latitude FLOAT NOT NULL,
longitude FLOAT NOT NULL,
description TEXT NOT NULL);
```

```
sqlite> SELECT * FROM SENSORS_INFO LIMIT 3;
31|Temperature sensor|37.375671|-6.001995|EPS - Aula 2.2 BIS
```

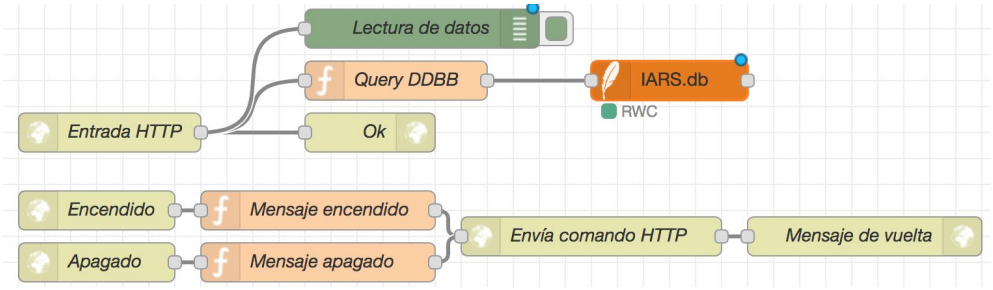
Esta tabla almacena para cada sensor su tipo ('Temperature Sensor' en este caso), sus coordenadas geográficas y una breve descripción (para este caso se ha escrito el emplazamiento donde se encuentra, 'EPS - Aula 2.2 BIS'. Finalmente, la tabla 'physical_properties' viene dada por su creación y la query 'SELECT' de ejemplo:

```
CREATE TABLE physical_properties (
property_id INTEGER PRIMARY KEY,
property TEXT NOT NULL,
units TEXT NOT NULL,
units_info TEXT NOT NULL
);
```

```
sqlite> SELECT * FROM physical_properties;
1|Temperature|ºC|Degree Celsius
```

Aquí se almacenan todas las magnitudes físicas leídas por sensores de la red, indicando su nombre, sus unidades (con lo que se puede hacer merge entre esta tabla y 'sensor_data') y el nombre de sus unidades.

De cara a la implementación en la instancia de Node-RED, el flujo de la Raspberry se mantiene igual y para el del ordenador se parte del mismo flujo de la práctica anterior pero añadiéndole el nodo 'node-red-node-sqlite' para comunicarse con bases de datos SQLite:



Al nodo de conexión de la base de datos se le inyecta una query definida en el nodo de función 'Query DDBB', la cual es:

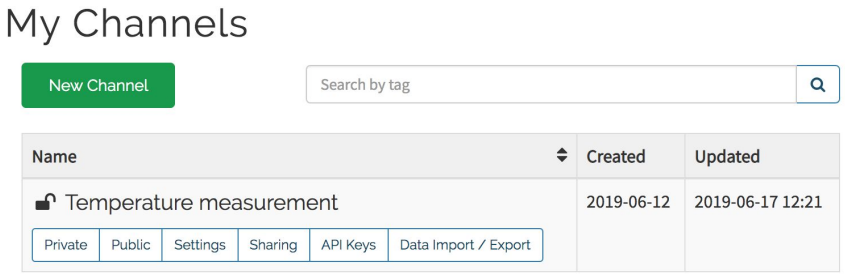
```
// Creación de query
var query = 'INSERT INTO sensor_data (reading_id,timestamp,value,units,sensor_id)'+
            'VALUES ('+msg.payload['reading_Id']+",'"+msg.payload['Timestamp']+"',"+
            msg.payload['Temperature']+",'"+msg.payload['Units']+"',"+
            msg.payload['Sensor_Id']+');'

