



TECNOLOGÍAS IOT HANDS- ON Y MVPS

MVP Para Depuradora

Beatriz Seoane
Domingo Alcalá
Jesús Tallon
30/04/2021

Contenido

- Ejercicio 1 4
 - 1.1 Peticiones de datos 4
 - 1.2 Recepción información..... 5
 - 1.3 Dashboard 6
- Ejercicio 2 12
 - 2.1. Node-Red..... 12
 - 2.2. Grafana..... 14
- Ejercicio 3 18
 - 3.1. Bases de diseño:..... 18
 - 3.1 Calibración del nivel cero en el tanque..... 19
 - 3.2 Servicio meteorológico Web 23
 - 3.2.2. Dashboard 24
 - 3.2.3. Flujo..... 25
 - 3.3 Seguridad..... 30
 - 3.4 Consideraciones sobre la precisión del sistema..... 31
- 4. Anexos 32
 - 4.1. Virtual commissioning..... 32
 - 4.1.1. Simulador 1 32
 - 4.1.2. Simulador 2 34
 - 4.2. Servicio de monitorización y alertas via telegram 34

Ejercicio 1

Implementad un programa en Node-RED que realice una petición MQTT a la placa IoT-02 de las lecturas en el formato JSON indicado, y las represente en un Dashboard de Node-RED que tenéis que crear.

1.1 Peticiones de datos

El programa utilizado se basa en la petición (pub) cada 5 segundos a la placa IoT de la información asociada a la boya mediante el protocolo MQTT. Para ello utilizaremos dos nodos:

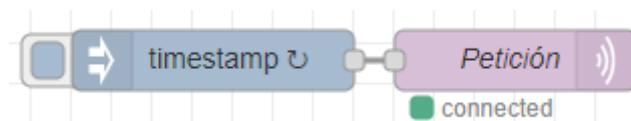


Ilustración 1. Nodos utilizados en la petición de datos

- Un nodo **“inject”** configurado como timestamp que cada 5 segundos nos generara el evento para realizar la petición de datos.

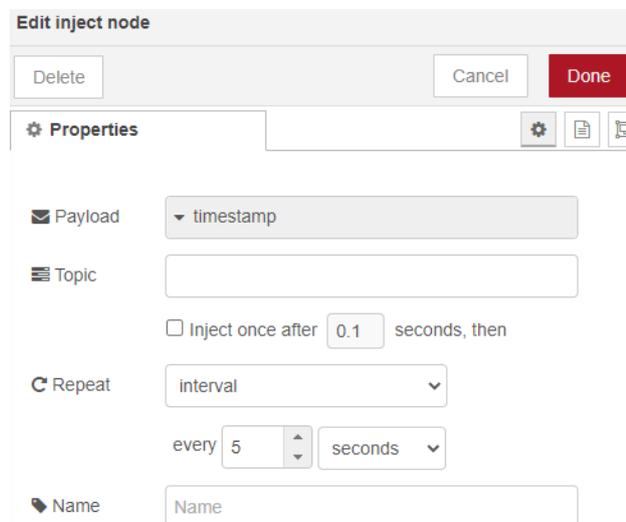


Ilustración 2. Configuración peticiones de información

- Un nodo **“MQTT out”** para el envío de la petición cada vez que nos genera un evento el nodo **“inject”**.

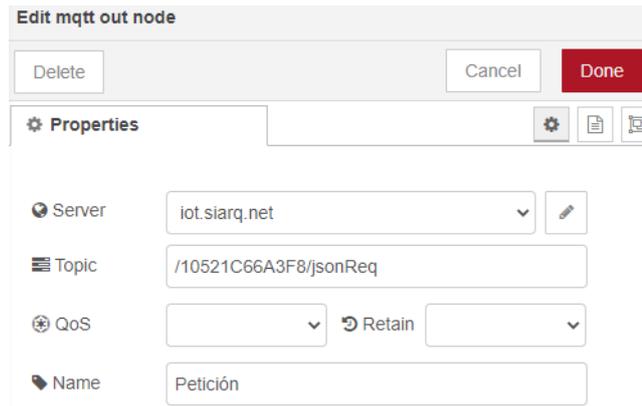


Ilustración 3. Publicación Topic MQTT peticiones datos

Podemos ver a continuación la configuración del browser MQTT al que nos conectamos para realizar el pub/sub con la placa IoT:

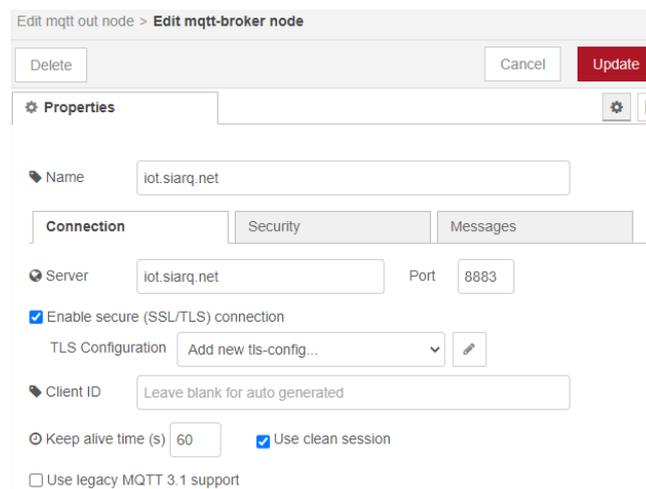


Ilustración 4. Configuración para la conexión con el browser MQTT

1.2 Recepción información

El funcionamiento de la recepción de la información se basa en la utilización de un nodo **“Mqtt Out”** mediante el cual nos suscribimos al topic del browser MQTT mediante la placa envía la información procesada. Los nodos utilizados para realizar esta función del programa son dos:

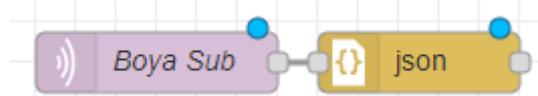


Ilustración 5. Nodos recepción datos

- Un nodo **“MQTT Out”** mediante el cual recibimos la información.

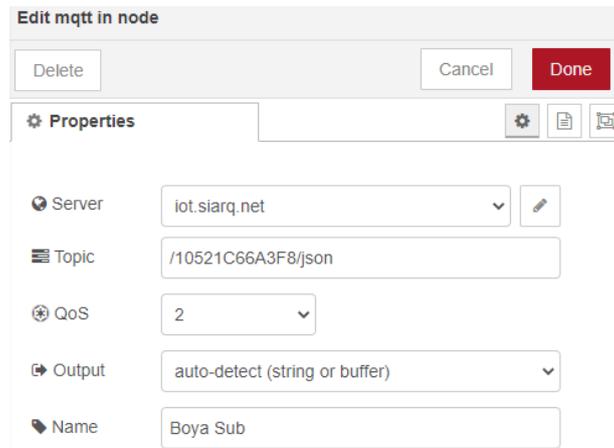


Ilustración 6. Configuración del nodo

- Un nodo del tipo **“JSON”** que nos permitirá con posterioridad procesar de forma cómoda para su traspaso al dashboard o a la base de datos influxdb.

1.3 Dashboard

El cuadro de mandos en tiempo real se compone de diversos elementos concretamente:

- 1) Imagen de la boya para tener una representación visible del elemento monitorizado. En este caso se ha cargado una librería complementaria a la básica de dashboard (node-red-contrib-ui-media v1.2.0).



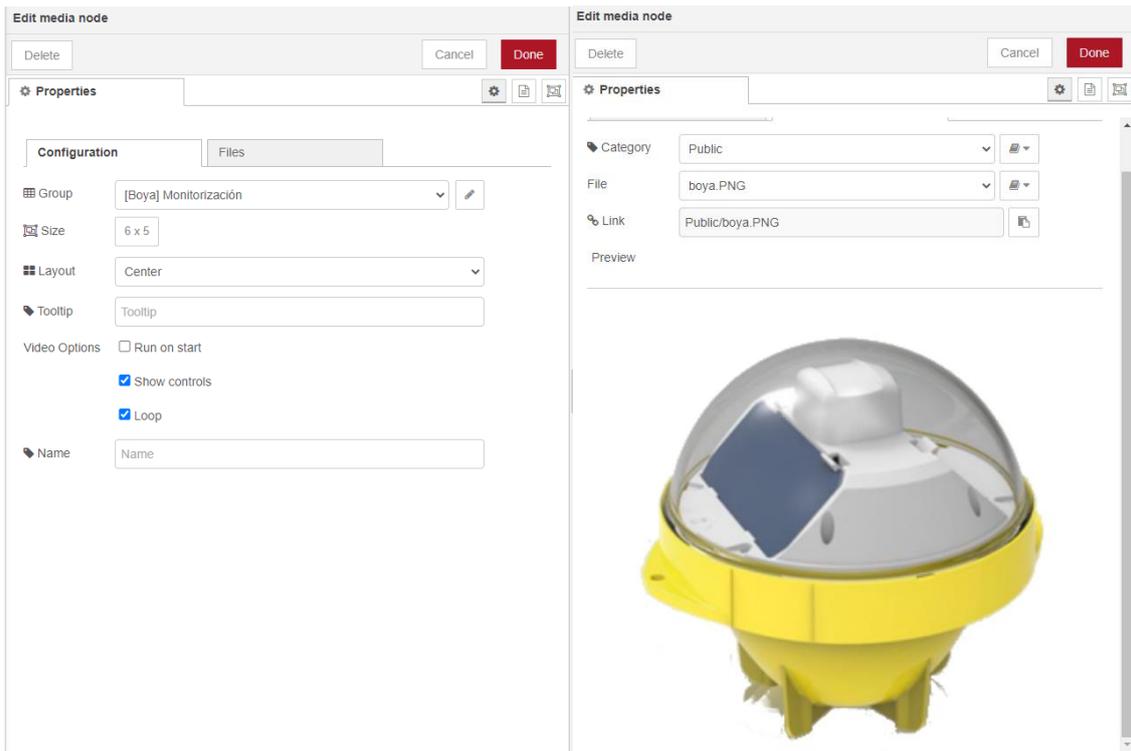
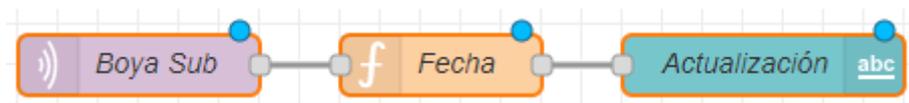


Ilustración 7. Configuración del objeto Media

- 2) Fecha y hora en la que se ha realizado la última actualización de los datos. Cada vez que se recibe la llegada de la información enviada por la placa obtenemos la información de tiempo el sistema y se representa en el dashboard en un nodo de tipo **“text in”**.



```

1 fecha = new Date();
2 msg.payload=fecha.getDate()+'/'+fecha.getMonth()+'/'+fecha.getFullYear();
3 msg.payload= msg.payload + '+'+fecha.getHours()+':'+'+fecha.getMinutes()+':'+'+fecha.getSeconds();
4 flow.set('Act',msg.payload);
5 return msg;

```

Ilustración 8. Código de la función

- 3) Switch que nos permite indicar cuando queremos almacenar la información en nuestra base de datos influxdb, permitiendo de esta manera no estar siempre registrando información.

