

TD ARDUINO

1. Présentation de l'Arduino MKR1000

Arduino **MKR1000** is a board that combines the functionality of the **Zero** and the Wi-Fi Shield. It is the ideal solution for makers wanting to design IoT projects.



Arduino MKR1000 is based on the Atmel **ATSAMW25** SoC (System on Chip), that is part of the SmartConnect family of Atmel Wireless devices, specifically designed for IoT projects and devices.

The ATSAMW25 is composed of three main blocks:

- **SAMD21** MCU with ARM **Cortex-M0+** 32bit low power
- WINC1500 low power 2.4GHz IEEE® 802.11 b/g/n Wi-Fi
- ECC508 CryptoAuthentication

The ATSAMW25 includes also a single 1x1 stream PCB Antenna.

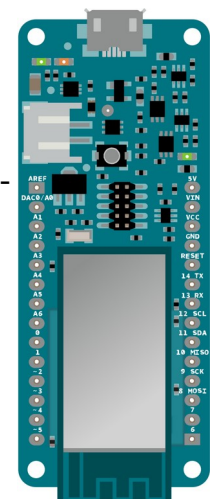
The design includes a Li-Po charging circuit that allows the Arduino/Genuino MKR1000 to run on battery power or external 5V, charging the Li-Po battery while running on external power. Switching from one source to the other is done automatically.

A good 32 bit computational power similar to the Zero board, the usual rich set of I/O interfaces, low power Wi-Fi with a Cryptochip for secure communication, and the ease of use of the Arduino Software (IDE) for code development and programming.

All these features make this board the preferred choice for the emerging IoT battery-powered projects in a compact form factor.

The USB port can be used to supply power (5V) to the board.

The Arduino MKR1000 is able to run with or without the Li-Po battery connected and has limited power consumption.



Microcontroller	SAMD21 Cortex-M0+ 32bit low power ARM
Board Power Supply (USB/VIN)	5V
Supported Battery(*)	Li-Po single cell, 3.7V, 700mAh minimum
Circuit Operating Voltage	3.3V
Digital I/O Pins	8
PWM Pins	12 (0, 1, 2, 3, 4, 5, 6, 7, 8, 10, A3 - or 18 -, A4 -or 19)
UART	1
SPI	1

I2C	1
Analog Input Pins	7 (ADC 8/10/12 bit)
Analog Output Pins	1 (DAC 10 bit)
External Interrupts	8 (0, 1, 4, 5, 6, 7, 8, A1 -or 16-, A2 - or 17)
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	no
Clock Speed	32.768 kHz (RTC), 48 MHz
LED_BUILTIN	6
Full-Speed USB Device and embedded Host	
Length	61.5 mm
Width	25 mm
Weight	32 gr.

The MKR1000 Wifi module supports WPA and WPA2.
However, the WiFi101 library does not support WPA2-Enterprise yet.

Warning: Unlike most Arduino & Genuino boards, the MKR1000 runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V.

Applying voltages higher than 3.3V to any I/O pin could damage the board. While output to 5V digital devices is possible, bidirectional communication with 5V devices needs proper level shifting.

La carte est détaillée ici :
<https://store.arduino.cc/arduino-mkr1000-wifi-with-headers-mounted>

Vous avez un kit (*bundle*) à votre disposition :
<https://store.arduino.cc/arduino-mkr-iot-bundle>

Ne connectez pas du 9V à la carte, cela va l'endommager. Utilisez le connecteur pour batterie 9V uniquement pour alimenter un composant externe.

2. Mise en place

1) Téléchargez la version de l'IDE pour Linux 64 bits (x86 pas ARM !) :
<https://downloads.arduino.cc/arduino-1.8.19-linux64.tar.xz>

2) Dézippez le dans votre dossier de travail ~/espaces/travail :

```
$ unxz arduino-1.8.19-linux64.tar.xz
$ tar xf arduino-1.8.19-linux64.tar
```

3) Allez dans le dossier arduino et créez un dossier portable :

```
$ cd ~/arduino-1.8.19
$ mkdir portable
```

4) Lancez l'IDE :

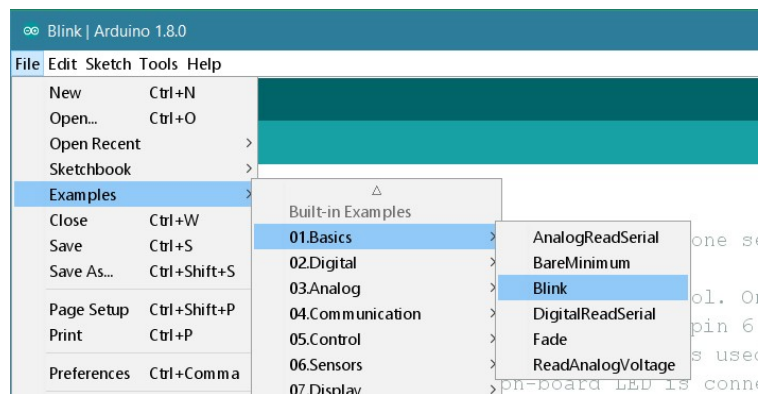
```
$ ./arduino &
```

5) Enlevez la mousse noire sous la carte avant utilisation !

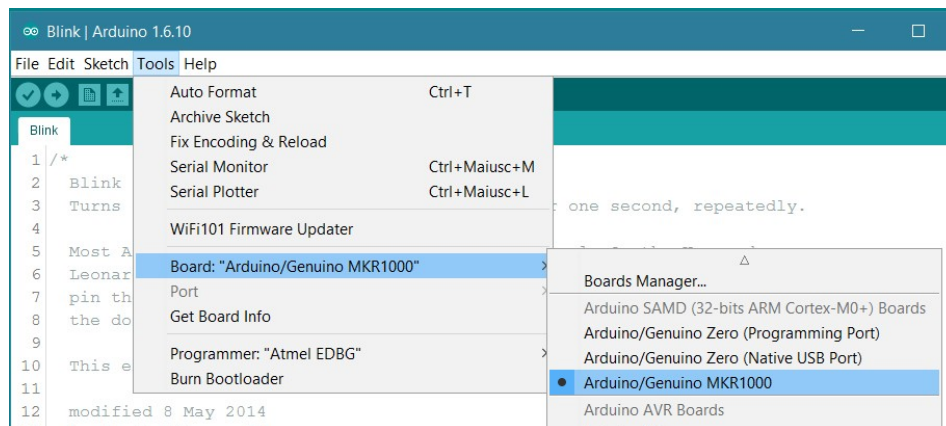
6) Pour programmer la carte MKR1000, il faut ajouter le package : Atmel SAMD Core. Pour cela, sélectionnez le menu **Tools**, puis **Boards** puis enfin **Boards Manager** (comme expliqué ici : [Arduino Boards Manager](#)).

Dans la fenêtre qui s'ouvre, cherchez le package **Arduino SAMD Boards 32-bits ARM Cortex-M0+** puis cliquez sur **Install** (avec dans la liste déroulante, la version correspondante de l'IDE, normalement la dernière).

7) Un projet Arduino s'appelle un **sketch**, sélectionnez et ouvrez le sketch **LED blink**: **File > Examples > 01.Basics > Blink**.



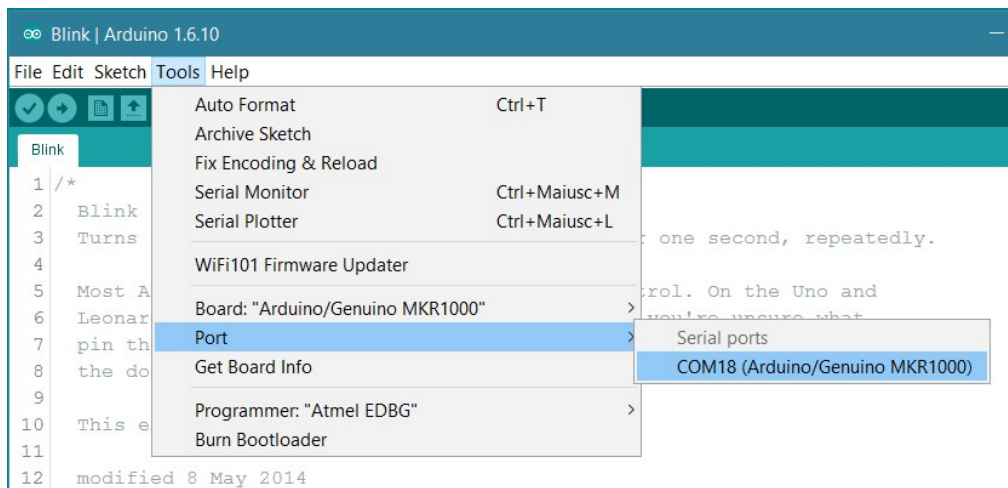
Sélectionnez la carte et le port avec le menu **Tools > Board** qui correspond à la carte Arduino MKR1000.



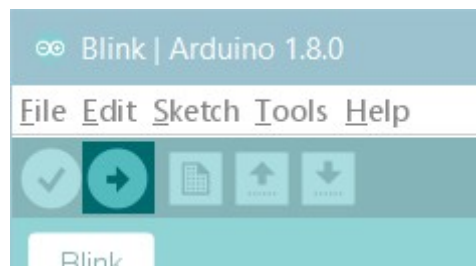
8) Sélectionnez le périphérique série (serial device) de la carte à partir du menu **Tools > Port > Serial Ports**. C'est normalement le port nommé **/dev/ttyACM0** (COM1 et COM2 sont habituellement réservés pour les ports série matériels).

Attention, sur la figure ce n'est pas le même (COM18).

Pour être sûr, vous pouvez déconnecter la carte et rouvrir le menu; l'entrée qui a disparu doit être la carte Arduino MKR1000. Reconnectez la carte et sélectionnez ce port série.



9) Cliquez sur le bouton **Upload** situé sous le menu principal de l'IDE. Attendez quelques secondes – vous devriez voir sur la carte les LEDs RX et TX clignoter. Si l'upload est réussi, le message **Done uploading** apparaîtra dans la barre de status.



Quelques secondes après la fin de l'upload, vous devriez voir la LED clignoter en orange.

Des informations supplémentaires sur Arduino sont disponibles ici : <https://www.arduino.cc/en/Guide/HomePage>

10) Lancez le **Library Manager** via **Tools > Manage Libraries...** et installez les bibliothèques suivantes :

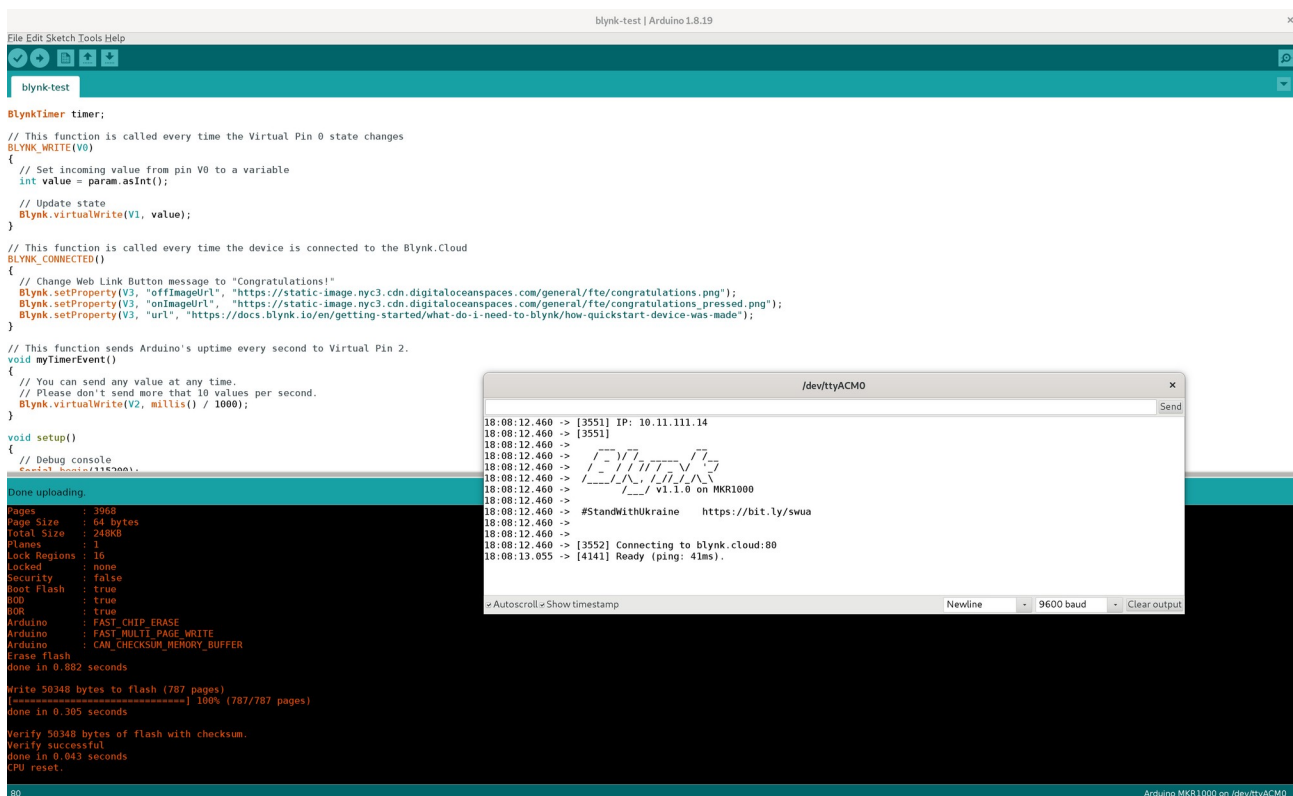
- **WiFi101** : pour se connecter à internet et scanner des réseaux
- **FlashStorage** : pour sauvegarder des valeurs afin qu'elles ne soient pas effacées à chaque reboot
- **RTCZero** : pour gérer des événements déclenchés par des temporisations
- **Arduino Low Power** : pour économiser de la batterie
- **WiFiUdp** : pour obtenir l'heure et la date depuis internet (déjà préinstallé)
- **Blynk** : pour interfacer avec l'application en ligne Blynk

