Tic tac, tic tac

Ahora que ya hemos conectado nuestro Arduino a internet, iremos un paso más allá y conectaremos nuestro Arduino a IoT Cloud para crear una alarma que funcione con el reloj interno de nuestro Arduino. Esta alarma lo que hará es encender un LED virtual cuando sea la hora que nosotras decidamos. No vamos a necesitar material adicional únicamente nuestro NANO 33 IoT y el cable USB para conectarlo a nuestro ordenador.

RTC: el reloj de nuestro Arduino

Un reloj en tiempo real es un reloj que lleva la cuenta de la hora actual y puede utilizarse para programar acciones específicas en un momento determinado. El término **reloj en tiempo real (RTC)** se utiliza para evitar la confusión con los **relojes de hardware ordinarios**, que son sólo señales que gobiernan la electrónica digital, y no son usados para contar en horas, minutos y segundos, sino que se utilizan para el funcionamiento interno del propio componente electrónico.

La mayoría de los RTC (Real Time Clock) utilizan un oscilador de cristal cuya frecuencia es de 32,768 KHz (la misma frecuencia utilizada en los relojes de cuarzo). Además, el RTC sigue funcionando en modo de reposo, por lo que se puede utilizar para despertar el dispositivo de los modos de reposo de forma programada. Cada vez que se alimenta la placa, el RTC se resetea y comienza desde la fecha estándar. Para mantener la hora y el RTC en funcionamiento es necesario mantener el dispositivo alimentado.

Para utilizar este reloj, usaremos **la librería RTCZero**. Esta librería nos permite controlar y utilizar el RTC interno en todas las placas Arduino con la arquitectura SAMD (como la que tiene nuestro NANO 33 IoT). La librería se encarga de la inicialización del RTC interno e incluye varias funciones que utilizaremos en esta práctica.

> Para saber más sobre esta librería, haz click aquí.

Primer paso: Creamos nuestra primera Thing

Para poder conectar a nuestro NANO 33 IoT, tendremos que crear un componente virtual llamado 'Thing'. Estas 'cosas' son las que permiten que nuestros proyectos se comuniquen con IoT Cloud. Como dicen en la imagen siguiente, estas Thing pueden también comunicarse entre ellas, aunque por el momento, nosotros solo crearemos una de ellas. Para crearla, iremos al menú horizontal superior (Things, Dashboards, Devices, Integrations and Templates) y haremos click donde pone **Things**.

Copyright 2026

image-1665055346267.18.27.png

Renombramos nuestra thing como **Nano Alarm**, aunque puedes escoger otro nombre si lo prefieres, te aconsejo que sea descriptivo, haciendo click sobre el rectángulo en el que pone **Untitled**:

image-1665055416726.18.41 copia.png

Una vez hemos cambiado el nombre, el siguiente paso será vincular un dispositivo a nuestra Thing. En este punto, conectaremos nuestro Arduino al ordenador por **USB.** Ahora, haremos click en el apartado **Associated Device** y ahí en **Select Device.**

image-1665055483548.18.41 copia 2.png

Como ya lo habíamos asociado antes, nos debería aparecer esta pantalla, en la que IoT Cloud ya lo reconoce, si no, seguiremos los pasos que nos indique y que serán parecidos a los que ya seguimos:

image-1665055511230.19.08.png

Segundo paso: creamos nuestras variables

Cuando hayamos configurado nuestro NANO 33 IoT, nos aparecerá en la columna de la derecha, como en la imagen inferior.

> Aunque en esta imagen aparezca online, por el momento **no te preocupes** si el tuyo aparece **offline**.

