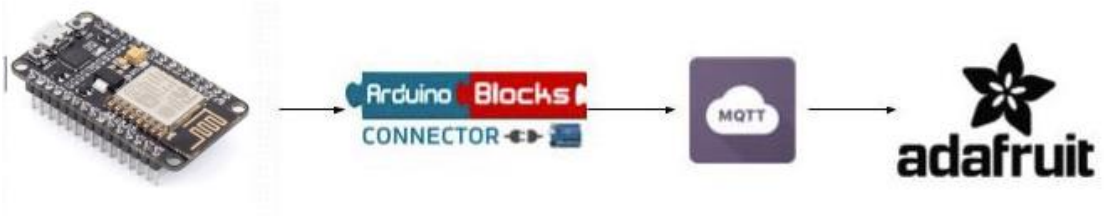
Módulo 2: ArduinoBlocks. Programación por bloques



1 ArduinoBlocks. Introducción 2
1.1 Características 2
1.2 Crear una cuenta 4
1.3 Empezar un proyecto 5
1.4 El entorno 6
1.5 Instalar los drivers de la placa 7
2 Prácticas de programación básicas 10
2.1 Ejemplo 1: alumbrado de escalera 10
2.2 Ejemplo 2: Medida de temperatura con sensor DHT22. 12
2.3 Ejemplo3: Medida de nivel de iluminación con LDR. 14
3 Prácticas de programación IoT con ThingSpeak 16
3.1 Enviar datos a una canal de ThingSpeak 16
3.2 Recibir datos de un canal de ThingSpeak. 20
4 Prácticas de programación IoT con Adafruit 24
4.1 Control de NodeMCU a distancia. 24
4.2 Control de NodeMCU a distancia y recepción de su estado 27
5 IoT con Arduino + ESP-01 31
5.1 Reprogramación del ESP-01 para funcionar a 9600bps 33
5.2 Comunicación bidireccional entre Arduino y un móvil 35
6 Bibliografía 41
7 Créditos y licencia 41

v 2.0 (octubre 2020)

1 ArduinoBlocks. Introducción

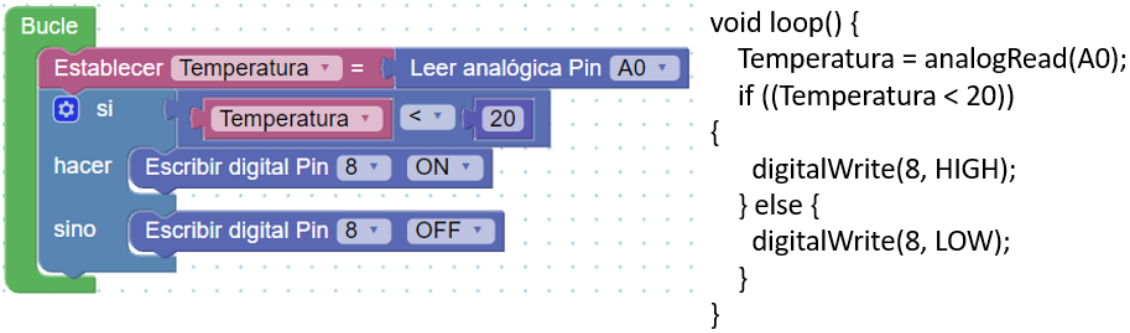
1.1 Características

ArduinoBlocks es una aplicación web destinada a programar tarjetas de desarrollo basadas en microcontroladores, tales como la popular familia de tarjetas "Arduino". Entre sus características cabe señalar:

- La programación es **gráfica**, es decir a través de bloques que se van apilando como si fueran las piezas de un puzzle, al estilo de programas como Scracth, mBlock, bitBloq, AppInventor, Visualino, Facilino… Por lo tanto resulta muy **sencillo de usar** y es ideal en etapas educativas como primaria o primeros cursos de secundaria.
- Es una **plataforma web**, por lo que en principio no se necesita instalar ningún programa en el ordenador, solo se necesita tener una conexión a internet. Ideal por tanto en centros escolares donde el **parque informático** suele estar algo **desfasado** pero en cambio se cuenta con buenas conexiones a internet, ya que la carga de procesamiento la soporta el servidor en lugar del ordenador local desde el que accedemos al servicio web. En realidad si que hay que instalar una aplicación, que se necesita para transferir el programa creado (que en la aplicación web se llama proyecto) a la tarjeta de desarrollo que estemos programando: ArduinoBlocks Connector.
- Soporta varios tipos de tarjetas de desarrollo, mayormente de la familia Arduino: Uno, Mega, Nano, Leonardo… y desde la actualización de ArduinoBlocks Connector a la v4 incorpora también las tarjetas NodeMCU y WeMos (basadas en el SoC ESP8266), los que nos permite **crear proyectos IoT** de forma sencilla.
- Incluye una **gran colección de sensores y actuadores**, que en otras aplicaciones gráficas como S4A (Scracth for Arduino), Visualino, mBlock, etc. no existen, como por ejemplo los módulos Reloj RTC, GPS, tarjeta SD, pantallas OLED, matrices de leds 8x8, LM35, DS18B20, DHT22… por lo que el límite lo pone nuestra creatividad.
- Los programas que realizamos (**proyectos**) se quedan guardados en nuestra cuenta en el servidor web de la aplicación, con lo que los tenemos siempre **disponibles**, **independientemente del ordenador** que usemos cada vez que nos conectemos. Es lo que se llama actualmente "la nube" (vamos, lo que de toda la vida ha sido guardar los datos en el servidor). También tenemos la posibilidad de hacer una copia del proyecto y guardarla en nuestro ordenador local.
- Permite **compartir** nuestros programas, o ver los programas que comparten otros usuarios, y lo que es más útil: hacer una copia de esos otros proyectos en nuestra cuenta en la nube, y así **aprender** como otros han resuelto un determinado problema, hacer modificaciones, y compartirlas, y fomentar de esta manera el conocimiento entre la comunidad.

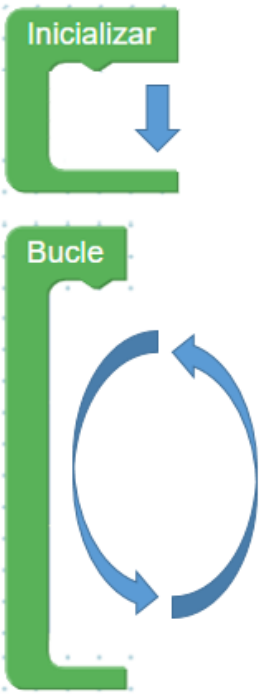
El único inconveniente de esta plataforma, es que debido a que está pensaba para el mundo educativo, su principal característica es su sencillez de uso, y ésta es a su vez su talón de Aquiles: no permite aprovechar todas las características que nos ofrece la tarjeta NodeMCU. Por ejemplo no podemos poner el SoC ESP8266 en modo AP (Access Point) para difundir una red Wifi a la

que se conecten otros dispositivos, ni tampoco nos permite crear una pequeña página web en la tarjeta NodeMCU a la que accedamos vía Wifi para ver información de sus sensores conectados o activar sus salidas digitales, cosas que programando la tarjeta con el IDE Arduino si podemos hacer, como se verá posiblemente en un futuro curso de nivel avanzado (continuación a éste de nivel básico). La solución si deseamos hacer algo así es montar un servidor MQTT en la red local como Mosquito y que sea éste quien se encargue de enviar los datos a un servidor web local.



Como ya se ha dicho, la programación se realiza de forma gráfica (visual) mediante bloques que se van apilando unos encima de otros, en lugar de escribir código. Aunque la plataforma permite exportar el proyecto construido con estos bloques a su equivalente en código, permitiendo así que los alumnos empiecen a comprender la programación mediante código escrito, o su uso en entornos de programación como el IDE de Arduino. Los bloques tienen distinto color en función de la categoría (o tipo) a la que pertenecen, facilitando así su identificación a la hora de buscar el bloque deseado.

Estos bloques se colocarán dentro de dos bloques de color verde llamados “Inicializar” y “Bucle” que representan las dos partes principales de que consta un programa en estas tarjetas de desarrollo:



- **Inicializar**: En este bloque se introducen las órdenes que queremos ejecutar una única vez, y es el primero de los dos bloques en ejecutarse. Se utiliza por tanto para llevar a cabo configuraciones iniciales en nuestra tarjeta, que se realizan al principio del programa y luego ya no es necesario cambiarlas. Por ejemplo, si vamos a conectar nuestra tarjeta a una red Wifi para un proyecto de IoT, en este bloque podemos colocar los datos necesarios para establecer la comunicación como el nombre de la red, el usuario, la contraseña, etc. Sí, ya sé que el nombre es muy rimbombante, y debería llamarse “Iniciar” pero es lo que tiene meter anglicismos en nuestro idioma a pesar de tener un término análogo que lo describe a la perfección.

- **Bucle**: Aquí colocamos las órdenes que van a formar el núcleo de nuestro programa. Estas órdenes se van a ejecutar de arriba abajo, y cuando se acaben, se volverá a repetir el conjunto "Bucle" de nuevo, y así sucesivamente repitiéndose por tanto cientos de veces por segundo. A cada repetición se le llama ciclo, y la tarjeta estará repitiendo ciclos mientras tenga alimentación. Dentro de este conjunto de instrucciones que forman el ciclo, podemos incluir aquellas que habíamos ubicado en el bloque "Inicializar", pero aunque el programa tal vez funcione igual, estos datos se van a estar leyendo cientos de veces por segundo, de forma innecesaria y alargando el tiempo de ejecución. Por tanto, aquellas instrucciones que no es necesario repetir es mejor colocarlas en el bloque "Inicializar".

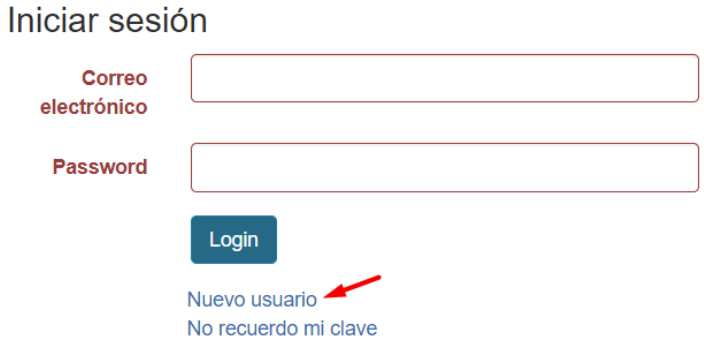
1.2 Crear una cuenta

Para realizar programas en ArduinoBlocks hay que tener creada una cuenta. Posteriormente veremos que, para cargar en la placa el programa realizado, hay que tener instalado y ejecutándose en el ordenador, un pequeño archivo (ArduinoBlocks-connector), que tiene las librerías y los drivers necesarios para interpretar el programa realizado.

Entramos en la página del proyecto en la dirección: arduinoblocks.com. Al hacerlo, aparece la pantalla inicial, en la que se pueden ver proyectos compartidos por otros usuarios. Además, en la parte superior se dispone de un buscador de proyectos.

El acceso a usuarios se encuentra en la parte superior derecha.

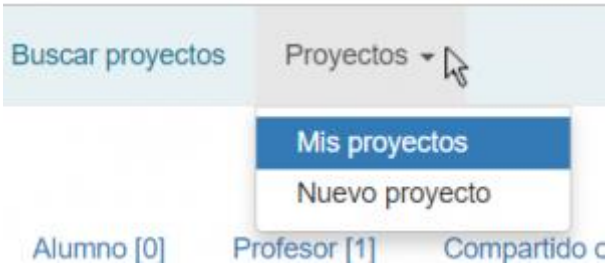
Tras hacer clic en "Iniciar sesión", se accede a la pantalla de Inicio de Sesión. Si aún no dispones de usuario creado, ahora tendrás que hacer clic sobre Nuevo usuario.



Al acceder a la siguiente pantalla, ya puedes registrarte con tu correo electrónico, contraseña y demás datos solicitados. Una vez realizado este último paso, ya está la cuenta creada. No olvides visitar tu correo para confirmar tu dirección en el enlace del email que la plataforma te habrá enviado. Si no lo encuentras, ya sabes, revisa la carpeta de correo basura o "spam".

1.3 Empezar un proyecto

Una vez has accedido a tu cuenta, pinchando con el ratón en el desplegable que hay en la parte superior de la pantalla, junto a Proyectos, se abren dos opciones: Mis proyectos y Nuevo proyecto.



En Mis proyectos se accede a trabajos anteriores que hayas realizado. Si es la primera vez que accedes, no te asustes, es normal que la lista esté en blanco. En Nuevo proyecto se comienza un nuevo proyecto.

Al entrar en Nuevo proyecto, ya se puede seleccionar cómo se va a trabajar. Si va a ser un trabajo personal (Proyecto personal), si vas a ser un profesor que realice el seguimiento de proyectos de alumnos (Profesor), o si se accede como alumno de un proyecto educativo (Alumno).

Cuando inicias un proyecto personal, aparece la última pantalla que hay que configurar antes de entrar en el editor del programa. En ella hay que elegir el tipo de placa con la que se va a trabajar. En este curso del Internet de las cosas (IoT) se va a utilizar habitualmente la placa NodeMCU, pero se puede usar indistintamente cualquiera de las otras, solo con cambiar esta opción de configuración inicial.

A continuación, el último campo obligatorio es el nombre del proyecto.

El resto de parámetros están para completar información al respecto, pero no es obligatorio rellenarlos.

Finalmente, en la parte inferior de la pantalla hay que hacer clic sobre el icono Nuevo proyecto. De esta forma se accede ya al editor.

1.4 El entorno

Una vez dentro de la aplicación podemos distinguir las siguientes partes:

- En la parte superior derecha, los iconos de guardar programa, cargarlo en la tarjeta, abrir la consola del terminal de E/S y refrescar la conexión.
- En el centro, la parte destinada a incluir el programa, dividida en los bloques "Inicializar" y "Bucle"
- En la parte izquierda, la lista de apartados en los que se clasifican los diferentes bloques disponibles.

No te preocupes si se te olvida guardar el proyecto, la plataforma lo hace cada cierto tiempo de forma automática, pero puedes forzar a guardarlo en ese instante pulsando en el icono rojo del disquete.

También es normal al empezar comprobar que el botón de cargar el programa está deshabilitado (aparece con un color más tenue). Eso es debido a que antes hay que instalar los

drivers en el ordenador que permitan reconocer la placa desde el editor web y comunicarse con ella.

Haciendo clic sobre cualquiera de los apartados de la parte izquierda de la pantalla, se encuentran los diferentes bloques de funcionalidades, que son los que se arrastran dentro de *Inicializar* y *Bucle*. Sabed también que hay partes del programa que se pueden incluir también fuera de estas dos partes principales: las funciones.

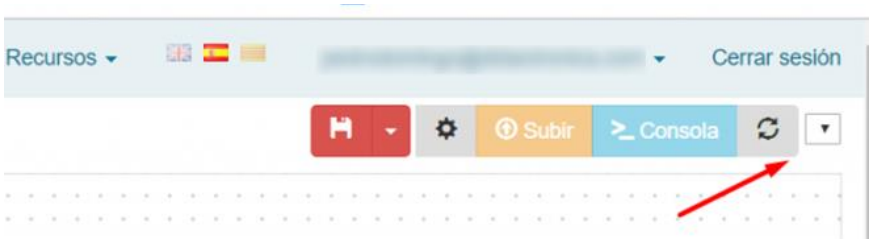
1.5 Instalar los drivers de la placa

Ahora vamos a ver como instalar los drivers (ArduinoBlocks-connector) necesarios para poder conectar con la placa y comprobar que la conexión es correcta. Es importante recordar que sin ejecutar ArduinoBlocks-connector, se pueden realizar programas pero NO se pueden cargar en la placa.

Para poder cargar los programas desde ArduinoBlocks en una placa (NodeMCU o Arduino):

1. Hay que tener instalado el programa ArduinoBlocks-connector.
2. Hay que estar ejecutando ArduinoBlocks-connector.

Desde el editor en el que realizamos los programas, es fácil ver si se ha detectado una placa o no. Para hacerlo, basta con fijarse en la parte superior derecha de la pantalla:

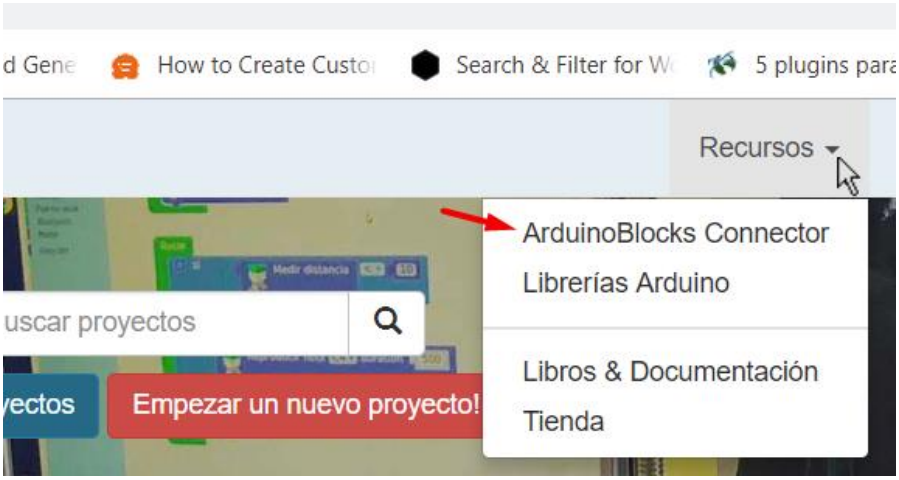


Cuando la placa NodeMCU está correctamente conectada, aparecen en el espacio señalado con la flecha roja en la imagen superior, una serie de caracteres que indican el número de puerto

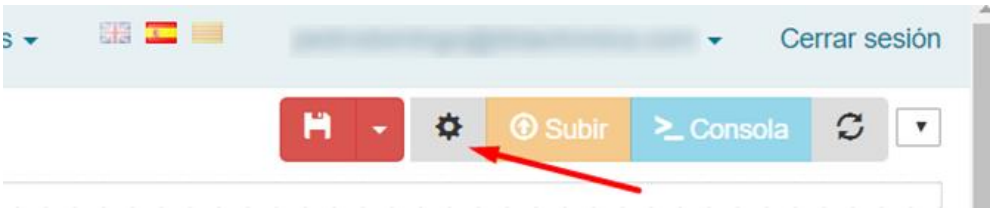
al que se ha conectado (por ejemplo COM3 en Windows). En la imagen aún no se aprecia, porque no se ha realizado la conexión.

La instalación de ArduinoBlocks-connector solo hay que realizarla una vez por ordenador, pero el archivo habrá que tenerlo abierto cada vez que se quiera cargar un programa. Podemos acceder al área de descarga directamente desde la página principal de arduinoblocks.com, o desde el editor en el que estemos realizando un programa.

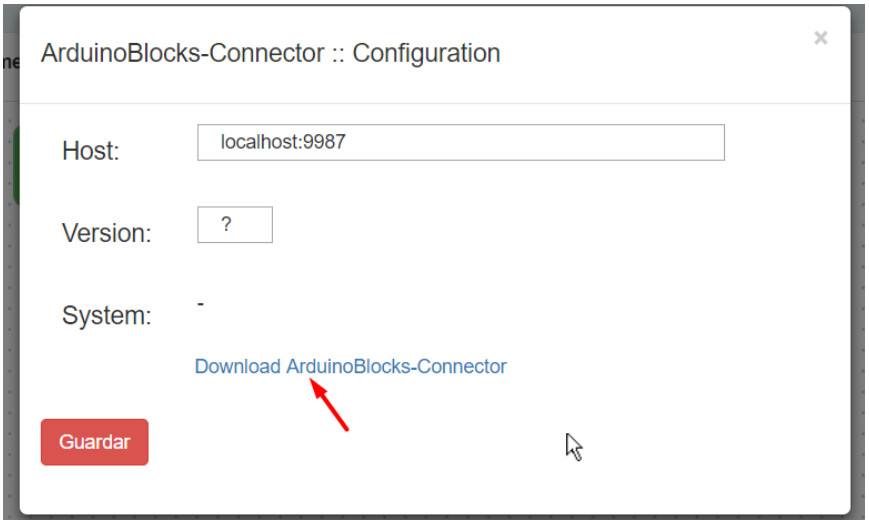
Desde la página principal, el instalador se localiza en Recursos:



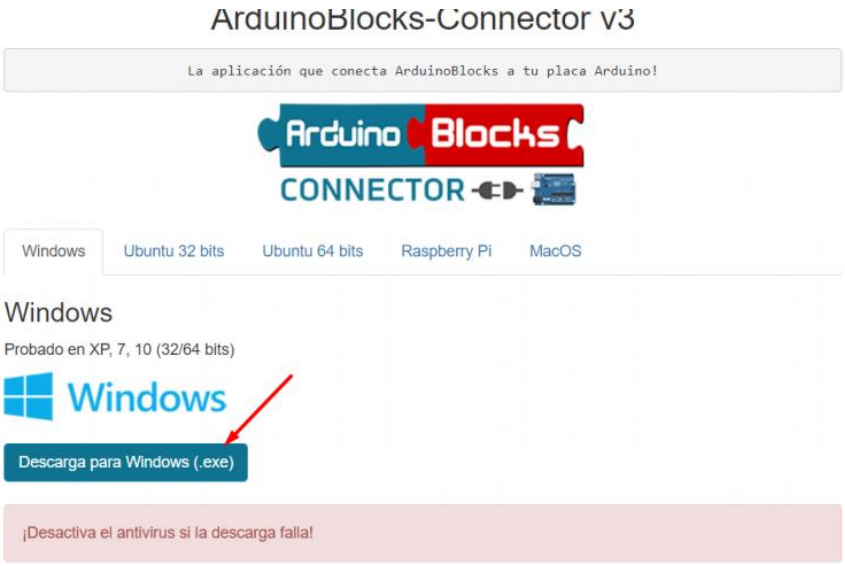
Desde el propio editor de programas, el instalador se localiza en la parte superior derecha, haciendo clic sobre el icono del «engranaje»:



Y a continuación, pinchando sobre el enlace señalado:



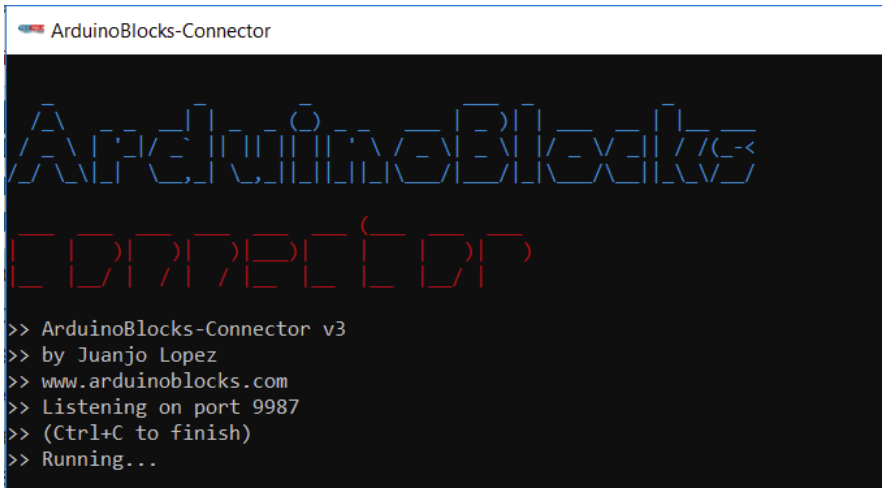
Por cualquiera de las dos vías, se llega a la siguiente pantalla, en la que hay que elegir qué sistema operativo tenemos en el ordenador en el que vamos a instalar ArduinoBlocks-connector, y a continuación, darle a descargar.



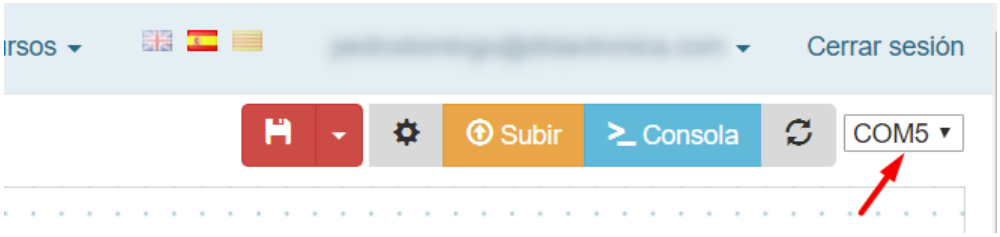
Una vez descargado el archivo, hay que ejecutarlo y esperar a que se instale completamente. El proceso puede tardar varios minutos, dependiendo del ordenador. Al acabar, aparecerá un icono en el escritorio.

Ese icono es el que hay que ejecutar cuando queramos cargar un programa.

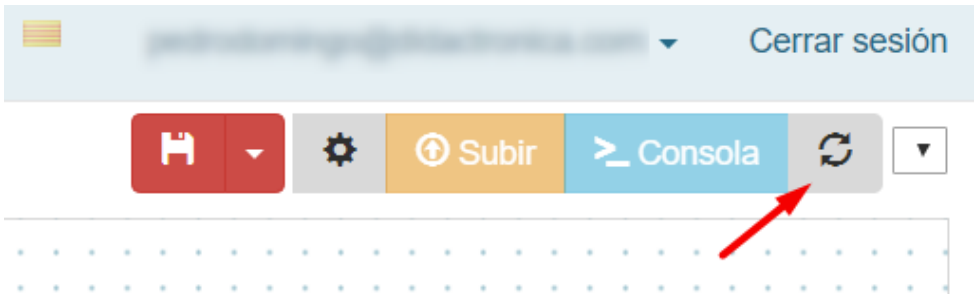
Al hacer doble clic sobre el icono anterior, se abrirá la siguiente pantalla en el escritorio. Simplemente hay que dejarla abierta (se puede minimizar).



Ahora falta lo mas obvio, pero también lo mas importante; conectar la placa NodeMCU al ordenador. Al hacerlo aparecerá un número en la parte superior derecha del editor, que es el puerto al que está conectada. No es necesario hacer nada con ese dato, simplemente indica que está correctamente conectada.



Si no aparece, basta con hacer clic sobre el icono de «refrescar» señalado con la flecha roja en la imagen siguiente:



Si la placa está bien conectada y Arduinoblocks-connector está ejecutándose, además de aparecer el puerto de conexión, veremos también que los iconos Subir y Consola aparecen con mayor nitidez, y que ya sí se puede hacer clic sobre ellos.

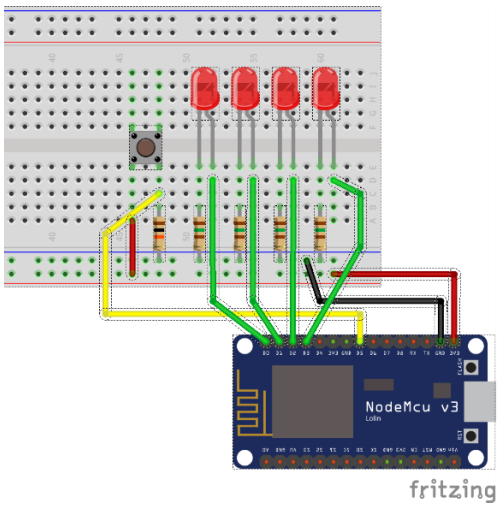
Todo está listo para cargar el primer programa.

2 Prácticas de programación básicas

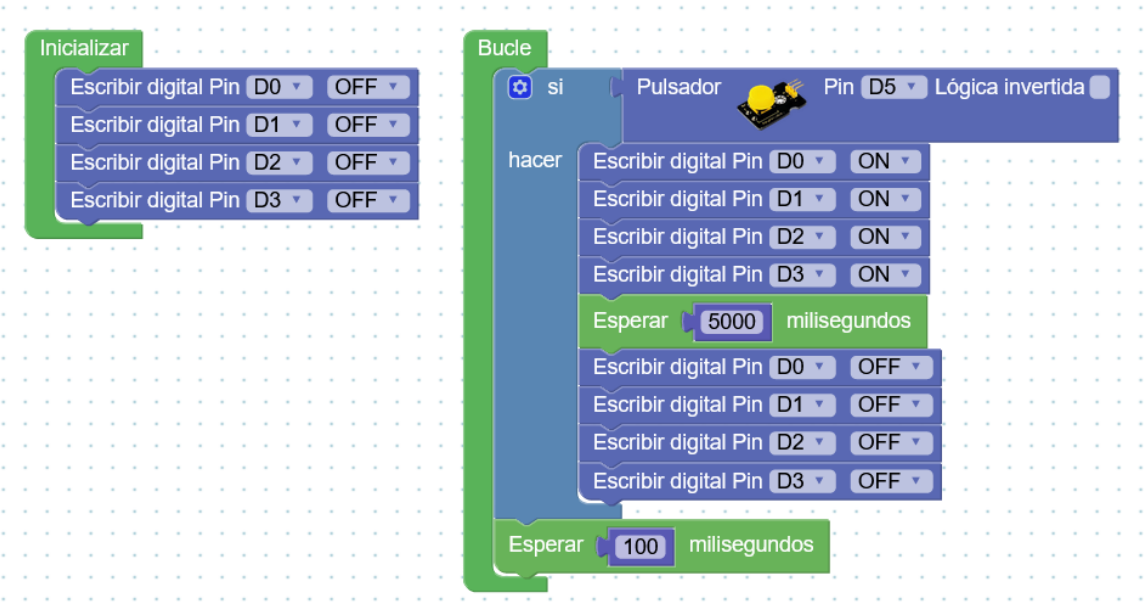
2.1 Ejemplo 1: alumbrado de escalera

Con un pulsador encenderemos cuatro leds que permanecerán encendidos durante 5 segundos. Al cabo de este tiempo se apagarán. El proceso se repite cada vez que se active el pulsador.

Esquema de conexión:



Solución propuesta:



Los bloques a usar son:

CATEGORIA	COLOR	BLOQUES	USO
Entrada/Salida	Violeta	Escribir digital Pin	Activar o desactivar los leds
Sensor	Violeta	Pulsador	Leer el estado del botón
Tiempo	Verde	Esperar milisegundos	Producir un retardo
Lógica	Azul	Si	Tomar una decisión
Matemáticas	Violeta	Número	Introducir valores numéricos en los bloques Esperar

En este ejemplo hacemos uso del bloque Esperar de 100 milisegundos para provocar una espera que evite los rebotes del pulsador.

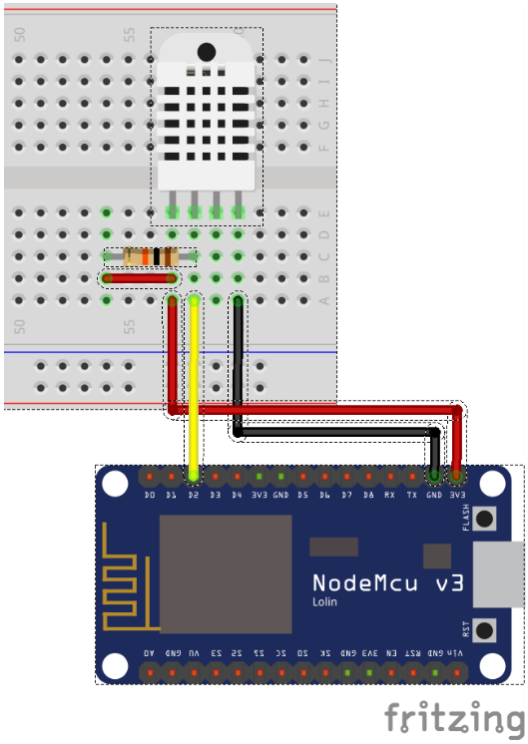
El pulsador lo hemos leído con un bloque del grupo de Sensores pero perfectamente podríamos haber usado el bloque de "Leer digital Pin…" del grupo Entrada/Salida.

PRÁCTICA 1: A partir del ejemplo anterior, realizar las modificaciones necesarias para conseguir que, cuando se pulse el pulsador, las luces se enciendan de una en una (cada 100ms) siguiendo la secuencia D1, D2, D3, D4 y se apaguen al cabo de 5 segundos en orden inverso.

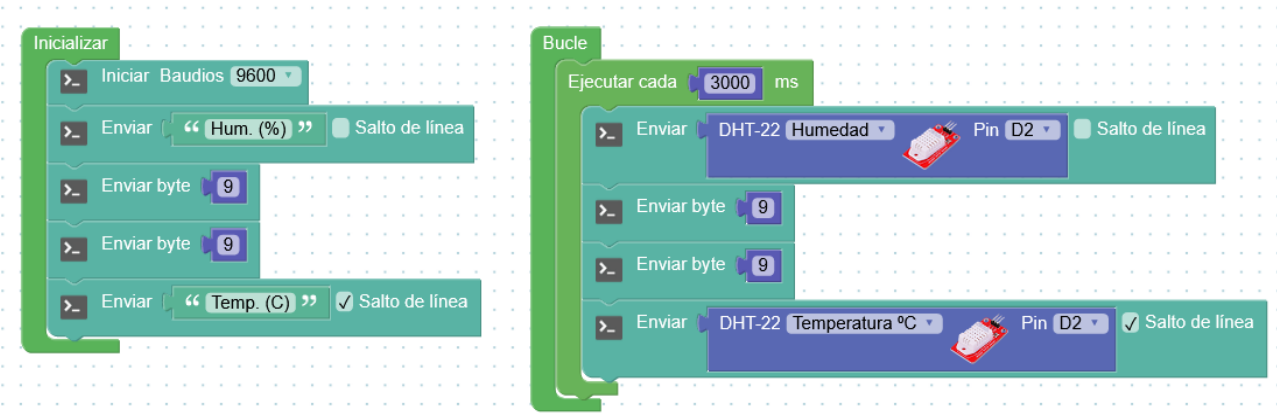
2.2 Ejemplo 2: Medida de temperatura con sensor DHT22.

Con ayuda de un sensor de temperatura DHT22 (más preciso que un DHT11) leeremos el valor de temperatura y humedad relativa y lo mostraremos en el monitor serie cada 3 segundos.

Esquema de conexión:



Solución propuesta:



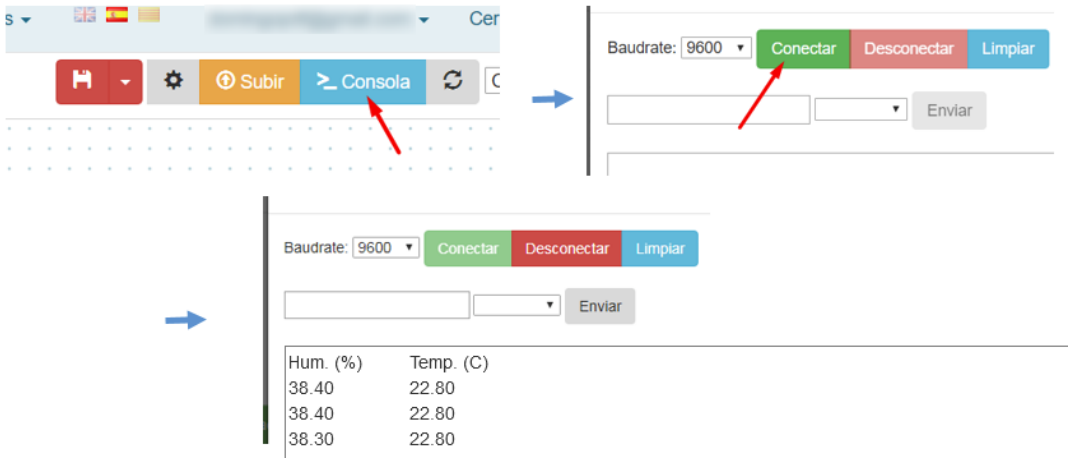
Los bloques a usar son:

CATEGORIA	COLOR	BLOQUES	USO
Puerto Serie	Aguamarina (azul verdoso)	Iniciar Enviar texto Enviar byte	Gestionar la comunicación por el puerto serie.
Sensor	Violeta	DHT-22	Leer los valores del sensor
Tiempo	Verde	Ejecutar cada	Producir un retardo
Matemáticas	Violeta	Número	Introducir valores numéricos
Texto	Aguamarina	Texto	Introducir una cadena de caracteres en los bloques de Enviar

Se utiliza el bloque de "Ejecutar cada…" del grupo Tiempo para dar tiempo al sensor a medir la siguiente pareja de valores (tarda 2 segundos en obtener los valores).

