

Contents

Introduction.....	3
Arduino IoT Cloud.....	3
Things	4
Networks	5
Devices.....	5
Variables	6
Sketches	7
Dashboards.....	8
Getting Started with the Arduino IoT Cloud	8
Cloud Plans	8
Install the Arduino Create Agent	9
Cloud + Arduino Nano 33 IoT	10
Nano 33 IoT Hookup	10
Making your First Thing & Dashboard	11
Network Settings.....	13
Device Settings	14
Create a Cloud Variable.....	15
Adding Code	18
Modifying the Sketch.....	19
Build a Dashboard.....	20
Testing your First Thing	23
Using the Mobile App.....	23
Adding the Temperature & Humidity Sensor.....	24
Temperature & Humidity Code Changes & Library	25

Temperature & Humidity Dashboard Changes.....	27
Cloud + Arduino Nano RP2040 Connect	29
Nano RP2040 Connect Hookup (x2)	29
Thing Setup – Device 1.....	30
Coding – Device 1	31
Thing Setup – Device 2.....	32
Coding – Device 2	33
Testing the Devices	34
Cloud + ESP32.....	35
ESP32 Hookup.....	35
Build an ESP32 Thing	36
ESP32 Coding.....	37
ESP32 Dashboard	38
Conclusion	39

Introduction

The Internet of Things has opened the door to a number of great projects for makers. Being able to build remote sensors and devices is both entertaining and useful.

Allowing IoT devices from different manufacturers to interact with your own creations can really open up a number of useful applications, some that you may not have previously considered building.

There are many ways to build IoT devices, some of them more complex than others. Many experimenters make use of cloud platforms such as Blynk, or “roll their own” solutions with protocols like MQTT and development applications like IBM Node-Red.

Today we will look at another way of putting together IoT applications using microcontrollers and the Arduino IoT Cloud.

Arduino IoT Cloud

On February 6th, 2019 Arduino announced the public beta of the Arduino IoT Cloud.



The Arduino IoT Cloud was, in many ways, similar to the existing Blynk product. It allowed you to create IoT applications for Arduino microcontrollers and connect them to a cloud service, to control them or to interface them with existing IoT devices.

The initial implementation of the cloud was a bit limited in the number of Arduino boards it supported. It had both a free and paid tier, and made use of the existing Arduino online sketch editor.

At the end of 2020 Arduino made a number of improvements to the IoT Cloud, and changed its account structure. The free tier remained, and was enhanced with the ability to use selected third-party boards. Other features like API and data retention were reserved for the paid tiers, of which there are now three.

The Arduino IoT Cloud allows you to leverage your existing Arduino expertise and create your own IoT devices.

In order to make use of the cloud, you'll first need to understand a bit of its unique terminology.

Things

The basis of all Arduino IoT Cloud projects is the “Thing”.

A Thing is a virtual object that resides within the cloud. It is used to securely hold variables and information regarding connected devices and networks.



You'll build a Thing for every project you create inside the cloud, using an online Thing editor. Your thing will consist of a mixture of Variables, a Device, network information, and a Sketch.

Networks

You will need to configure your Thing with your network connection information. For most devices, this will be your Wi-Fi credentials, your SSID, and password.

Some third-party devices will also require an additional "secret key", which will be generated by the Arduino IoT Cloud editor when the device is added. This secret key is entered along with the network information.

Devices

Devices are simply microcontrollers that connect to the cloud. These can be certain Arduino microcontrollers, as well as a few third-party boards like the ESP32 and ESP8266.

As of this writing (July 2021), the list of supported devices is as follows:

- Arduino Nano RP2040 Connect
- Arduino Nano 33 IoT
- Arduino MKR 1000 Wi-Fi
- Arduino MKR Wi-Fi 1010
- Arduino MKR WAN 1300 (with Arduino PRO Gateway LoRa)
- Arduino MKR WAN 1310 (with Arduino PRO Gateway LoRa)
- Arduino MKR GSM 1400 (with SIM card)
- Arduino MKR NB 1500 (with SIM card)
- Arduino Pro Portents H7
- Expressive ESP8266-based boards
- expressive ESP32-based boards

You'll need to grab a couple of these devices in order to experiment with the cloud. I'll be using the Nano 33 IoT and Nano RP2040 Connect in these experiments, as well as an ESP32 WROVER board.

Variables

Yes, we all know what variables are, or at least I hope we all know that! But in this case, I'm referring to a special type of variable, which I'll refer to as "cloud variables" for lack of a better term.

These variables are the key to getting your IoT applications to work, as they are available to both your attached device and to any other devices or dashboards you are using.

You define your variables when you build your Thing, you also get to choose if they are Read & Write or Read Only, and if they should synchronize with other cloud variables in other Things which you have constructed.

The "cloud variables" have many, many types that you can use to define them. Aside from the "basic" types that we all know, like Booleans and integers, there are also specialized types like Temperature, Velocity, and Luminance.

Before you decide to use one of the newer variable types, I have a bit of a warning for you, however. During my experimenting, I have discovered that these newer types, at least the ones I tried, did not work with the mobile dashboard. The mobile dashboard only seems to work with the “basic” types.

This is quite obviously a bug, and hopefully, it will be fixed soon. Again, I’m writing this at the end of July 2021, so perhaps by the time you read this, you won’t have to follow this restriction.

Sketches

Sketches, as all Arduino users know, is the term used for the C++ programs that we write for the Arduino and ESP32 boards. And you’ll need to create a sketch for the device associated with your Thing.

One great feature of the Arduino IoT Cloud is that when you build a new Thing, the majority of your sketch is written for you automatically. All you’ll need to do is fill in the bits that make the variables you have defined come into action.

You can edit your sketches in two different ways:

- Directly within the Thing editor.
- Using the more advanced Arduino Web Editor.

For basic sketches, either way will work fine. If you want to include libraries and other dependent files in your sketch, then you’ll want to use the Web Editor instead.

All of your sketches will be saved online to your Arduino account. You can also download a copy of your sketch in a ZIP file when you use the Web Editor.

Dashboards

A “dashboard” is a method of controlling your IoT Cloud Thing.

You build a dashboard by dragging widgets onto a working surface. You link the widgets to appropriate cloud variables, and then you label them and (on some of them) set their parameters.

Your dashboard can then be displayed on a web browser. You can also use the Arduino IoT Remote app on your Android or IOS device.

Getting Started with the Arduino IoT Cloud

So now that you have a good idea as to what the Arduino IoT Cloud is and what it can do, you’ll likely want to get started using it. You’ll need to get a few things prepared first.

Cloud Plans

The first thing you will need to do is create an Arduino account and select a cloud plan.

You might already have an existing Arduino account if you’ve purchased from their online store or have participated in their forums.

Either way, head over to the Arduino Login page, where you can either sign in or create a new account. Creating a new account is free.

Once you are in, you’ll want to head over to the Arduino Store and choose a plan. You have four to choose from:

- Free Plan
- Entry Plan
- Maker Plan
- Maker Plus Plan

By default, your account will be set up on the Free Plan, which is sufficient to perform the experiments listed here. However, for more practical projects you'll need to subscribe to one of the higher, paid plans.

Install the Arduino Create Agent

You have one more task to complete before working with the Arduino IoT Cloud. You'll need to install the Arduino Create Agent onto the workstation that you'll be using.

The Arduino Agent is the link between your workstation's USB port (where you attach your microcontroller) and the Arduino IoT Cloud. It's required because the security built into your web browser prevents websites from directly connecting to your computer's resources, which is of course a requirement for the Thing and Web editors.

The agent is available for Linux, Windows and macOS. It's just a file that you download and install in the conventional fashion. The agent will run automatically when you start your workstation, and a small Arduino icon on your taskbar permits you to stop or pause it.

You can also install the agent onto several of your computers and work with the cloud on any of them, in fact, while creating this article and video I did exactly that.