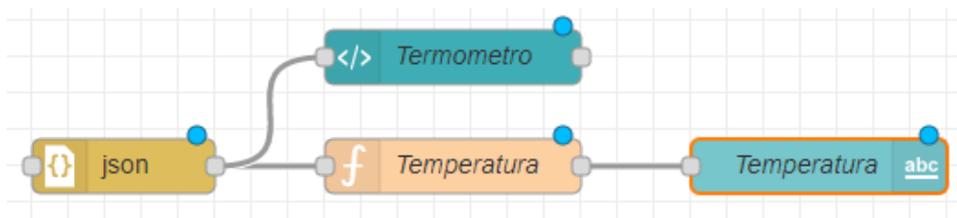


```
1 flow.set('Guardar',msg.payload);
2 return msg;
```

Ilustración 9. Gestión del modo guardado en base de datos

Utilizamos una variable a nivel de flujo para saber cuándo recibimos la información de la placa si queremos guardarla en la base de datos o no.

- 4) Datos de la temperatura actual representada tanto de forma textual como por un gauge en formato de termómetro realizado en HTML 5.0 mediante un template.



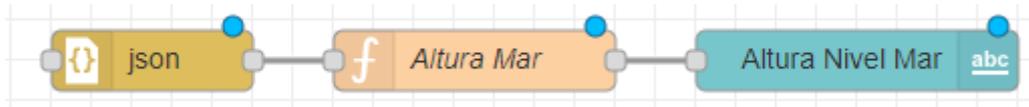
```
1 flow.set('T',msg.payload.T);
2 msg.payload=msg.payload.T.toFixed(2);
3 return msg;
```

Ilustración 10. Función temperatura

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     #contenedor {
6       height: 20px;
7       width: 300px;
8       background-color:LightGray;
9       vertical-align:middle;
10  }
11  meter {
12    height: 15px;
13    width: 200px;
14  }
15  </style>
16 </head>
17 <body>
18 <center>Temp.(°C)</center>
19 <div id="contenedor">
20   -30°C
21 <meter id="thermometer" min="-30" max="50" low="0" high="20" optimum="10" value="{{msg.payload.T}}"></meter>
22   50°C
23 </div>
24 </body>
25 </html>
```

Ilustración 11. Código HTML del template que representa el termómetro

- 5) Información de la altura a nivel del mar donde se encuentra ubicada la placa (Text in).



```

1 flow.set('A',msg.payload.A);
2 msg.payload=msg.payload.A.toFixed(2);
3 return msg;

```

La función simplemente accede al campo correspondiente del json recibido y guarda la información en una variable a nivel de flujo.

- 6) Indicador del porcentaje de humedad relativa (type Gauge).



```

1 flow.set('RH',msg.payload.RH);
2 msg.payload=msg.payload.RH.toFixed(2);
3 return msg;

```

La función simplemente accede al campo correspondiente del json recibido y guarda la información en una variable a nivel de flujo.

- 7) Indicador del porcentaje de intensidad lumínica (type Donut).



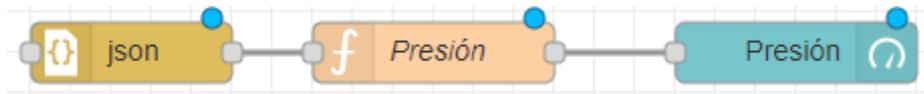
```

1 flow.set('L',msg.payload.L);
2 msg.payload=(msg.payload.L/4095.0)*100.0;
3 msg.payload=msg.payload.toFixed(2);
4 return msg;

```

La función accede al campo correspondiente del json recibido y guarda la información en una variable a nivel de flujo.

8) Indicador de la presión atmosférica en mBar (Type Gauge).



```
1 flow.set('P',msg.payload.P);  
2 msg.payload=msg.payload.P.toFixed(2);  
3 return msg;
```

La función accede al campo correspondiente del json recibido y guarda la información en una variable a nivel de flujo.

9) Indicador del nivel del depósito monitorizado en metros (Type Level).



```
1 flow.set('AR',msg.payload.AR);  
2 msg.payload=msg.payload.AR.toFixed(2);  
3 return msg;
```

La función accede al campo correspondiente del json recibido y guarda la información en una variable a nivel de flujo.

A continuación vemos el esquema general del NodeRed:

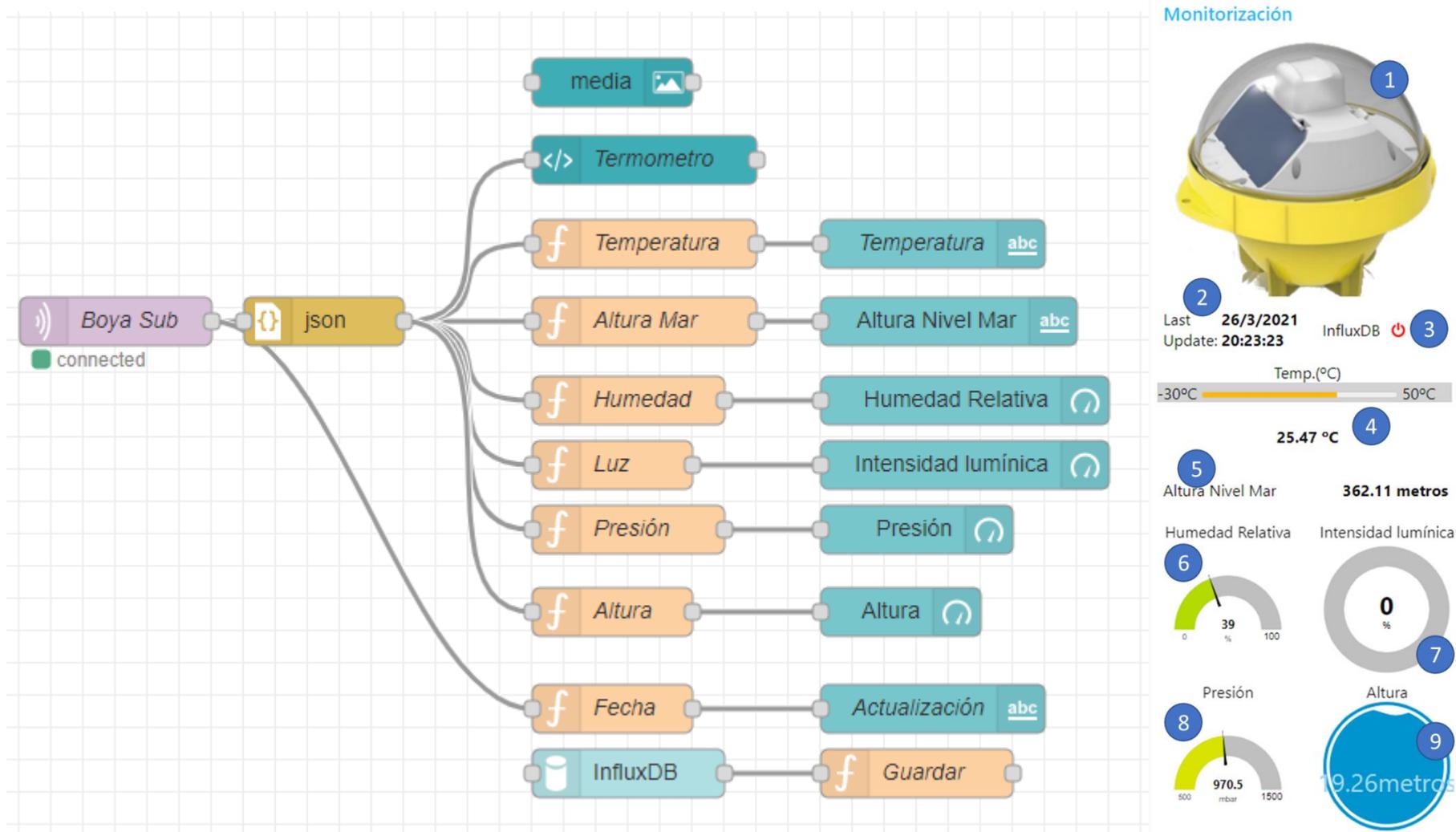


Ilustración 12. Esquema programa Node-Red base (Recepción datos y Dashboard)

Ejercicio 2

2.1. Node-Red

Implementar un proceso que se suscriba al *topic* de publicación de la IoT-02 de forma que las lecturas se almacenen en una base de datos InfluxDB. Para ello puede utilizarse como lenguaje de programación Node-RED.

En cuanto al InfluxDB, deberéis anteponer el prefijo “m” + xy + “_” al nombre del measurement (equivalente a una tabla) de la base de datos.

La obtención de la información de la placa ya se ha comentado con anterioridad en el primero de los ejercicios, nos centraremos aquí en explicar los nodos asociados al almacenamiento en la base de datos.



Ilustración 13. Nodos asociados al almacenamiento en la base de datos

- 1) Una vez recibido el JSON lo primero que nos preguntamos es si queremos o no almacenar la información en la base de datos. Para ello hemos de recordar que se había añadido un switch en el dashboard para indicarlo y se tiene una variable a nivel de flujo que nos indica dicha opción. Mediante un nodo de tipo “función” recuperamos el estado en el que se encuentra el switch.

```

1 var guardar = flow.get('Guardar');
2 if (guardar===false) {msg.payload='false';}
3 else {msg.payload=msg.payload;}
4 return msg;

```

- 2) Utilizamos un nodo de tipo “switch” para consultar el valor del estado del switch y tomar la decisión de guardarlo o no en base al mismo.

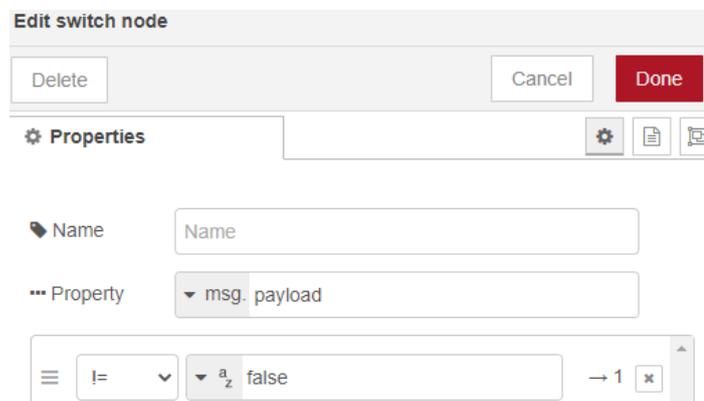


Ilustración 14. Configuración del nodo Switch

- 3) En caso de que la opción de guardado este activa procedemos a guardar el json en la base de datos influxdb mediante un nodo de tipo **“influxdb out”**.

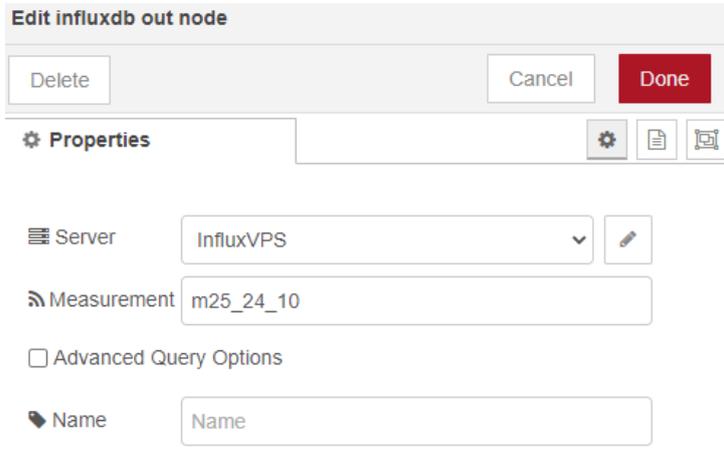


Ilustración 15. Configuración nodo

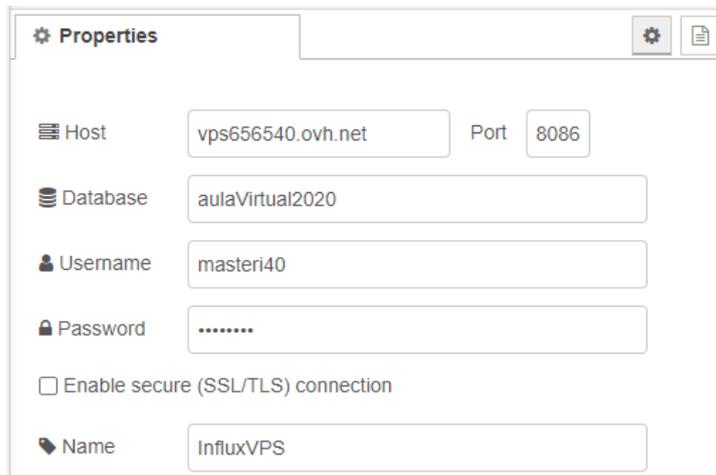


Ilustración 16. Configuración a la BD

En caso de querer ver el programa en más detalle, se adjunta el link de acceso y usuario y contraseña:

<http://vps656540.ovh.net:25880/#flow/3d23d7ec.863448>

Usuario: admin

Pass: admin

2.2. Grafana

Representar en el sistema Grafana las lecturas de presión, temperatura, humedad relativa y nivel de luz almacenadas en InfluxDB.

Destacar que, al realizar el Panel de Grafana, ya se tenía preparadas las herramientas de calibración correspondientes al ejercicio 3 y que se detallarán en este.

Para la implementación de las lecturas de datos de la placa desde Grafana se ha utilizado el topic M25_24_10, siguiendo las codificaciones de los números asignados a los miembros del máster.

Se ha partido la pantalla de Grafana en 3 zonas, para poder apreciar todos los datos correctamente.

Panel general:



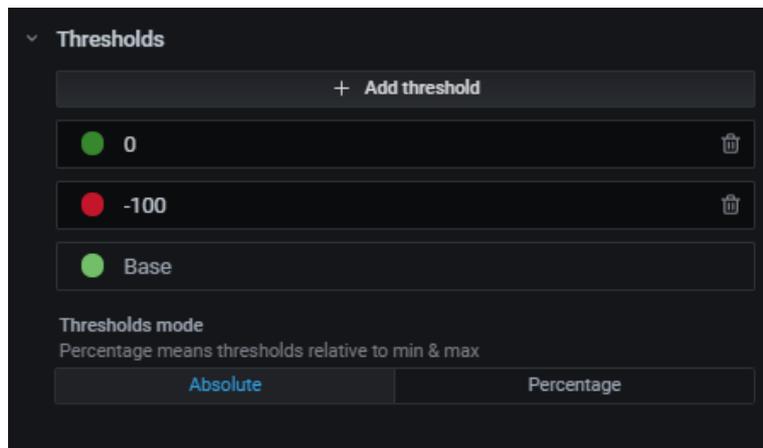
Lectura de nivel del depósito

Entendiendo la utilidad de la placa, el primer valor a tener en cuenta es el nivel del depósito en el que se encuentra, por ello es el primer nivel y el más visible dentro del panel.



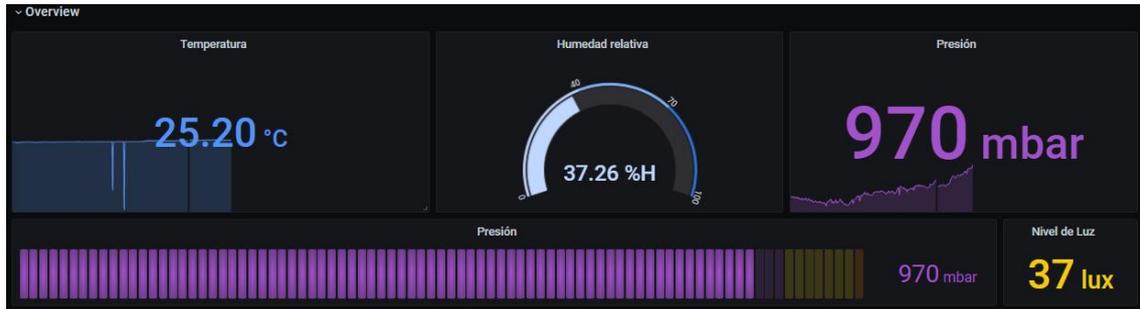
En este caso, se observa el valor actual del nivel del depósito respecto al 0 calibrado y, la línea inferior, muestra la evolución de esta medida.

Se ha añadido un límite en caso de observar valores negativos. Si se diera este caso, el panel se apreciaría en rojo, ya que seguramente se trate de un error de calibración inicial de la placa que compromete todas las mediciones posteriores, por lo que es recomendable visualizarlo y solucionarlo cuanto antes.



Overview

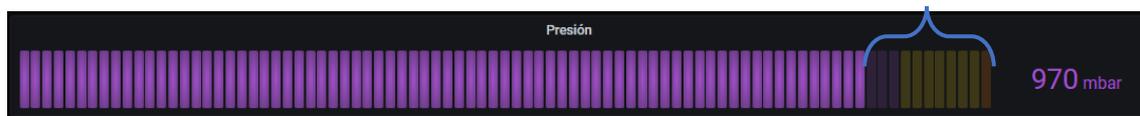
Nos ofrece un vistazo rápido del estado actual de los principales valores de la placa:



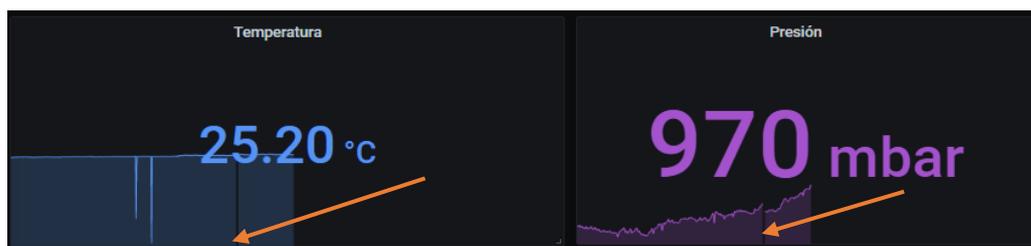
En esta parte del panel, podemos observar la temperatura y su evolución, la humedad relativa y su proximidad al 100% según la gama de colores, la presión y su evolución y el nivel de luz.

Destacamos la barra medidora de presión, ya que los valores que se aprecian destacados en otro color son imposibles o altamente improbables, por lo que nos permite controlar de forma fácil si el sensor funciona correctamente.

Zona de control



La visualización de la evolución de la temperatura y la presión también ayudan a detectar fallos de conexión con la placa, ya que estos aparecen como un vacío en el gráfico.



Evolución

Esta es la zona del panel más extensa, pensada para ser consultada en caso de necesitar una visión más global de lo sucedido en la placa durante un periodo de tiempo determinado. En esta área podemos encontrar los gráficos más detallados relacionados con el nivel del depósito, la altura sobre el nivel del mar, la evolución de la temperatura, la evolución del nivel de luz, a la evolución de la humedad relativa y la evolución del nivel de presión.



Para las diversas gráficas también podemos apreciar el valor máximo, mínimo y la media de los valores representados.

En caso de querer ver más en detalle todas las opciones implementadas en grafana, se puede acceder con el siguiente link: <https://acortar.link/XEq7f>

Ejercicio 3

Se pide que incorporéis en el dashboard del ejercicio 1 la visualización de la altura de la placa IoT-02 respecto al suelo en tiempo real. Valorad la precisión del sistema y proponed un sistema de redondeo para aplicar en depósitos de depuradora.

3.1. Bases de diseño:

La propuesta que hemos desarrollado acomete la solución al problema planteado apuntalándose en tres iniciativas:

- Posibilidad de calibración del dispositivo (ajuste del nivel 0 en el tanque) a través de una acción directa sobre la placa: mantener pulsado i35 durante más de tres segundos. El valor obtenido se almacenará en memoria remanente EEPROM y será válido hasta una nueva acción de calibración.
- Cálculo del nivel del tranque como diferencia del valor actual de altura absoluta y el valor de “nivel 0” almacenado en el momento de la calibración. Para evitar pequeñas oscilaciones, se establece un filtro exponencial.
- Corrección periódica de la presión atmosférica obteniendo el valor actualizado de alguna página web meteorológica. NodeRed enviará ese valor periódicamente a la placa, en su ausencia, la placa utilizará el valor establecido por defecto.

Lógicamente, estas incorporaciones implican algunos cambios en el firmware proporcionado, a continuación se detallan los más significativos.



Indicar adicionalmente que el firmware proporcionado ocasionaba un problema con la visualización de caracteres en la pantalla. Se ha localizado ese problema en la naturaleza dual core del microcontrolador ESP32. Por defecto, el compilador asocia las tareas de forma alternativa a uno u otro core, ello posiblemente crea algún tipo de incompatibilidad con la librería de la pantalla OLED (algunas de estas librerías proceden de arduino tecnología AVR y no están depuradas para procesadores con más de un núcleo). Finalmente se ha resuelto el problema forzando la asociación de las tareas al core 1.

3.1 Calibración del nivel cero en el tanque.

Como se introdujo en 3.1., la finalidad de esta modificación es incorporar la funcionalidad de supervisión del botón i35, desencadenando la calibración del punto cero si el pulsador se acciona por más de 3 segundos.

Para clarificar las acciones a realizar en función del estado del pulsador se ha creado una máquina de estados para cada situación posible:

```
typedef enum {pbRELEASED, pbPRESSED, pbHOLD, pbWAITRELEASE} pbSTATES_type;
```

Una vez detectada la orden de calibración, esta se pasa a la tarea encargada de realizar las lecturas mediante una cola:

```
xQueueSend(calibrationQueue, &doCalibration, ( TickType_t ) 10);
```

A continuación se muestra el código completo de la nueva tarea creada para la supervisión del botón i35 y gestión de la acción de calibración:

```
/**
 * chequeo periodico del boton i35 para atender a la orden de calibrar
 */

void vUserButton(void *parameter) {
    typedef enum {pbRELEASED, pbPRESSED, pbHOLD, pbWAITRELEASE} pbSTATES_type;
    static pbSTATES_type stateMachine = pbRELEASED;
    static long pushTimeStamp;
    static boolean doCalibration = true;
    float tankLevel = 0.0;
    char tankLevelStr[10];

    for (;;) {

        // check if there is new tank level value in queue
        xQueueReceive(tankLevelQueue, &tankLevel, ( TickType_t ) 10);

        if (stateMachine == pbRELEASED || stateMachine == pbPRESSED) {
            if (WiFi.status() == WL_CONNECTED) {
                vScreen16pixelText(0,0, "conectado a:\n" + WiFi.SSID() + "\n> nivel:"+(String)((int)(tankLevel *
100))+ " cm");
            } else {
                vScreen16pixelText(0,0, "conectando...");
            }
        }
    }
}
```

```

switch (stateMachine) {
  case (pbRELEASED) : if (bPressedButton(BT_I35)) {
    pushTimeStamp = millis();
    stateMachine = pbPRESSED;
  } break;

  case (pbPRESSED) : if (bPressedButton(BT_I35)) {
    if (millis() - pushTimeStamp > 3000) {
      stateMachine = pbHOLD;
    }
  } else {
    stateMachine = pbRELEASED;
  } break;
  case (pbHOLD) : {
    vScreen16pixelText(0,0,"calibrar nivel");
    xQueueSend(calibrationQueue, &doCalibration, ( TickType_t ) 10);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    vScreen16pixelText(0,0,"ok!");
    stateMachine = pbWAITRELEASE;
  } break;

  case (pbWAITRELEASE) : if (!bPressedButton(BT_I35)) {
    stateMachine = (pbRELEASED);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  } break;
}
vTaskDelay(10); // To avoid: Task watchdog got triggered. The following tasks did not reset the
watchdog in time
}
vTaskDelete(NULL); // There is an infinite loop before. This line will never be reached.
}

```

La función encargada de realizar la lectura del sensor y composición de la cadena JSON es la receptora de la orden de calibración a través de la cola. En el momento de recibir la orden, se almacena el valor actual de altitud en la variable fAref y se graba en memoria remanente EEPROM.

```

if (xQueueReceive(calibrationQueue, &doCalibrateAlt, ( TickType_t ) 10) == pdPASS) {
  if (doCalibrateAlt) {
    fAref = ((float)nAx100) / 100;
    byte fA_MSB = nAx100 >> 8;
    byte fA_LSB = nAx100 & 0xFF;
    EEPROM.write(0, fA_MSB);
    EEPROM.write(1, fA_LSB);
    EEPROM.commit();
    fTankLevel = 0;
    fTankLevelOld = 0;
    Serial.print("..... calibrando h0 = ");
    Serial.println(fAref);
    doCalibrateAlt = false;
  }
}
}

```

Para permitir la compensación de la presión atmosférica y reducir así el error, se obtiene el valor real periódicamente desde node-red (ver detalles en 3.2.). Se han adaptado algunas de las funciones para que sea posible pasar ese parámetro a la función de lectura del sensor:

```
vReadingBME280(&nTx100, &nPx100, &nRHx100, &nGr, &nAx100, seaLevelPress);
```

Se ha creado la variable booleana **firstScan** para detectar un reset en la placa, en caso afirmativo se recupera el valor de nivel tanque cero previamente depositado en la memoria remanente EEPROM. Se muestra a continuación el cálculo de la altura, aplicación del filtro exponencial y recuperación del valor cero almacenado cuando procede:

```
// after reset, take reference level from permanent EEPROM memory
if (firstScan) {
  firstScan = false;
  int fArefx100 = (EEPROM.read(0) << 8) + EEPROM.read(1);
  fAref = (float)fArefx100 / 100;
  Serial.print(".....recuperado h0 = ");
  Serial.println(fAref);
}

// tank level = relative altitude from calibrated value
fTankLevel = fA - fAref;

// cutoff to prevent negative values
if (fTankLevel < 0) {
  fTankLevel = 0;
}

// apply smoother. If first scan, previous measurement is not available
if (!firstScan) {
  fTankLevel = fTankLevel * alfa + (1 - alfa) * fTankLevelOld;
  fTankLevelOld = fTankLevel;
}

// add tank level to json structure
json_IoT02["AR"] = fTankLevel;

// send tank level to screen visualization task
xQueueSend(tankLevelQueue, &fTankLevel, ( TickType_t ) 10);
```

Finalmente, mostramos la creación de la nueva tarea freeRTOS en la función setup() y la corrección con respecto a la asignación de cores comentada con anterioridad:

```
#ifdef USING_WIFI

xTaskCreatePinnedToCore(
    vConnectingMqttTask, /* Task function. */
    "Connecting MQTTS Task", /* name of task. */
    //10000, /* Stack size of task */
    8192, /* Stack size of task */
    NULL, /* parameter of the task */
    2, /* priority of the task */
    NULL,
    1); /* use core 1 */

#endif /* USING_WIFI */

xTaskCreatePinnedToCore(
    vUserButton, // Task function.
    "check button i35 to calibrate", // name of task.
    //10000, // Stack size of task
    8192, // Stack size of task
    NULL, // parameter of the task
    1, // priority of the task
    NULL,
    1);
}
```

3.2 Servicio meteorológico Web

Anteriormente en el primer ejercicio ya se habían añadido los componentes necesarios para poder visualizar en tiempo real tanto la altura asociada al propio depósito como la altura relativa del mismo sobre el nivel del mar.

Para intentar hacer lo más preciso posible el sistema se ha desarrollado a nivel de Node-Red el acceso en tiempo real a un servicio web meteorológico que nos permite conocer la presión atmosférica en mbar en cualquier ciudad del mundo:

<https://openweathermap.org/api>

Para poder utilizar dicho servicio hay que suscribirse lo que permite obtener una Key que para acceder a la consulta de los datos meteorológicos. Con dicha Key, el nombre de la ciudad y el país ya es posible acceder a dicha información.

Key: 5296b2ae0aa5c40e1a95397c09d4af09

A continuación, podemos ver las llamadas a la API proporcionada:

API call

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code},{country code}&appid={API key}
```

Ilustración 17. Llamadas a la API

Esto permite que desde Node-Red podamos mediante una petición http acceder a los datos en cualquier momento. Para ello utilizaremos un nodo de tipo “HTTP Request” configurado de la siguiente manera:

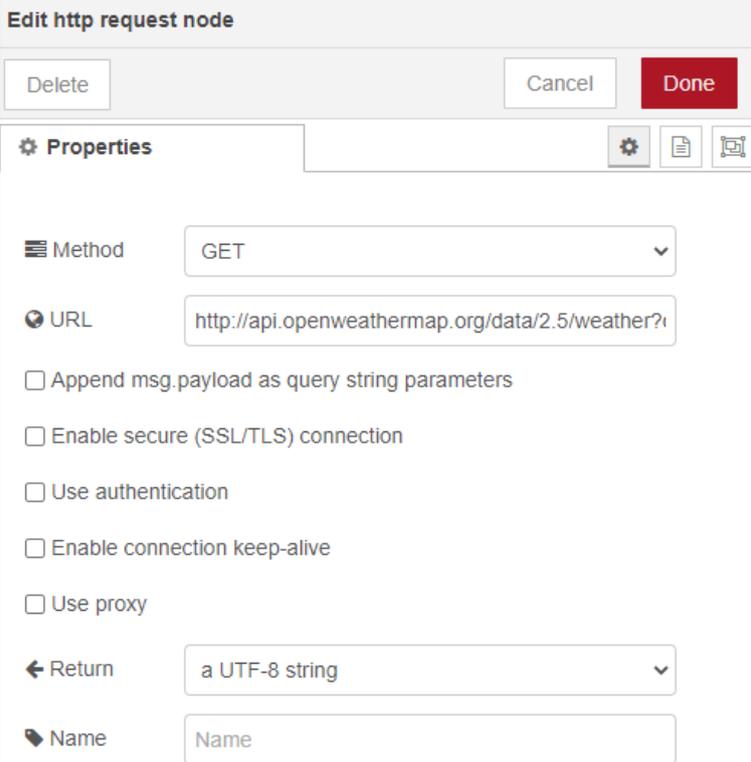


Ilustración 18. Configuración del node HTTP Request

URL: utilizada:

`http://api.openweathermap.org/data/2.5/weather?q={{payload}}&APPID=5296b2ae0aa5c40e1a95397c09d4af09`

Donde payload = "Ciudad, País" → ejemplo: "Barcelona,ES"

La información recibida es un JSON que contiene:

- Latitud y longitud (coordenadas) de la estación meteorológica de la ciudad indicada.
- Información sobre temperatura, presión, humedad, visibilidad, viento y nubosidad.
- Información sobre la salida y puesta del sol.

3.2.2. Dashboard

Una vez ya se sabe cómo obtener dicha información se toma la decisión de ampliar el dashboard para gestionar la consulta de esta y su traspaso a la placa IoT. Se añade al dashboard dos entradas de texto para introducir la ciudad y el país que son parámetros necesarios para pedir la información de la presión atmosférica y un botón que nos permite lanzar la llamada al servicio meteorológico como podemos ver en la siguiente captura de pantalla:

Datos meteorológicos

Ciudad
Igualada

País
ES

ACTUALIZAR



Lat.: **41.581** Lon.: **1.6172**

Presión Atmosférica **1005 mBar**

26/4/2021 21:53:44 node: 4bf713ff.421e0c

msg.payload : Object

- ▼ object
 - ▼ coord: object
 - lon: 2.159
 - lat: 41.3888
 - ▼ weather: array[1]
 - ▶ 0: object
 - base: "stations"
 - ▼ main: object
 - temp: 286.97
 - feels_like: 286.55
 - temp_min: 285.37
 - temp_max: 288.15
 - pressure: 1006
 - humidity: 82
 - visibility: 10000
 - ▼ wind: object
 - speed: 5.14
 - deg: 320
 - ▼ clouds: object
 - all: 75
 - dt: 1619466626
 - ▼ sys: object
 - type: 1
 - id: 6398
 - country: "ES"
 - sunrise: 1619412910
 - sunset: 1619462570

Ilustración 19. Configuración datos meteorológicos y json recibido

Con la información proporcionada por el servicio mostramos en un mapa la posición de las coordenadas devueltas y el dato de la presión atmosférica.

3.2.3. Flujo

- 1) Para gestionar la información recibida del servicio web definiremos dos variables a nivel de flujo denominadas ciudad y país. Las inicializaremos utilizando como ciudad "Barcelona" y país "ES".



```
1 flow.set('Ciudad',msg.payload);
2 return msg;
```

```
1 flow.set('País',msg.payload);
2 return msg;
```

Ilustración 20. Inicialización de la ciudad y el país

Pondremos estos valores también en los textboxes existentes en el dashboard.

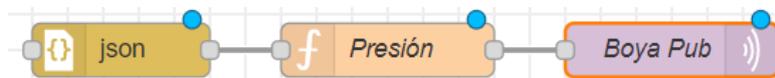
- 2) Cada vez que pulsamos el botón actualizar leemos las dos variables de tipo flujo y pasamos los parámetros necesarios para realizar la petición http al servicio meteorológico tal y como hemos visto con anterioridad. Recibimos el json resultante de la llamada.



```
1 var ciudad = flow.get('Ciudad');
2 var pais = flow.get('País');
3 msg.payload=ciudad+', '+pais;
4 return msg;
```

Ilustración 21. Petición HTTP

- 3) Una vez obtenido el json lo descomponemos para acceder a la información referente a la presión atmosférica y la publicamos vía MQTT sobre el topic que se ha definido para traspasarle la información a la placa.



```
1 msg.payload=msg.payload.main.pressure;
2 return msg;
```

Ilustración 22. Función de tratamiento del json

Edit mqtt out node

Delete Cancel Done

⚙️ Properties

🌐 Server: iot.siarq.net

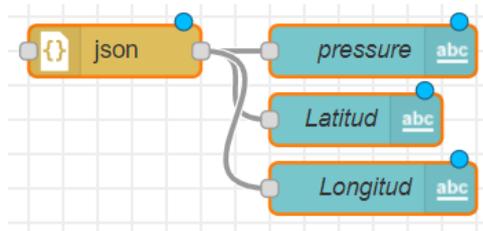
📄 Topic: /10521C66A3F8/setSeaP

🛡️ QoS: [] Retain: []

🏷️ Name: Boya Pub

Ilustración 23. Configuración de la publicación topic MQTT

- 4) También procedemos a obtener los campos de coordenadas y presión atmosférica para actualizarlos en el Dashboard en los objetos textbox correspondientes.



Edit text node

Delete Cancel Done

⚙️ Properties

📄 Group: [Boya] Datos meteorológicos

📏 Size: auto

🏷️ Label: Presión Atmosférica

📄 Value format: {{msg.payload.main.pressure}} mBar

📄 Layout: [label value] [label value] [label value]

🏷️ Name: pressure

Ilustración 24. Representación de los campos de información en el dashboard

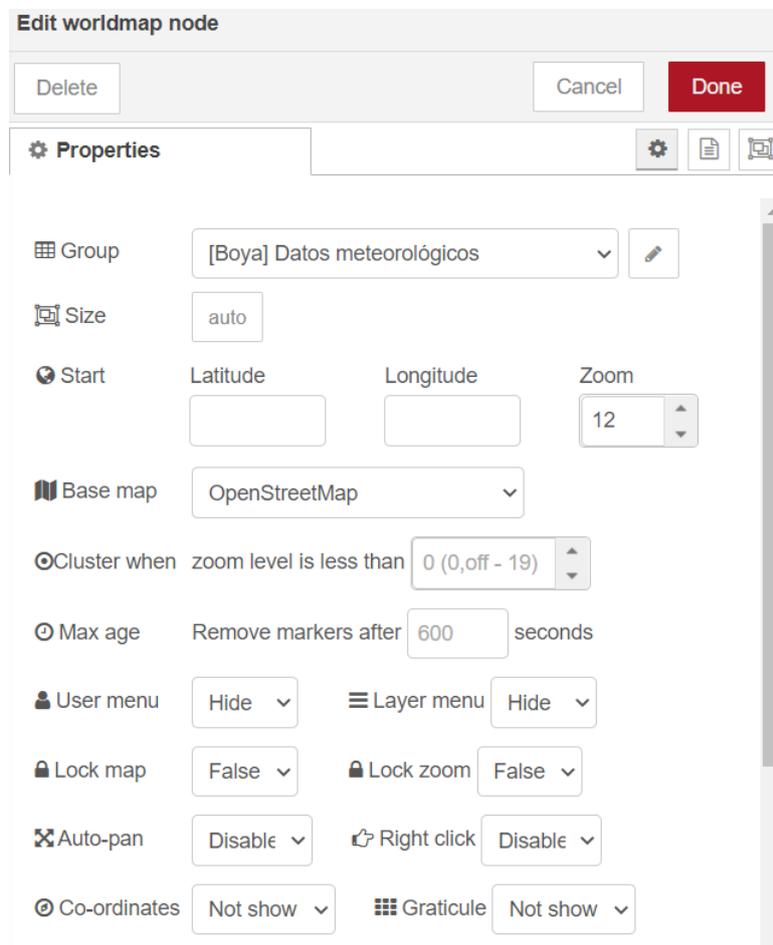
- 5) Aprovechando que se no devuelven también las coordenadas procedemos a utilizar un objeto para mostrar dicha ubicación sobre un mapa similar a los utilizados por Google maps.



```
1 msg.payload = msg.payload = { "command": {"lat":msg.payload.coord.lat, "lon":msg.payload.coord.lon, "zoom":10}};
2 return msg;
```

Ilustración 25. Flujo Representación de la localización.

Para poder realizarlo hemos utilizado la librería `node-red-contrib-web-worldmap`. La información con la que se alimenta este nodo es un json en el que se indica la latitud, longitud y aunque no es obligatorio diferentes parámetros de visualización. A continuación, podemos ver los parámetros de configuración del nodo en concreto.