// Guardado de query
msg.topic = query
return msg;

```

Como se puede ver, se trata de un comando de SQL 'INSERT INTO tabla (variables) VALUES (valores)' en el que se introducen los valores recibidos en el mensaje de entrada proveniente de la Raspberry.

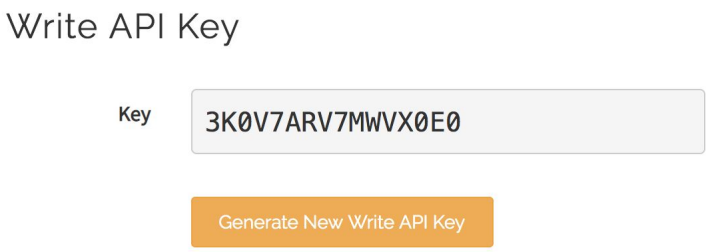
6. Práctica 5: Visualización en ThingSpeak

ThingSpeak es una aplicación de código abierto de Internet de las cosas y una API para almacenar y recuperar datos de cosas mediante el protocolo HTTP a través de Internet o a través de una red de área local. La aplicación pertenece a Mathworks, empresa desarrolladora de MATLAB, por lo que también tiene bastante integración con este último.

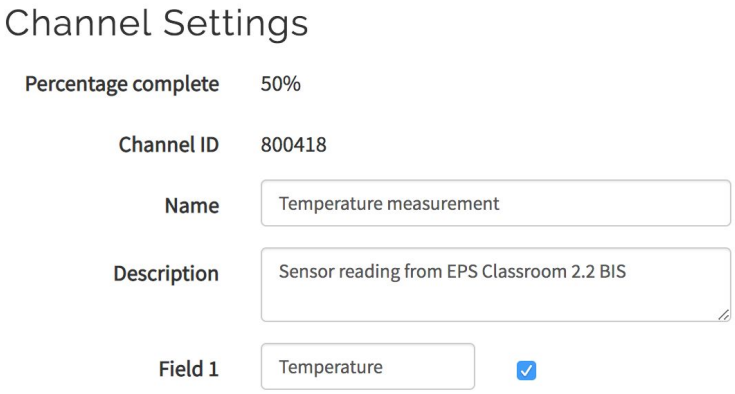
Una vez dentro de ThingSpeak y habiendo obtenido una cuenta de Mathworks, se crea un canal que va a funcionar a modo de dashboard:



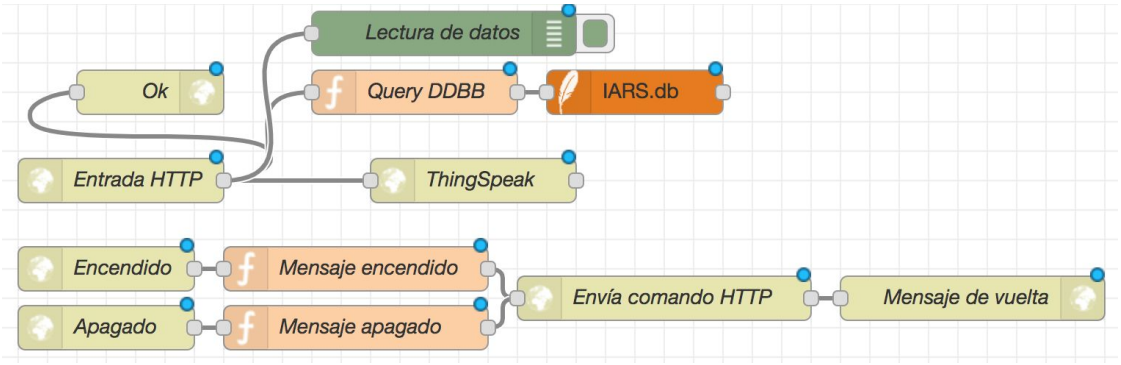
Éste se ha abierto al público para que los profesores puedan visualizarlo, en el siguiente enlace: https://thingspeak.com/channels/800418 . Una vez dentro del canal, hay que generar una API Key de escritura para poder mandar datos desde nuestro ordenador, por lo que en la pestaña de API Keys se genera una nueva:



Después se configura el canal, dándole un nombre, descripción y definiendo los campos de interés que se van a recibir y se quieren representar:



Además se han introducido la latitud y longitud del sensor. Una vez ya está todo configurado, basta con añadir un nodo de HTTP request al flujo de Node-RED del ordenador de la anterior práctica, quedando así:



El nodo HTTP se configura siguiendo las indicaciones de la propia plataforma de ThingSpeak, estableciendo el método GET apuntando a la siguiente dirección: 'https://api.thingspeak.com/update.blank?api_key=3K0V7ARV7MWVX0E0&field1={{payload.Temperature}}&created_at={{payload.Timestamp}}'. Como se puede ver, en el enlace hay que

introducir la API Key generada, el valor leído por el sensor que en este caso es la temperatura y se almacena en el campo 1, y un timestamp para poder utilizarlo en las visualizaciones.

Ahora que el flujo de Node-RED está operativo y el canal de ThingSpeak configurado, tenemos el siguiente dashboard donde podemos ver una gráfica de la recepción en tiempo real de los datos, el último valor recibido y su antigüedad y un mapa con la localización del sensor: