



# Whitecatboard.org

## IoT DEVKIT MANUAL

Rev 1.0 (ES) 2017

## Índice

1. Introducción a Whitecatboard.org
2. Descripción y componentes del Kit
  - 2.1. Sensores
  - 2.2. Actuadores
  - 2.3. Placa ESP32N1
  - 2.4. Placa DEVKIT para ESP32N1
3. Entorno de Desarrollo Whitecat IDE
  - 3.1. Características generales e Instalación
  - 3.2. Funciones del IDE
4. LuaRTOS Sistema Operativo del ecosistema Whitecatboard
  - 4.1. Diseño
  - 4.2. Funciones principales
5. Conectividad (WiFi / Bluetooth / LoRaWAN)
6. LoRaWAN Introducción y Administración en The Things Network (TTN)
  - 6.1. The Things Network (TTN)
7. La “Nube” (IBM Bluemix y NodeRED)
  - 7.1. IBM Bluemix
  - 7.2. Node-RED
8. Casos Prácticos
  - 8.1. Caso Práctico 1: Configuración de un sensor
  - 8.2. Caso Práctico 2: Envío de información a través de la red LoRaWAN
  - 8.3. Caso Práctico 3: Recepción de información a través de la red LoRaWAN
  - 8.4. Caso Práctico 4: Publicación de información mediante MQTT
  - 8.5. Caso Práctico 5: Suscripción y recepción de información mediante MQTT
  - 8.6. Caso Práctico 6: Lógica en el Nodo “Edge Computing”
  - 8.7. Caso Práctico 7: Lógica en la Nube “Cloud Computing”
  - 8.8. Caso Práctico 8: Despliegue Nodo Tipo
9. Anexos
  - 9.1. Esquématicos
  - 9.2. Documentación Técnica
  - 9.3. Referencias y enlaces

## 1. Introducción a Whitecatboard.org

En primer lugar gracias por adquirir un kit de desarrollo de Whitecatboard.org para el ecosistema IoT (Internet de las Cosas). Con este kit y la documentación que le acompaña, trataremos de introducir al usuario en el ecosistema IoT o ampliar sus conocimientos, en caso de que este no sea su primer contacto con esta tecnología.

El kit ha sido diseñado teniendo en mente su empleo tanto en un entorno académico-profesionalizador así como su uso en entornos más industriales. Asimismo, el producto también constituye una buena herramienta para aquellas personas que deseen adentrarse o ampliar sus conocimientos de IoT de modo autodidacta.

Esta amplia gama posible de destinatarios del producto, es consecuencia del trabajo coordinado de diferentes profesionales que han participado en el diseño tanto del kit como de su documentación. Así, en su elaboración han colaborado:

-Ingenieros especializados en el desarrollo de soluciones de software, los cuales han diseñado el **sistema operativo LuaRTOS** sobre el que se sustenta el entorno de desarrollo, así como el funcionamiento de los diferentes elementos del kit de un modo robusto y al mismo tiempo sencillo por parte del usuario.

Así mismo se ha diseñado un **entorno de desarrollo basado en bloques** que permite una introducción al mundo de la programación de un modo más fácil y ameno. No obstante, el entorno también permite la posibilidad de llevar a cabo la programación con código en formato texto.

-Ingenieros especializados en el desarrollo de soluciones de hardware, los cuales han diseñado la **placa ESP32N1** y la **placa DEVKIT** para la **ESP32N1** para facilitar la interconexión de los diferentes dispositivos. Asimismo, también han **seleccionado los diferentes elementos del kit** con el objetivo de facilitar el uso del mismo por parte de los usuarios finales.

-Docentes que han **revisado toda la documentación y los casos prácticos** que acompañan el kit, con el objetivo de maximizar el aprendizaje por parte del usuario. Como se podrá comprobar, los casos desarrollados están ordenados en orden creciente de complejidad, de modo que con su implantación, el usuario pone en práctica y utiliza los procedimientos y herramientas que se consideran más habituales en la implantación de la tecnología IoT.

El desarrollo de todo este trabajo ha tenido como escenario la Fundación para el Fomento de la Sociedad del Conocimiento (Citilab) de Cornellà y ha sido llevado a cabo de forma conjunta por **CSS Ibérica** especialmente en la parte de hardware, **Iberoxarxa** en lo que corresponde a software y por parte de personal del propio **Citilab** los cuales han participado principalmente como asesores pedagógicos, selección de componentes y banco de pruebas.

## 2. Componentes del Kit

El kit de desarrollo de Whitecatboard.org para IoT viene organizado en una caja de plástico y se presenta tal como se muestra a continuación:

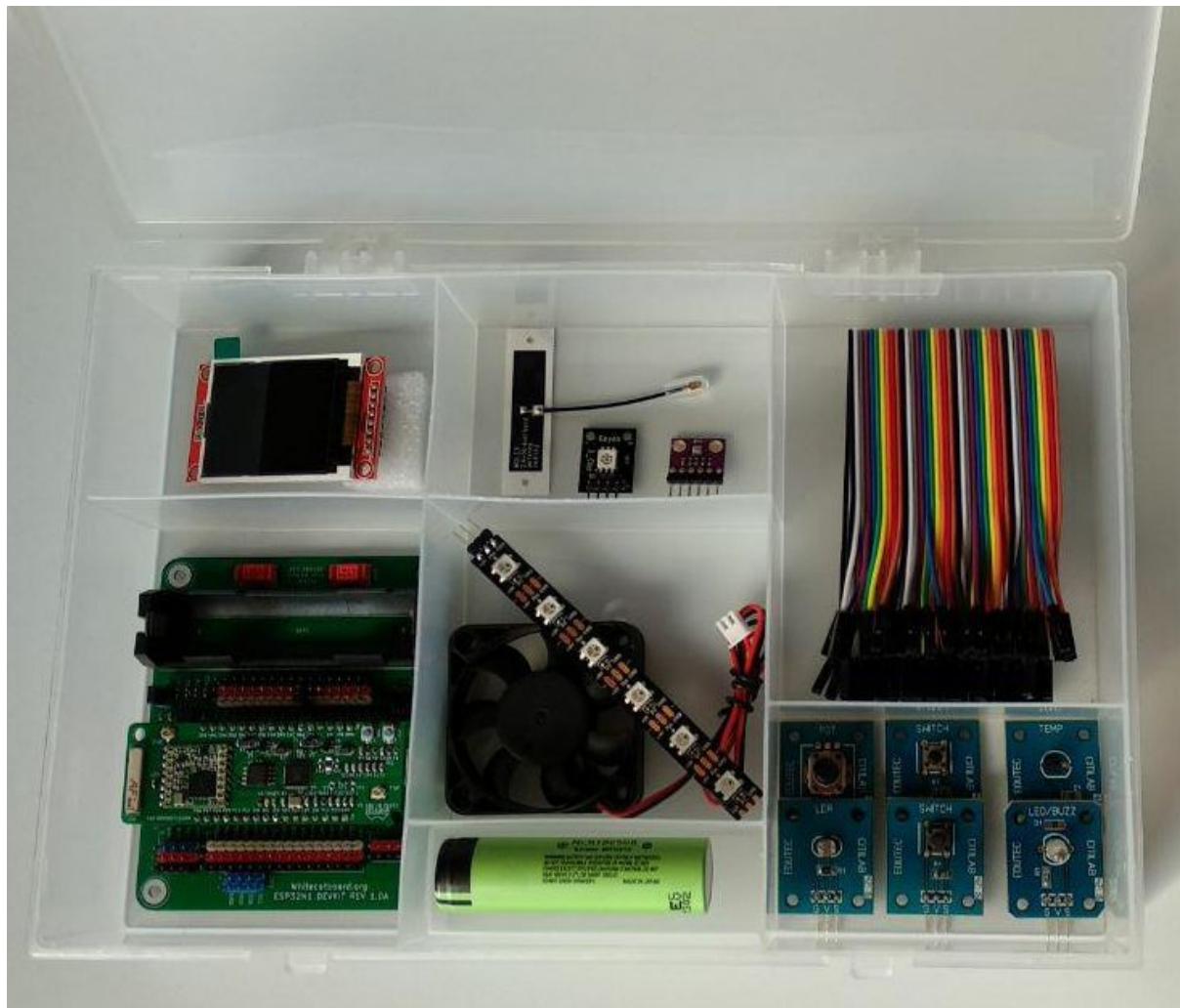


Fig-1. Kit IoT de Whitecatboard.org

En los apartados sucesivos, se detallan los componentes del kit, los cuáles se dividen en sensores, actuadores, placa ESP32N1 y placa DEVKIT para ESP32N1.

### 2.1. Sensores

Un sensor es un elemento que es capaz de medir magnitudes físicas o químicas y transformarlas en un tipo de señal eléctrica que se puede medir mediante el empleo de una entrada. Así por ejemplo, una sonda de temperatura sería un sensor que indicaría el valor (magnitud) de una propiedad física como es la temperatura.

El mundo físico es analógico motivo por el cual, cuando se quieren medir propiedades mediante un sensor, en muchas ocasiones es preciso el uso de conversores analógicos a digital ADC.

Estos reciben una señal analógica correspondiente a una magnitud y la codifican a unos y ceros (10011001). La cantidad de unos y ceros que pueden emplear para dar un valor de salida se denomina la resolución, de modo que a mayor cantidad de unos y ceros mayor resolución.

Como se podrá comprobar, el kit dispone de varios sensores que emplean señales analógicas (variables en magnitud) y digitales (unos y ceros). Estos tipos de señales disponen de unas entradas específicas en la placa DEVKIT ESP32N1, que se explicarán posteriormente cuando se explique dicha placa.

Algunos sensores y actuadores, tienen inscritas las siglas G V S, que corresponden a G (Masa), V (Voltaje) y S (Señal).

Los sensores que incorpora el kit actualmente son:

#### **- Pulsador**

El pulsador es un simple interruptor momentáneo de modo que mientras se mantiene pulsado, se permite que el paso de corriente a través de él. Normalmente este tipo de dispositivos se conectan por defecto a un valor positivo (pull-up) o negativo (pull-down).

Tal como se muestra en la figura, estos dispositivos se asocian a una resistencia para evitar la presencia de una señal no deseada.



Fig-2. Pulsador momentáneo (switch)

#### **- Potenciómetro**

El potenciómetro es una resistencia (dificultad al paso de la corriente) de valor variable que adquiere un valor u otro en función, por ejemplo, de la posición de su eje (es el caso de la versión suministrada con el kit). El valor máximo de resistencia que tenemos en el kit es de  $10K\ \Omega$  que es un valor bastante común en este tipo de dispositivos. Según se conecten sus entradas y salidas, y si el movimiento es en el sentido de las agujas del reloj o en sentido contrario, se producirá un incremento o decremento en el valor de la resistencia.

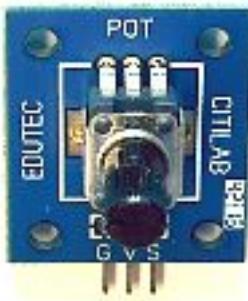


Fig-3. Potenciómetro

#### - Fotoresistencia (LDR)

Del mismo modo que el potenciómetro que hemos visto anteriormente, la LDR (fotoresistor) es una resistencia de valor variable pero en lugar de variar su valor por una actuación mecánica como en el caso anterior, lo realiza en función de la cantidad de luz que recibe (fotones). Estos interactúan con la superficie de la fotoresistencia haciendo que su resistencia disminuya con el aumento de la intensidad de luz que incide sobre su superficie.

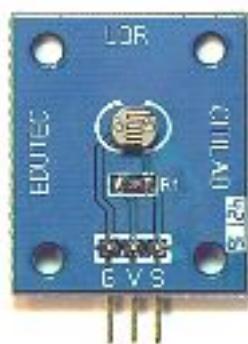


Fig-4. Fotoresistor (LDR)

### - LM35 Sensor de Temperatura

LM35 es un sensor encapsulado en el mismo formato que un transistor. Es un pequeño circuito integrado que toma la temperatura y da una señal eléctrica de tipo analógico proporcional a la temperatura medida.

Este tipo de sensor debe ser conectado a una entrada marcada como AIN(X) de la placa DEVKIT y requiere una entrada de alimentación de 2.7 V a 20 V.

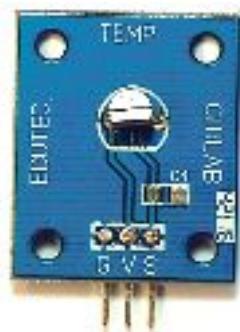


Fig-5. LM35 Sensor de Temperatura

### - BME280 Sensor I2C de Temperatura, Presión Atmosférica y Humedad

El sensor BME280 es un encapsulado especial con un circuito integrado interno diseñado por BOSCH, que mediante el bus de comunicaciones I2C entrega la lectura internamente procesada de Temperatura, Presión Atmosférica y Humedad.

En el bus I2C podemos disponer de un maestro y varios sensores esclavos. Al usar este tipo de bus, no se recomienda que la longitud del cable que unen los diferentes esclavos sea muy larga, ya que I2C se diseñó y se emplea como un bus de comunicaciones entre dispositivos que se encuentran normalmente en la misma placa de circuito impreso.

Las señales del bus I2C son SDA (que es la línea de datos) y SCL (que es la línea de reloj). A diferencia del bus 1-Wire, el bus I2C sincroniza las transacciones a través de la línea de reloj SCL mediante pulsos ON/OFF. Así, para acceder a los diferentes dispositivos de un bus I2C, se emplea una dirección única de cada dispositivo en el bus, de forma que éste escuchará peticiones en dicha dirección y cada dispositivo será el único que responderá.



Fig-6. Sensor BME280

## 2.2. Actuadores

Un actuador realiza la labor contraria al sensor de modo que recibe una señal eléctrica y muestra o realiza una acción (actúa) en base a dicha señal eléctrica.

El kit dispone de varios actuadores que emplean señales analógicas y digitales. Estos tipos de señales disponen de unas salidas específicas en la placa DEVKIT ESP32N1, que se explicarán más adelante cuando se hable de dicha placa.

Actualmente, los actuadores que incorpora el kit son:

### - Pantalla TFT

La pantalla es uno de los actuadores más versátiles ya que permite mostrar información de una manera instantánea. La pantalla de 1.8" incluida en el kit dispone de 65535 colores, una resolución de 128x160 píxeles y conectividad de alta velocidad mediante puerto SPI.

El puerto SPI es más rápido que el puerto I2C ya que permite velocidades mucho más elevadas. Las líneas de control de dicho puerto son SCK (pulso de reloj), MISO (entrada master), MOSI (salida master) y CS (chip select). Se puede ampliar más información sobre este tipo de bus en [https://es.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://es.wikipedia.org/wiki/Serial_Peripheral_Interface).

El puerto SPI es similar al del I2C con la diferencia que la selección de cada uno de los elementos en el bus SPI, se realiza haciendo uso de una línea CS (chip select) por dispositivo. Así, un dispositivo con la señal CS en estado bajo (OFF) indicará que las instrucciones del bus son para él, mientras que el resto de dispositivos estarán en estado alto (ON). Por lo tanto, este tipo de bus requerirá de un cable adicional al I2C, por cada dispositivo conectado.



Fig-7. Pantalla TFT incluída en el kit

### - LED RGB

Se ha incluido con el kit un LED RGB. Este tipo de LED en realidad incorpora 3 LEDS en su interior. El funcionamiento del LED (Light Emitting Diode) es idéntico al de un diodo que solo permite el paso de corriente en un sentido pero con la diferencia que ese paso de corriente genera un salto de electrones en el elemento semiconductor produciendo luz.

El LED RGB dispone de un cátodo (NEGATIVO -) común y tres ánodos (POSITIVOS +) donde se conectan las salidas de la placa, de modo que para controlar un solo LED RGB serán necesarios 4 cables.



Fig-8. LED RGB

Las luces LED actualmente están reemplazando a todos los sistemas de iluminación tradicionales ya que son mucho más eficientes a nivel de consumo así como en durabilidad.

### - LED Direccionable RGB

En el kit se ha incluido otro tipo de LED RGB cuyo esquema principal de funcionamiento es idéntico al primero: opera como un LED RGB normal, pero la manera de seleccionar el color y de encender o apagar el LED se realiza mediante un protocolo 1-Wire. En el apartado de anexos se suministra más información acerca de este protocolo.

Este protocolo permite controlar un gran número de LEDs simplemente con 3 cables, 2 de alimentación y uno de control. Así, en el caso anterior sería muy elevado el número de líneas de salida para controlar el mismo número de LEDs. En cambio, en el caso del kit se controlarán 6 LEDs solamente con 3 cables.



Fig-9. Tira de LED RGB direccionable

### - Micro Ventilador 3/5VDC

Con el kit se incluye también un pequeño ventilador para la realización de los casos de uso que se presentarán más adelante. Este dispositivo tiene un consumo de aproximadamente 120 mA con lo que será preciso conectarlo a una salida especial que pueda entregar esa cantidad de corriente. Cabe recordar que normalmente las salidas y entradas digitales tienen un límite de unos 25 mA de corriente. Por ese motivo el ventilador se conectara a la salida OUT de la ESP32N1, que es la salida del segundo regulador.

Lo interesante de los ventiladores construidos de este modo es que el motor principal no posee escobillas y por lo tanto no existe un desgaste en las mismas. Por lo tanto, estos ventiladores son diseñados para durar mucho tiempo en el equipamiento que los incorpora. En nuestro caso lo hemos seleccionamos por su seguridad y facilidad en el uso, ya que cumple el propósito de demostrar el concepto de funcionamiento de unidades más grandes que se conectan, por ejemplo mediante un relé a la red eléctrica tradicional de 220V.



Fig-10. Micro Ventilador 3/5VDC

### - Zumbador (Buzzer)

El zumbador o buzzer es un elemento formado por un material piezoeléctrico, esto es, un material que al recibir un impulso eléctrico altera su estado mecánico y vibra, en este caso produciendo un sonido.

Al igual que sucede en otros componentes, el buzzer tiene una polaridad que hay que respetar para que funcione.

Existen dos tipos de buzzer activos y pasivos. Mientras que los activos sólo producen un sonido cuando se aplica un voltaje a sus pines, los pasivos pueden generar sonidos en función de la frecuencia que reciben en su pin de señal. En el kit se suministra un buzzer de tipo activo.



Fig-11. Zumbador (Buzzer)

### - Antena Externa WiFi / Bluetooth / BLE

En el kit se incorpora también una antena externa para ofrecer una mejora en la cobertura WIFI / Bluetooth / BLE (Bluetooth de bajo consumo). Una antena en realidad se puede considerar como sensor cuando recibe ondas y como actuador al mismo tiempo cuando las emite.

La antena se conecta mediante un conector marcado en la placa como UFL1. De este modo se mejora la calidad de envío y recepción de la señal.



Fig-12. Antena externa con conector UFL

### - Batería de Iones de Litio (Li-Ion)

Otro de los elementos que se han añadido al kit ha sido una batería de 3200 mAh de Iones de Litio.

Como el objetivo del kit es proporcionar una formación enfocada al “IoT” Internet de las Cosas, es importante tener en cuenta como diseñar soluciones que permitan ser alimentadas desde una batería. Este es el motivo por el que el kit incorpora este componente.

El formato de batería seleccionado es el 18650. Éste es bastante común y permite una relación tamaño / capacidad eléctrica bastante buena.

Este tipo de baterías son las que se emplean habitualmente en los ordenadores portátiles, vehículos eléctricos, etc. En esos casos cada una de las pilas 18650 se unen en serie o paralelo para formar unidades más grandes a las que se denomina también baterías. En ese caso cada una de las pilas 18650 pasa a ser denominada celda.

Uno de los principales fabricantes de batería del mundo es Panasonic y la batería incluida en el kit es la NCR18650B de dicho fabricante.



Fig-13. Bateria NCR18650B de Li-Ion de Panasonic

### - Cable Mini-USB y conjunto de cables dupont de colores

El kit además incluye un cable mini USB para la conexión de la placa al ordenador así como un conjunto de cables dupont hembra-hembra de colores para realizar las conexiones entre los diferentes dispositivos.

En los esquemas de conexión mostrados en el apartado de los casos prácticos, se ha empleado la herramienta Fritzing, y se ha empleando los colores en los cables para poder estandarizar las conexiones.



Fig-14. Cable mini USB y set de cables dupont de colores.

## 2.3. Placa ESP32N1

La placa ESP32N1 es la placa principal sobre la que se desarrollan todos los casos prácticos contenidos en el kit.

Esta pieza de hardware es fruto del trabajo y desarrollo del equipo whitecatboard.org durante mas de 2 años de prototipos hasta la versión final de la placa.

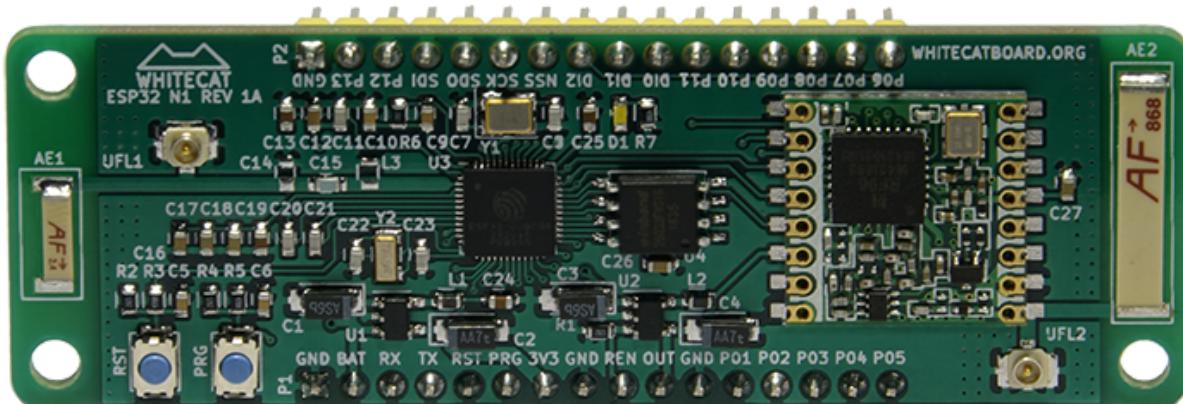


Fig-15. Placa ESP32N1

Esta placa está diseñada teniendo en mente su versatilidad así como la comunidad que existe detrás del procesador principal. Las características principales de dicha placa son:

- Procesador ESP32 de Espressif 2 núcleos hasta 240Mhz
- Conectividad Bluetooth y BLE
- Conectividad WiFi
- Conectividad LoRa

A continuación, se detallan la estructura y cada una de las partes que la componen así como el esquemático de las diferentes partes de la placa.

La alimentación principal de la placa ESP32N1 se realiza a través del pin marcado como BAT. Dicho pin permite un voltaje de entrada de 2,7 a 5,5 voltios y está pensado para ser alimentado o bien por una fuente de 5V como podría ser un puerto USB o bien una fuente de 4,2V cómo sería una batería cargada de iones de litio (Li-Ion).

Como se verá en el apartado de la placa DEVKIT, la celda de alimentación escogida es el formato estándar 18650 por la posibilidad de encontrar dichas baterías de un modo bastante extendido, y las capacidades de carga que se explicarán más adelante en los casos de uso.

La entrada de voltaje desde los 2.7 V hasta los 5.5 V pasa a través de un regulador DC-DC llamado step-down que rebajará el voltaje por ejemplo de 5 V a 3.3 V que es el voltaje al cual funcionan el resto de componentes de la placa.

Por eficiencia energética se ha empleado un step-down (comutado) en vez de un LDO (térmico). La diferencia entre los dos sistemas reside en que mientras el primero baja la tensión mediante la conmutación de la salida (ON/OFF) muy rápidamente en general a frecuencias de 2 Mhz (2 millones de veces por segundo) lo cual resulta muy eficiente desde el punto de vista de los consumos, el segundo sistema rebaja la tensión simplemente transformando la energía sobrante en calor, disipando y desperdimando energía térmica al ambiente. Esta circunstancia no lo hace un sistema apto para aplicaciones de bajo consumo o alimentadas por baterías.

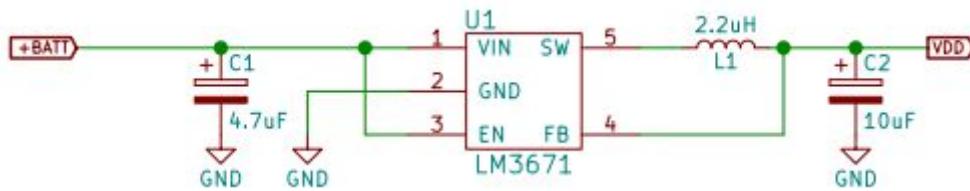


Fig-16. Esquema alimentación principal de entrada

El mismo sistema que se emplea en la alimentación principal se usa también para el regulador secundario. En este caso mediante un pin de entrada/salida del ESP32 controlamos el estado de dicho regulador, permitiendo así que este esté operativo o no en función de las necesidades de comunicación, sensórica, etc. Este segundo regulador está conectado a su salida en el pin marcado como OUT y al mismo tiempo también alimenta el módulo HopeRF de LoRaWAN. El pin de control REG\_EN para (ON/OFF) es el GPIO27.

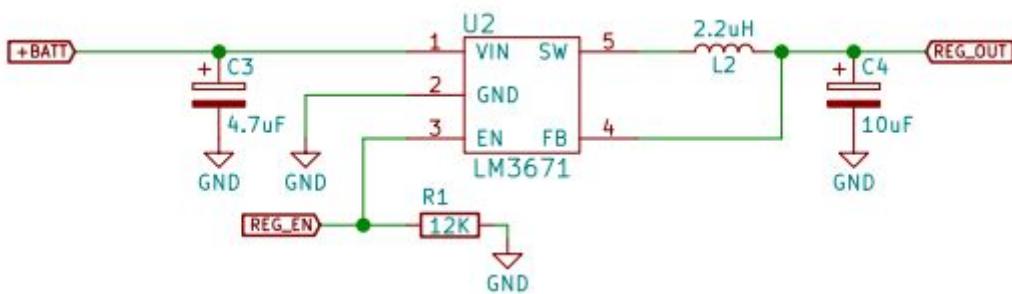


Fig-17. Esquema alimentación secundaria de salida

La placa incorpora una memoria de almacenamiento externa para poder guardar el programa y los datos, en este caso el sistema operativo LuRTOS y el sistema de ficheros que acompaña a dicho sistema operativo, incluyendo los ejemplos que vienen preinstalados de serie.

La memoria flash está alimentada desde un pin especial del procesador principal ESP32. Cuando éste entra en modo de reposo, la memoria deja de estar alimentada, con el consiguiente ahorro de energía.

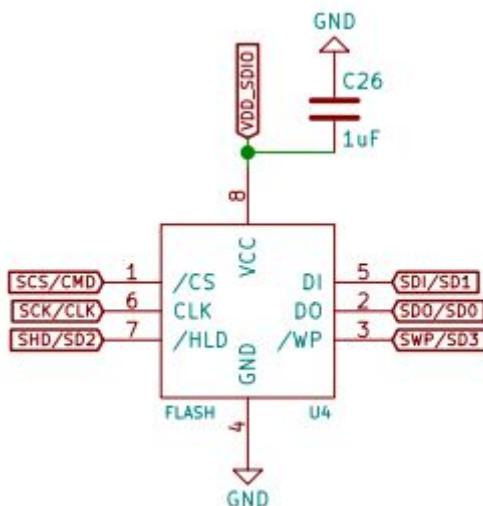


Fig-18. Esquema memoria de almacenamiento flash

Para su conexionado con elementos externos o bien con la placa de desarrollo DEVKIT, la ESP32N1 cuenta con 2 tiras de pines una a cada lado y con 16 conexiones cada una. En la siguiente figura se puede observar cada uno de esos pines y en la figura que le sigue se detallan cada una de las funciones de dichos pines.

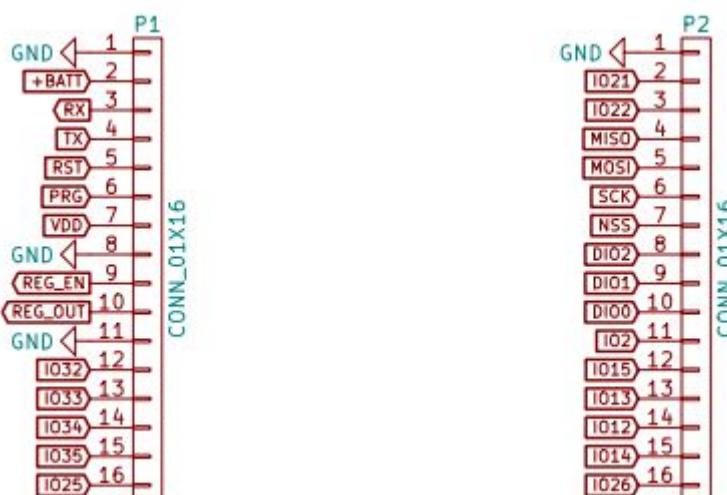
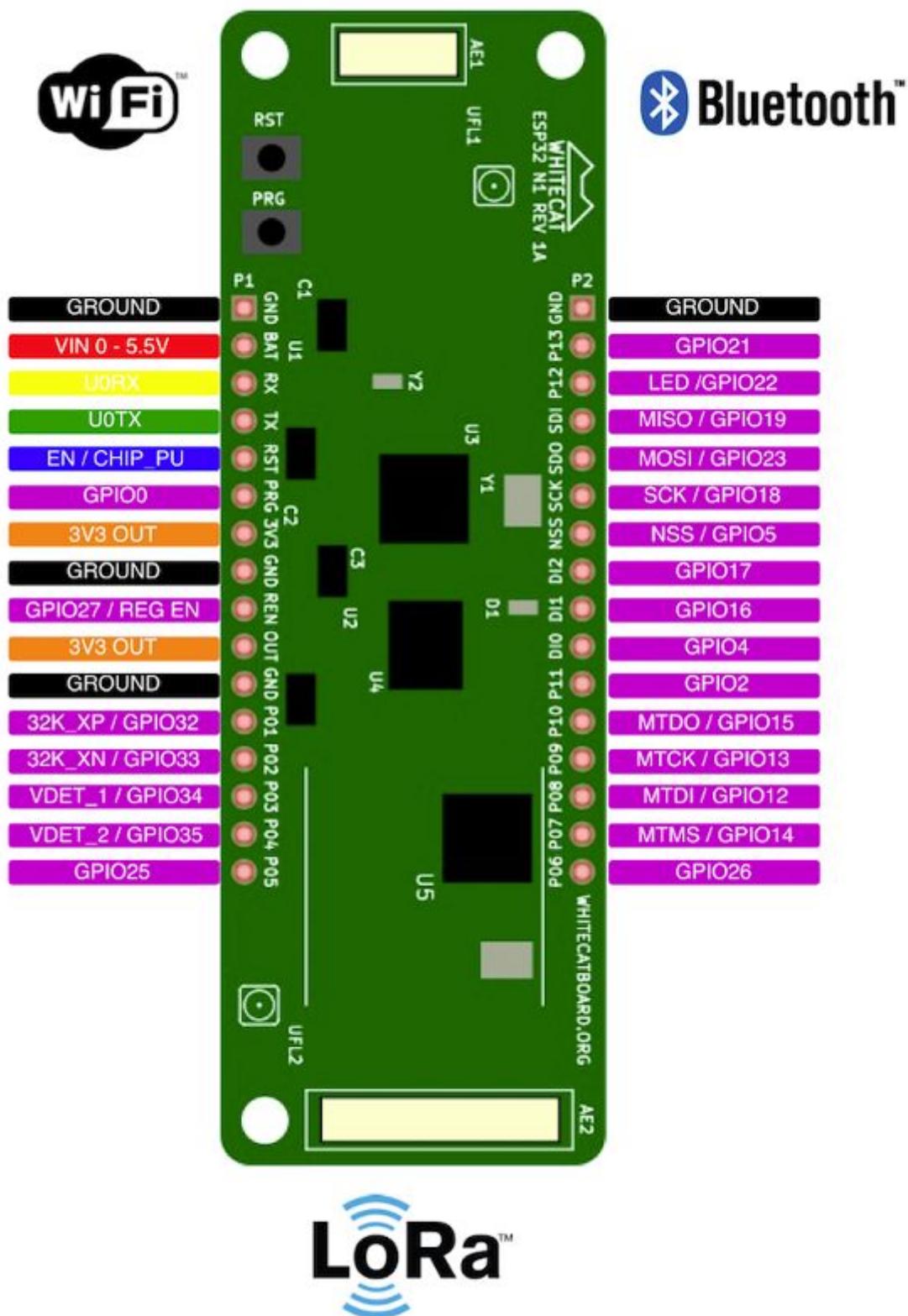


Fig-19. Esquema pines de conexiones ESP32N1

En la siguiente figura se describen cada una de las funciones que incorporan los pines laterales de la placa ESP32N1.



La placa cuenta con un botón para producir un reinicio por hardware. Dicho tipo de reinicio se emplea en el caso de un fallo inesperado que dejase la placa en un estado bloqueado y principalmente es empleado para entrar en el modo bootloader conjuntamente con el botón de programación.

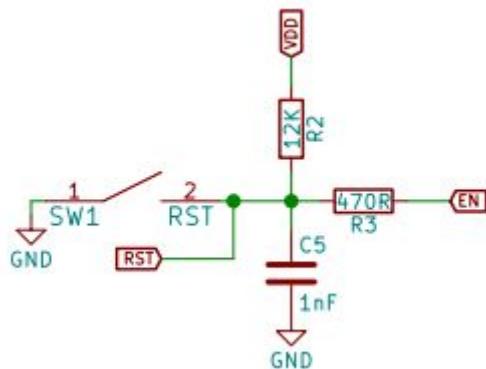


Fig-21. Esquema botón de reset

Si se mantiene pulsado el botón de programación PRG mientras se reinicia la placa ESP32N1, se entra en un modo especial que permite subir el programa desarrollado a la memoria flash. En el caso de empleo de la placa con el IDE de bloques de LuaRTOS esto no será necesario usando la placa DEVKIT ya que esta posee un sistema “autoprogram”

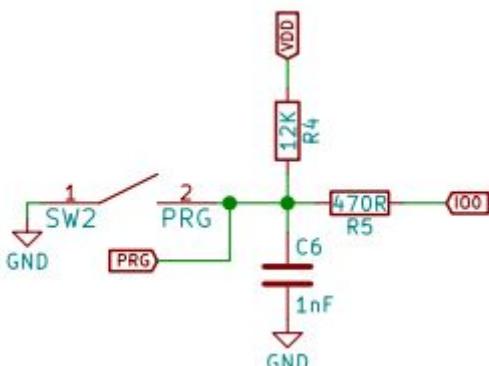


Fig-22. Esquema botón de programación (modo bootloader)

El módulo HopeRF de radio es el encargado de realizar las comunicaciones en la banda LoRaWAN de cada país. Por defecto, en la Unión Europea se monta el módulo en su versión de 868Mhz y la antena cerámica correspondiente de la misma frecuencia. El uso de antena cerámica simplifica y acorta los tiempos de diseño de la placa, y permite mayor versatilidad ya que se pueden adaptar en el mismo formato diferentes frecuencias sin necesidad de cambiar el diseño.

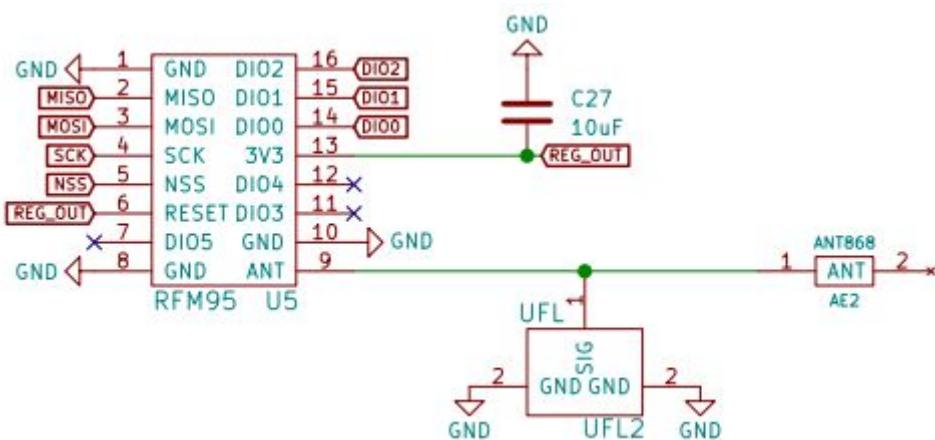


Fig-23. Esquema radio LoRaWAN (transceiver HopeRF)

Existe un led de estado conectado al IO22. Este led con LuaRTOS cargado en memoria y ejecutándose, parpadea una vez por segundo.

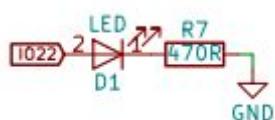


Fig-24. Led de estado

Por último la parte principal del módulo ESP32N1 es la CPU. A continuación se muestra un esquemático con las diferentes partes que integran el circuito y hacen posible su funcionamiento.

En la línea LNA\_IN se encuentra la antena de WiFi y bluetooth así como un conector UFL. Para la antena externa, se ha provisionado el pcb para poder realizar un match network y optimizar los parámetros de la antena cerámica en placa.

El componente marcado como Y1 es el cristal de cuarzo principal que hace funcionar la CPU. Éste opera internamente a 40 Mhz pero mediante el uso de PLL se incrementa la frecuencia de funcionamiento hasta los 240 Mhz.

El resto de componentes son condensadores de desacoplo, condensadores en las salidas / entradas SENSOR y condensadores en CAP1 y CAP2.

El oscilador de 32.768Khz no está operativo en la revisión actual de la CPU ya que forma parte de una de las erratas. Por tal motivo es una de las partes que no están soldadas en la placa físicamente.

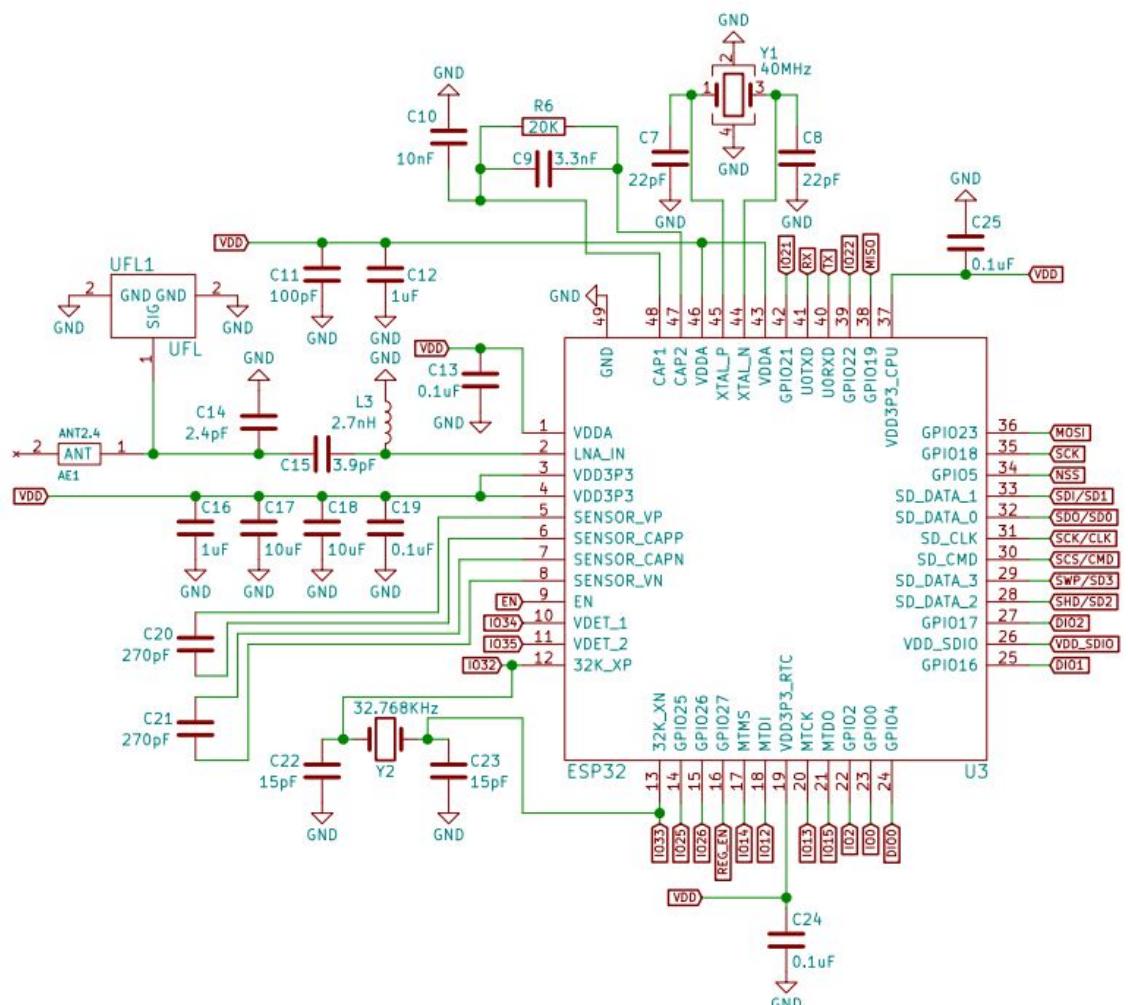


Fig-25. Esquemático de la CPU ESP32

Como última nota sobre la placa ESP32N1 se muestra una imagen con la antena externa conectada en el puerto UFL1. Dicha antena mejora las prestaciones de la WiFi y el Bluetooth en aquellas aplicaciones donde sea preciso y no sea suficiente con la antena cerámica incluida.

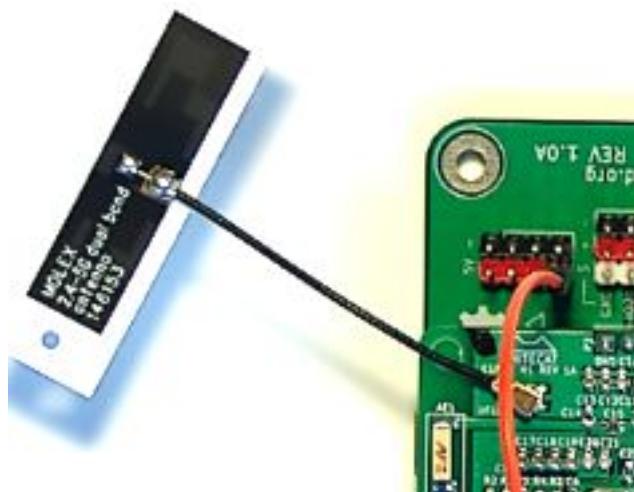


Fig-26. Detalle conexión antena externa WiFi / Bluetooth en conector UFL1

## 2.2. Placa DEVKIT para ESP32N1

La placa ESP32N1 DEVKIT es un componente de hardware que mejora las prestaciones de la placa ESP32N1 incrementando sus funcionalidades. Asimismo, facilita al mismo tiempo la interconexión de otros dispositivos, la comunicación con el PC y la programación automática desde el entorno de desarrollo WhitecatIDE.

A continuación se muestra una imagen de la placa ESP32N1 DEVKIT.

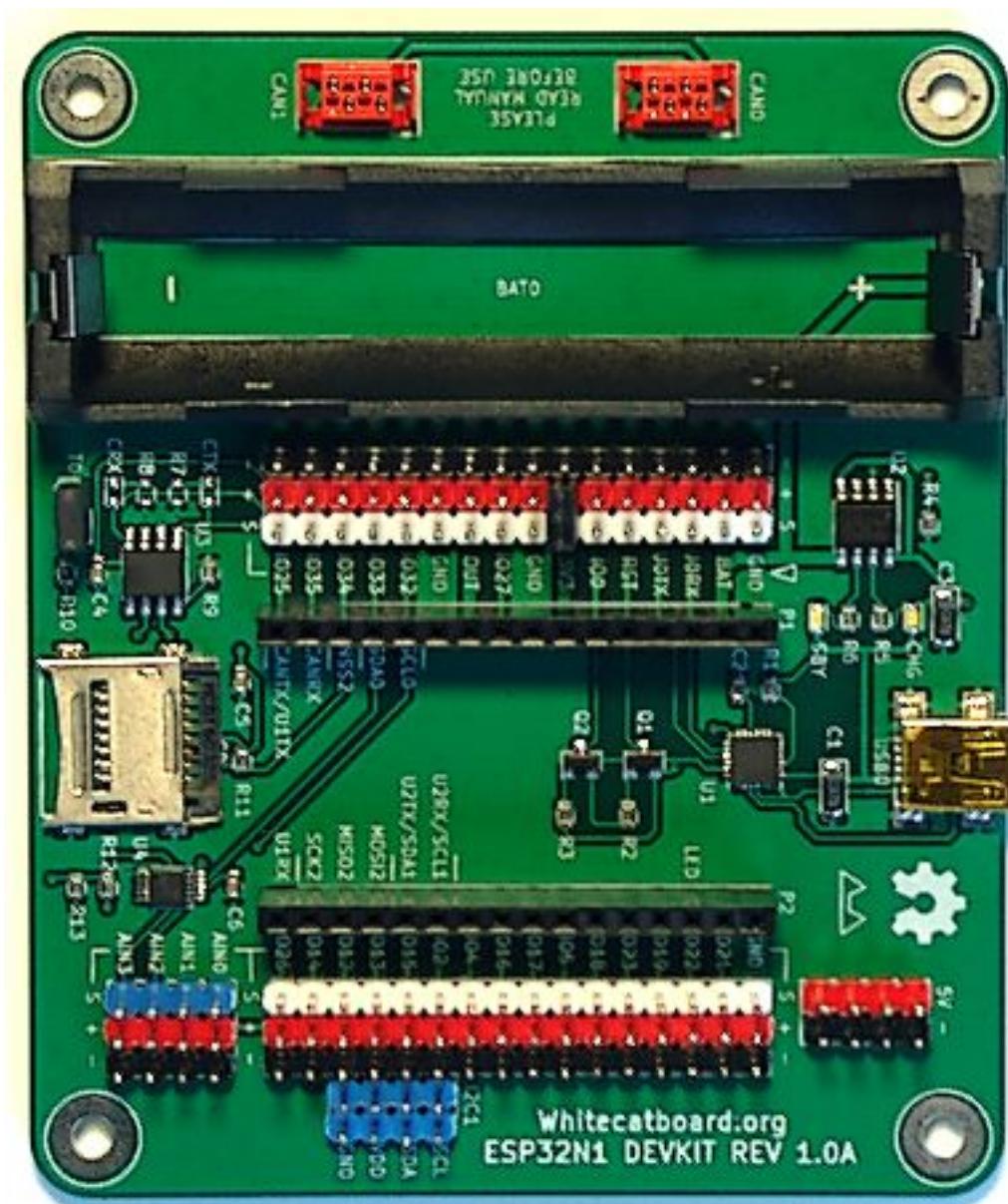


Fig-27. Placa de desarrollo ESP32N1 DEVKIT Revisión 1.0A

La placa ESP32N1 se inserta en los pines de la ESP32N1 DEVKIT como se muestra a continuación en la siguiente imagen.



**ES IMPORTANTE TENER EN CUENTA LA MANERA DE INSERTAR LA PLACA ESP32N1 SOBRE LA PLACA DE DESARROLLO YA QUE UNA INSERCIÓN INCORRECTA PODRÍA DAÑAR LAS 2 PLACAS.**

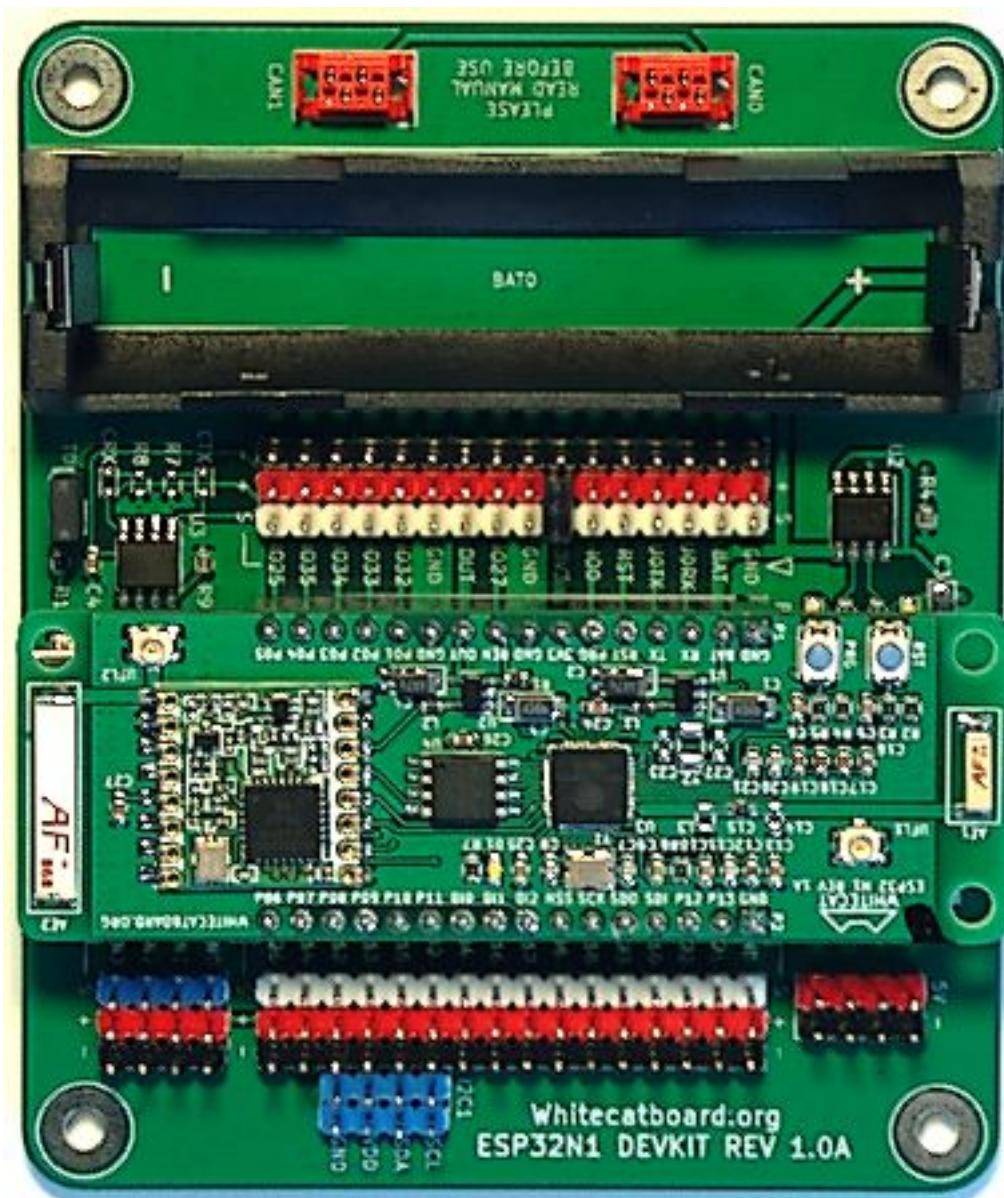


Fig-28. Placa ESP32N1 Insertada en la placa de desarrollo DEVKIT

La revisión 1.0A de la placa cuenta con un cargador de baterías de Ion de Litio (Li-Ion) incorporado. Esto es debido a que es posible que la batería no esté presente al no ser fija

en la configuración de la placa. Por tanto, es necesario usar un cable desde cualquier pin marcado como 5 V hasta el pin de señal marcado como BAT.



**ATENCIÓN !! SOLAMENTE ES PRECISO USAR ESTE CABLE SI LA BATERÍA NO ESTÁ PRESENTE EN EL ZÓCALO MARCADO COMO BAT0.**

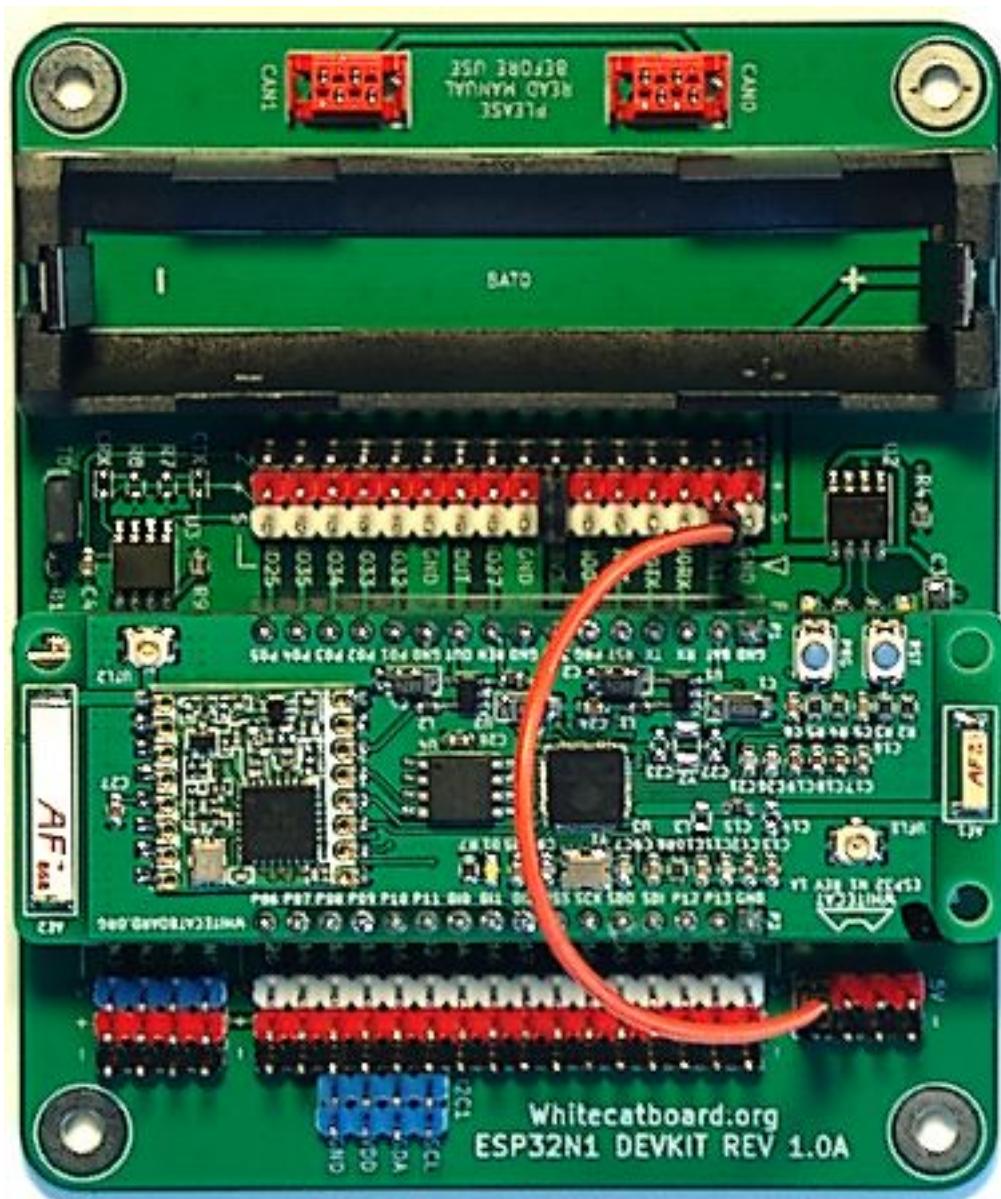


Fig-29. Cable usado cuando la batería no está insertada en el zócalo marcado como BAT0

Si por el contrario hemos insertado la batería en el zócalo marcado como BAT0 tal y como muestra la imagen, es importante que no esté presente el cable desde el pin 5 V hasta el pin de señal BAT.

**!** **QUITAR EL CABLE ENTRE 5 V Y BAT SI TENEMOS INSERTADA LA BATERÍA PARA UN CORRECTO FUNCIONAMIENTO. ES IMPORTANTE EXTREMAR LAS PRECAUCIONES SIEMPRE QUE SE TRABAJAN CON BATERÍAS. SI POR ALGÚN MOTIVO NOTASE EXCESO DE TEMPERATURA EN LA BATERÍA, DESCONECTE LA PLACA INMEDIATAMENTE.**



Fig-30. Placa DEVKIT con la batería insertada en el zócalo BAT0.



**UNA VEZ REPASADO EL PUNTO INICIAL DE LA BATERÍA, POR FAVOR TENGA EN CUENTA QUE LAS BATERÍAS NO SON UN JUGUETE. SI EL KIT VA HA SER USADO POR UN MENOR RECOMENDAMOS O BIEN QUE NO SE USE LA BATERÍA O BIEN, SI SE HACE USO DE ELLA, QUE EXISTA SUPERVISIÓN POR PARTE DE UN ADULTO.**

A continuación, procederemos a detallar las diferentes partes de la placa de desarrollo siguiendo el esquemático, tal como se ha realizado con la placa ESP32N1.

La parte inicial es la que se conecta al ordenador mediante un conector USB tipo Mini-B. Esta conexión aportará alimentación a la placa así como una conexión serie sobre el puerto USB, necesaria para la comunicación con la placa.

La conexión serie a USB la realiza el integrado Silabs CP2104, que además soporta el uso de las líneas TX / RX / RTS y DTR que serán precisas también para la programación automática, tal como veremos más adelante.

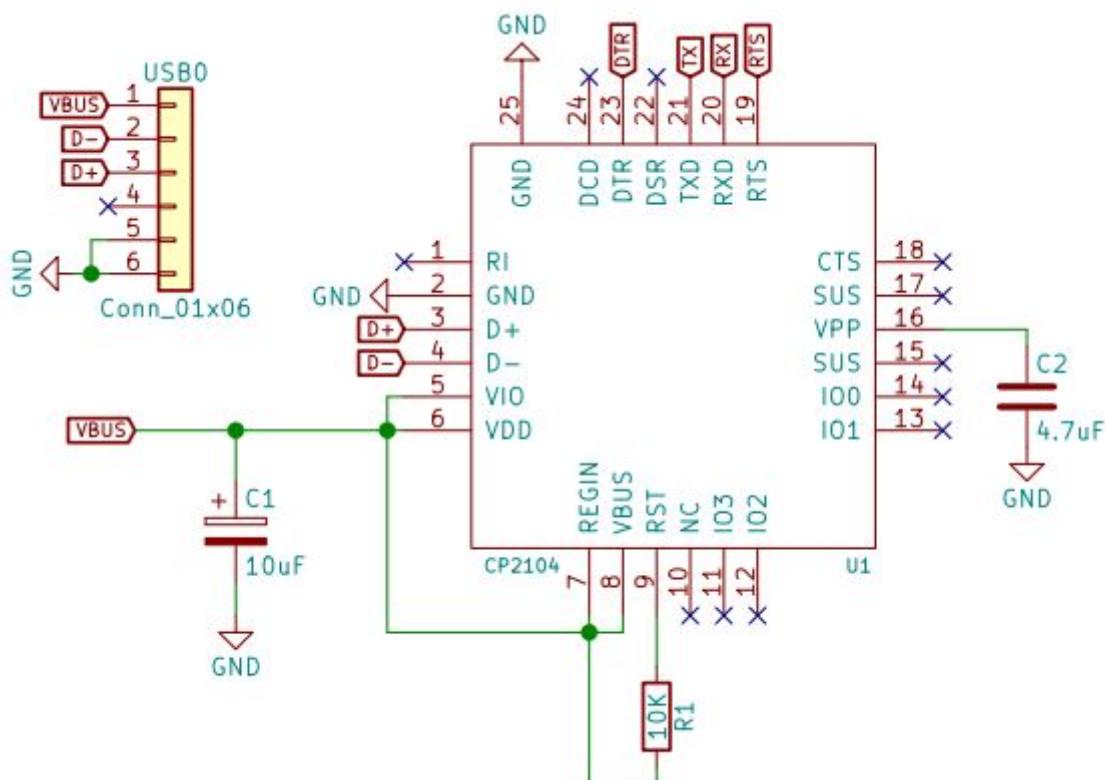


Fig-31. Esquemático del Integrado Silabs CP2104

Como hemos comentado, la conexión USB proporciona a la placa a parte de la conectividad, energía eléctrica, en este caso a través del cargador de baterías TP4056 que suministra un volumen de tensión máximo de 4.2 V y un volumen de corriente máximo de 580 mA.

El cargador de baterías TP4056 es un cargador del tipo voltaje/corriente que consiste en un circuito integrado para la carga de baterías bastante popular y extendido. En su caso es capaz solamente de cargar y gestionar una sola celda, como es el caso en la placa DEVKIT. Por tanto, el circuito integrado TP4056 es el encargado de cargar la batería y suministrar energía a la ESP32N1.



**ATENCIÓN, RECUERDE EL USO DE BATERÍAS DEBE ESTAR SUPERVISADO.  
SI APRECIA CUALQUIER ANOMALÍA O EXCESO DE TEMPERATURA POR FAVOR  
DESCONECTE LA PLACA.**

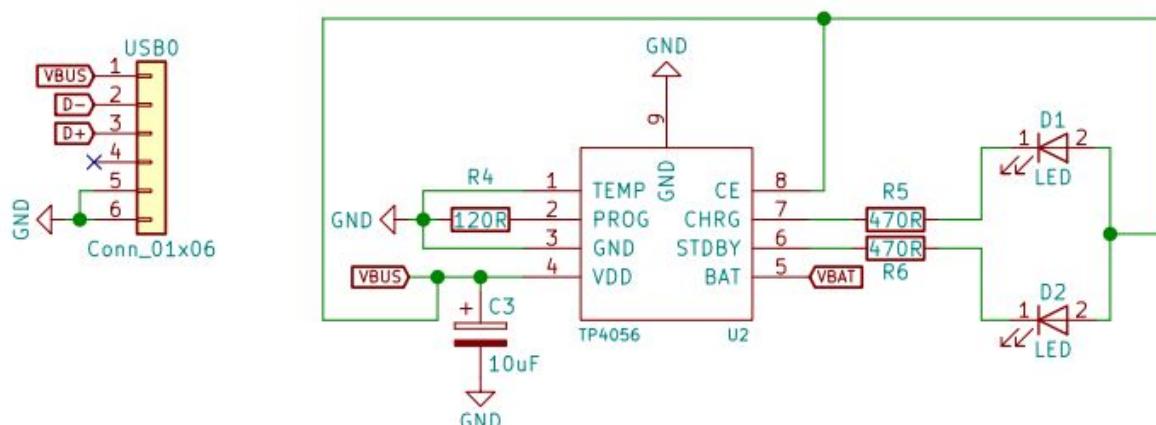


Fig-32. Esquemático del circuito integrado TP4056

La siguiente ampliación que presenta la placa de desarrollo es la implementación del transceiver para comunicaciones CANBus. Un transceiver, es un elemento que contiene un emisor y un receptor en el mismo encapsulado, en este caso orientado a un propósito como es la comunicación en CANBus.

Este tipo de red es una red con 2 cables CANH y CANL desarrollada por BOSCH principalmente para su uso como bus principal de comunicaciones en vehículos, pero también ampliamente utilizada en entornos industriales.

A bajas velocidades la longitud de los cables empleados en CANBus puede sobrepasar los 200 metros, por tal motivo es un tipo de red interesante para instalaciones industriales, domóticas, etc.

La ventaja del CANBus radica por un lado, en la inmunidad de la señal permitiendo su uso en entornos con grandes interferencias, y por otro en el concepto de que cada elemento del bus puede ser al mismo tiempo maestro o esclavo, no teniendo las limitaciones que tenían los buses 1-WIRE , I2c y SPI.

A continuación se muestra el esquemático del transceiver de Texas Instruments empleado para dar conectividad canbus a la placa ESP32N1.

El Jumper (puente) situado en T0 es para añadir una terminación de 120OHM al bus CANBus.

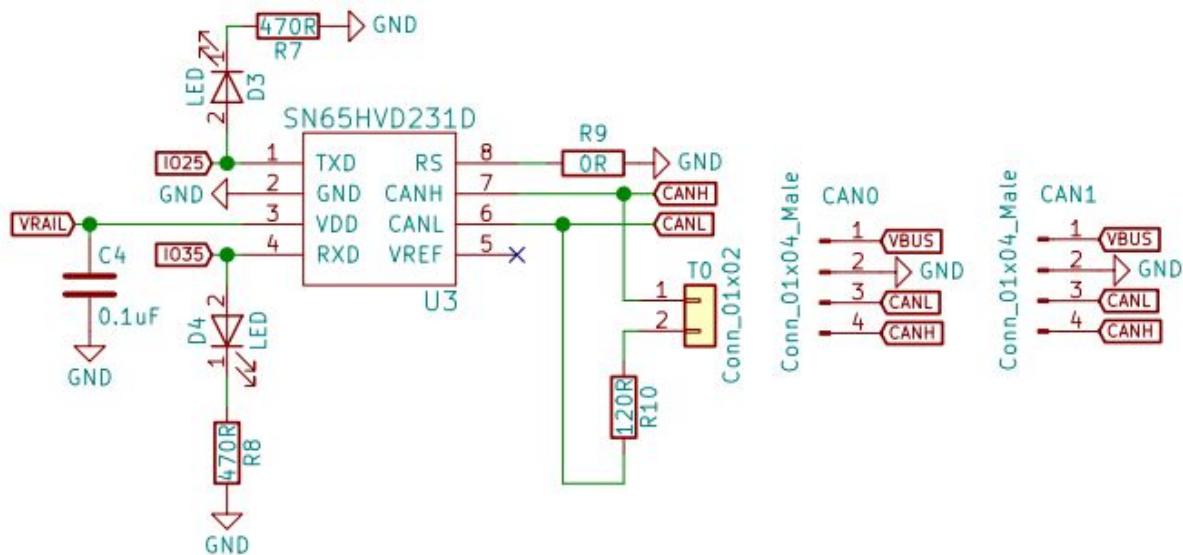


Fig-33. Esquemático del transceiver SN65HVD231D para conectividad CANBus.

Debido a las limitaciones en el apartado de conversores analógicos a digital que posee la CPU ESP32 se ha optado también por añadir en la placa de desarrollo un circuito integrado específico para dicho propósito, presentando así flexibilidad y valores mucho más precisos.

Anteriormente, se ha comentado la diferencia entre una señal analógica y una digital y se han introducido los conversores, así como el concepto de resolución. En el caso de la revisión 1.0 de la placa de desarrollo, hemos incorporado el conversor ADC ADS1015 de 12 bits de resolución, es decir doce unos o ceros para indicar el valor medido.

Existe la contrapartida a la entrada ADC o conversor Analógico Digital, que es el conversor Digital Analógico DAC. El propósito como su propio nombre indica, es exactamente el opuesto al ADC y convierte un valor digital de unos y ceros en un valor analógico, en este

caso nosotros lo llamaremos PWM y lo implementaremos en los pines de la ESP32N1 directamente mediante las funciones de LuaRTOS.

A continuación se muestra el esquemático del conversor analógico - digital ADC.

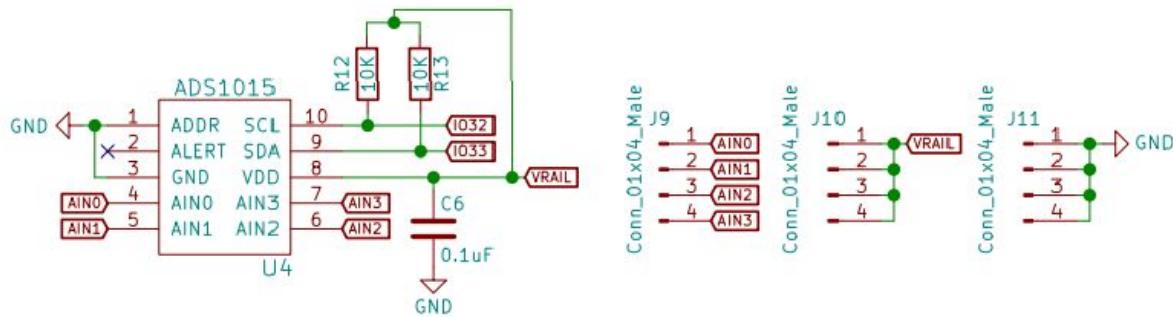


Fig-34. Esquemático conversor analógico-digital ADC 12/16 bit

Es importante destacar que el circuito integrado que realiza las mediciones es el mencionado ADS1015 de 12 bit, pudiéndose emplear un ADS1115 en futuras revisiones, lo que ofrecería 16 bits.

El circuito integrado ADS1X15 funciona mediante bus I2c y en el caso de la placa de desarrollo éste está conectado al bus I2c (0) que incluye los pines SCL (IO32) y SDA (IO33). Así mismo la dirección en el bus del chip ADS1X15 es la 0.

Por último se muestran los diferentes pines de señal que están directamente vinculados a la placa ESP32N1. A estos pines se les ha añadido un pin de alimentación y un pin de masa común.

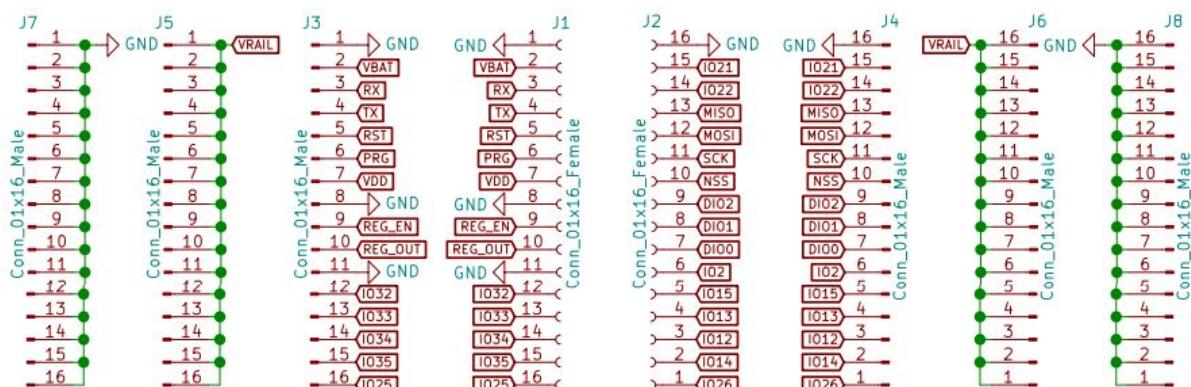


Fig-35. Esquemático de los pines de la placa de desarrollo

Para facilitar la identificación de los diferentes pines se ha ampliado la serigrafía en la placa de desarrollo indicando a qué pin GPIO corresponde cada uno de ellos.

Por otro lado, también se han marcado con diferentes colores los pines en la placa siendo de color rojo los pines de alimentación, negro los pines de masa y blancos o azules los pines de señal.

Se han incorporado también varios pines al puerto I2C1 para facilitar su conexión, tal como muestra la imagen siguiente:

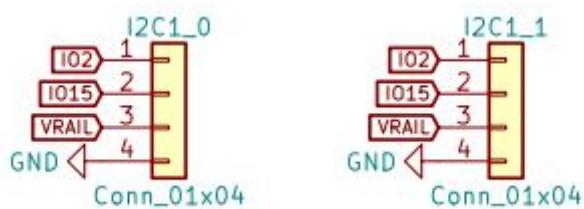


Fig-36. Esquemático pines I2C-1

### 3. Entorno de Desarrollo Whitecat IDE

#### 3.1. Características generales e Instalación

El entorno de desarrollo del ecosistema Whitecatboard se llama WhitecatIDE. Dicho entorno está basado en tecnología HTML5 y se ejecuta desde dentro de un navegador de tal manera que el usuario siempre dispondrá de la versión más actualizada de dicho entorno de desarrollo sin necesidad de instalarlo en su PC. El único elemento que se deberá instalar en el PC del usuario, es el agente que es el encargado de comunicar el navegador con la placa en local en el PC.

Además, el entorno de desarrollo de WhitecatIDE permite el trabajo principalmente con programación mediante bloques, pero también permite una consola para la edición de texto.

#### 3.2 Funciones del IDE

A continuación, se definen las funciones de los elementos principales que aparecen en la pantalla principal del IDE. Para utilizarla, es preciso acceder mediante una cuenta de Google ya que los ficheros de ejemplo se guardarán en una carpeta de drive del usuario.

La dirección para acceder al entorno de desarrollo es <https://ide.whitecatboard.org>

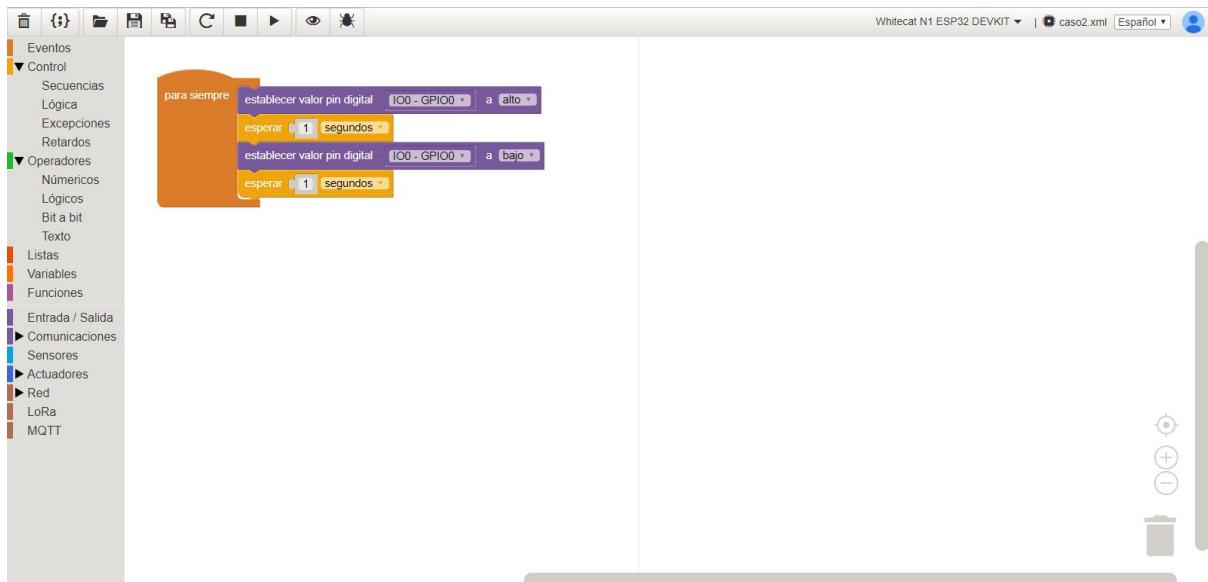
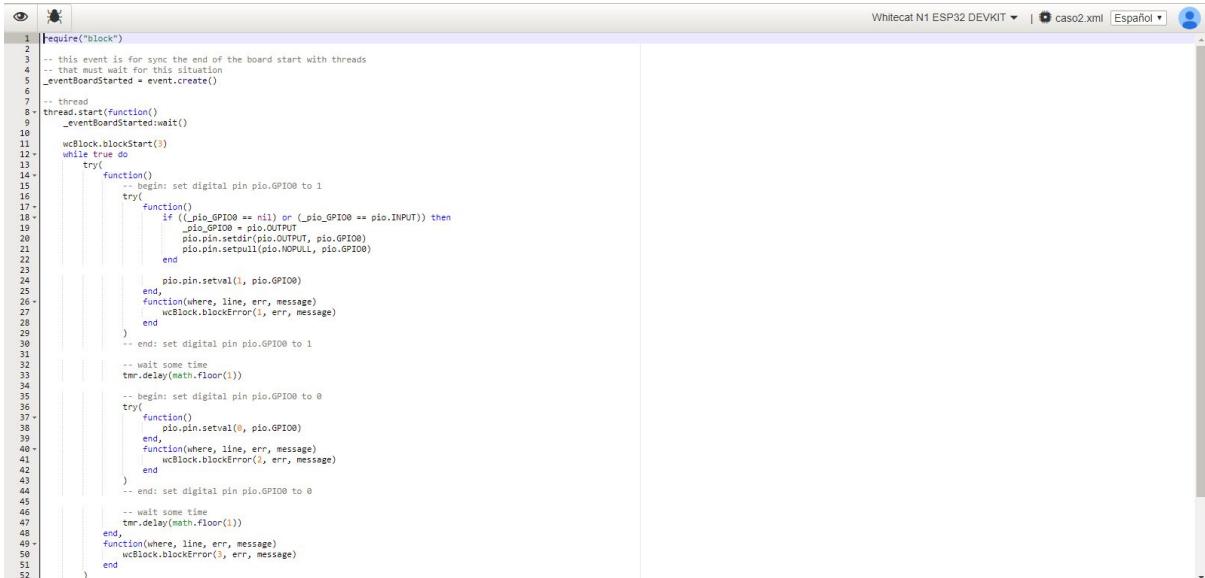


Fig-37. Pantalla principal del entorno de desarrollo en modo bloques



```

1 |require("block")
2 |
3 |-- this event is for sync the end of the board start with threads
4 |-- that must wait for this situation
5 |_eventBoardStarted = event.create()
6 |
7 |-- thread
8 |thread.start(function()
9 |  _eventBoardStarted:wait()
10 |
11 |  wcBlock.blockStart(3)
12 |  while true do
13 |    try{
14 |      function()
15 |        begin: set digital pin pio.GPIO0 to 1
16 |        try(function()
17 |          if (_pio.GPIO0 == nil) or (_pio.GPIO0 == pio.INPUT) then
18 |            pio.GPIO0 = pio.OUTPUT
19 |            pio.pin.setdir(pio.OUTPUT, pio.GPIO0)
20 |            pio.pin.setpull(pio.NOPULL, pio.GPIO0)
21 |          end
22 |
23 |          pio.pin.setval(1, pio.GPIO0)
24 |        end,
25 |        function(where, line, err, message)
26 |          wcBlock.blockError(1, err, message)
27 |        end
28 |
29 |      )-- end: set digital pin pio.GPIO0 to 1
30 |
31 |      -- wait some time
32 |      tmr.delay(math.floor(1))
33 |
34 |      -- begin: set digital pin pio.GPIO0 to 0
35 |      try(function()
36 |        pio.pin.setval(0, pio.GPIO0)
37 |      end,
38 |      function(where, line, err, message)
39 |        wcBlock.blockError(2, err, message)
40 |      end
41 |
42 |      )-- end: set digital pin pio.GPIO0 to 0
43 |
44 |      -- wait some time
45 |      tmr.delay(math.floor(1))
46 |
47 |      function(where, line, err, message)
48 |        wcBlock.blockError(3, err, message)
49 |      end
50 |
51 |    end
52 )

```

Fig-38. Pantalla principal del entorno de desarrollo en modo código

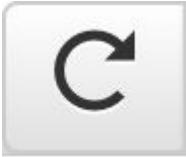
## Barra superior

En la barra superior encontramos diferentes botones en función del estado, estado de comunicación con la placa, selector de idioma y perfil de usuario.



Fig-39. Barra superior de herramientas.

|  |   |
|--|---|
|  | Botón para eliminar el programa actual.   |
|  | Botón para alternar entre el modo de Bloques y el modo de Código.   |
|  | Botón para abrir un programa existente. Estos se pueden abrir desde la placa o desde el PC o desde la nube. |

|   |   |
|---|---|
|    | Botones para guardar el programa y guardar como ... , se pueden almacenar los programas tanto en la placa como en la nube.  |
|    | Botón para reiniciar la placa. Este botón provoca un reinicio de la placa por hardware.   |
|    | Botón para parar la ejecución del programa en curso, de modo que si existe algún programa ejecutándose en la placa, lo detiene.   |
|    | Botón para la ejecución de un programa. Este botón ejecuta los bloques o el código que tengamos en el área de trabajo de manera instantánea, después de copiar el código a la placa.                  |
|   | Botón que muestra el código generado mediante los bloques. Se puede utilizar a modo de aprendizaje o revisión del código generado. No confundir con el botón que alterna entre modo código o bloques. |
|  | Botón que habilita el modo de depuración para la búsqueda de errores.   |



```

Lua RTOS beta 0.1. Copyright (C) 2015 - 2017 whitecatboard.org

build 1511857188
commit b8545c9a4ce242147bb5dbe92de9bcdedda5bf7
Running from factory partition
board type N1ESP32-DEVKIT
cpu ESP32 rev 0 at 240 Mhz
flash EUI d66634415a4e8010
spiffs0 start address at 0x310000, size 512 Kb, partition spiffs
spiffs0 mounted
sd0 is at spi2, cs=GPIO21
fat0 can't mounted
can't redirect console messages to file system, an SDCARD is needed

Lua RTOS beta 0.1 powered by Lua 5.3.4
Lua RTOS-boot-scripts-aborted-ESP32
/ > 

```

Fig-40. Consola.

Mediante el desplegable es posible acceder a la consola de texto de la placa.



Fig-41. Desplegable selección de idioma.

Los últimos elementos que se muestran en la parte superior son:

- Nombre del fichero con el que estamos trabajando.
- Selección de Idioma.
- Perfil de usuario para cerrar la sesión, etc.

## Área de bloques

A la izquierda encontramos el área de trabajo. En ella, están presentes todos los bloques que se pueden utilizar en el entorno de desarrollo, organizados por categorías y tipos.



Fig-42. Categorías de bloques.

## Área de trabajo

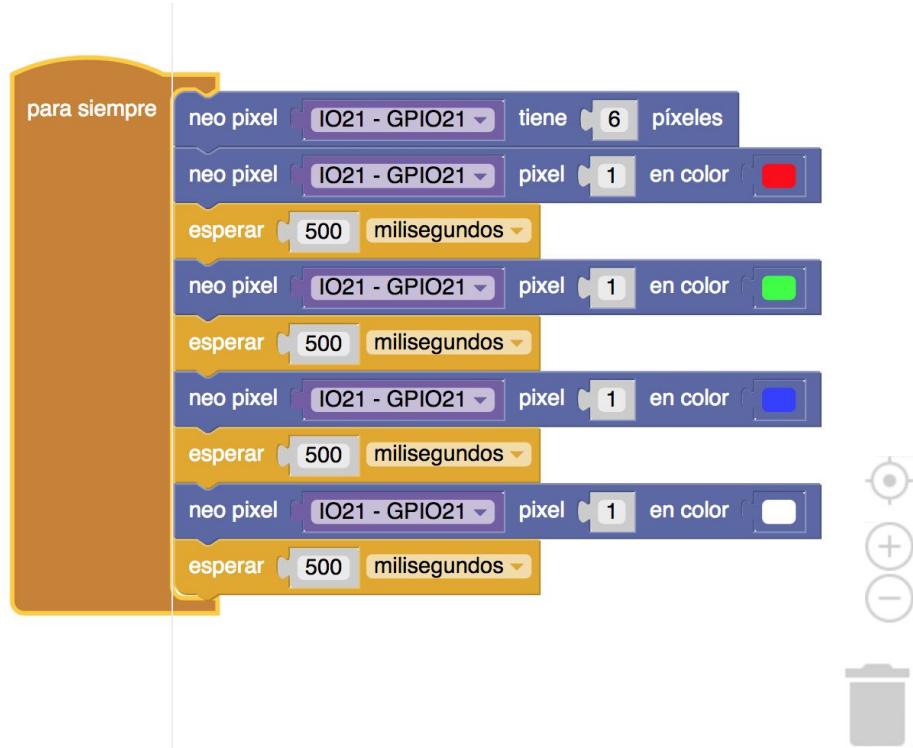


Fig-43. Área de trabajo con bloques.

## Los Bloques

Los bloques son elementos que generan código a partir de unos procesos ya predefinidos por el equipo de desarrollo de whitecatboard.org. De este modo, se facilita enormemente la labor al usuario a la hora de desarrollar proyectos utilizando los bloques.

**! IMPORTANTE, TODOS LOS BLOQUES ESTÁN DOCUMENTADOS MEDIANTE UN MENÚ CONTEXTUAL. EN EL PROPIO IDE Y UTILIZANDO EL BOTÓN DERECHO DEL RATÓN SOBRE UN BLOQUE, PODREMOS ACCEDER A SU AYUDA EN LÍNEA.**

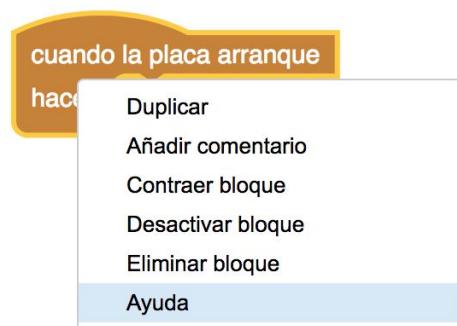


Fig-44. Menú contextual de un bloque botón derecho del ratón.

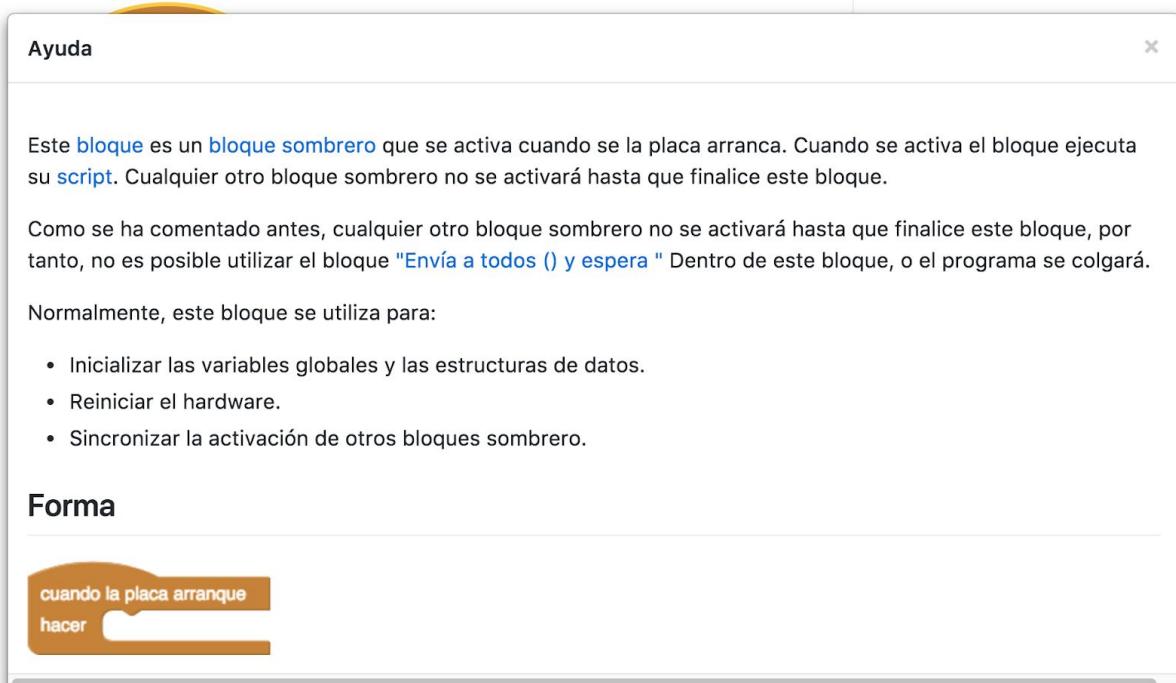


Fig-45. Pantalla de ayuda de un bloque.

## 4. LuaRTOS Sistema Operativo del ecosistema Whitecatboard

Lua RTOS es un sistema operativo en tiempo real diseñado para funcionar en sistemas embebidos, con requisitos mínimos de FLASH y memoria RAM. Actualmente, Lua RTOS está disponible para las plataformas ESP32, ESP8266 y PIC32MZ.

Los dispositivos que funcionan con Lua RTOS pueden ser programados directamente desde la consola del propio dispositivo, o mediante el entorno de desarrollo integrado The Whitecat IDE utilizando bloques, o Lua.

Lua RTOS ha sido desarrollado íntegramente en CitiLab Cornellà, Barcelona.

### 4.1 Diseño

El diseño del sistema operativo se puede describir en tres niveles o capas:

- En la capa superior corre un intérprete Lua 5.3.4 que ofrece al programador todos los recursos proporcionados por el lenguaje de programación Lua, además de módulos especiales para acceder al hardware (PIO, ADC, I2C, RTC, etc ...), y servicios de middleware (Lua Threads, LoRa WAN, MQTT, ...).
- En la capa media hay un micro-kernel en tiempo real, alimentado por FreeRTOS. Este es el responsable de que las cosas sucedan en el tiempo esperado.
- En la capa inferior hay una capa de abstracción de hardware, que habla directamente con el hardware de la plataforma.

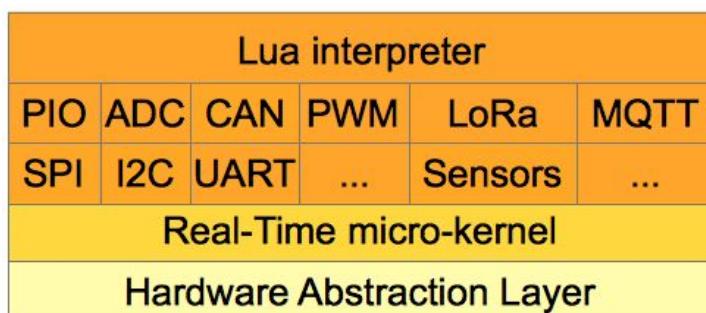


Fig-46. Estructura LuaRTOS.

## 4.2 Funciones principales

Las funciones principales que incorpora de serie el sistema operativo LuaRTOS son:

- Lua Threads: permite ejecutar funciones de Lua en paralelo.
- Shell integrado: permite ejecutar ciertos comandos de una manera similar a linux.
- Editor integrado: permite editar archivos.
- Módulos de acceso a sensores. Dispone de una API unificada de acceso a los sensores más utilizados en la industria.
- Módulos de acceso al hardware: GPIO, ADC, CAN, PWM, SPI, I2C, UART, TIMER.
- Módulos middleware: LoRaWAN, MQTT, HTTP server.
- Sistema de archivos: SPIFFS / FAT.
- Almacenamiento: FLASH interna / SD Card.
- Comunicaciones: Ethernet, Bluetooth, WiFi, LoRaWAN.
- Otros módulos: TFT display, NeoPixel, LCD, Stepper, Servos.

## 5. Conectividad (WiFi / Bluetooth / LoRaWAN)

Uno de los aspectos más importantes del kit de desarrollo IoT es la capacidad de comunicación del entorno y el hardware.

Por defecto, la placa incorpora de serie conectividad en WiFi, Bluetooth y LoRaWAN, de modo que las dos primeras dependen de infraestructura estándar.

Los estándares de WiFi / Bluetooth y BLE (Bluetooth de bajo consumo) están soportados por defecto en la placa en la misma CPU ESP32. Por otro lado, el estándar LoRaWAN está soportado mediante el transceiver HopeRF, tal como se comentó en el apartado de hardware de la placa ESP32N1.

Ahora bien, análogamente que para el uso de WiFi o Bluetooth son necesarios o bien un punto de acceso a la red WiFi o un dispositivo cercano que soporte bluetooth, para el uso del estándar LoRaWAN será preciso el uso de un Gateway LoRaWAN que permite conectar nuestro dispositivo con Internet mediante la red de Radio LoRaWAN.



Fig-47. Gateway LoRaWAN desarrollado por whitecatboard.org

Como el desarrollo del Kit se realizó en Cornellà del Llobregat, en dicho municipio contamos por defecto, con acceso a tres gateways LoRaWAN con una cobertura muy amplia sobre el municipio y alrededores, como se puede ver en la siguiente imagen.

Los gateways enlazan a los dispositivos (nodos) que transmiten y reciben datos mediante ondas de radio en el protocolo LoRaWAN, con Internet.



Fig-48. Cobertura y ubicación de los Gateways LoRaWAN en Cornellà del Llobregat.

Como se abordará en el siguiente capítulo, no solamente es necesario la presencia de la infraestructura de red suficiente para comunicar vía radio con los dispositivos, sino que también necesitaremos de una cuenta en un proveedor de servicios LoRaWAN. Posteriormente, explicaremos como realizar dicho proceso con The Things Network (TTN).

En lo relativo a WiFi, la conectividad, la búsqueda de redes y el servicio de Access Point AP, están implementados en el propio sistema operativo LuRTOS y soportado por tanto en el entorno de desarrollo mediante sus bloques y ejemplos correspondientes.

Actualmente, el Bluetooth y BLE, están soportados a nivel de hardware pero debido a su dificultad de implementación, no están incorporados a nivel de sistema operativo ni de bloques.

## 6. LoRaWAN Introducción y Administración en The Things Network (TTN)

Como se ha comentado en el capítulo anterior, para poder desarrollar comunicaciones en el ámbito de una red LoRaWAN debemos contar con los siguientes elementos:

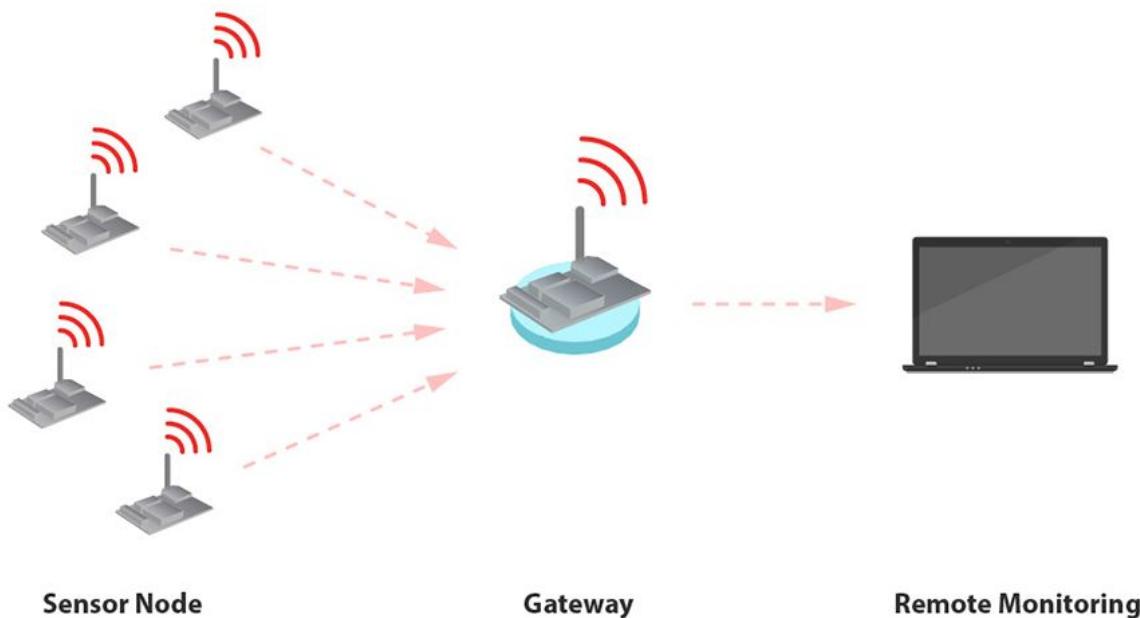


Fig-49. Elementos de una red LoRaWAN.

-Un nodo: nuestro kit de desarrollo DEVKIT ESP32N1 es un nodo con soporte para LoRaWAN.

-Un Gateway: actualmente en Cornellà del Llobregat hay desplegados, tal como ya se ha comentado, 3 gateways pero este número está creciendo en diferentes municipios y la tendencia es que cada vez existan más Gateways disponibles ya que son los encargados de conectar nuestros nodos con Internet.

-Un espacio de monitorización y trabajo: aquí es donde interviene el objeto de este capítulo que tratará de explicar los elementos que precisamos para poder empezar a trabajar con el sistema LoRaWAN. En nuestro caso emplearemos la consola The Things Network o, en adelante, TTN.

## 6.1 The Things Network (TTN)

En primer lugar debemos conocer el proyecto The Things Network (TTN). Dicho proyecto pretende democratizar el uso de los gateways entre los participantes de dicha red de tal modo que podremos emplear los gateways y la infraestructura de otros y análogamente otros podrán utilizar nuestro gateway o gateways en caso de que los tenga conectados a dicha red. De este modo se pretende dinamizar el crecimiento de una red LoRaWAN pública.

Recomendamos ir consultando el estado de este proyecto en la web <https://www.thethingsnetwork.org/> ya que es un proyecto dinámico que va cambiando con el paso del tiempo y van surgiendo novedades.

Una vez visto el proyecto de The Things Network, procederemos a registrarnos como usuario para poder utilizar su entorno web.



### CREATE AN ACCOUNT

Create an account for The Things Network and start exploring the world of Internet of Things with us.

---

**USERNAME**  
This will be your username — pick a good one because you will **not** be able to change it.

 ACME

**EMAIL ADDRESS**  
You will occasionally receive account related emails. This email address is not public.

 acme@acme.com

**PASSWORD**  
Use at least 6 characters.

 ······|

[Create account](#)

By registering an account you agree to our [Terms and Conditions](#) and [Privacy Policy](#).

Already have an account? [Log in](#)

Fig-50. El proceso de registro en TTN.

Una vez registrados accederemos a la consola. Desde ella, se administran los dos elementos principales de una red LoRaWAN que son los nodos y los gateways (en el caso que dispusieramos de gateways propios).

El objetivo de este manual no se centra tanto en nodos individuales sino en aplicaciones, es decir, agrupaciones de nodos dentro de un mismo contexto que permiten ser administrados dentro de una aplicación. Por ello, accederemos al apartado Applications de la consola:

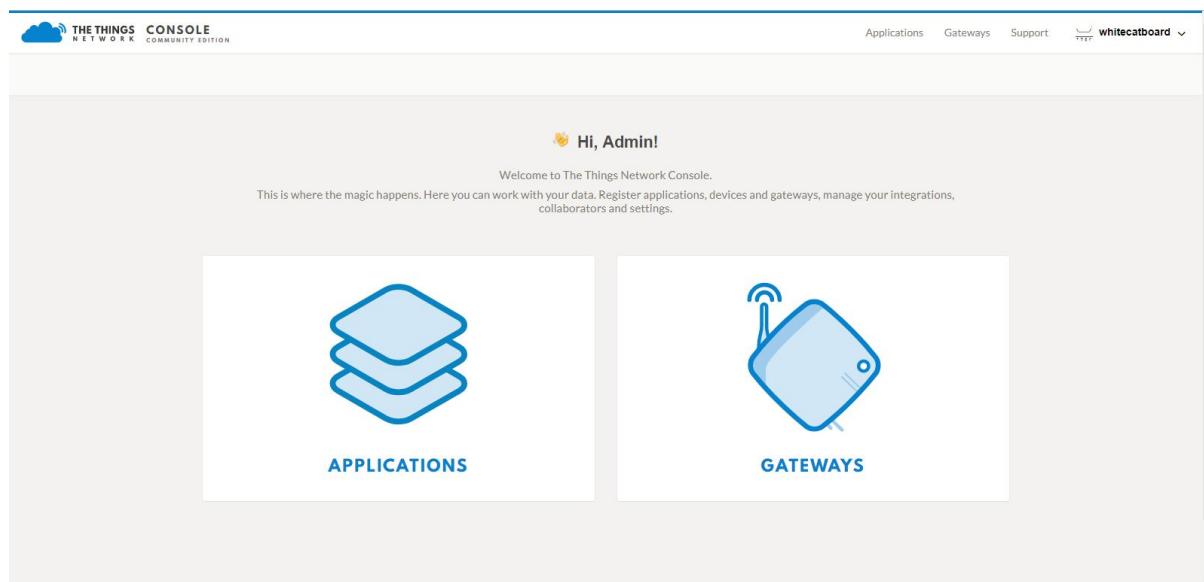


Fig-51. Pantalla principal de la consola de TTN.

En la siguiente pantalla una vez seleccionada las aplicaciones podremos seleccionar, crear o eliminar más aplicaciones.

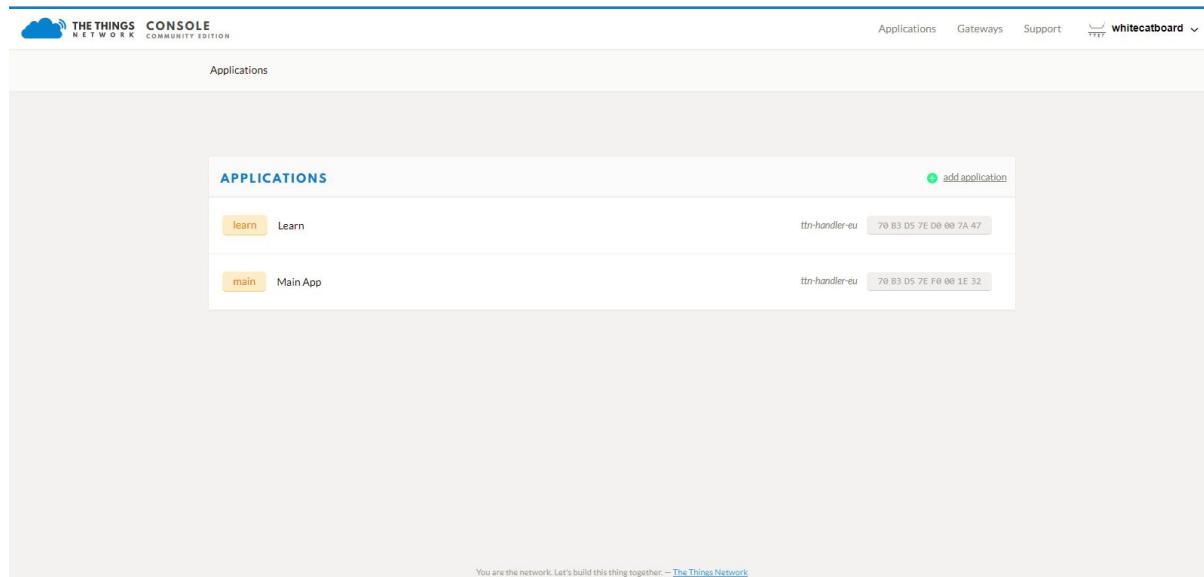


Fig-52. Pantalla de aplicaciones de la consola TTN.

Al seleccionar una aplicación vamos a acceder a una serie de apartados que nos aparecen en la pantalla de la aplicación:



Fig-53. Barra de navegación.

En primer lugar tenemos la barra de navegación que nos indica en qué lugar estamos, en este caso en aplicaciones y dentro de la aplicación main.



Fig-54. Pestañas de opciones.

|                        |   |
|------------------------|---|
| <b>Overview</b>        | La primera pestaña, es el resumen de los parámetros principales de la aplicación que veremos a continuación.  |
| <b>Devices</b>         | La segunda pestaña, es donde realizaremos la gestión de los nodos que enviarán y recibirán información desde la aplicación.   |
| <b>Payload Formats</b> | La tercera pestaña, permite que definamos un código en javascript, para la transformación de la información recibida / enviada, etc. En nuestro caso, como la información se empaqueta y desempaquea con un proceso especial, usaremos esta opción para facilitar el procesamiento de la información.   |
| <b>Integrations</b>    | La cuarta pestaña, permite que agreguemos aplicaciones corriendo también en la nube dentro del ecosistema de TTN , como por ejemplo podría ser la aplicación de Data Storage que permite almacenar los datos recibidos y enviados en una base de datos en la nube de TTN durante un periodo de 7 días y puede ser consultada mediante el uso de una API específica. |
| <b>Data</b>            | La quinta pestaña, permite visualizar en tiempo real la información que está siendo recibida desde los nodos y enviada hacia los nodos. Si la desplegamos nos mostrará además el número de gateways con la información asociada por donde han sido recibidos los paquetes de información.   |
| <b>Settings</b>        | La sexta pestaña, permite especificar las diferentes opciones de la aplicación.   |

A continuación vamos a tratar en profundidad cada uno de estos apartados.

## Overview

Dentro de Overview encontramos:

The screenshot shows a 'APPLICATION OVERVIEW' section. It includes fields for Application ID (set to 'main'), Description ('Main App'), Created ('11 months ago'), and Handler ('ttn-handler-eu (current handler)'). There is also a link labeled 'documentation' in the top right corner.

Fig-55. Sección resumen de la aplicación.

|                             |  |
|-----------------------------|--|
| <b>Application Overview</b> | Desde este apartado se muestra el application ID que será necesario para conectar más adelante mediante MQTT (protocolo que será explicado posteriormente) y la comunicación de los datos.                           |
| <b>Description</b>          | Es simplemente un texto para identificar la aplicación.  |
| <b>Created</b>              | Tiempo que hace que se creó la aplicación.   |
| <b>Handler</b>              | Sistema en la nube que procesa y gestiona nuestra aplicación. En este caso lo gestionaremos desde ttn-handler-eu, que hace referencia a los servidores ubicados en Europa, existiendo también en EEUU, Asia, etc.... |

En el link de documentation arriba a la derecha, accederemos a la documentación que explica cómo conectar con la API de TTN para poder explotar los datos.

The screenshot shows a 'APPLICATION EUIS' section. It displays a single EUI key: '70 B3 D5 7E F0 00 1E 32'. There is a 'manage euis' link in the top right corner.

Fig-56. Claves EUI de la aplicación.

**Application EUIS:** desde este apartado veremos el application EUI y podremos asignar más de uno. EUI es el acrónimo de (Extended Unique Identifier) Identificador Único Extendido, este tipo de identificador está creado para no repetirse e identificar de manera

inequívoca las aplicaciones, también se emplea el mismo concepto en los nodos y sumando el Application EUI y el EUI del nodo se asegura que la información solo llegue al lugar que sea preciso.

## Devices



Fig-57. Número de dispositivos registrados.

Como comentamos al principio, las aplicaciones son agrupaciones de dispositivos y en este apartado gestionaremos los dispositivos que queramos conectar a nuestra aplicación. Pero antes de poder empezar a enviar y recibir información es importante que dispongamos de los nodos (dispositivos) registrados. A continuación y a modo de ejemplo, se muestra una pantalla con todos los dispositivos de una aplicación.

| DEVICES   |     | register device         |
|-----------|-----|-------------------------|
|           | < > | 1 – 7 / 7               |
| node_0001 |     | 4E 4F 44 45 00 00 00 01 |
| node_0002 |     | 4E 4F 44 45 00 00 00 02 |
| node_0003 |     | 4E 4F 44 45 00 00 00 03 |
| node_0004 |     | 4E 4F 44 45 00 00 00 04 |
| node_0005 |     | 4E 4F 44 45 00 00 00 05 |
| node_0006 |     | C4 60 C8 20 97 8E 2F 2D |
| nodo_ota  |     | CB 62 54 18 5B 43 0E 2E |

Fig-58. Lista de dispositivos registrados en la aplicación.

En las aplicaciones, cada dispositivo consta de un nombre para facilitar su identificación y un EUI que en este caso se llama Device EUI.

Los nodos de Whitecatboard.org obtienen el Device EUI desde el arranque del LuRTOS ya que la memoria flash empleada en el dispositivo tiene un identificador único. En otros casos, se emplea la MAC Address o dirección de red del dispositivo que es un número que se registra en fábrica.

En la siguiente figura podemos ver el arranque de LuRTOS desde consola y la indicación de la Flash EUI:

```

Lua RTOS beta 0.1. Copyright (C) 2015 - 2017 whitecatboard.org

build 1512648088
commit e5f1fd94d481f8a10f536d9179b5b03a8c946570
Running from factory partition
board type N1ESP32-DEVKIT
cpu ESP32 rev 0 at 240 Mhz
flash EUI d66634415a653030
spiffs0 start address at 0x310000, size 512 Kb, partition spiffs
spiffs0 mounted
sd0 is at spi2, cs=GPIO21
fat0 can't mounted
can't redirect console messages to file system, an SDCARD is needed

Lua RTOS beta 0.1 powered by Lua 5.3.4
Lua RTOS-boot-scripts-aborted-ESP32
/ > ■

```

Fig-59. Consola mostrando el inicio de LuRTOS con la flash EUI.

Los dispositivos deben autenticarse y codificar los datos que envían y reciben en base a unas claves públicas y privadas, que son las claves de aplicación y de sesión. Estas juntamente a los EUI's, conforman el espacio de direcciones y las variables suficientes para poder ofrecer garantías de seguridad a la comunicación en red.

Existen dos tipos de dispositivos en función de cómo gestionan esas claves público/privadas que comentamos para codificar/decodificar la información: OTA y ABP.

### **-OTA (On the Air)**

Se refiere a dispositivos que se registran solamente con una AppKey y un Device EUI. Cuando el dispositivo se conecta y mediante esos dos valores, negocia con el servidor la obtención de una clave de sesión para cifrar las comunicaciones en los 2 puntos.

Es un sistema más rápido a la hora de desplegar nodos ya que solamente es preciso emplear el AppKey y en cada sesión puede cambiar la clave. Por contrapartida, a la hora de realizar comunicaciones es más lento, ya que se establece un protocolo de comunicación envío / recepción entre la red y el nodo para negociar precisamente las claves.

### **-ABP (Address By Personalization)**

En este caso las claves para codificar la información se programan desde fábrica y no se cambian durante toda la vida del dispositivo. Es más rápido en términos de comunicación ya

que de antemano TTN y el nodo conocen las claves de codificación y por lo tanto no es preciso que se obtengan en ninguna comunicación.  
A continuación vemos los parámetros de un dispositivo.

The screenshot shows a 'DEVICE OVERVIEW' page with the following details:

- Application ID:** main
- Device ID:** node\_0001
- Activation Method:** ABP
- Device EUI:** 4E 4F 44 45 00 00 00 01
- Application EUI:** 70 B3 D5 7E F0 00 1E 32
- Device Address:** 26 01 13 12
- Network Session Key:** (redacted)
- App Session Key:** (redacted)
- Status:** 13 days ago
- Frames up:** 0 ([reset frame counters](#))
- Frames down:** 0

Fig-60. Resumen parámetros dispositivo.

|                            |  |
|----------------------------|--|
| <b>Application ID</b>      | El identificador de la aplicación.   |
| <b>Device ID</b>           | El identificador del dispositivo.  |
| <b>Activation Method</b>   | ABP / OTA el método de activación del dispositivo como hemos visto anteriormente.                              |
| <b>Device EUI</b>          | El identificador único de dispositivo.   |
| <b>Application EUI</b>     | El identificador único de aplicación.  |
| <b>Device Address</b>      | La dirección única del dispositivo.  |
| <b>Network Session Key</b> | La clave de sesión. En el caso de ABP se genera en la creación del dispositivo, en el caso de OTA es dinámica. |
| <b>App Session Key</b>     | Del mismo modo que el Network Session Key en ABP se genera a la creación del dispositivo y en OTA es dinámica. |

|                    |  |
|--------------------|--|
| <b>Status</b>      | Color y tiempo que indican si el dispositivo está activo o no (verde y rojo) y la última vez que envió o recibió un mensaje desde la aplicación en minutos, días, etc.   |
| <b>Frames Up</b>   | El número de paquetes de información que ha enviado el dispositivo hacia el servidor. Existe la posibilidad de resetear el contador. Hay que tener en cuenta que por motivos de seguridad se pretende que el contador sea único para que no se hagan reenvío de paquetes de un modo malintencionado. |
| <b>Frames Down</b> | El número de paquetes que el servidor ha enviado al nodo.  |

The screenshot shows the 'DOWNLINK' section of a configuration interface. It includes fields for 'Scheduling' (with options 'replace', 'first', and 'last'), 'FPort' (set to 1), and a checked 'Confirmed' checkbox. The 'Payload' section shows 'bytes' selected and a field containing '0 bytes' with a green checkmark icon.

Fig-61. Sección para introducir un downlink (paquete desde el servidor al dispositivo).

Es posible desde la consola del dispositivo enviar un downlink, esto es, un paquete de información desde el servidor al dispositivo. Es importante saber, como se remarca en el siguiente párrafo, que actualmente en una red LoRaWAN siempre es el nodo el que inicia la comunicación. Por ese motivo los downlink caducan por tiempo.



The screenshot shows the 'SIMULATE UPLINK' section of a configuration interface. It includes fields for 'FPort' (set to 1) and 'Payload' (containing '0 bytes' with a green checkmark icon).

Fig-62. Sección para simular un uplink (paquete desde un dispositivo al servidor).

También es posible desde la consola del dispositivo simular un uplink, información como si llegara del dispositivo al servidor para probar por ejemplo, los scripts en el apartado de Payload Formats.

Fig-63. Sección de usuarios con acceso a la aplicación.

Además en TTN y mediante la opción de Collaborators, es posible colaborar con otros usuarios en la gestión de la red. De este modo podemos asignar permisos para que otros usuarios accedan a la información por ejemplo de diferentes nodos, o puedan gestionar gateways, etc.... Por defecto sólo el usuario registrado que ha creado la aplicación tendrá permisos para gestionarla.

Fig-64. Claves de acceso a la aplicación empleadas por ejemplo para el acceso vía MQTT.

Por último referente a la aplicación tenemos el Access Keys. Con estas claves por ejemplo, es posible como veremos más adelante, acceder a la cola MQTT de la aplicación para obtener los datos de los dispositivos.

## Payload Formats

```

1 function Bytes2Float32(bytes) {
2     bytes = parseInt(bytes);
3
4     var sign = (bytes & 0x80000000) ? -1 : 1;
5     var exponent = ((bytes >> 23) & 0xFF) - 127;
6     var significand = (bytes & ~(-1 << 23));
7
8     if (exponent == 128)
9         return sign * ((significand) ? Number.NaN : Number.POSITIVE_INFINITY);
10
11    if (exponent == -127) {
12        if (significand == 0) return sign * 0.0;
13        exponent = -126;
}

```

Fig-65. Consola para programar en JavaScript los payload formats.

Como se comentó con anterioridad, es posible definir una serie de scripts en Javascript que permiten realizar tareas con la información “payload” que se envía y se recibe en la red. En la sección Payload Formats es donde se definen y prueban dichos scripts.

Podemos definir un script por cada tipo de acción: decoder (para la recepción de datos), converter (para convertir de un formato a otro), validator (para validar que la información es correcta) o encoder (para el envío de datos).

## Integrations

The screenshot shows the 'INTEGRATION OVERVIEW' section of a web application. It displays the following information:

- Status:** Running (indicated by a green dot)
- Integration info:** [go to platform](#)
- Platform:** Data Storage (v2.0.1) (represented by a database icon)
- Author:** The Things Industries B.V.
- Description:** Stores data and makes it available through an API. Your data is stored for seven days.

Fig-66. Sección que muestra una integración con la aplicación de TTN.

Con Integration Overview es posible integrar aplicaciones de terceros dentro de la aplicación de TTN. De este modo, es posible realizar otro tipo de tareas con la información que recibe o envía nuestra aplicación. En este caso como se puede observar la integración que tenemos es una integración de almacenamiento en una base de datos la url que nos devuelve es externa y con la información de la API es posible que accedemos a los datos.

En este caso en concreto de almacenamiento, los datos se conservarán durante 7 días. Una vez pasado ese tiempo serán eliminados.

## Data

The screenshot shows the 'APPLICATION DATA' section of a web application. It includes the following interface elements:

- Filters:** uplink, downlink, activation, ack, error (with 'uplink' selected)
- Time controls:** pause, clear
- Columns:** time, counter, port

Fig-67. Sección que muestra los datos que se envían o reciben desde la aplicación.

Desde Application Data podremos ver la información que se envía y se recibe a través de la red.

Los paquetes se muestran con la información del payload procesado mediante el payload format y además incluyen información del nodo que lo envía, el gateway por donde llega, tiempo, etc.

Los tipos de paquetes que podemos monitorizar son: recibidos (uplink), enviados (downlink), activación (para dispositivos OTA), ack (paquetes con confirmación) y error (frames con error).

## Settings

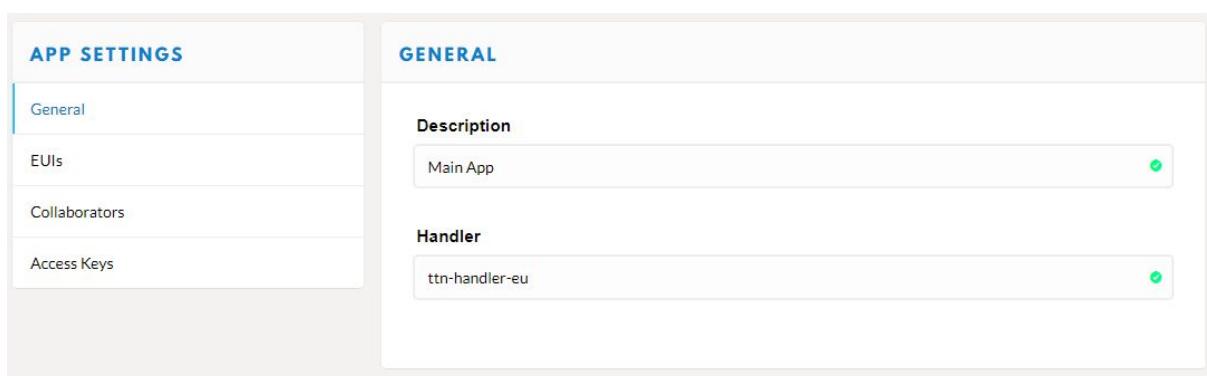


Fig-68. Sección de ajustes de la aplicación.

Por último la pestaña de App settings nos permite definir las opciones de la aplicación que vimos con anterioridad.

Para finalizar y con el objetivo de poder realizar los casos de uso del capítulo 8, vamos a ver como crear un nodo ABP y un nodo OTA.

### - Creación de un Nodo OTA

Para la creación de un nodo OTA es muy sencillo, solamente necesitaremos especificar los siguientes parámetros, como muestra la imagen:

**REGISTER DEVICE**

[bulk import devices](#)

**Device ID**  
This is the unique identifier for the device in this app. The device ID will be immutable.

nodo\_ota

**Device EUI**  
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

01 02 03 04 05 06 07 08 8 bytes

**App Key**  
The App Key will be used to secure the communication between your device and the network.

this field will be generated

**App EUI**

70 B3 DS 7E F0 00 1E 32

Cancel Register

Fig-69. Sección para registrar un dispositivo.

donde Device ID (nombre que queramos darle), DeviceEUI (devuelto por LuRTOS al arranque), AppKey será generado y AppEUI es el de la Aplicación. Con estos parámetros ya podemos generar el dispositivo.

### - Creación de un Nodo ABP

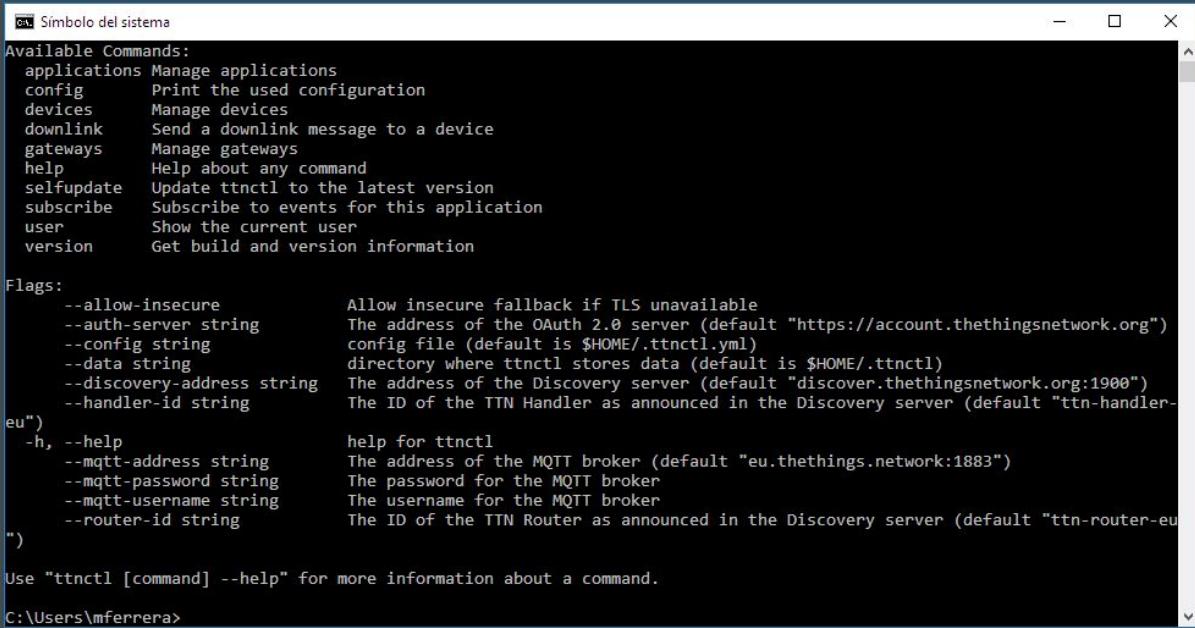
Para generar un nodo ABP el proceso es el mismo que se ha explicado anteriormente pero seleccionando en overview la opción ABP antes de realizar la activación. De todos modos, vamos a ver como generarlo utilizando la herramienta ttncctl, que es una herramienta de consola para la gestión de los parámetros de TTN tal y como hemos visto en su formato WEB, pero permite realizar ciertas tareas que no están disponibles en la web como es la creación de un nodo ABP.

En primer lugar descargamos la herramienta ttncctl desde la web de TTN, en:

<https://www.thethingsnetwork.org/docs/network/cli/quick-start.html>

Lo descomprimimos y lo copiamos en alguna ruta dentro del path y lo renombramos a ttncctl. Por ejemplo en Windows lo ideal es copiarlo a la carpeta Windows con el nombre ttncctl.exe.

Una vez tengamos acceso lo ejecutamos y vemos una salida como la mostrada a continuación:



```

Símbolo del sistema
Available Commands:
  applications      Manage applications
  config            Print the used configuration
  devices           Manage devices
  downlink          Send a downlink message to a device
  gateways          Manage gateways
  help              Help about any command
  selfupdate        Update ttncctl to the latest version
  subscribe         Subscribe to events for this application
  user              Show the current user
  version           Get build and version information

Flags:
  --allow-insecure      Allow insecure fallback if TLS unavailable
  --auth-server string   The address of the OAuth 2.0 server (default "https://account.thethingsnetwork.org")
  --config string        config file (default is $HOME/.ttncctl.yml)
  --data string          directory where ttncctl stores data (default is $HOME/.ttncctl)
  --discovery-address string The address of the Discovery server (default "discover.thethingsnetwork.org:1900")
  --handler-id string   The ID of the TTN Handler as announced in the Discovery server (default "ttn-handler-eu")
  -h, --help             help for ttncctl
  --mqtt-address string The address of the MQTT broker (default "eu.thethings.network:1883")
  --mqtt-password string The password for the MQTT broker
  --mqtt-username string The username for the MQTT broker
  --router-id string    The ID of the TTN Router as announced in the Discovery server (default "ttn-router-eu")

)
Use "ttncctl [command] --help" for more information about a command.
C:\Users\mferrera>

```

Fig-70. Consola con herramienta ttncctl.

En primer lugar, necesitaremos acceder a la cuenta del mismo modo que hacíamos en formato web. Para hacer esto iremos a la dirección <https://account.thethingsnetwork.org/>

Desde allí copiaremos el código de acceso que nos facilitaran en la web.

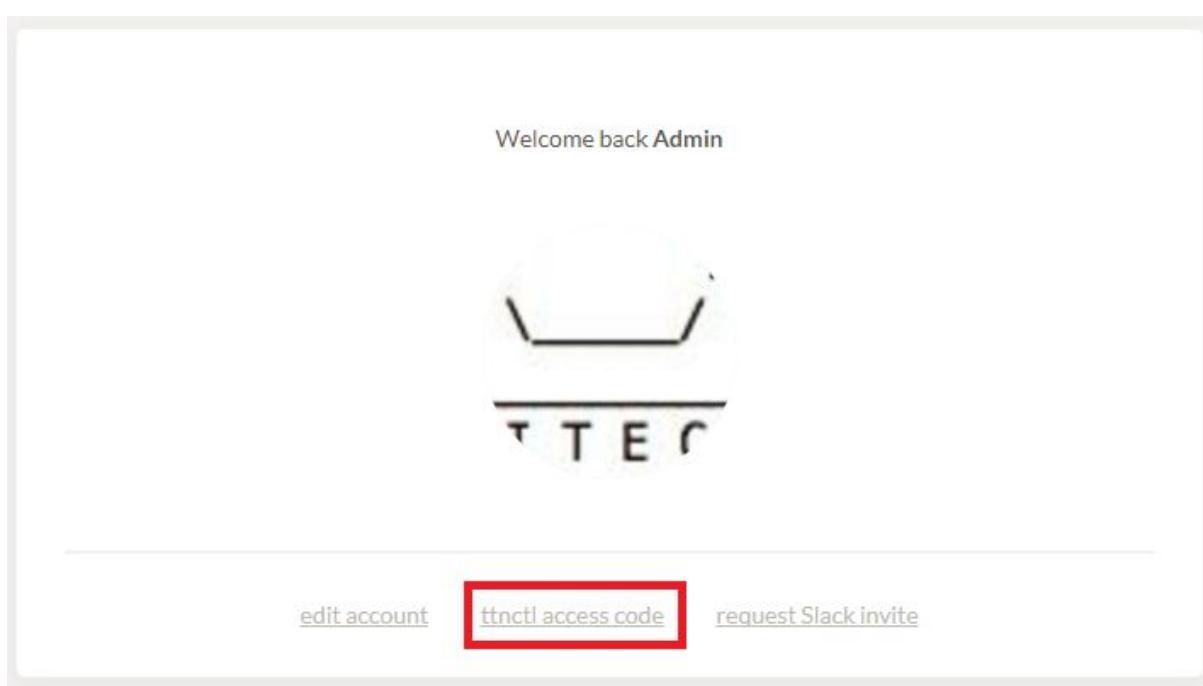


Fig-71. Enlace para solicitar código de acceso a la consola mediante ttncctl.

Si pinchamos en el enlace de ttncctl access code nos facilitarán un código válido para acceder mediante la herramienta ttncctl durante 5 minutos.

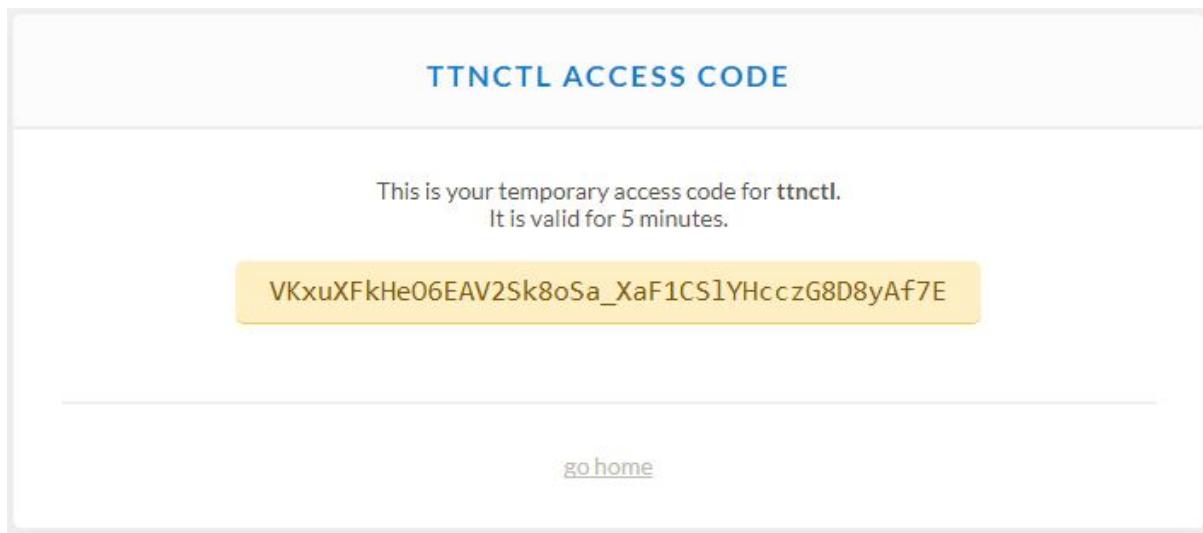


Fig-72. Código de acceso temporal proporcionado por el sistema.

Una vez tenemos el código accedemos mediante el siguiente comando:

***ttncctl user login VKxuXFkHe06EAV2Sk8oSa\_XaF1CSlYHcczG8D8yAf7E***

```

Símbolo del sistema
help      Help about any command
selfupdate Update ttncctl to the latest version
subscribe Subscribe to events for this application
user       Show the current user
version   Get build and version information

Flags:
--allow-insecure          Allow insecure fallback if TLS unavailable
--auth-server string       The address of the OAuth 2.0 server (default "https://account.thethingsnetwork.org")
--config string             config file (default is ${HOME}/.ttncctl.yml)
--data string               directory where ttncctl stores data (default is ${HOME}/.ttncctl)
--discovery-address string The address of the Discovery server (default "discover.thethingsnetwork.org:1900")
--handler-id string        The ID of the TTN Handler as announced in the Discovery server (default "ttn-handler-eu")
-eu
-h, --help                 help for ttncctl
--mqtt-address string      The address of the MQTT broker (default "eu.thethings.network:1883")
--mqtt-password string     The password for the MQTT broker
--mqtt-username string     The username for the MQTT broker
--router-id string          The ID of the TTN Router as announced in the Discovery server (default "ttn-router-eu")

Use "ttncctl [command] --help" for more information about a command.

C:\Users\mferrera>ttncctl user login VKxuXFkHe06EAV2Sk8oSa_XaF1CSlYHcczG8D8yAf7E
INFO Successfully logged in as whitecatboard (admin@whitecatboard.org)

C:\Users\mferrera>

```

Fig-73. Autenticación en ttncctl con el código.

Y obtendremos la siguiente pantalla que nos indica que ya hemos accedido a nuestra cuenta de usuario.

El siguiente paso es seleccionar la aplicación donde queremos crear el nodo. Para ello listamos las aplicaciones para seleccionar una de ellas, mediante el comando:

***ttncctl applications list***

```

Símbolo del sistema
--auth-server string      The address of the OAuth 2.0 server (default "https://account.thethingsnetwork.org")
--config string            config file (default is $HOME/.ttnctl.yml)
--data string              directory where ttncctl stores data (default is $HOME/.ttnctl)
--discovery-address string The address of the Discovery server (default "discover.thethingsnetwork.org:1900")
--handler-id string        The ID of the TTN Handler as announced in the Discovery server (default "ttn-handler-eu")
-h, --help                 help for ttncctl
--mqttt-address string    The address of the MQTT broker (default "eu.thethings.network:1883")
--mqttt-password string   The password for the MQTT broker
--mqttt-username string  The username for the MQTT broker
--router-id string         The ID of the TTN Router as announced in the Discovery server (default "ttn-router-eu")

Use "ttncctl [command] --help" for more information about a command.

C:\Users\mferrera>ttncctl user login VKxuXFkHe06EAV2Sk8oSa_XaF1CSlyHcczG8D8yAf7E
INFO Successfully logged in as whitecatboard (admin@whitecatboard.org)

C:\Users\mferrera>ttncctl applications list
INFO Found 2 applications:

  ID      Description      EUIs      Access Keys      Collaborators
1  learn    Learn           1          1                  1
2  main    Main App         1          1                  1

C:\Users\mferrera>

```

Fig-74. Listado de aplicaciones.

Una vez que tengamos la lista de aplicaciones seleccionaremos la que nos interesa con el comando:

***ttncctl applications select 2***

```

Símbolo del sistema
C:\Users\mferrera>ttncctl applications select
INFO Found 2 applications:

  ID      Description
1  learn    Learn
2  main    Main App

Which one do you want to use?
Enter the number (1 - 2) > 2
INFO Found one EUI "70B3D57EF0001E32", selecting that one.
INFO Updated configuration                         AppEUI=70B3D57EF0001E32 AppID=main

C:\Users\mferrera>

```

Fig-75. Selección de aplicación para trabajar.

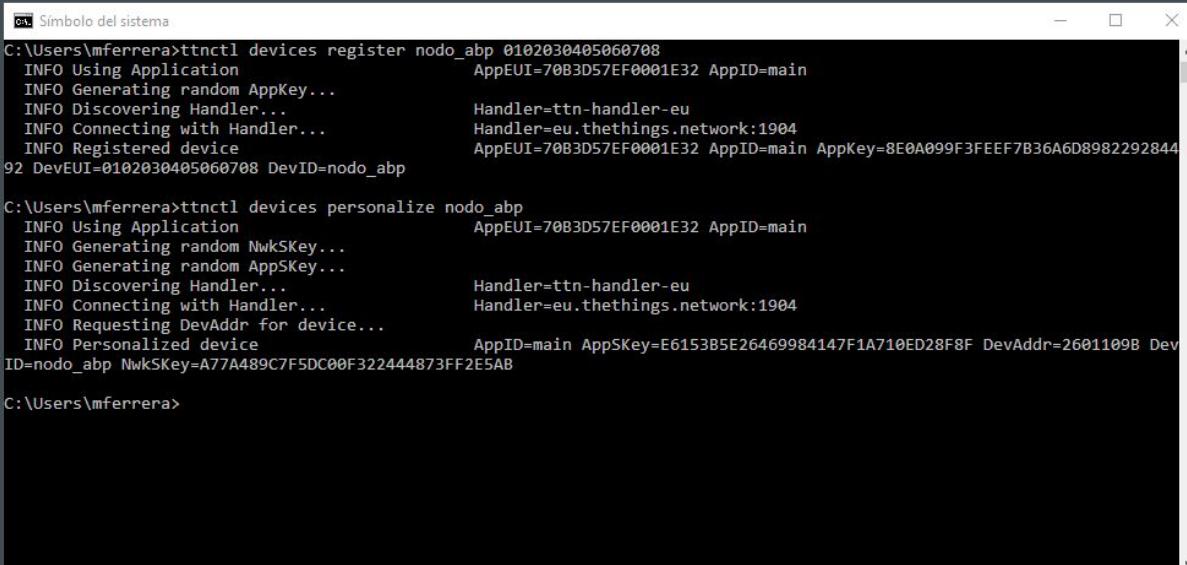
Nos aparecerá la siguiente pantalla confirmando la selección de aplicación.

El siguiente paso será crear el nodo mediante el siguiente comando:

***ttncctl devices register nodo\_abp 0102030405060708***

y acto seguido lo personalizamos para indicar a TTN que será un nodo ABP:

***ttnctl devices personalize nodo\_abp***



```
C:\Users\mferrera>ttnctl devices register nodo_abp 0102030405060708
INFO Using Application AppEUI=70B3D57EF0001E32 AppID=main
INFO Generating random AppKey...
INFO Discovering Handler... Handler=ttn-handler-eu
INFO Connecting with Handler... Handler=eu.thethings.network:1904
INFO Registered device AppEUI=70B3D57EF0001E32 AppID=main AppKey=8E0A099F3FEEF7B36A6D8982292844
92 DevEUI=0102030405060708 DevID=nodo_abp

C:\Users\mferrera>ttnctl devices personalize nodo_abp
INFO Using Application AppEUI=70B3D57EF0001E32 AppID=main
INFO Generating random NwkSKey...
INFO Generating random AppSKey...
INFO Discovering Handler... Handler=ttn-handler-eu
INFO Connecting with Handler... Handler=eu.thethings.network:1904
INFO Requesting DevAddr for device...
INFO Personalized device AppID=main AppSKey=E6153B5E26469984147F1A710ED28F8F DevAddr=2601109B DevID=nodo_abp NwkSKey=A77A489C7F5DC00F322444873FF2E5AB

C:\Users\mferrera>
```

Fig-76. Registro de nuevo dispositivo.

De este modo ya tendríamos creado el nodo en formato ABP para poder realizar las pruebas de los siguientes casos de uso (capítulo 8). Desde la web, podemos comprobar que el nodo se ha creado correctamente, así como su formato, confirmando que TTN lo ha creado como ABP.

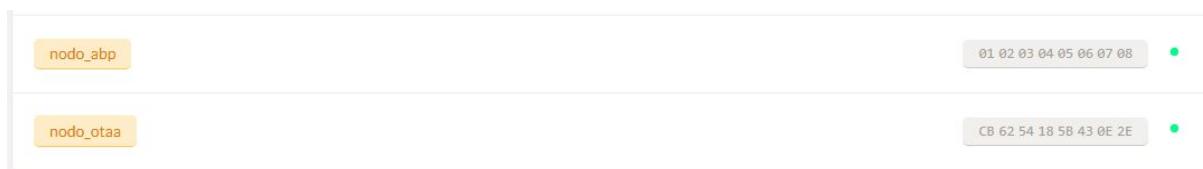


Fig-77. Resumen listado dispositivos desde la web para confirmar su registro.

En la imagen podemos confirmar que TTN ha creado los dos nodos con las opciones que hemos especificado, uno en método OTA y otro en método ABP.

Ya los tendríamos preparados para las prácticas que desarrollaremos en el capítulo 8.

El último punto es acceder a la información. Este punto se abordará en el siguiente capítulo y se desarrolla cuando se explique el objeto MQTT del editor de Bloques de NodeRED, ya que el método preferido para acceder a la información en tiempo real en TTN será a través de su servidor MQTT.

MQTT es el acrónimo de Message Queuing Transport Telemetry. Es un protocolo desarrollado por los laboratorios I+D de IBM que tiene una estructura de publicación suscripción que lo hacen muy interesante para el desarrollo de aplicaciones IoT como es el caso que nos ocupa. En el siguiente capítulo abordaremos más en profundidad su uso.

## 7. La “Nube” (IBM Bluemix y NodeRED)

En el capítulo anterior hemos visto cómo enviar y recibir información desde un espacio en la nube hasta los nodos, y desde estos últimos a la nube. El siguiente paso es ver que hacemos con dicha información.

En el entorno whitecatboard.org proponemos el uso de tecnologías de servidor consolidadas que permitan disponer de un sistema en la nube robusto y eficaz para el desarrollo de la parte de servidor, aquella que aporta la visión global de todos los dispositivos que están enviando información y va a permitir desarrollar las soluciones que trabajen con los datos de dichos dispositivos y sensores.

La solución en la nube consta de dos partes: una parte de infraestructura, es decir allí donde va a residir la lógica y la herramienta con la que vamos a generar dicha lógica, que llamaremos genérica IBM Bluemix y otra parte que es la herramienta con la que desarrollaremos e implementaremos la lógica desde el servidor, que llamaremos Node-RED.

## 7.1 IBM Bluemix

Como comentamos en el párrafo anterior, desde nuestro punto de vista IBM Bluemix es para nosotros un conjunto de productos y servicios que van a permitir que nuestra solución funcione en un entorno robusto y que permita despreocuparse de la infraestructura.

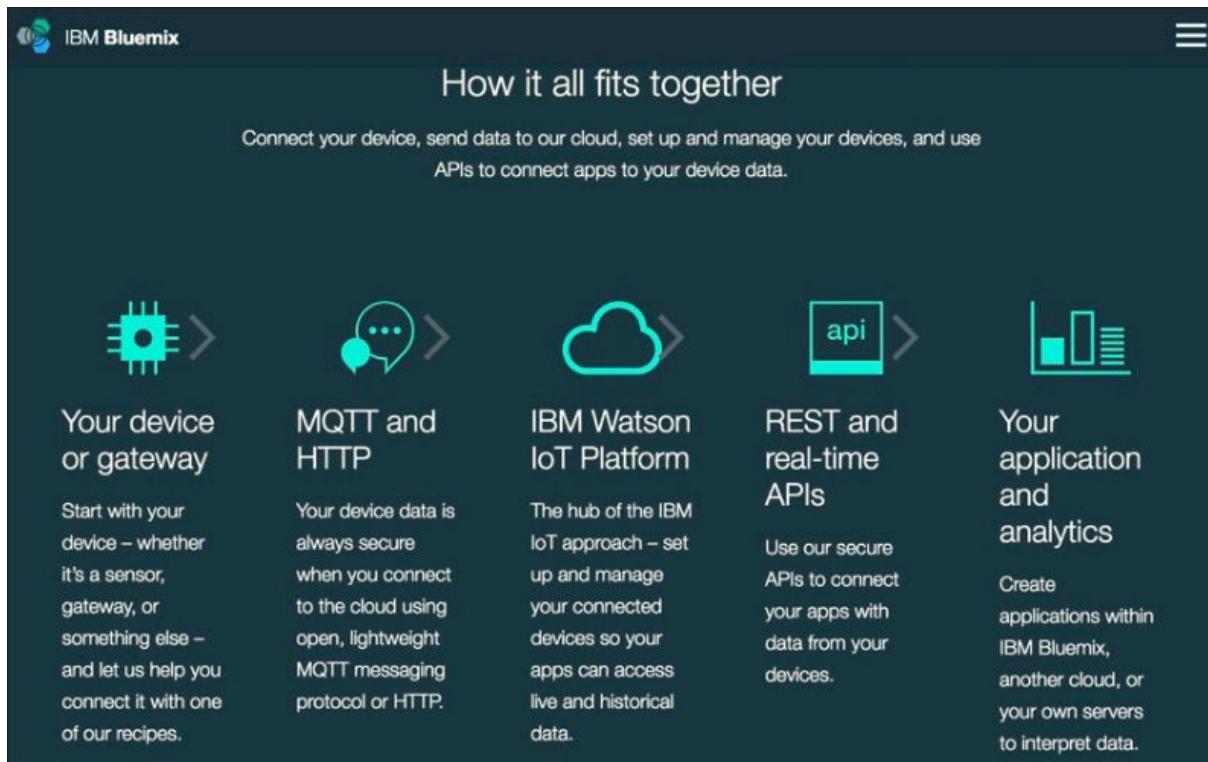


Fig-78. Resumen productos y servicios bluemix.

La solución basada en la nube permite desplegar máquinas virtuales que corren y solicitan recursos bajo demanda. Es el equivalente a disponer de un sistema SAS pero en hardware y software para el despliegue de soluciones IoT.

Además, toda la tecnología de IA por ejemplo de IBM con Watson, puede formar parte de dicha solución, permitiendo una escalabilidad en el futuro muy interesante.

Actualmente, nuestro uso de bluemix se centra en el servicio de máquina virtual y sobre ella hacemos correr el sistema de NodeRED.

A continuación, mostramos los rasgos más importantes en diferentes pantallas de la consola de administración de Bluemix.

-Pantalla donde están nuestros servicios (servidores) corriendo:

<https://idaas.iam.ibm.com/idaas/mtfim/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:basicldapuser>



Fig-79. Acceso a herramienta cloud bluemix.

-Pantalla de login a la URL de IBM Cloud donde accederemos a la gestión de nuestros productos y servicios.

Fig-80. Panel de control “dashboard” principal.

Catálogo de recursos del IBM Cloud: una vez que accedemos podemos visualizar los diferentes recursos. En este caso podemos ver operando una app de cloud foundry que en realidad es un servicio de máquina virtual.

También podemos ver corriendo un servicio que se instala por defecto de una base de datos en el momento que ponemos en marcha el servidor virtual.

Lo importante de IBM Cloud y Bluemix es que el servicio o producto se modifica bajo demanda, modo que la facturación por el servicio o producto también es variable en función del uso.

Por otra parte, uno de los elementos interesantes es que IBM Cloud permite un periodo de evaluación que comprende un mes de uso sin cargo alguno. Este tipo de servicio es el que se propone en las prácticas que se verán en el capítulo siguiente.

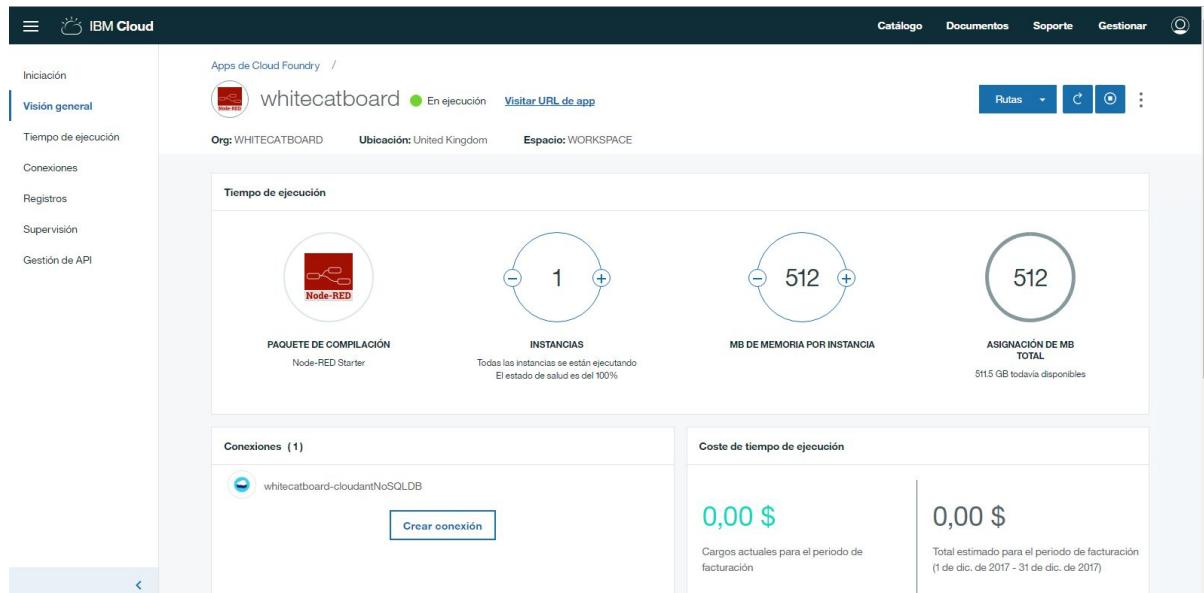


Fig-81. Pantalla de servicios desplegados.

En la figura anterior, mostramos un ejemplo del dashboard de IBM Cloud donde vemos, entre otros, el servicio corriendo y el resumen de facturación. Desde esta pantalla se pueden configurar diferentes opciones de nuestro servicio. En nuestro caso, lo que nos interesa es la aplicación NodeRED, que veremos a continuación.

## 7.2 NodeRED

NodeRED es una herramienta diseñada por IBM para la programación visual mediante bloques y en entorno Web de aplicaciones que corren sobre un servidor Node.js.

Node.js es un servidor que ejecuta Javascript en una máquina virtual propia sin necesidad de que los programas escritos en ese lenguaje se ejecuten en la parte cliente, normalmente un navegador.

Históricamente Javascript siempre se ha ejecutado dentro de un navegador web por lo tanto siempre se ha ejecutado en la parte cliente. Con Node.js se aprovechó el desarrollo de aplicaciones en Javascript para que estas se pudieran ejecutar también por ejemplo en un servidor.

Esto permite aprovechar todo el potencial del lenguaje de programación Javascript en entornos de servidor para el despliegue de aplicaciones orientadas al manejo de la información en el servidor de datos y procesos en la nube.

Por tanto como se comentaba anteriormente, NodeRed es un conjunto de módulos desarrollados en Javascript que permiten de un modo sencillo e intuitivo la programación de eventos más complejos en un ámbito de red distribuido.

A continuación, podemos ver la pantalla de inicio de sesión para poder acceder al workspace o entorno de programación de NodeRED:

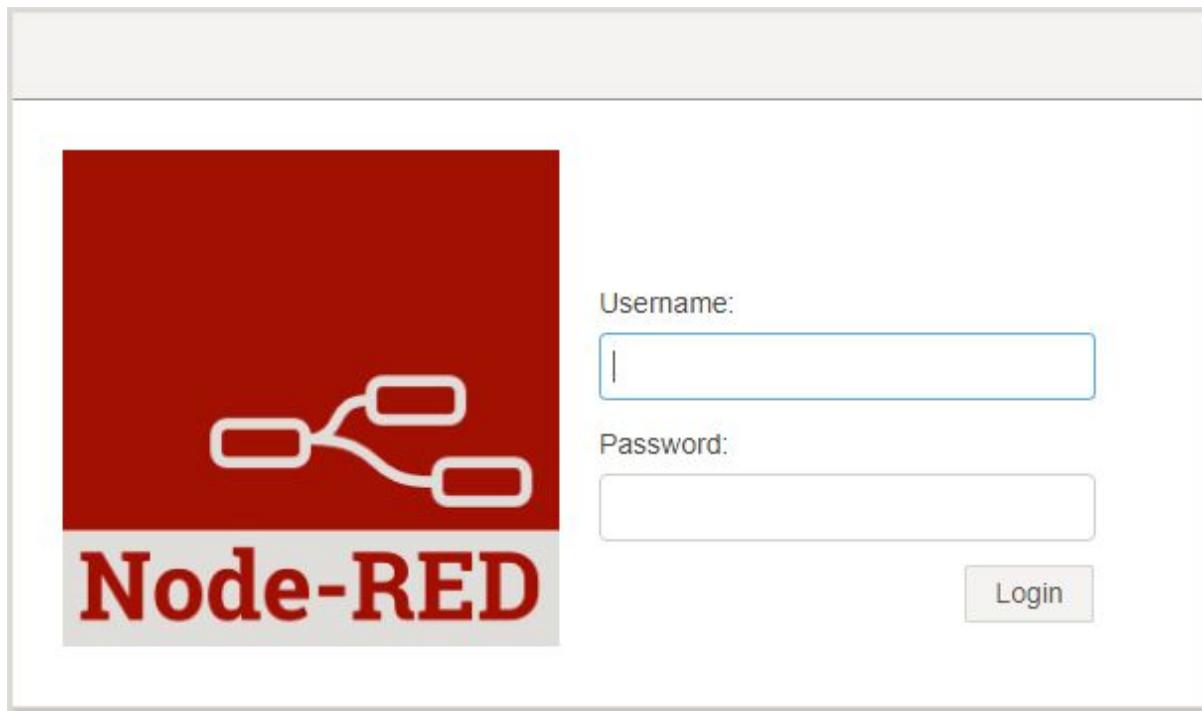


Fig-82. Pantalla de acceso a Node-RED.

Mediante esta pantalla accederemos al área de trabajo de NodeRed que pasamos a describir a continuación.

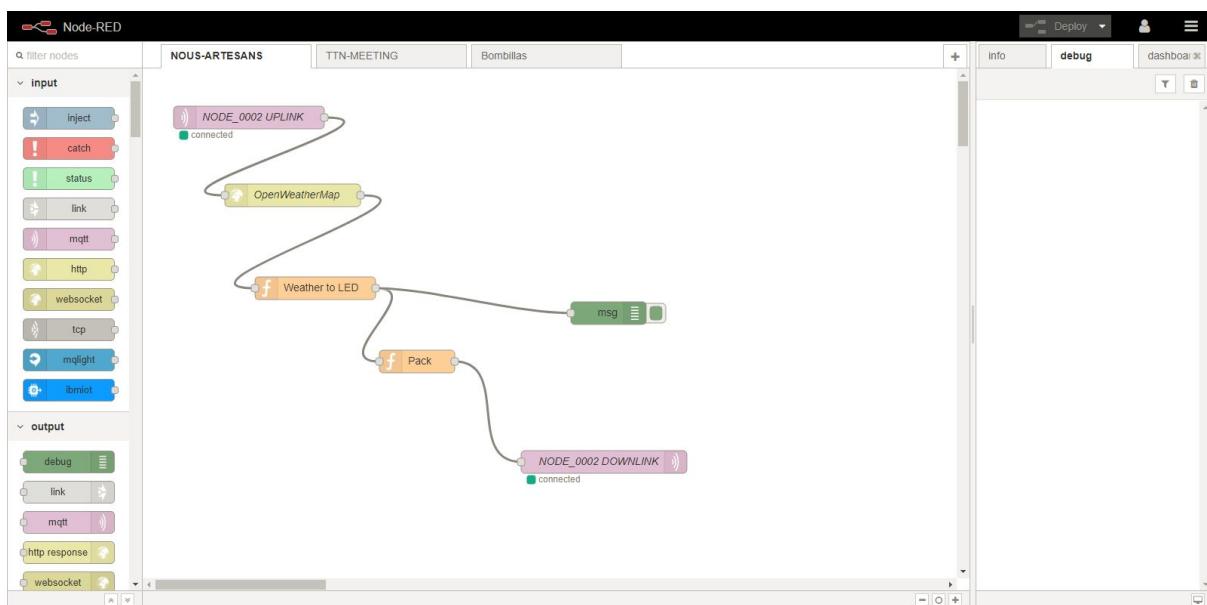


Fig-83 Área de trabajo de Node-RED.

La figura anterior muestra la pantalla principal, el área de trabajo de NodeRED. A continuación veremos cada uno de los apartados del área de trabajo.

-Barra superior:



Fig-84. Icono de NodeRED

Logotipo de Node-RED, en según qué instalaciones este logotipo se puede personalizar para adecuarlo a las necesidades estéticas del proyecto / cliente.

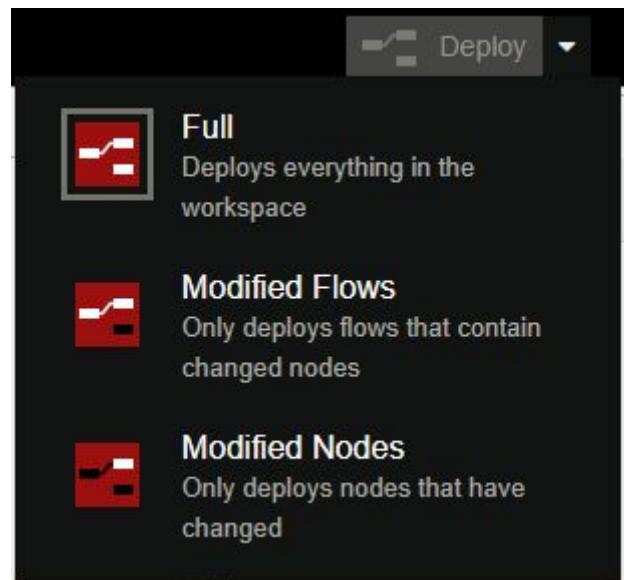


Fig-85. Opciones de despliegue del “workflow”

Cada vez que se modifique algo en las áreas de trabajo se puede desplegar en el servidor para que empiecen a funcionar los nuevos bloques y su código. En este momento si hubiera cualquier error en nuestros bloques o código, NodeRED nos lo notifica antes de proseguir con el despliegue.

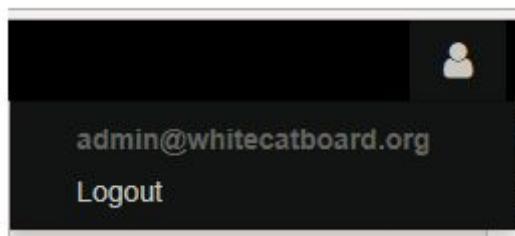


Fig-86. Perfil de usuario.

-Icono de usuario: identifica la cuenta con la que hemos accedido y permite que hagamos un logout para cerrar la sesión.

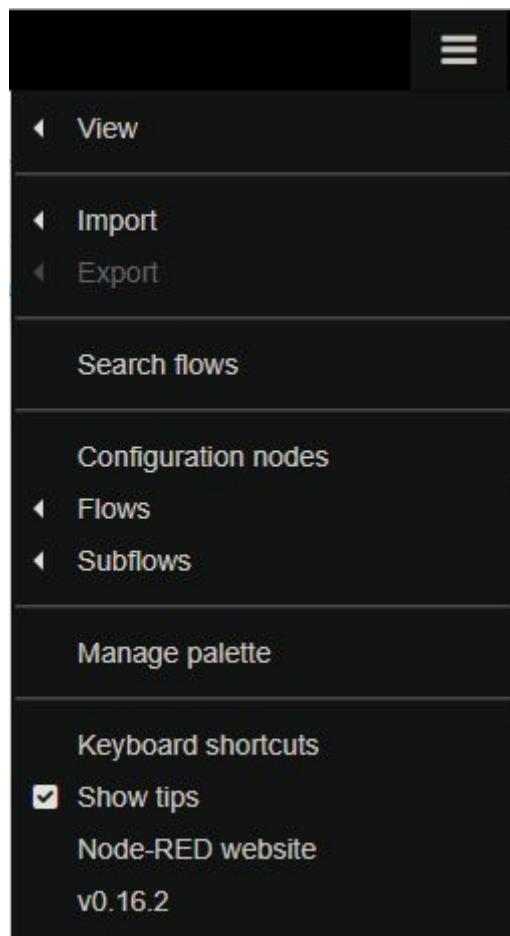


Fig-87 Menu opciones barra superior.

-Menú de opciones de NodeRED: desde este menú se permite cambiar aspectos de la configuración a nivel de las vistas, importar / exportar o eliminar “flows” que es como se llaman los programas en NodeRED.

También, se pueden crear “subflows”, es decir, pequeñas partes de bloques / código dentro de partes mayores.

Además, se pueden modificar los parámetros de los nodos (bloques) de configuración para aquellos nodos que requieran dichos parámetros, como por ejemplo la conexión a servidores, etc.



Fig-88. Categorías de los bloques de NodeRED.

En la parte izquierda tenemos los nodos, que son los bloques que podemos arrastrar al área de trabajo. Como podemos apreciar en la imagen, están divididos en diferentes secciones, de modo que cada una de ellas tiene diferentes nodos relacionados para hacer tareas específicas dentro de cada sección.

A continuación vemos unos cuantos nodos de ejemplo dentro de la categoría input.



Fig-89. Bloques de la categoría input.

Estos son los nodos que aparecen en la categoría de entradas y todos ellos hacen referencia a maneras en las que podemos recibir información, desde una petición web, desde MQTT, desde la plataforma IBM IoT, desde un socket TCP, etc.

En nuestro caso trabajaremos con LoRa desde TTN. Así, en los casos prácticos emplearemos principalmente los nodos de MQTT, y también los emplearemos en los ejemplos que se realicen empleando la conectividad WiFi.

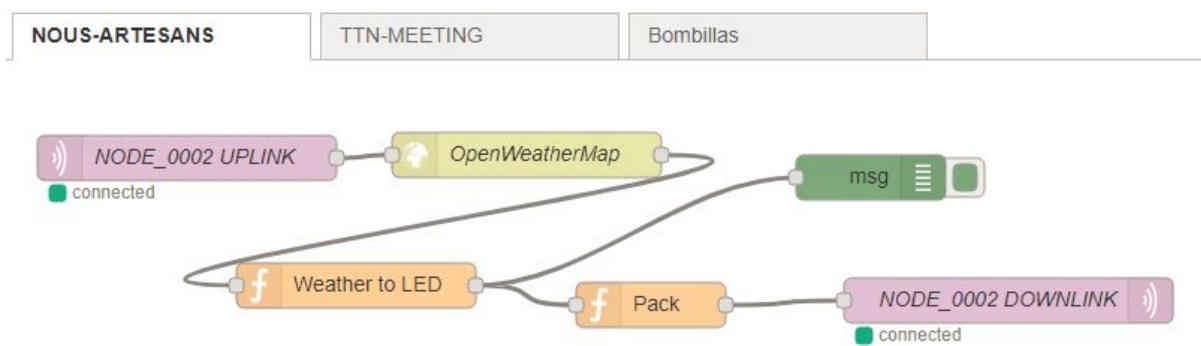


Fig-90. Ejemplo de un workflow.

En la imagen podemos ver los diferentes flows en las diferentes pestañas y los nodos que componen uno de los flows. Como se puede observar, el modelo de programación en bloques se basa en entradas y salidas en cada bloque.

Para profundizar en el uso de la herramienta de NodeRED se puede acceder al sitio de desarrollo y documentación en la siguiente URL:

<https://nodered.org/docs/>

## Documentation



Fig-91. Sección de ayuda de NodeRED en la web.

En su web, se dispone de una documentación muy bien elaborada y muy extensa sobre cada una de las funcionalidades de NodeRED.

A parte de la gran potencia que posee NodeRED para el desarrollo de aplicaciones de manera visual, uno de sus valores clave es que es Software Libre y su desarrollador principal es IBM.

Es importante tener conocimientos de Javascript para poder aprovechar al máximo las posibilidades de NodeRED y como veíamos con anterioridad lo mismo sucede con los scripts que podemos configurar en los payload functions de TTN.

## 8. Casos Prácticos

A continuación presentamos las “fichas” de varios casos prácticos que se han diseñado con el objetivo de empoderar al usuario de un modo gradual. Así, se parte de unos ejemplos muy simples donde se puedan comprender los diferentes tipos de elementos que pueden conformar un dispositivo considerado IoT, hasta el último caso práctico donde se pondrá en valor todo lo aprendido en los primeros casos.

Cada caso práctico está explicado de una forma simple y sencilla para facilitar su comprensión.

En cada ficha se explicará el objetivo del caso, los elementos necesarios así como el esquema de conexión mediante un diagrama, el código en bloques y el código en Lua.

También se explicará la parte local y la parte que se ejecuta en el servidor, en el caso práctico que lo precise.

## 8.1. Caso 1 (Configuración de un sensor)

### - Descripción

En este primer caso vamos a conectar un sensor BME280 de presión, temperatura y humedad y mostrar los datos recogidos por el sensor en la consola de la ESP32N1.

### - Objetivos

El objetivo de este primer caso práctico básico, es familiarizarnos con los diferentes elementos del kit así como con el entorno de desarrollo y empezar con los primeros bloques.

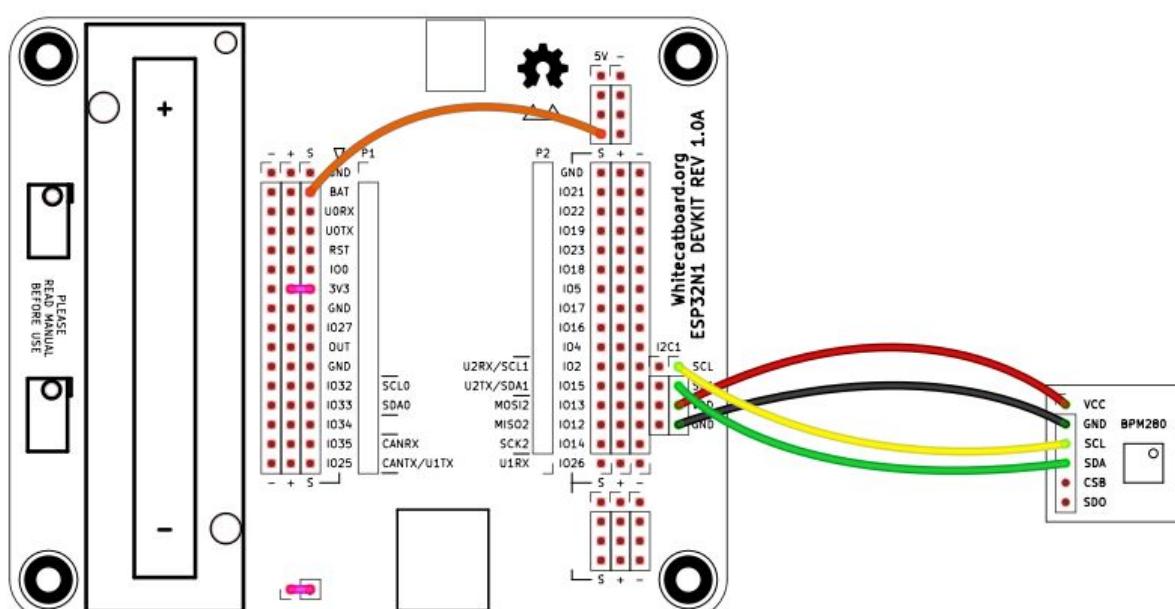
### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BOSCH BME280
- 1x SET DE CABLES
- 1x PC CON CONEXION USB

### - Diagrama de Conexión

Este es el diagrama de conexión (recordar que si no se emplea batería hay que usar el cable naranja desde el pin marcado como bat a un pin +5V)



fritzing

Fig-92. Diagrama de conexión.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

Este sensor opera por el puerto I2C. En nuestro caso y como muestra el diagrama, lo conectaremos al puerto I2C número 1. Para ello será preciso conectar VCC, GND, SCL y SDA.

Para estandarizar las prácticas y las líneas de datos en los cables, etc, se recomienda seguir también el patrón de colores.

Una vez conectado todo como se indica en el diagrama procederemos a usar los bloques necesarios para la lectura de los valores que aporta el sensor y su impresión por consola.

#### - Programa en bloques



Fig-93. Creación de un sensor.

Para poder emplear el sensor BME280 necesitaremos configurarlo en el entorno. Para ello, seleccionamos la opción sensores y ahí dentro nuevo sensor.

Acto seguido seleccionaremos por ejemplo, la temperatura como magnitud a medir ya que es una de las que dispone el sensor. Para finalizar, seleccionaremos el sensor llamado BME280 de la lista y el puerto I2C el número 1, y le asignaremos un nombre al sensor, en este caso aparece por defecto como sensor1.



Fig-94. Creación de un sensor (continuación).

Una vez creado el sensor procederemos a añadir el resto de los bloques para la práctica, quedando del siguiente modo:



Fig-95. Programa de bloques.

Con estos bloques se realizará un bucle infinito en la placa, en el cual se imprimirá por la consola los 3 valores obtenidos del mismo sensor, se hará una pausa de 5 segundos y se repetirá la operación. En la consola podemos ver el valor de los bloques para imprimir.



The screenshot shows a software interface for the Whitecat N1 ESP32 DEVKIT. At the top, there is a menu bar with "Whitecat N1 ESP32 DEVKIT" and "caso1.xml". There is also a language selection dropdown set to "Español" and a user profile icon.

The main window displays a series of sensor readings and control messages:

```
21.63152
***** FIN LECTURA VALORES SENSOR *****
***** INICIO LECTURA VALORES SENSOR *****
48.37318
993.1817
21.63776
***** FIN LECTURA VALORES SENSOR *****
***** INICIO LECTURA VALORES SENSOR *****
48.34554
993.1424
21.62653
***** FIN LECTURA VALORES SENSOR *****
***** INICIO LECTURA VALORES SENSOR *****
48.27376
993.1638
21.63402
***** FIN LECTURA VALORES SENSOR *****
```

Fig-96. Valores mostrados en consola Humedad %, Presión mBar, Temperatura °C.

## 8.2. Caso 2 (Envío de información a través de la red LoRaWAN)

### - Descripción

En este segundo caso vamos a seguir utilizando el sensor BME280 de presión, temperatura y humedad pero en esta ocasión vamos a mostrar los datos recogidos por el sensor en la consola de administración de TTN.

### - Objetivos

El objetivo de este segundo caso práctico es, una vez asimilados los conocimientos del primer caso práctico, realizar el envío de los datos recogidos por el sensor a la nube mediante el uso de la red LoRa. Al final de la práctica el usuario habrá aprendido cómo enviar los datos a la nube para su procesamiento mediante TTN.

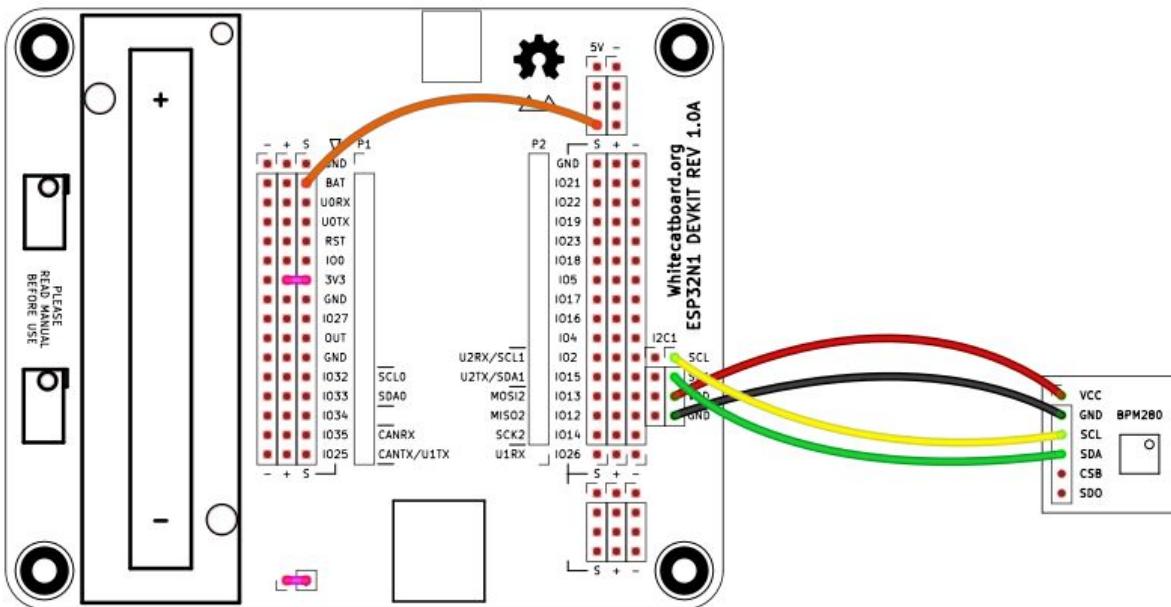
### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BOSCH BME280
- 1x SET DE CABLES
- 1x CONECTIVIDAD LoRaWAN (GATEWAY O ÁREA CON COBERTURA LoRaWAN)
- 1x PC CON CONEXION USB

### - Diagrama de Conexión

Este es el diagrama de conexión (recordar si se emplea batería usar el cable naranja desde el pin marcado como bat a un pin +5V).



fritzing

Fig-97. Diagrama de conexión.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

Tal y como se puede ver en el diagrama anterior las conexiones son las mismas que en el primer caso práctico.

## - Programa en bloques

En esta segunda práctica ya dispondremos del sensor creado en el primer caso, pero deberemos configurar la red LoRaWAN en el dispositivo. Para realizar esta tarea, procederemos de un modo similar a como hicimos con el sensor, en este caso desde el apartado de LoRa.

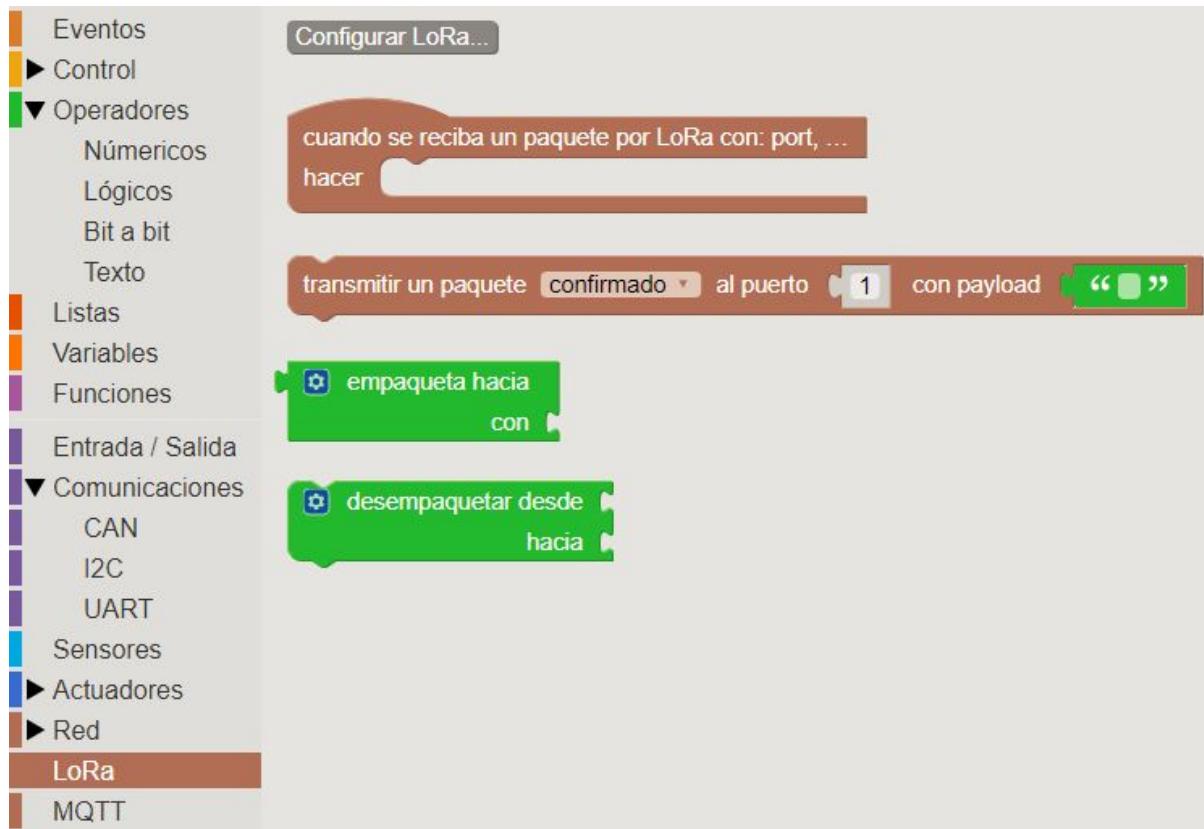


Fig-98. Configuración parámetros LoRaWAN.

Seleccionar el apartado de LoRa y seleccionar la opción Configurar LoRa.

Antes de configurar LoRa será preciso que tengamos los datos desde la consola TTN. Tal como vimos en el ejemplo del capítulo 6 de TTN, usaremos el nodo de ejemplo de ABP que creamos que se llamaba nodo\_abp, y vamos a consultar sus datos.

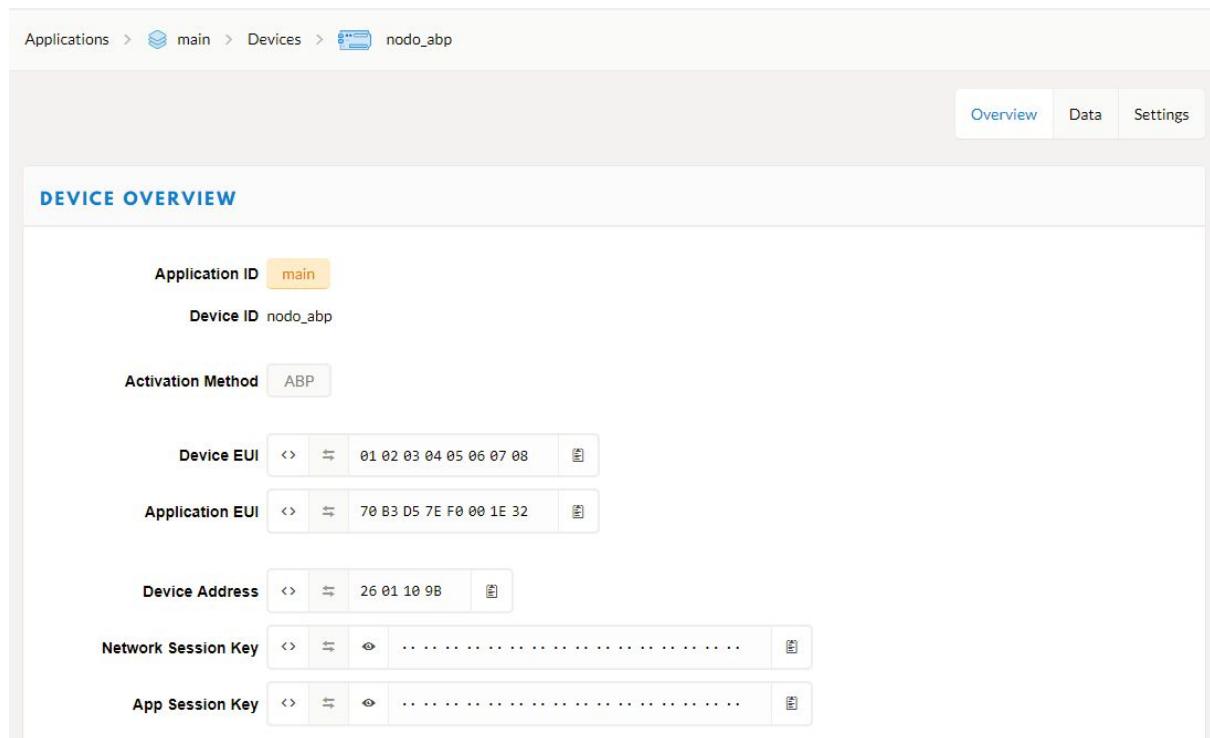


Fig-99 . Pantalla de TTN de resumen de dispositivo para obtener los parámetros de configuración.

En la consola de administración de TTN seleccionamos el nodo\_abp y desde la opción de device overview copiamos los datos precisos para la conexión.

Volvemos a la configuración en bloques y los introducimos de modo que como tienen exactamente el mismo nombre, quedaría tal como muestra la siguiente imagen:

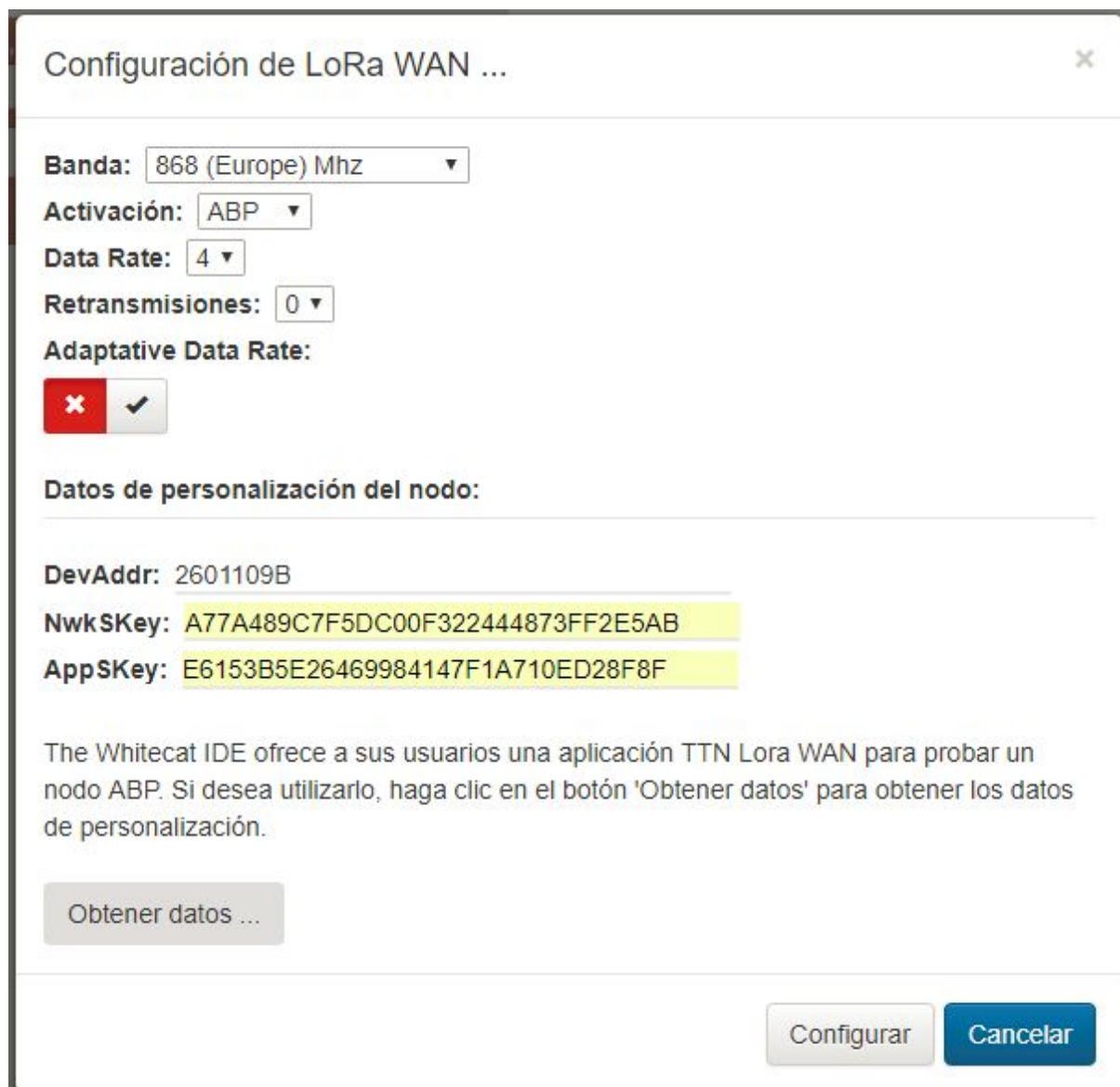


Fig-100. Introducción de los parámetros de configuración.

Así nos quedaría la configuración de la red LoRa copiando los parámetros de la consola de TTN.

Una vez completado los parámetros de conexión ya podemos añadir el resto de bloques para que la práctica quede como se muestra en la siguiente figura:



Fig-101. Programa de bloques.

En la consola veremos el siguiente texto:



The screenshot shows the Whitecat N1 ESP32 DEVKIT software interface. At the top, there is a header bar with the text "Whitecat N1 ESP32 DEVKIT" and a dropdown menu, followed by a file icon labeled "caso2.xml" and a language selection "Español". On the right side of the header is a user profile icon. The main area is a large text box containing the following text:

```
Lua RTOS beta 0.1 powered by Lua 5.3.4
Lua RTOS-boot-scripts-aborted-ESP32
/ > require("block");wcBlock.delevepMode=true;dofile("/_run.lua")
/ > <blockStart,5>
<blockStart,4>
<blockEnd,4>
Paquete LoRa enviado
```

Fig-102. Salida del programa en consola

Si todo es correcto, en la consola de TTN veremos la información tal y como llega desde el nodo.

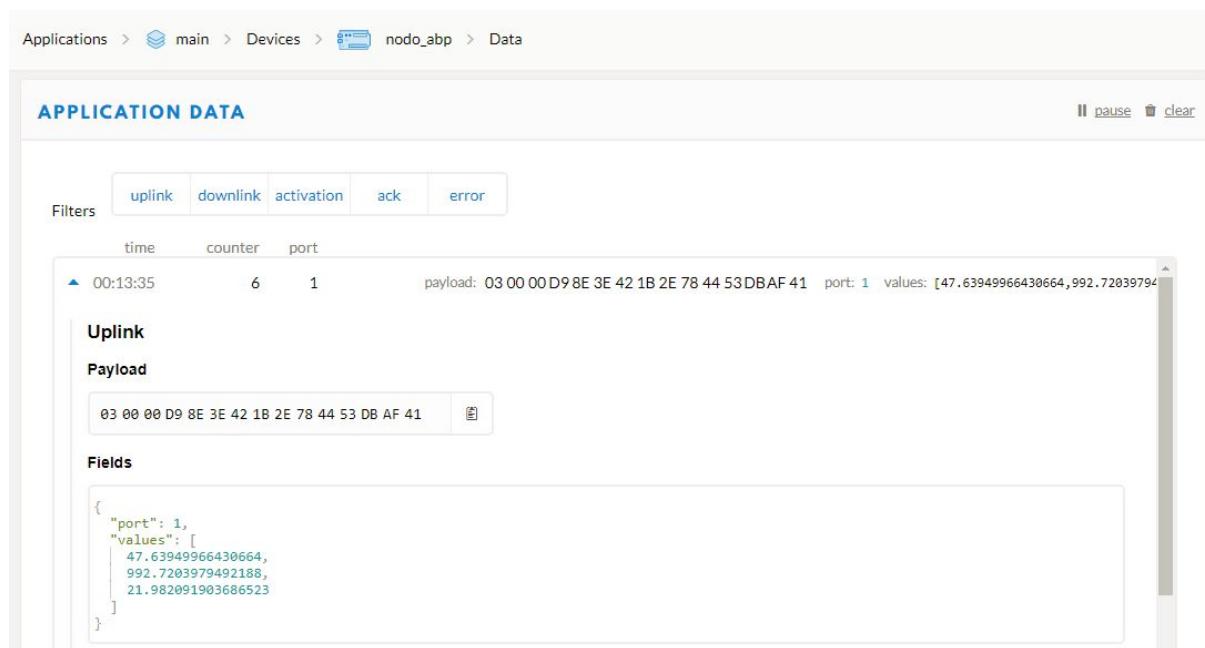


Fig-103. Datos que han llegado a TTN.

**ENHORABUENA!** Si esto es lo que observas en la consola de TTN, es que tu nodo está enviando la información del sensor cada 30 segundos de una manera correcta, como se puede apreciar en la imagen.

Con este resultado hemos aprendido a enviar información desde el nodo a la nube empleando para ello la red de comunicaciones IoT de LoRaWAN. En la siguiente práctica veremos cómo recibir un valor ahora desde la nube hacia el nodo y operar en consecuencia en la placa.

## 8.3. Caso 3 (Recepción de información a través de la red LoRaWAN)

### - Descripción

En este tercer caso vamos a conectar la tira de leds direccionables NeoPixel y a encender y apagar leds de dicha tira en base a un valor enviado desde TTN.

### - Objetivos

El objetivo de este tercer caso práctico es una vez asimilados los conocimientos del primer y segundo caso práctico, ver como se envían ahora los datos desde la nube al nodo. De este modo, completamos las opciones de comunicación de un nodo con el servidor TTN.

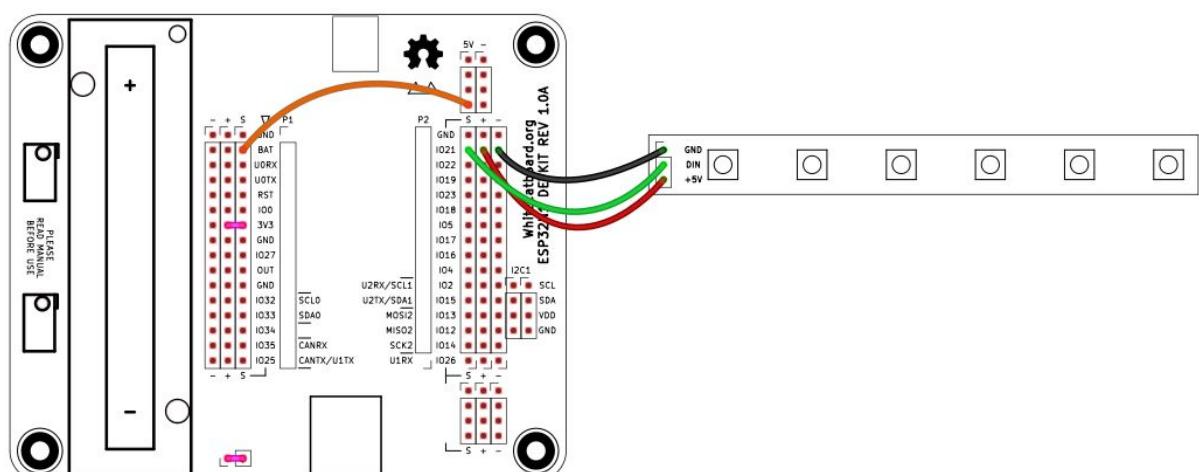
### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x TIRA 6 LEDS NEOPixel
- 1x SET DE CABLES
- 1x CONECTIVIDAD LoRaWAN (GATEWAY O ÁREA CON COBERTURA LoRaWAN)
- 1x PC CON CONEXION USB

### - Diagrama de Conexión

Este es el diagrama de conexión (recordar si no se emplea batería usar el cable naranja desde el pin marcado como bat a un pin +5V)



fritzing

Fig-104. Diagrama de conexión.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

Como se comentó en el capítulo de actuadores, los leds NeoPixel usan un protocolo parecido al 1-Wire, de tal manera que en el caso del ejemplo y tal como muestra el diagrama de conexiones, sólo necesitaremos 3 cables conectados a la placa: 2 para alimentación y un cable para datos en el IO21.

### - Programa en bloques

En este caso mantenemos la configuración de Lora como la teníamos en el caso anterior pero agregaremos la opción de recibir datos.

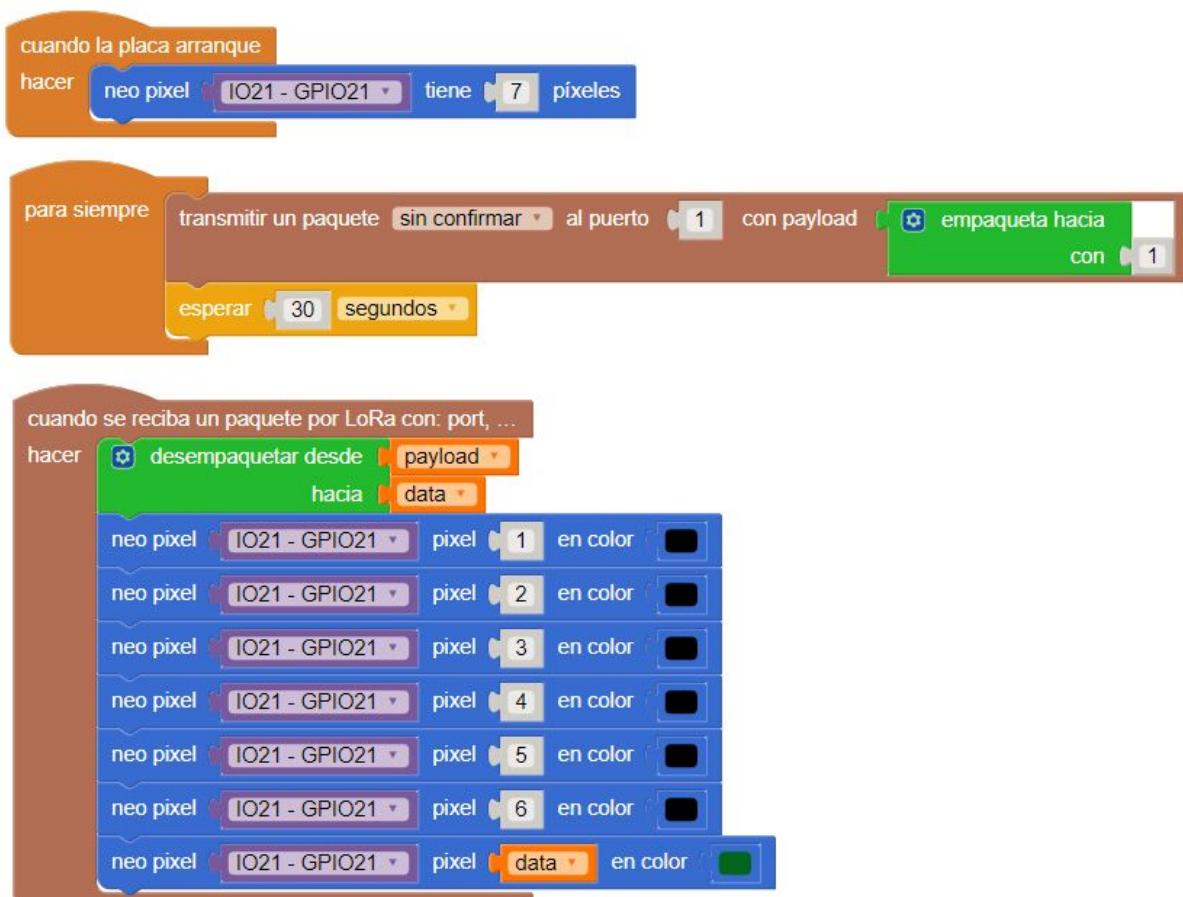


Fig-105. Programa de bloques.

Cuando se inicia el programa se definen el número de leds que hay en la tira. En este caso ponemos uno más, ya que el 7 lo usaremos para apagar todos los leds.

Acto seguido, es importante recordar que los nodos tipo A de LoRaWAN como es nuestro caso, son siempre los que inician la comunicación por eso mantenemos la transmisión de un paquete, en este caso con el valor 1 fijo, simplemente a efectos de iniciar la comunicación con el servidor.

Acto seguido esperamos para recibir un paquete LoRaWAN desde el servidor. Si es así, el paquete esperado es un número entre 1 y 7 que indicará a NeoPixel encender en color Verde.

Recordemos que en las funciones de Format Payload teníamos el método de downlink para empaquetar. Por ese motivo, usamos el bloque desempaquetar hacia una variable que es la que empleamos después.

Como se muestra en la siguiente imagen, la manera de transmitir el dato hacia el nodo, se realiza mediante la opción downlink del device. Por ejemplo, para encender el led número 3 en la tira de NeoPixel pondremos dentro del device overview algo como lo que se muestra a continuación:



Fig-106. Envío de un paquete LoRaWAN desde el servidor en la consola de TTN.

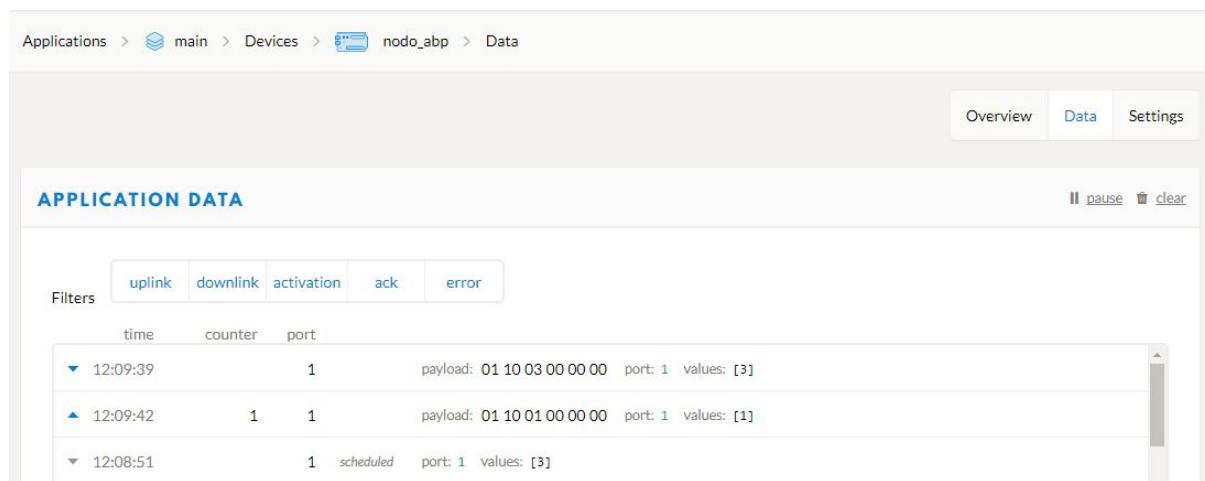


Fig-107. Preparación y envío del paquete de datos

Al realizar el proceso de envío del downlink desde la consola de datos, aparecerá primero como scheduled y posteriormente cuando recibimos el payload para iniciar la comunicación el valor es enviado.

Simplemente queda confirmar visualmente que el led número 3 se ha encendido en color verde. Si es así, podremos dar la práctica por finalizada, y decir que ya somos capaces de enviar órdenes por LoRaWAN también hacia los nodos.

En el siguiente caso de uso emplearemos la red WiFi para realizar una comunicación de datos vía MQTT contra un servidor MQTT público en este caso el de [iot.eclipse.org](http://iot.eclipse.org).

## 8.4. Caso 4 (Publicación de información mediante MQTT)

### - Descripción

En este cuarto caso utilizaremos también el sensor BME280 y realizaremos el envío de la información por MQTT a través de la red WiFi local. Posteriormente, usaremos la opción Dashboard de Node-RED para informar de los valores recibidos de un modo más visual.

### - Objetivos

El objetivo de este cuarto caso es que el usuario se familiarice con la configuración de la conectividad a través de una red WiFi, el uso y configuración del protocolo MQTT para publicar información y el uso de los indicadores de Node-RED para mostrar la información recibida.

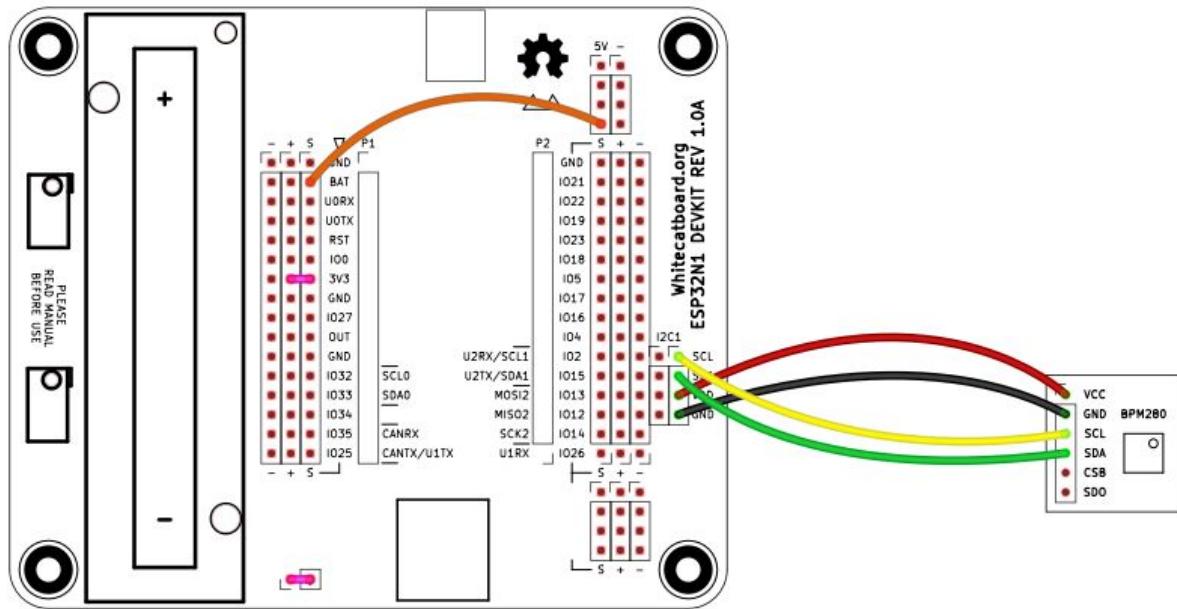
### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BME280
- 1x SET DE CABLES
- 1x CONECTIVIDAD WIFI
- 1x BROKER MQTT (en este ejemplo usaremos iot.eclipse.org)
- 1x PC CON CONEXION USB

### - Diagrama de conexión

El conexionado del sensor será idéntico al primer caso de uso.



fritzing

Fig-108. Diagrama de conexión



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

## - Programa en Bloques

Lo primero que deberemos hacer para iniciar el programa de este caso de uso, es ajustar los parámetros de configuración de la red WiFi, en este caso el SSID (Nombre de Red) y la clave que se utilizará para conectar a dicha red.

Seleccionamos la categoría WiFi y a continuación Configurar....

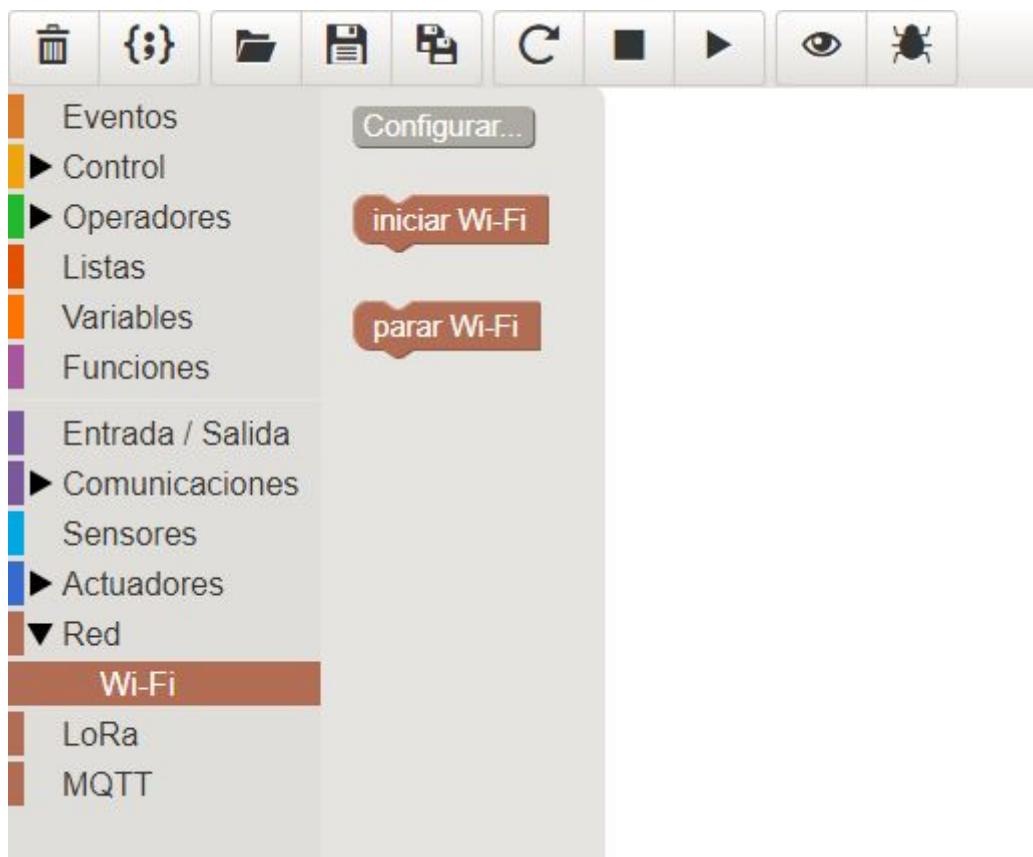


Fig-109. Configuración conexión WiFi.

Una vez que aparece el recuadro de configuración de WiFi introducimos los datos para la conexión.

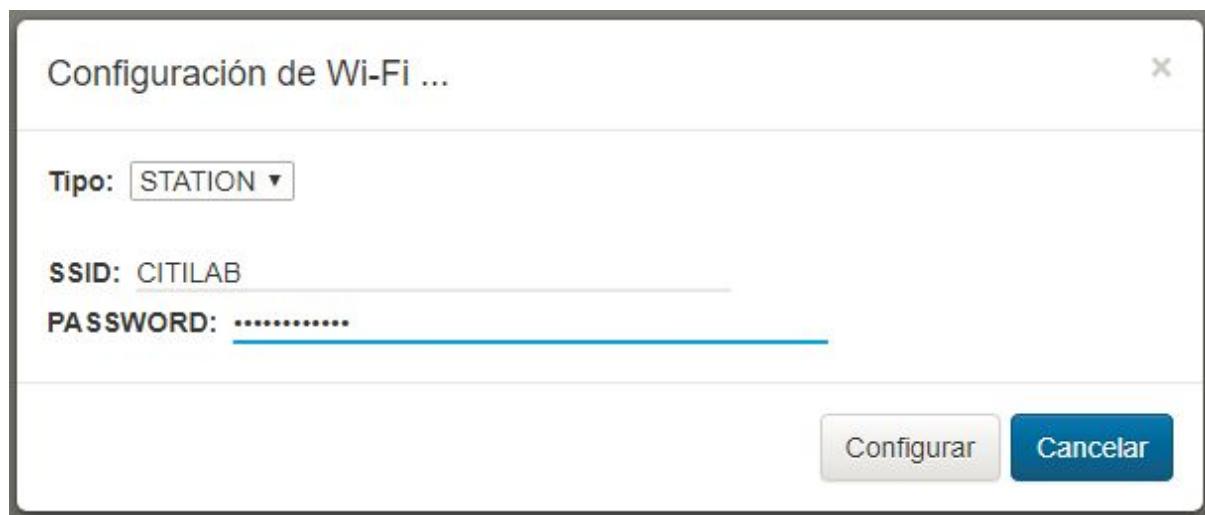


Fig-110. Parámetros configuración WiFi.

Una vez que hayamos configurado la conexión a la red WiFi, el siguiente paso será configurar los datos del servidor MQTT (Broker). Más adelante explicaremos como funciona el sistema MQTT para que se pueda entender que hace el servidor llamado Broker normalmente.

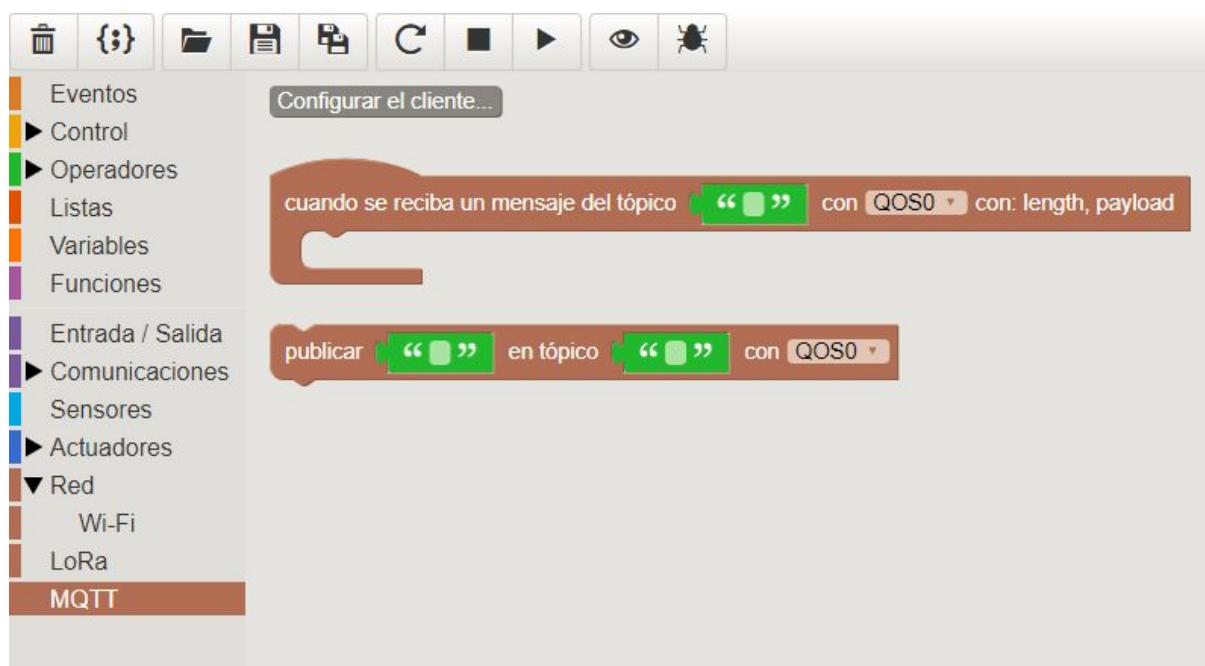


Fig-111. Selección de la configuración MQTT



Fig-112. Introducción parámetros MQTT

|                 |  |
|-----------------|--|
| <b>ClientID</b> | Nombre identificador del cliente que se conecta puede cualquier valor aleatorio números letras, etc se emplea para identificar al cliente conectado, se precisa un nombre que no se repita |
| <b>Host</b>     | Dirección IP o nombre del servidor MQTT (Broker) al que vamos a conectar en el ejemplo usaremos el de iot.eclipse.org  |
| <b>Port</b>     | Puerto al que vamos a conectar con el Broker normalmente es el 1883  |
| <b>Secure</b>   | Si se habilita esta opción nos permitirá conectar con un nombre de usuario y clave   |



Fig-113. Programa en bloques

El programa en bloques está dividido en dos apartados. Como se puede observar, lo primero que hacemos es usar un bloque que indica lo que se debe realizar cuando la placa arranque. En dicho bloque estableceremos en este caso la conexión WiFi y los parámetros iniciales del sensor.

En el segundo apartado, se realiza un bucle continuo que lee la información del sensor como son la humedad, presión atmosférica y temperatura y las publica en tres “topics” diferentes cada uno representado por el mismo nombre de la lectura.

Posteriormente realiza una pausa de 5 segundos y vuelve a realizar la lectura y envío de la información.

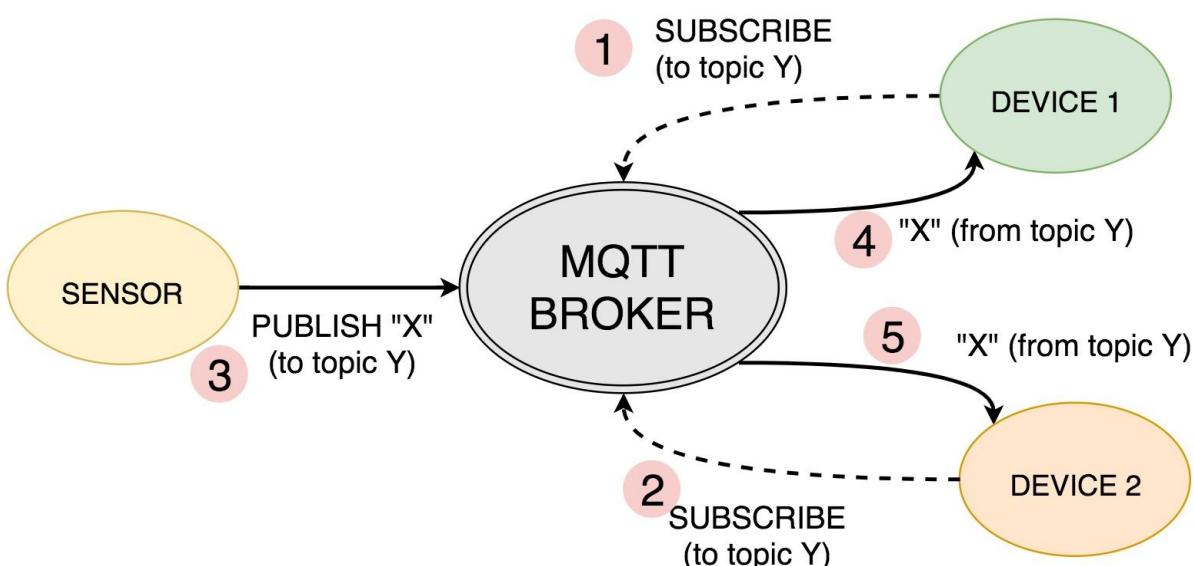


Fig-114. Estructura de un sistema MQTT.

El sistema MQTT funciona mediante el uso de un concentrador central de mensajes llamado broker de modo que los dispositivos que se conectan a dicho concentrador pueden hacer dos cosas: publicar o suscribirse a una cola de mensajes que se conoce como “topic”. De este modo, por ejemplo si un sensor se conecta a un broker “io.eclipse.org” en este caso y publica bajo el topic “iot/caso4/temperature” otro dispositivo que se conecte y se suscriba a dicha cola de mensajes recibirá la información que dicho sensor publique.

De este modo resumiendo, el uso de MQTT se considera un sistema distribuido donde los integrantes del sistema se comunican con el servidor principal para enviar “publicar” o escuchar “suscribirse” a unos canales “topics”.

En el caso de esta práctica el sensor junto a la ESP32N1 montada sobre la DEVKIT será considerado el “sensor”, y a continuación mostraremos como recibir dicha información desde NodeRED en el PC, que en este caso sería considerado el “device” según el diagrama anterior.

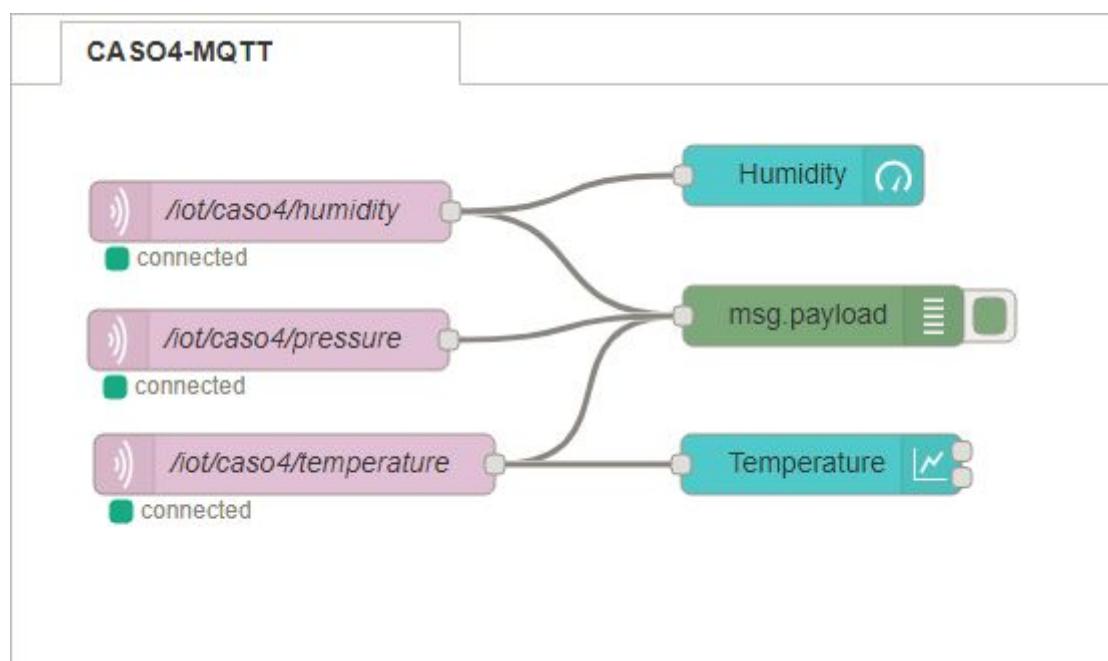


Fig-115. Programa en bloques en NodeRED.

Los bloques que aparecen en color lila son los bloques de MQTT que se emplean para conectar al servidor MQTT y a un topic.

Por otro lado, el bloque en verde es el que permite mostrar en la consola de depuración los mensajes provenientes de los bloques de MQTT.

Los bloques azules corresponden a los elementos del dashboard de NodeRED y son indicadores. En este caso un indicador de aguja sobre escala para la humedad “Humidity” y un indicador en formato de tabla con ejes de valor y tiempo para “Temperature”.

A continuación, se puede observar lo que aparece en la pestaña “debug” en el momento que usamos un bloque a tal efecto y lo enlazamos con otro elemento dentro del programa “flow”. En este caso podemos ver la información que viene desde los “topics” de MQTT, para confirmar que la configuración es correcta.

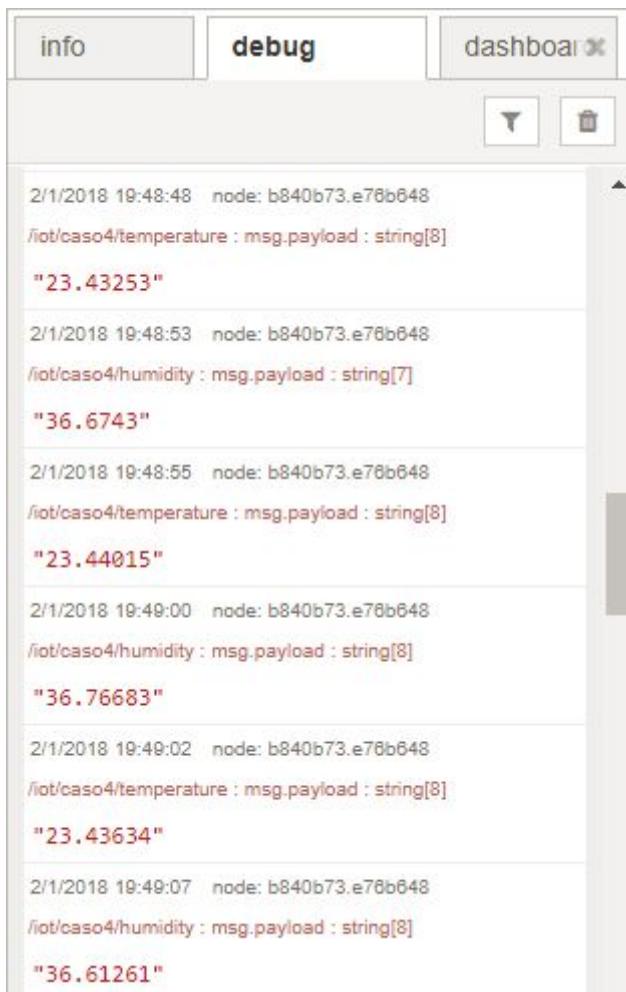


Fig-116. Pestaña de debug mostrando información desde MQTT

A continuación, vamos a ver como configurar los parámetros de conexión MQTT. Lo primero que deberemos realizar es la creación de un servidor (Broker) en la lista de Server, si este no existe aún. Una vez creado se podrá reutilizar para diferentes topics etc.

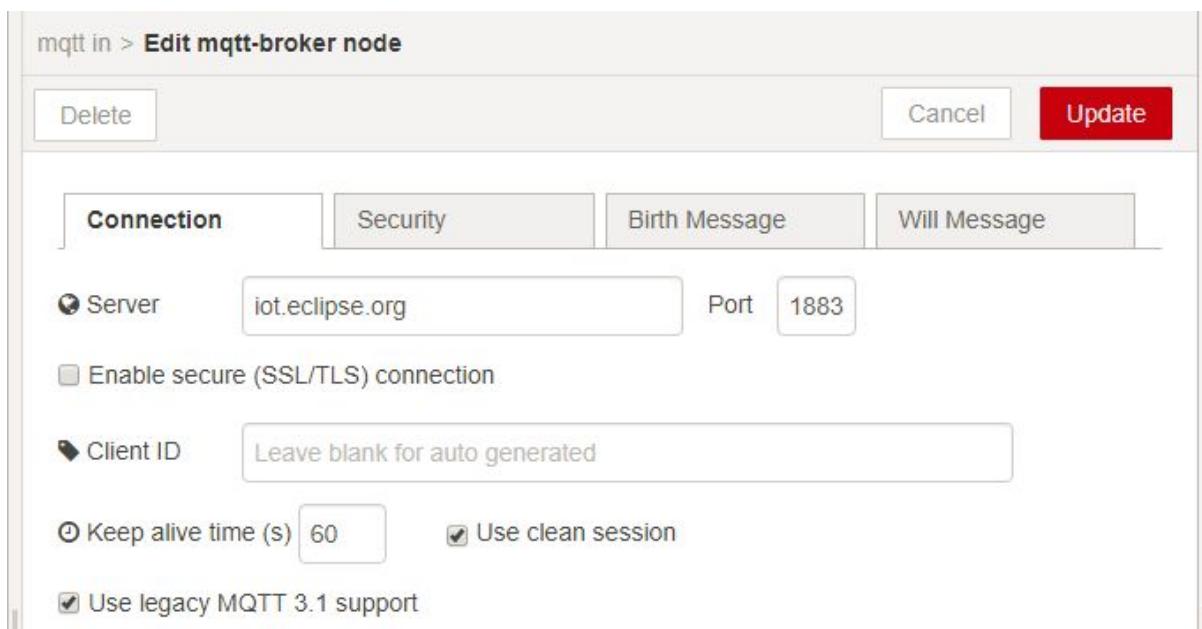


Fig-117. Creación de conexión a servidor broker MQTT.

Acto seguido una vez introducidos los parámetros del servidor como se indica en la figura, podremos proceder a crear las conexiones a los diferentes topics. En nuestro caso usaremos dos que serán humidity y temperature tal y como se enviaban desde el programa en bloques del whitecatIDE.

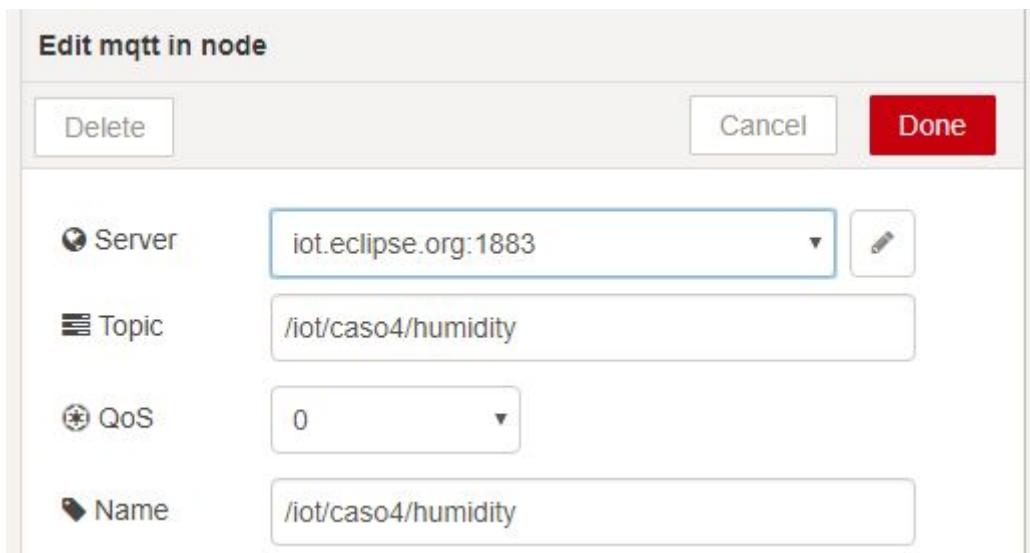


Fig-118. Ejemplo de suscripción al topic humidity, lo mismo seria para temperature, etc.

Una vez que hemos configurado la adquisición de los datos mediante los parámetros de MQTT procederemos a la representación de los mismos, empleando para ello los elementos que encontramos dentro del apartado dashboard de NodeRED. Estos elementos incluyen indicadores de agujas, gráficas, texto, etc.

El dashboard es un conjunto de pestañas y páginas html que generará NodeRED para agrupar los diferentes elementos que mostrarán la información. Las opciones que podemos especificar en el dashboard como se muestra en la siguiente figura, son:

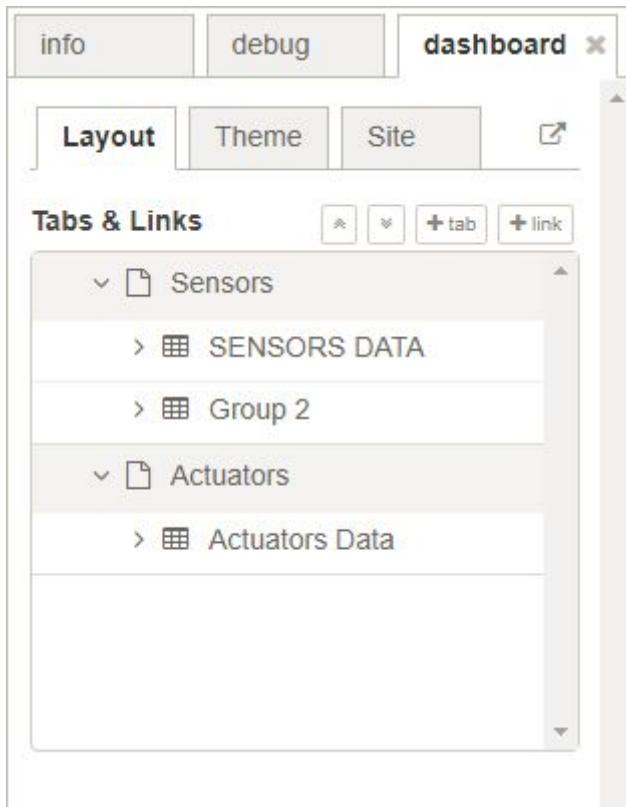


Fig-119. Opciones y configuración del dashboard.

|               |  |
|---------------|--|
| <b>Layout</b> | Especificaremos las secciones donde vamos a distribuir los diferentes elementos en la pantalla o pantallas |
| <b>Theme</b>  | Es un conjunto de temas predefinidos que cambiarán el aspecto de nuestro dashboard                         |
| <b>Site</b>   | Permite definir unas opciones genéricas del aspecto, título, formato, espacios entre elementos, etc        |

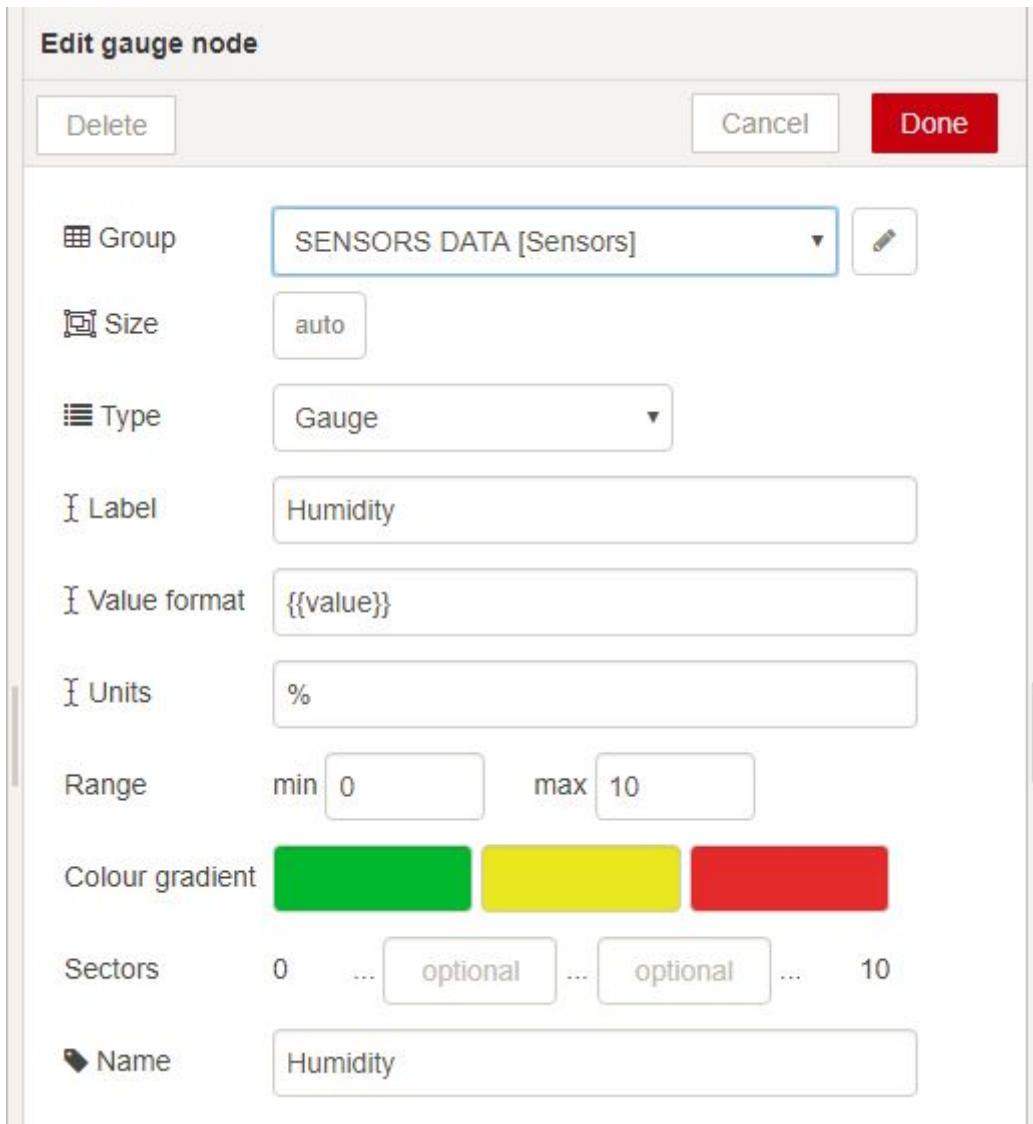


Fig-120. Opciones y configuración del indicador.

|              |   |
|--------------|---|
| Group        | Grupo al que pertenece el indicador dentro del layout                       |
| Size         | Tamaño en celdas del indicador  |
| Type         | Tipo de indicador en este caso dial con aguja                               |
| Label        | Etiqueta para mostrar en el indicador                                       |
| Value format | Valor y formato del indicador, desde aquí se especifica el origen del valor |
| Units        | Tipo de unidades del valor  |
| Range        | valores mínimos y máximos para mostrar                                      |

|                |                                 |
|----------------|---------------------------------|
| Color gradient | Escala de colores.              |
| Sectors        | Sectores de la escala opcional. |
| Name           | Nombre del indicador.           |

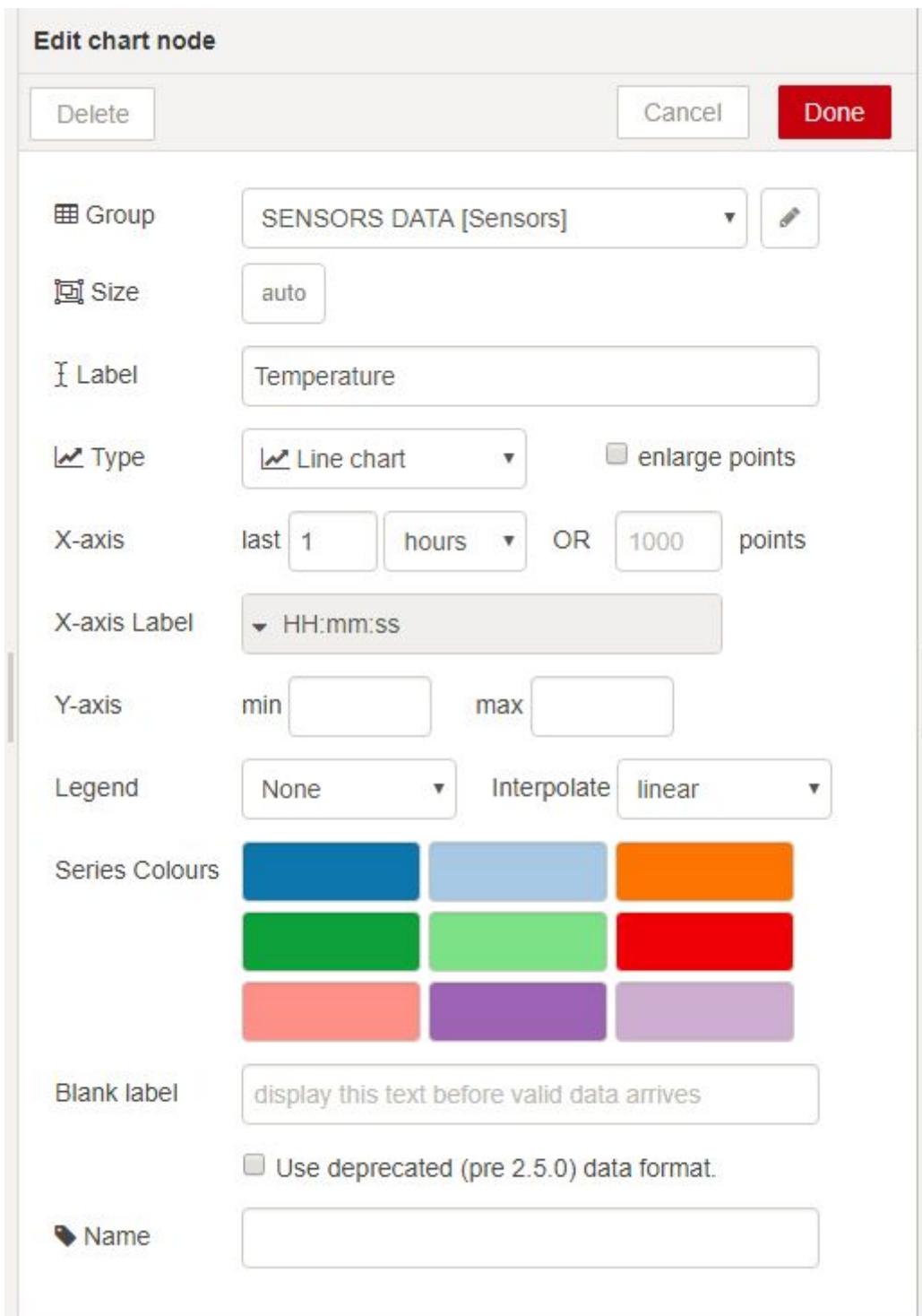


Fig-121. Opciones y configuración del indicador (tabla)

|                    |   |
|--------------------|---|
| <b>Group</b>       | Grupo al que pertenece el gráfico dentro del layout     |
| <b>Size</b>        | Tamaño del gráfico dentro del layout                    |
| <b>Label</b>       | Etiqueta de la tabla                                    |
| <b>Type</b>        | Tipo de tabla   |
| <b>X-axis</b>      | Eje X (tiempo)  |
| <b>Y-axis</b>      | Eje Y valores   |
| <b>Legend</b>      | Leyenda de la tabla                                     |
| <b>Interpolate</b> | Tipo de Interpolación, línea, curva, etc, entre valores |
| <b>Series</b>      | Colores de las series                                   |
| <b>Blank Label</b> | Texto para mostrar cuando aún no hay datos              |
| <b>Name</b>        | Nombre del gráfico                                      |

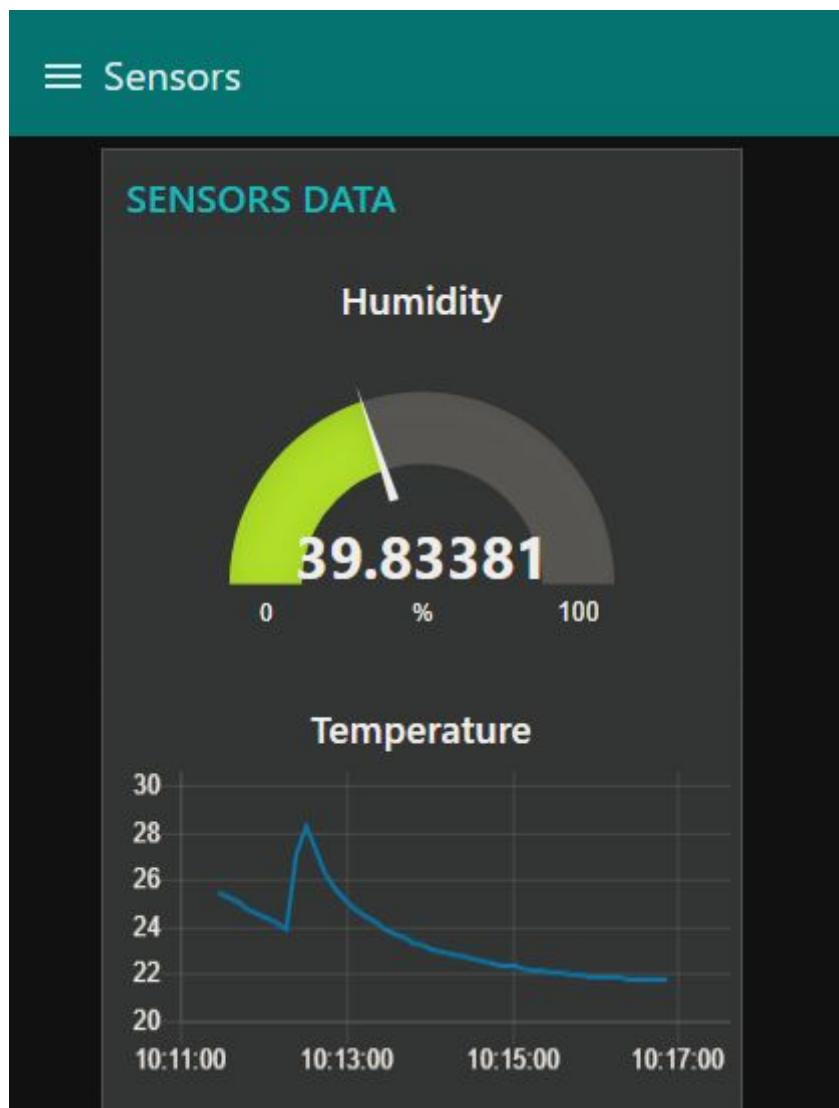


Fig-122. Resultado final del indicador y la tabla, en el dashboard.