Se ha usado el byte 9 (primer bloque del grupo Matemáticas) porque no es posible introducir el código ascii del tabulador horizontal usando la secuencia de escape "\t". También se ha evitado el carácter ascii 169 (el símbolo "º" para los grados) porque la consola de ArduinoBlocks no es capaz de mostrarlo e interfiere en los datos recibidos por el puerto serie.

Para ver el resultado en la consola de ArduinoBlocks debemos pulsar sobre el icono superior con el texto "Consola", y en la nueva ventana elegir la velocidad de conexión utilizada en nuestro programa (9600 baudios en el ejemplo) y pulsar sobre "Conectar" (vea la siguiente imagen).



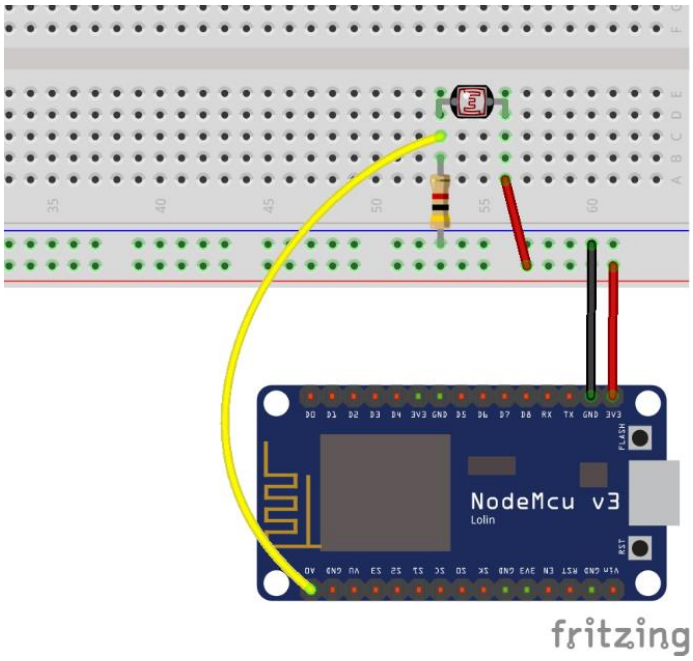
PRÁCTICA_2: Termómetro digital. A partir del ejemplo anterior, conectar cuatro Leds en los pines D0, D1, D3 y D4 y realizar las modificaciones necesarias para conseguir que se enciendan en función de la temperatura. (D0=20º,D1=21º,D3=22º,D4=23º). Añadir un pulsador 'P' que encienda todos los leds cuando se mantenga pulsado.

Nota1: Recuerda conectar el pulsador con una resistencia de pull-up/down externa de 10k

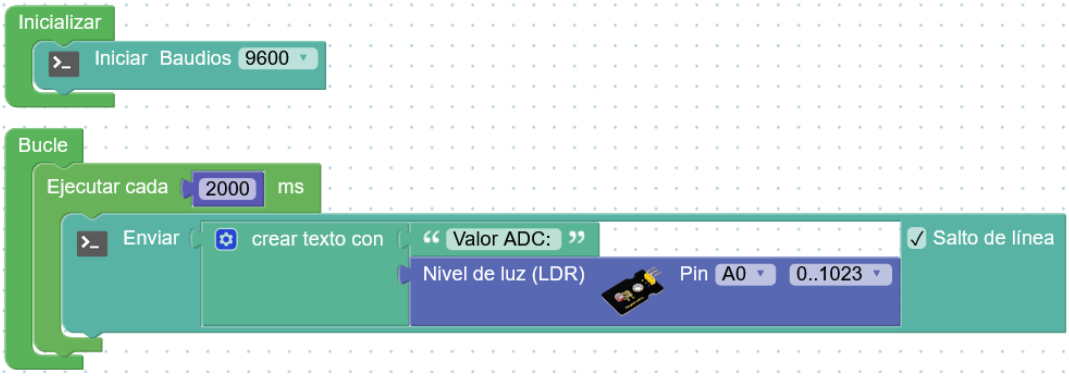
2.3 Ejemplo3: Medida de nivel de iluminación con LDR.

Con ayuda de una resistencia dependiente de la luz (LDR) vamos a probar el conversor ADC de la placa. El programa leerá el valor analógico en el pin A0 y lo mostrará por el terminal serie cada 2 segundos.

Esquema de conexión:



Solución propuesta:



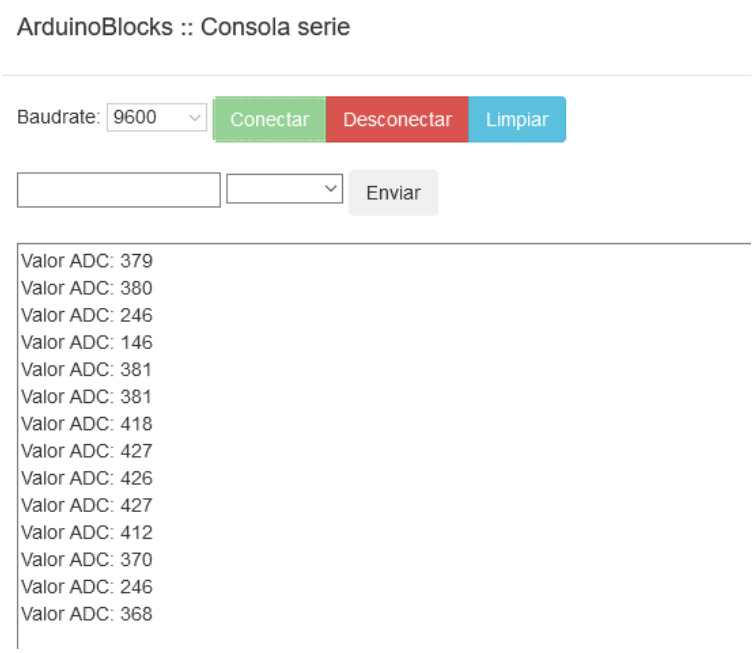
Los bloques a usar son:

CATEGORIA	COLOR	BLOQUES	USO
Puerto Serie	Aguamarina (azul verdoso)	Iniciar Enviar texto	Gestionar la comunicación por el puerto serie.
Sensor	Violeta	Nivel de luz (LDR)	Leer los valores del sensor
Tiempo	Verde	Ejecutar cada	Producir un retardo
Matemáticas	Violeta	Número	Introducir valores numéricos
Texto	Aguamarina	Texto Crear texto con	Crear la cadena de caracteres que se usa en el bloque de Enviar

Para el pin A0 se ha usado el bloque de Nivel de Luz del grupo de sensores, pero perfectamente se podría haber sustituido por el bloque "Leer Analógica Pin…" del grupo de Entrada/Salida.

También se hace uso del bloque "Crear texto con…" del grupo Texto para concatenar el texto introducido con el valor que devuelve el sensor.

Salida por pantalla:



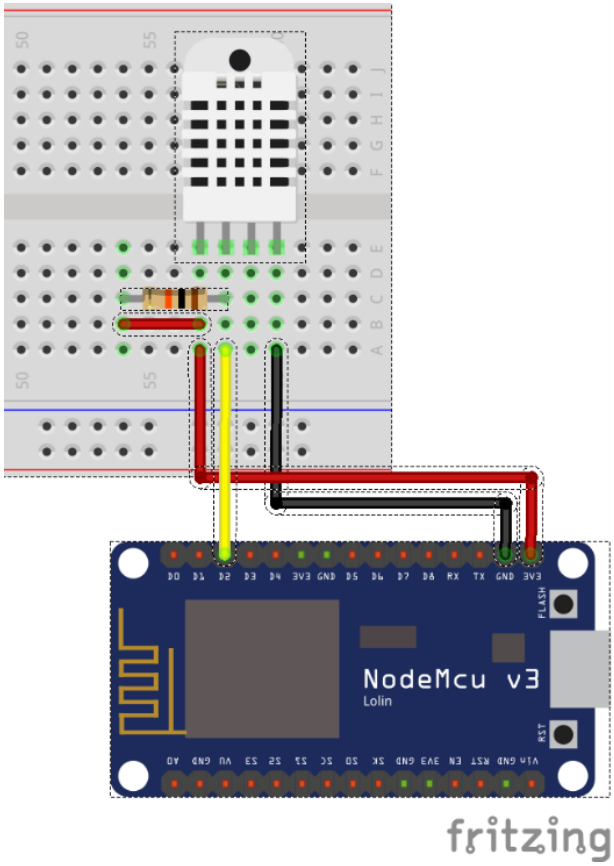
PRÁCTICA_3: A partir del ejemplo anterior, conectar un Led en el pin D1 y un pulsador en el pin D5. El programa debe mostrar por el monitor serie el valor leído del ADC y el estado del Led (ON-OFF). El pulsador 'P' encenderá el Led, si se mantiene pulsado. *Nota1: Coloca el led frente a la LDR en la protoboard.*

3 Prácticas de programación IoT con ThingSpeak

3.1 Enviar datos a una canal de ThingSpeak

Tomar lecturas de temperatura con un DHT22 y enviarlas a un canal creado en nuestra cuenta de ThingSpeak.

Esquema de conexión:



Solución propuesta:

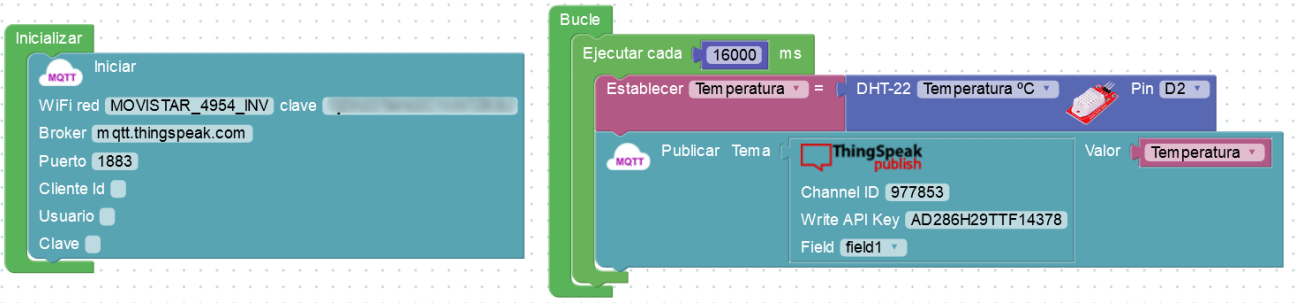
Los bloques que necesitaremos para comunicarnos con el servidor ThingSpeak están en el grupo MQTT y básicamente son tres:

- "MQTT Iniciar" para establecer la comunicación con el servidor de ThingSpeak (iniciarla).
- "MQTT Publicar Tema" para realizar el envío de datos al servidor.
- Y el bloque "ThingSpeak publish" que es el que contiene los datos de configuración del canal de ThingSpeak donde vamos a publicar los valores.

A parte usaremos el bloque del DHT22 del grupo de sensores para tomar la temperatura, el de "Establecer…" dentro de la categoría Variables, pues guardaremos en una variable el dato recibido por el sensor, y que posteriormente enviaremos, y el bloque "Ejecutar cada…" del grupo

Tiempo, para establecer cada cuanto tiempo enviamos al servidor ThingSpeak un dato de temperatura. **Hay que tener en cuenta que a diferencia de las prácticas anteriores, el bloque "Ejecutar cada" no puede ser sustituido por el de "Espera… milisegundos" porque este último detiene la ejecución del programa y se interrumpiría la conexión con el servidor MQTT. Sin embargo con el bloque "Ejecutar cada" conseguimos el retardo deseado pero sin bloquear los procesos que haya en segundo plano como la comunicación con el servidor que tenemos abierta.**

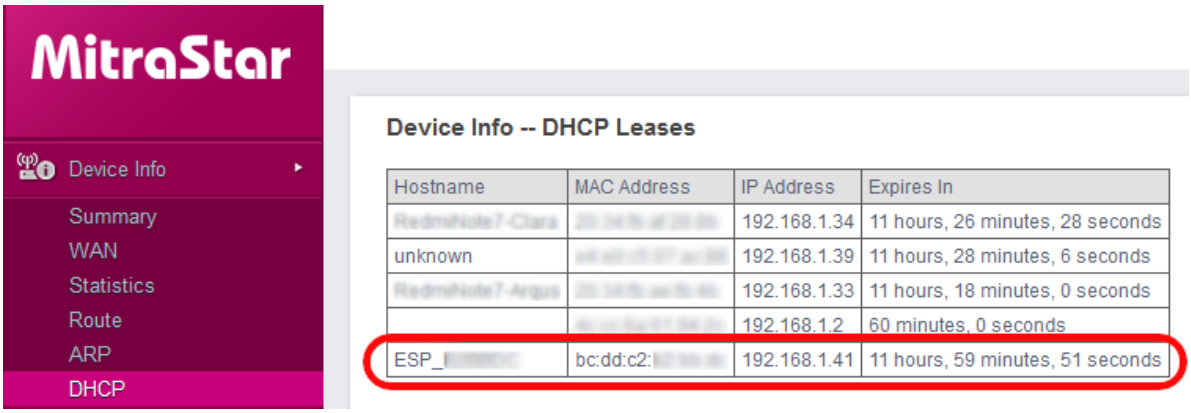
Montamos los bloques como se ven en la siguiente imagen:



En el bloque de "MQTT Iniciar" debemos rellenar los datos necesarios para establecer la comunicación con el servidor de ThingSpeak, en el que previamente debemos tener una cuenta creada (ya lo hicimos en el módulo 3 de este curso).

Los primeros datos a rellenar son los relativos a la conexión Wifi a la que vamos a conectar nuestra tarjeta NodeMCU para que disponga de conexión a Internet y que pueda enviar los datos de temperatura a ThingSpeak. Esos datos son el nombre de la red Wifi y la clave. Recordad que las claves actuales (WPA2) necesitan una longitud mínima de 8 caracteres.

No está de más, una vez montada la práctica y funcionando, comprobar que nuestra tarjeta se ha conectado correctamente a la Wifi entrando en la configuración del router (si tenemos acceso a él) y buscando la tabla DHCP para ver si está ahí la tarjeta (suponemos que el router tiene activado el servicio DHCP para asignar IPs). En la siguiente captura podemos ver que aparece la tarjeta bajo el nombre de ESP_XXXXXX donde XXXXXX es la parte final de la MAC de la tarjeta Wifi que identifica a ésta. Vemos también su MAC, y su IP asignada por el router (192.168.1.41). De esta forma descartamos en caso de fallo de la práctica que estamos montando, que sea porque no se ha conectado a la Wifi correctamente.



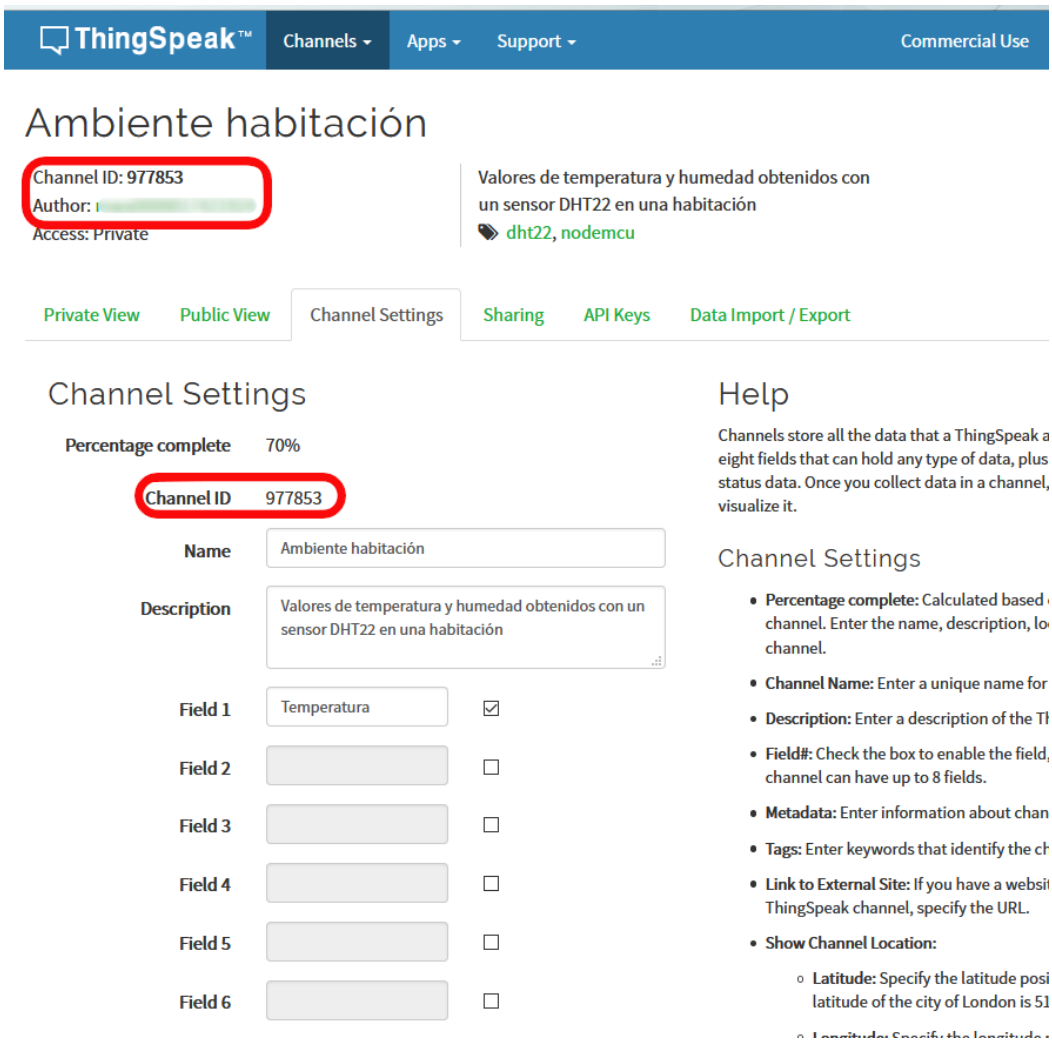
El resto de datos a rellenar en el bloque “MQTT Iniciar” hacen referencia a la configuración del servidor ThingSpeak. En Broker ponemos **mqtt.thingspeak.com**, que es la dirección URL del servidor, y en puerto **1883** (es el puerto que tiene definido el servidor de ThingSpeak para recibir comunicaciones).

Los campos Cliente ID, usuario y contraseña se pueden dejar en blanco, porque no son necesarios para publicar datos en el canal de ThingSpeak (aunque si se rellenan no pasa nada).

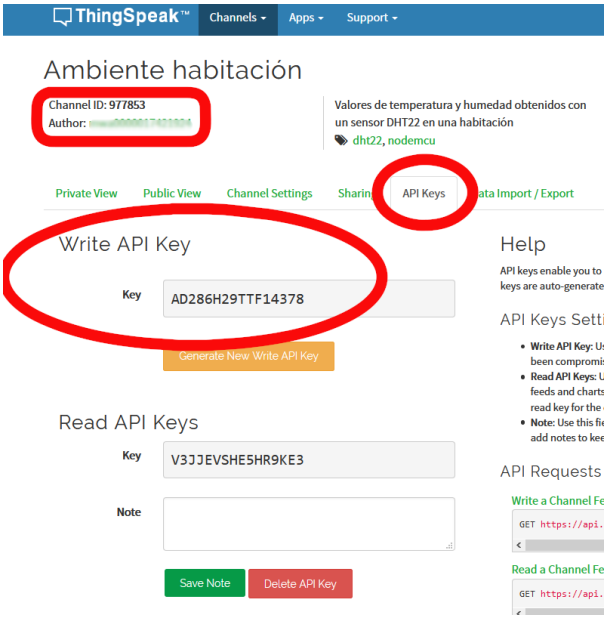
Ahora pasamos a los bloques del programa en sí. En el bloque de “Bucle” hemos empezado poniendo el bloque de “Ejecutar cada 16000 ms” ya que las cuentas gratuitas de ThingSpeak no permiten el envío de datos con una frecuencia mayor de 1 dato cada 15 segundos. Luego para evitar que nos banee el servidor, vamos a enviar temperaturas cada 16 segundos.

Lo que hacemos es leer el dato del bloque “DHT-22” que está conectado al pin D2 de la tarjeta, y guardamos el dato recibido en una variable llamada “Temperatura”. Posteriormente enviaremos esa variable con el bloque “MQTT Publicar tema” para alojar su información en el canal que habremos creado previamente en ThingSpeak.

En la siguiente imagen vemos el canal creado, al que he llamado “Ambiente habitación”, y dentro he creado un campo o Topic (tema), el **Field 1,** llamado “Temperatura”. Se aprecia en la imagen dentro de los marcos rojos el nº de canal (**Channel ID**), que es uno de los campos a introducir en el bloque “ThingSpeak publish”.



En el bloque “ThingSpeak publish” hace falta también escribir la clave API para enviar datos (**Write API key**). Ese dato lo podemos obtener si vamos a la solapa “API Keys” del canal.



Y si no nos hemos equivocado en ningún campo de configuración, podremos ver funcionando la práctica en nuestro canal de ThingSpeak, donde cada 16 segundos se mostrará un dato nuevo. He puesto durante un tiempo el dedo en la carcasa del DHT-22 para hacer variar la temperatura y que la gráfica muestre el pico que se observa.

3.2 Recibir datos de un canal de ThingSpeak.

Realiza un montaje con la NodeMCU que lea datos del canal de ThingSpeak creado en la práctica anterior, y muestre los valores de temperatura recibidos por el puerto serie.

Esquema de conexión:

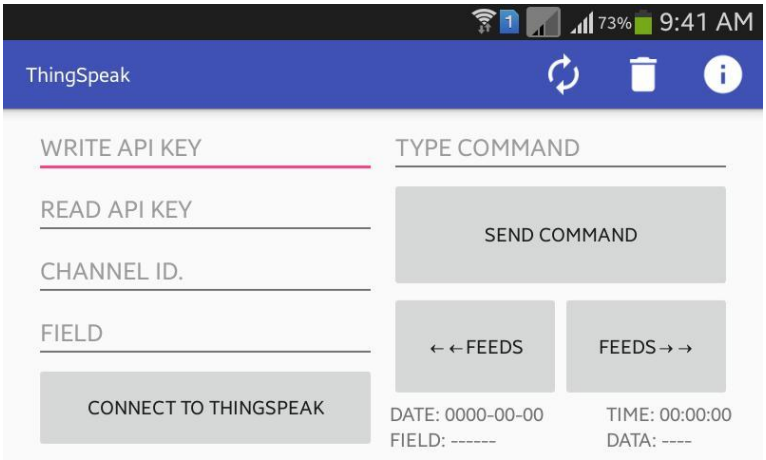
Como no tenemos ningún componente electrónico conectado a la tarjeta no hay que realizar ningún montaje. Solamente conectaremos nuestra NodeMCU con el cable microUSB al ordenador para mostrar en la consola de ArduinoBlocks los datos recibidos.

Solución propuesta:

Vamos a suscribir nuestra NodeMCU al canal de ThingSpeak para recibir datos. Eso implica que otro dispositivo tiene que estar enviando datos (publicando) al canal simultáneamente. Bien puede ser otra NodeMCU con un DHT-22 (la de un compañero del curso) o bien un Arduino con el módulo ESP-01 como se muestra en el punto 5 de estos apuntes.

Sin embargo, por si no tienes disponible otra tarjeta ni un compañero que te eche una mano, en esta práctica vamos a usar para enviar datos un teléfono móvil. Instalaremos en él la app **ThingSpeak (IoT)** desde la Play Store.

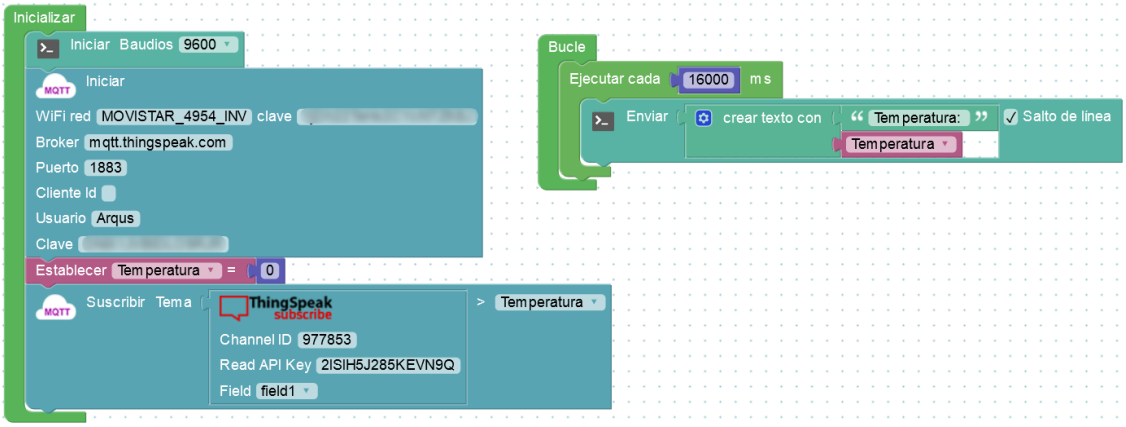
Una vez instalada en nuestro dispositivo, al abrirla nos saldrá una ventana para configurar los datos de nuestro canal: Write y Read API Keys, ID del canal y Field a suscribir o publicar. Rellenaremos los 4 campos con los datos de nuestro canal, y pulsaremos en el botón "Connect to ThingSpeak" que hay debajo para iniciar la conexión.



Posteriormente tecleamos un valor de temperatura en el campo "Type Command" y pulsando en el botón "Send Command" se enviará al canal ese valor.

Con esto ya podemos enviar datos al canal mientras nuestra NodeMCU trabaja recibiendo datos y enviándolos por la consola del puerto serie.

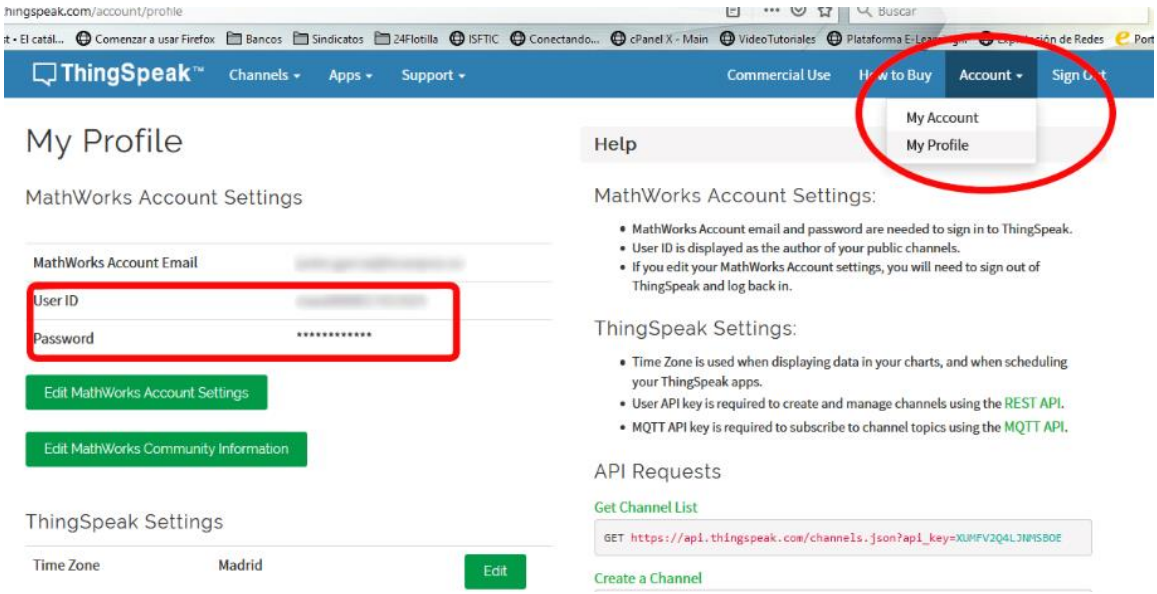
Vamos ahora a preparar el programa para la tarjeta:



En el bloque de Iniciar la comunicación MQTT ponemos los datos de nuestra Wifi (nombre y contraseña) para darle salida a internet a la tarjeta NodeMCU y el servidor a usar que será el de ThingSpeak junto a su puerto (mqtt.thingspeak.com por el puerto 1883).

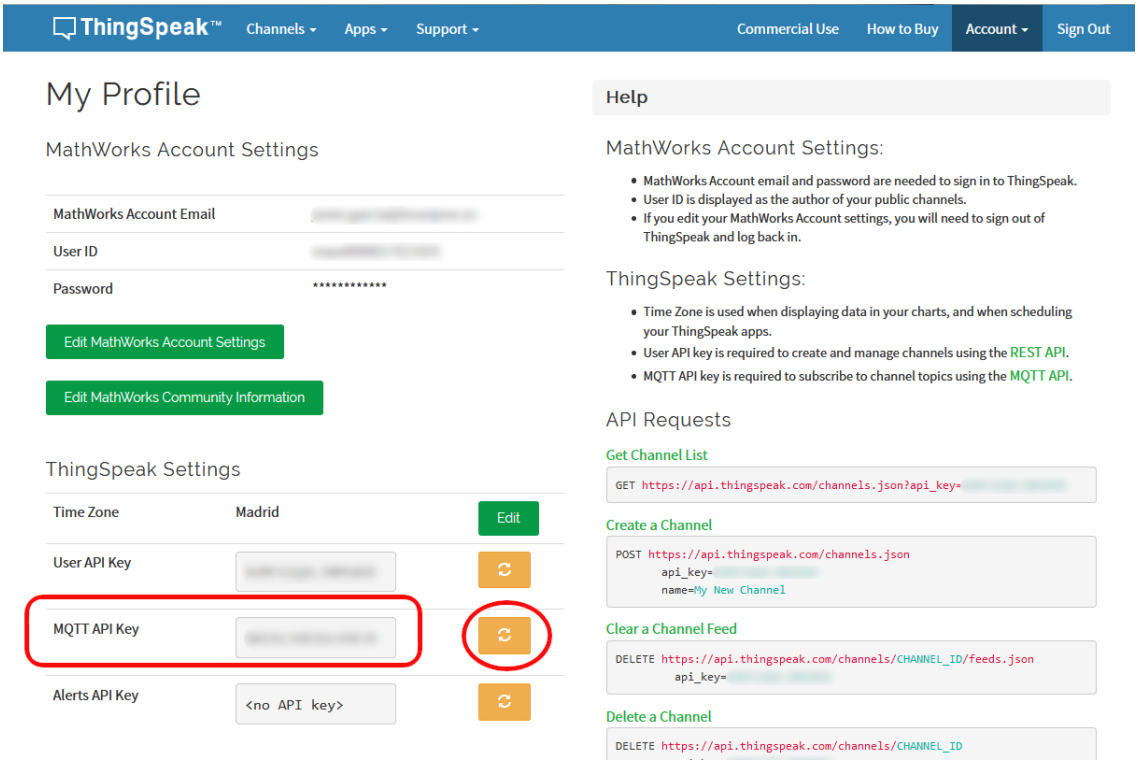
Hasta aquí era igual que en la práctica de publicar datos al canal. Pero ahora viene la diferencia. Para subscribirse a un canal (leer datos) debemos poner un nombre que sea único (vale inventárselo) y una contraseña que será la que nos identifique. Yo en el ejemplo he puesto el nick que suelo usar en internet, pero si queréis aseguraros de que sea único, podéis usar el nombre de usuario que teneis en la cuenta de ThingSpeak. Ese nombre aparecía cuando mirábais el ID del canal, justo debajo bajo el nombre de "Author" (ver capturas de la práctica

anterior). También podemos consultar el nombre de usuario de la cuenta accediendo al menú superior "**Account -> My Profile**":



En la imagen observamos en User ID nuestro nombre de usuario.

Y ahora viene lo más importante de todo: la contraseña. Consultando las instrucciones en inglés de ThingSpeak podemos leer que a la hora de subscribirnos a un canal hay que identificarse con la llamada "**MQT API Key**". Este valor lo tenemos en la ventana de nuestro perfil (Account -> My Profile), donde acabamos de ver que también está nuestro nombre de usuario.



Si veis que el campo de la "MQTT API Key" está en blanco (pone <no API key>) pulsad sobre el botón naranja que hay a la derecha para crear una nueva clave. Este botón se utiliza para cambiar la clave por una nueva cuando se ha visto comprometida la que tenemos (alguien ha tenido acceso a ella). **Esta clave es la que hay que poner en el campo "Clave" del bloque MQTT Iniciar de ArduinoBlocks para subscribirse a un canal.** Esta clave es realmente la que nos identifica y por eso el campo usuario, aunque es obligatorio de rellenar, es indiferente lo que pongamos en él mientras sea único.

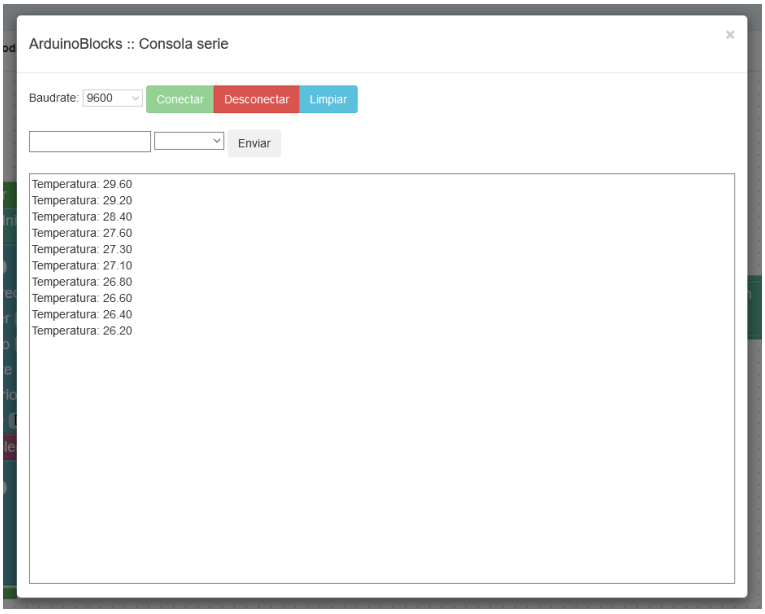
Posteriormente, como veis en la captura de los bloques, creamos una variable de tipo numérica llamada "Temperatura" para almacenar en ella el dato que recibamos del canal.

Y a continuación ponemos el bloque de subscribirnos a un tema. En ese bloque ponemos el bloque de subscripciones a ThingSpeak con los datos necesarios:

- ID del canal a leer
- La **Read API Key** que la podemos encontrar en la solapa de API Keys del canal, justo debajo de la de Write API Key (ver práctica anteior).
- El campo o Topic a leer, que en nuestro ejemplo es el Field1, el mismo que usamos en la práctica anterior.

