RÍO NEGRO
UNIVERSIDAD NACIONAL

PRACTICA PROFESIONAL SUPERVISADA

"Investigación y desarrollo sobre la plataforma IOT OpenRemote"

PROFESOR	Javier Areta
TUTOR	Javier Barragan
ALUMNO	Ignacio Egea

Índice

Introducción **3**
Inicialización de la herramienta **3**
Modificación de la interfaz **4**
Creación de un realm 5
Cambio de logo 5
Cambio de mapa 6
Exploración de la plataforma **8**
Assets 8
Agents 8
Attributes 8
Configuration items 8
Usuarios 9
Reglas 9
Implementación de un sensor **10**
Configuración de servidor LoRaWAN 10
Conexión por MQTT 15
Conexión por WebSocket 18
Conexión por HTTP 20
Conexión por TCP y UDP 22
Demostración de datos en el mapa **24**
Implementación de una regla **27**
Creación de un usuario **30**
Conclusiones **33**
Anexo 1 : Repositorio del proyecto **34**

Introducción

Este informe trata acerca del trabajo que se realizó durante la práctica profesional supervisada en ALTEC S.E., acerca de la plataforma de IOT OpenRemote. Esta se trata de una aplicación capaz de manejar dispositivos o sensores, de manera que se envíen o reciban datos de manera automatizada. Además, cuenta con un Front-End para interactuar y configurar los distintos sistemas conectados. Cuenta con un mapa, en el cual se pueden posicionar geográficamente los dispositivos, para un rápido acceso a la información que proporcionen.

La idea principal fue conocer los protocolos de comunicación que permite utilizar la herramienta, los métodos de automatización que dispone y conectar algún dispositivo para controlar y observar. Además, se instaló la plataforma sobre un servidor, que podrá ser accedido por los que deseen conocer la información que otorgan los sensores conectados al sistema.

La estructura del informe será de forma tal que, al ser leído, el lector pueda obtener los conocimientos suficientes para implementar su propia plataforma OpenRemote. El desarrollo será llevado a cabo sobre el sistema operativo Linux.

Inicialización de la herramienta

Se inició leyendo la documentación que los desarrolladores disponen en github. Esta consiste de un pequeño tutorial para configurar el software sobre el sistema operativo. Al ser un proyecto realizado sobre Docker, se procedió a instalar el software para manejar las imágenes del contenedor, llamado Docker Desktop. Luego, se descargó el archivo compose de docker *docker-compose.yml* para OpenRemote en un directorio dedicado y se corrió el contendor. Este genera, por defecto, un entorno de la herramienta en el sistema operativo. Para acceder, se debe ingresar al localhost desde cualquier navegador web: https://localhost. En este enlace se encontrará el menú de ingreso a la plataforma, donde se deberá hacer un login para acceder a la misma. Cabe destacar que, se puede cambiar el URL por defecto para acceder a la plataforma. Recomiendo seguir los pasos para esta configuración inicial en https://github.com/openremote/openremote.

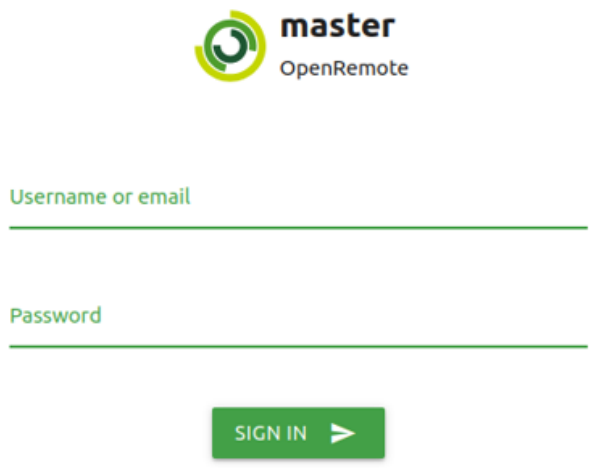

Figura 1 : Pantalla de logueo de OpenRemote

Con el usuario por defecto "*admin*" y contraseña "*secret*" accedemos a la interfaz de la herramienta.

Modificación de la interfaz

Al acceder a la plataforma, nos encontramos con un mapa por defecto de la ciudad de Rotterdam, como también las distintas pestañas para interactuar con la herramienta.

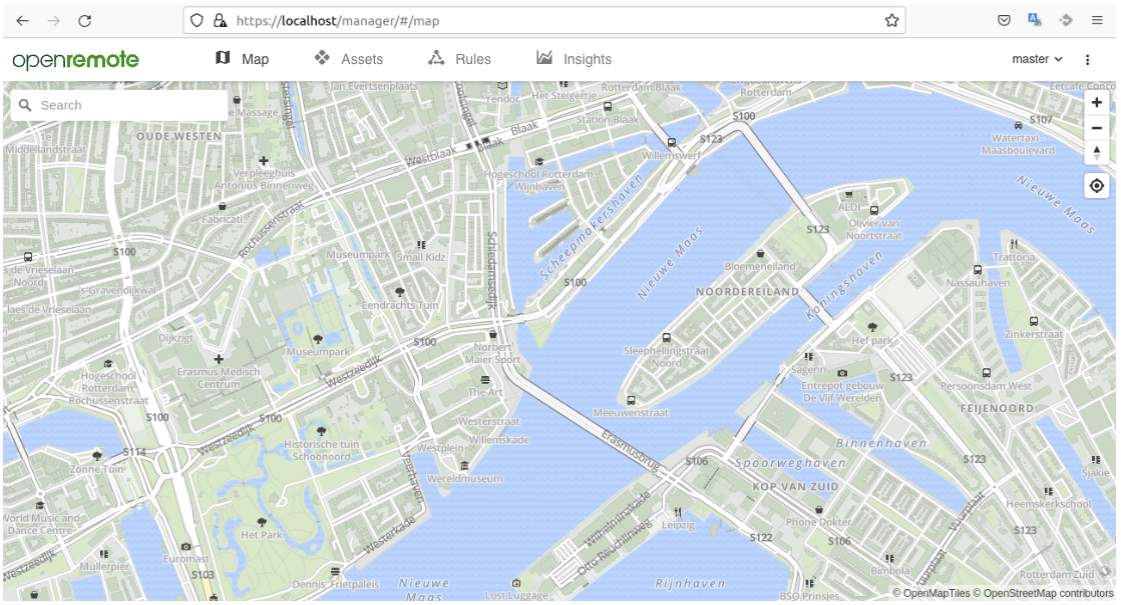

Figura 2 : Interfaz inicial OpenRemote

Como objetivo inicial, se planteó la modificación del mapa a la ciudad de Bariloche y la colocación de algún logo de la empresa en el frontend, con la idea de conocer de a poco el programa.

Creación de un realm

Un realm es una capa de aislamiento dentro del programa, que permite tener usuarios, dispositivos, reglas y hasta un estilo propio de interfaz dentro de cada realm creado. En este caso, crearemos un realm llamado Bariloche donde luego asignaremos el mapa y logos correspondientes. Para esto, entramos a la pestaña realms dentro de la interfaz y lo creamos. Por último, reiniciamos la plataforma para ver los cambios.

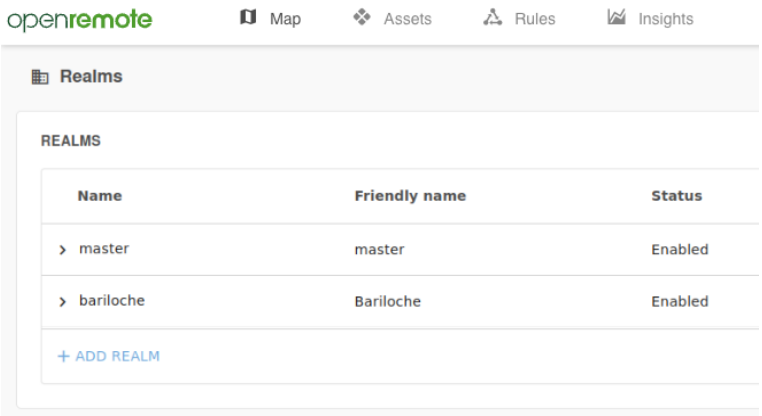

Figura 3 : Creación de un nuevo realm

Cambio de logo

Para realizar el cambio de logotipos en la plataforma, se debe generar una cadena de directorios en el directorio dedicado de OpenRemote, con la siguiente estructura: */deployment/manager/app*. Aquí dentro, se deberá generar un archivo JSON con nombre *manager_config.json*, donde definiremos los colores de la plataforma, como además el cambio de logos:

```
{
  "realms": {
    "default": {
      "appTitle": "ALTEC IoT",
      "styles": ":host > * {--or-app-color2: #F9F9F9;
                            --or-app-color3: #22211f;
                            --or-app-color4: #6fa8dc;
                            --or-app-color5: #CCCCCC;}",
      "logo": "/images/logo.png",
      "logoMobile": "/images/logo-mobile.png",
      "favicon": "/images/favicon.png"
    }
  }
}
```

Figura 4 : Estructura y contenido de *manager_config.json*

Generamos otro directorio llamado *images*, donde colocaremos la imagen del icono que deseamos. En este caso, se eligió el logotipo de ALTEC y se le asignó el mismo a los tres logos de la aplicación. “logo” corresponde al logo en la página, “logoMobile” es el mismo pero en versión móvil para teléfonos y “favicon” es el icono de favoritos. Los colores son los mismos que tiene por defecto. Este cambio afectará a todos los realms de la plataforma, ya que esta como default. Una vez completado esto, se debe reiniciar la plataforma para ver los cambios.

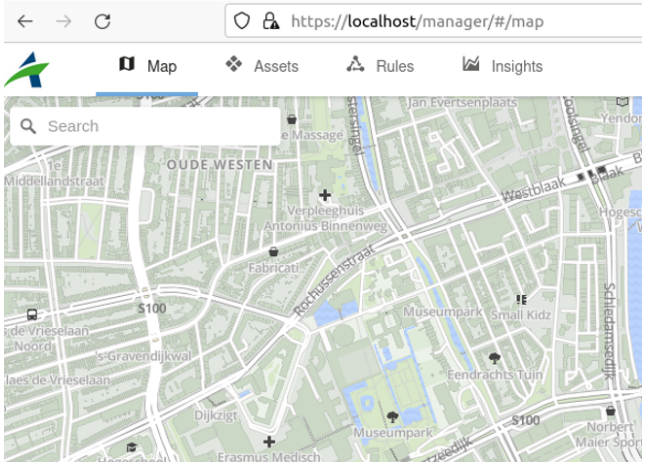

Figura 5 : Interfaz con logos modificados

Cambio de mapa

La modificación del mapa se hace a través de un archivo de mapa, el cual fue descargado de openmaptiles.org, un sitio que contiene mapas gratuitos. Con el archivo del mapa de Bariloche, generamos una cadena de directorios de la forma */deployment/map*, donde colocamos dicho mapa y lo renombramos *mapdata.mbtiles*. Además, debemos agregar al mismo directorio un archivo *mapsettings.json*, para la configuración del archivo de tipo mapa. Se puede acceder a descargar un template del github. Aquí podemos ajustar el punto central del mapa, los límites y el zoom.

```
{
  "options": {
    "bariloche": {
      "center": [ -71.3, -41.05 ],
      "bounds": [ -71.59, -41.21, -71.13, -41.03 ],
      "zoom": 10,
      "maxZoom": 19,
      "boxZoom": false
    },
  },
  "version": 8,
  "sources": {
    "vector_tiles": {
      "type": "vector",
      "url": "mbtiles://mapdata.mbtiles"
    }
  },
```

Figura 6 : Estructura y contenido del template *mapsettings.json*

Agregando “options” al template, podemos elegir los parámetros de mapa para cada realm que exista en la plataforma. En ese caso, agregamos “bariloche” y configuramos los valores del mapa a gusto. Por último, editamos el archivo de docker *docker-compose.yml* para reflejar los cambios, añadiendo el volumen *./deployment:/deployment* al container *manager*.

```
  manager:
#    privileged: true
    restart: always
    image: openremote/manager:${MANAGER_VERSION:-latest}
    depends_on:
      keycloak:
        condition: service_healthy
    environment:
      OR_SETUP_TYPE:
      OR_ADMIN_PASSWORD:
      OR_SETUP_RUN_ON_RESTART:
      OR_EMAIL_HOST:
      OR_EMAIL_USER:
      OR_EMAIL_PASSWORD:
      OR_EMAIL_X_HEADERS:
      OR_EMAIL_FROM:
      OR_EMAIL_ADMIN:
      OR_HOSTNAME: ${OR_HOSTNAME:-localhost}
      OR_ADDITIONAL_HOSTNAMES: ${OR_ADDITIONAL_HOSTNAMES:-}
      OR_SSL_PORT: ${OR_SSL_PORT:--1}
      OR_DEV_MODE: ${OR_DEV_MODE:-false}

      # The following variables will configure the demo
      OR_FORECAST_SOLAR_API_KEY:
      OR_OPEN_WEATHER_API_APP_ID:
      OR_SETUP_IMPORT_DEMO_AGENT_KNX:
      OR_SETUP_IMPORT_DEMO_AGENT_VELBUS:
    volumes:
      - manager-data:/storage
      - ./deployment:/deployment
```

Figura 7 : Estructura y contenido del archivo *docker-compose.yml*

Una vez completado esto, se debe reiniciar la plataforma para ver los cambios.

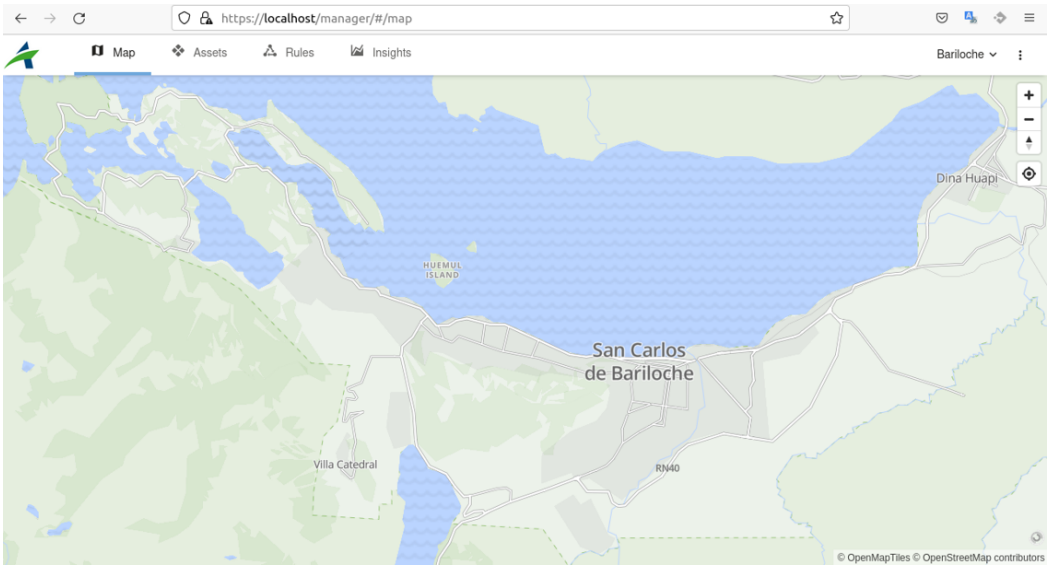

Figura 8 : Interfaz con mapa modificado

Exploración de la plataforma

Con el realm creado, ya podemos empezar a interactuar con la plataforma dentro del mapa de Bariloche. Iniciaremos conociendo los distintos aspectos de la herramienta para poder asociar algún tipo de sensor e interactuar con él.

Assets

Un asset es una representación digital de algún objeto físico o lógico que se conecta con la plataforma. Son definidos para cada realm y quedan aislados en el mismo. Se le pueden asignar reglas para automatizar el manejo de la información. Como ejemplo, un sensor podría ser un asset.

Agents

Un agente es un tipo especial de asset que se encarga de la conexión entre la plataforma y dispositivos o servicios externos, a través de protocolos de comunicación (HTTP/TCP/IP/…). El agente contiene los parámetros de configuración de la comunicación como atributos.

Attributes

Cada asset puede tener uno o más atributos, los cuales contienen algún valor. Un valor puede ser cualquiera que pueda ser representado como JSON. Como ejemplo, el valor de temperatura que lee un sensor puede ser un atributo.

Configuration items

Además de tener un valor, el atributo puede tener definido ítems de configuración, que controlan el comportamiento del mismo atributo. Se puede tener más de un ítem por cada atributo. Como ejemplo, la unidad del valor del atributo puede ser un ítem de configuración. Algunos de ellos son:

- Agent link: Parámetro para conectar el atributo con un agente, permitiendo recibir datos del mismo.
- Show on dashboard: Ítem que permite mostrar o no información del atributo en el mapa.
- Label: Etiqueta con la que se mostrará el atributo en el mapa.
- Attribute links: Parámetros que permite la conexión entre atributos, para enviar datos entre ellos.

Usuarios

Se pueden definir dos tipos de usuarios en la plataforma, dependiendo de su funcionalidad.

- Usuario regular: Este tipo de usuario es el que se loguea introduciendo su nombre y contraseña. Es el típico definido para el uso de la plataforma, pudiendo restringir el acceso de lectura/escritura a distintos parámetros.

- Usuario de servicio: Este tipo de usuario se loguea de forma programada usando un id de cliente y está diseñado para conectarse y manejar algunas APIs de comunicación con el exterior.

Reglas

Openremote tiene un motor de reglas que pueden ser utilizadas para cumplir varias funciones, como por ejemplo la modificación de algún atributo en función de otros atributos o parámetros de la plataforma, o el envío de notificaciones a partir de alguna condición. Las reglas pueden ser globales, permitiendo usar cualquier atributo de cualquier realm, o también locales, funcionando solo con lo disponible en el realm asignado. Existen 3 tipos distintos de lenguajes para programar las reglas, todos con un propósito distinto:

- When-Then: Ideal para crear reglas en base a eventos. Permite seleccionar los atributos y condiciones desde una UI, logrando una interacción más intuitiva con los usuarios. Además, permite el envío de notificaciones y el uso de timers.

- Flow: Usado para comunicar atributos, utilizando algún tipo de conversión con sus datos. Por ejemplo, para el cálculo del consumo eléctrico de una ciudad, se pueden sumar los valores de todos los medidores y guardarlo en otro atributo que contiene el consumo total.

- Groovy: Usado para la creación de reglas complejas, siendo programadas en el lenguaje Groovy. Se tiene poder sobre toda la plataforma, pero es más difícil ya que requiere de un gran conocimiento de la herramienta y el lenguaje de programación.

Implementación de un sensor

Se decidió por implementar la comunicación con un sensor DHT11 de temperatura y humedad. Se utilizaron los protocolos TCP, UDP, MQTT, HTTP y Websocket. Además, se hizo uso de la infraestructura LoRa que hay en la empresa, en la cual se tiene implementado un servidor LoRaWAN para utilizar los protocolos de comunicación antes mencionados.

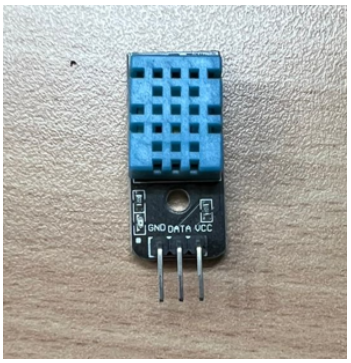

Figura 9 : Sensor de temperatura y humedad DHT11

Configuración de servidor LoRaWAN

En la empresa se encuentra instalada la infraestructura para implementar una comunicación por LoRa con el sensor. Se encuentra funcionando un servidor LoRaWAN que escucha a los dispositivos que se conectan al gateway de LoRa.

La idea de implementar este protocolo es, comunicar por LoRaWAN el sensor con el servidor, y desde el servidor enviar los datos por alguno de los protocolos a la plataforma OpenRemote. El servidor en cuestión es el lorawan-server de Petr Gotthard, alojado en https://github.com/gotthardp/lorawan-server. Cabe aclarar que, la configuración de OpenRemote para la comunicación por los protocolos mencionados funciona con cualquier otra herramienta, no únicamente con el servidor LoRaWAN utilizado en este caso.

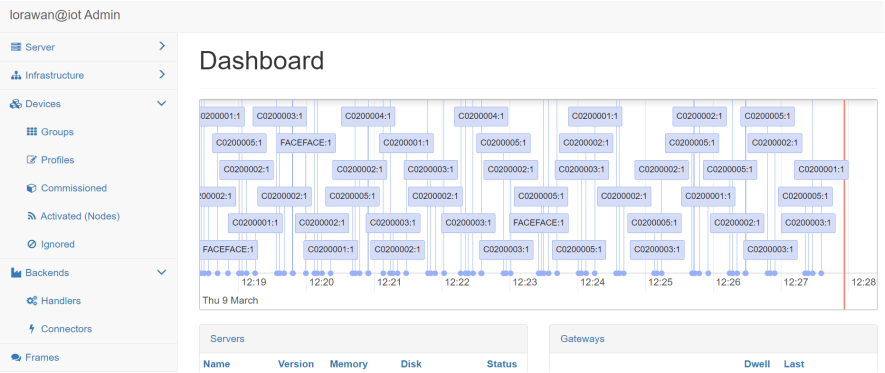

Figura 10 : Servidor LoRaWAN de Petr Gotthard

Para esta implementación se utilizó un Adafruit Feather M0, el cual cuenta con radio LoRa en la placa. El firmware fue desarrollado sobre la plataforma Arduino y fue almacenado en un repositorio, que se encuentra como anexo a este documento.

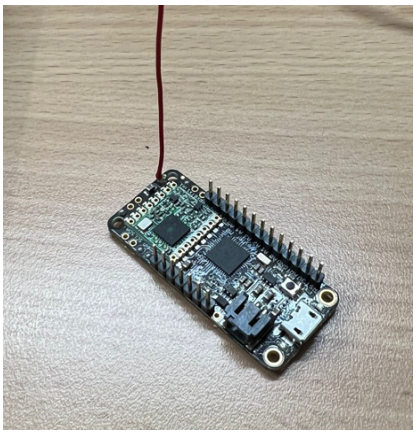

Figura 11 : Placa Adafruit Feather M0

Con el sensor reportando al gateway de LoRa, podemos iniciar la configuración para que el servidor lo escuche. Inicialmente, apuntamos el gateway al servidor, y creamos una nueva instancia de gateway en la pestaña Gateways. Aquí, solo basta con agregar la dirección MAC del mismo. Esto ya estaba implementado, dado que el servidor ya estaba funcionando con otros dispositivos.

El siguiente paso es crear una red en la pestaña Networks. Esta configuración contiene todos los parámetros de comunicación, dependiendo de la zona y las leyes aplicadas.

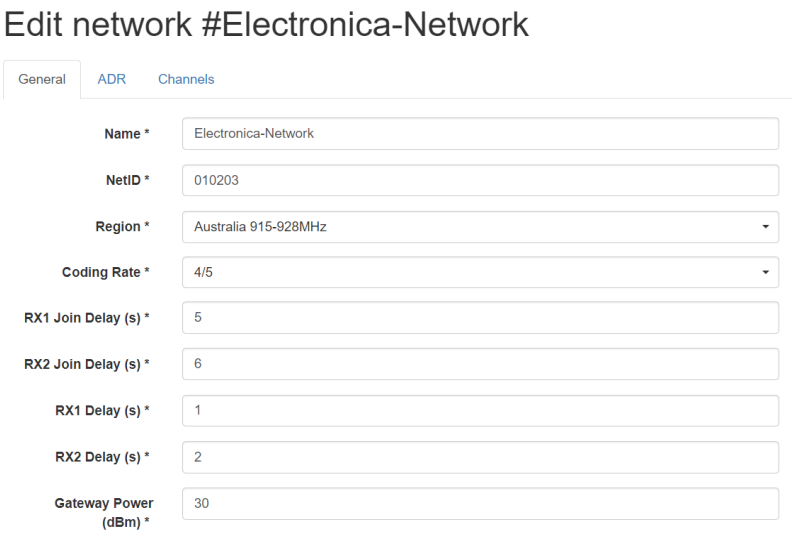

Figura 12 : Configuración de una red LoRa

En Argentina, la banda de LoRa es de 915-928 MHz. Los datos de configuración para esta banda se pueden encontrar en el manual del protocolo LoRaWAN.

Ahora, creamos un grupo para asignarle esta red. En la pestaña Devices, seleccionamos la opción Groups. Simplemente creamos uno nuevo y le asignamos la red anteriormente configurada.

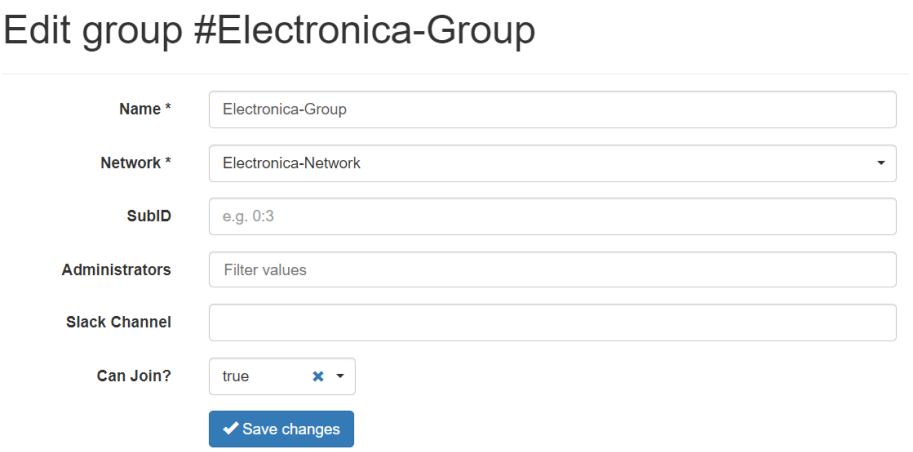

Figura 13 : Creación de un grupo

A continuación, debemos configurar un handler para el sensor, el cual se encargará de interpretar el mensaje enviado. En la pestaña Backends del servidor de LoRaWAN encontraremos la opción Handlers.

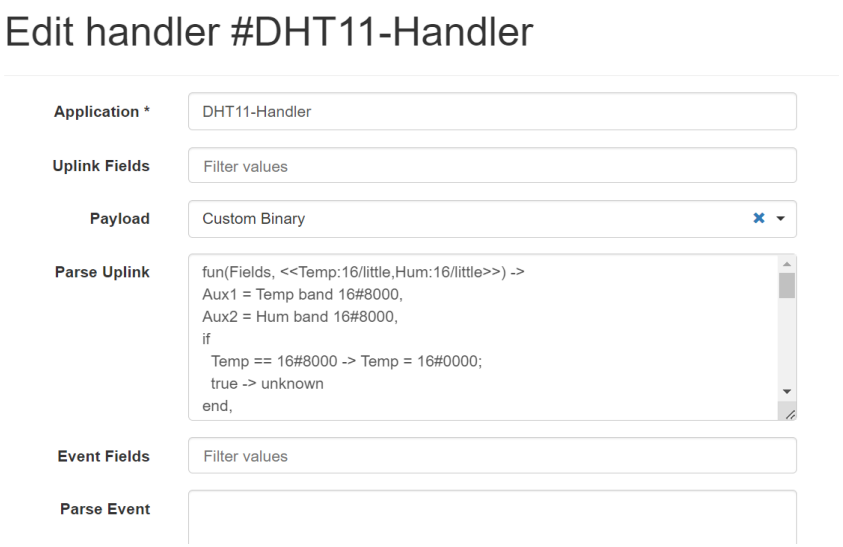

Figura 14 : Handler para el sensor DHT11

Debemos ponerle un nombre e indicarle que tipo de información va a recibir. En nuestro caso, el Feather M0 envía la información codificada en sflt16, un tipo de datos que representa un unsigned float en 16 bits. Esta codificación es necesaria por el poco ancho de banda que se dispone en la comunicación LoRa. Se utilizarán 2 sflt16, uno para la temperatura y otro para la humedad.

Dentro del campo Parse Uplink, se debe colocar una función que decodifique la información recibida. Esta debe estar escrita en Erlang, el lenguaje de programación con el cual fue implementado el servidor. Esta función también se encuentra en el anexo. Esta devuelve al ejecutarse, la temperatura, humedad y estado de conexión del sensor.

A continuación, creamos un perfil en el servidor para el dispositivo DHT11. En la pestaña Devices, seleccionamos la opción Profiles y creamos uno nuevo.

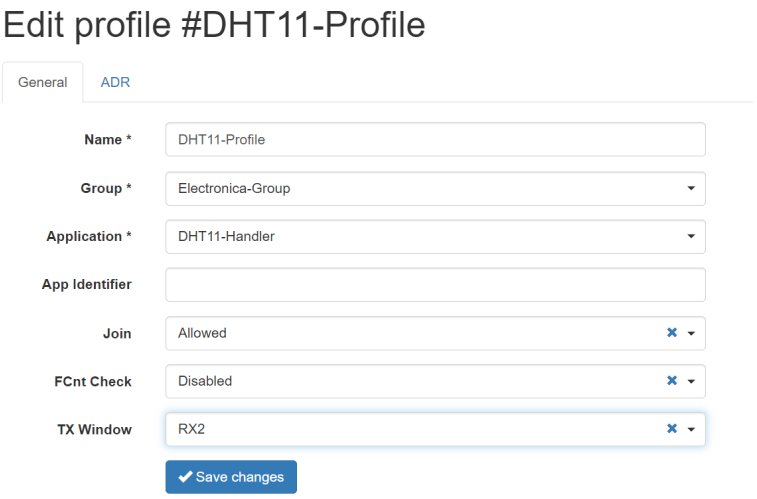

Figura 15 : Creación de un perfil para el sensor DHT11

Debemos asignarle el grupo creado anteriormente, como también el handler que va a usar. El resto de los parámetros van por defecto.

Con el perfil creado, debemos ahora agregar el dispositivo al servidor. En la pestaña Devices, seleccionamos la opción Activated (Nodes). Aquí, agregamos un nuevo nodo y lo configuramos.

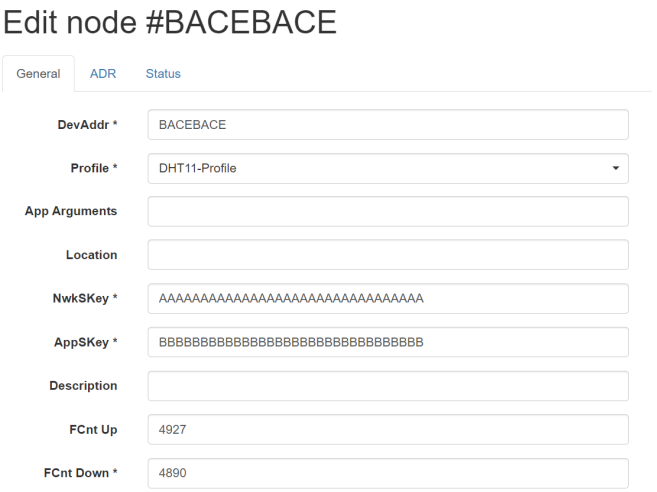

Figura 16 : Configuración del dispositivo

Debemos rellenar los campos DevAddr, NwkSKey y AppSkey con las credenciales definidas en el firmware. Estos datos son parte del protocolo LoRaWAN y corresponden a la capa de seguridad. Por último, en el campo Profile seleccionamos el perfil del DHT-11 creado anteriormente.

Solo queda alimentar el Feather M0 para ver que los mensajes llegan al servidor. Podemos ver en el dashboard el uplink y downlink del dispositivo. En la pestaña Frames, podemos ver el contenido del mensaje para este caso, un hexadecimal de 32 bits sin decodificar. El handler hará la decodificación cuando se envíe la información con un conector a otro lado.

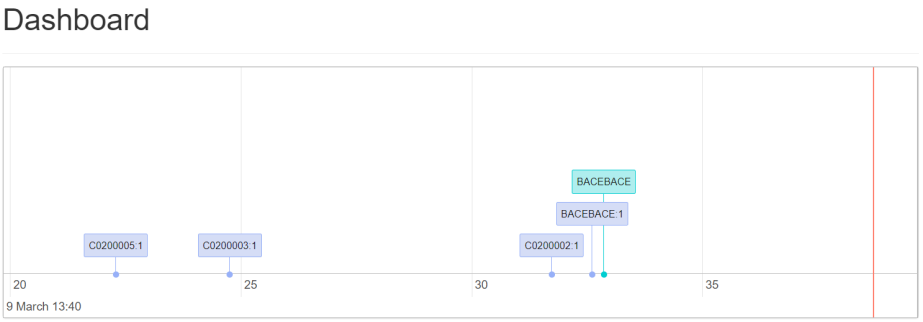

Figura 17 : Datos llegando al servidor

Por lo tanto, nos falta hacer el conector que vincule al servidor con OpenRemote. Pero primero, se debe preparar a la plataforma para una conexión por alguno de los protocolos mencionados.

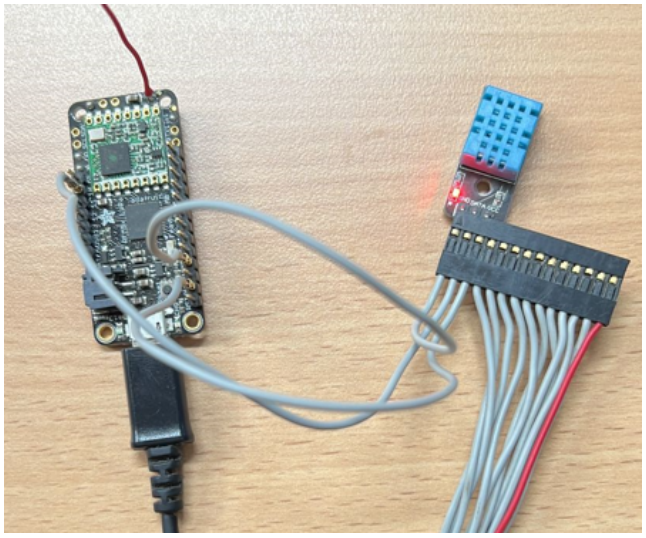

Figura 18 : Placa Adafruit Feather M0 y sensor DHT11 conectados

Conexión por MQTT

Se inició generando el asset correspondiente para el sensor DHT11. En la pestaña assets de la plataforma tenemos las herramientas para crearlo. Le damos al símbolo + y se desplegará la lista de templates de assets que contiene la plataforma. En nuestro caso, al ser un sensor de temperatura y humedad, aprovecharemos el template Weather Asset, que corresponde con las variables a medir. En caso de no encontrar un template adecuado, se puede generar un Thing Asset, que es completamente configurable.

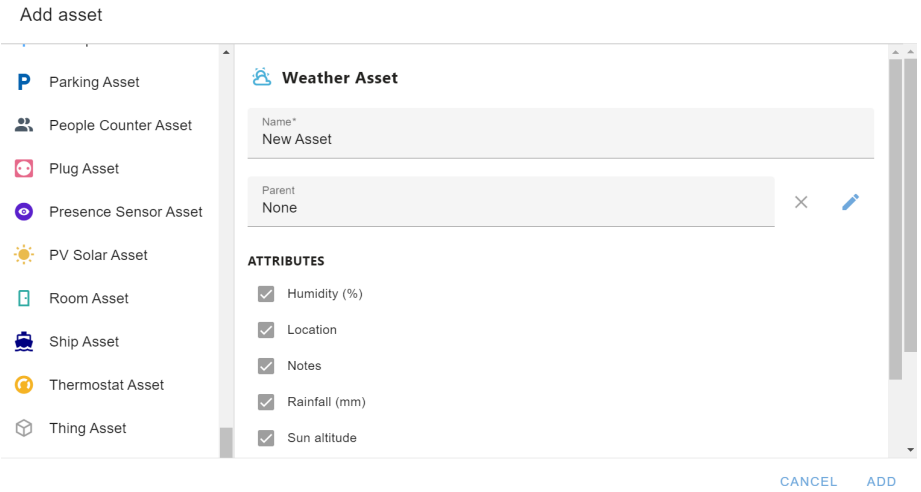

Figura 19 : Creación de un nuevo Weather Asset

Debemos ponerle un nombre y debajo enseña los atributos que contiene por defecto. En nuestro caso, lo llamaremos Sensor DHT11 - Área Electrónica. Ahora, agregamos un atributo en donde se recibirán los datos de temperatura, humedad y estado del sensor. El servidor envía los datos en formato JSON, por lo tanto crearemos un atributo de dicho tipo.

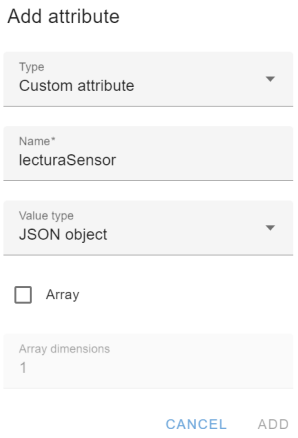

Figura 20 : Creación de un nuevo atributo tipo JSON

Apretamos ADD ATTRIBUTE y lo definimos como Custom attribute. Le ponemos un nombre y definimos el valor que contiene como JSON object. Con esto, el sensor está listo para recibir los datos en este atributo.

Para lograr una conexión por MQTT, OpenRemote funciona como Broker mediante la creación de un usuario de servicio para la conexión con la API de dicho protocolo. Primero, se debe crear el usuario. Para ello, dentro de la plataforma vamos a la pestaña usuarios y dentro de la categoría Service Users, creamos la instancia de MQTT.

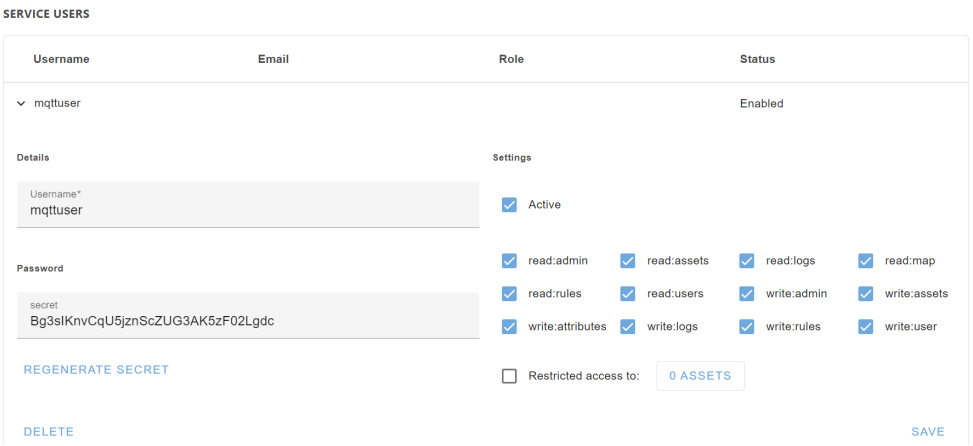

Figura 21 : Configuración del usuario de servicio para MQTT

Debemos nombrar al usuario "*mqttuser*", y darle los permisos de lectura/escritura correspondientes para que pueda modificar atributos del asset sensor creado anteriormente. En este caso, se le dio la totalidad de permisos. Una vez generado, se completará el campo "*Password*" con la correspondiente credencial para realizar la comunicación.

Lo que falta configurar es el conector en el servidor de LoRaWAN. En la pestaña Backends, seleccionamos la opción Connectors. Aquí debemos configurar de forma tal que el conector llegue a la plataforma OpenRemote.

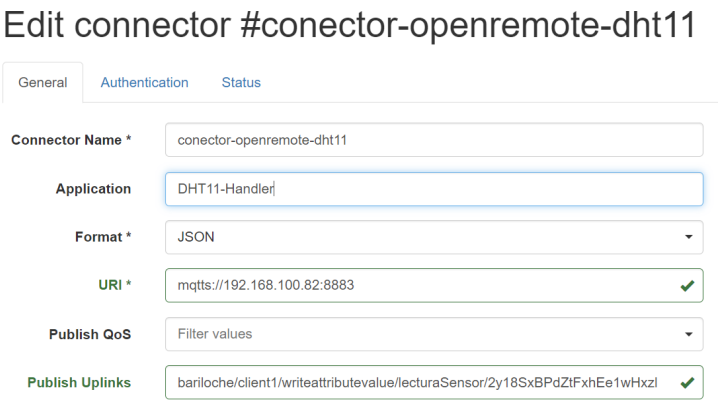

Figura 22 : Configuración del conector MQTT

Debemos colocarle un nombre y asignarle el handler que configuramos en la sección anterior. El tipo de dato que va a entregar es JSON, por lo tanto seleccionamos ese formato en la casilla Format. Para que llegue al Broker de MQTT en OpenRemote, debemos darle la dirección IP y el puerto 8883 (asignado para MQTT) de la máquina donde se esté corriendo la plataforma, con la salvedad de usar el formato mqtts://<IP>:8883. En la casilla Publish Uplinks, debemos colocar el topic de MQTT donde se quiere publicar los datos recibidos. El formato de topic que usa OpenRemote para escribir algún atributo es:

{realm}/{clientId}/writeattributevalue/{attributeName}/{assetId}

- realm: El realm donde se encuentra el asset.
- clientId: Nombre arbitrario pero único para cada cliente que se conecta a la plataforma.
- attributeName: Nombre del atributo donde se publicaran datos.
- assetId: Id único de cada asset. Se puede leer en el url de OpenRemote cuando se selecciona cada asset.

Por lo tanto, en nuestro caso quedaria:

bariloche/client1/writeattributevalue/lecturaSensor/2y18SxBPdZtFxhEe1wHxzl

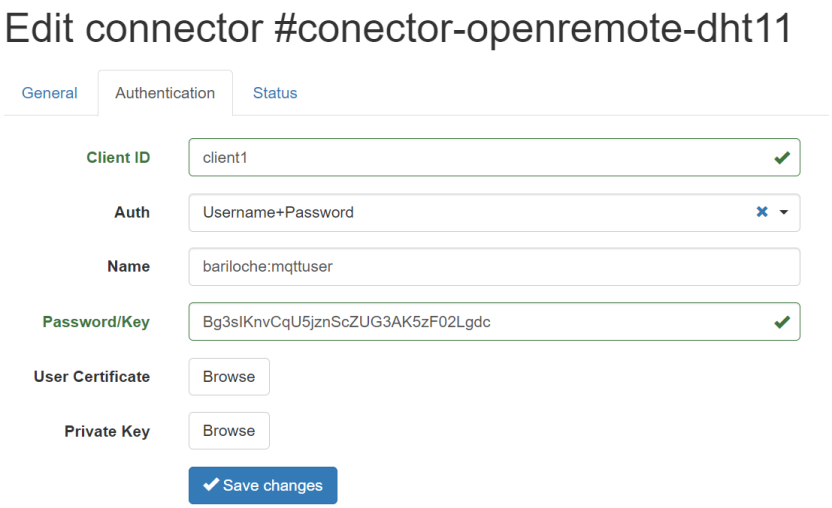

Figura 23 : Autenticación del conector para MQTT

Además, en la pestaña Authentication debemos completar con las credenciales para autenticar la comunicación. En Client ID colocamos el mismo cliente que elegimos antes, en Name deber ir el usuario de MQTT con el realm antepuesto, en nuestro caso bariloche:mqttuser, y por último, en Password la clave del usuario en cuestión.

Por último, guardamos el conector y esperamos que llegue el mensaje a la plataforma.

Figura 24 : Recepción del mensaje por MQTT en OpenRemote

Podemos ver que recibimos un JSON con tres parámetros, el estado de conexión, la temperatura y la humedad.

Conexión por WebSocket

Para lograr una conexión por WebSocket, se generó un nuevo conector en el servidor LoRaWAN, pero esta vez utilizando dicho protocolo de comunicación. En este caso, se hizo uso de un agente en la plataforma para la conexión con el servidor.

Primero, se debe configurar el servidor. Generamos un nuevo conector WebSocket en la pestaña Connectors. Seleccionamos el mismo handler y formato, ya que la información que queremos recibir es la misma y en formato JSON. En la casilla Publish Uplinks, componemos la URL en la cual se va a abrir el nuevo WebSocket, en este caso utilizamos la forma ws/uplink/BACEBACE, para hacer referencia que es un mensaje del sensor. En la opción URI, colocamos únicamente ws:.

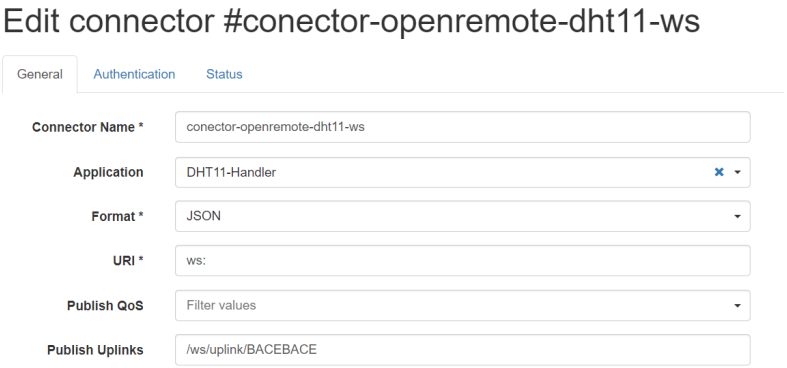

Figura 25 : Creación de un conector para WebSocket

Luego, creamos un agente en OpenRemote para invocar a la API de WebSocket. Dentro del nuevo asset, debemos configurar la URL con la que definimos en el servidor, esta vez anteponiendo ws://<IP>:8080. En nuestro caso queda:

ws://192.168.100.82:8080/ws/uplink/BACEBACE

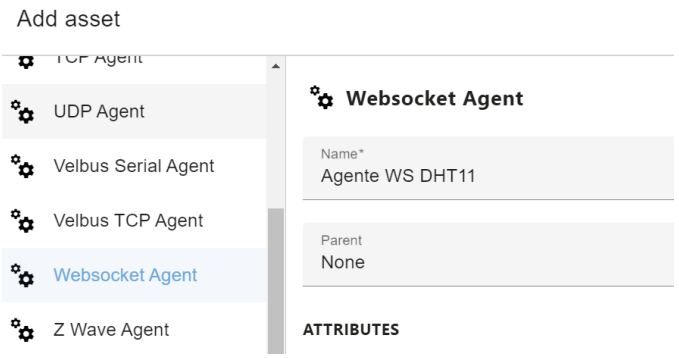

Figura 26 : Generación de un agente WebSocket

Al introducir la URL, deberíamos ver en el atributo agentStatus un mensaje de CONNECTED, dando a entender que tenemos conexión con el WebSocket generado por el servidor LoRaWAN.

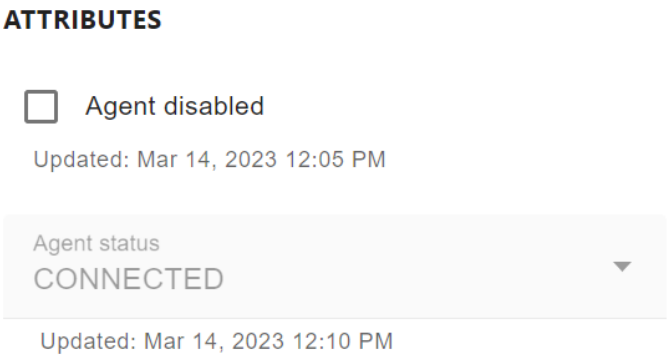

Figura 27 : Conexión con el WebSocket

Ahora, creamos nuevamente un asset de clima pero esta vez le vamos a asignar como padre al agente creado anteriormente. De esta forma, el agente puede acceder a los atributos del sensor y modificarlos.

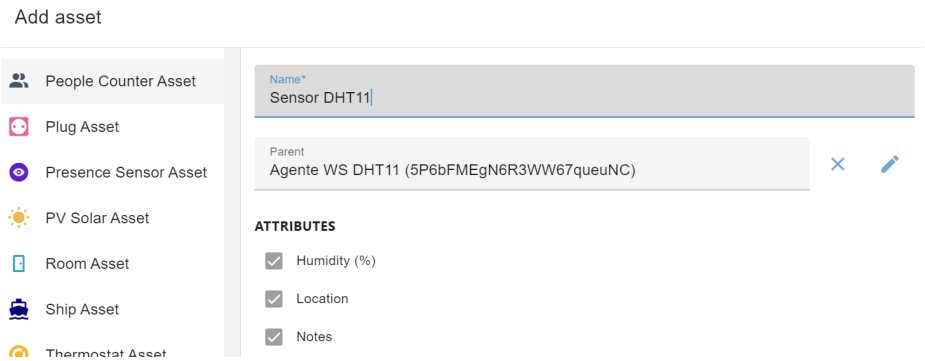

Figura 28 : Asignación de un asset como padre de otro

De nuevo, agregamos el atributo lecturaSensor de tipo JSON para recibir el mensaje, pero esta vez utilizaremos un Agent Link para conectarnos con el agente. Para esto, dentro del atributo lecturaSensor añadimos el ítem de configuración Agent Link. Ahora, seleccionamos el agente de WebSocket creado y además le agregamos el parámetro Message Match Predicate con valor Any Value. Esto hace que cualquier mensaje que se manda por el protocolo vaya a parar a este atributo. El mensaje debería empezar a llegar correctamente al atributo, en formato JSON.

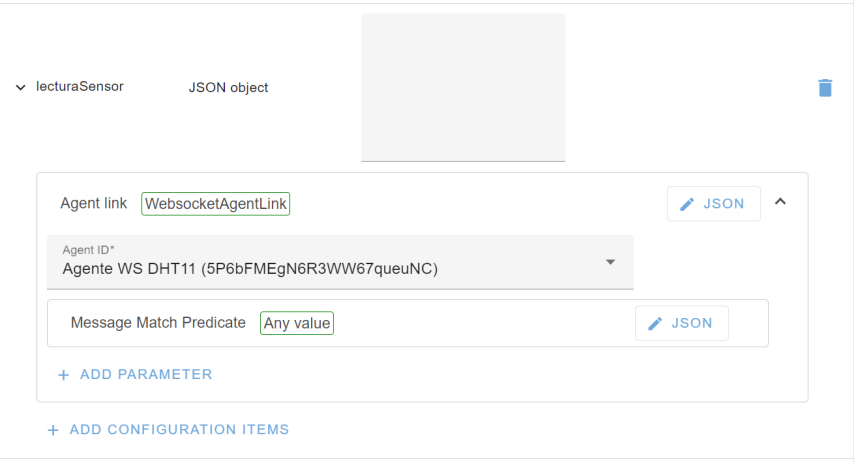

Figura 29 : Configuración de un Agent Link

Conexión por HTTP

Para este caso, no se utilizó el servidor LoRaWAN para gestionar la conexión por protocolo HTTP, ya que este no funciona como servidor HTTP, sino como cliente que postea la información. OpenRemote funciona también como cliente, por lo que necesitamos un servidor HTTP de donde sacar la información del sensor. Para ello, se utilizó un microcontrolador ESP8266, que funciona como servidor y contiene las lecturas del sensor DHT11.

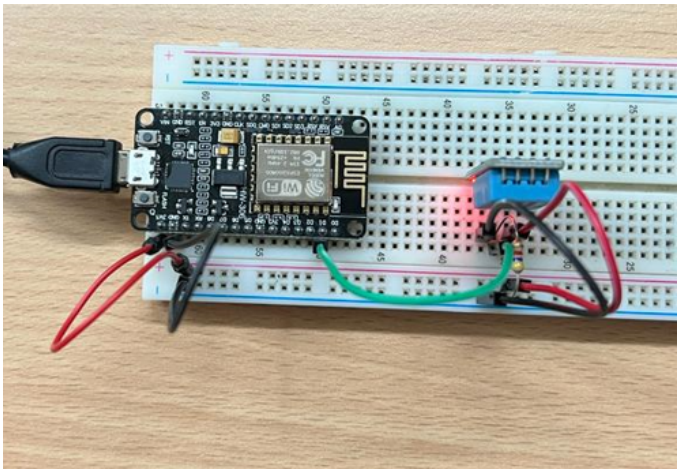

Figura 30 : ESP8266 con el sensor DHT11 conectado

El código corriendo en el microcontrolador se encuentra en el repositorio. Básicamente, se genera un servidor HTTP al cual se accede mediante el IP del dispositivo. Aquí, el ESP8266 postea un JSON con los mismos datos que se recibían antes por LoRa, estado de conexión, temperatura y humedad. La IP del dispositivo se imprime por puerto serial, para saber donde conectarse.

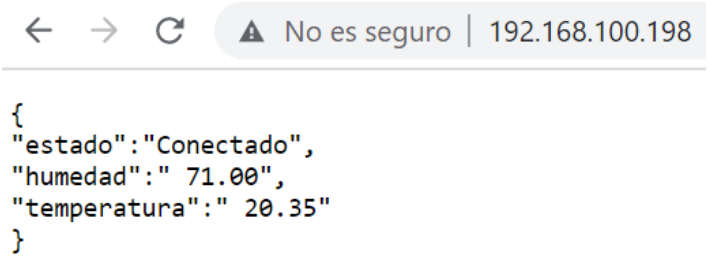

Figura 31 : Servidor HTTP generado por el ESP8266

En la plataforma, crearemos un nuevo agente, esta vez uno HTTP que funciona como cliente. Dentro de la configuración, colocamos la URL con el IP del dispositivo en el campo Base URL. La casilla Agent Status nos dará a saber si el agente pudo conectarse al servidor con el valor CONNECTED.

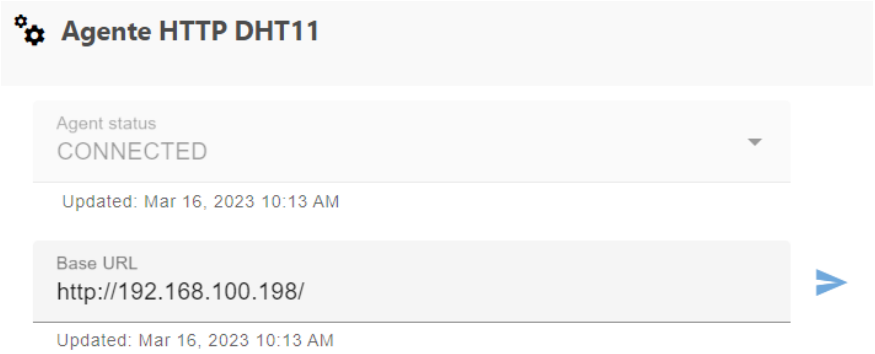

Figura 32 : Conexión de Agente al servidor HTTP

Ahora, generamos nuevamente un asset sensor, que será hijo de este agente. Creamos el mismo campo lecturaSensor con tipo de valor JSON y le asignamos un Agent Link con el agente HTTP creado anteriormente. Además, debemos agregar a este Agent Link el periodo de tiempo con el que se solicitan los datos al servidor. Esto lo logramos mediante el ítem de configuración Polling Millis, que corresponde a tiempo en milisegundos. En este caso, lo configuramos con el número 1000.

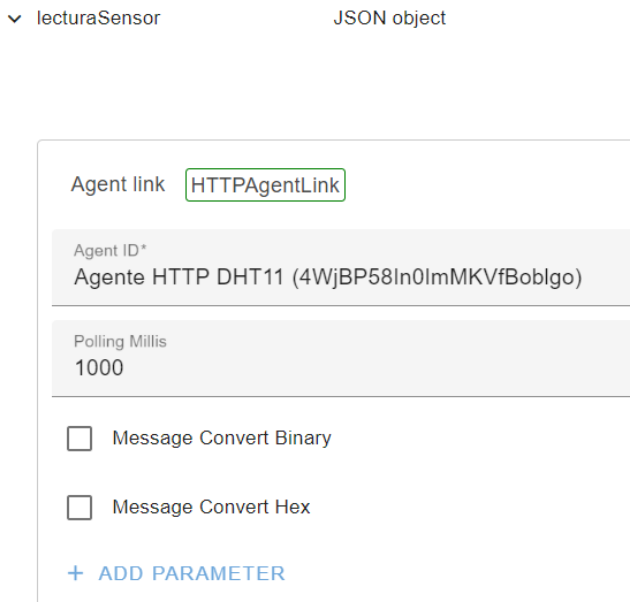

Figura 33 : Configuración del Agent Link HTTP

En el caso que los datos no se encuentren en el URL base del servidor, también podemos agregar un item Path, que añade la dirección exacta donde se aloja la información. Finalmente, deberíamos ver como llega el JSON al atributo del OpenRemote, actualizándose cada 1 segundo.

Conexión por TCP y UDP

Para conectarse mediante un socket TCP o UDP, se utilizó el mismo ESP8266 pero esta vez abriendo un socket para que OpenRemote se conecte como cliente y pueda recibir los datos. En este socket, el microcontrolador envía los datos periódicamente mientras algún cliente esté conectado. El firmware correspondiente se encuentra en el repositorio.

Teniendo el socket abierto, debemos crear un agente dentro de la plataforma OpenRemote para conectarnos como clientes. Para esto, creamos un agente TCP o UDP, dependiendo del tipo de socket abierto. En este caso, abrimos uno de tipo TCP. Dentro de este agente, deberemos configurar el parámetro Host con la IP del socket, en este caso la IP del microcontrolador, y el puerto. Además, debemos agregar el atributo Message Delimiters para saber con qué caracter termina un mensaje recibido. En este caso, enviamos un JSON con un \n al final, por lo que pondremos \n como delimitador. El casillero Agent Status nos dirá si llegamos al socket con el valor CONNECTED.

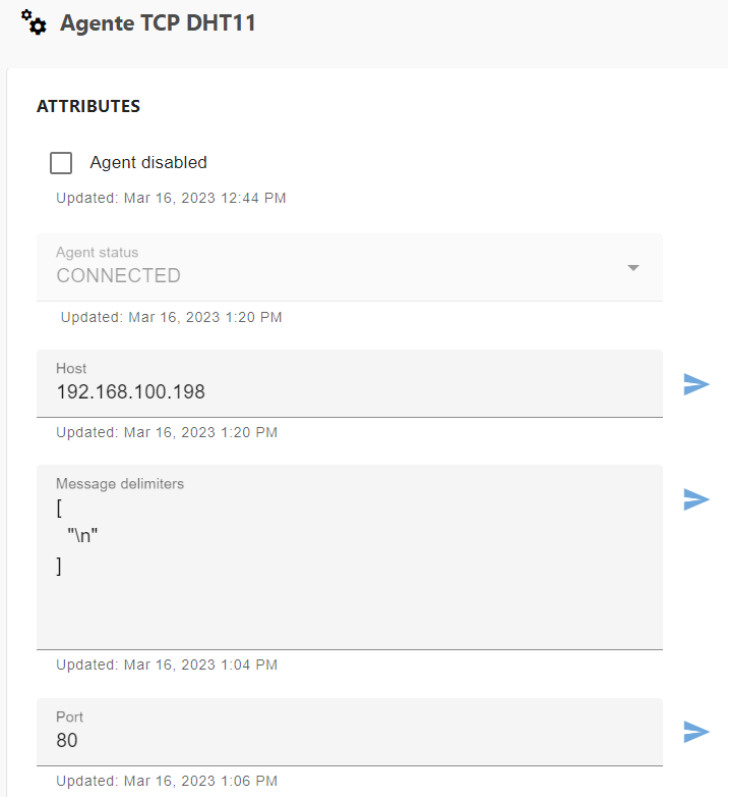

Figura 34 : Configuración del agente TCP

Ahora, generamos nuevamente un asset sensor, que será hijo de este agente. Creamos el mismo campo lecturaSensor con tipo de valor JSON y le asignamos un Agent Link con el agente TCP creado anteriormente. Además, debemos agregar a este Agent Link el ítem de configuración Message Match Predicate con el valor Any value, para que cualquier mensaje que se mande por el protocolo vaya a parar a este atributo. El mensaje debería empezar a llegar correctamente al atributo, en formato JSON.

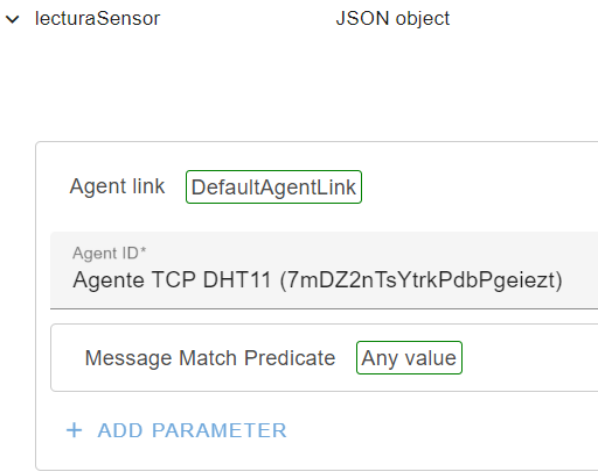

Figura 35 : Configuración del Agent Link TCP

Demostración de datos en el mapa

Hasta ahora, únicamente se recibe un JSON con los datos por varios protocolos de comunicación. Este formato de datos no es ideal para visualizar, así que se exploró la capacidad de mostrar datos en el mapa de la plataforma. Se trabajó únicamente sobre los datos recibidos por MQTT, pero lo mismo puede ser llevado a cualquiera de los otros protocolos, ya que se recibió exactamente lo mismo.

Primero, ubicamos el asset sensor en el mapa mediante el atributo location. Al seleccionar esta opción, deberemos elegir en una ventana emergente la posición del icono en el mapa.

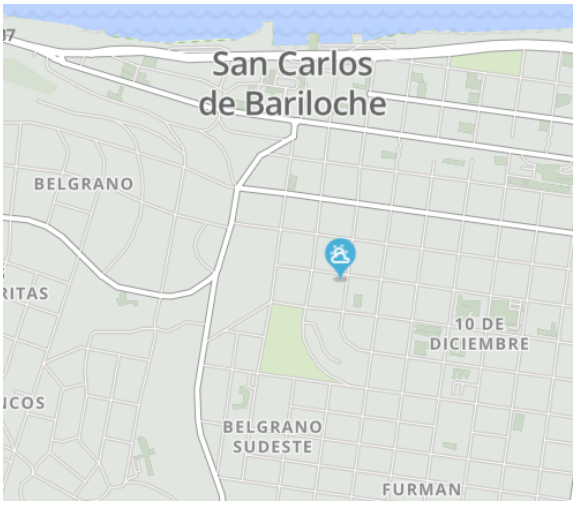

Figura 36 : Sensor ubicado en el mapa

Si presionamos el icono en el mapa, veremos el atributo lecturaSensor sin ningún valor, ya que la plataforma no puede mostrar un tipo de dato JSON. Lo que haremos es enviar los datos del JSON a otros atributos, los cuales puedan ser mostrados en el mapa correctamente.