With the agent installed, we are ready to begin working with the Arduino IoT Cloud

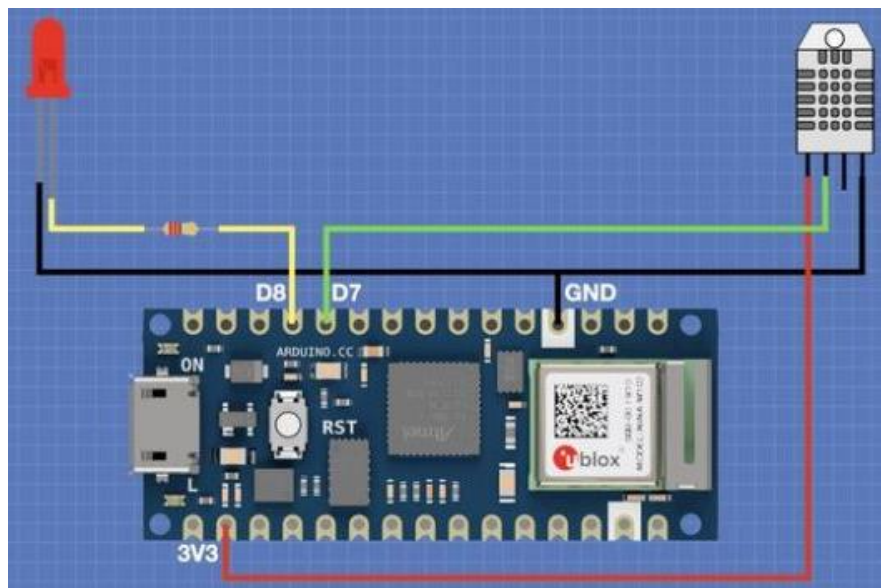
Cloud + Arduino Nano 33 IoT

We are going to use an Arduino Nano 33 IoT at the device for our first cloud experiment, however, any cloud-compatible Arduino will work just as well.

Of course, you will also need to have a cloud account setup, and the Create Agent installed and running on your workstation.

Nano 33 IoT Hookup

Here is how we will be wiring up our Arduino Nano 33 IoT board:



All that we are connecting to the board is an LED and a DHT-22 temperature and humidity sensor. You can use any color of LED that you want, and you could also substitute a less-expensive DHT-11 if you're willing to modify a line of code.

Once you have everything hooked up, connect the Arduino Nano 33 IoT board to your workstation with an appropriate Micro USB cable.

Making your First Thing & Dashboard

We are going to create a “Thing” using the Nano 33 IoT and construct a dashboard to control it online, both with a web browser and using the mobile app.

Our Dashboard will have a switch (or two) to control the LED, and we’ll be able to read the current temperature and humidity as well.

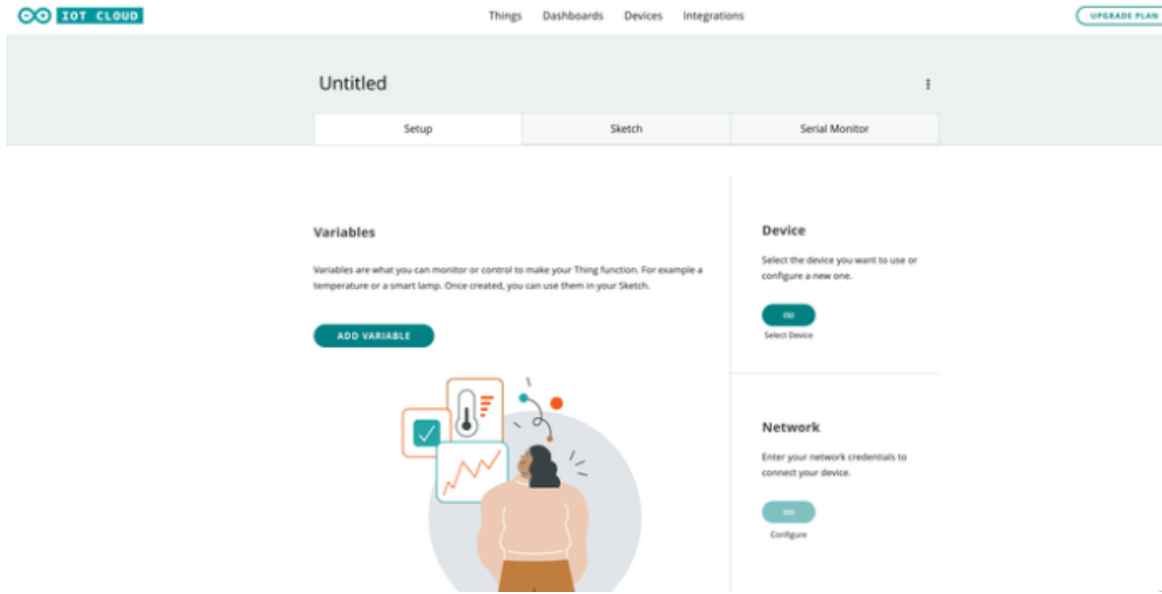
We’ll get started by connecting to the Arduino IoT Cloud in Arduino Create, so open this page up if you don’t have it open already.

When you open up the Arduino IoT Create page for the first time, you’ll see a button inviting you to create your first Thing. If you have already created a Thing, you’ll be directed to the Things page instead, with a list of your existing things and a button to create a Thing.

In addition to the Things page, there are three other pages you can work with, all of them accessible through the menu at the top of the page.

- Dashboards – This allows you to manage and create new Dashboards to control and monitor your Things.
- Devices – These are the microcontrollers that you have attached to the Arduino IoT Cloud.
- Integrations – This is where you manage your API integrations (not available on the Free plan).

We will want to create a Thing, so go to the Things page and click the Create Thing button. The Things workspace will appear.



At the top of the page, you'll see the Thing name, which is "untitled". You can type over this field to give your Thing a proper name.

There are three main areas on the Thing creation page:

- Variables – The cloud variables your thing will use to exchange data.
- Device – The microcontroller that you are attaching to this thing.
- Network – The network credentials for your device.

Let's fill them out.

Network Settings

Your Network settings are simply your Wi-Fi SSID (network name) and Password. If you haven't entered them yet, you'll have a button enabled to do so.

Configure network

×

Your will find these network parameters in the secret tab in your sketch, and your device will be able to connect to the network once the sketch will be uploaded.

Name (SSID) *

DBWSNET

Password *

.....

👁

SAVE

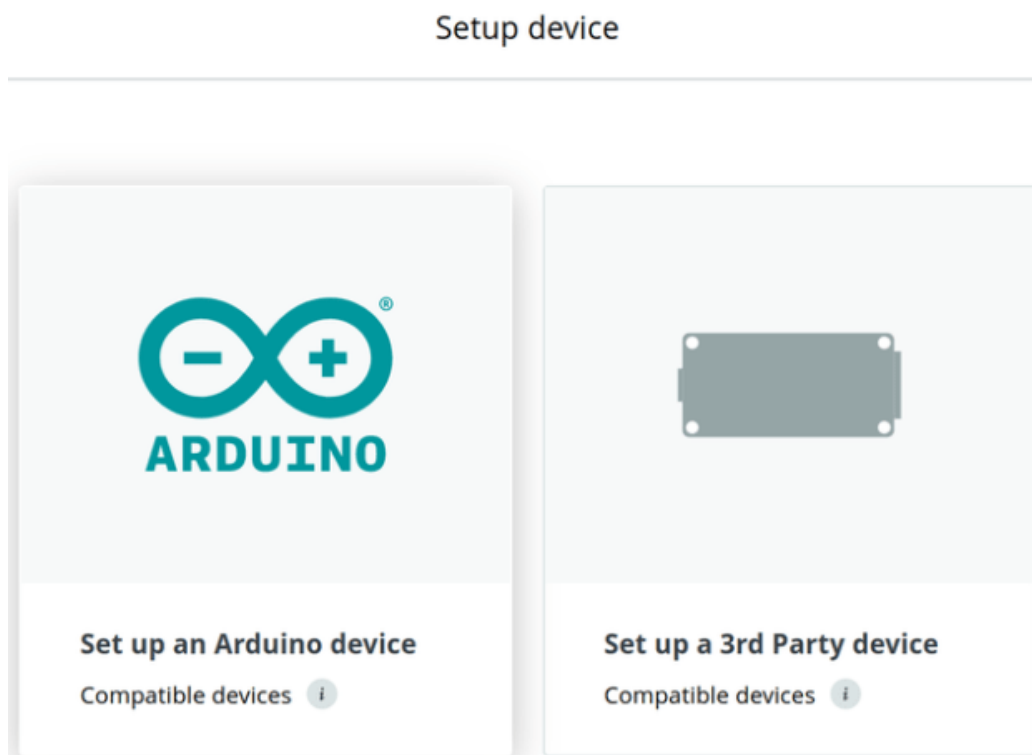
If you already have worked with the cloud and entered a network, then the information is preserved. You can click to change the network information if required.