3. Puzzle IoT

Nous allons réaliser un puzzle IoT avec le kit MKR1000. Afin de résoudre le puzzle, vous devrez tourner les potentiomètres jusqu'à ce que vous obteniez la combinaison correcte. La combinaison peut être fixée grâce à l'application **Blynk** sur votre interface Web ou votre smartphone. Une LED vous aidera à deviner la bonne combinaison, en vous fournissant un retour d'information (*feedback*) par des couleur : plus vous vous rapprochez, plus la couleur est chaude. Lorsque la combinaison correcte est trouvée, le *buzzer* va jouer une chanson !

1) Allez sur le site Web de **Blynk** situé ici : <https://blynk.io/> puis créez un compte. (optionnel) Vous pouvez aussi installer l'application **Blynk** sur votre smartphone puis créez un compte.

2) Une fois le compte validé en cliquant sur le mail envoyé à votre adresse, vous pouvez suivre le tutorial intitulé **Getting Started**. Dans les mini-pages qui s'enchainent, choisissez « Arduino », puis « Arduino IDE » puis « Wifi ». A la fin, vous avez une page de code à copier-coller dans votre Arduino IDE dans un nouveau sketch. Collez le code puis mettez les bonnes valeurs pour le SSID et le mot de passe du réseau Wifi sur lequel va se connecter votre MKR1000. Dans le code se trouve aussi un **template ID**, un **device name** et un **Auth Token** qui serviront à communiquer avec le Cloud de Blynk. Après upload, vous obtenez ceci dans le serial monitor :



3) Allez ensuite dans le panneau **template** et modifiez le template fourni. Vous pouvez le renommer **puzzle template**. Puis configurez **3 datastreams** comme suit avec un type entier, un min à 0 et un max à 9 :

Puzzle Template

Info Metadata Datastreams Events Automations Web Dashboard Mobile Dashboard

Q Search datastream

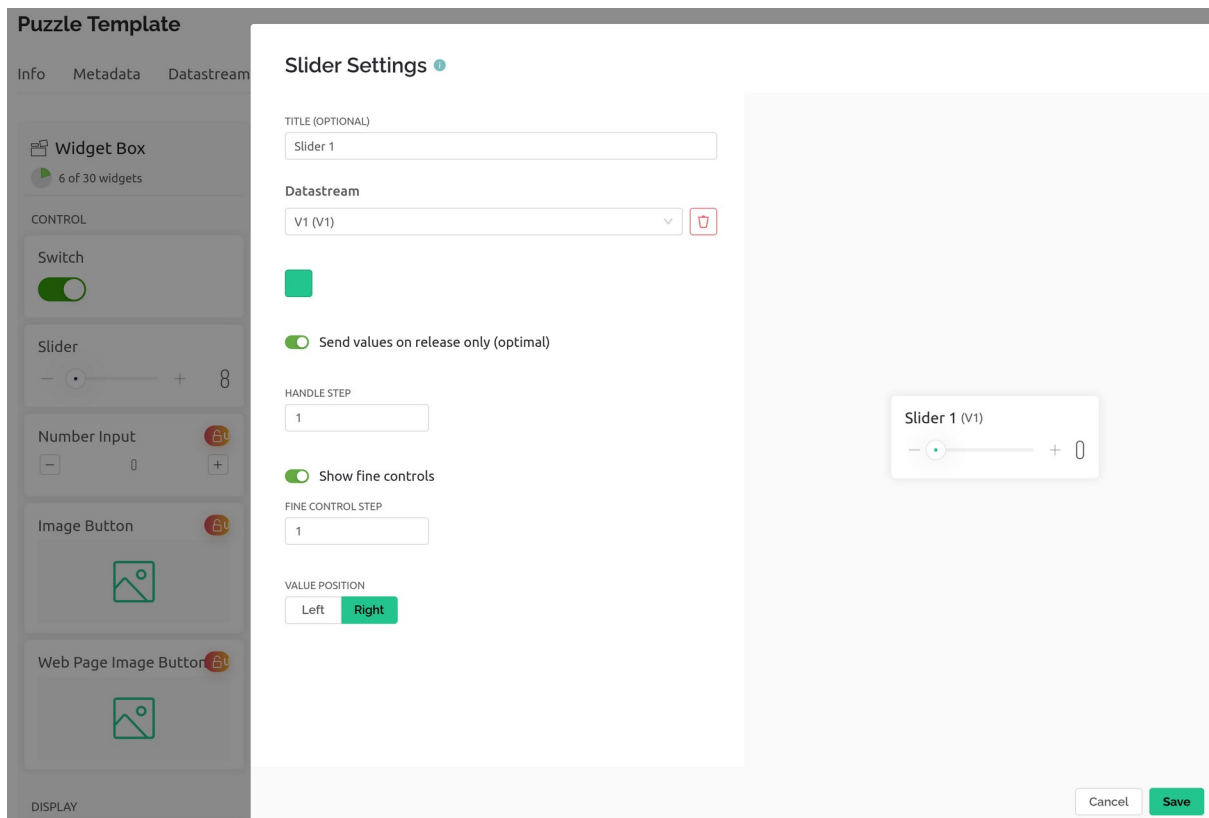
Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max
2	V1	V1	■	V1	Integer		false	0	9
3	V2	V2	■	V2	Integer		false	0	9
4	V3	V3	■	V3	Integer		false	0	9

4) Après avoir cliqué sur **Web Dashboard**, cliquez sur **Edit** en haut à droite et vous avez un canevas qui apparaît et une barre verticale de widgets à gauche.

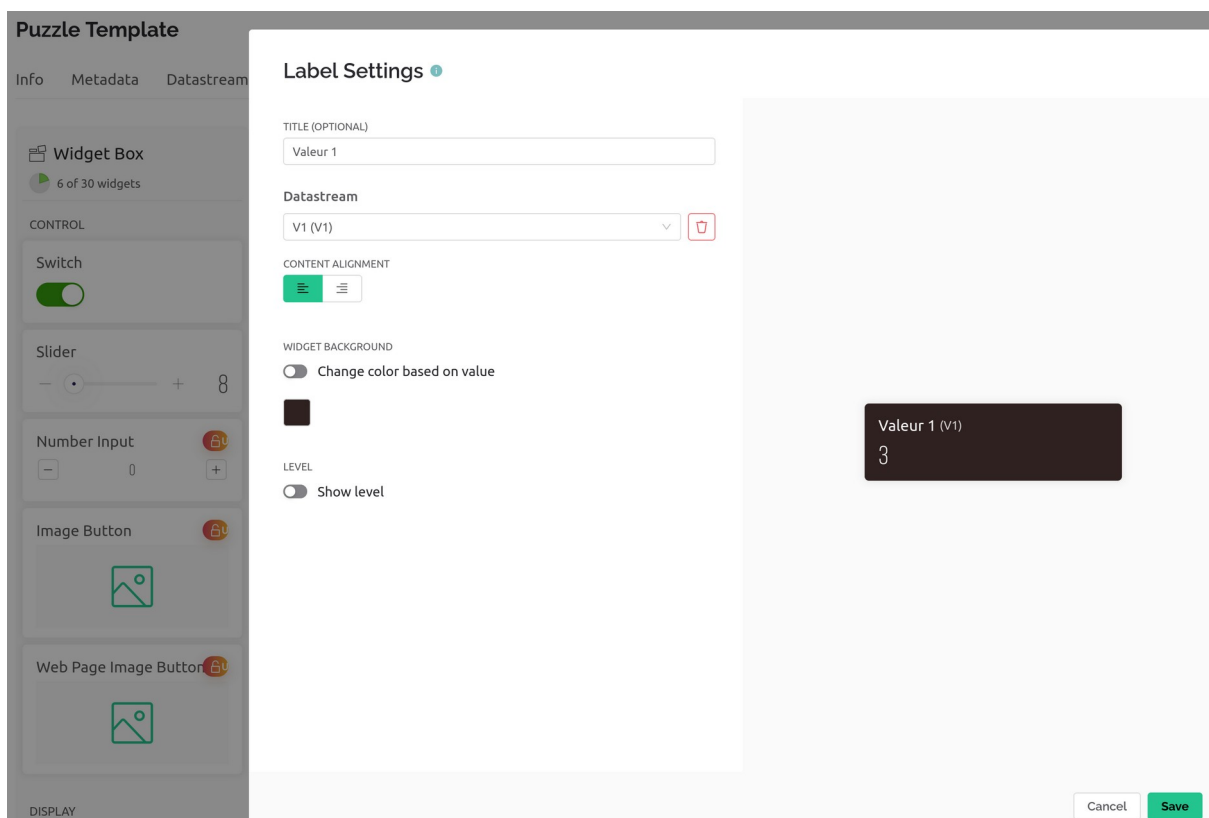
The screenshot shows the 'Puzzle Template' interface in 'Web Dashboard' mode. On the left is a vertical 'Widget Box' containing various controls: a Switch (checked), a Slider (set to 8), a Number Input (set to 0), an Image Button, and a Web Page Image Button. The main canvas displays a device status 'Device name' (Online) with fields for 'Device Owner' and 'Company Name'. Below this is a 'Dashboard' section with a time range selector (Last Hour, 6 Hours, 1 Day, 1 Week, 1 Month, 3 Months, Custom). Three data cards are visible: 'Valeur 1 (V1)' with value 6, 'Valeur 2 (V2)' with value 3, and 'Valeur 3 (V3)' with value 2. Below these are three sliders labeled 'Slider 1 (V1)', 'Slider 2 (V2)', and 'Slider 3 (V3)', each set to 0.

Sur les widgets de gauche choisissez **3 sliders** et placez les sur le canevas. Ils sont gratuits contrairement aux autres widgets qui nécessitent d'upgrader en version PRO (payante). Puis faites de même avec **3 labels**.