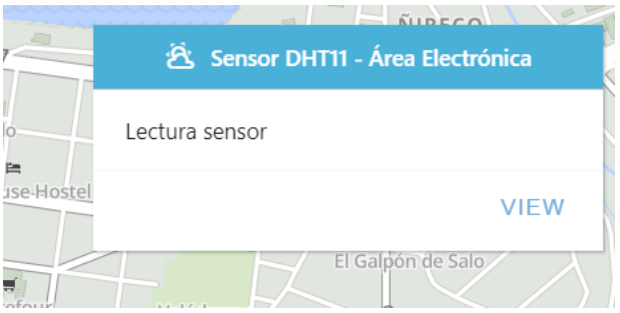

Figura 37 : Datos dentro del icono del mapa

Primero, habilitamos la opción para que se deje de ver el atributo lecturaSensor en el mapa, mediante un ítem de configuración en el atributo llamado Show on dashboard. Para enviar los datos del JSON recibido a otros atributos, se debe usar otro ítem llamado Attribute links, el cual comunica atributos entre assets de la plataforma. Los atributos en cuestión serán temperature y humidity, que ya están definidos por defecto en el tipo de asset clima, y además un nuevo atributo que crearemos, llamado estado, de tipo texto.

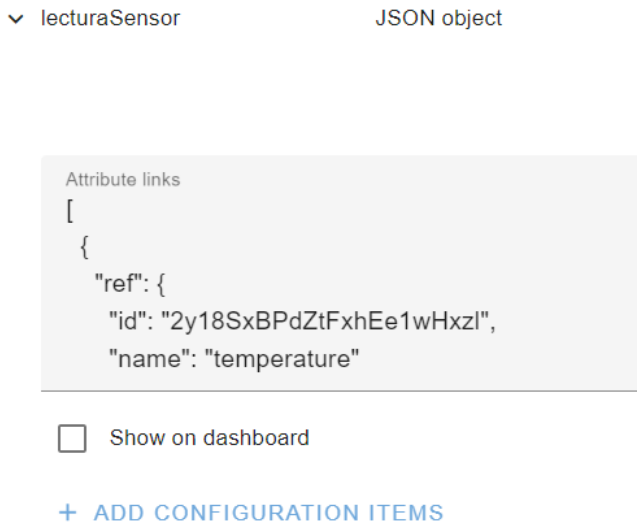

Figura 38 : Configuración de Attribute links para comunicar atributos

Dentro de Attribute links, debemos escribir en formato JSON como queremos que se distribuya la información hacia otros atributos. En este caso, enviaremos el valor de temperatura en el JSON al atributo temperature, el valor de humedad a humidity y por último el valor de estado de conexión a estado. Utilizaremos el caso de temperatura para entender cómo comunicar atributos.

```
[
  {
    "ref": {
      "id": "2y18SxBPdZtFxhEe1wHxzl",
      "name": "temperature"
    },
    "filters": [
      {
        "type": "jsonPath",
        "path": "$.temperatura",
        "returnFirst": true,
        "returnLast": false
      }
    ]
  },
```

Figura 39 : Estructura y contenido del JSON dentro de Attribute links

Dentro de “ref” debemos definir el atributo a conectar con su nombre e id del asset donde se encuentra, en este caso el id del asset sensor y nombre temperature. En “filters” vamos a filtrar el JSON para obtener solo la temperatura, colocando el tipo de dato y el path a la temperatura. En nuestro caso, el tipo de dato JSON y el path es temperatura, ya que así se llama en el JSON. Repetimos lo mismo para los otros dos parámetros, humedad y estado. Deberíamos ver como llegan los datos directo a los atributos mencionados.

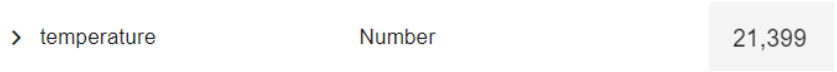

Figura 40 : Recepcion del valor de temperatura en el atributo temperature

Ahora, configuramos etiquetas en los atributos, las cuales corresponden a los nombres que vamos a ver en el mapa. En el caso de temperature, le pondremos el nombre Temperatura.

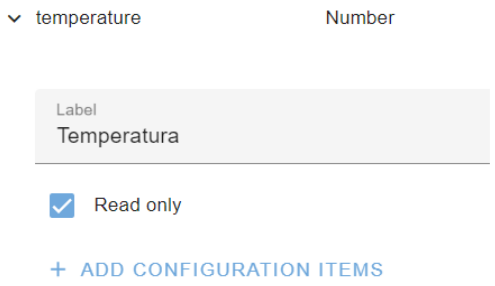

Figura 41 : Colocación de una etiqueta

Finalmente, vemos como se ven reflejados los atributos en el icono del mapa. Podemos observar que la temperatura y humedad tienen sus unidades y cantidad de dígitos configurada. Esto se debe a que, en el asset clima que usamos de base para el sensor, esto ya se encuentra definido por defecto. En caso de querer cambiar la unidad, deberíamos agregar un ítem de configuración a los atributos llamado Units, donde podemos realizar dicho cambio. Por otro lado, la cantidad de dígitos que se desean mostrar puede ser configurado con otro ítem llamado Format.

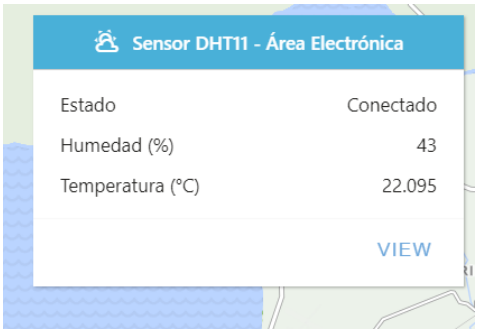

Figura 42 : Datos en el mapa

Implementación de una regla

Para una correcta visualización del dispositivo en el mapa, se decidió crear una regla que permita saber cuando el sensor se ha desconectado o ha dejado de comunicarse. Para ello, se escribirá en el atributo estado el valor Desconectado, en caso que el sensor no haya reportado después de cierto tiempo.

Lo que debemos definir inicialmente, son los ítems de configuración en el atributo estado, para que este pueda interactuar con el motor de reglas. Estos son Rule event y Rule state.

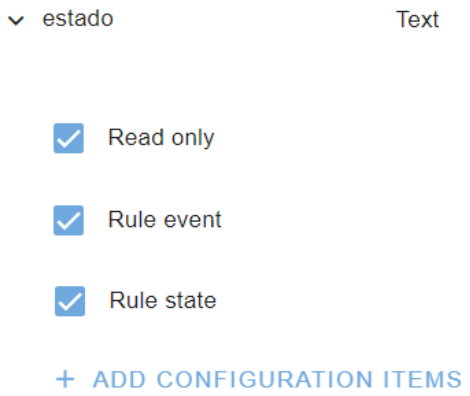

Figura 43 : Configuración para la interacción con el motor de reglas

Se decidió implementar la regla con Groovy, para conocer el lenguaje y la plataforma más a fondo. Dentro de la pestaña Rules de OpenRemote, seleccionamos la opción para crear una nueva regla de tipo Groovy. Esta solo interactúa en el realm Bariloche, por lo tanto definimos su scope al realm.

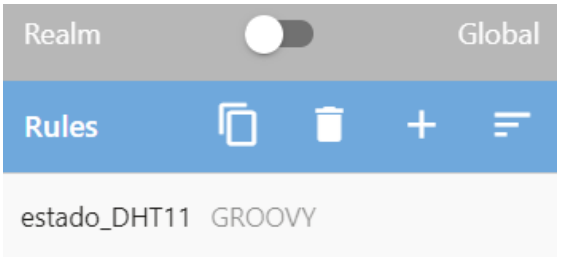

Figura 44 : Creación de una nueva regla

Se nos presentará una suerte de IDE, donde podremos programar la regla.

Rule name*

estado_DHT11

ALWAYS ACTIVE

```
1  package DHT11.rules
2
3  import org.openremote.model.query.AssetQuery
4  import org.openremote.manager.rules.RulesBuilder
5  import org.openremote.model.asset.*
6  import org.openremote.model.attribute.*
7  import org.openremote.model.value.*
8  import org.openremote.model.rules.*
9  import java.util.logging.*
10
11 Logger LOG = binding.LOG
12 RulesBuilder rules = binding.rules
13 Assets assets = binding.assets
14
15 rules.add()
16         .name("Estado DHT11")
17         .when(
18             {facts ->
19             facts.matchAssetEvent(
20                 new AssetQuery()
21                     .ids("2y18SxBPdZtFxhEe1wHxzl")
22                     .attributeName("estado")
23             ).count() == 0 && !facts.matchFirst(String, {it == "VirtualSensor2"}).isPresent()
24         })
25         .then(
26             {facts ->
27             assets.dispatch(new AttributeEvent("2y18SxBPdZtFxhEe1wHxzl", "estado", "Desconectado"))
28             facts.putTemporary("PT1M", "VirtualSensor2")
29         })
```

Figura 45 : Regla para el estado del sensor

Primero, se definen las librerías que se utilizaran en la regla, utilizando la palabra dedicada import. Debemos crear un objeto de tipo RulesBuilder para crear la nueva regla, y otro de tipo Assets para interactuar con los assets que queremos modificar. Con el método add(), añadimos la nueva regla, defiendo su nombre, que evento detona la regla (when) y que hace cuando se cumpla este evento (then).

En el método when(), definimos el evento. En este caso, preguntamos si el estado del sensor no fue actualizado hace cierto tiempo, utilizando un ítem de configuración llamado Rule event expires, en nuestro caso definido como 1 minuto.

estado Text

Read only

Rule event

Rule event expires
PT1M

Rule state

+ ADD CONFIGURATION ITEMS

Figura 46 : Configuración de Rule event expires

Este parámetro permite definir un timer, el cual empezará a correr desde la última actualización del atributo. Cuando se setea el timer, generamos además un fact (hecho) a este evento, el cual expirará cuando el tiempo termine. Por lo tanto, cuando la cantidad de facts llegue a 0, significa que el timer expiró y que la regla debe actuar.

También podemos ver otra condición, la cual pregunta por la existencia de un fact en virtual sensor. Este virtual sensor se ocupa únicamente de que la regla no se ejecute sin parar, ya que no hay nada que detenga que se dispare de nuevo. Para ello, la primera vez que se detona la regla, se le define un fact temporal al virtual sensor, el cual expira en 1 minuto. Por lo tanto, la regla se ejecuta cada 1 minuto, dado que el sensor se haya desconectado.

En el método then(), solicitamos que se escriba Desconectado en el estado del sensor mediante el método dispatch del objeto asset. Luego, creamos el fact temporal de virtual sensor discutido anteriormente.

Con la regla escrita, la habilitamos con el botón Enable y probamos que se modifique el estado, cuando haya una desconexión por más de 1 minuto.