También hay que indicar en el bloque de subscribirnos el nombre de la variable donde se guardará el dato leido del canal: "Temperatura"

Y ya en el bloque de "Bucle", lo que hacemos es enviar por el puerto serie un texto con el valor de la variable "Temperatura", cuyo contenido irá cambiando cada vez que en el canal se publique un dato nuevo. Como en la práctica anterior se publicaba un dato cada 16 segundos, aquí he puesto el mismo tiempo en el bloque "Ejecutar cada…" pero podéis poner el que queráis porque a la hora de leer datos del canal no iniciamos nosotros la comunicación como en la práctica de publicar, sino que es el servidor el que inicia la comunicación enviando un mensaje cada vez que se recibe un dato en el canal, y por tanto no tenemos el límite de los 15 segundos por saturación del servidor. Asi que podeis enviar el dato al puerto serie cada segundo y si publicais datos nuevos en el canal con la App que os he recomendado para el móvil, vereis que la temperatura se actualiza instantáneamente.



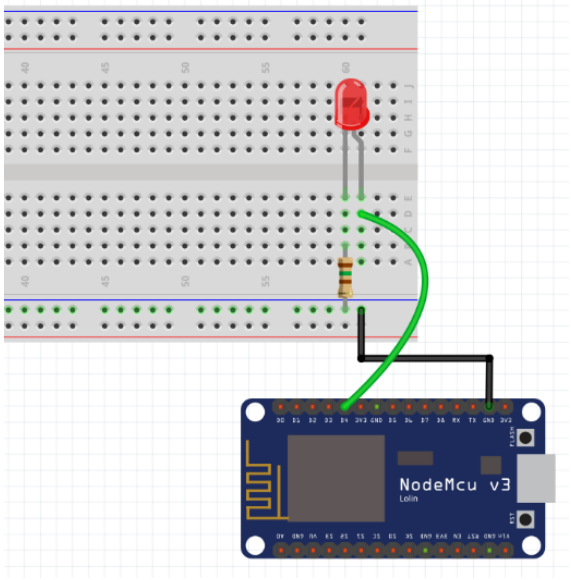
La salida serál algo como esto (aquí he usado dos NodeMCU, una enviando y otra recibiendo, peo podeis enviar datos con la app citada al principio y ver los datos como llegan en la consola serie):

4 Prácticas de programación IoT con Adafruit

4.1 Control de NodeMCU a distancia.

Enciende o apaga un led con la tarjeta NodeMCU, enviando la orden a distancia desde la plataforma IoT de Adafruit.

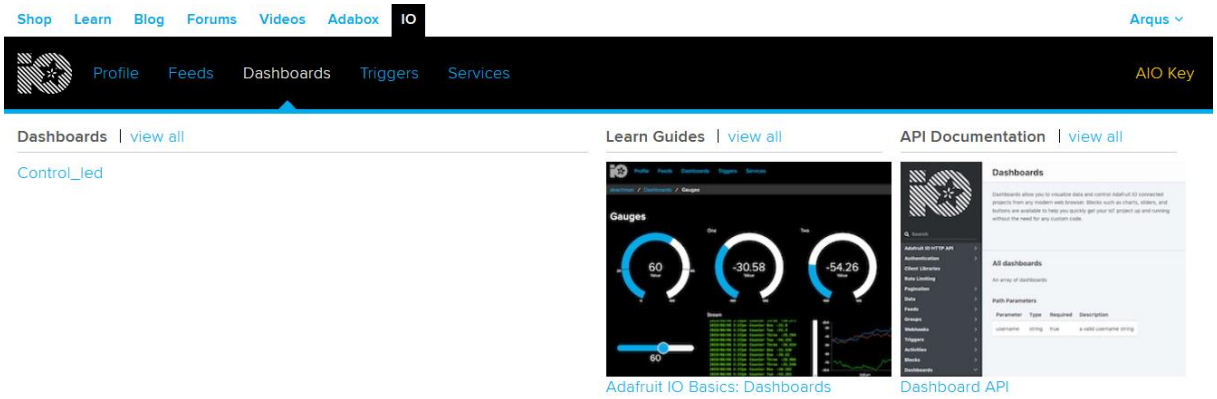
Esquema de conexión:



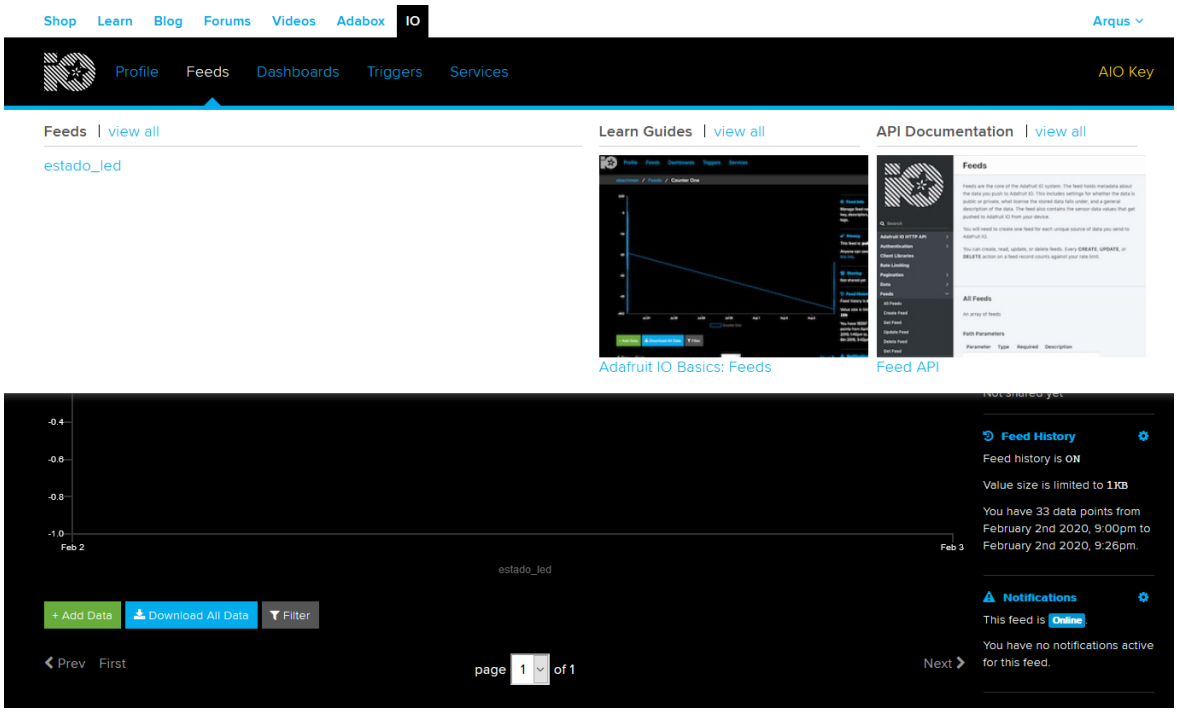
Conectaremos un led con su correspondiente resistencia de polarización al pin D4.

Solución propuesta:

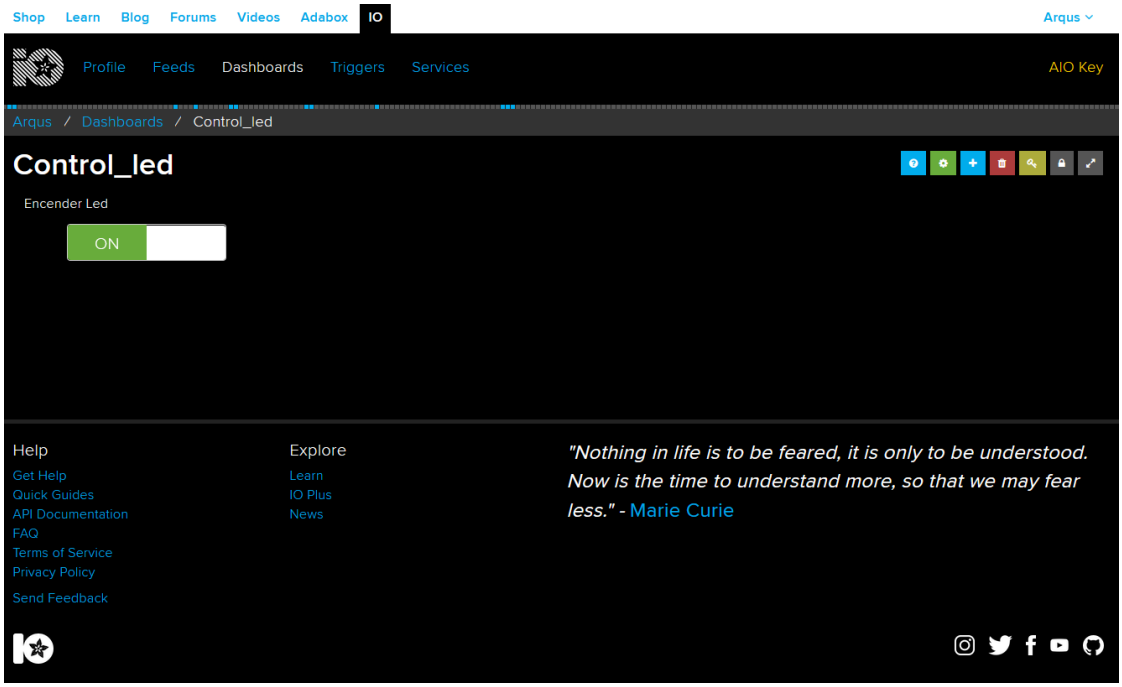
Previamente nos crearemos una cuenta gratuita (si no la tenemos ya) registrándonos en io.adafruit.com. Crearemos un "**Dashboard"** o Panel que es el lugar donde mostramos la información almacenada en nuestra cuenta (lo que en ThingSpeak era un canal). En la imagen inferior se observa que la he llamado "Control_led".



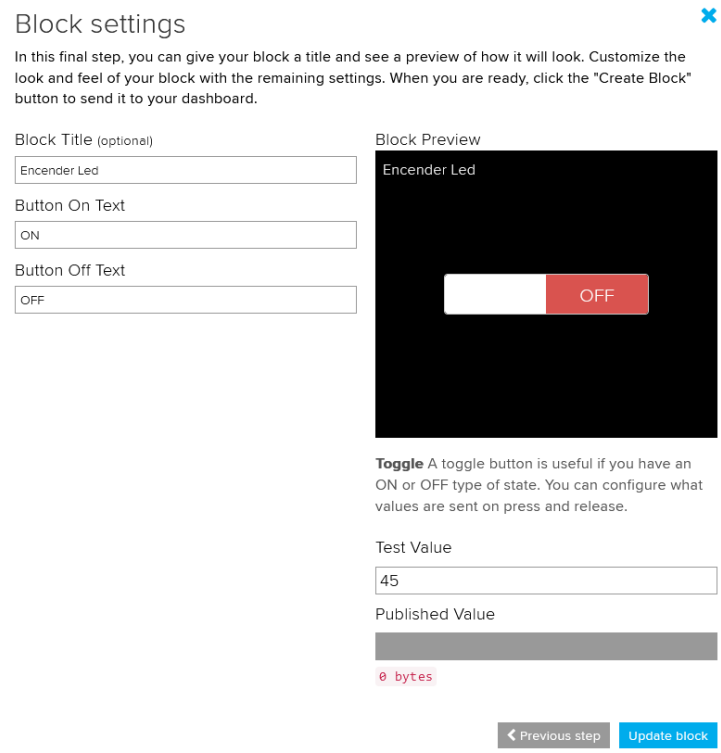
También crearemos un **"Feed"** que es el campo donde guardamos datos (en ThingSpeak se les denomina Field, y en las comunicaciones MQTT Topic o tema), que en este ejemplo he llamado "estado_led".



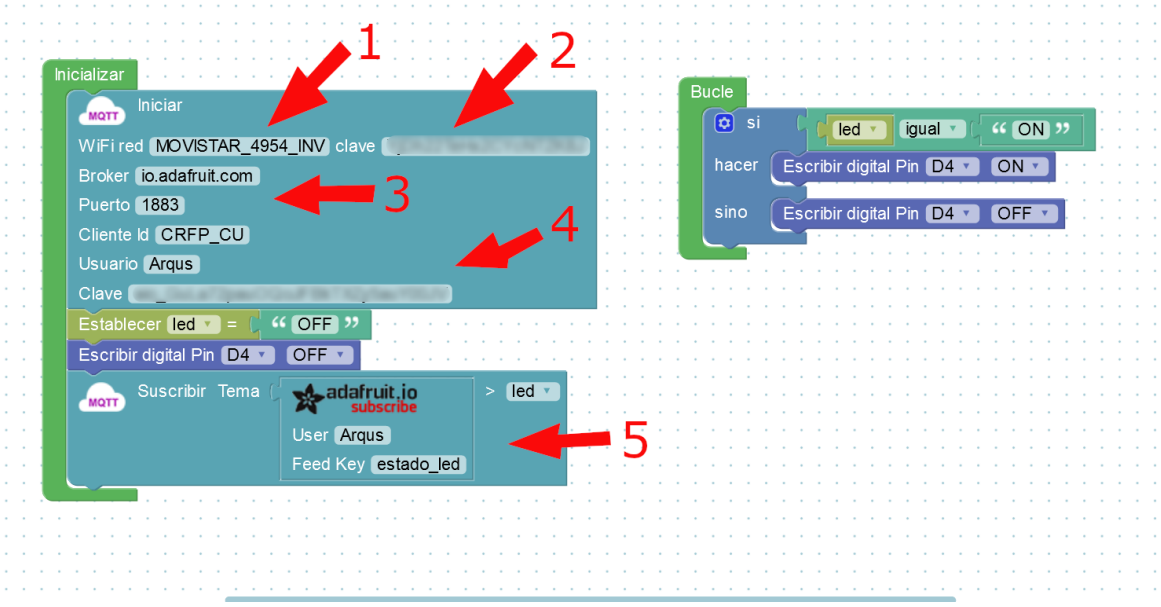
Dentro del panel o dashboard hemos insertado un elemento gráfico que en Adafruit se llaman "**Bloques",** para mostrar el valor guardado en el Feed (o campo) "estado_led". En la imagen se observa que es del tipo interruptor.



El interruptor lo configuramos con un nombre (Encender Led), un texto para la posición de encendido (ON) y un texto para la posición de apagado (OFF). Estos textos serán los valores a guardar en el campo (Feed) asociado.



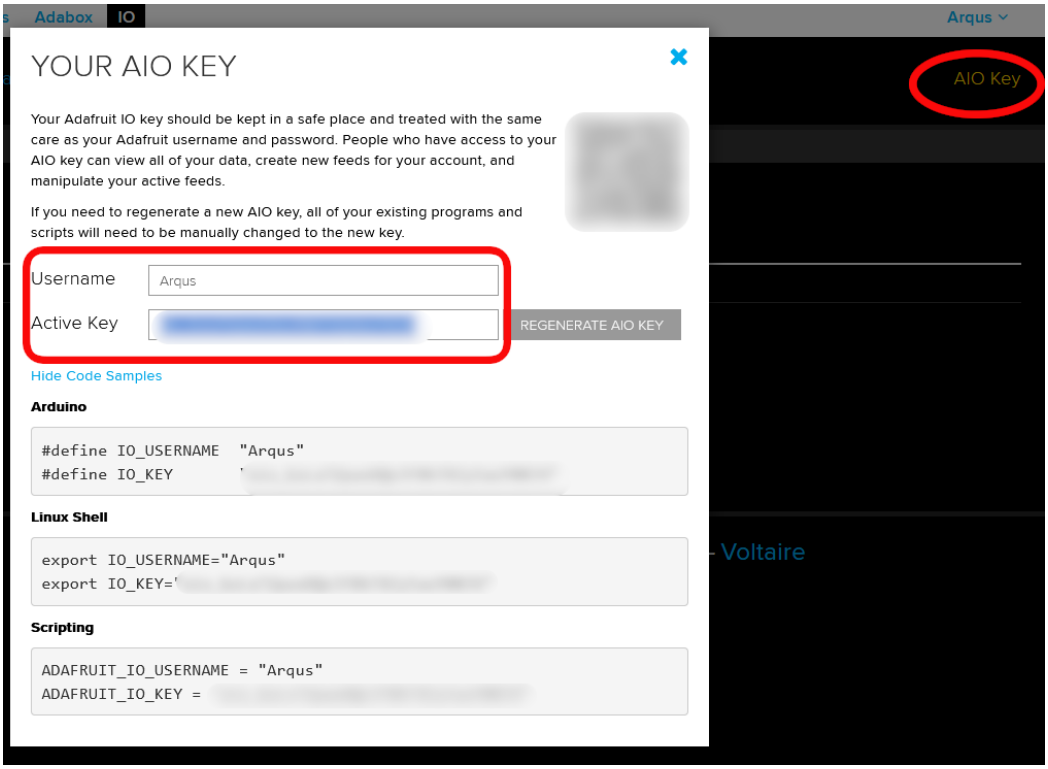
A continuación se muestran los bloques a usar en ArduinoBlocks:



Los datos a introducir en los bloques son los siguientes:

- Nombre de la red Wifi a la que conectaremos la tarjeta NodeMCU
- Contraseña de la red Wifi
- Datos del servidor: io.adafruit.com por el puerto 1883
- Datos de la cuenta en Adafruit. El Cliente ID es indiferente, porque en Adafruit no existe, asi que nos podemos inventar el texto que queramos (igual que en ThingSpeak que lo habíamos dejado en blanco). El usuario y contraseña lo encontramos en el io

de Adafruit pinchando en “AIO key”. La contraseña aparece bajo el campo “Active Key”.

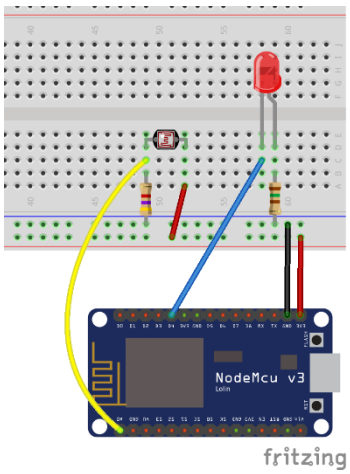


- Los datos para el módulo de subscripción de Adafruit son el nombre de usuario de la cuenta de Adafruit (el mismo que el del punto 4) y el nombre del Feed o campo donde almacenamos los datos.

4.2 Control de NodeMCU a distancia y recepción de su estado

Modificar el ejercicio anterior colocando una LDR que reciba la luz del led, y que la tarjeta NodeMCU envíe al servidor de Adafruit su valor para así tener constancia de cómo está el led después de usar el interruptor del dashboard.

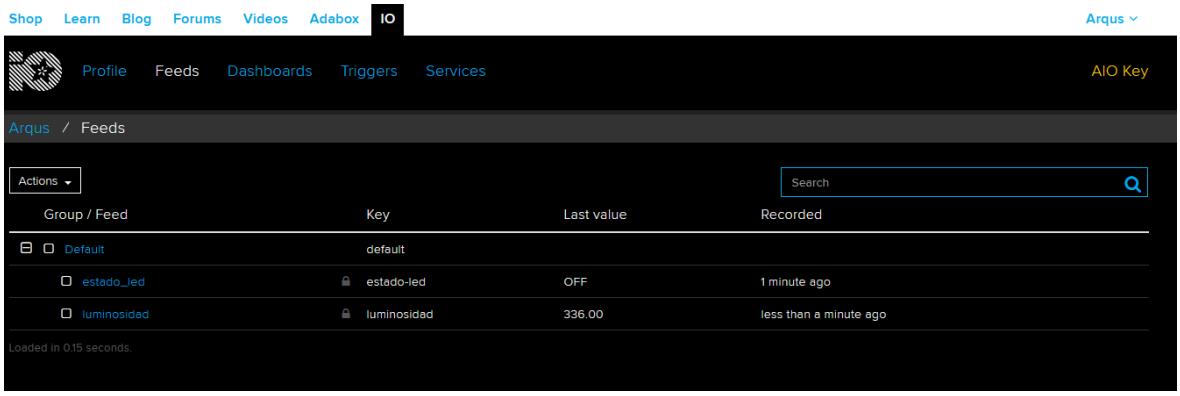
Esquema de conexión:



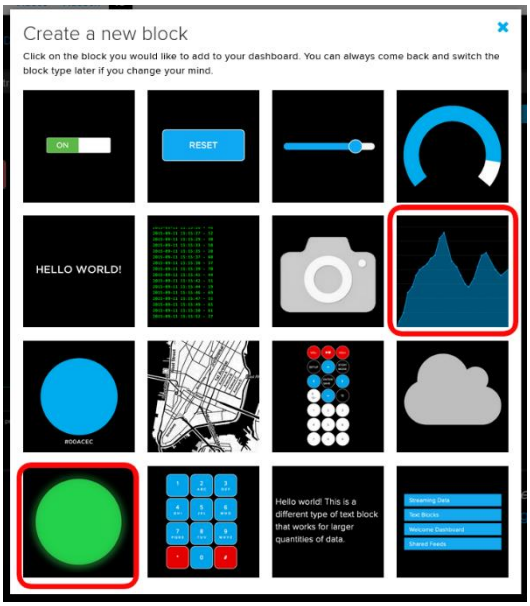
Conectamos el led al pin D4, y la LDR al pin A0. Es muy importante colocar la LDR en frente del LED para que capte la máxima cantidad de luz de éste. Doblad 90º las patillas si es necesario del LED y la LDR para que se miren mutuamente.

Solución propuesta:

Usaremos el dashboard que creamos en la anterior práctica en el servidor IO de Adafruit. Lo que haremos ahora será crear un nuevo campo para almacenar los datos de luminosidad recibidos por la LDR. Ese nuevo campo o "Feed" lo he llamado luminosidad como se puede apreciar en la siguiente imagen.

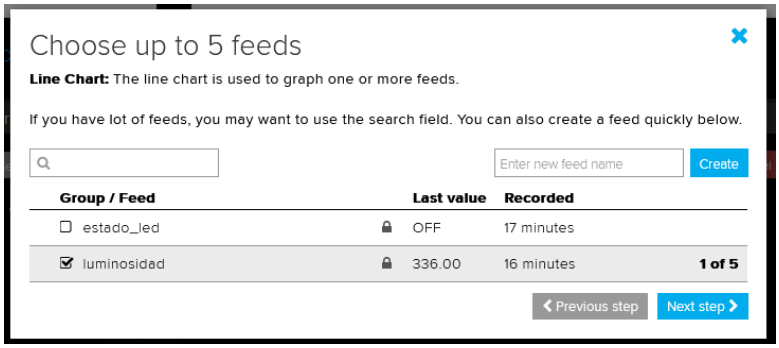


Incluiremos un nuevo bloque en el tablero (o dashboard) para representar estos valores, y lo más vistoso es una gráfica, donde veamos cómo va cambiando el valor con el tiempo a medida que se enciende o se apaga el led. En Adafruit este bloque se llama "line chart".

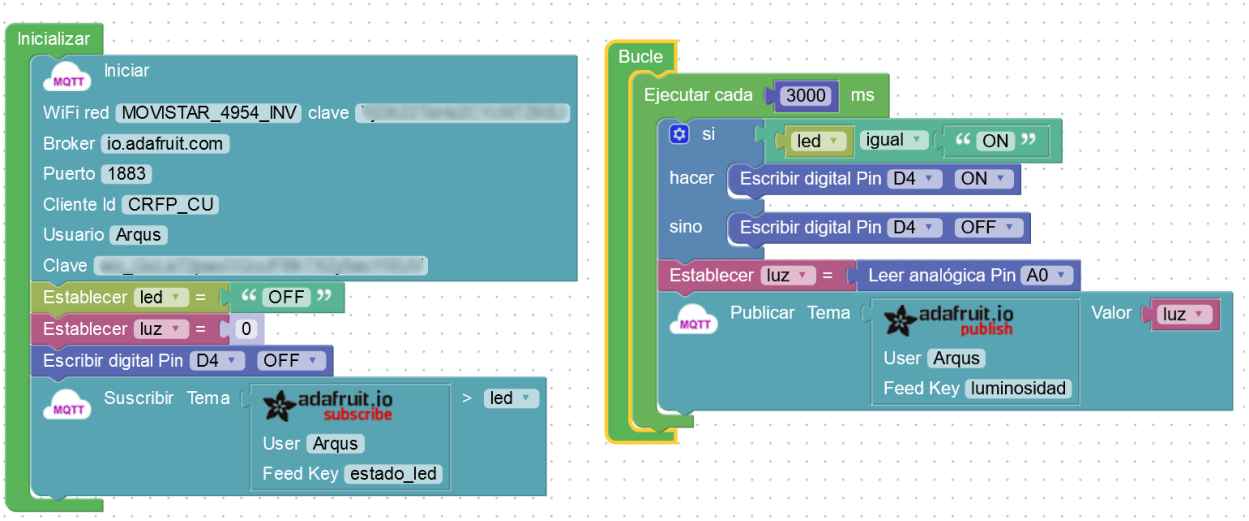


También vamos a añadir otro bloque que nos indique cuando el led está apagado o encendido, el llamado "indicator". En la imagen de al lado se muestran estos dos nuevos bloques a incluir en el tablero.

Ambos bloques van a trabajar con los valores del "Feed" luminosidad, pero de momento asociemos solo el gráfico a este Feed, pulsando en el botón del engranaje del tablero (editar el tablero) y luego en el engranaje del gráfico para elegir el Feed al que va asociado (ver imagen de abajo)



Pasemos ahora a crear el programa en ArduinoBlocks para nuestra tarjeta NodeMCU.



Como se ve en la imagen, iniciamos la comunicación como siempre, poniendo el nombre de la red Wifi y su contraseña, el nombre del servidor y su puerto, el Cliente ID (que no sirve para nada en Adafruit), nuestro usuario en la cuenta de Adafruit y su contraseña.

A continuación creamos dos variables, una de texto ("led") que contendrá el valor del Feed "estado_led" al que nos vamos a suscribir (recibimos el dato de Adafruit), y otra numérica ("luz") con el valor leído por el pin A0 procedente de la LDR. A la variable led le ponemos el valor inicial "OFF" para que empiece con el led apagado si no se reciben datos de Adafruit porque no se actúe sobre el interruptor del dashboard. También apagamos el led con el bloque de "escribir digital" aunque es redundante, se va a apagar en cuanto empiece el bloque de "Bucle".

Y por último en el bloque de "Inicializar" ponemos la suscripción al tema ("Feed") de "estado_led" del servidor de Adafruit con los bloques correspondientes y que el valor recibido se guarde en la variable de texto "led".

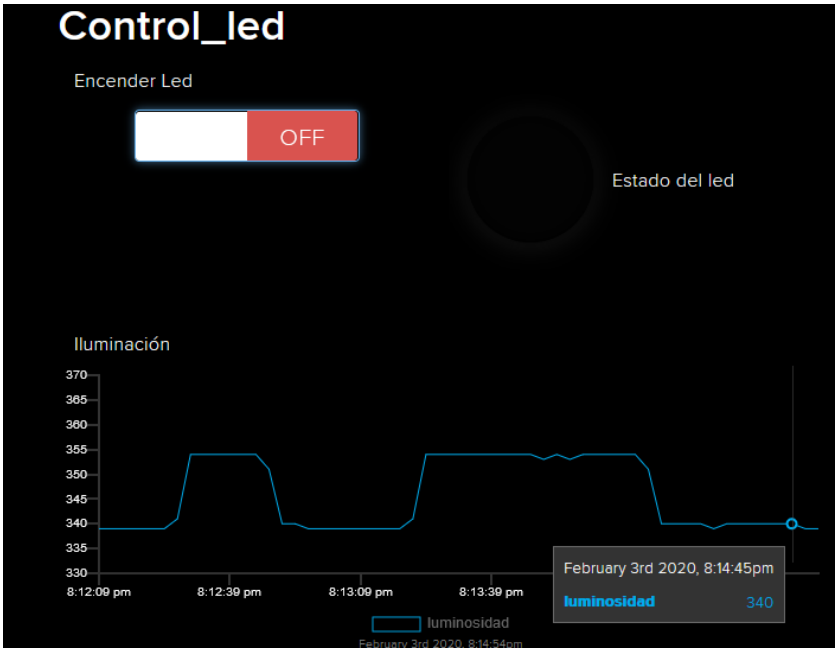
Pasamos ahora al bloque de "Bucle". Vamos a ejecutar cada 3 segundos la actualización del valor a enviar al led físico (apagarlo o encenderlo) y el envío de la luz medida por la LDR. El tema de los 3 segundos viene porque en las cuentas gratuitas de Adafruit estamos limitados al envío de datos cada 2 segundos como mínimo.

El encendido o apagado del led se decide con un bloque "Si" en función de la variable "led" que habrá sido escrita si se ha recibido de Adafruit un mensaje del cambio en el Feed correspondiente. Esta parte podría ir fuera del bloque de "Ejecutar cada 3 segundos", porque el límite lo tenemos al publicar, no al recibir datos por suscripción.

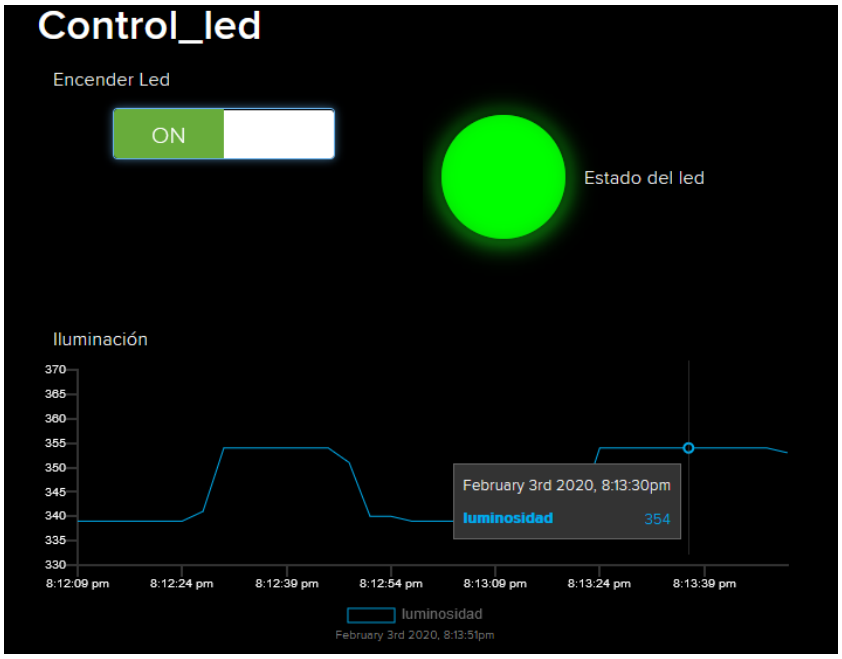
La publicación del valor de la LDR se hace de forma similar a la suscripción, se indica el usuario de la cuenta, el Feed a publicar, y la variable que contiene el valor a enviar a Adafruit.

Una vez construidos los bloques del programa y pasados a la tarjeta, esta se pondrá a enviar datos de la luz recibida por la LDR y podremos ver como aparecen en la gráfica del dashboard de Adafruit.

Los valores enviados dependerán del estado del led, del tipo de LDR, del valor del resistor que tiene en serie, y también de la iluminación de la habitación donde se encuentra la tarjeta. Si miramos estos valores con el led apagado, en mi caso, de noche con la lámpara de la habitación encendida, recibo valores cercanos a 340, como se aprecia en la imagen.

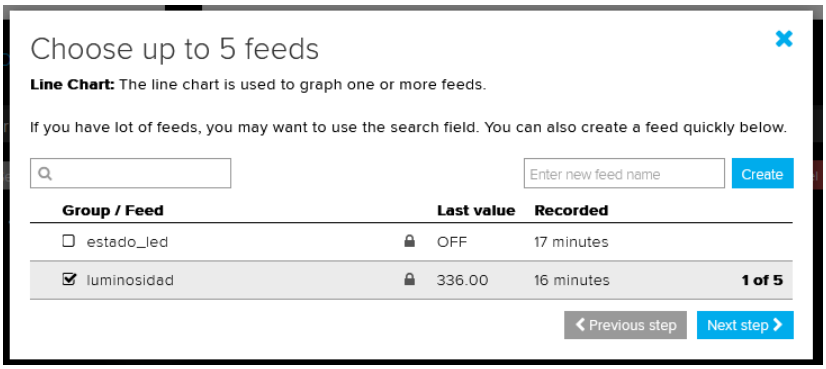


En cambio, si actúo sobre el interruptor del tablero, y se enciende el led del montaje en la placa de prototipos, la LDR recibe algo más de luz (la lámpara de la habitación más el led) y el valor sube a valores cercanos a 354. La variación en mi caso no es mucha porque la luz ambiente es mucho más potente que el led, pero si apagamos la lámpara, o hacemos la práctica de día, o usamos un led de alta luminosidad, se apreciará mayor diferencia.



En mi caso, como los valores son aproximadamente 340 y 354 voy a fijar como valor límite que me sirva para discriminar si el led está encendido o apagado el valor intermedio de 350. Así que ahora vamos a configurar el bloque del "indicador" para que si el led está encendido en la placa, se encienda el "indicador" en el tablero, y si el led está apagado, el "indicador" se encuentre también apagado.

Le damos al botón del engranaje del tablero para su edición, y después al icono del engranaje en el bloque del "indicador". Seleccionamos el Feed al que lo vamos a asociar:

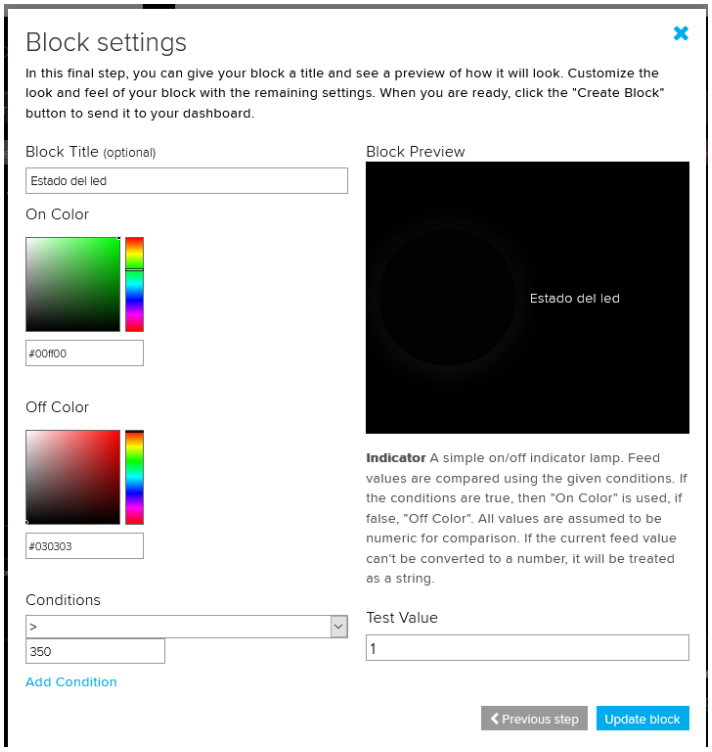


Y le damos al botón "Next step" para seguir con la configuración.

Como se aprecia en la imagen de abajo, he puesto un título para el bloque, he seleccionado un tipo de verde para el estado de encendido, y un color cercano al negro para el estado de apagado. Como condición para el funcionamiento he puesto ">" y e valor "350". De esta forma si el valor del Feed luminosidad es mayor que 350 el indicador se verá en color verde, y si es menor o igual a 350 se "verá" de color negro, es decir, no lo apreciaremos sobre el fondo del dashboard.