Edit worldmap node

Delete Cancel Done

Properties

Group: [Boya] Datos meteorológicos

Size: auto

Start: Latitude, Longitude, Zoom: 12

Base map: OpenStreetMap

Cluster when zoom level is less than: 0 (0,off - 19)

Max age: Remove markers after 600 seconds

User menu: Hide, Layer menu: Hide

Lock map: False, Lock zoom: False

Auto-pan: Disable, Right click: Disable

Co-ordinates: Not show, Graticule: Not show

Ilustración 26. Configuración del nodo worldmap

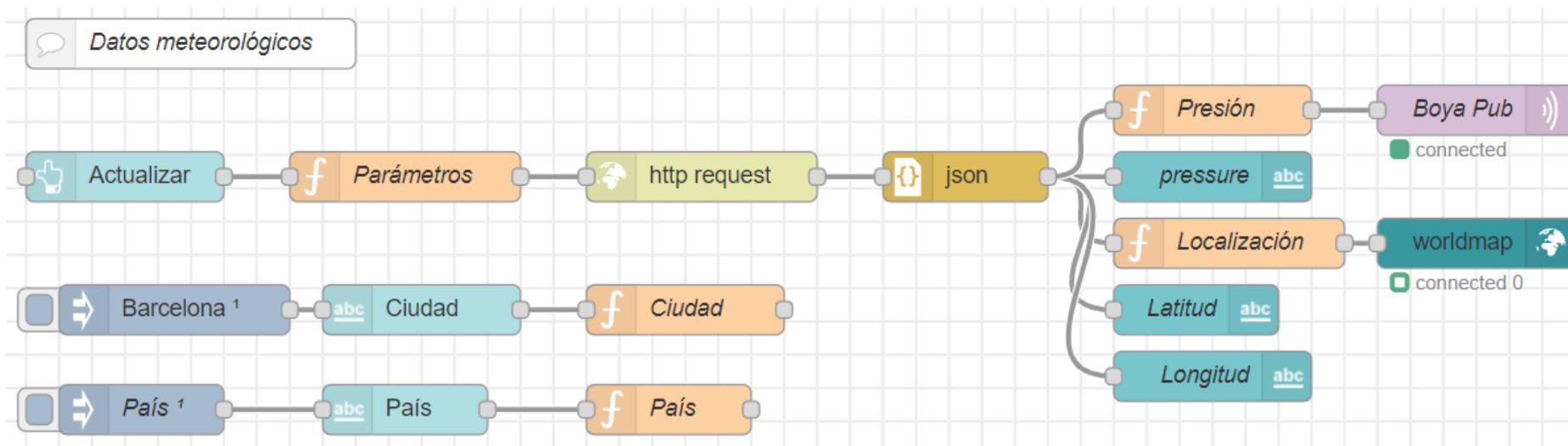


Ilustración 27. Flujo Node-Red para el acceso a los datos meteorológicos

3.3 Seguridad

Como tarea opcional, se pide activéis un acceso con usuario/contraseña para vuestra instancia de Node-RED.

Se pide poder securizar el acceso a NodeRed mediante usuario y contraseña, para conseguirlo se han seguido los siguientes pasos:

- 1) Se debe modificar el fichero settings.js que se encuentra ubicado en `/home/masterXX/node_modules/node-red`. Concretamente se debe descomentar el siguiente apartado:

```
// Securing Node-RED
// -----
// To password protect the Node-RED editor and admin API, the following
// property can be used. See http://nodered.org/docs/security.html for details.
adminAuth: {
  type: "credentials",
  users: [{
    username: "admin",
    password: "$2b$08$Yzpl.gvqNwbMRJNbZsue8.N2bEb6vD74FG2rgV4go2x5SOpvF0Cf.",
    permissions: "*"
  }]
},
```

En nuestro caso solo permitiremos el acceso al usuario administrador con todos los accesos tanto de lectura, modificación y deploy → permissions: "*" (en caso de que solo se quisiera dar permisos de lectura indicaríamos read).

A nivel de password este debe estar encriptado en formato hash utilizando el algoritmo bcrypt. Para obtener dicha encriptación utilizamos un comando de sistema:

node-red-admin hash-pw

Una vez ejecutado nos pedirá el password que queremos encriptar, utilizaremos "admin" obteniendo como salida dicho password encriptado que añadiremos al fichero de settings.js:

```
$2b$08$Yzpl.gvqNwbMRJNbZsue8.N2bEb6vD74FG2rgV4go2x5SOpvF0Cf.
```

- 2) Una vez realizadas las modificaciones paramos y arrancamos node-red para que se cargue la nueva configuración definida.

Sudo pm2 stop node-red25;Sudo pm2 start node-red25;

- 3) Para comprobar que todo ha funcionado correctamente volvemos a entrar a la página web de node-red e introducimos el usuario y el passwords definidos.

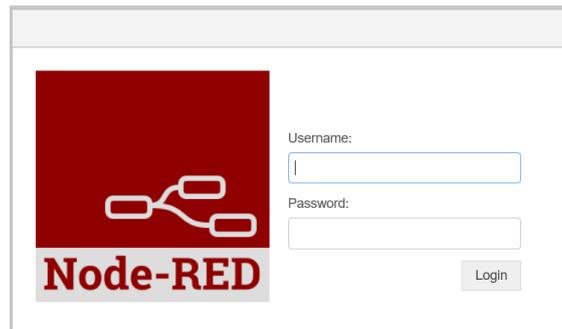


Ilustración 28. Pantalla de login Node-red

3.4 Consideraciones sobre la precisión del sistema

Hemos podido verificar que la presión atmosférica varía entre uno y dos milibares a lo largo del día, ello puede implicar errores de más de una decena de metros. Con la corrección constante de la presión atmosférica hemos conseguido un funcionamiento más o menos aceptable.

Considerando que el dispositivo irá aislado en una carcasa estanca, se plantea el problema de proveerle de una buena conexión al exterior para permitir una correcta medición sin comprometer la estanqueidad. Consideremos también la presencia de gases más o menos corrosivos como el sulfhídrico en una EDAR.

Resta pendiente la corrección de la temperatura, se aprecia que el sensor tiende a mostrar de dos a tres grados por encima de la temperatura real, posiblemente debido al calor generado por los componentes en la propia placa.

4. Anexos

4.1. Virtual commissioning

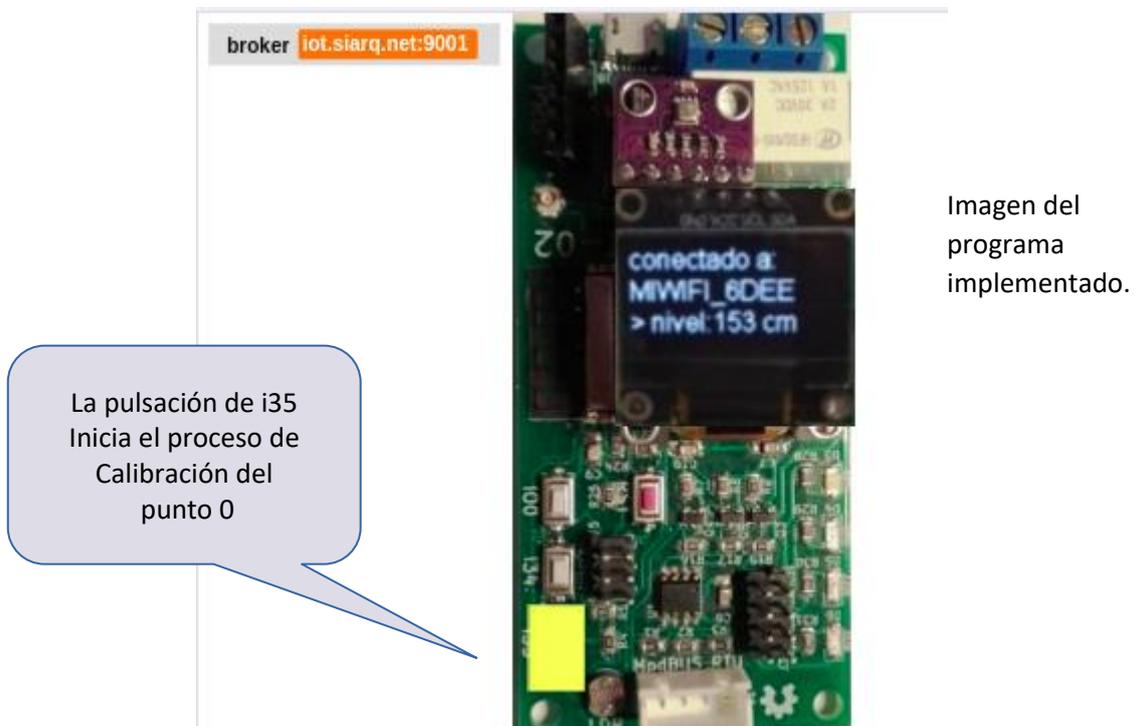
4.1.1. Simulador 1

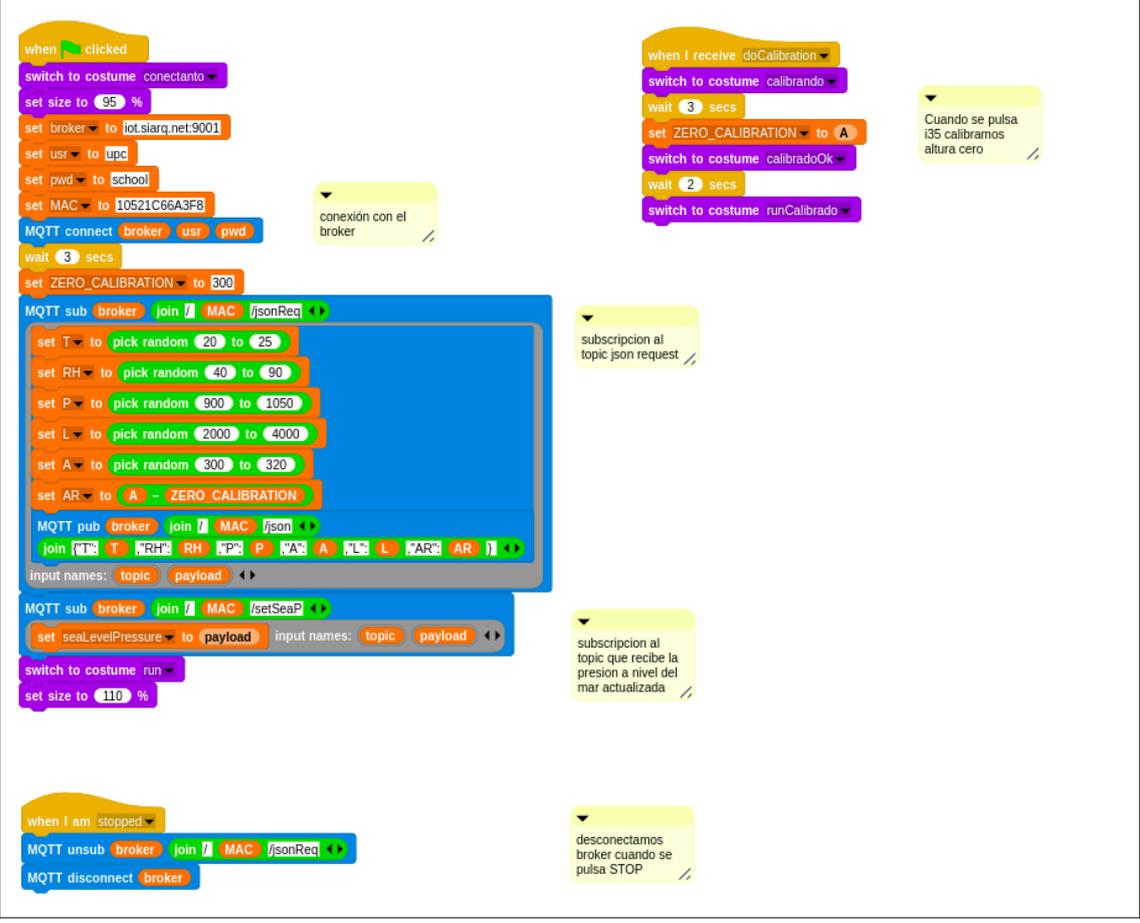
Este primer simulador tiene por objeto la recreación de la placa IoT, de manera que puedan desarrollarse las aplicaciones de captura y tratamiento de la información aún sin disponer del hardware o su firmware desarrollado.