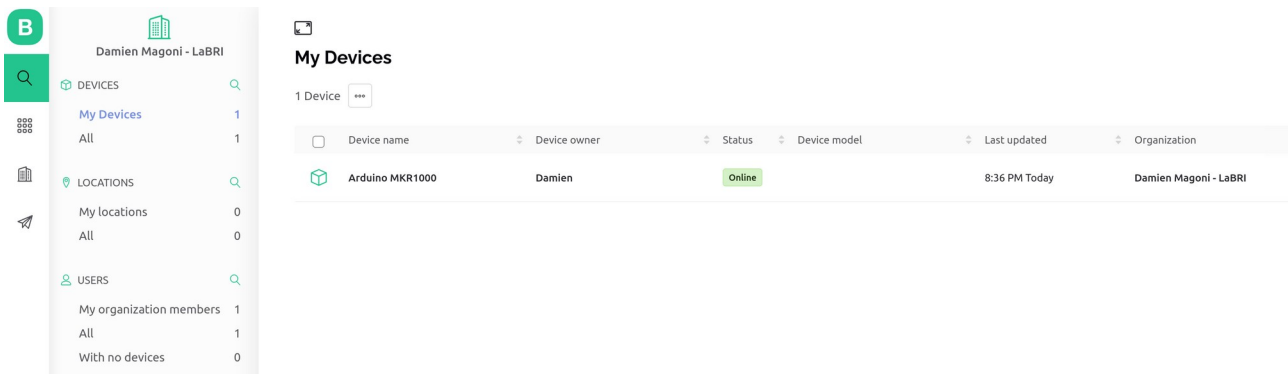
4) Configurez chaque slider en définissant min = 0, max = 9 et la sortie sur le datastream V1 pour le slider 1, V2 pour le slider 2 et V3 pour le slider 3, comme suit :



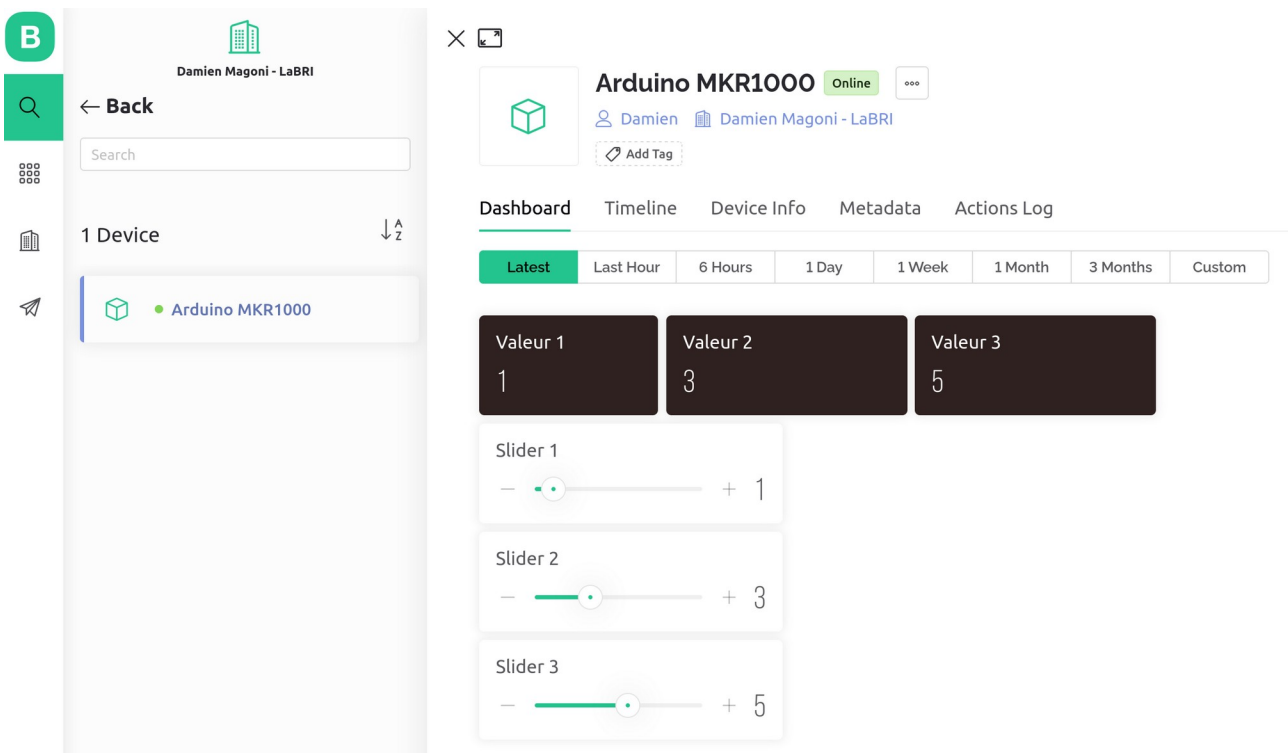
Configurez de même chaque label afin qu'ils affichent les valeurs de V1, V2 et V3.



Retournez dans le panneau **search** (la loupe en haut à gauche) vous devez voir votre MKR1000 :



Puis cliquez sur votre device et vous verrez apparaitre le **dashboard** de votre MKR1000 :



5) Chargez le code du fichier `puzzle-1-v2.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE, modifiez les valeurs indiquées dans le code puis uploadez le code dans le MKR1000. Dans la console de l'IDE située en bas, vous devez voir ceci :

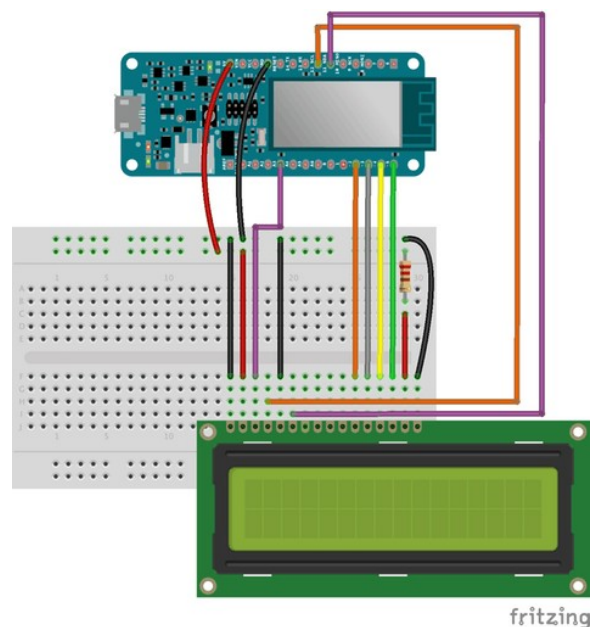
```
Verify successful
done in 0.03... seconds
CPU reset.
```

6) Lancez le contrôleur de port série : **Tools > Serial Monitor**.

Lorsque vous faites varier les valeurs de vos sliders avec les boutons – et + , les labels se mettent à jour et vous devez voir les valeurs s'afficher comme sur la figure ci-dessous. A noter que le temps est très court pour modifier les 3 sliders.


```
 /dev/ttyACM0
New combination: 5 3 3
New combination: 5 3 8
New combination: 7 3 8
New combination: 7 0 8
New combination: 7 0 0
New combination: 0 0 0
New combination: 3 0 0
New combination: 3 7 0
New combination: 3 7 9
New combination: 8 7 9
New combination: 8 7 4
New combination: 8 2 4
Autoscroll Show timestamp Newline 9600 baud Clear output
```

7) Vous allez maintenant ajouter un écran LCD. Connectez l'écran LCD comme suit :



Nous utilisons l'alimentation 5V et une résistance de 220 Ohm.

La luminosité peut être contrôlée en changeant la valeur de sortie du pin analogique 3 de 0 à 255, 0 étant la valeur maximale :

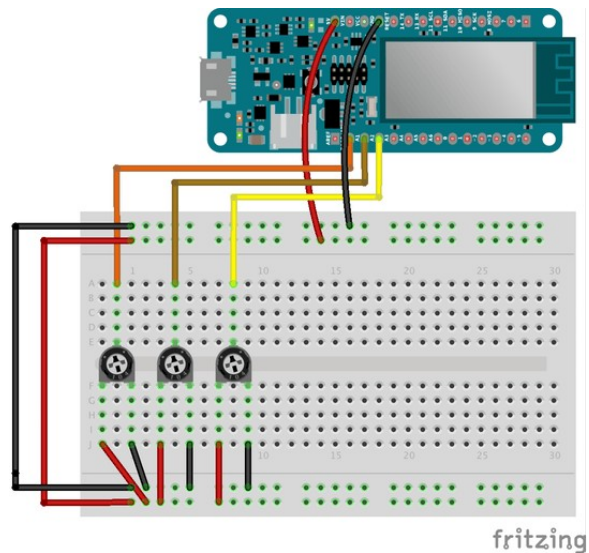
```
analogWrite(A3, 0);
```

8) Chargez le code du fichier puzzle-2.ino situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de vérifier que l'écran LCD fonctionne bien.

9) Ajoutez les potentiomètres. Pour lire les valeurs des potentiomètres, vous devez faire un appel à la fonction `analogRead()` sur le pin correct. Connectez les sur les pins analogues 0, 1, 2.



Notez que la valeur d'un potentiomètre s'étale de 0 à 1023 ce qui rend la combinaison impossible à deviner. Pour faire correspondre ces valeurs à la plage de 0 à 9, utilisez la fonction `map()`.

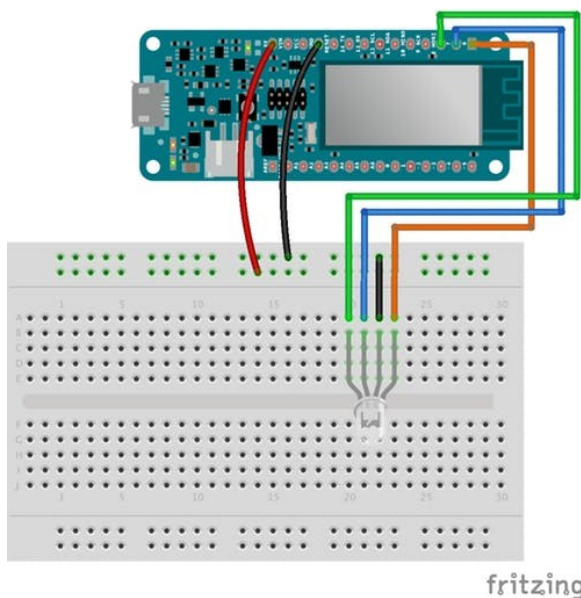
```
int PotOne = map(analogRead(A0), 0, 1023, 0, 9);
```

10) Chargez le code du fichier `puzzle-3.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin d'afficher sur l'écran LCD les valeurs du potentiomètre.

11) Ajoutez la LED RGB. Vous allez utiliser la LED RGB comme rétroaction pour aider les gens à deviner la combinaison, plus ils s'approchent de la bonne valeur, plus la couleur est chaude, allant du bleu, au jaune, puis au rouge.



G = ground

LED pinout

12) Chargez le code du fichier `puzzle-4.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin d'actionner la LED RGB.

13) Connectez la carte MKR1000 à Blynk, les potentiomètres à l'écran LCD et faites clignoter la LED en vert lorsque la combinaison est correcte.

14) Utilisez la fonction `giveColorFeedback()` pour régler la couleur de la LED lorsque la valeur absolue de chaque potentiomètre est plus proche qu'un certain seuil de la combinaison correcte.

```
void giveColorFeedback(int PotOne, int PotTwo, int PotThree){...}
```

Utilisez les variables suivantes pour stocker les valeurs envoyées de l'application et donc la combinaison :

```
int SliderValueOne = 1;  
int SliderValueTwo = 1;  
int SliderValueThree = 1;
```

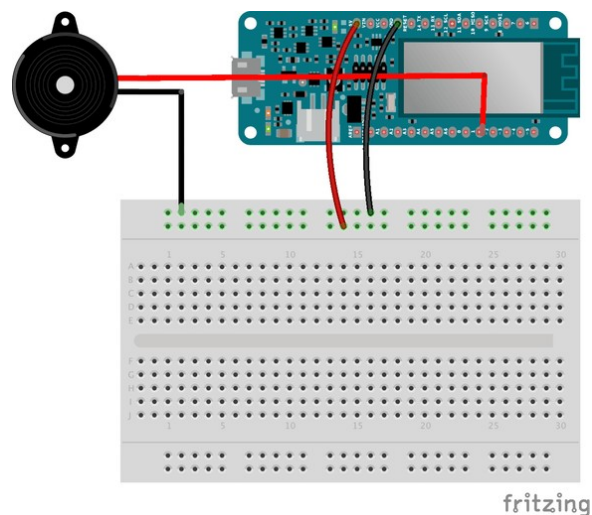
Les valeurs initiales sont fixées à 1. Elles ne changeront que si vous modifiez les valeurs des sliders dans l'application. Si vous réinitialisez la carte, la combinaison va revenir à sa valeur initiale.

Une variable booléenne `start = true;` est utilisée pour détecter si une combinaison a déjà été devinée.

15) Chargez le code du fichier `puzzle-5.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>
dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de le voir fonctionner.

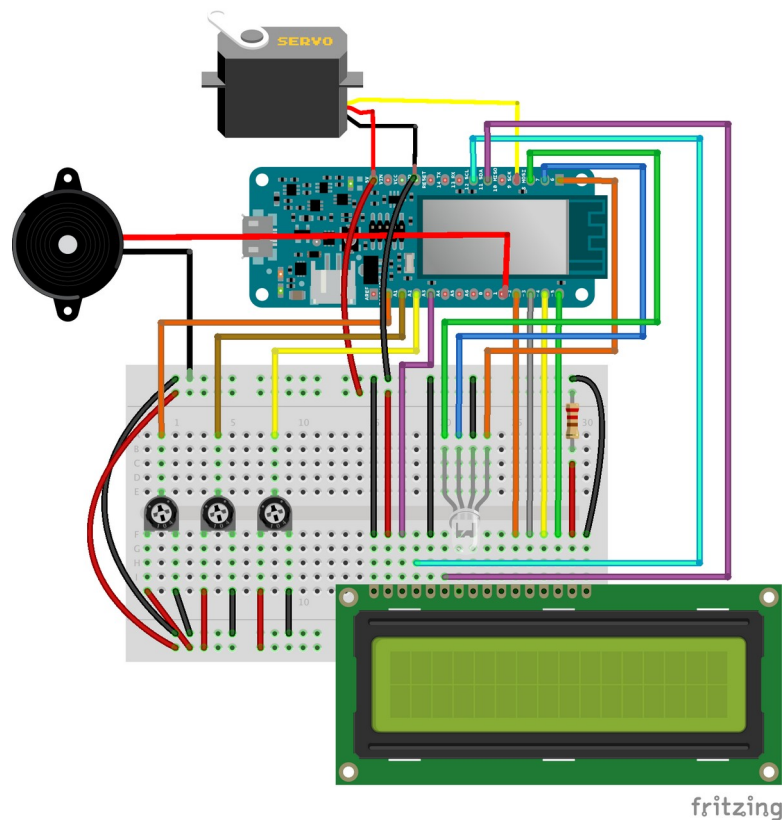
16) Ajoutez le buzzer. Utilisez le buzzer pour jouer une mélodie lorsque la combinaison est trouvée. Connectez le buzzer de la façon suivante :



17) Chargez le code du fichier `puzzle-6.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>
dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de tester le buzzer.

Le montage final doit être le même que celui de la figure ci-dessous sans le servo-moteur.



Le sketch final complet et les fichiers annexes sont dans l'archive ci-dessous :
http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/2_Puzzle_Box.zip

4. Tamagochi IoT : le 'Nerd'

Le *Nerd* est un tamagochi électronique sans fil qui survit en collectionnant des SSIDs de réseaux WiFi ainsi que du repos et de la lumière.

Afin qu'il se développe bien, il faut équilibrer les modes *offline* et *online* avec lumière et ténèbres pour s'assurer qu'il ait une routine manger-dormir journalière correcte.

Si il est hors de portée WiFi pendant trop longtemps, il va communiquer un SOS en code Morse en utilisant un haut-parleur piezo-électrique. Plus longtemps il est *offline*, plus il va beeper.

Le Nerd se réveille chaque demi-heure pour scanner les réseaux autour de lui. Si il détecte des nouveaux réseaux, il les stocke et retourne dormir en mode basse consommation (pour économiser la batterie). Sinon il va râler en faisant du bruit avec le buzzer jusqu'à que vous le nourrissiez ou que vous le mettiez dans le noir.

Il va aussi reconnaître lorsqu'il est à la maison en se connectant à votre réseau WiFi. De là il va se connecter à internet et récupérer l'heure et la date actuelles.

Si il n'est pas nourri pendant plus de deux jours, il va mourir dramatiquement en faisant beaucoup de bruit.

Les composants utilisés et leur montage pour créer le Nerd sont les suivants :

- RGB LED
- Phototransistor
- Buzzer
- Batterie
- Résistance de 220 Ohm

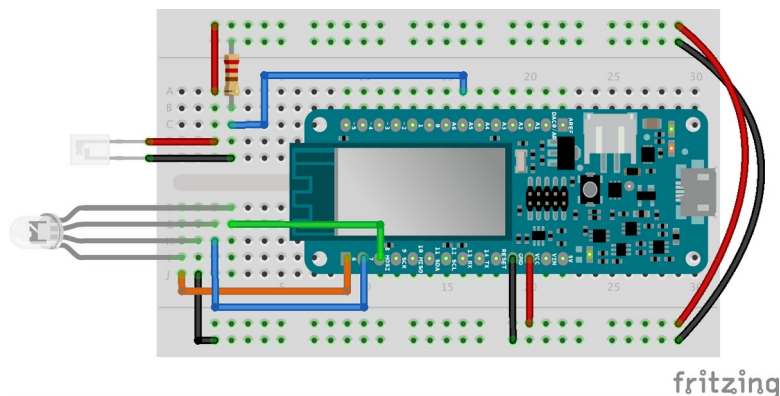


Schéma de câblage complet

1) Scannez les réseaux WiFi.

Chargez le code du fichier `nerd-1.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>
dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de tester le scan.

Vous pouvez aussi voir dans **File > exemples > WiFi101 > ScanNetworksAdvanced** pour une version plus étendue de cet exemple.

2) Stocker des valeurs en mémoire Flash.

Afin d'éviter que le Nerd meure lorsque la batterie est vide, il faut sauvegarder des variables (e.g., quantité de nourriture) dans la mémoire Flash afin qu'elles puissent être récupérées même après que la carte ait été éteinte puis allumée.

Afin de comprendre les fonctionnalités basiques vous pouvez tester : **File > Exemples > FlashStorage > FlashStoreAndRetrieve**.

Sauvegardez les noms des réseaux afin que le Nerd ne les mange qu'une seule fois. Utilisez le tableau qui stocke ces SSIDs afin de compter la quantité de nourriture qu'il a mangé durant la journée.

Les valeurs sauvegardées en mémoire Flash survivent à un *reset* de la carte mais pas à l'upload d'un nouveau sketch (i.e., chaque fois que vous uploadez un nouveau sketch, la mémoire flash est effacée aussi).

Chargez le code du fichier `nerd-2.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>
dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de scanner les réseaux WiFi et de stocker les SSIDs dans la mémoire Flash de la carte.

Notez que l'on fixe un nombre maximum de réseaux pouvant être sauvegardés afin d'éviter les problèmes d'espace mémoire. On stocke aussi la prochaine position disponible dans le tableau (i.e., quantité de nourriture déjà mangée) :

```
#define MaxNet 30
int PosToBeSaved = 0;
```

3) Gestion du temps.

Vous pouvez combiner les fonctionnalités de l'horloge temps-réel (**Real Time Clock (RTC)**) avec le WiFi pour obtenir la date et l'heure courante et ensuite fixer l'horloge interne de la carte. De cette manière vous pouvez déclencher des événements sur une période long sans avoir à utiliser la fonction `millis()` qui peut être délicate à utiliser lorsqu'il faut convertir des millisecondes en jours.

Il faut récupérer la date depuis un serveur de temps **Network Time Protocol (NTP)** puis régler le RTC avec. Le temps reçu est un temps UNIX, au format *Epoch*, qui est le nombre de secondes depuis le 1^{er} janvier 1970.

Vous pouvez tester l'exemple suivant : **File > Examples > WiFi101 > WiFiUdpNtpClient.**

- `bool atHome = false` :est utilisé pour déclencher la connexion WiFi et la requête au serveur pour obtenir la date et l'heure.
- `check_home()` :est utilisé pour scanner tous les réseaux disponibles et voir si l'un d'entre eux est le réseau WiFi de la maison.
- `connect_WiFi()` : si le réseau n'est pas celui de la maison, cette fonction va connecter la carte au WiFi, déclencher une requête au serveur et afficher/imprimer la date courante.
- `rtc.setEpoch(epoch + GMT)` : est utilisé pour initialiser la RTC avec la date courante au format *Epoch*. Modifiez la variable GMT pour ajuster l'heure à notre fuseau horaire.

Chargez le code du fichier `nerd-3.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de gérer les périodes de temps entre les scans des réseaux WiFi.

Obtenir la date permet de gérer facilement la vie et la mort du Nerd :

```
if(rtc.getEpoch() - values.last_time_fed >= 86400*2){
  // complain and eventually die :(
}
```

4) Mode basse puissance/consommation.

Utilisez ce mode pour mettre la carte en sommeil. Cela signifie que la carte va désactiver la plupart de ses fonctionnalités (dont le WiFi) pour économiser de la batterie.

Comme le Nerd doit se réveiller régulièrement, fixez un temporisateur (i.e., une minuterie).

Chargez le code du fichier `nerd-4.ino` situé ici :

<http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/>

dans l'Arduino IDE puis uploadez le code dans le MKR1000 afin de gérer les périodes de temps entre les scans des réseaux WiFi.

Notez que la fonction `WakeUp()` est attachée à l'interruption, ce qui signifie **qu'elle ne peut contenir aucun code incluant des délais**. Mais vous pouvez fixer une variable booléenne pour déclencher des événements dans la boucle.

Le sketch final complet et les fichiers annexes sont dans l'archive ci-dessous :
http://dept-info.labri.fr/~magoni/risc/TD-ARDUINO/4_The_Nerd.zip

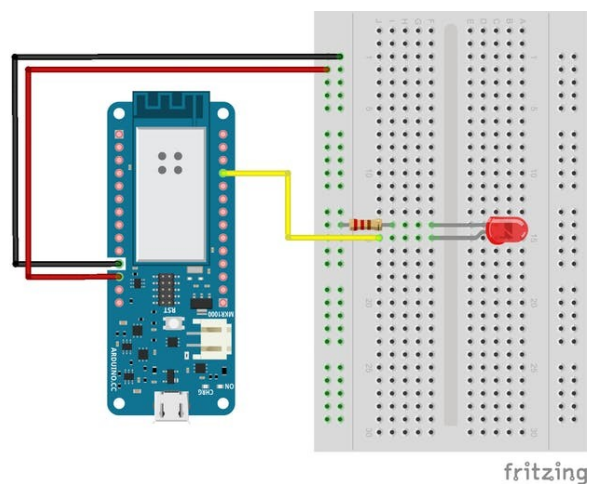
5. IoT Cloud

Cette manipulation va utiliser le service IoT fourni par le Cloud d'Arduino. Vous allez connecter la carte MKR1000 à l'*Arduino IoT Cloud* via WiFi. Vous contrôlerez et surveillerez ensuite la carte via Internet en utilisant le site *Web Arduino IoT Cloud*.

Pour cela, la carte sera ajoutée à l'Arduino IoT Cloud en tant que *Thing* - une représentation de la carte dans le cloud. Ensuite vous allez donner un ensemble de propriétés à votre *Thing* qui représente des capteurs, des LEDs, des moteurs et beaucoup d'autres composants que vous souhaitez accéder depuis le cloud.

1) Créez une *Thing* et contrôlez une LED via le Cloud.

Créez un circuit simple comprenant une LED branchée à l'Arduino MKR1000. Connectez la **patte ositive** de la LED au **Digital Pin 2** de la carte et la **patte négative** à la masse via une résistance de **150 Ohm**. Notez bien que l'alimentation de la plaquette d'expérimentation (*breadboard*) provient de **Vcc**, pas du pin **5V** de la carte MKR1000. Si vous la connectez au pin **5V**, vous pouvez endommager la carte durant les étapes suivantes, lorsque vous ajouterez les boutons poussoirs (*push button*).



LED connectée au pin digital 2.





Maintenant que la LED est branchée, il faut la rendre *IoT-enabled*. Avant cela, il faut [configurer la carte Arduino](#) pour lui permettre de communiquer avec le cloud. **Il faut créer un compte**. Une fois la carte configurée, allez à [Arduino IoT Cloud](#) et suivez le processus guidé **Getting Started** afin de configurer la carte, lui donner un nom et installer les clés de chiffrement.


IoT CLOUD BETA

YOUR THINGS

Arduino IoT Cloud allows you to connect devices to the internet and to other devices. This tool makes the creation of connected objects quick, simple and secure. [Read more...](#)

[NEW THING](#)

Thing	Properties	Widgets
	temperature read only	26.15
	state read & write	<input checked="" type="checkbox"/> ON
	position read & write	




IoT CLOUD BETA


CREATE NEW THING

Things are the logical representation of a connected object, they represent the inherent properties of the object, with as little reference to the actual hardware used to implement them. Each thing is represented by a collection of properties (e.g. temperature, light).


Select a board to configure for Create IoT Cloud:



Set up a MKR1000




Set up a MKR WiFi 1010



GETTING STARTED

WELCOME TO MKR1000 IOT CONFIGURATION!



This is the start of the process to configure your MKR1000. It will involve the following:

- Installing the Arduino Create Plugin, if you haven't already. This will make communication possible between your computer and Arduino boards.
- Configuring your board to be attached to a Thing in the Arduino Create IoT Cloud. Your device will then be listed in the Arduino Create Device Manager.


[START](#)

GETTING STARTED

CONNECT YOUR MKR1000 TO YOUR COMPUTER

SETUP STEPS

1. CONNECT VIA USB
2. NAME BOARD
3. CONFIGURE BOARD
4. CONGRATULATIONS!



Please connect your MKR1000 board to your computer using a USB micro cable.

Waiting for your board to become available...

GETTING STARTED

GIVE YOUR BOARD A NAME

SETUP STEPS

- ✓ 1. CONNECT VIA USB
2. NAME BOARD
3. CONFIGURE BOARD
4. CONGRATULATIONS!

Enter a name for your MKR1000 that you'll be able to recognize.

Name of Board?

NEXT

GETTING STARTED

CONFIGURE YOUR BOARD TO BE IOT-READY

SETUP STEPS

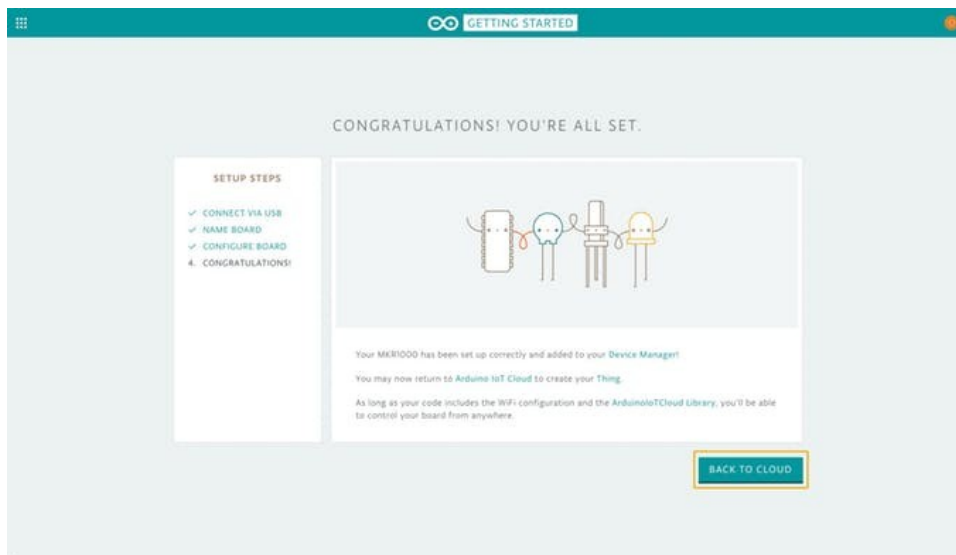
- ✓ 1. CONNECT VIA USB
- ✓ 2. NAME BOARD
3. CONFIGURE BOARD
4. CONGRATULATIONS!

We will now configure the Crypto Chip on your board.

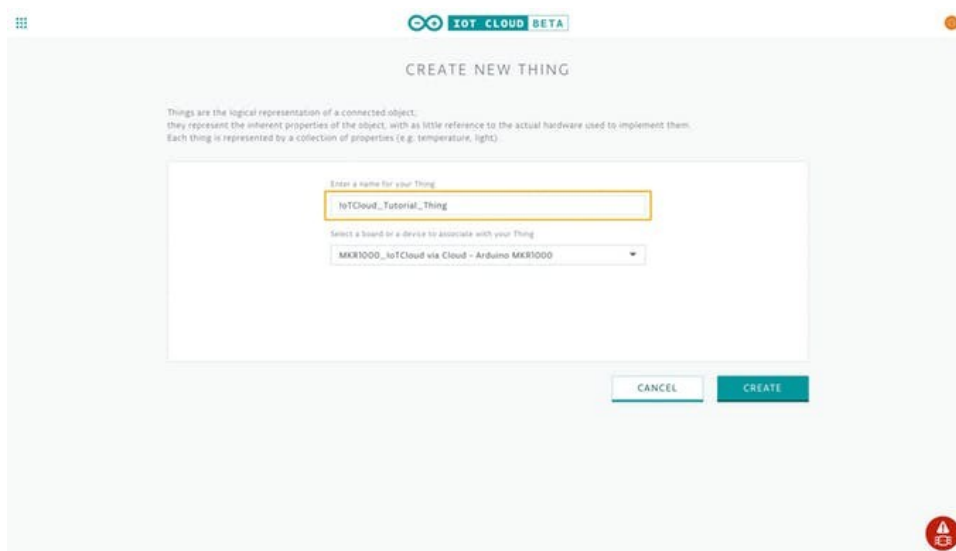
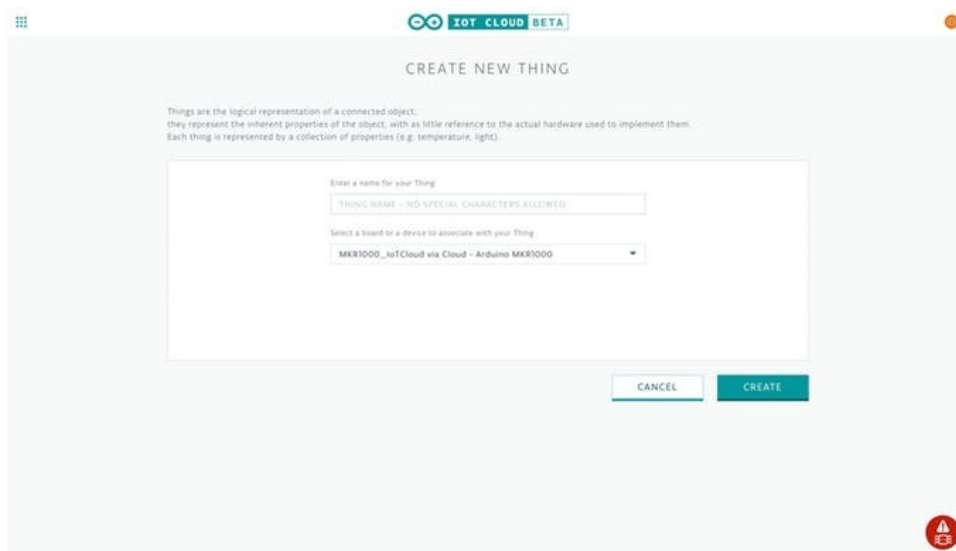
This will allow your MKR1000 to communicate securely with the Arduino IoT Cloud, and you'll be able to use the Network Monitor.

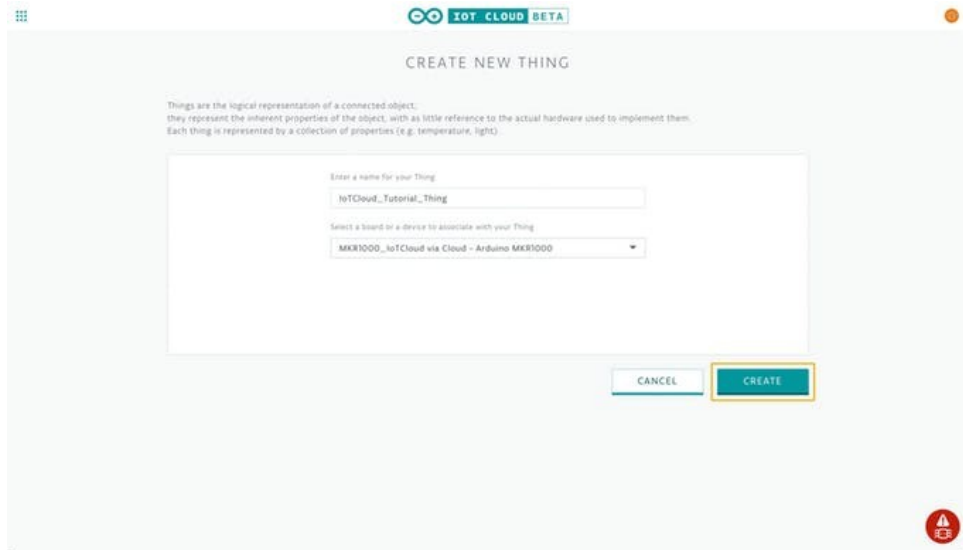
If you don't configure the Crypto Chip at this time, you won't be able to complete this setup.

NO THANKS CONFIGURE

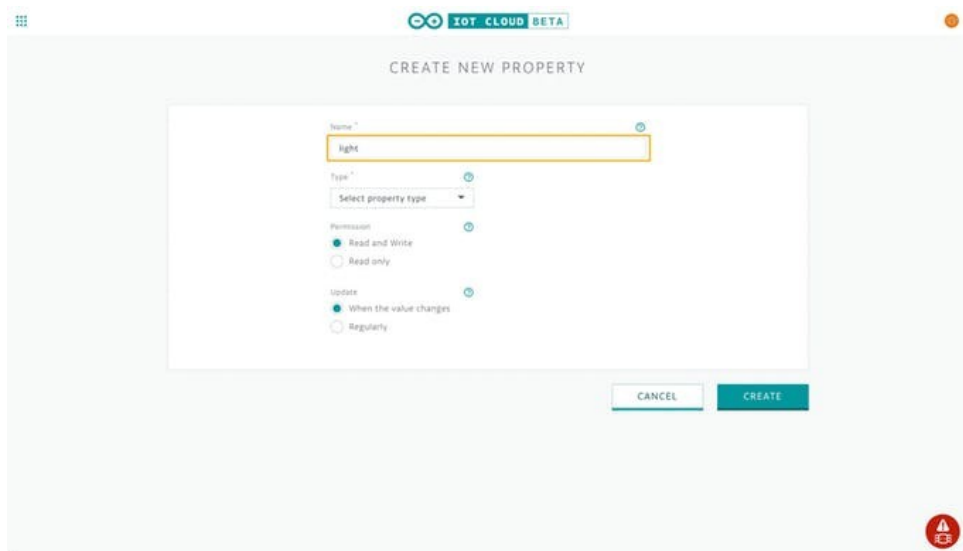
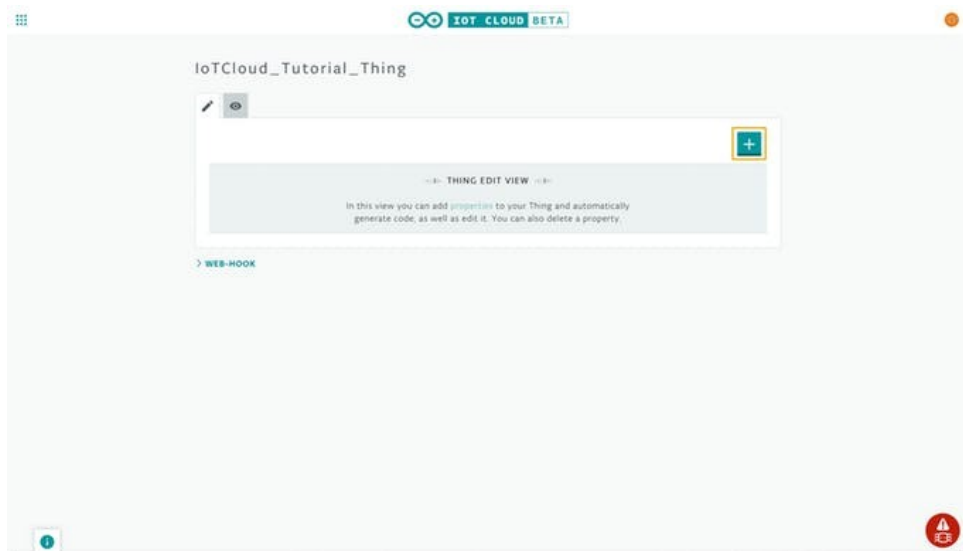


En cliquant sur le bouton "**BACK TO CLOUD**" vous pourrez créer votre première **Thing**. La carte juste configurée sera automatiquement sélectionnée pour être associée avec votre **Thing**, il ne reste plus qu'à la nommer. Dans l'exemple ci-dessous elle est nommée **IoTCloud_Tutorial_Thing**.

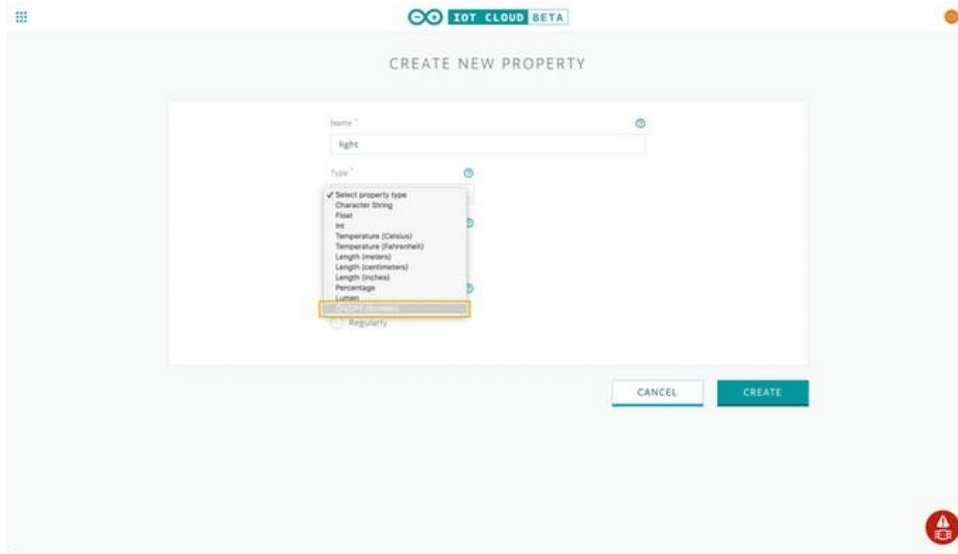




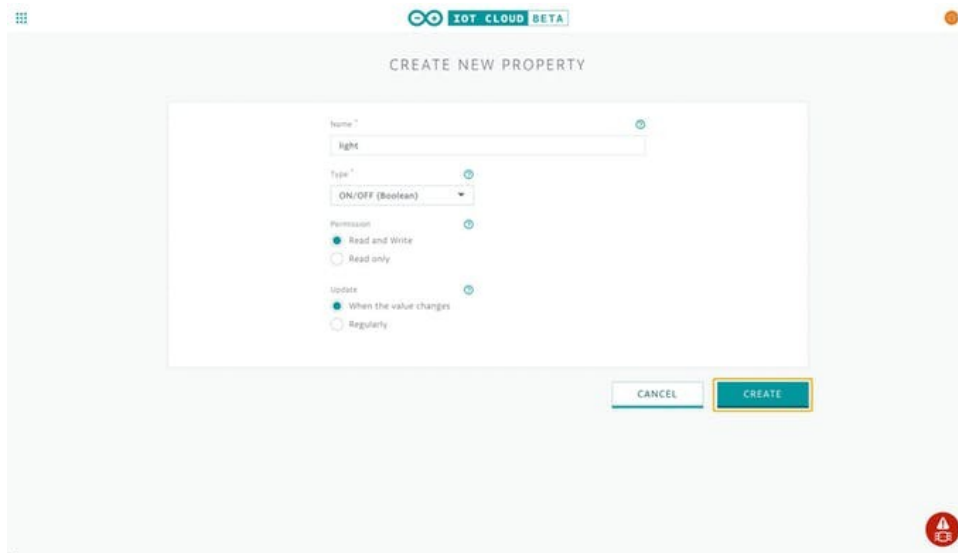
Vous êtes redirigés à la vue édition de votre **Thing**, où vous pouvez créer et modifier ses **Properties** - la représentation des capteurs et actuateurs que vous voulez accéder via le cloud. On veut allumer/éteindre la LED (**ON** et **OFF**) via le Cloud en basculant un commutateur graphique dans le *browser*. Pour cela il faut créer une **Property** en cliquant sur le bouton + comme montré ci-dessous.



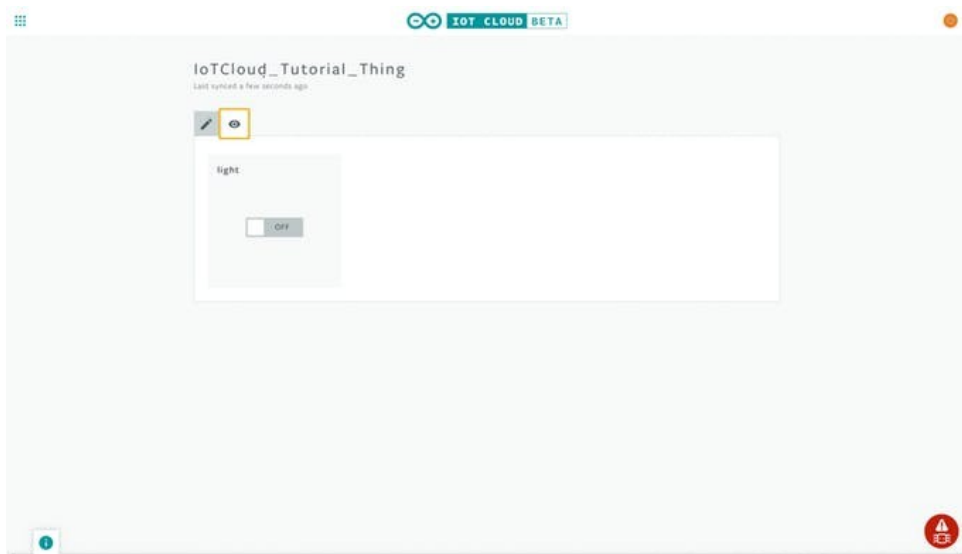
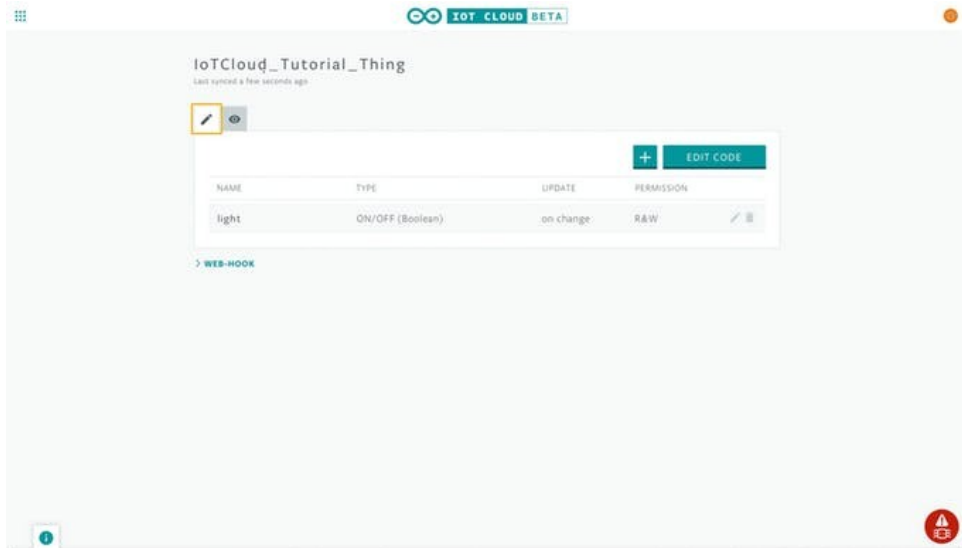
Utilisez **light** comme nom, il sera utilisé comme nom de variable dans le sketch.



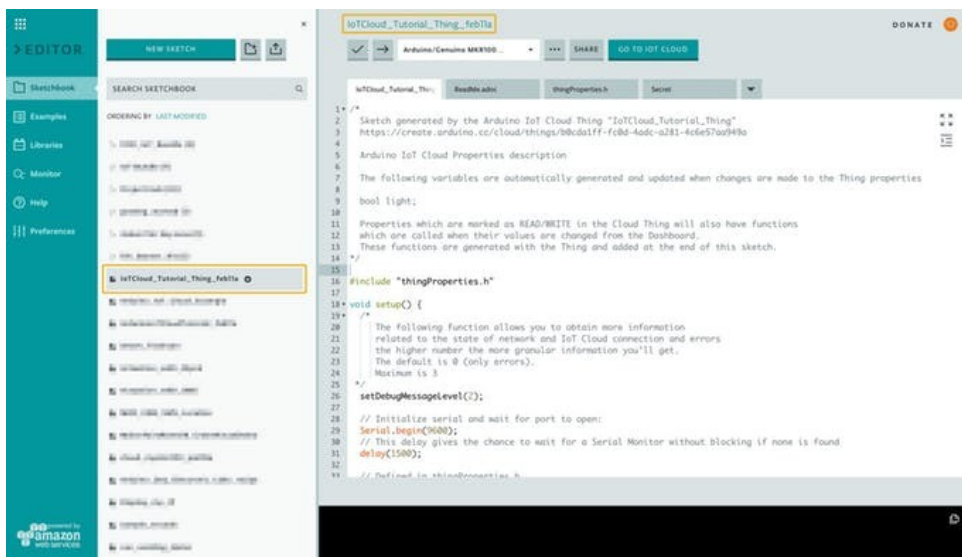
Le **Type** de cette propriété doit être "**ON/OFF (Boolean)**".



Laissez la **Permission** à "**Read and Write**" afin de pouvoir faire **ON** et **OFF** sur la LED depuis le browser. Laissez **Update** à "**When the value changes**", afin que les changements de valeur de la propriété/variable dans le sketch soient envoyés au Cloud. Puis cliquez sur **CREATE**.

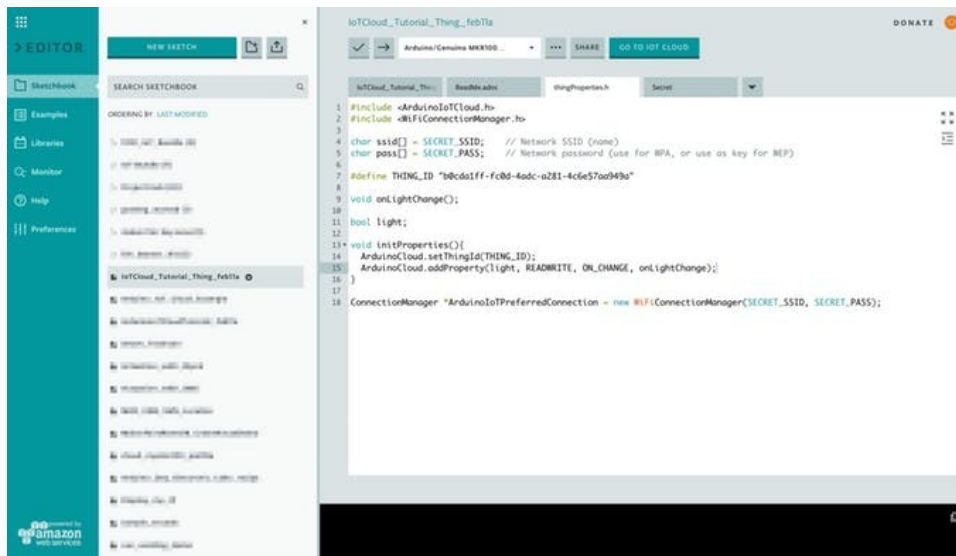


Depuis la vue **Edit** cliquez sur **"EDIT CODE"**. Cela vous renvoie à l'**Editor** montrant un sketch prêt à l'emploi généré pour votre **Thing**.

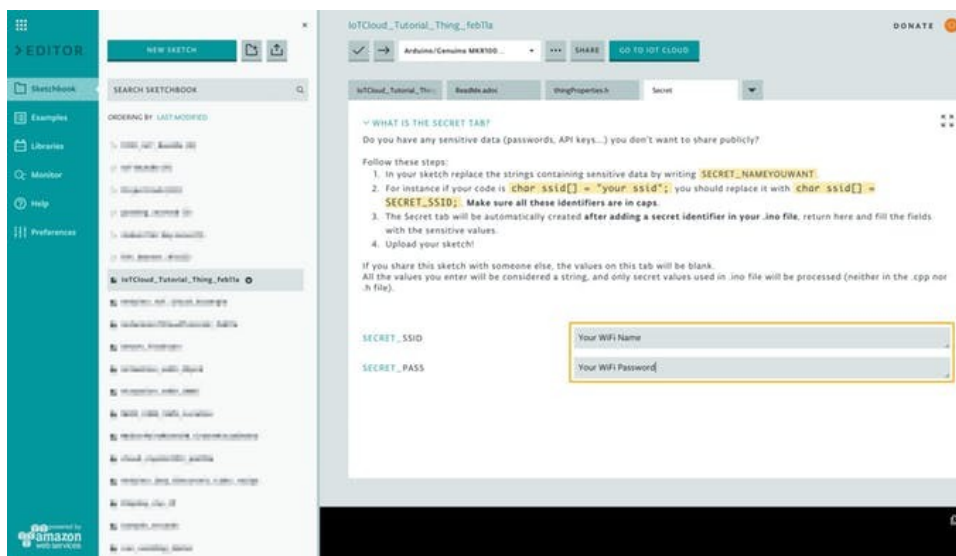


Le sketch va être nommé comme votre *Thing*, plus la date de création et éventuellement un nombre si une sketch de même nom existe déjà. En plus du fichier **.ino** montré ci-dessus, trois autres fichiers sont créés :

- **ReadMe.adoc**: fichier texte contenant des informations sur le sketch, l'auteur et le projet.