De esta manera, con el mismo envío de datos para el Feed de la gráfica estamos haciendo funcionar el indicador.

Observaréis que al actuar sobre el interruptor del tablero se enciende o apaga casi de inmediato el led en la placa de prototipos, y unos segundos después aparece reflejado su estado en el indicador del tablero. El retardo se debe que nuestra placa envía los datos cada 3 segundos, pero tenemos una realimentación a nuestra orden de lo que ocurre en la tarjeta a distancia, tanto en la gráfica como en el indicador.

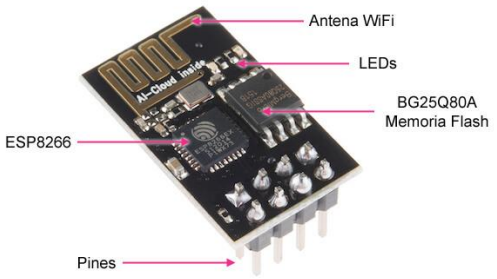


5 IoT con Arduino + ESP-01

El curso IoT que estamos realizando está pensado para ser usado con la tarjeta NodeMCU. Se eligió esta tarjeta frente a Arduino por ser una solución más económica, más potente (microcontrolador Tensilica L106 vs ATMega 328P) y más sencilla (sin adaptadores).

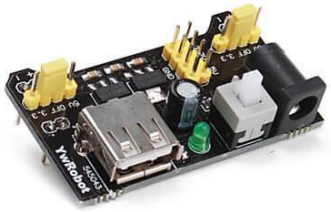
Sin embargo es muy probable que los asistentes a este curso cuenten con una tarjeta Arduino Uno, y probablemente tengan también el módulo ESP-01 para darle conectividad a internet. El módulo ESP-01 contiene en su interior el mismo chip que la nodeMCU, el ESP8266.

Por ello he decidido incluir un apartado al uso de Arduino Uno + ESP-01 para su uso en proyectos IoT, por si alguien no tiene la tarjeta NodeMCU.

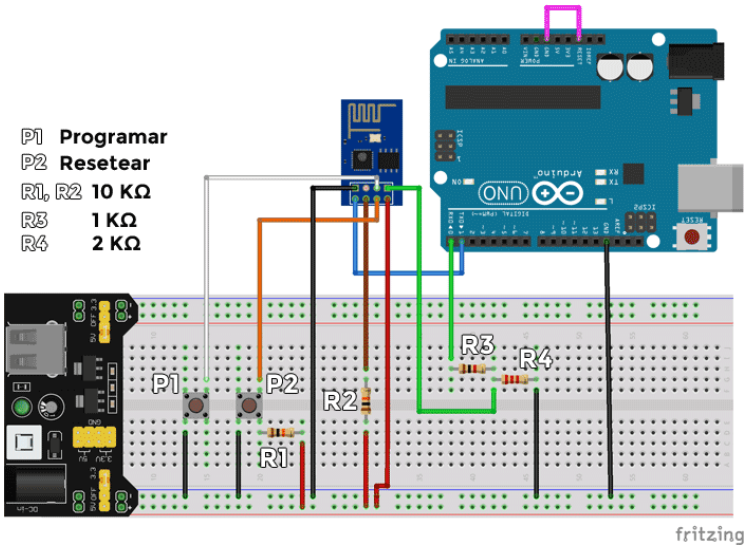


Lo primero que hay que tener claro es que mientras con la tarjeta NodeMCU estamos programando directamente el chip ESP8266, usando el microcontrolador que tiene este chip, el Tensilica L106, si usamos Arduino + módulo ESP-01, es el microcontrolador de Arduino el que programaremos, y el ESP8266 solo se encarga de gestionar la comunicación Wifi para Arduino. Esto significa que el microcontrolador que estamos usando es menos potente, pues el de Arduino es de 8 bits, mientras el del ESP8266 es de 32 bits, aparte que el de Arduino va a una velocidad como mínimo 10 veces más lenta, por lo que su uso en comunicaciones más exigentes hará que el sistema se resienta frente al uso directo de la NodeMCU.

También has de saber que no podemos conectar directamente el módulo ESP-01 a Arduino, pues Arduino trabaja con 5V, y el chip ESP8266 funciona con 3,3V. Alguno pensará que como Arduino tiene un pin de 3,3V, tenemos el problema resuelto, pero no es así. El regulador de tensión de 3,3V de Arduino solo puede entregar 50 mA, y el módulo ESP-01 necesita 200 mA en algunos momentos, por lo que podemos sufrir cortes e inestabilidades en la comunicación. La solución sería usar una fuente de alimentación como por ejemplo las tarjetas MB102 que se acoplan sobre la placa de prototipos, dan tanto 5V como 3,3V y pueden entregar 800 mA sin despeinarse.



Pero tenemos otro problema adicional. La comunicación entre Arduino y el módulo ESP-01 se realiza a través del puerto serie, es decir los famosos pines RX y TX (0 y 1 respectivamente), que funcionan a 5V, y esos valores hay que convertirlos a 3,3V usando por ejemplo un divisor de tensión con unos resistores.



Para solucionar todos esos problemas (fuente MB102 y resistores colocados en la protoboard) es mucho más práctico comprar un adaptador para el módulo ESP-01, que es una tarjeta con un conector para los 8 pines de que consta el ESP-01, y que una vez montada la ESP-01 en este adaptador, tenemos en el borde solo 4 pines para conectar con Arduino: Vcc, GND, Tx y Rx. Lo bueno de este adaptador es que se conecta directamente a Arduino, ya que admite los 5V, pues en el adaptador ya va integrado un regulador de 3,3V para alimentar al ESP-01, y el coste de esta tarjeta es de 1 a 2€.

Con lo cual, comprando un ESP-01 (6€) y un adaptador (2€), tenemos conectividad Wifi para nuestro Arduino a bajo precio, aunque si sumamos el precio de la tarjeta Arduino, es mucho más caro que comprar una NodeMCU donde tenemos todo en uno.

5.1 Reprogramación del ESP-01 para funcionar a 9600bps

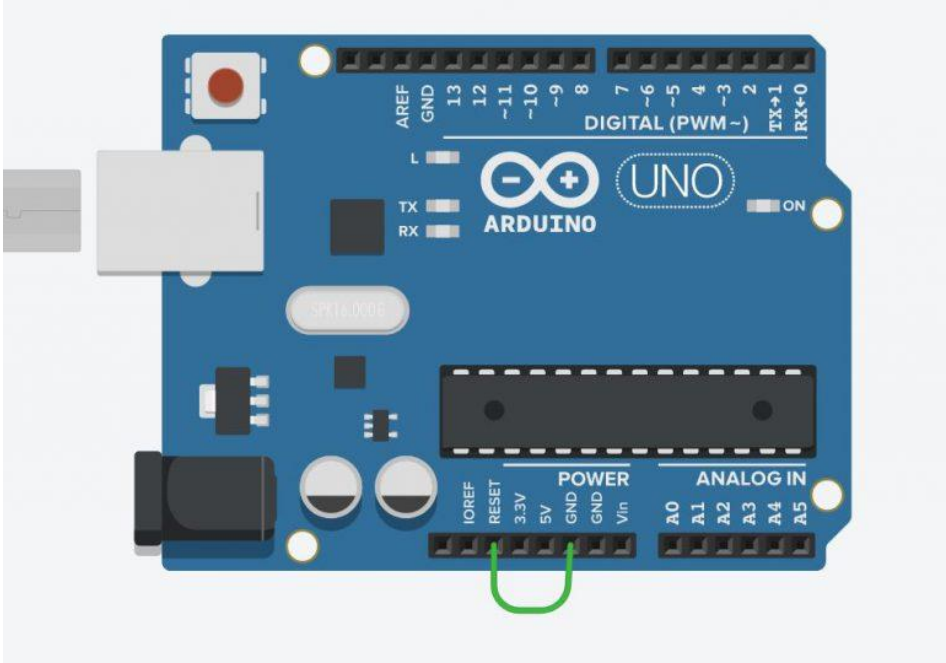
A la hora de usar el módulo ESP-01 con Arduino para tener comunicación via Wifi, parece lógico pensar que debemos conectar el módulo ESP-01 con Arduino a través de los pines Rx y Tx. Esto funciona, pero entonces tenemos ocupados los dos pines del puerto serie de Arduino con el módulo y no podemos usarlo para enviar información vía el cable USB hacia el ordenador, es decir, no podemos programar Arduino. Eso implica que cada vez que pasemos un programa a la tarjeta Arduino tengamos que desconectar el módulo ESP-01, y posteriormente con el sketch ya cargado volverlo a poner.

Pero hay otro inconveniente. Si queremos mostrar información de Arduino hacia el ordenador usando el puerto serie, no podemos porque lo tenemos conectado al módulo ESP-01. La solución es bien fácil, usar los pines Rx y Tx de Arduino para el cable USB (comunicación con el ordenador) y conectar el módulo ESP-01 a otros pines, por ejemplo el 2 y el 3, creando a través de software la simulación de un segundo puerto serie para comunicarnos por esos dos pines con el módulo ESP-01. Esa simulación es muy sencilla, solo hay que incluir la librería SoftwareSerial.h y tendremos funciones para enviar y recibir datos similares a las del puerto serie nativo de Arduino (begin, print, read…)

Pero antes hay que salvar un pequeño escollo. Los módulos ESP-01 suelen venir configurados de fábrica para comunicarse a 115200bps y si usamos el puerto serie por software (el de la librería SoftwareSerial) no podemos pasar de 9600bps porque la gestión por software de las comunicaciones es más lenta que con el hardware real. Asi que tenemos que reprogramar la tarjeta ESP-01 a 9600bps.

Para ellos vamos a usar Arduino como intermediario. Conectaremos Arduino al ordenador via USB, y conectaremos el módulo ESP-01 a Arduino por los pines 0 y 1. De esta forma los datos que enviaremos al ESP-01 para reprogramarlo irán a 115200bps. Pero como tanto el cable USB (ordenador) como el ESP-01 están en los mismos pines, lo que haremos será anular a Arduino para que los datos pasen directamente del cable USB al ESP-01. Para ello mantendremos a Arduino durmiendo, es decir que su microcontrolador estará totalmente desactivado y no se le ocurrirá interferir en la comunicación intentando controlar el puerto serie. Y eso lo

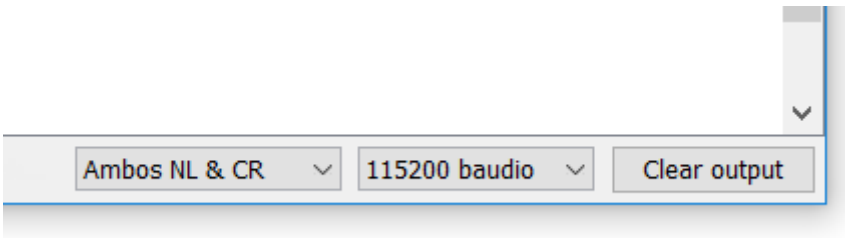
conseguiremos poniendo a Arduino en Reset de forma permanente. **Asi que colocamos un cable entre el pin GND y el pin Reset.**



Ahora ya colocamos el adaptador del ESP-01 a Arduino de la siguiente forma:

- Pin Rx_adaptador -> pin Rx Arduino
- Pin Tx_adaptador -> pin Tx Arduino
- Pin Vcc_adaptador -> pin 5V Arduino
- Pin GND_adaptador -> pin GND Arduino

Ahora conectamos Arduino con el ordenador via USB, y en el IDE de Arduino abrimos el monitor serie, y lo configuramos a 115200bps y seleccionamos “Ambos NL & CR”.



La configuración se hace con comandos AT, similar a la programación de los módulos de Bluetooth vistos en el curso de Arduino nivel medio.

Si escribimos el comando “**AT**” (en mayúsculas y sin las comillas) y le damos a enviar, el módulo nos debe responder con un OK si todo ha ido bien.

Ahora escribimos y enviamos el comando:

AT+UART_DEF=9600,8,1,0,0

Este comando le dice al módulo que se ponga a 9600bps (y otras cosas como nº bits para datos, bit de parada…)

Ahora cambiamos el monitor serie a 9600bps, enviamos el comando AT y si nos responde con OK, lo hemos conseguido, ya está reprogramado el ESP-01 a 9600bps.

5.2 Comunicación bidireccional entre Arduino y un móvil

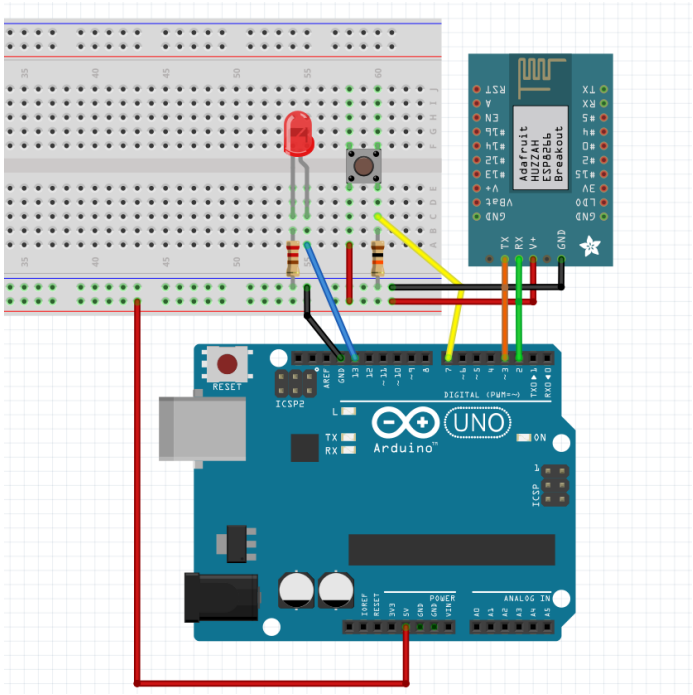
En este ejemplo vamos a crear un proyecto en ArduinoBlocks que nos permita enviar datos desde Arduino hacia un dispositivo móvil, como nuestro teléfono smartphone, y a la inversa, que desde el teléfono enviemos órdenes a Arduino. Todo esto a distancia, que es la filosofía del IoT: poder ver datos o controlar dispositivos desde cualquier parte del mundo de forma inalámbrica a través de internet.

Lo que haremos en este ejemplo será conectar a Arduino un led, que encenderemos o apagaremos desde el móvil.

Y a la inversa, pondremos un pulsador en Arduino, y Arduino enviará la información de cómo se encuentra el pulsador (presionado o sin pulsar) al móvil. Pensad que dependiendo del sensor usado podemos enviar cualquier otro tipo de información, como por ejemplo temperatura, presión atmosférica, presencia de alguien, un escape de gas butano, la humedad de nuestras plantas… Las posibilidades son enormes.

Además del led y el pulsador, hay que conectar a Arduino el ESP-01 a través de su adaptador a los pines 2 y 3 por ejemplo, y así tenemos libres el 0 y el 1 para programar Arduino con ArduinoBlocks sin necesidad de andar desconectando el módulo ESP-01.

Por tanto el esquema de conexiones será el siguiente:



Se observa el led conectado al pin 13 y el pulsador conectado al pin 7.

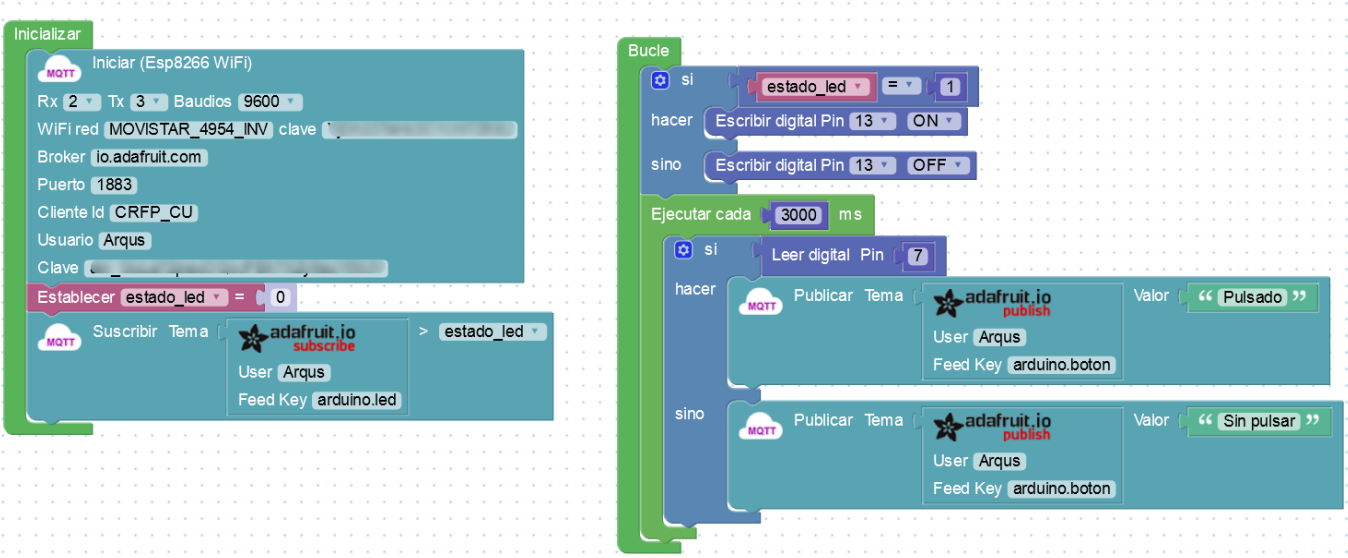
Ahora pasamos a la programación de la tarjeta Arduino:

- Como el estado del pulsador se va a enviar al móvil, será Arduino quien envíe esa información a un servidor MQTT. Por tanto el estado del pulsador será **publicado** en un **topic** por Arduino. Y el móvil lo que hará será **subscribirse** a ese topic.
- En el led el proceso es a la inversa. Es el móvil el que debe enviar la orden a Arduino. Por tanto es el móvil el que **publicará** un **topic**, y Arduino que es el que recibe el dato de la orden lo que hará es una **subscripción**.