Device Settings

The device is the microcontroller that you are using for your Thing, which in this case is the Arduino Nano 33 IoT board.

You'll need to have your microcontroller attached to the workstation, and the Arduino Agent needs to be running – on most installations the agent runs by default on system startup, so as long as you have your board connected you are good to proceed.

Click on the Add Device button. A dialog box will display, asking what type of microcontroller you are using.



We are using an Arduino device, so we will select Arduino. The agent will then connect to both the device and the IoT Cloud, and with any luck it will detect the type of device you have.

Assuming that it made the correct selection, you'll need to let it load some software onto the microcontroller. This can take some time, as it needs to be downloaded.

You'll get an opportunity to name your device, by default it will select a name for you, but you'll likely want to change it to something more suitable. Type a new name, note that the name cannot have any spaces in it.

Create a Cloud Variable

Now that we have a Network and Device associated with our Thing, we will need to add some variables.

Although our circuit has both an LED and a temperature/humidity sensor, we will start simple, and just work with the LED. We will modify our Thing later to include the DHT-22.

We will need one cloud variable to represent the state of our LED, that way by modifying this variable we will be able to turn the LED on or off.

Click on the Add Variable button.

Add variable

Name

|

↺

 Sync with other Things

i

Select variable type

▼

Declaration

Variable Permission

i

☒ Read & Write

☐ Read Only

Variable Update Policy

i

☒ On change

☐ Periodically

Threshold

0

ADD VARIABLE

CANCEL

The Variable editor dialog box will open. This is where you will edit the variables parameters.

Start by giving the variable a name, similar to how you would name a variable in any Arduino sketch. I will call mine redLED, you can give yours another name (especially if your LED is not red!).

Next, we can select a variable type. There are a variety of types to choose from, including several “custom” types that you likely have not seen before.

My experience, at least as of this writing (July 2021), is that the mobile application doesn’t work well with a lot of “custom” variable types. So for this reason I recommend using a more “standard” variable type.

I am going to make my variable a Boolean, as it suits my requirements of turning the LED on or off. If I wanted to modify the LEDs intensity, I would have been wiser to have chosen an Integer.

Once you select a type, you’ll see the variable declaration printed below it, in exactly the same format you would use in any Arduino sketch.

Next, we need to address the variable permission. You have two choices here:

- Read & Write
- Read Only

For an LED you will need to select Read & Write, as you will need to be able to write to the LED to control it. If this had been a sensor, like our DHT-22, it would suffice to be read only.



Finally, the variable Update Policy. Again, you have two choices:

- On change – Update the variable whenever something changes.
- Periodically – Update the variable every defined time period.


We want to control the LED with a switch of some sort, so On change is the best selection for our variable.


Add variable

Name
redLED


 **Sync with other Things** 

Boolean eg. true ▼

Declaration
`bool redLED ;` 

Variable Permission 

☒ Read & Write ☐ Read Only

Variable Update Policy 

☒ On change ☐ Periodically

ADD VARIABLE **CANCEL**

Once that is done, we just need to click the Add Variable button, and our variable has been added to our Thing.

Now our Thing has a variable, and it's connected to a network-attached device. Time to move on to the next step and add some code.

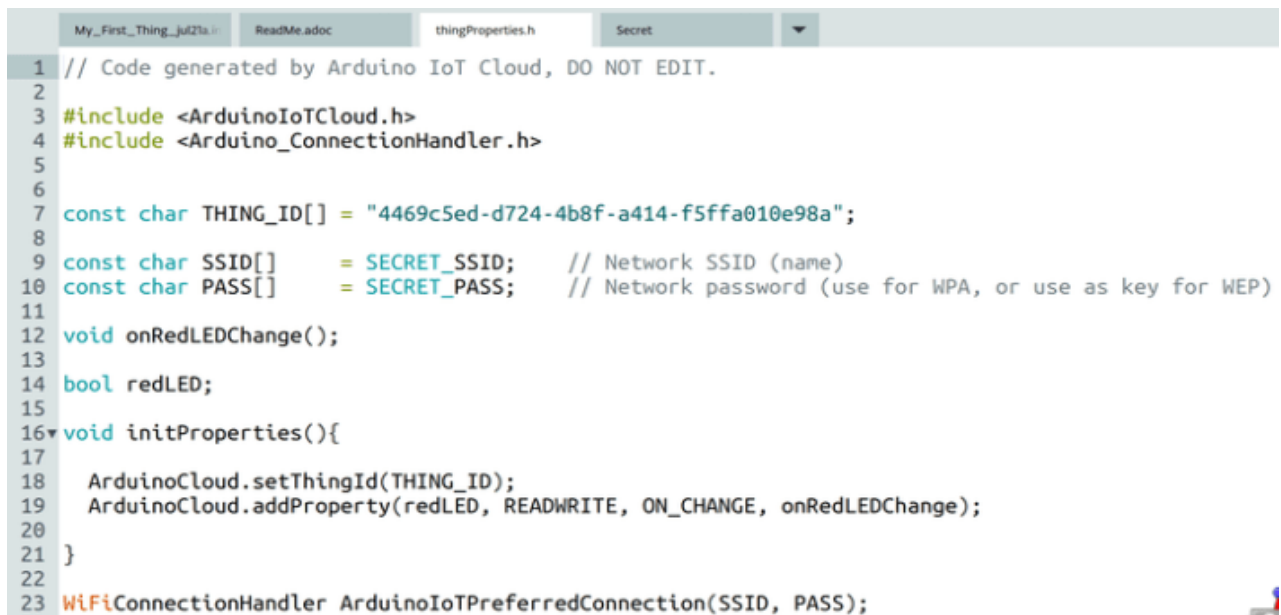
Adding Code

As with any Arduino project, we will need to load a sketch onto our device in order to make everything work. The wonderful news is that a lot of the sketch has already been written for you.

Click on the Sketch tab, and you'll be taken to the simple sketch editor. You can work on your sketch here, or you can click on the Open Full Editor button to open a more full-featured editor, one similar to the Arduino desktop IDE.

The sketch framework has already been written, and within the header it includes information regarding the cloud variable we just created.

You don't need to define this variable in your sketch, as it is already defined in `thingProperties.h` library, which has been included with your sketch. If you are in the advanced editor you can open this file and look at it, it has definitions for your variable(s) as well as a declaration for a function associated with that variables change event. There is also cloud and network connection information as well.



```
1 // Code generated by Arduino IoT Cloud, DO NOT EDIT.
2
3 #include <ArduinoIoTCloud.h>
4 #include <Arduino_ConnectionHandler.h>
5
6
7 const char THING_ID[] = "4469c5ed-d724-4b8f-a414-f5ffa010e98a";
8
9 const char SSID[]      = SECRET_SSID;    // Network SSID (name)
10 const char PASS[]     = SECRET_PASS;    // Network password (use for WPA, or use as key for WEP)
11
12 void onRedLEDChange();
13
14 bool redLED;
15
16 void initProperties(){
17
18   ArduinoCloud.setThingId(THING_ID);
19   ArduinoCloud.addProperty(redLED, READWRITE, ON_CHANGE, onRedLEDChange);
20
21 }
22
23 WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

You should never edit this file, it gets modified when you make changes using the Thing editor.

Modifying the Sketch

We will need to add a few lines of code to our sketch to allow the cloud variable we created to interact with the LED. Remember, we have yet to make that association.

Open the sketch for editing using either the advanced or simple editor.

We need to define an integer with the port number of our LED, which we have connected to data pin D8. In the declarations section of the sketch write the following code:

```
int myLED = 8;
```

Now in the Setup, we will need to define our pin as an output. Somewhere within the setup section add the following line of code:

```
pinMode(myLED,OUTPUT);
```

Now our LED connection is all set up. The next step is to control the LED based upon the value of the cloud variable we created, a Boolean which we named redLED.

The sketch has already come with a function that is called whenever the cloud Boolean changes state, as we set up the variable as On change.

If you look towards the end of the sketch, you'll see an empty onRedLEDChange() function. This function is called whenever something modified the value of the redLED boolean.

Add the following code to the function:

```
Serial.println(redLED);  
  