- **thingProperties.h**: ceci est du code généré par l'Arduino IoT Cloud lorsque vous avez ajouté la propriété `Light`. On peut y lire quelles variables dans le sketch (**.ino**) seront synchronisées avec le cloud.



- **Secret**: ce fichier permet de remplir les valeurs pour `SECRET_SSID` et `SECRET_PASS`, qui sont le nom et le mot de passe du réseau WiFi auquel se connecte la carte.

La signification du code du sketch est la suivante :

thingProperties.h

```
#include <ArduinoIoTCloud.h>
```

Importe la bibliothèque `ArduinoIoTCloud`, nécessaire pour synchroniser vos **variables** locales du sketch avec leurs propriétés dans le Cloud IoT.

```
#include <WiFiConnectionManager.h>
```

Le **WiFiConnectionManager** est utilisé pour gérer la connexion WiFi.

```
char ssid[] = SECRET_SSID;  
char pass[] = SECRET_PASS;
```

Ces valeurs sont extraites du *Secret tab*.

```
#define THING_ID "d276ab77-67cb-420b-9ea4-bd34cdf385d9"
```

L'ID unique du Thing.

```
void onLightChange();
```

Cette ligne déclare une fonction qui est appelée à chaque fois que la valeur de la propriété `light` est changée dans le tableau de bord (*Dashboard*). Ce type de fonction s'appelle une **callback**.

```
bool light;
```

Declaration de la variable `light`.

```
void initProperties()
```

Cette fonction sera appelée à l'intérieur du bloc `setup()` du fichier ***.ino**.

```
ArduinoCloud.setThingId(THING_ID);
```

Indique au sketch à quelle **Thing** se connecter.

```
ArduinoCloud.addProperty(light, READWRITE, ON_CHANGE, onLightChange);
```

Indique au sketch de traiter la variable `light` comme une **Property** de votre **Thing**, et d'exécuter la fonction **callback** `onLightChange` chaque fois que la valeur de la propriété est changée depuis l'Arduino IoT Cloud. Les permissions sont fixées à `READWRITE` pour cette propriété car c'est ce qui a été sélectionné à la création de cette **Property**.

```
ConnectionManager *ArduinoIoTPreferredConnection = new  
WiFiConnectionManager(SECRET_SSID, SECRET_PASS);
```

Initialisation du Connection Manager en utilisant le nom (**SECRET_SSID**) et le mot de passe (**SECRET_PASS**) du *Secret tab*.

***.ino**

Comme dans tout Sketch Arduino, il y a deux fonctions principales, `void setup(){...}` et `void loop() {...}`. Le `setup()` est appelé une seule fois au démarrage. La `loop()` est exécutée continuellement tant que la carte est alimentée.

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

Importe les variables et les fonctions déclarées dans **thingProperties.h** ainsi que les autres bibliothèques importées.

```
setDebugMessageLevel(2);
```

Fixe le niveau désiré des messages de *log* qui seront imprimés sur le *Serial Monitor*. Il peut être changé de **0** (qui logue seulement les erreurs) jusqu'à **3** (qui logue tout).

```
Serial.begin(9600);
```

Initialise le **Serial Monitor**, pour y imprimer et y lire.

```
delay(1500);
```

Attend 1,5 seconds pour donner au Serial Monitor le temps de s'initialiser.

```
initProperties();
```

Initialise les propriétés telles que définies dans **thingProperties.h**.

```
ArduinoCloud.begin(ArduinoIoTPreferredConnection);
```

InitializetheArduinoCloud en utilisant le **ConnectionManager**.

Dans la `loop()` vous trouvez :

```
ArduinoCloud.update();
```

Cette fonction gère la synchronisation des valeurs des **properties** entre le cloud et la carte, gère la connexion vers le cloud, etc. Si la valeur d'une propriété change dans le sketch, la librairie va automatiquement le détecter et notifier le Cloud, afin que la valeur soit modifiée dans le tableau de contrôle de l'Arduino IoT Cloud. De même, lorsque la valeur d'une propriété est changée dans le *Dashboard*, la librairie va mettre à jour la valeur correspondante sur la carte.

```
void onLightChange() {...}
```

L'implémentation de la callback qui sera appelée à chaque fois que la valeur de la propriété `light` change.

Pour allumer (ON) et éteindre (OFF) la LED depuis le *dashboard* Arduino IoT Cloud, il faut définir le pin auquel la LED est connecté. Ajoutez ce code en début de fichier :

```
#define LED_PIN 2
```

Dans la fonction `setup()`, initialisez le pin pour être un **OUTPUT**:

```
pinMode(LED_PIN, OUTPUT);
```

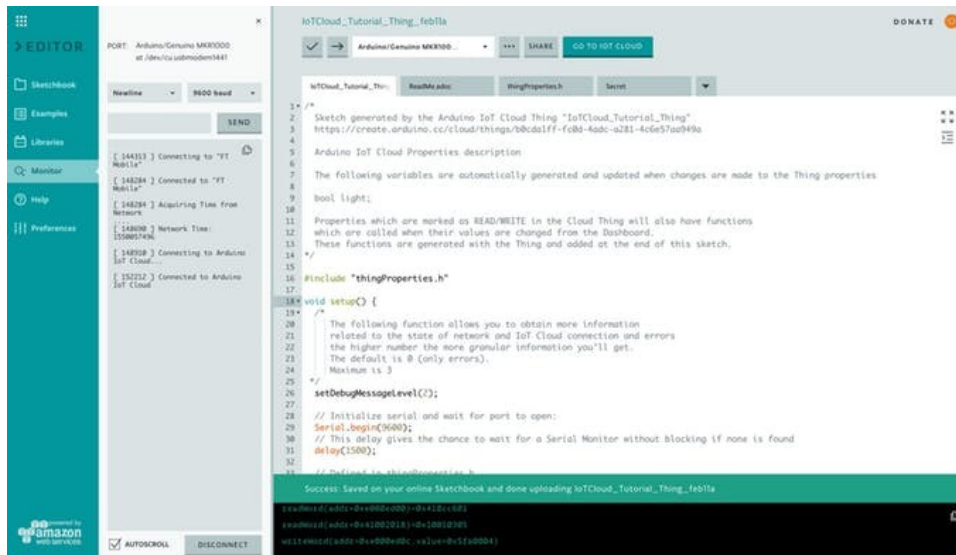
Dans la fonction `onLightChange()`, ajoutez du code pour suivre l'état `light` dans le *serial monitor* et pour contrôler la LED. Notez que cette fonction callback est automatiquement générée lorsqu'une nouvelle propriété est ajoutée avec les permissions **Read** et **Write**.

```
void onLightChange() {  
    digitalWrite(LED_PIN, light);  
    Serial.print("The light is ");  
    if (light) {  
        Serial.println("ON");  
    } else {  
        Serial.println("OFF");  
    }  
}
```


Vous pouvez uploader le sketch en cliquant sur le bouton **Upload**.

Puis ouvrez le **Serial Monitor** pour voir si tout fonctionne bien.

Parce le niveau de **logging** est fixé à 2, le Serial Monitor montre des informations sur la progression de la connexion de la carte à l'IoT Cloud.

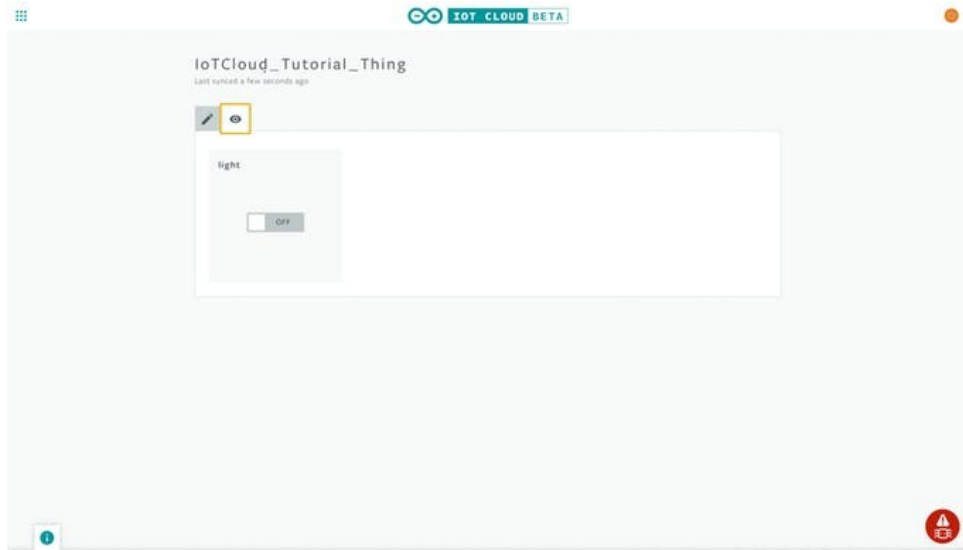


Une fois connecté au WiFi, avoir obtenu une adresse et initié un lien sécurisé, la carte va se connecter au IoT Cloud et va échanger des données avec celui-ci.

```
[ 144313 ] Connecting to "FT Mobile"
[ 148284 ] Connected to "FT Mobile"
[ 148284 ] Acquiring Time from Network
...
[ 148690 ] Network Time: 1550057496
[ 148910 ] Connecting to Arduino IoT Cloud...
[ 152212 ] Connected to Arduino IoT Cloud
```

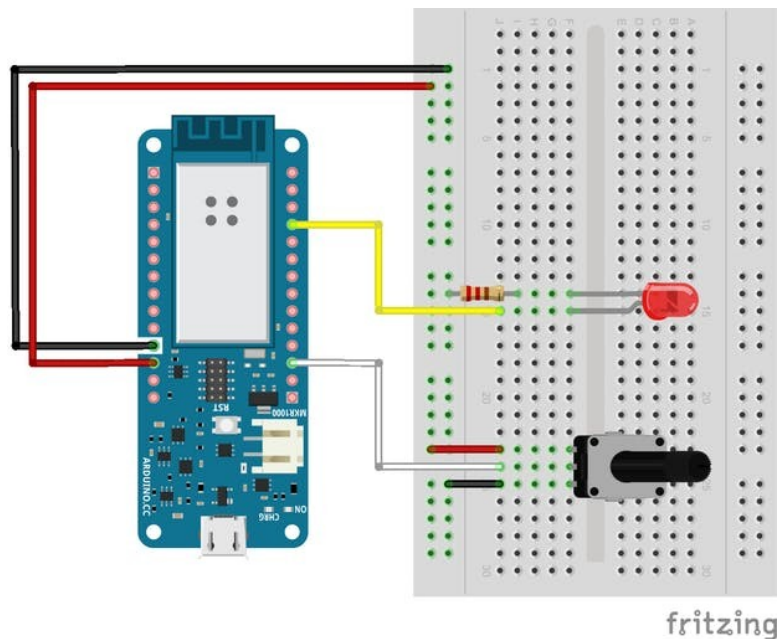
Si une étape quelconque échoue, une erreur est reçue. Vous pouvez alors faire un *reset* de la carte et réessayer.

Cliquez sur le bouton **GO TO IOT CLOUD** pour être redirigé vers la page du *Thing* sur l'Arduino IoT Cloud. De là, cliquez sur le bouton *dashboard*.