Para añadir nuestras variables, haremos click en **ADD VARIABLE.** Esto abrirá una nueva ventana, donde podemos elegir el nombre, el tipo de datos, la configuración de permisos y la configuración de actualización.

image-1665055900397.19.17 copia.png

Nos aparecerá la siguiente ventana:

image-1665055988275.32.35.png

En ella introduciremos la información de nuestra primera variable:
Name: **alarm_state**
Select variable type: **boolean**
Variable Permission: **Read & Write**

Copyright 2026

Variable Update Policy: **On change**

image-1665056132088.34.40.png

Le daremos a **ADD VARIABLE** y nos aparecerá así:

image-1665056261340.37.03.png

Haremos lo mismo con las siguientes variables:

Nombre	Tipo de datos	Permiso de la variable	Uso
alarm_state	boolean	Read & Write	Activa o desactiva la alarma
hours_alarm	int	Read & Write	Establece la hora de la alarma
minutes_alarm	int	Read & Write	Establece los minutos de la alarma
seconds_alarm	int	Read & Write	Establece los segundos de la alarma
led	boolean	Read Only	Indica si la alarma está activada o no

> El tipo de datos **int** nos aparecerá como **Integer Numbe**r:

image-1665056843240.39.18 copia.png

Una vez hayamos creado todas las variables, nos aparecerán como en la siguiente imagen:

image-1665057437084.56.57.png

> El orden en que nos aparezcan puede variar, pero eso no es problema. Lo importante es que alarm_state y led tengan valores boolean y que tanto hours_alarm, minutes_alarm y seconds_alarm tengan valores integer.

Tercer paso: introducimos las credenciales de nuestro Wi-Fi

No debemos olvidar introducir las credenciales de nuestra red. Ese proceso ya te lo he explicado en este apartado.

Una vez hayamos introducido el nombre y contraseña de nuestro Wi-Fi, haremos click sobre el apartado **Dashboards,** justo a la derecha de **Thing**.

Cuarto paso: creamos nuestro dashboard

Una vez hagamos click sobre **Dashboards**, veremos la siguiente pantalla, en la que pulsaremos sobre **BUILD DASHBOARD**:

image-1665057185882.51.34.png

Es el momento de cambiar el nombre de nuestro dashboard "**Untitled**" por **Nano Alarm** y empezar a añadir los diferentes widgets que necesitaremos:

image-1665095405976.54.06.png

Para ello hacemos clic en el botón **ADD,** en la esquina superior izquierda y nos aparecerá esta ventana emergente:

image-1665138945590.54.51.png

El dashboard necesita cinco widgets en total, cada uno de ellos aparece en la siguiente tabla.

Widget	Variable a la que se conecta
value	hours_alarm
value	minutes_alarm
value	seconds_alarm
switch	alarm_state
led	led

Una vez que seleccionemos cada uno de los widgets, necesitaremos **vincular cada widget a una de las variables** que ya creamos en el paso anterior. Revisa la tabla de arriba para conocer las correspondencias entre widgets y variables.

image-1665139294703.38.56.png

Una vez hayamos creado todos los widgets, podemos organizarlos como queramos pulsando el botón de las flechas de la parte superior izquierda, pero realmente, si los has creado en orden, deberían tener este aspecto, que está bastante organizado:

image-1665139716202.29.21.png

Quinto paso: creamos el código

Copyright 2026

Todas las configuraciones que hemos hecho hasta ahora, han sido básicamente compiladas en un sketch especial. Este sketch contiene información sobre el dispositivo que estamos utilizando, un ID único de nuestra Thing, las variables que hemos creado y la información de red que hemos introducido. Todo lo que tenemos que hacer ahora, es añadir algo de código al sketch para crear la alarma.

Le daremos a **Things** en el menu superior horizontal y nos aparecerá Thing llamada "Nano Alarm". Ahí, hacemos clic en la pestaña **Sketch**. Aquí, deberíamos ver un boceto generado automáticamente. Este boceto es prácticamente un "caparazón" para el código que añadiremos a continuación, lo que significa que aún no tiene ninguna funcionalidad.

Lo primero que haremos será añadir las librerías que necesitaremos y crear un objeto llamado alarma, del tipo RTCZero:

```
#include "thingProperties.h"
#include "RTCZero.h"


RTCZero alarm;
```

Después, necesitaremos iniciar la comunicación a través del puerto serial en 9600 baudios, como ya hicimos en la práctica del acelerómetro:

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
```

Para asegurarnos de que nuestro Arduino se conecta a IoT Cloud, le pediremos que nos imprima la información sobre esa conexión con printDebugInfo(). Seguidamente inicializaremos nuestro objeto alarm, que hemos creado un poco antes y apagaremos nuestro LED dándole el valor false:

```
ArduinoCloud.printDebugInfo();
  alarm.begin();
  led = false;
}
```

Ahora, pasaremos al la función loop(). Después de actualizar la información enviada a IoT Cloud con la función update(), imprimiremos por el puerto serie la hora, minutos y segundos reales:

```
void loop() {
  ArduinoCloud.update();
```

```
  Serial.print(alarm.getHours());
  Serial.print(":");
  Serial.print(alarm.getMinutes());
  Serial.print(":");
  Serial.println(alarm.getSeconds());
}
```

Ahora, veremos que después del loop() hay algunas funciones vacías. Estas funciones han sido creadas automáticamente desde la nube y no deben ser eliminadas o el sketch no compilará. Estas funciones se crean cuando añadimos una nueva variable con permiso de lectura y escritura, y son llamadas cada vez que cambiamos el valor de la variable desde IoT Cloud. Así, para establecer una nueva hora de alarma cada vez que cambiamos el valor desde la nube, tenemos que añadir una función .setAlarmTime() dentro de las funciones que estaban vacías:

```
void onHoursAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}

void onMinutesAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}

void onSecondsAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}
```

La función onAlarmStateChange() es llamada cada vez que encendamos o apaguemos nuestra alarma desde el interruptor (Switch) que hemos creado antes en nuestra Dashboard. Activaremos la alarma si el interruptor está en ON (true) y lo desactivaremos si lo colocamos en OFF (false). Aquí, necesitaremos un condicional para ver si el interruptor está encendido o apagado.

Si la alarma está encendida, usaremos las funciones **.setAlarmTime()**, **.enableAlarm()** y . **attachInterrupt()**, que ya vienen programadas y vinculadas a nuestro objeto alarm, para activarla.

Si el interruptor está apagado, usaremos la función .disableAlarm() para desactivar nuestra alarma y apagaremos nuestro LED:

Copyright 2026

```
void onAlarmStateChange() {
  if (alarm_state == true) {
    alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
    alarm.enableAlarm(alarm.MATCH_HHMMSS);
    alarm.attachInterrupt(alarmMatch);
  }
  else {
    alarm.disableAlarm();
    led = false;
  }
}
```

El último paso es crear una nueva función llamada **.alarmMatch()** que controlará lo que ocurre cuando la alarma coincida con la hora real enviada por nuestro NANO 33 IoT. En este caso, usaremos esta función para encender el LED de la Nube poniendo el valor del LED a true:

```
void alarmMatch() {
  led = true;
}
```

Por si te has perdido en algún paso copiando y pegando el código, lo encuentras todo a continuación:

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Nano_Alarm"
  Arduino IoT Cloud Variables description
  The following variables are automatically generated and updated when changes are made to the
Thing
  int hours_alarm;
  int seconds_alarm;
  int minutes_alarm;
  bool alarm_state;
  bool led;
  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/
```

Copyright 2026

```
#include "thingProperties.h"
#include "RTCZero.h"

RTCZero alarm;

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);

  // This delay gives the chance to wait for a Serial Monitor without blocking if none is
found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
     The following function allows you to obtain more information
     related to the state of network and IoT Cloud connection and errors
     the higher number the more granular information you'll get.
     The default is 0 (only errors).
     Maximum is 4
  */

  setDebugMessageLevel(2);

  ArduinoCloud.printDebugInfo();

  alarm.begin();

  led = false;
}
```