Finalmente como se puede observar en la imagen mostraremos los valores recibidos desde el servidor (broker) MQTT en los diferentes topics con el formato de Indicadores y Charts que resultan más cómodos para la interpretación de los datos, en la siguiente práctica veremos como se envia la informacion mediante MQTT a la placa.

## 8.5. Caso 5 (Subscripción y recepción de información mediante MQTT)

### - Descripción

En este quinto caso utilizaremos la pantalla TFT que acompaña el kit para mostrar información que proviene desde Node RED mediante MQTT y WiFi. En este caso el dispositivo que escuchará los mensajes y se suscribirán a un topic sera el nodo ESP32N1 Devkit.

### - Objetivos

El objetivo de este quinto caso es que el usuario se familiarice con la configuración de la conectividad a través de una red WiFi, el uso y configuración del protocolo MQTT para recibir información y el uso de los disparadores de Node-RED para consultar información desde un servicio web y enviarla por MQTT a un nodo.

### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x PANTALLA TFT
- 1x SET DE CABLES
- 1x CONECTIVIDAD WIFI
- 1x BROKER MQTT (en este ejemplo usaremos [iot.eclipse.org](http://iot.eclipse.org))
- 1x PC CON CONEXION USB

### - Diagrama de conexión

El conexionado del display se muestra en el siguiente diagrama. El display utiliza el bus SPI para comunicarse con la placa, así como alimentación y unas líneas de control.

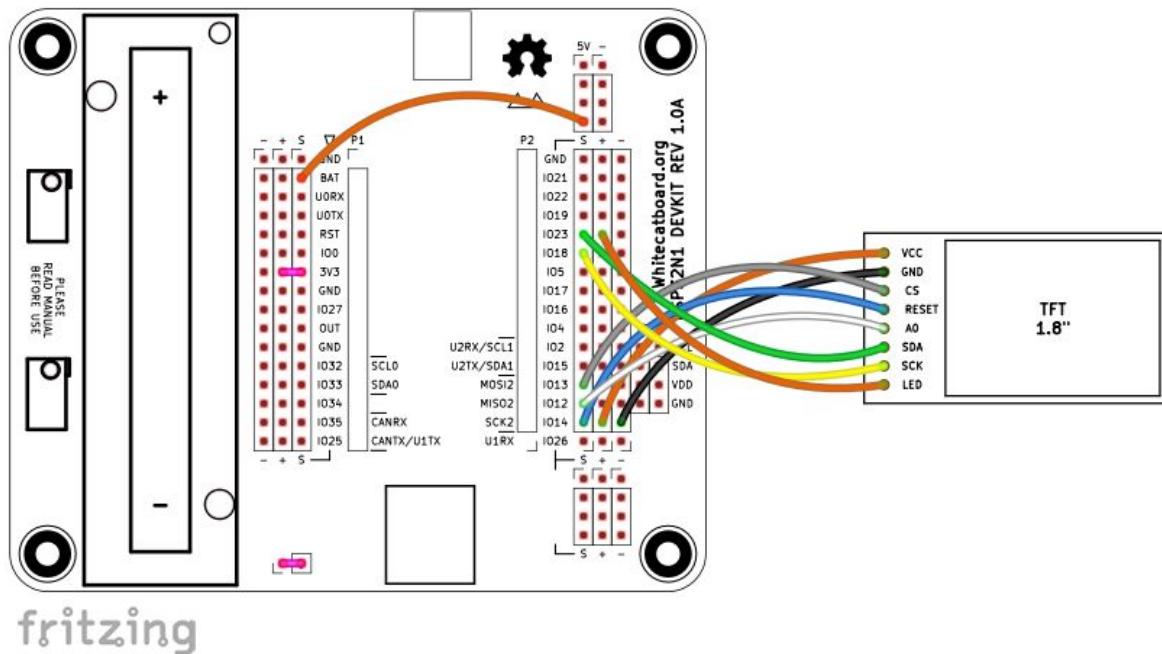


Fig-123. Conexionado de la pantalla TFT.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

## - Programa en Bloques

El programa en bloques incluye la conexión a la red WiFi como vimos en el caso anterior, así como la conexión al servidor MQTT (broker). Los bloques nuevos en este caso son los que hacen referencia al display, su inicialización, así como la suscripción al “topic” de MQTT correspondiente.



Fig-124. Programa en bloques.

El programa en bloques está dividido en tres apartados. En un primer apartado realizamos las acciones al inicializar la placa para iniciar la conexión WiFi así como las variables y la pantalla TFT.

Posteriormente nos suscribimos a 2 topics, temperatura y humedad, para que una vez que recibamos información desde Node-RED en alguno de dichos topics, mostremos dicha información en la pantalla.

Una vez enviado el programa a la placa ahora veremos como hacer llegar dicha información desde Node-RED. Para ello haremos un sencillo workflow ya que para esta práctica forzaremos los valores de los datos desde un botón y un control deslizante.

Como ya habíamos configurado el servidor en Node-RED para el caso anterior, simplemente tendremos que configurar los nodos de la siguiente manera:

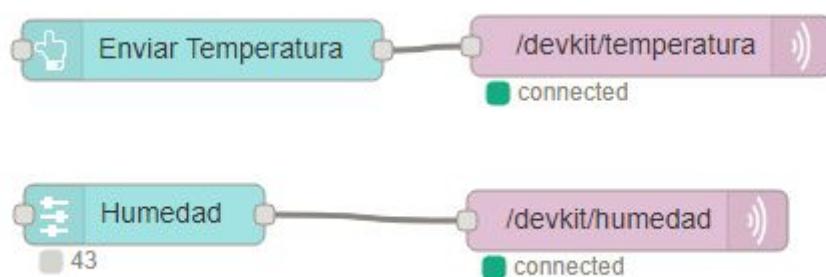


Fig-125. Workflow de Node-Red para el envío de los parámetros por MQTT.

Como se puede apreciar en la figura anterior, lo que hemos puesto son los bloques de los elementos del dashboard. En este caso un botón y un slider en color azul, ambos enlazados directamente con los bloques de publicación de MQTT en color lila. Es importante apreciar que la entrada de la conexión está a la izquierda.

De este modo, en el caso del botón cada vez que pulsemos se enviara un valor fijo a modo de ejemplo.

En el control deslizante se enviará el valor de la posición de la bola dentro de la escala (ver figura 130). A continuación, mostraremos los parametros de configuracion de cada uno de los bloques.

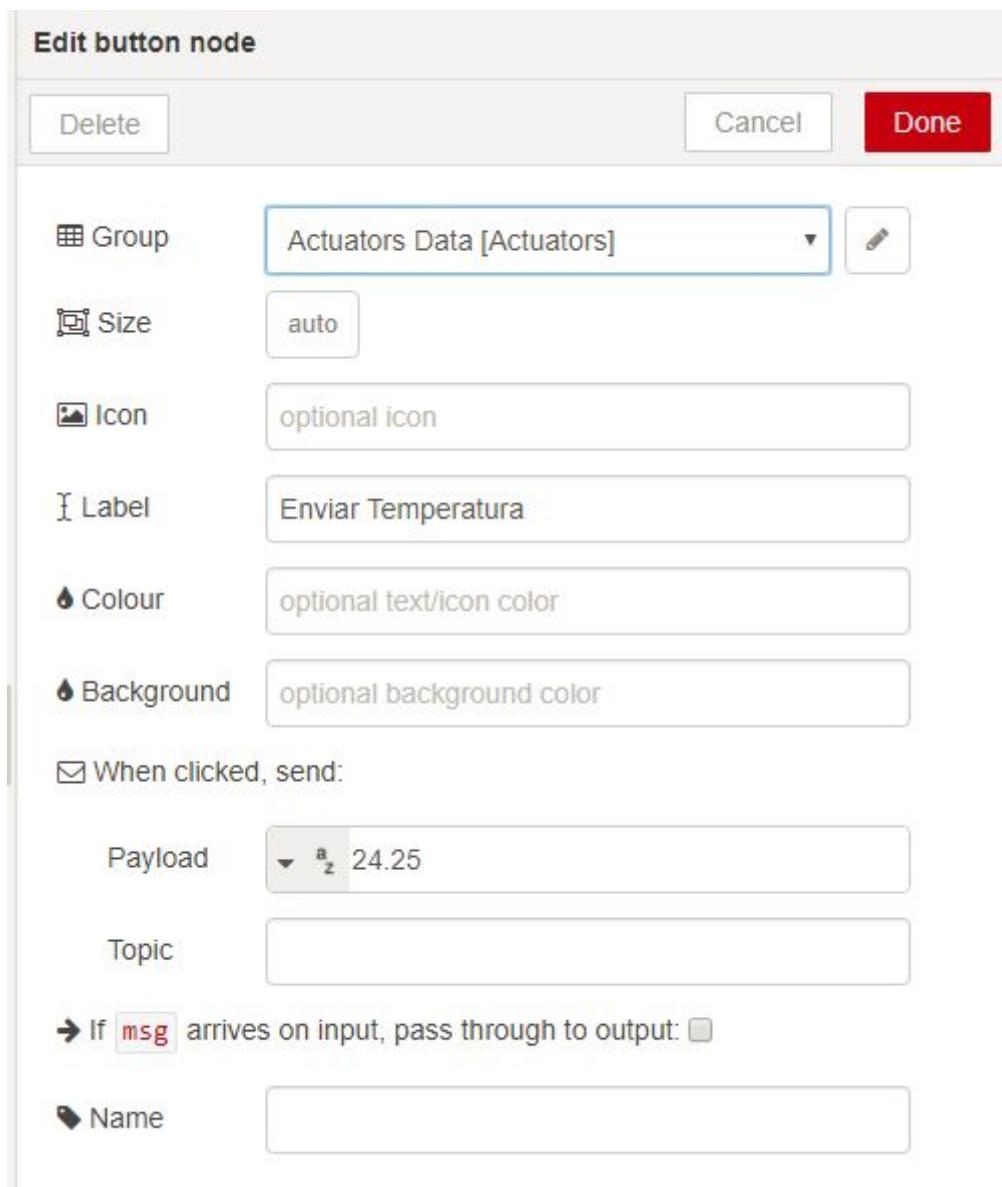


Fig-126. Configuracion del Boton.

|                   |   |
|-------------------|---|
| <b>Group</b>      | Grupo al que pertenece el botón dentro del layout |
| <b>Size</b>       | Tamaño del botón en cuadrados                     |
| <b>Icon</b>       | Icono del botón                                   |
| <b>Label</b>      | Etiqueta del botón                                |
| <b>Colour</b>     | Color del botón                                   |
| <b>Background</b> | Color del fondo del botón                         |

|                |   |
|----------------|---|
| <b>Payload</b> | Valor que enviara el botón al ser pulsado |
| <b>Topic</b>   | Topic que acompañara al payload           |
| <b>Name</b>    | Nombre del botón                          |

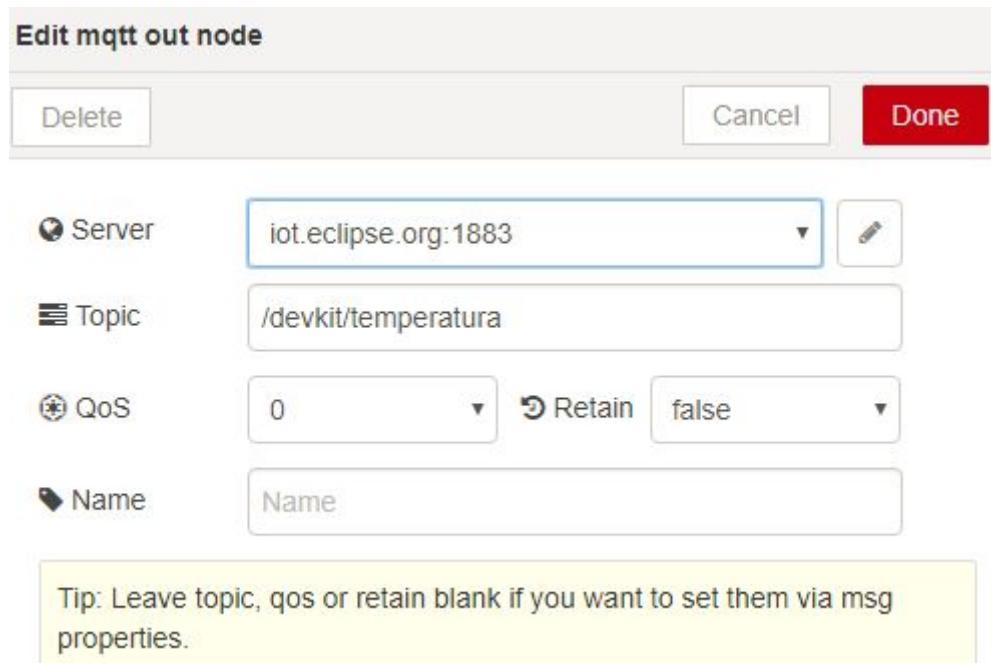


Fig-127. Configuración del bloque MQTT asociado al botón.

|               |  |
|---------------|--|
| <b>Server</b> | Servidor MQTT                                    |
| <b>Topic</b>  | Topic al que se enviarán los datos               |
| <b>QoS</b>    | Calidad de servicio, modo de asegurar la entrega |
| <b>Retain</b> | Retener en el servidor hasta ser entregado       |
| <b>Name</b>   | Nombre del bloque                                |

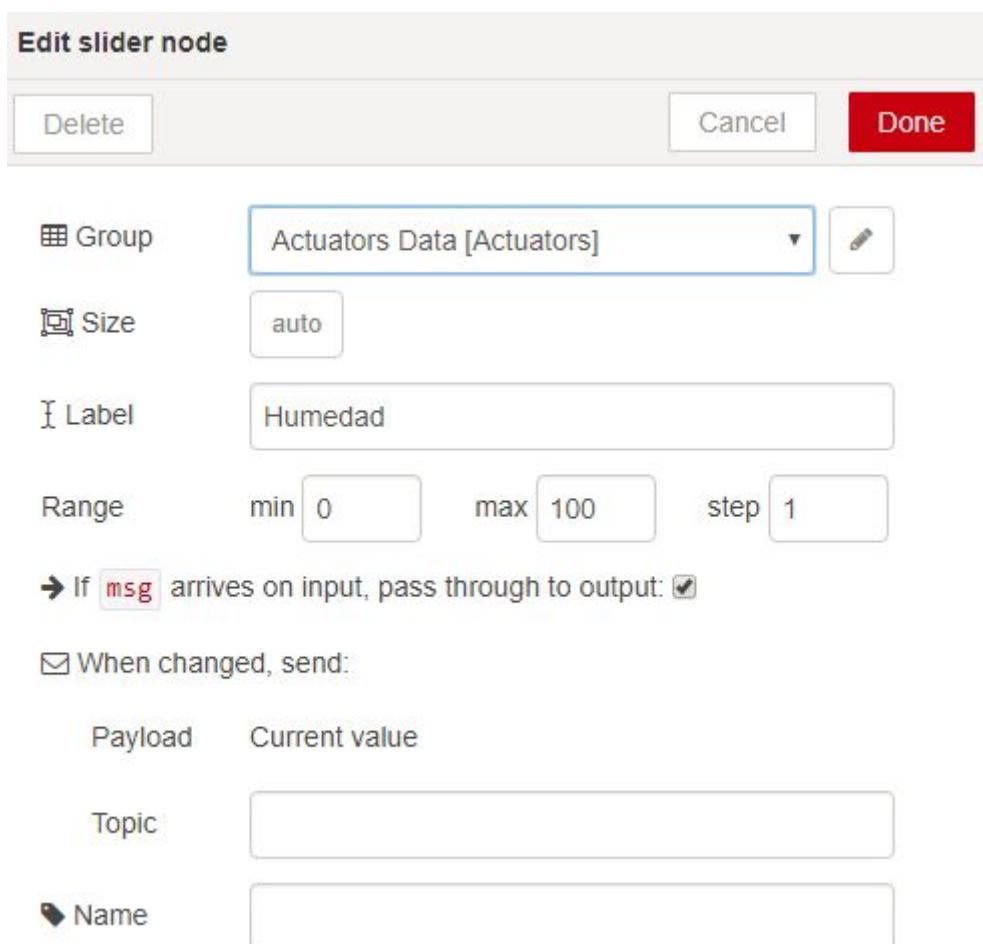


Fig-128. Configuración del bloque del slider.

|                |   |
|----------------|---|
| <b>Group</b>   | Grupo de layout al que pertenece el slider              |
| <b>Size</b>    | Tamaño del elemento en cuadros                          |
| <b>Label</b>   | Etiqueta para mostrar del slider                        |
| <b>Range</b>   | Rango mínimo , máximo y unidades de salto               |
| <b>Payload</b> | Payload por defecto el valor de la posición en el rango |
| <b>Topic</b>   | Topic del payload                                       |
| <b>Name</b>    | Nombre del bloque                                       |

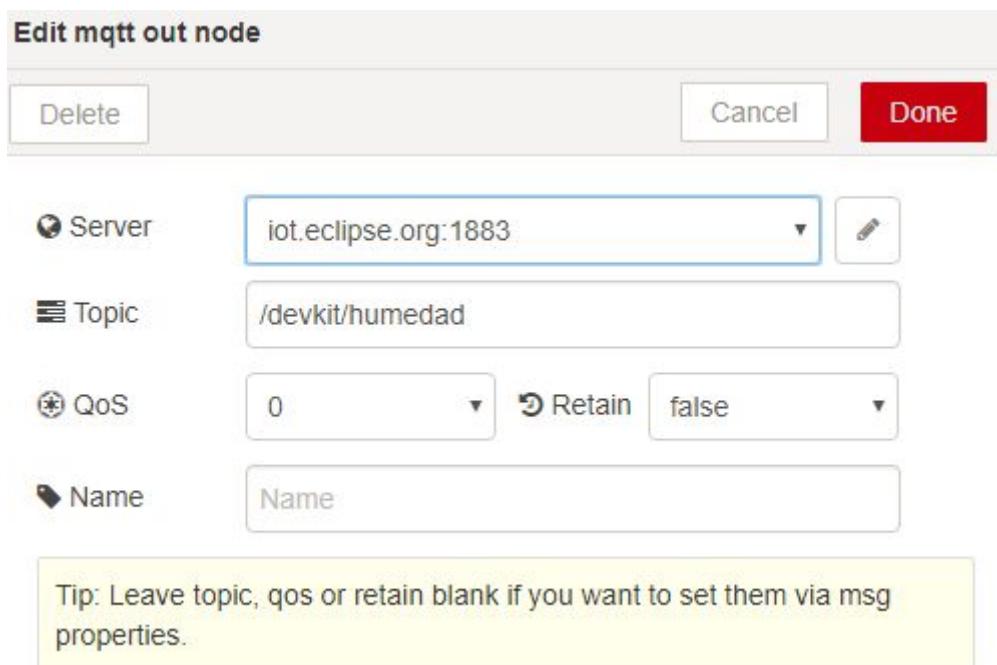


Fig-129. Configuración del bloque MQTT asociado al slider.

|               |  |
|---------------|--|
| <b>Server</b> | Servidor MQTT                                    |
| <b>Topic</b>  | Topic al que se enviarán los datos               |
| <b>QoS</b>    | Calidad de servicio, modo de asegurar la entrega |
| <b>Retain</b> | Retener en el servidor hasta ser entregado       |
| <b>Name</b>   | Nombre del bloque                                |

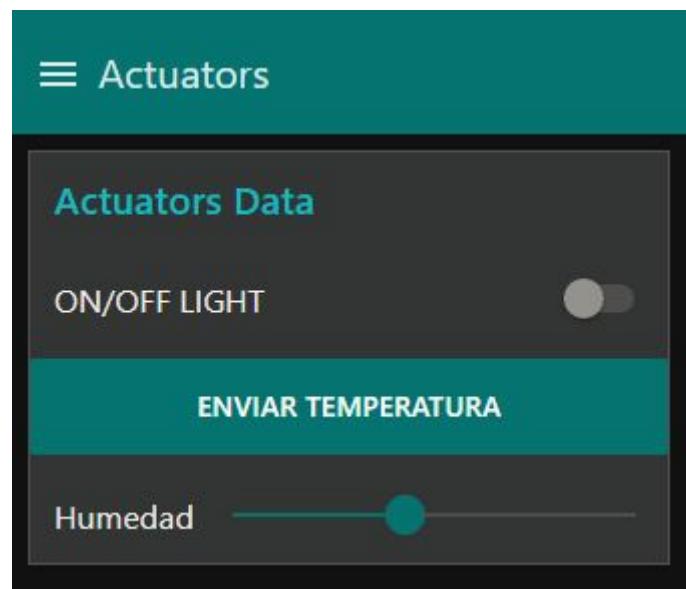


Fig-130. Resultado del dashboard con el botón de enviar temperatura y el slider de humedad debajo.



Fig-131. Imagen de la pantalla TFT al arranque.



Fig-132. Imagen de la pantalla TFT al recibir información desde el dashboard via MQTT.

## 8.6. Caso 6 (Lógica en el Nodo “Edge Computing”)

### - Descripción

En este sexto caso vamos a configurar el sensor BME280 conjuntamente con el actuador (ventilador) de modo autónomo para que sea la placa quien proceda a la puesta en marcha o parada del ventilador en función de los datos recogidos por el sensor.

### - Objetivos

Programar un ciclo de puesta en marcha y parada de un actuador, en este caso un ventilador, de manera autónoma con los datos pre-programados en la placa y los valores recibidos por el sensor. Esto se denomina edge computing y se aplica cuando los propios nodos toman las decisiones pre-programadas sin intervenir para ello el servidor.

### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BME280
- 1x VENTILADOR
- 1x SET DE CABLES
- 1x CONECTIVIDAD WIFI
- 1x BROKER MQTT (en este ejemplo usaremos [iot.eclipse.org](http://iot.eclipse.org))
- 1x PC CON CONEXION USB

### - Diagrama de conexión

El conexionado del sensor BME280 y el actuador ventilador se muestra a continuación. Para el ventilador usaremos la salida out del segundo regulador.

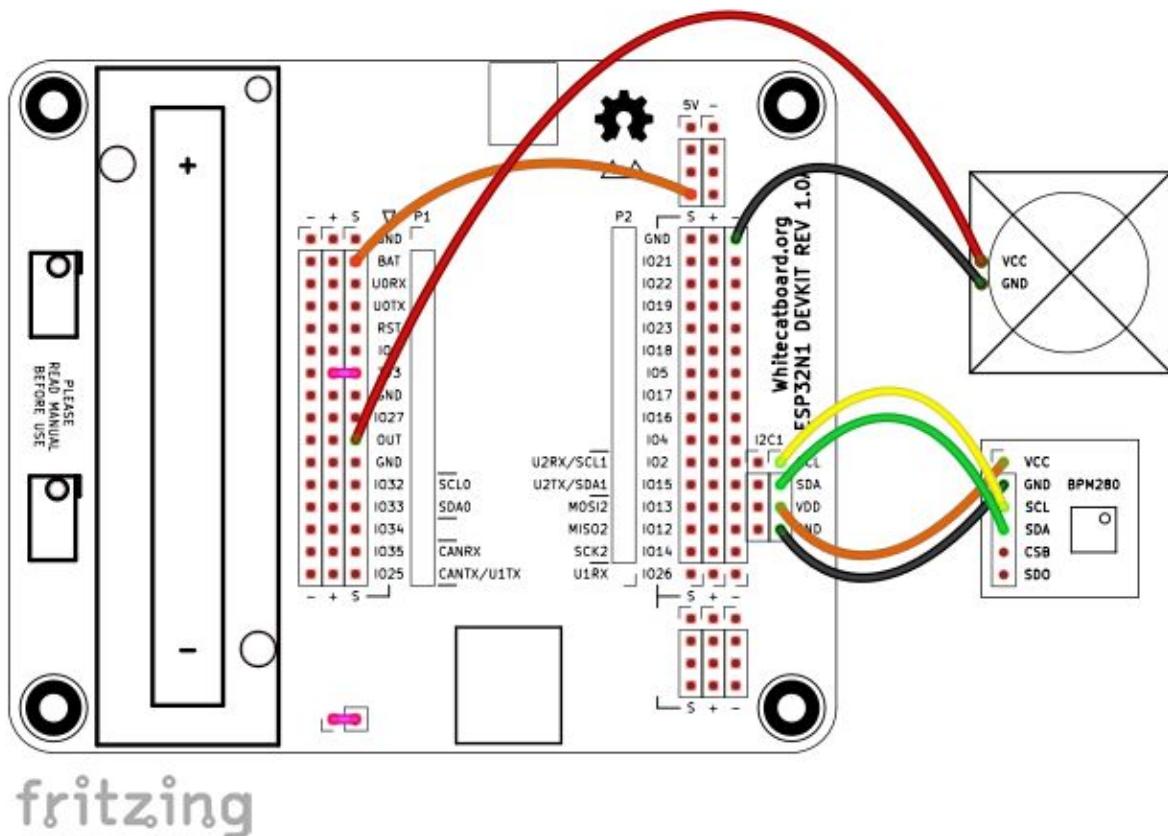


Fig-133. Conexionado del sensor y el actuador.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

## - Programa en Bloques

El programa en bloques está dividido en dos partes. La primera se produce al iniciar la placa, y es cuando se configura el sensor, se establece el pin del segundo regulador a nivel bajo y se establece la variable de temperatura a 0.

La segunda parte corresponde a un bucle infinito y es la que se encarga de la lógica de encendido y apagado del ventilador mediante el control del pin IO27 que activa el segundo regulador.

En esta práctica vemos que el valor 30 marcado en el bloque condicional está grabado en el propio programa de la placa. Por este motivo el sistema será “autónomo” y no precisará de información o comunicación con el mundo exterior.

Este tipo de sistemas se podría considerar IoT si generara o recibiera información sobre el estado.

En el próximo caso, veremos lo mismo pero cambiaremos el parámetro de control del ventilador e informaremos al servidor mediante comunicación MQTT, como vimos en casos anteriores.

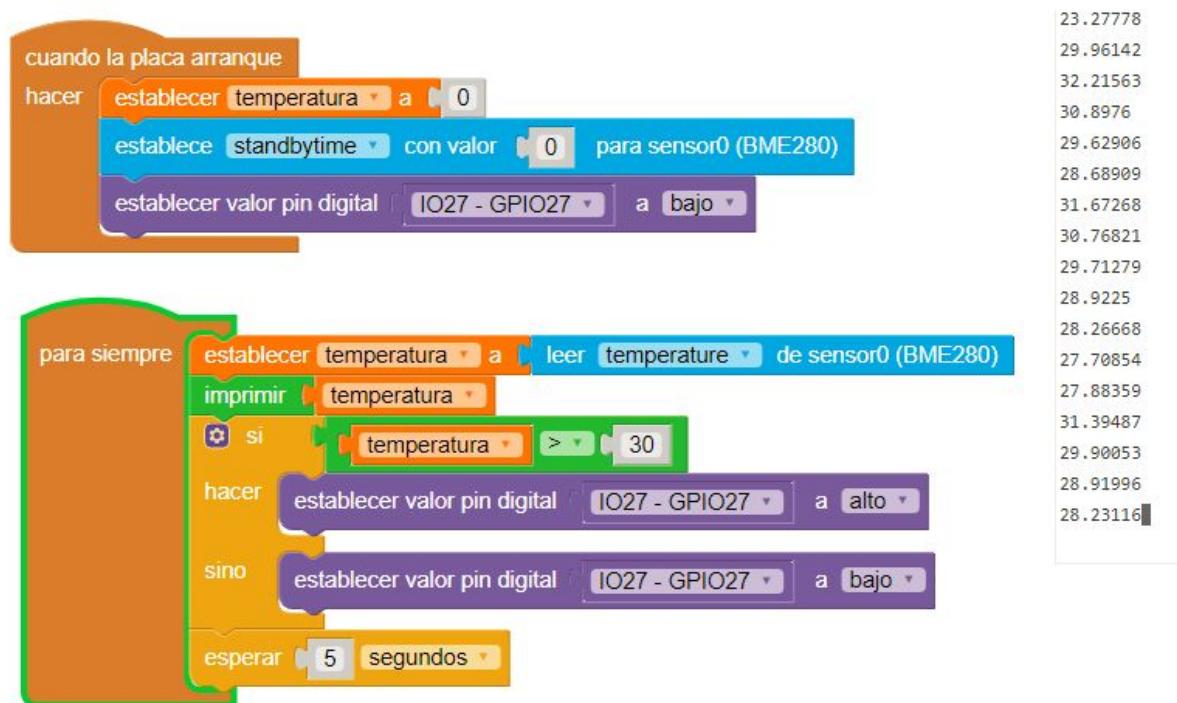


Fig-134. Programa en bloques.

## 8.7. Caso 7 (Lógica en la Nube “Cloud Computing”)

### - Descripción

Como en el caso anterior colocaremos un sensor de temperatura BME280 y un actuador (un ventilador). En este caso el control del parámetro de encendido y apagado del ventilador no estará “programado” en los propios bloques, si no que vendrá dado desde el NodeRED. De este modo en base a la información que circula hacia un sentido y otro se toman decisiones de actuación desde el servidor (“la nube”).

### - Objetivos

El objetivo de este caso, es que el usuario comprenda la opción de programar decisiones en la nube. De este modo, es factible realizar un programa más flexible en el dispositivo permitiendo así que el control resida en la nube siendo más accesible a la hora de introducir modificaciones. A nivel de desarrollo en el ámbito de IoT, es uno de los sistemas preferidos ya que la parte de código del nodo se reduce, dejando la complejidad y la gestión de errores etc, a un servidor que permite un modo más cómodo de trabajar.

### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BME280
- 1x VENTILADOR
- 1x SET DE CABLES
- 1x CONECTIVIDAD WIFI
- 1x BROKER MQTT (en este ejemplo usaremos iot.eclipse.org)
- 1x PC CON CONEXION USB

### - Diagrama de conexión

A continuación, se muestra el conexionado del sensor BME280 y del actuador (ventilador). Para este dispositivo, usaremos la salida out del segundo regulador. Como se puede comprobar, esta disposición es idéntica al caso anterior por lo que podemos decir que el cambio de un sistema Edge a uno Cloud, no implica normalmente variaciones en el hardware.

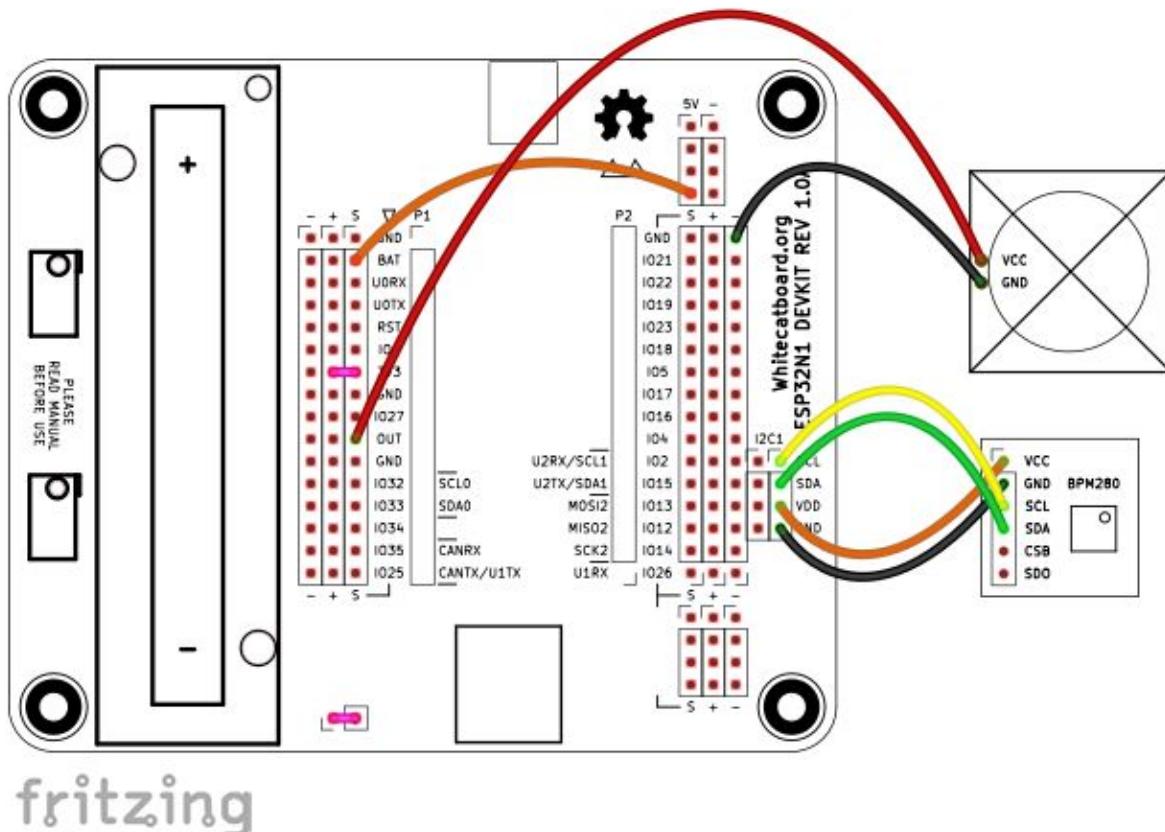


Fig-135. Conexionado del sensor y el actuador.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

## - Programa en Bloques

Del mismo modo que en el caso anterior, el programa se divide en un conjunto de inicio para inicializar las variables, sensor, conexión WiFi, etc. Un segundo bloque que estará funcionando en un bucle infinito pero en este caso además enviará información del sensor via MQTT al broker lo que nos permitirá tomar decisiones en función de dicha información.

Y un tercer bloque, que como vimos en casos anteriores, se suscribió a una cola MQTT para recibir la información de un valor que será el umbral, en este caso el de activación del ventilador.

Así, ahora el control de ese parámetro vendrá dado por un servidor y no estará programado en el nodo.

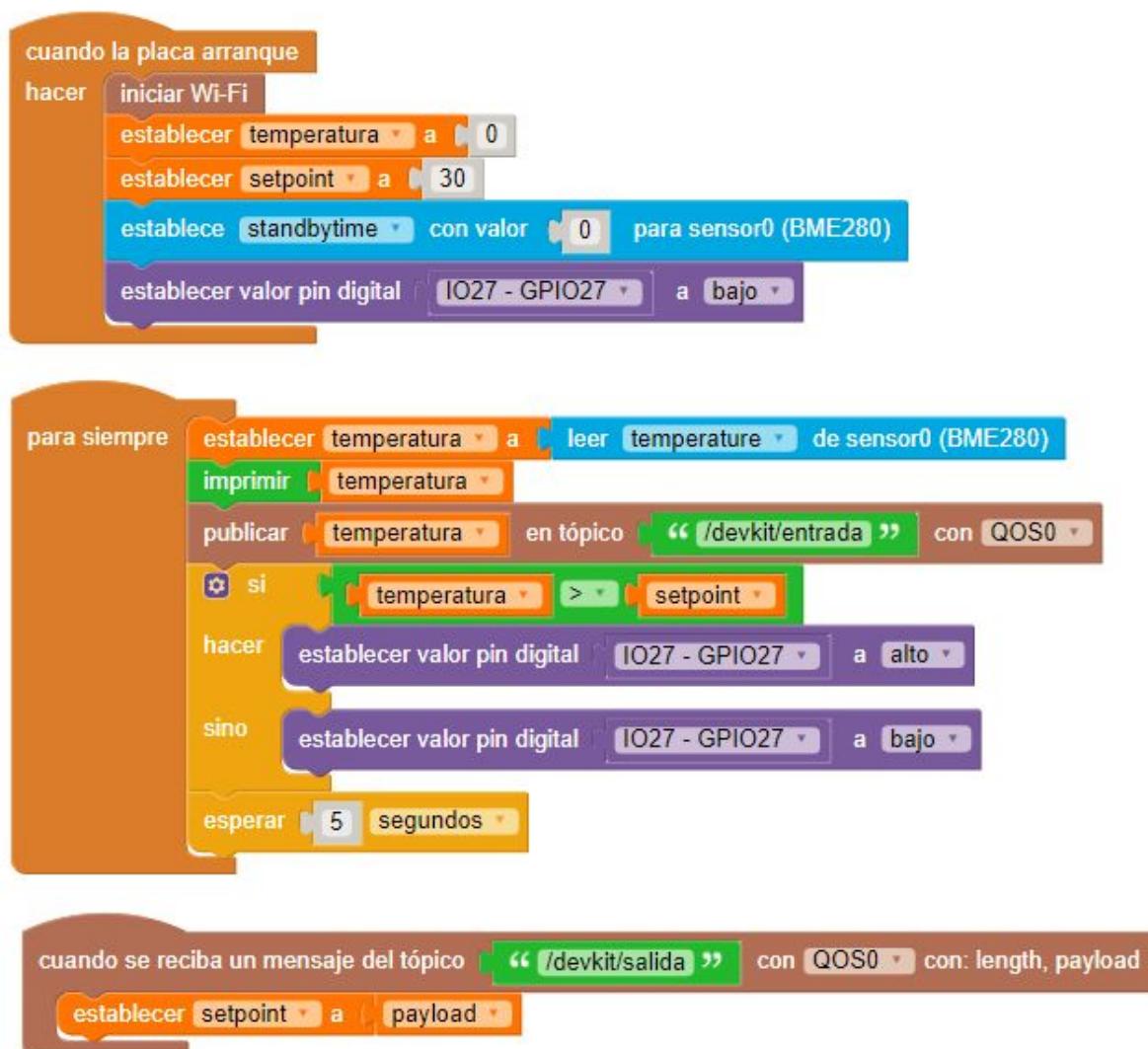


Fig-136. Programa de bloques.

A continuación, veremos la parte de Node-Red que mediante un pequeño código javascript actuará en función de los datos recibidos. En este caso usamos un setpoint pero la opción más sencilla sería enviar un estado de on/off.

Tenemos 3 bloques. Uno MQTT de suscripción para la entrada de información desde el nodo. Otro segundo bloque de lógica que es simplemente una función Javascript que veremos en la siguiente pantalla. Y un último bloque de salida que es una publicación en una cola MQTT, de un valor calculado en la función Lógica.