Respecto a la propuesta snap del enunciado, se ha cambiado la lógica de funcionamiento, ya que en nuestro caso, es node-red quien envía una petición de información a través del topic **jsonReq** (cada 5 segundos) y es la placa quien responde a esa petición a través del topic **json**.

También se ha incorporado el nuevo topic **setSeaP** a través del cual node-red facilita periódicamente (cada 10 minutos) la información sobre la presión atmosférica local de cara a realizar las correcciones pertinentes.

Finalmente, se ha incorporado la funcionalidad de calibración del punto cero, recreando en snap tanto la secuencia como la imagen real de la placa. Como puede verse a continuación:





The Scratch script is organized into several sections:

- Initial Setup:**
 - When clicked: switch to costume `conectanto`, set size to `95 %`.
 - set broker to `lotsiarq.net:9001`, set usr to `lupc`, set pwd to `school`, set MAC to `10521C66A3F8`.
 - MQTT connect broker, usr, pwd.
 - wait 3 secs.
 - set ZERO_CALIBRATION to `300`.
- MQTT Subscription and Publication:**
 - MQTT sub broker, join MAC, /jsonReq.
 - set T to pick random `20 to 25`, set RH to pick random `40 to 90`, set P to pick random `900 to 1050`, set L to pick random `2000 to 4000`, set A to pick random `300 to 320`, set AR to `A - ZERO_CALIBRATION`.
 - MQTT pub broker, join MAC, /json.
 - join T, "RH": RH, "P": P, "A": A, "L": L, "AR": AR.
 - input names: topic, payload.
- Calibration Phase:**
 - MQTT sub broker, join MAC, /setSeaR.
 - set seaLevelPressure to payload, input names: topic, payload.
 - switch to costume `run`, set size to `110 %`.
- Final Cleanup:**
 - when I am stopped: MQTT unsub broker, join MAC, /jsonReq.
 - MQTT disconnect broker.

Yellow callout boxes provide additional context:

- `conexión con el broker` (next to MQTT connect)
- `subscripción al topic json request` (next to MQTT sub /jsonReq)
- `subscripción al topic que recibe la presión a nivel del mar actualizada` (next to MQTT sub /setSeaR)
- `desconectamos broker cuando se pulsa STOP` (next to MQTT disconnect)
- `Cuando se pulsa i35 calibramos altura cero` (next to set ZERO_CALIBRATION)

El simulador es accesible a través de este link:

http://extensions.snap.berkeley.edu/snap/snap.html#present:Username=domalc&ProjectName=virtual_commissioning

4.1.2. Simulador 2

En este segundo simulador se pretende poder visualizar los datos que salen de la boya añadiendo un Debug. Así, por la propia pantalla del NodeRed podemos ver las lecturas de la placa.

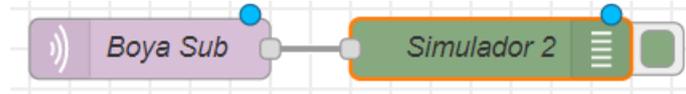


Ilustración 29. Nodos utilizados

```
27/4/2021 17:43:16 node: Simulador 2
/10521C66A3F8/json : msg.payload : string[123]
"
{"T":24.930000305175781,"RH":37,"P":9
69.6099853515625,"A":369.790008544921
88,"L":635,"AR":11.820232391357422,"S
LP":1013.25}"
```

Ilustración 30. Mensaje obtenido en la ventana de debugger

4.2. Servicio de monitorización y alertas via telegram

Como plus se ha montado un bot a nivel de telegram al cual se le pueden enviar comandos para conocer la información en tiempo real de la boya y al mismo tiempo nos notifica alarmas de si el depósito se encuentra ya lleno o vacío.

a) Generación Bot

En primer lugar debemos generar un bot que nos permita interactuar Node-Red con la aplicación móvil de Telegram. Los pasos a seguir para conseguirlo son los siguientes:

- 1) En Telegram existe un bot denominado "BotFather" cuya funcionalidad entre otras es la de generar nuevos bots para interactuar con ellos. A continuación podemos ver los comandos necesarios para crear el bot:

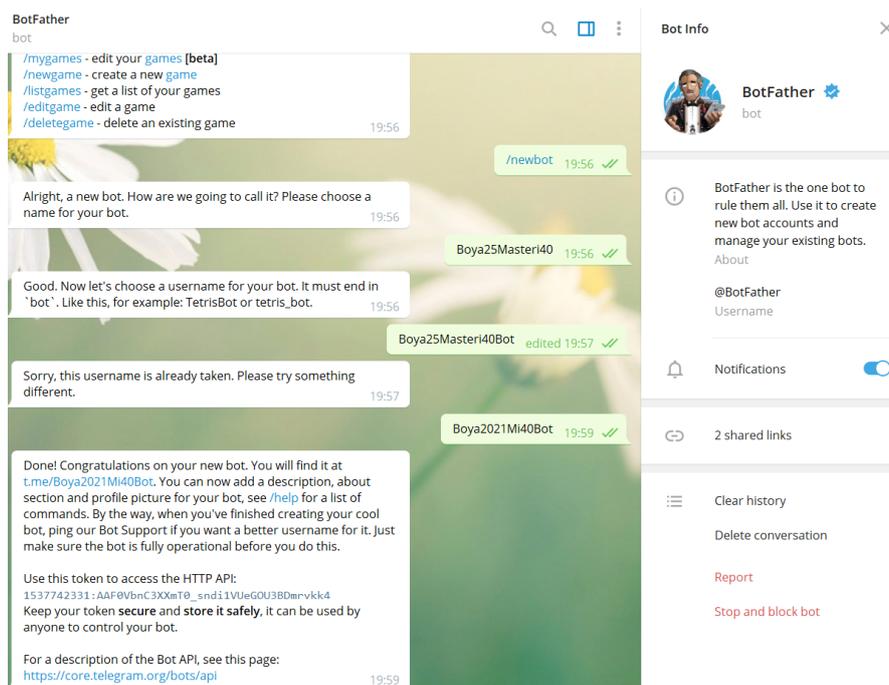


Ilustración 31. Instrucciones para crear el bot de monitorización

Los datos que necesitaremos en Node-Red para conectar con el bot son:

- Name: Boya25Masteri40.
- Token: 1537742331:AAF0VbnC3XXmT0_sndi1VUeGOU3BDmrvkk4

- 2) Una vez creado el bot se necesitará la información de los grupos con los cuales se quiera que interactúe. Para ello es necesario obtener los identificadores de estos que se consigue de nuevo añadiendo un bot propio de Telegram denominado "GroupID".

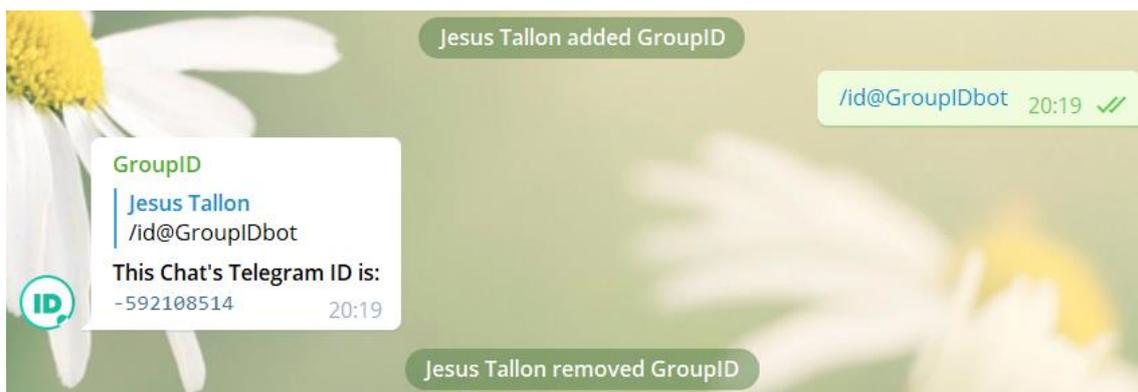
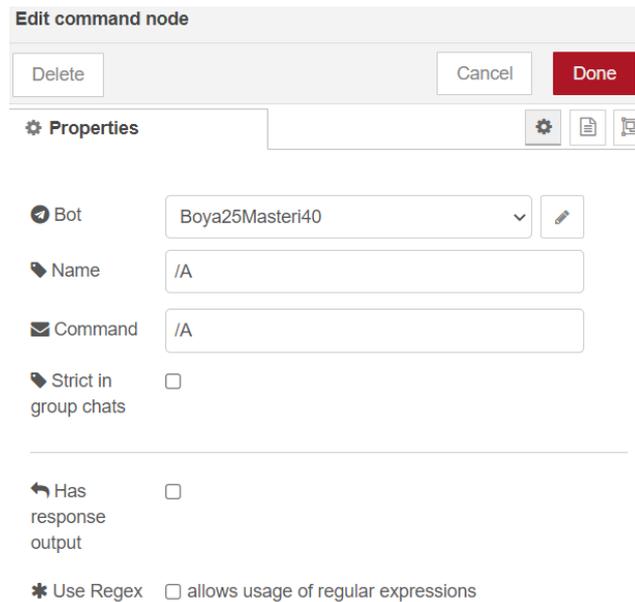


Ilustración 32. Obtención del identificador del grupo de Telegram

b) Comandos Telegram

Para interactuar con Telegram desde Node-Red utilizaremos la librería node-red-contrib-telegrambot. Esta librería tiene dos tipos de nodos básicos que utilizaremos para interactuar con el bot:

- **Command:** que permite recibir las peticiones de comandos que se hagan al bot desde el chat del grupo. Los comandos van precedidos de el carácter “/” y en nuestro caso los utilizaremos para proporcionar los últimos datos recibidos de la placa:
 - **/A:** Altura.
 - **/RH:** Humedad relativa.
 - **/T:** Temperatura.
 - **/P:** Presión atmosférica.
 - **/L:** Radiación solar.
 - **/AR:** Altura relativa.