Copyright 2026

```
void loop() {
  ArduinoCloud.update();

  Serial.print(alarm.getHours());
  Serial.print(":");
  Serial.print(alarm.getMinutes());
  Serial.print(":");
  Serial.println(alarm.getSeconds());
}

void onHoursAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}

void onMinutesAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}

void onSecondsAlarmChange() {
  alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
}

void onAlarmStateChange() {
  if (alarm_state == true) {
    alarm.setAlarmTime(hours_alarm, minutes_alarm, seconds_alarm);
    alarm.enableAlarm(alarm.MATCH_HHMMSS);
    alarm.attachInterrupt(alarmMatch);
  }
  else {
    alarm.disableAlarm();
    led = false;
  }
}

void alarmMatch() {
```

Copyright 2026

```
  led = true;
}
```

Una vez lo tengamos en nuestro sketch, llegará el momento de verificarlo, haciendo click sobre el icono del tick y posteriormente subirlo a nuestro Arduino haciendo click sobre la flecha hacia la derecha:

image-1665163975375.04.50 copia.png

Una vez lo hayamos subido, veremos algo parecido a esto:

image-1665164989181.gif

Si te has fijado, la hora en mi ordenador son las 14:50 y no las 12:50 cuando la alarma se enciende. Eso es porque mi Arduino daba la hora con 2h de retraso. ¿Cómo podemos solucionarlo? Muy fácil, en nuestro código he modificado una línea, la que aparece en el siguiente bloque en la línea 3, y le he restado 2h a la hora a la que he fijado mi alarma, para que así coincida con la hora de mi Arduino:

```
void onAlarmStateChange() {
  if (alarm_state == true) {
    alarm.setAlarmTime(hours_alarm-2, minutes_alarm, seconds_alarm);
    alarm.enableAlarm(alarm.MATCH_HHMMSS);
    alarm.attachInterrupt(alarmMatch);
  }
  else {
    alarm.disableAlarm();
    led = false;
  }
}
```

> Para saber la hora de nuestro Arduino, abviamente con nuestro Arduino conectado, tendremos que abrir el puerto serie. Para ello, haremos clic en la lupa que se ve en la siguiente imagen:

Captura de pantalla 2023-01-27 a las 21.53.07.png

Una vez sepamos la hora de nuestro Arduino, podremos escribirla correctamente en nuestra Alarma.

Ahora ya funciona perfectamente:

image-1665165906946.gif

> Tenemos que considerar que al ser un servicio conectado a Internet, hay cierta latencia y los valores no se actualizan inmediatamente. Si queremos comprobar que nuestra alarma funciona, es mejor que la pongas con un margen de unos 2 o 3 minutos.

Problema frecuente

Puede que nuestro Arduino nos aparezca offline aunque nos permita subir el código.
1. Elimina tu Arduino completamente de **Devices** y vuélvelo a instalar.
2. Como IoT Cloud es un servicio que vamos a utilizar en su versión gratuita, puede presentar latencias y problemas de conexión si hay mucha gente utilizándolo. En mi experiencia personal, he tenido más problemas tratando de conectarme por las mañanas que por las tardes.
3. Para realizar pruebas, te aconsejo que te crees dos cuentas en Arduino IoT Cloud, utilizando dos cuentas de email diferentes, para poder alternar tus proyectos entre ellas. Puede ocurrir que tu Arduino te aparezca offline en una de tus cuentas y online en la otra.

¿Qué tenemos que entregar?

Graba un video similar al .gif que aparece en el apartado anterior, que se vea la alarma apagada y que cuando marca la hora se encienda el LED.

Para enviarlo puedes mandar el video en formato gif o subirlo a Youtube o a Drive, configurarlo como oculto solo para aquellos que tengan el enlace, y compartir el enlace en esta tarea.

FUENTES:
Alarma con Arduino 33 IoT: https://docs.arduino.cc/tutorials/nano-33-iot/iot-cloud

Financiado por el Ministerio de Educación y Formación Profesional y por la Unión Europea - NextGenerationEU

logo.png

Revision #25
Created 2022-09-15 13:38:47 CEST by Marta P. Campos
Updated 2023-01-27 22:00:38 CET by Marta P. Campos

Copyright 2026