Fig-137. Workflow de Node-RED

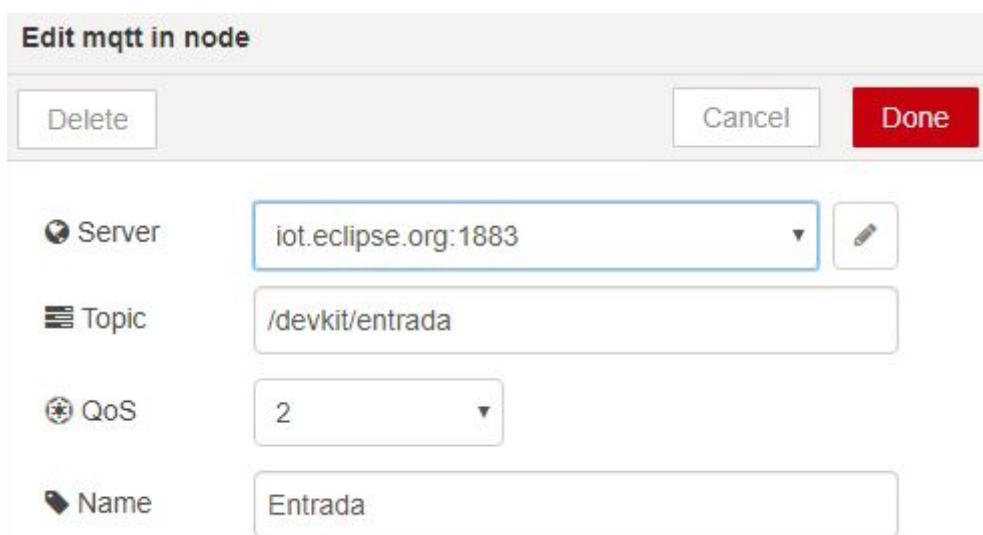


Fig-138. Bloque MQTT de entrada.

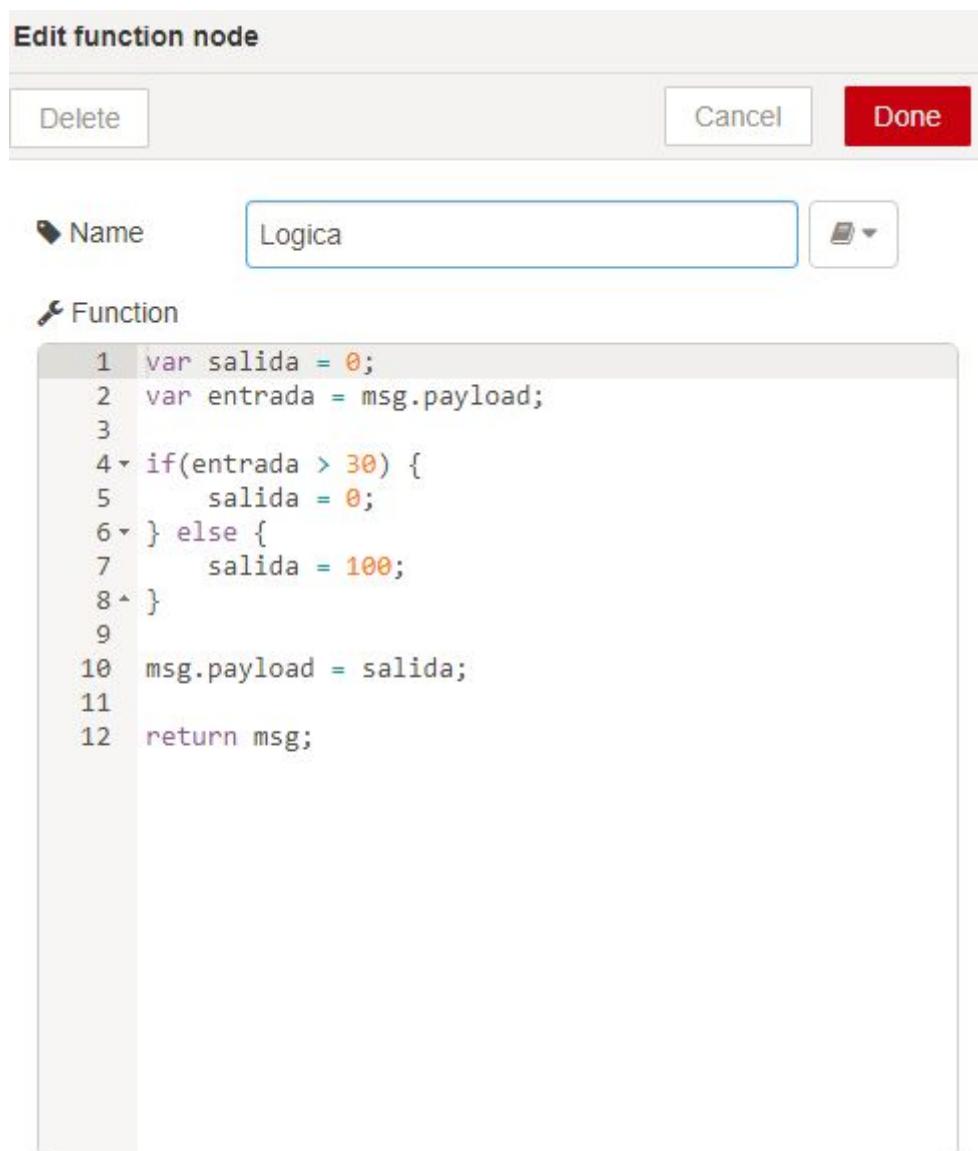


Fig-139. Código de la función en Javascript.

En el código en javascript lo que haremos es leer el payload de entrada desde la cola MQTT y en función de su valor (entrada) estableceremos el valor de una salida. Posteriormente, esta función conecta a una cola MQTT de salida para publicar los datos. En concreto, la variable salida que hemos establecido o bien a 0 o bien a 100, en función de la entrada.

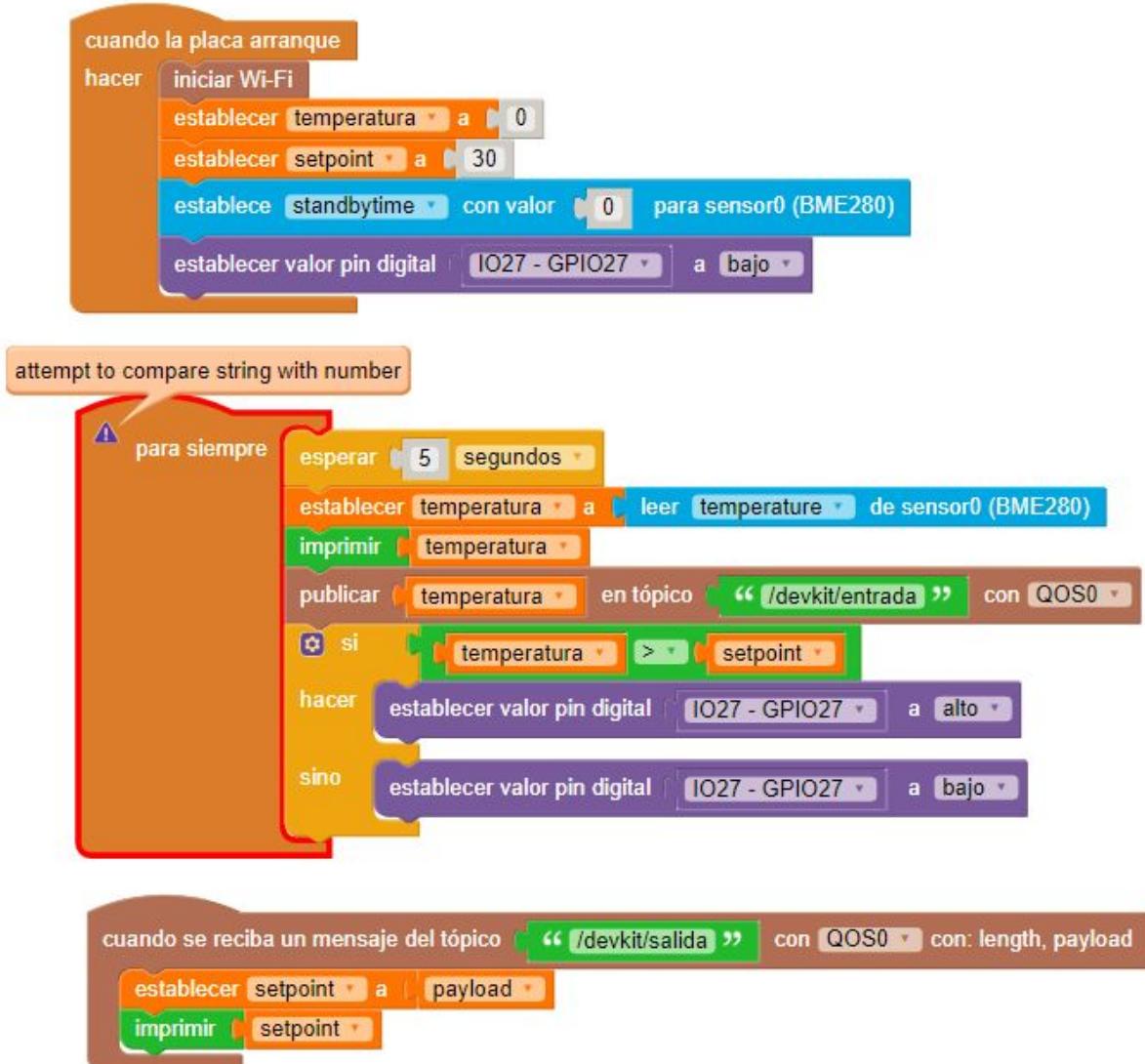


Fig-140. Programa en bloques , ejecutandose si se produce un error será informado en el bloque en cuestión como se puede apreciar.

## 8.8. Caso 8 (Despliegue Nodo Tipo)

### - Descripción

En este último caso utilizaremos la batería y el sensor BME280 que acompañan el kit para enviar información a TTN mediante LoRaWAN y luego recogerla desde Node-RED via MQTT.

### - Objetivos

El objetivo de este último caso de uso es agrupar diferentes partes que hemos visto hasta el momento para conseguir emular lo que realmente sería un nodo IoT de sensorica, aplicando para ello el uso de baterías, bajo consumo eléctrico, bajo coste, etc.

### - Materiales

Para la realización de este caso práctico necesitaremos:

- 1x ESP32N1
- 1x ESP32N1 DEVKIT
- 1x SENSOR BME280
- 1x BATERÍA 18650
- 1x SET DE CABLES
- 1x CONECTIVIDAD WIFI
- 1x BROKER MQTT (en este ejemplo usaremos iot.eclipse.org)
- 1x PC CON CONEXION USB

### - Diagrama de conexión

A continuación, se muestra el conexionado del sensor BME280 idéntico al de las prácticas anteriores. En este caso, el cable de la batería se coloca en la entrada del sensor analógico AIN0 para medir su estado de carga ya que de este modo la batería pasará a ser un sensor más, que nos informará cuando sea preciso reemplazar o recargar.

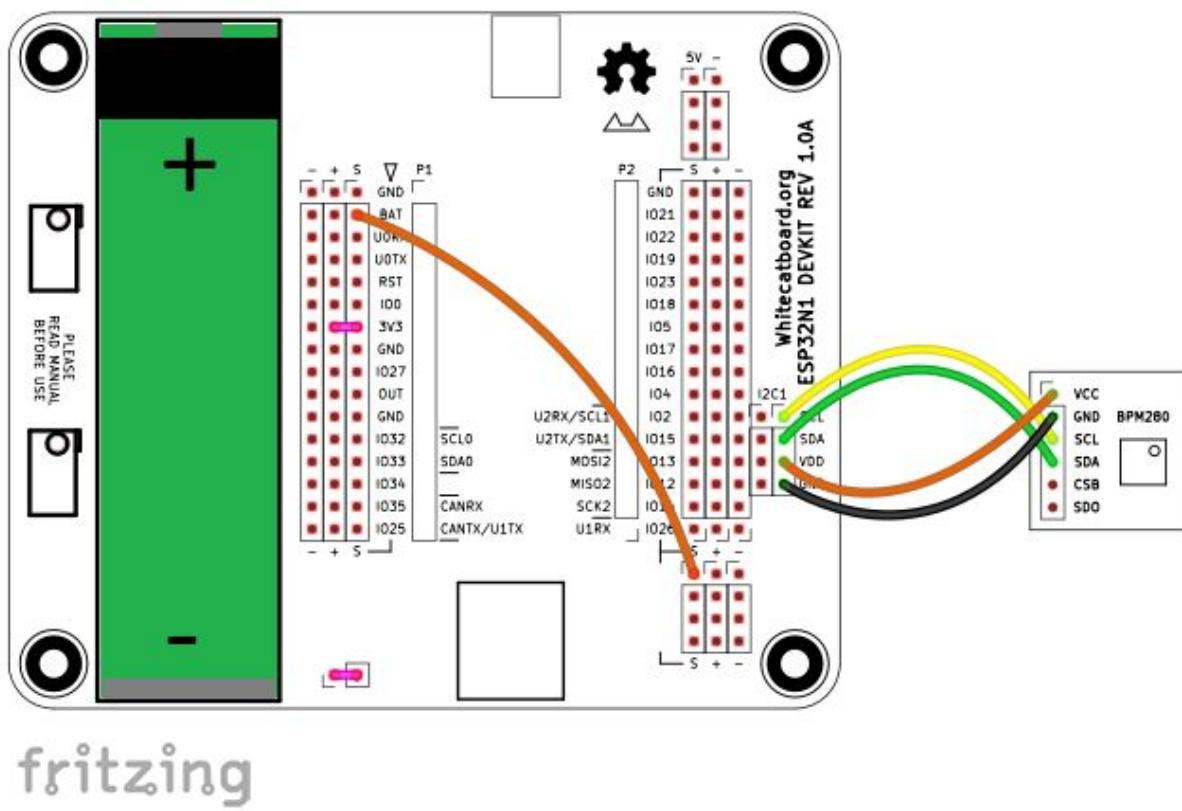


Fig-141. Diagrama de conexión con la batería en su zócalo y el sensor BME280.



**SE RECOMIENDA DESCONECTAR CUALQUIER FUENTE DE ENERGÍA DE LA PLACA MIENTRAS SE REALIZA EL CONEXIONADO DE LOS ELEMENTOS DE LA PRÁCTICA.**

**ATENCION !! AL EMPLEAR LA BATERÍA SE DEBERÁ QUITAR EL PUENTE ENTRE 5V Y BAT, Y PRESTAR ATENCIÓN A LA CARGA Y TEMPERATURA DE LA MISMA.**

**RECORDAR QUE EL USO DE BATERÍAS REQUIERE DE ATENCIÓN POR PARTE DEL USUARIO EN LAS FASES DE CARGA, ETC.**

### - Programa en Bloques

Antes de mostrar el programa en bloques completo, mostraremos dos bloques para comprobar como curiosidad, que mientras estemos conectados por USB, el valor de la batería aumentará indicando que esta se está cargando.



Fig-142. Muestra de carga mientras esta conectado via USB.

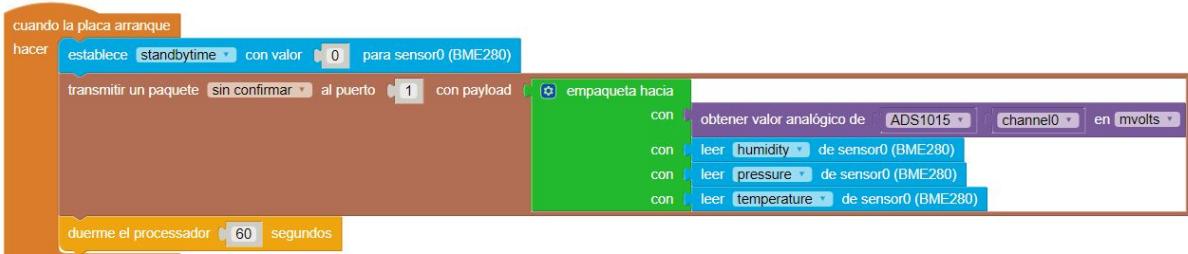


Fig-143. Programa en bloques.

En la imagen anterior podemos comprobar lo sencillo que resulta el programa en bloques. Únicamente tendremos un solo bloque que se ejecutará cada vez que la placa arranque, lo que sucederá después de haber “dormido” durante un periodo de 60 segundos.

El término de “dormir” la placa no lo habíamos visto hasta este momento. Lo que este método hace es desconectar todos los periféricos y reducir el consumo del sistema hasta los 14uA (micro amperios) con el objetivo de alargar la vida de la carga de la batería.

Por tal motivo uno de los puntos importantes es calcular cuánto tiempo vamos a estar leyendo y transmitiendo los datos y cuanto tiempo vamos a tener la placa “durmiendo”. Mediante esos parámetros y mediante unas fórmulas específicas (que contemplan además un porcentaje de error sobre pérdidas que suceden por temperatura, degradación de la propia batería, etc), es posible estimar el tiempo de duración de las baterías.

Como se puede apreciar en la imagen Fig-143, uno de los objetivos que cumple el entorno de bloques es reducir la complejidad de un programa de estas características a simplemente unos pocos bloques.

A continuación, comprobamos en la consola de TTN como los datos que estamos encapsulando, nos llegan al servidor tal y como muestra la siguiente imagen:

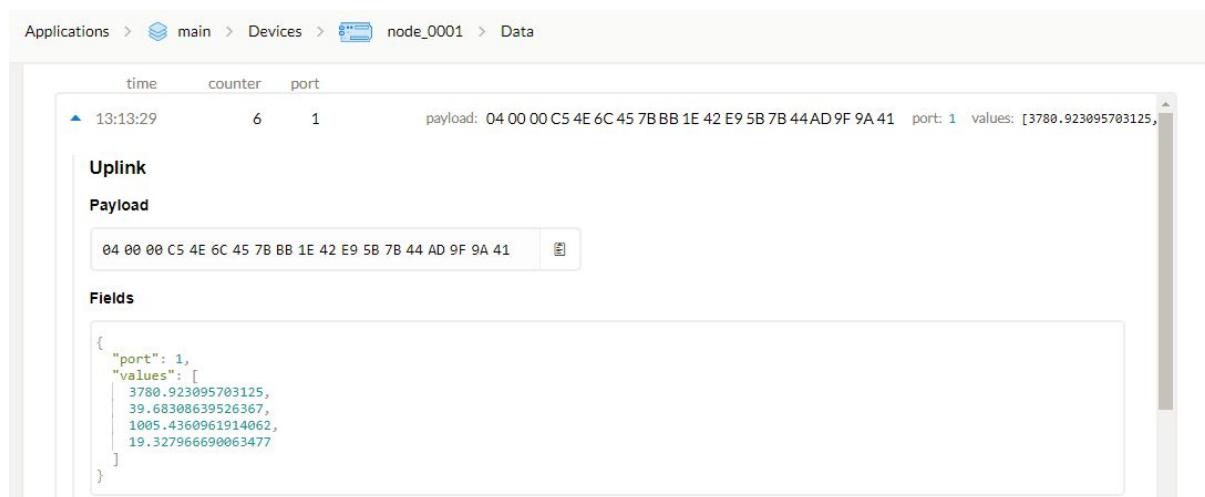


Fig-144. Datos llegando a TTN, en este caso como node\_0001.

Y por último ya solo nos queda obtener dichos datos por MQTT desde TTN hacia Node-RED y mostrarlos en un dashboard como habíamos visto anteriormente. En este caso como muestra la siguiente imagen, primero comprobaremos que los datos están llegando desde el MQTT. Una vez comprobado, añadiremos un bloque para separar la información que nos llega y enviarla a los diferentes indicadores.

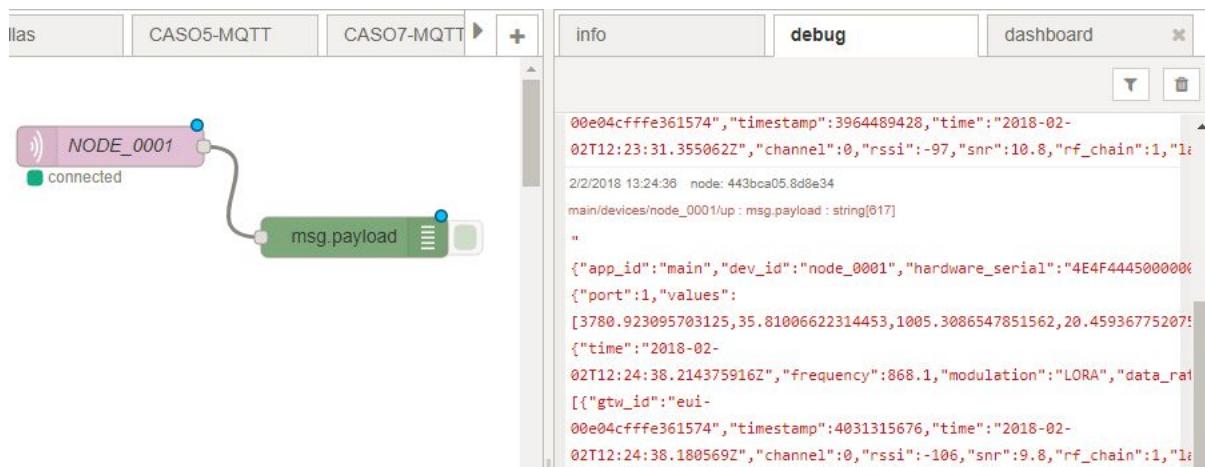


Fig-145. Datos llegando desde TTN via MQTT.

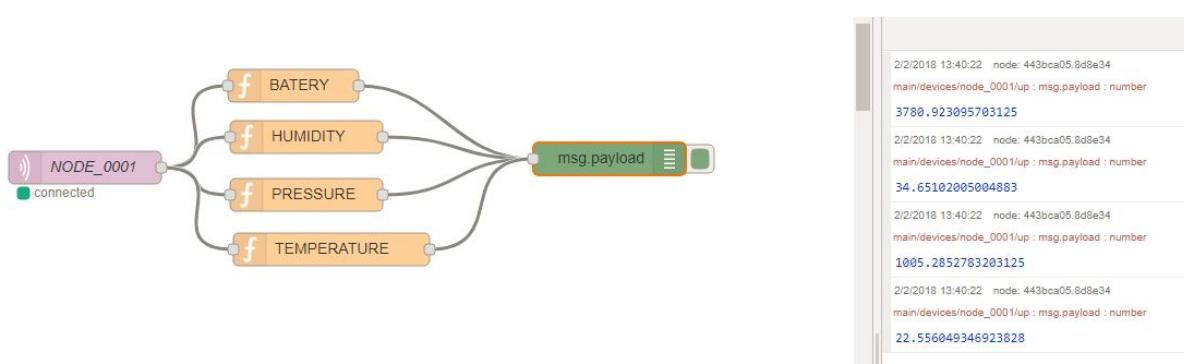


Fig-146. Datos llegando desde TTN via MQTT y separados por funciones javascript.

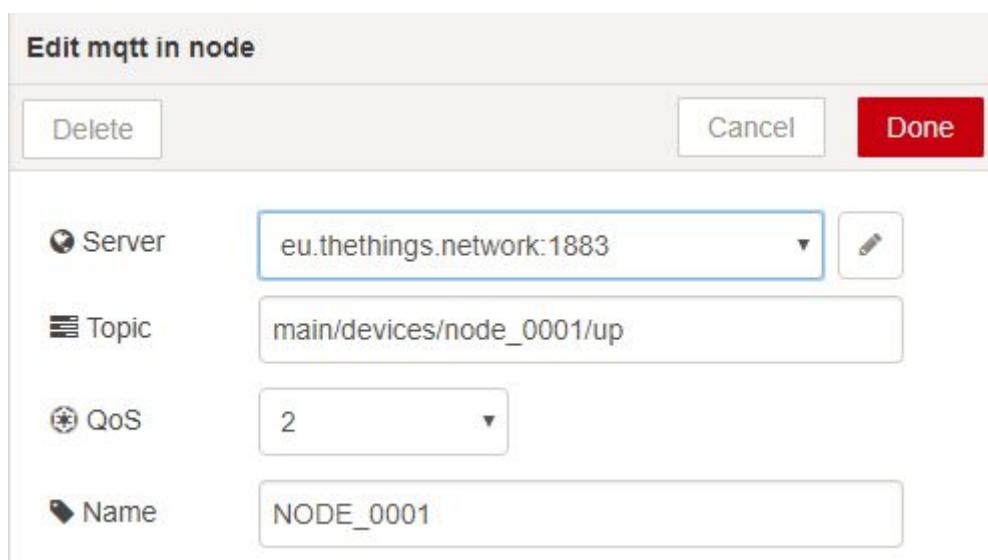


Fig-147. Nodo MQTT.

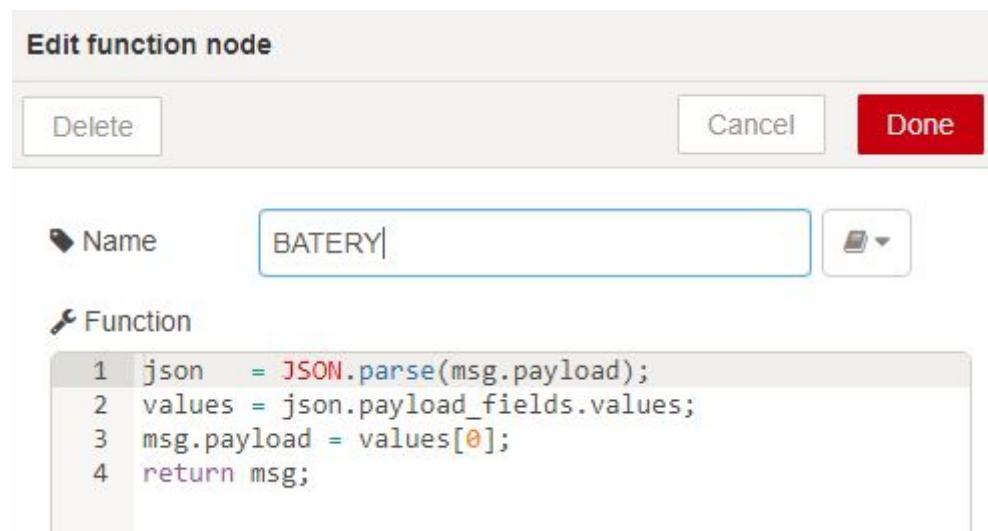


Fig-148. Función javascript que procesa el JSON y selecciona el valor en este caso de la bateria que es el primer valor del array.

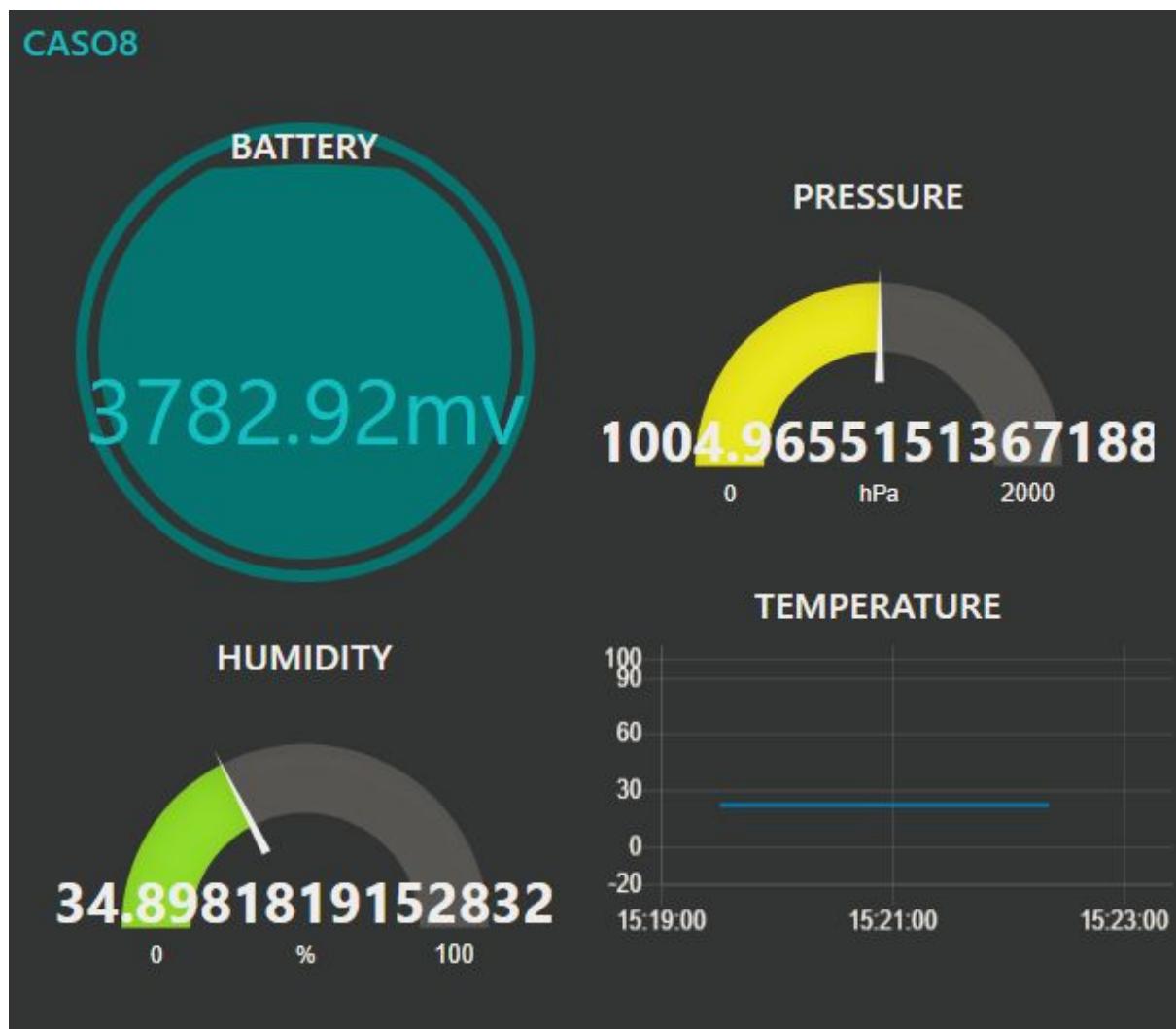


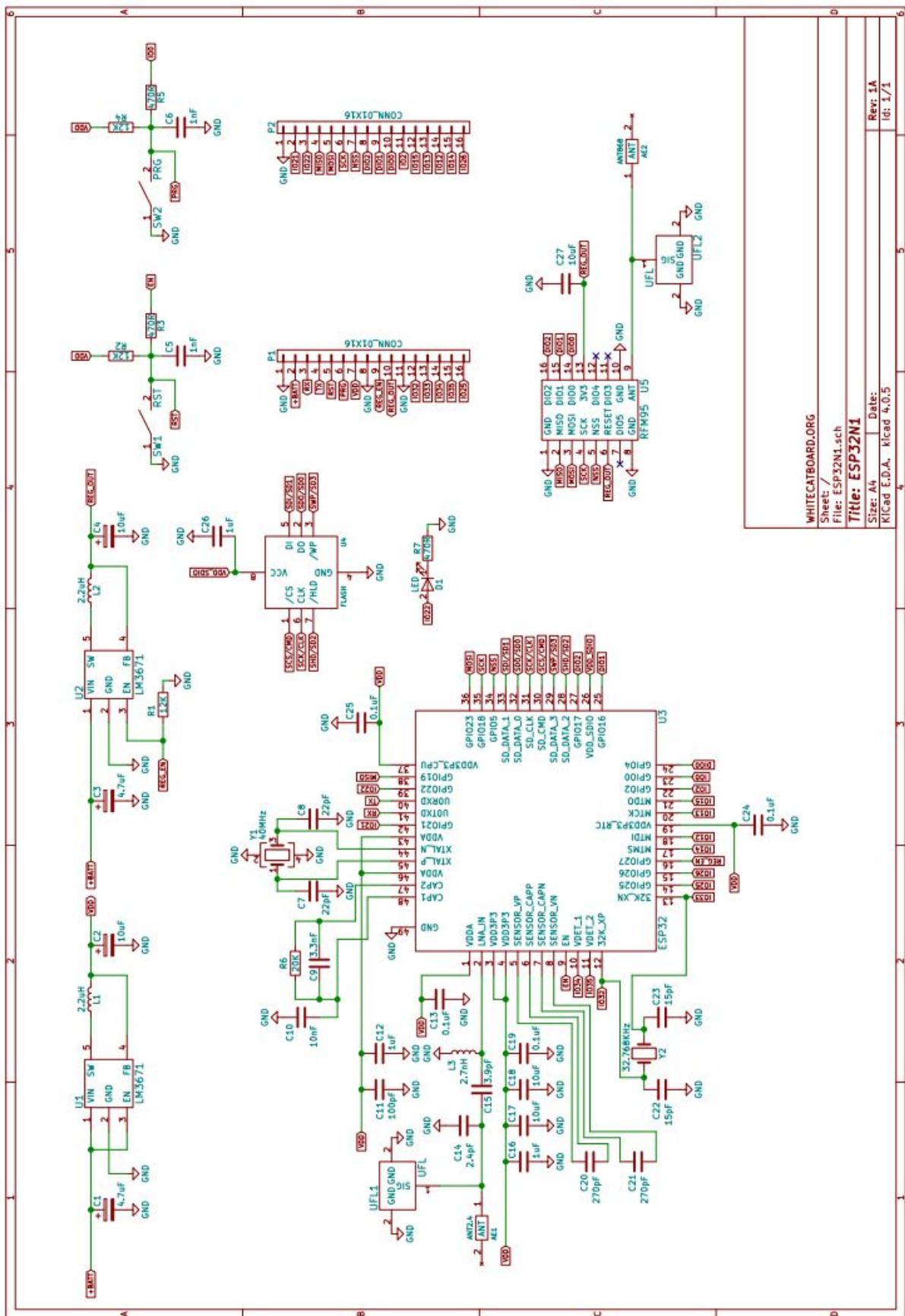
Fig-149. Dashboard con los datos recibidos.

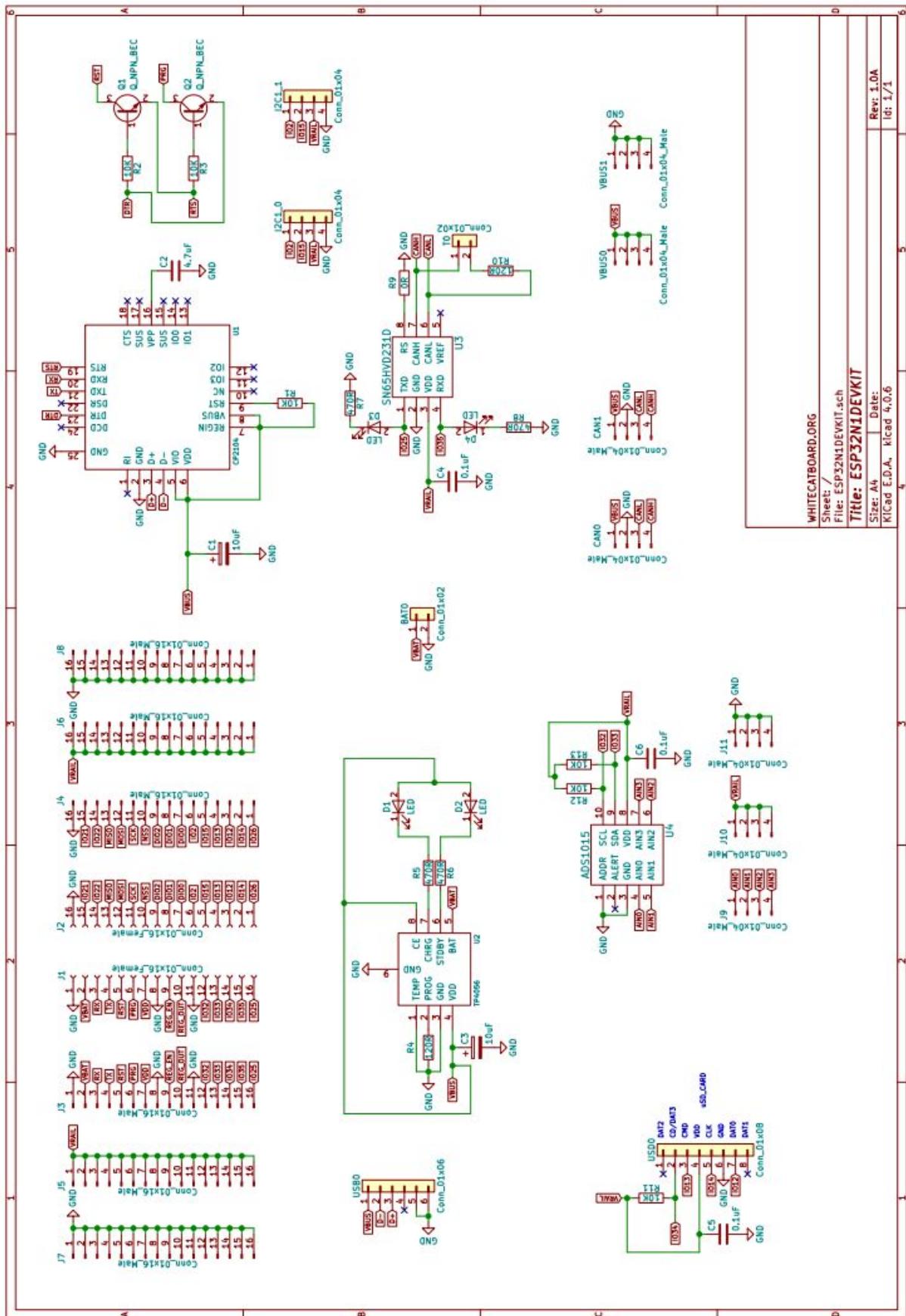


Fig-150. Nodo en su hábitat natural, recuerda meterlo en una caja.

## 9. Anexos

## 9.1. Esquemáticos





## 9.2. Documentación Técnica

Dado que es un proyecto de origen público, toda la documentación técnica referente al mismo, se encuentra en este caso en los diferentes repositorios de github. Todo uso que se haga de dicha documentación será siempre bajo la responsabilidad del usuario, y la atribución del origen de la fuente a Whitecatboard.org.

Repositorio del sistema operativo Lua-RTOS:

<https://github.com/whitecatboard/Lua-RTOS-ESP32>

Repositorio de la consola de Whitecat:

<https://github.com/whitecatboard/whitecat-console>

Repositorio del agente entre el navegador y la placa:

<https://github.com/whitecatboard/whitecat-create-agent>

Repositorio de los diseños hardware:

<https://github.com/whitecatboard/whitecat-hardware>

Repositorio del IDE:

<https://github.com/whitecatboard/whitecat-ide>

### 9.3. Referencias y enlaces

Otras tecnologías que se emplean en el ecosistema Whitecat y que por su extensión no han sido incluidas en dicho manual son:

Lenguaje de programación Lua:

<https://www.lua.org/home.html>

Sistema operativo base del sistema operativo Lua RTOS (FreeRTOS):

<https://www.freertos.org/>

Sistema Node-RED:

<https://nodered.org/>

Sistema IBM Cloud Bluemix:

<https://www.ibm.com/cloud/>

Proyecto Blockly (proyecto de Google sobre bloques para la programación):

<https://developers.google.com/blockly/>

Espressif (fabricante del chip utilizado en la ESP32N1):

<http://espressif.com/>

Estándar MQTT desarrollado por IBM:

<http://mqtt.org/>

LoRa Alliance (Alianza de fabricantes e integradores de la tecnología LoRaWAN):

<https://www.lora-alliance.org/technology>

Semtech (empresa desarrolladora de la tecnología LoRa):

<https://www.semtech.com/>

Protocolo One-Wire

<https://es.wikipedia.org/wiki/1-Wire>