Edit command node

Delete Cancel Done

⚙️ Properties

🤖 Bot Boya25Masteri40

📄 Name /A

✉️ Command /A

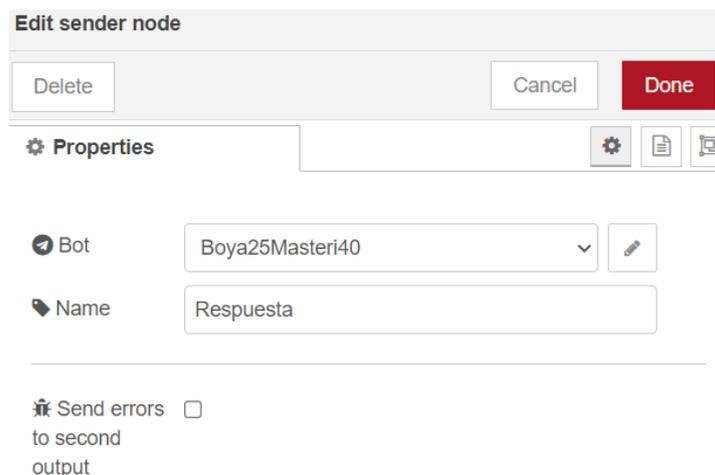
🔒 Strict in group chats

↩️ Has response output

⚡ Use Regex allows usage of regular expressions

Ilustración 33. Configuración de un nodo de tipo Command

- **Sender:** que nos permite realizar el envío de un mensaje al chat del grupo.



Edit sender node

Delete Cancel Done

⚙️ Properties

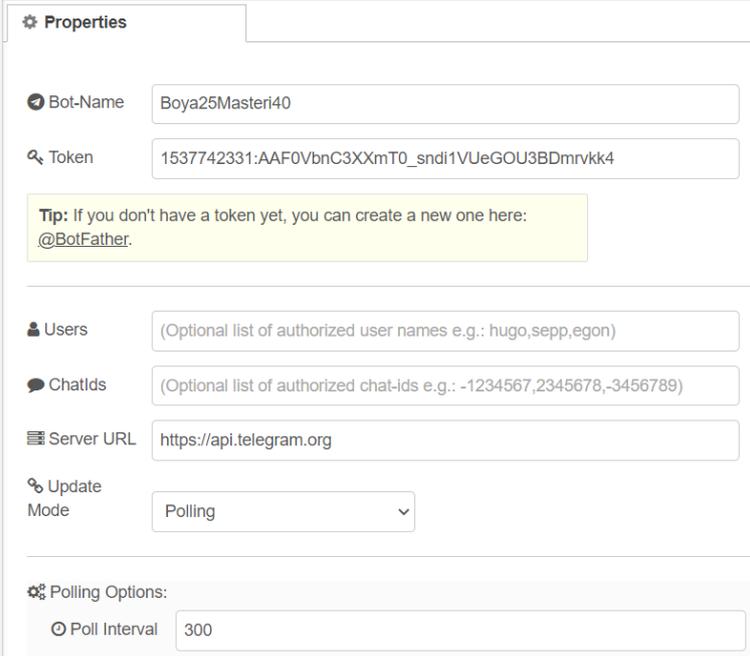
🤖 Bot Boya25Masteri40

📄 Name Respuesta

🚨 Send errors to second output

Ilustración 34. Configuración de un nodo de tipo Sender

Como se puede observar para que funcionen estos nodos es necesario haber dado de alta el bot en Node-Red por lo que a continuación se presenta la configuración de este con los datos obtenidos anteriormente:



The screenshot shows the configuration panel for a Telegram bot in Node-Red. It includes fields for Bot Name (Boya25Master140), Token (1537742331:AAF0VbnC3XXmT0_sndi1VUeGOU3BDmrvk4), a tip to create a token using @BotFather, a list of authorized users (Optional list of authorized user names e.g.: hugo,sepp,egon), a list of authorized chat IDs (Optional list of authorized chat-ids e.g.: -1234567,2345678,-3456789), Server URL (https://api.telegram.org), Update Mode (Polling), and Polling Options (Poll Interval: 300).

Ilustración 35. Configuración del bot en Node-Red

El flujo completo para la gestión de comandos simplemente se encarga de detectar cuando se lanza un comando en el chat y según este generamos el mensaje de respuesta. El mensaje estará compuesto a partir de las variables de flujo que se habían montado con anterioridad para tener la última lectura de los datos recibidos y es aquí donde se indicará el chat donde se enviará el mensaje:

```

1 msg.payload={};
2 msg.payload.chatId='-592108514';
3 msg.payload.type='message';
4 var A = flow.get('A');
5 msg.payload.content=A+' metros';
6 return msg;

```

Ilustración 36 Ejemplo de función de generación del mensaje

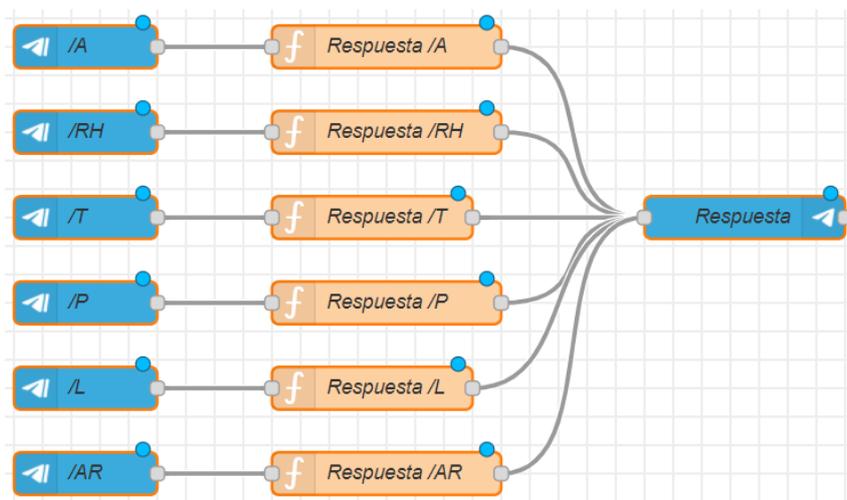


Ilustración 37. Comandos Node-Red Telegram

c) Alarmas Telegram

De forma similar al sistema que hemos montado para aceptar comandos se propone enviar mensajes a partir de un evento o dato generado por la placa. Con nodos ya visto con anterioridad lo que se hace es simplemente comprobar al recibir los datos de la placa si la altura relativa está por encima de 50 metros o por debajo de los 0 metros lo que hace que se envíe un mensaje indicando si el depósito está lleno o vacío.

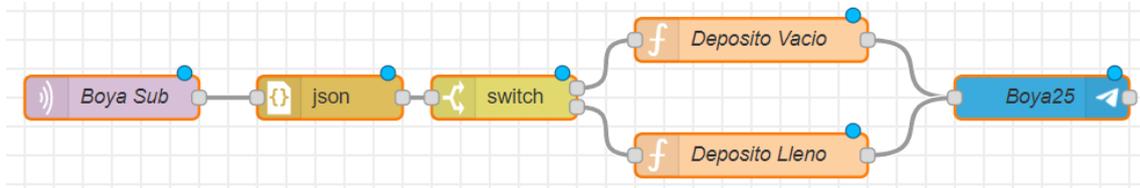


Ilustración 38. Notificación alarmas Node-Red Telegram

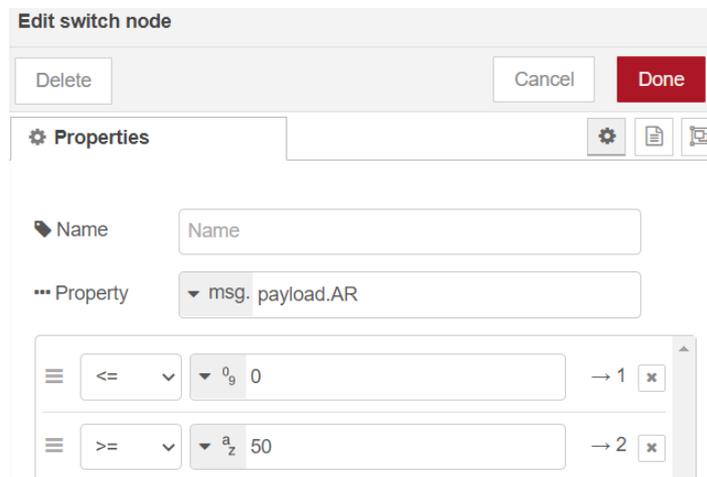


Ilustración 39. Configuración del nodo switch



Ilustración 40. Ejemplo de generación del mensaje de alarma