Por tanto tendremos dos topics o campos creados en el servidor MQTT que usemos. Uno de ellos lo publicará Arduino y el otro lo publicará el móvil. Asi que fijándonos en la parte que le toca a Arduino, Arduino publicará un topic (el del pulsador) y se subscribe a otro topic (el del led).

Vamos a usar como servidor gratuito MQTT Adafruit, pues permite publicar datos cada 2 segundos (en ese sentido es mejor que ThingSpeak) pero hay varios servidores gratuitos disponibles en internet (incluso podemos montarnos uno nosotros mismos como por ejemplo con la aplicación Mosquito).

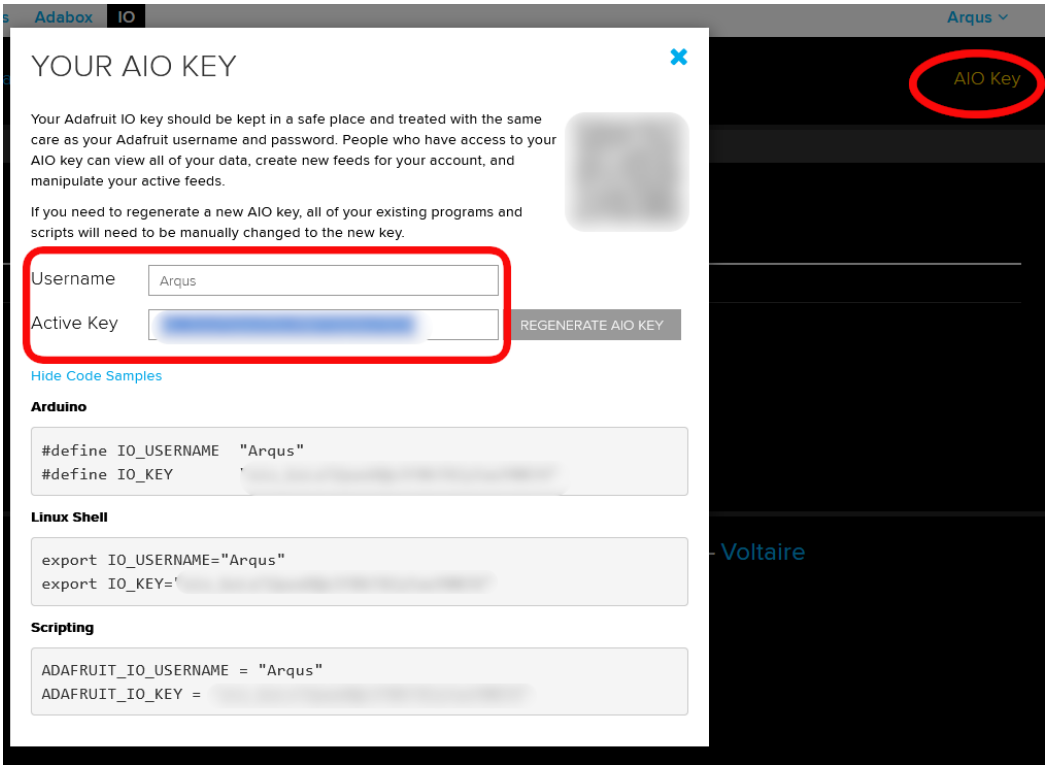
Se muestra a continuación los bloques a usar en ArduinoBlocks para programar la tarjeta Arduino:



En el bloque de inicialización hemos puesto 3 bloques: iniciar la conexión al servidor MQTT de Adafruit, crear una variable que recoja del servidor MQTT el valor del led, y subscribirse al topic del led.

Veamos primero el bloque de iniciar conexión con Adafruit. El bloque lo hemos cogido de la categoría MQTT, y se llama “Iniciar (Esp8266 Wifi)”. En este bloque hemos rellenado sus campos para configurarlo:

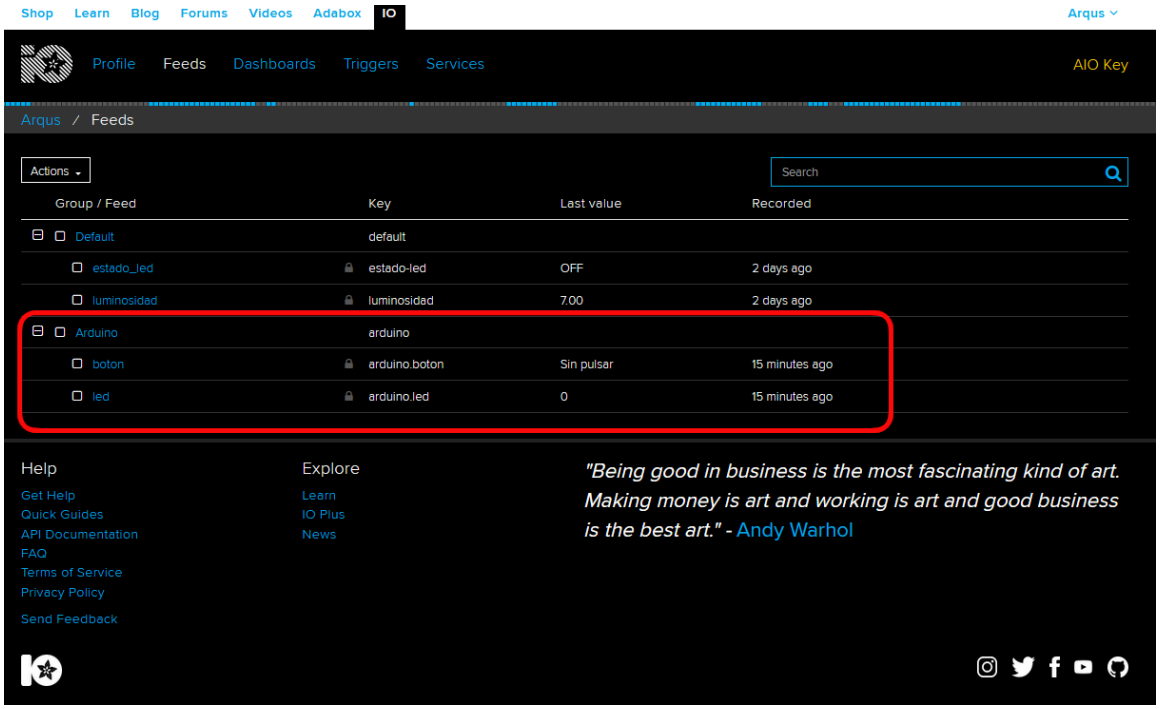
- Pines y velocidad: el pin 2 de Arduino al Rx del adaptador del ESP-01, el pin 3 al Tx y la velocidad a 9600bps que es la que habíamos dejado configurada con los comandos AT.
- Nombre de la Wifi y contraseña: la de la Wifi a donde se va a conectar el ESP-01
- Servidor (bróker): la dirección es “io.adafruit.com” y el puerto 1883 que es el que tiene Adafruit configurado para recibir las comunicaciones.
- Cliente ID: nos inventamos un nombre, en realidad no se usa en Adafruit.
- Usuario y clave: estos datos son los de nuestra cuenta en Adafruit y los encontramos pulsando en el menú “AIO Key”



Ya que estamos en la cuenta de Adafruit, vamos a aprovechar y creamos los topic necesarios, uno para el estado del led, y otro para el estado del botón.

Recuerda que en Adafruit los topic se llaman “feeds”. Yo los he llamado “led” y “botón”, y para organizarme mejor, los he creado dentro de un grupo llamado Arduino, y así los tengo separados de los que habíamos utilizado en las prácticas 4.1 y 4.2 de este documento.

Observa que esos topic aparte del nombre tienen una key, y que son “arduino.led” y “Arduino.boton”. Estos van a ser los datos que necesitaré para programar el bloque de ArduinoBlocks y la aplicación del móvil.



Bien, ya hemos terminado en Adafruit, volvamos a ArduinoBlocks. Después del bloque de iniciar la comunicación con el sevidor de Adafruit, hemos puesto la creación de una variable de tipo numérico llamada “estado_led”.

Y a continuación hemos puesto el bloque de subscripción al topic del led. Como la subscripción solo hay que hacerla una vez, la hemos puesto dentro del bloque “Inicializar” en vez de “Bucle”. En la subscripción hemos usado el bloque de subscripciones de Adafruit que se encarga de crear la cadena de texto correcta para que la entienda Adafruit. En ese bloque hemos puesto el nombre de usuario de la cuenta de Adafruit (el mismo que usamos al iniciar la comunicación en el bloque de Iniciar MQTT), y el nombre del topic a subscribir. He puesto como topic su key: arduino.led, pero también es válido el nombre (“led”). El valor recibido de la subscripción a ese topic se guardará en la variable “estado_led”.

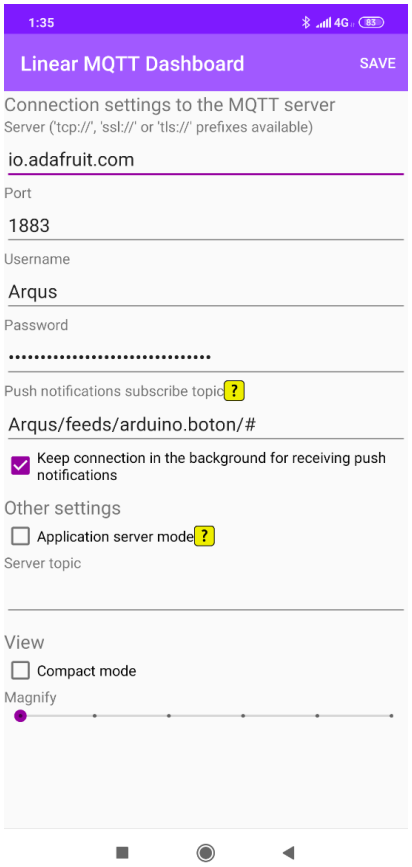
Este topic del led va a ser enviado por el móvil, y configuraremos la app para que envíe un 1 (encender led) o un 0 (apagar led).

Pasamos al bloque “Bucle”. Empezamos comprobando con un bloque “Si” (categoría Lógica) el valor de la variable. Si es un 1 activamos el pin digital 13, que es donde está el led conectado a Arduino, y si no lo apagamos.

Ahora pasamos a ver cómo está el pulsador (leer pin 7 de Arduino) y mandamos su estado al topic “botón”. Como solo podemos publicar cada 2 segundos si no queremos que el servidor de Adafruit nos banee por exceso de tráfico, hemos puesto el bloque de tiempo “Ejecutar cada 3000ms”. **Recuerda que no es válido poner el bloque “Esperar XXX milisegundos” porque interrumpe la ejecución de todos los procesos y se cortaría la comunicación con el servidor MQTT**. Al publicar tenemos que poner nuestro usuario de Adafruit (igual que al subscribir) y el topic (feed) a publicar. Yo he vuelto a usar en vez del nombre del feed su key: arduino.boton. Y lo que publica es un texto: “Pulsado” si el botón está presionado, y “Sin pulsar” si el botón no se

utiliza. Como veis los topic no tienen por qué guardar números, también pueden contener textos.

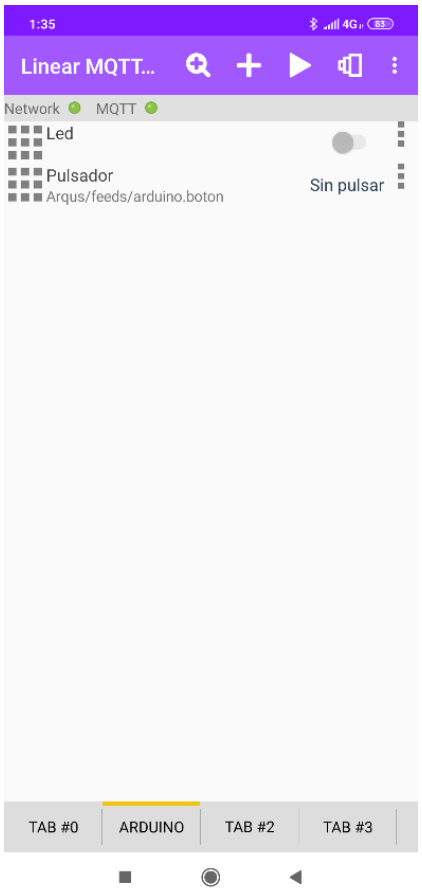
Y con esto ya hemos terminado la programación de Arduino. Cargamos el programa en la tarjeta con el ArduinoBlocks-connector y pasamos a configurar la app del móvil.



La app que vamos a usar es la Linear MQTT Dashboard, que se ve en el módulo 6 del curso.

Configuramos los datos del servidor: io.adafruit.com con el puerto 1883. El nombre de usuario es el de nuestra cuenta de Adafruit (lo hemos usado en los bloques de ArduinoBlocks también), y la clave es la "Active Key" que podíamos consultar en el menú AIO Key de nuestra cuenta en Adafruit.

Ponemos también el topic (feed) al que vamos a subscribir el móvil, el del botón. La cadena de texto que usa Adafruit para ello tiene el siguiente formato: "usuario/feeds/key_del_topic". Se puede cambiar la key del topic por su nombre (el nombre del feed). También se permite cambiar la palabra feeds por la letra f, cosa que viene bien en dispositivos IoT que no dispongan de mucha memoria y necesiten que las cabeceras de los datos a enviar sean más cortas.



A continuación en una de las 4 Tabs de la app creamos dos elementos:

- Un widget de tipo "switch" (interruptor) para indicar el valor del led. Lo he llamado Led.
- Un widget de tipo "value" para mostrar un valor. Lo he llamado Pulsador porque será donde se muestre el estado del pulsador de Arduino.

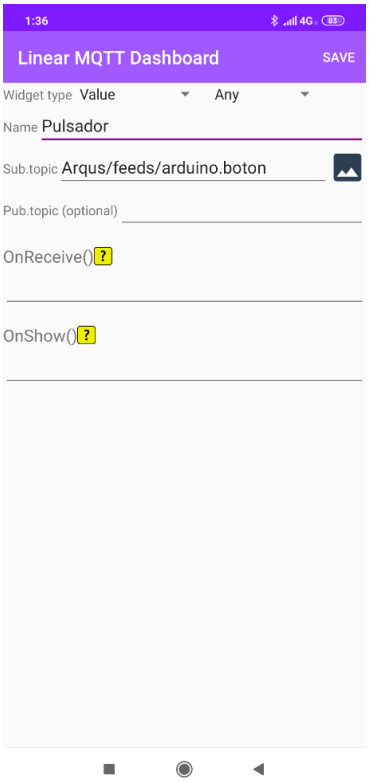
El interruptor lo vamos a asociar con el topic "led". Cuando lo accionemos en el móvil, la app Linear MQTT Dashboard debe publicar su estado en el feed de Adafruit, y como Arduino está subscrito a ese feed, recibirá de inmediato su valor, y actuará en consecuencia sobre el led real que hay en el montaje de la placa de prototipos.

Veamos como hemos configurado el interruptor.

Como se puede apreciar, he puesto que al activarlo envíe un 1 y si está desactivado que envíe un 0, tal como dije cuando estaba explicando la configuración de la subscripción en los bloques de ArduinoBlocks.

La cadena de texto para la publicación por parte de la app del móvil es como dije "usuario/feeds/key_del_feed" por lo tanto he puesto Arqus/feed/arduino.led

Y si pasamos a ver como está la configuración del widget del valor que recogerá el topic del pulsador publicado por Arduino… Veremos que la cadena es Arqus/feeds/Arduino.boton

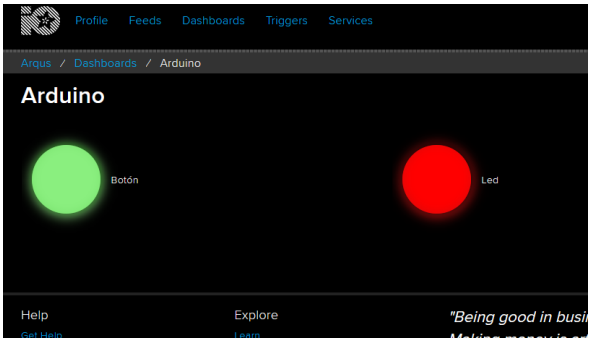


Fijarse bien que ahora la cadena de texto no está en la línea Pub (publicar) sino en la línea Sub (subscribir).

Con todo eso ya hemos terminado el proyecto. Ponemos en marcha Arduino, y la app Linear MQTT Dashboard, y veremos que cuando pulsamos el botón, aparece en la app la palabra "Pulsado" donde está el Widget Pulsador, y si lo soltamos pondrá "Sin pulsar" que eran los textos que enviaba Arduino al topic boton.

También podemos actuar sobre el interruptor que hemos puesto en la app y comprobaremos que se enciende o se apaga en Arduino ¡por control remoto sin que haya conexión por cable!

Por último he querido hacer un tablero o dashboard en adafruit para mostrar como están ambos elementos (led y pulsador), con lo cual no solo lo vemos a distancia con el móvil sino que también lo podemos ver desde cualquier navegador de cualquier dispositivo como un ordenador conectados a nuestra cuenta de Adafruit.



6 Bibliografía

http://arduinoblocks.didactronica.com/

Blog de proyectos en ArduinoBlocks

http://kio4.com/arduino/117G_Wemos_IOT_Bloques.htm

Blog con proyectos de Arduino y App Inventor

https://es.mathworks.com/help/thingspeak/index.html

Información sobre los protocolos de ThingSpeak

7 Créditos y licencia

Autor: Javier García Real. Publicado por el Centro Regional de Formación del Profesorado de Castilla-La Mancha.

http://centroformacionprofesorado.castillalamancha.es/comunidad/crfp