    if(redLED){  
        digitalWrite(myLED,HIGH);  
    }else{  
        digitalWrite(myLED,LOW);  
    }
```

You can see that we first print the state of the redLED variable to the serial monitor,

Next, we examine the value of the redLED boolean. If it is TRUE (i.e. equal to 1) then we turn on the LED by sending the output HIGH. If it isn't true, then we turn off the LED by sending the output LOW.

The editor will add ending brackets when you type, so you'll want to verify your code, to make sure that you didn't get an extra one in there by mistake.

Once you have checked your code, you can upload it to the target device. You'll see a status window displaying your progress.

Now we have the Arduino programmed, and if we open the Serial Monitor tab we can see if we have successfully connected to the cloud.

If we have, then all we need to do to control our LED is to modify that cloud variable. And to do this we will need a dashboard

Build a Dashboard

The Dashboard is our GUI interface that we can use on the web or on the mobile application. We can add elements like switches and displays to the dashboard, and link those elements to cloud variables.

We are going to add a switch to our dashboard and use it to set the value of the redLED cloud Boolean variable we created for our Thing. The code we've uploaded to our Arduino will use that variable to set the LED connected to pin D8.

Go to the Dashboard tab and click the Build Dashboard button. A blank dashboard will appear.

The dashboard has no elements and is called “Untitled”, and to modify it you’ll need to switch to edit mode (the dashboard is in view mode when you first open it). Click on the edit (pencil) icon in the upper left corner to enter edit mode.

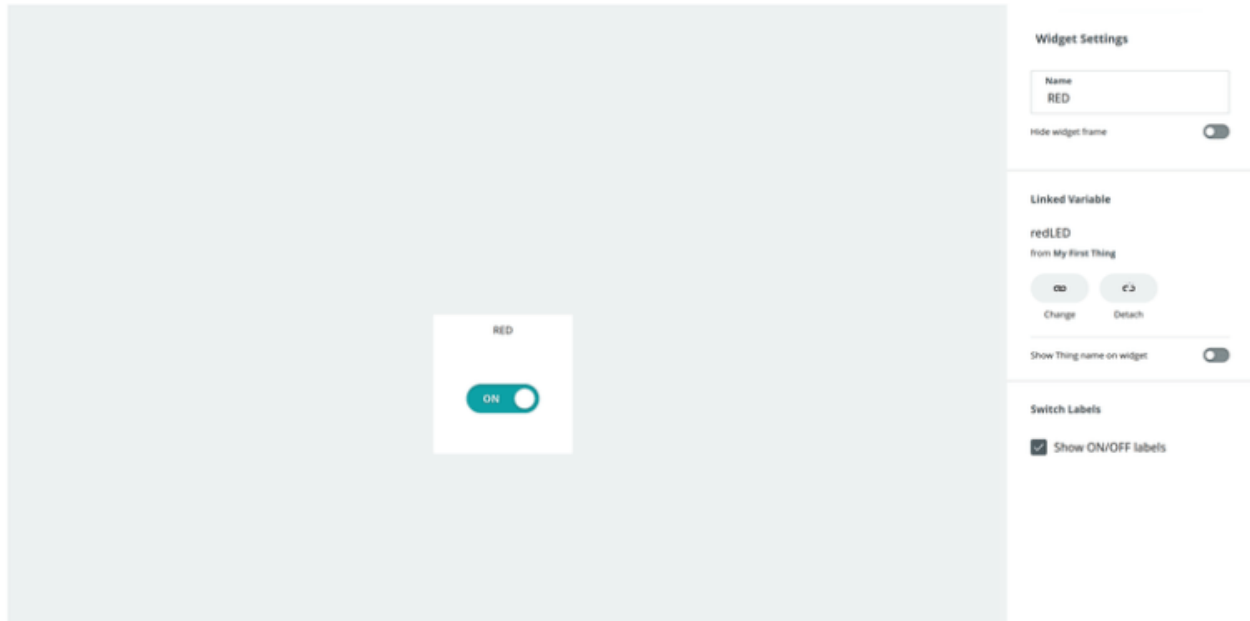


Once you are in edit mode you will see an Add drop-down button. You will also be able to edit the title of your dashboard.

The Add drop-down lets you select from a number of different Widgets, which are elements you can include on your dashboard.

The first one is a Switch, which seems appropriate for controlling an LED. Select Switch, and one will be deposited on your dashboard.

Click on the menu button of the switch to edit it. You’ll see an editor window.



In this editor window you can give your switch a name, or just leave that blank. I called mine “RED”, as it controls a red LED.

Click on Linked Variable to attach a cloud variable to the switch. Another window will open, allowing you to link to an existing cloud variable.

<INSERT LINK VARIABLE>

On the left side of the window you’ll see the Things you have created. When you select one of them (and you may only have one to start with) you’ll get a list of available cloud variables on the right.

I selected my Thing and the redLED variable. Then I clicked the Link Variable button. My redLED variable is now being controlled by the switch element.

Once I fine-tune my switch by selecting or deselecting the ON/OFF labels, I can click the Done button. And my switch is now part of my dashboard.

Testing your First Thing

Testing our Thing is simple, just try the switch. If all is working then you should be able to control the LED with it.

Try opening the Dashboard in another window and observe the Serial Monitor while trying the switch. You should see the values of “0” or “1” printed out, as these were the values for redLED when the function was called.

After you get tired of playing with the switch go back into the dashboard. Put it in edit mode if it isn’t already (you can run the dashboard in both edit and view modes).

Add another widget, this time a pushbutton. Edit its properties and attach it to the same redLED cloud variable. You can attach the same variable to more than one widget.

Now try it out. Observe that when you hold down the push button you’ll not only activate the LED, you’ll also toggle the switch. And it works both ways, the elements interact with one another through the cloud variable.

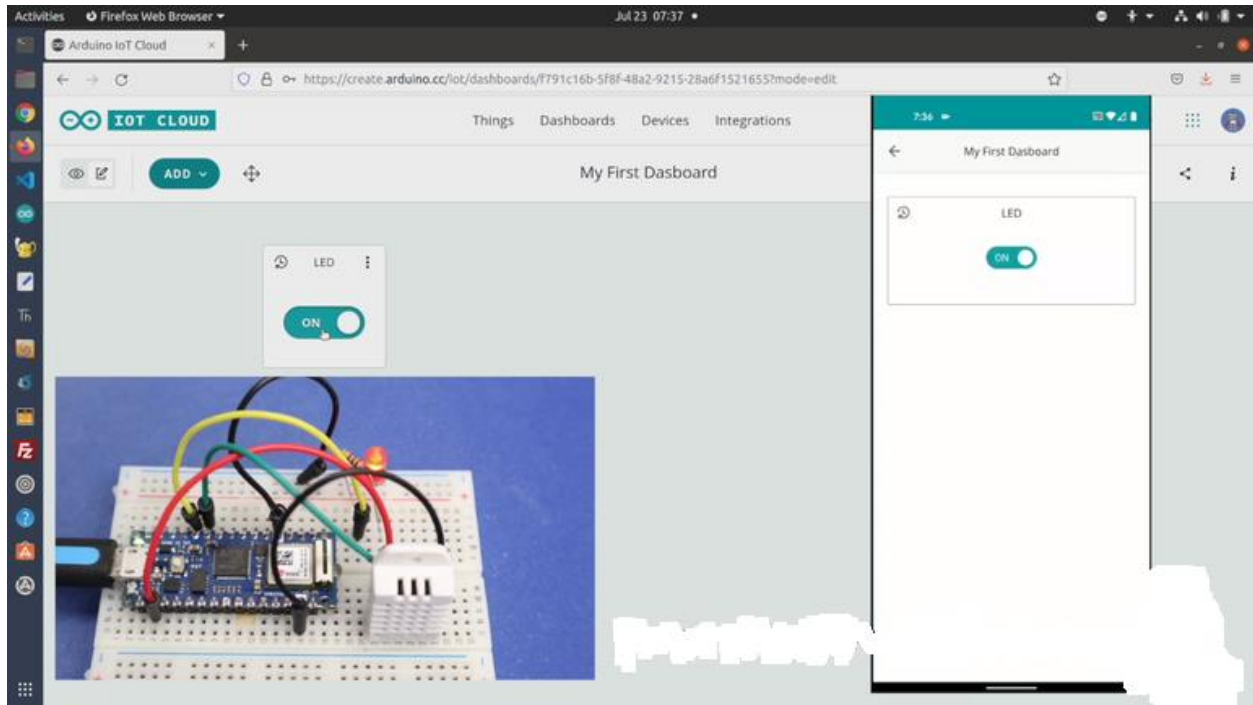
Using the Mobile App

You can also control your LED using the Arduino IoT Remote mobile app, which is available for both Android and IOS.

Load the app onto your device. Once it is installed, log in with your credentials.

The app only allows you to display and use dashboards, you can’t do any development with it. When you open it, you’ll get a list of your dashboards. Select the one you just made.

You will see your dashboard with the switch and pushbutton switch, and they should work to control the LED.



While you operate the mobile dashboard observe your web dashboard. Again, you'll see the switch and pushbutton actions are synchronized.

Adding the Temperature & Humidity Sensor

Now that we have mastered the control of our LED, we can focus on our DHT-22 temperature and humidity sensor. This will illustrate how we can get data back from a sensor to our dashboard.

Open the Things editor and select the Thing we created earlier.

Now click the Add button to add another variable. I will name mine boardTEMP, you can choose another name if you wish.


I'll resist the urge to use a fancy variable type and just choose a Floating Point variable. The declaration will show this as a float.

Permission is just Read Only, as a sensor like this cannot be written to. And I will keep the Update Policy set to On change, so I get a new reading every time the temperature changes. You can leave the Threshold at zero.


Add variable

Name

boardTEMP




Sync with other Things




Floating Point Number

eg. 1.55




Declaration

`float boardTEMP ;`




Variable Permission



☐ Read & Write

☒ Read Only

Variable Update Policy



☒ On change

☐ Periodically

Threshold

0

ADD VARIABLE

CANCEL

Click Add Variable to add this new variable to the Thing.

Now repeat the process and add another variable, this time for the humidity value. The parameters are all the same as the temperature variable except, of course, for the name. I named mine boardHUMID.

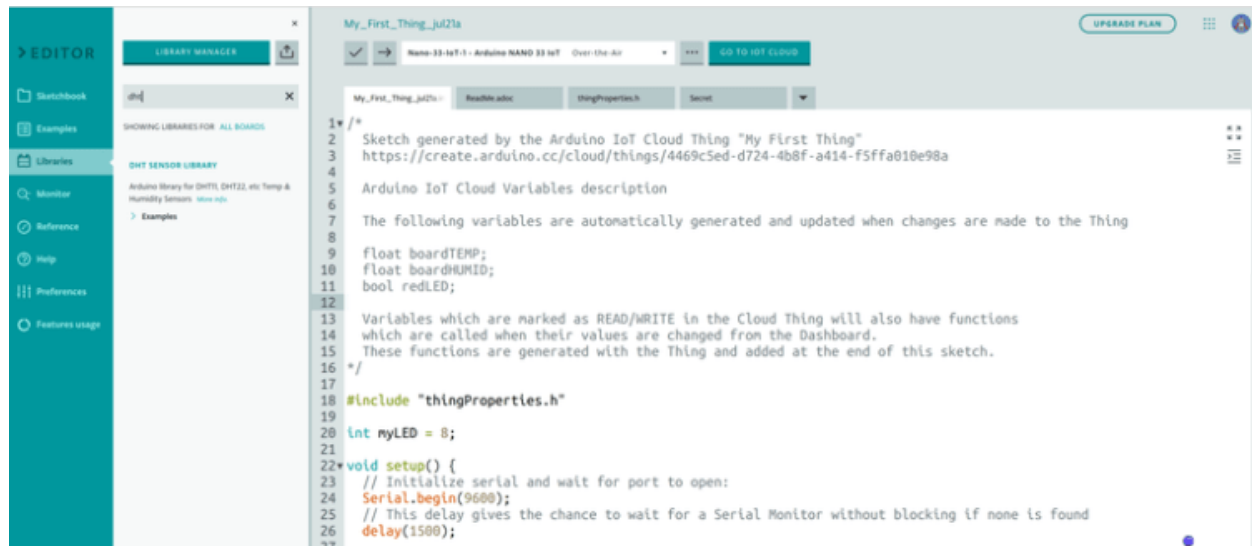
Temperature & Humidity Code Changes & Library

Of course, we will now need to make some code changes in order to populate our new cloud variables with actual temperature and humidity data.

You need to go to Sketch and this time you really need to use the advanced editor, as we will need to install a library to work with our DHT-22 sensor.

You will note that the code has been modified a bit since you last looked at it, the header includes the names of your two new cloud variables. Unlike the previous time, there are no new functions created.

To work with the DHT-22 we will need to include a library in our code. And the advanced editor has a library manager, with all the features of the Library Manager you're familiar with in the desktop IDE.



Use the Search box and search for “DHT”. The result will be the DHT Sensor Library. Highlight the result and use the Include button to add the lines of code required to include the library in the sketch. Note that the button allows you to use older versions of the library as well.

You'll note that the following code has now been added to your sketch:

```
// DHT sensor library - Version: Latest
```

```
#include <DHT.h>
```

```
#include <DHT_U.h>
```

Now we need to add the code to work with the sensor and to get back the temperature and humidity values.

Add the following code to the declarations section of the sketch:

```
#define DHTPIN 7
```

```
#define DHTTYPE DHT22
```

```
DHT dht(DHTPIN, DHTTYPE);
```

This declares the pin we are using (D7) for the sensor connection and the sensor type. If you have used a DHT-11 then change the second line to DHT11.

In the setup we need to initialize the sensor with the following code:

```
dht.begin();
```

We will place the code to read the temperature and humidity into the Loop, as we want to continuously do this. We also need to give the DHT-22 about two seconds between readings to stabilize, as its capacitive sensor based design is rather slow.

Here is the code for our Loop

```
delay(2000);  
  
boardTEMP = dht.readTemperature();  
  
boardHUMID= dht.readHumidity();
```

So what we are doing is reading the temperature and humidity values every two seconds and updating the values of the two cloud variables.

Make the changes, save and upload the sketch to the Nano 33 IoT board.

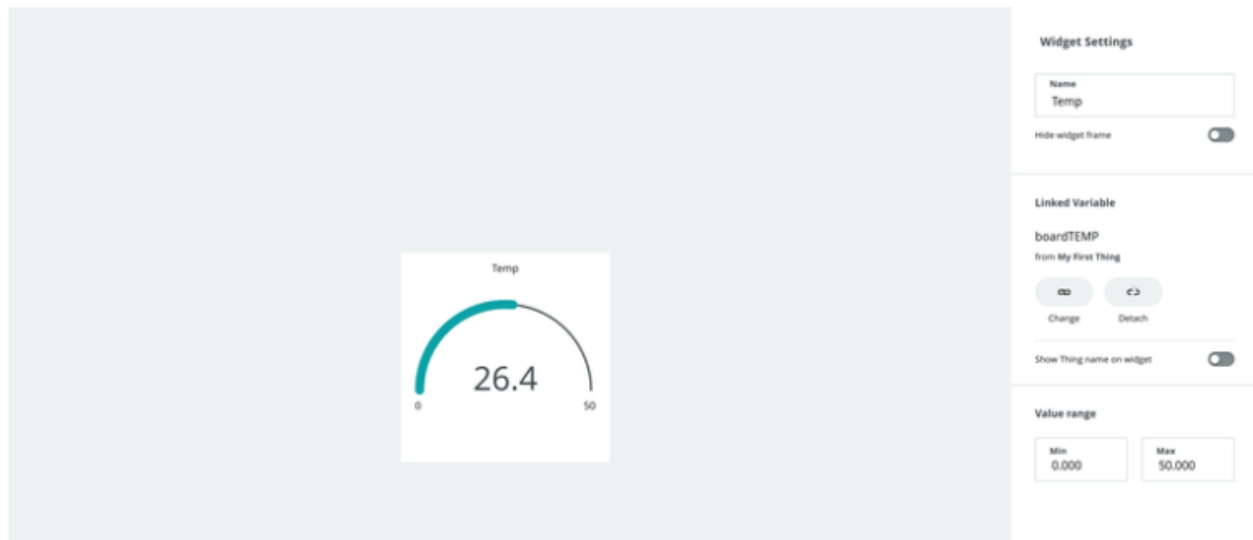
Temperature & Humidity Dashboard Changes

Now that we have added temperature and humidity to our Thing, we will want to display these values on our dashboard.

Go back into the Dashboard list and select the dashboard you have already constructed. Put it into edit mode.

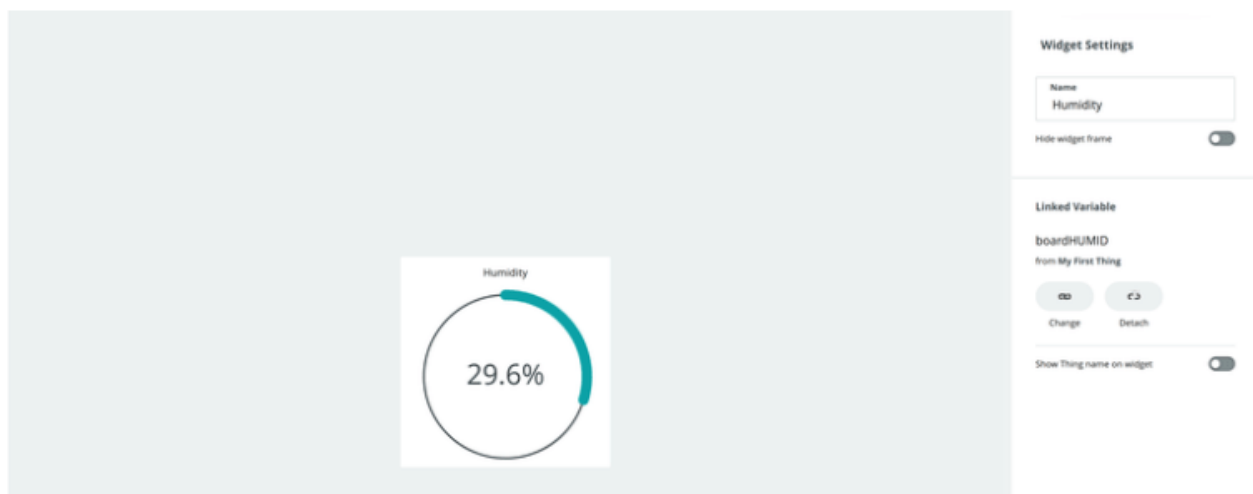
Now use the Add button to add another widget. This time, choose a Gauge widget.

Set the name as “Temp” or “Temperature” and link it to the boardTEMP cloud variable. You may also want to change the value range, as our reading is in Celsius and 100 is a bit high for room temperature. I set mine to 50 and left the lower limit at zero.



Within a few seconds of clicking the Done button, your temperature gauge should be working.

You can do the same thing with humidity. I chose to use a Percentage widget instead of a gauge, and I set it to link with the boardHUMID cloud variable.



Once again, it will just start working, displaying the humidity reading from the DHT-22 sensor.

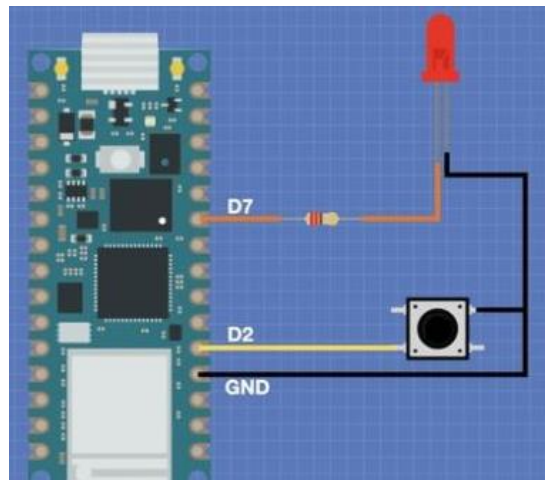
Cloud + Arduino Nano RP2040 Connect

In this next experiment, we will not be building a dashboard. Instead, we will connect two microcontrollers through the cloud, and we'll have them control one another.

We will be using two Arduino Nano RP2040 Connect boards for this experiment, but as with the last demo, you could substitute a different board if you wish. You could even use two different microcontrollers, as there is no requirement that they are identical.

Nano RP2040 Connect Hookup (x2)

Each of the Arduino Nano RP2040 Connect boards has an identical hookup, which is illustrated here:



It's a pretty simple circuit, consisting of just a pushbutton and an LED.

Our project operation is simple. Each device has a pushbutton and an LED. Each pushbutton acts like a toggle switch and controls the LED on the OTHER device.

This project is one of the many examples from the [Arduino IoT Cloud Documentation](#) page, specifically the “Device to device with Arduino IoT Cloud” experiment. You can follow the

instructions there as well as here, but you should know that they have one slight error in the code they provide – I’ll explain in a moment!

It would be a good idea to label your two circuits as “Device 1” and “Device 2” so that you don’t get them mixed up!

Thing Setup – Device 1

We will start by setting up our first Thing, the one for Device 1.

Open the Thing editor and create a new Thing. As per the Arduino example, I named mine “Remote Light One”, but of course you’re free to name it whatever you wish.

Make sure that the microcontroller you have labeled as “Device 1” is connected to the USB port of your workstation. Then add the Device using the Select Device button, as we did in the last experiment. Again, I used the Arduino-suggested name of “Device_1” for my RP2040 Connect board.

After you set up your device, you’ll need to add a couple of variables. Once again, I used the names and settings suggested by Arduino:

- switch_1 – an Integer with Read & Write permissions.
- switch_2 – a Boolean with Read & Write permissions.

The final step in setting up your Thing is to add your network credentials. After that, you can switch over to the sketch tab.

Coding – Device 1

In the sketch tab for the first Thing, you'll need to define the connections to the LED and pushbutton as follows:

```
int buttonPin = 2;
```

```
int ledPin = 7;
```

Next, you'll add the following lines to the Setup, to set up the mode of both pins:

```
pinMode(buttonPin, INPUT_PULLUP);
```

```
pinMode(ledPin, OUTPUT);
```

In the Loop, you'll need to add the toggle code for the pushbutton:

```
int buttonState = digitalRead(buttonPin);
```

```
if(buttonState == LOW) {
```

```
    switch_1 = !switch_1;
```

```
    delay(500);
```

```
}
```

And finally, we come to the error in the Arduino documentation!

When we added our two cloud variables to our Thing, it created two “on change” functions as follows:

onSwitch1Change()

onSwitch2Change()

In the Arduino example, they added code to the onSwitch1Change() function. That is incorrect, on Device_1 the following code should be added to the onSwitch2Change() function:

```
if(switch_2) {
```

```
    digitalWrite(ledPin, HIGH);
```

```
}
```

```
else{
```

```
digitalWrite(ledPin, LOW);
```

This is the code that monitors the state of the cloud variable and toggles the LED.

Once you have modified the sketch, upload it to your device. Now disconnect the device and connect the Device 2 microcontroller.

Thing Setup – Device 2

The setup of the second Thing is very similar to the first, with one important difference – we are going to synchronize our variables this time.

Begin by creating a new Thing, and naming it “Remote Light Two”.

Add your second microcontroller as the device, and name it “Device_2”.

Now we come to the variables. This is where things are a bit different, as once we create them, we will need to synchronize them with the variables in our “Remote Light One” Thing.

Start with the first variable, which, like with the first Thing, is named “switch_1”. However, instead of defining the type and permissions, click the “Synchronize Variables” link.

You will arrive at a screen that lists all the variables in your existing Things. Select the “switch_1” variable from the “Remote Light One” Thing. Once you have done that, click the Synchronize Variables button.

You have now synchronized the two variables. So if the state of one changes, the state of the other will change to match it.

You’ll note that you don’t need to define the type or permissions of your new variable, as it will inherit these properties from the other Thing’s variable.

Repeat this process with the second variable, synchronizing it to the equivalent “switch_2” variable on the first Thing.

Now that you have done that, it’s time to modify the sketch for the new Thing.

Coding – Device 2

As you have likely suspected, the code for Device_2 is almost identical to the code we entered for Device_1.

The declarations for the pushbutton and LED connections are the same:

```
int buttonPin = 2;  
int ledPin = 7;
```

The same goes for the pin mode settings in the Setup function:

```
pinMode(buttonPin, INPUT_PULLUP);  
pinMode(ledPin, OUTPUT);
```

The code in the Loop is a bit different, as we are now toggling the switch_2 variable and not switch_1.

```
int buttonState = digitalRead(buttonPin);  
  
if(buttonState == LOW) {  
    switch_2 = !switch_2;  
    delay(500);  
}
```

And the code in the onSwitch1Change() is also similar (and they have it in the correct place this time).

```
if(switch_1) {  
    digitalWrite(ledPin, HIGH);  
}  
else{  
    digitalWrite(ledPin, LOW);  
}
```

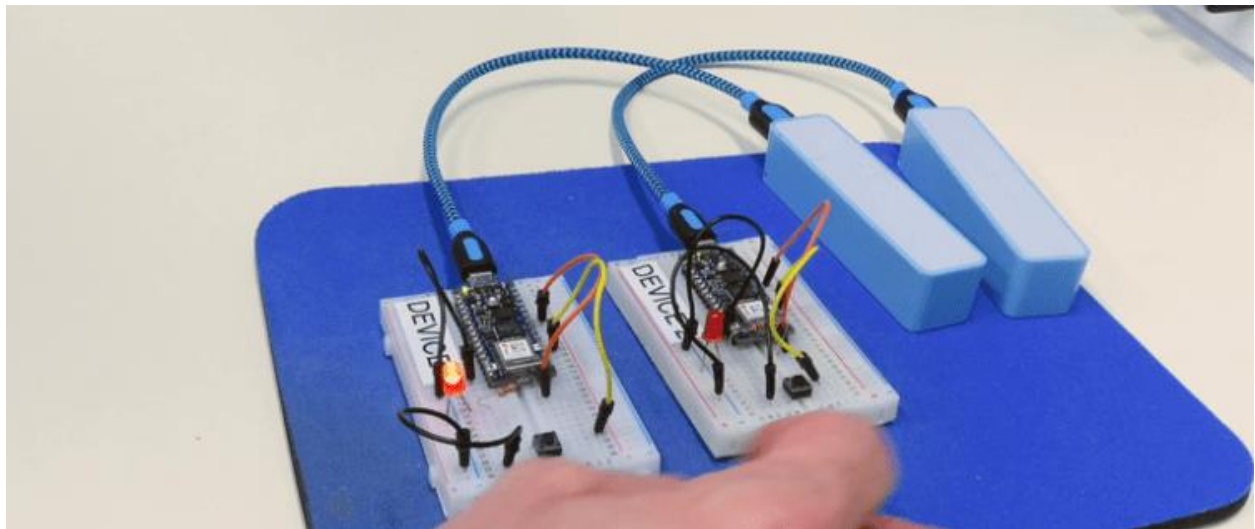
Load the code onto the microcontroller to complete the setup.

Testing the Devices

You will need to supply power to both devices using their USB ports. You can do this in any convenient way, I used a couple of USB power banks to test.

Power up the devices and wait a bit until the network connections are established. Then try pushing the pushbutton on Device 1. This should result in the LED on Device 2 illuminating and staying lit. Press the button again and the LED should extinguish.

Now try the pushbutton on Device 2. It should operate the LED on Device 1.



While this is a simple demo, you can likely see a lot of potential applications for this. Keep in mind that the two devices do not need to be on the same network, as long as they both have Internet connections and can connect to the Arduino IoT Cloud they will work. So Device 1 and Device 2 could be in different parts of the world!

Incidentally, I'm not sure why Arduino chose to use an Integer and a Boolean, as two Booleans (or two Integers for that matter) would have worked fine. Perhaps it was just to illustrate that either of them will work in this application.

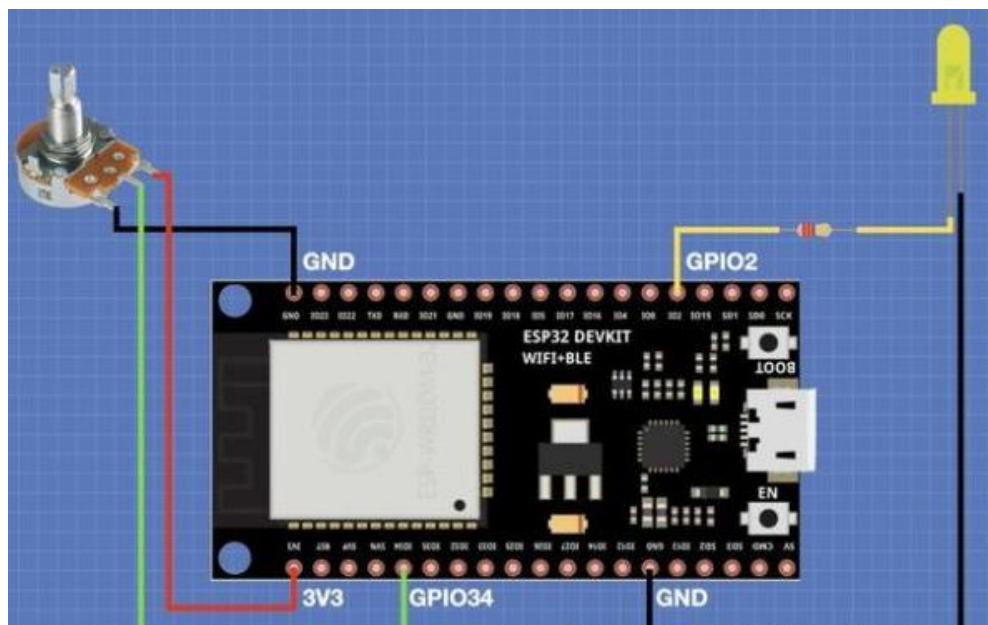
Cloud + ESP32

Our final experiment with the Arduino IoT Cloud will be with a non-Arduino microcontroller.

The Arduino IoT Cloud now supports both the ESP8266 and ESP32 boards. It should be noted that as of this writing, support for the ESP32 is still considered “experimental”, but it seems pretty solid.

ESP32 Hookup

Once again, the hookup is quite simple, we are going to be adding an LED and a potentiometer to our ESP32.



Any linear-taper pot of 5K or greater will work, I used a 10K pot in my experiment.

Note the GPIO numbers and check the pinout of your ESP32 module, as it may not be the same as mine. I used an ESP32 WROVER module.

Build an ESP32 Thing

Constructing a Thing for an ESP32 is almost the same as it is for an Arduino board, with one exception. The ESP32, or any non-Arduino board, will need to be tagged with a secret key number.

Start by creating a new Thing, and give it a name – I called mine ESP32 Thing, but I’m sure you can be more creative than that!

Let’s add the variables first, we will need one for the LED state and one for the value from the potentiometer. Set them up as follows:

esp32LED – Boolean – Read & Write – On Change

esp32POT – Integer – Read Only – On Change

Now we need to set up our device. Make sure you have your ESP32 board plugged into the serial port, and then click the Select Device button.

This time, we will need to select 3rd Party Boards instead of Arduino Boards. Click to set up a new device. On the next screen, select ESP32. You’ll also need to select the model of ESP32 that you are using. On the next screen, give your device a name, and click the Next button.

Now here is where things are different. You’ll now see a screen with a ‘Secret Key’ on it. This is the only place you will see this key, so you’ll need to copy it. You can copy it by clicking on the icon beside it, or you can download a PDF document with the key in it.

After making absolutely sure that you have a copy of the secret key, you can confirm it and move to the next screen. Your ESP32 setup will be complete.

Now it’s time to set up your network credentials. Click the Configure button in the Network section.

You’ll note that this time you need to provide three parameters, instead of two.

- SSID
- Password
- Secret Key

Save these, and we can now go over to the sketch.

ESP32 Coding

The coding for our ESP32 is pretty basic, we will start by declaring the pin connections to the LED and the potentiometer:

```
int pinLED = 2;  
int pinPOT = 34;
```

In the Setup function, we only need to define the LED pin as an output, as the potentiometer pin will be used as an analog input.

```
pinMode(pinLED,OUTPUT);
```

We will take care of reading the potentiometer and updating its cloud variable inside the Loop. Note the half-second delay, this is done to reduce the amount of data that we need to exchange with the cloud server.

```
esp32POT = analogRead(pinPOT);  
delay(500);
```

And finally, in the onEsp32LEDChange() function we will add the code to toggle our LED:

```
if(esp32LED) {  
    digitalWrite(pinLED,HIGH);  
}  
else{  
    digitalWrite(pinLED,LOW);  
}
```

Once you have entered the code modifications, upload your sketch to the ESP32.

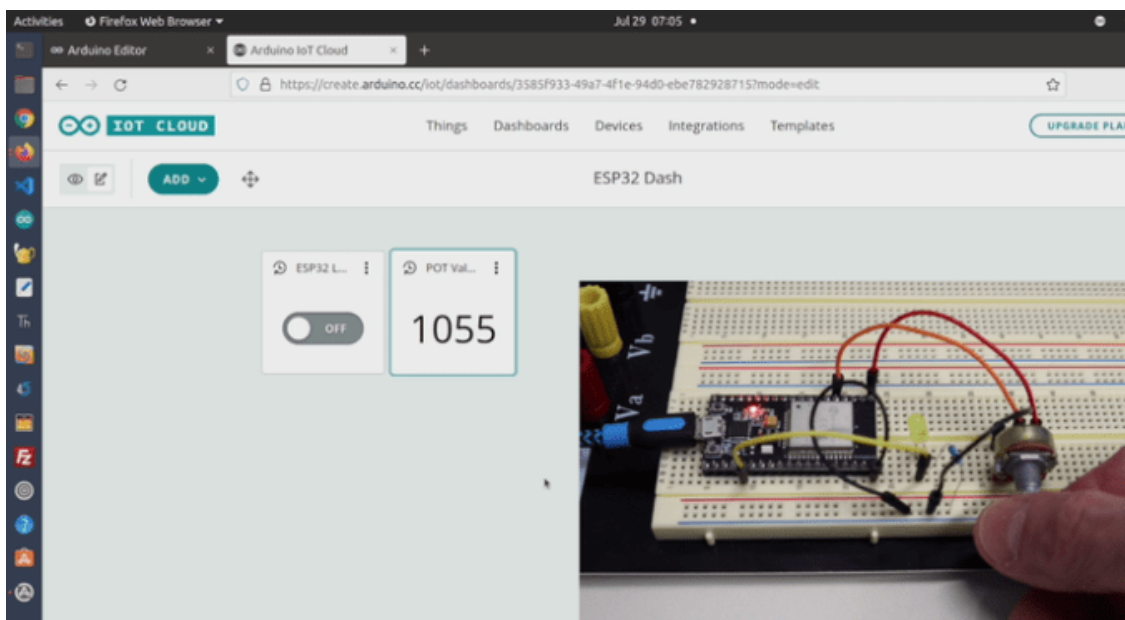
ESP32 Dashboard

Now that we have our ESP32 programmed, it's time to put together a dashboard to control it.

Creating a dashboard for an ESP32 is no different from building one for an Arduino board. You just create a new dashboard, put it into edit mode, give it a name, and then start adding widgets.

I used a switch widget for the LED, and the instructions for using it are exactly the same as they were for our first experiment. Just link the switch to the esp32LED cloud variable, and you're done.

For the potentiometer, I chose a Value widget, which just displays a numerical value. I linked it to the esp32POT variable.



And that's it! The dashboard should work, you should be able to control the LED with the switch and read the value of the potentiometer on the display. Remember that the ESP32 has a 12-bit A/D converter, so the range is 0 to 4095.

Conclusion

The Arduino IoT Cloud certainly opens up a whole new world of applications that we can construct with microcontrollers. It's pretty easy to use, and I certainly see a lot of potential with it.

Now as of this writing, July 2021, the product still seems to have a few rough edges. Here are a few I have noticed so far:

- As mentioned earlier, the mobile app only seems to work with “basic” variable types. I had this problem, and the Arduino Forum has a number of posts about the same issue.
- If you make changes to a dashboard, you'll need to log out and back into the mobile app to see them. I should note that I am running the Android version, and so I can't speak as to if this also happens with the IOS version.
- During the filming of the video accompanying this article, I had several instances of getting logged out of my account for no apparent reason. I will admit that it hasn't happened for about a week, so perhaps this was cured.
- I had much better results using Firefox than Google Chrome. Some screens didn't seem to display correctly on Chrome. I should point out that I have a similar issue in the Arduino Store, so I suspect it is a style sheet issue.
- I had a few instances of the microcontroller attached to the USB port suddenly “disappearing”. In one case, I had to reboot the workstation to get it back. I have observed this in three workstations, so I don't think it's related to a hardware issue on my side.
- I would love to see more documentation!

Having said all of that, I understand that this is a product that is constantly being developed and improved. In fact, just after I filmed the last scene of the video they added another tab to the top called Templates, which seems to have templates for a number of IoT projects. I'll have to explore that.

But despite the “growing pains” I still see this as a good product that is worthy of using. It will certainly make developing IoT applications a lot simpler.

So, I invite you to give the Arduino IoT Cloud a try. You can join with a free plan to get a feel for it, and if you like it (as I do) then subscribe to a higher-level plan.