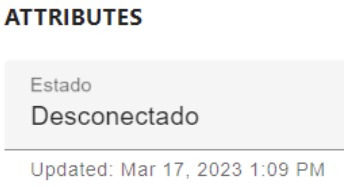

Figura 47 : Regla modificando el estado a desconectado

Para ver el estado del sensor rápidamente, asignaremos color al icono en el mapa dependiendo si está conectado o no. Para ello, debemos modificar el archivo *manager_config.json* dentro del directorio */deployment/manager/app*:

```
{
  "pages": {
    "map": {
      "markers": {
        "WeatherAsset": {
          "attributeName": "estado",
          "showLabel": false,
          "showUnits": false,
          "colours": {
            "type": "string",
            "Conectado": "39B54A",
            "Desconectado": "C1272D"
          }
        }
      }
    }
  },
  "realms": {
    "default": {
```

Figura 48 : Modificación de *manager_config.json*

Esta modificación permite que se asigne el color del icono a verde, si se encuentra conectado, y rojo en caso contrario. Esto se logra indicando a OpenRemote el nombre del atributo cuyo valor se utilizará como parámetro para el cambio, y el tipo de asset en cuestión. En nuestro caso, el asset es de tipo clima y el atributo responsable del cambio es estado.

Para ver el cambio, debemos reiniciar la plataforma y loguearnos con un usuario que no sea el administrador.

Creación de un usuario

Hasta el momento, todos los cambios que hicimos en la plataforma fueron utilizando al usuario Admin, el cual tiene la totalidad de permisos de lectura y escritura. Para el acceso restringido y uso de la plataforma, se necesita de la creación de un usuario regular. Además, algunos cambios que se realizan sobre la plataforma no son visibles por el administrador, como la coloración de iconos vista anteriormente.

Crearemos un usuario regular, el cual solo tendrá acceso a las mediciones del sensor por MQTT, teniendo permiso únicamente de lectura. Para ello, vamos a la pestaña usuarios en la plataforma y creamos un nuevo Regular User.

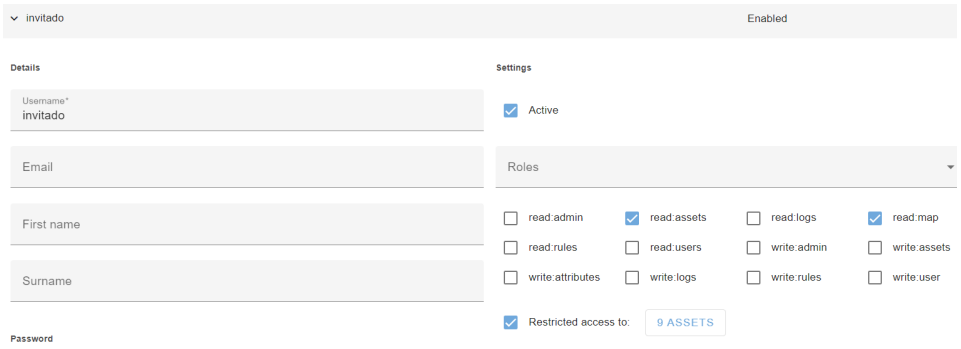

Figura 49 : Creación de un nuevo usuario regular

Le damos un nombre y password, en nuestro caso invitado/invitado, y le damos solo permiso de lectura de assets y de mapa. Además, en la opción Restricted access to, le damos accesos únicamente a la instancia del sensor que se comunica por MQTT. Por último, guardamos el usuario.

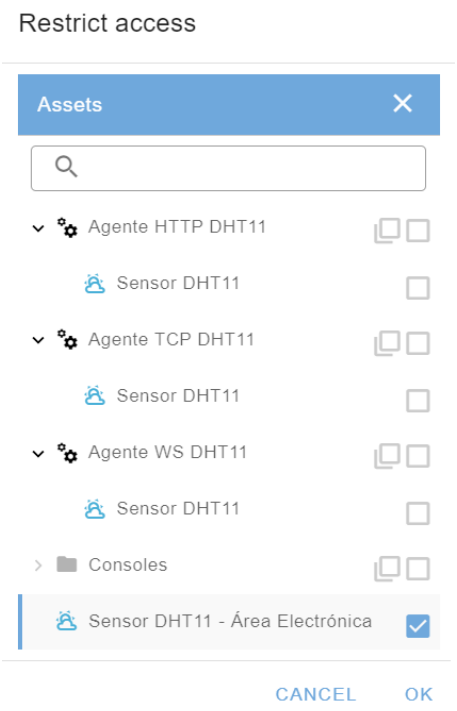

Figura 50 : Restricción del acceso a assets para el usuario

Ahora, agregamos el ítem de configuración Access restricted read a los atributos que queremos que vea el usuario, en este caso la temperatura, la humedad y el estado.

Para loguearnos con este nuevo usuario, debemos ingresar por algún realm que no sea el master mediante el siguiente URL:

https://{IP}/manager/?realm={REALM}

Reemplazando la IP por el host donde se encuentra la plataforma y REALM por algún realm que no sea el master, en nuestro caso Bariloche.

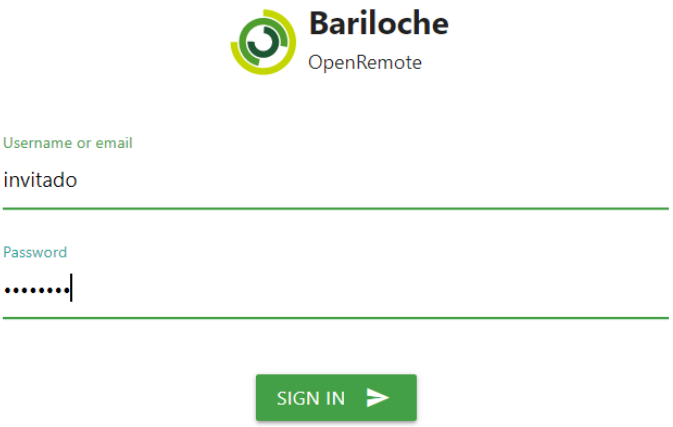

Figura 51 : Ingresando a un realm específico

Al entrar, veremos que solo podemos acceder a la instancia de MQTT del sensor, y no podemos modificar nada, únicamente leer como se reciben los datos y el estado. Además, veremos el color del icono en el mapa, dependiendo de su estado de conexión.

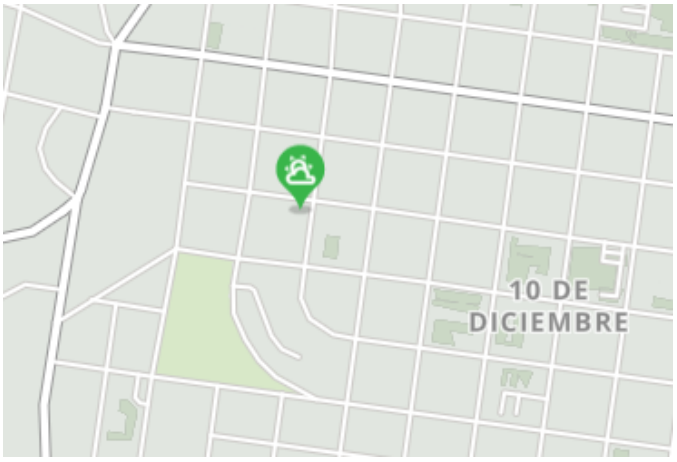

Figura 52 : Color en los iconos del mapa

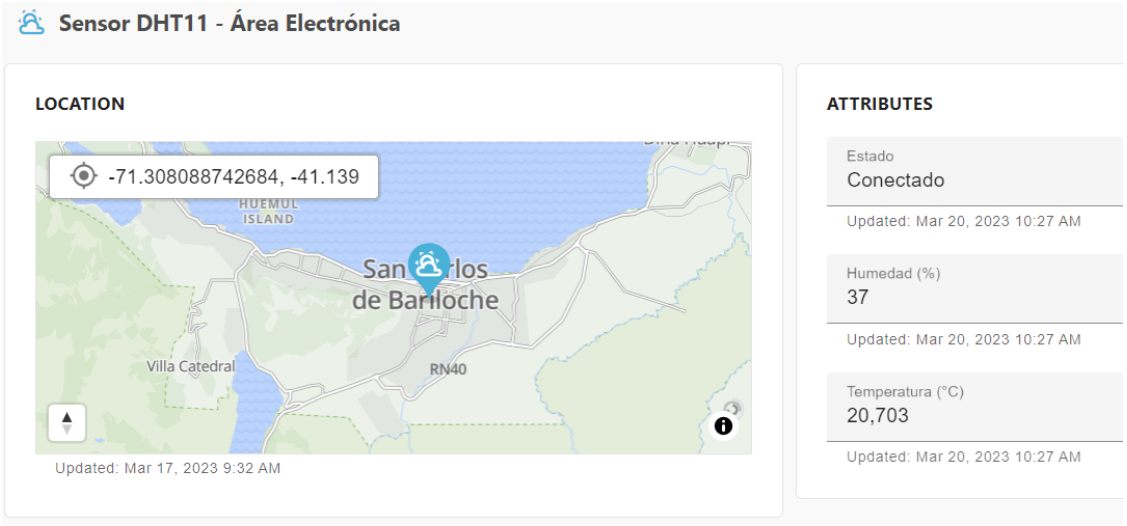

Figura 53 : Acceso restringido al sensor

Conclusiones

Conocer la plataforma OpenRemote e implementarla fue un desafío muy interesante, ya que al ser un software reciente, la documentación es un poco escasa. Se recurrió al foro principal de la plataforma para despejar dudas, como también al GitHub que contiene varios tutoriales útiles. Al ser un proyecto en constante evolución, la documentación fue variando al mismo tiempo que se iba desarrollando, generando algunos conflictos.

Se lograron completar las 200 horas de manera exitosa, haciendo que la plataforma se conecte por varios protocolos de comunicación. Se implementó la red LoRa que existe en la empresa, ideal para conectar sensores a OpenRemote. Además, se desarrolló firmware para dos placas distintas, con el fin de poder conectar un sensor de temperatura y humedad a la plataforma. Se aprendió a implementar OpenRemote con sus funciones básicas, como es el manejo de usuarios, comunicación con el exterior, motor de reglas, configuración de mapa, etc..

Hubo un gran seguimiento por parte de la empresa, dando el espacio para consultas y prestando ayuda cuando se necesitaba. ALTEC dispuso de la información necesaria para aprender a utilizar el server LoRaWAN, algo indispensable para la práctica.

Puedo decir que quedan varias cosas por aprender de OpenRemote, como conocer más a fondo el motor de reglas y la herramienta para gráfico de datos integrada, la cual usa una base de datos para recolectar la información a largo plazo. Se ha logrado integrar el conocimiento adquirido a lo largo de la práctica y se ha ampliado el conocimiento acerca de esta plataforma IOT.

Anexo 1 : Repositorio del proyecto

https://github.com/Practica-Profesional-Supervisada/OpenRemote

Dentro de este repositorio se encuentra todo el código desarrollado durante la práctica. Incluye firmwares y función Handler de LoRaWAN server.