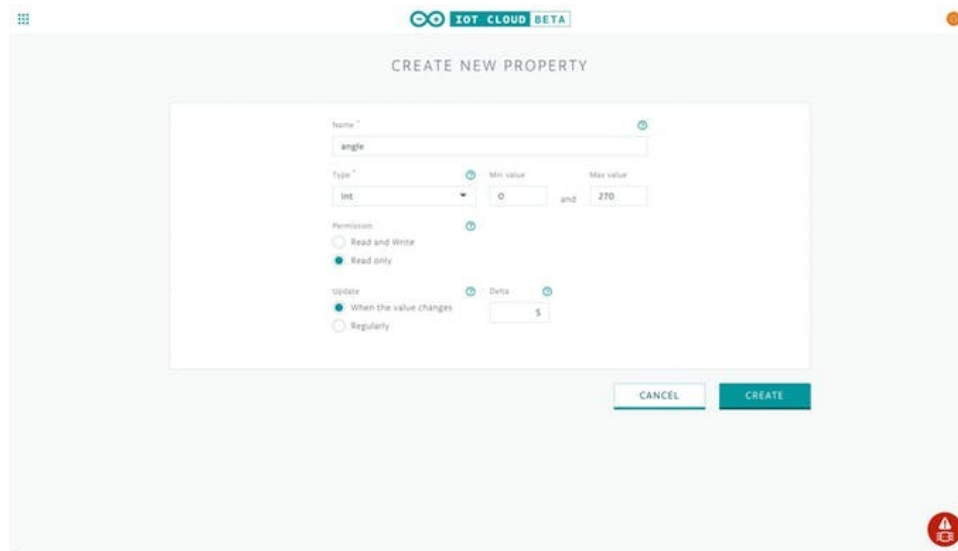


Vous devriez voir un *widget* montrant l'état de la propriété `light` et il doit être fixé à `Off`. Cliquez plusieurs fois pour voir la LED s'allumer et s'éteindre.

2) Ajoutez un potentiometre et faites le correspondre à une propriété `Int`. Liez une nouvelle propriété à un **potentiometre** qui doit être ajouté au circuit. Le potentiometre est connecté à l'alimentation et à la masse par ses pins respectifs et le pin de signal est connecté au pin **Analog A1** de la carte.



Dans la vue *Thing's properties*, cliquez sur le bouton `+` et créez une propriété nommée `angle`. Fixez son type à `Int` avec **Min Value** et **Max Value** fixés à `0` et `270` respectivement. La **permission** doit être fixée à **Read only** et la propriété doit être mise à jour **when the value changes**; vous pouvez fixer une valeur **Delta** supérieure à zéro si vous voulez introduire une tolérance dans la mise à jour (e.g.: si $\Delta = 5$, la valeur sera mise à jour via le cloud seulement si la différence entre la nouvelle et l'ancienne valeur est supérieure à 5).



Cliquez sur **CREATE** pour ajouter la nouvelle propriété à votre *Thing* et revenir à la vue *Property edit*.

Le sketch a été mis à jour, cliquez sur **EDIT CODE** pour revenir à l'éditeur. En regardant dans **thingProperties.h**, notez que deux lignes ont été ajoutées :

```
int angle;
```

Déclare la variable représentant la propriété que vous avez créée

```
ArduinoCloud.addProperty(angle, READ, ON_CHANGE, NULL, 5.000000);
```

Connecte la variable **angle** à sa propriété correspondante, avec permission **READ** (i.e. : elle ne pourra être modifiée depuis le tableau de bord). De part la permission **Read Only**, aucune fonction callback ne sera générée et le deuxième au dernier argument de la méthode **addProperty** seront fixés à **NULL**. Le dernier argument représente la valeur **Delta**.

Pour que le potentiometre interagisse avec le cloud, il faut définir le pin auquel il est connecté :

```
#define POTENTIOMETER_PIN A1
```

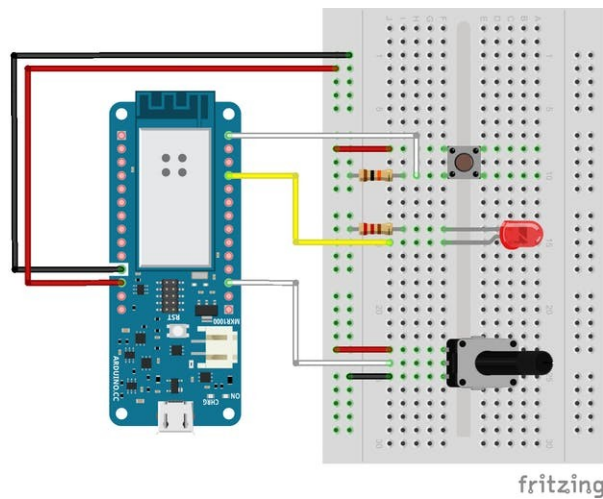
Puis dans la fonction **loop()**, lisez l'entrée analogique depuis le potentiomètre et faites le correspondre à la variable **angle**. Ainsi tourner le potentiometre change la valeur de la propriété correspondante dans le tableau de bord du cloud.

```
int angleSensor = analogRead(POTENTIOMETER_PIN);
angle = map(angleSensor, 0, 1023, 0, 270);
```

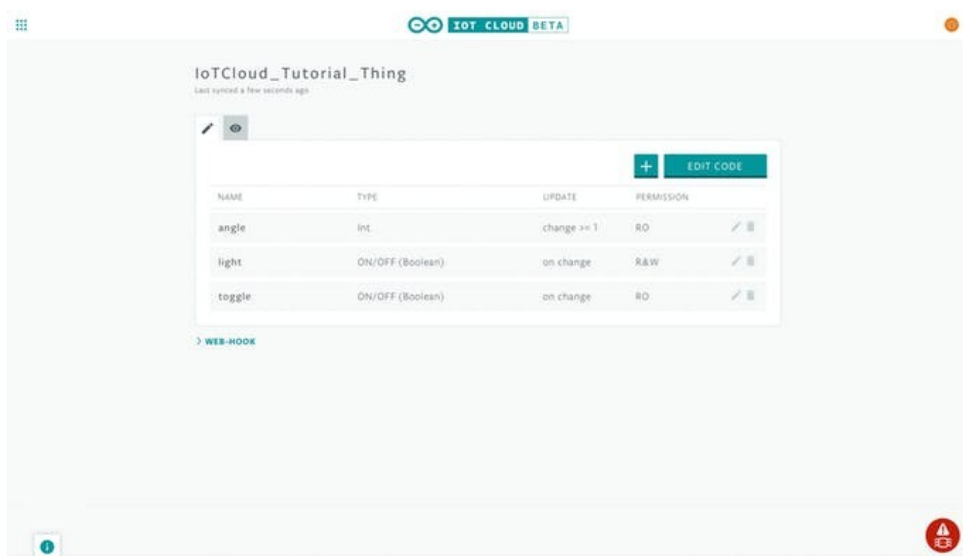
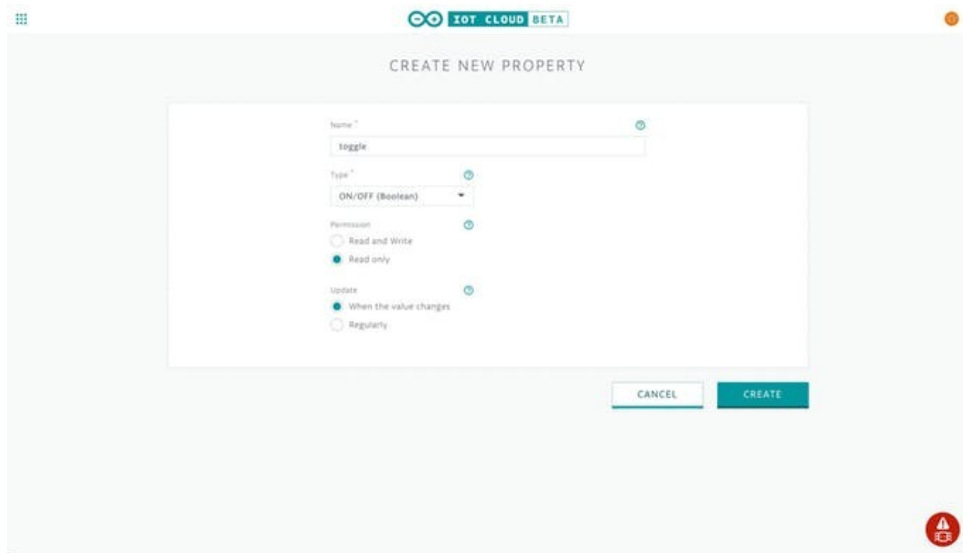
Uploadez le sketch et voyez se qui se passe sur le *thing's dashboard* lorsque vous tournez le bouton du potentiometre. Vous devez voir la valeur varier de 0 à 270.

3) Ajoutez un bouton poussoir et liez le à une propriété **Boolean**.

Ajoutez une nouvelle propriété associée à un bouton poussoir ajouté au circuit précédent comme montré sur le schéma ci-dessous : un pin du bouton est connecté au rail d'alimentation positif (Vcc), l'autre pin est connecté au pin **Digital 5** (via le fil blanc) et à la masse par une résistance **pull-down** de **10k**. Cette configuration forces le pin à un niveau logique bas (**LOW**) lorsque le bouton est au repos, et conduit **Vcc** au travers lorsqu'il est appuyé (niveau logique haut (**HIGH**)).



Depuis l'éditeur, allez au **IOT CLOUD** et créez une nouvelle propriété nommée `toggle`, avec un type **ON/OFF (Boolean)**, une permission à **Read only** et mise à jour **When the value changes**.



Faites **EDIT CODE** pour revenir à l'éditeur. Un coup d'oeil à **thingProperties.h** montre qu'une nouvelle variable `Toggle` a été définie et associée à sa propriété via `ArduinoCloud.addProperty(...)`.

Dans le fichier `.ino` définissez le nouveau new pin et deux variables reliées à l'état du bouton :

```
#define BUTTON_PIN 5
int btnState;
int btnPrevState = 0;
```

`btnPrevState` est requis car la propriété ne doit être mise à jour qu'une fois lorsque le bouton est appuyée et non lorsqu'il est relâché.

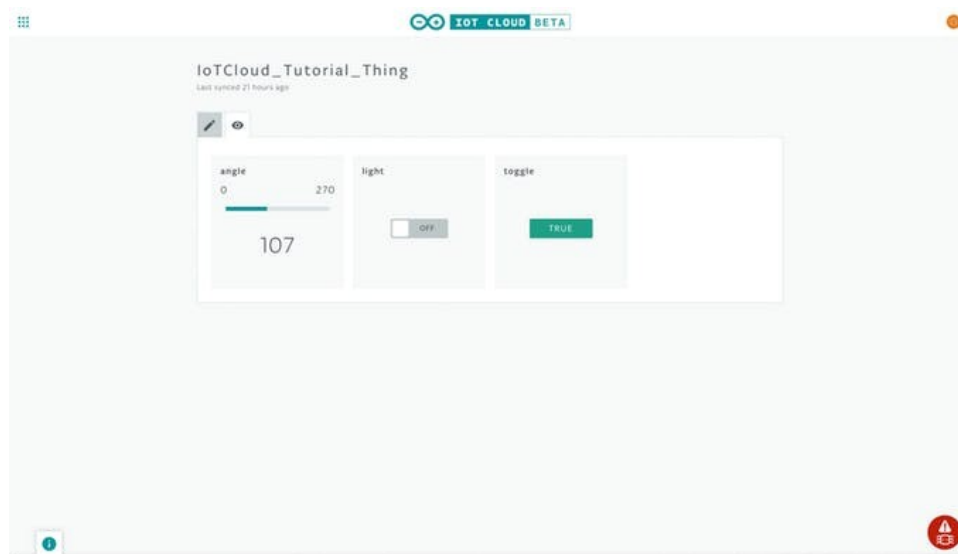
Dans `setup()` fixez le `pinMode` à `INPUT`

```
pinMode(BUTTON_PIN, INPUT);
```

Ajoutez ces lignes à la fin de `loop()`

```
btnState = digitalRead(BUTTON_PIN);
if (btnPrevState == 0 && btnState == 1) {
  toggle = !toggle;
}
btnPrevState = btnState;
```

De cette manière, le bouton agit comme un **commutateur (toggle)** et lorsque vous l'appuyez, vous devriez voir le commutateur du tableau de bord alterner entre **ON** et **OFF**. Testez votre montage.



6. References

- <https://create.arduino.cc/projecthub/arduino/puzzlebox-c1f374>
- <https://create.arduino.cc/projecthub/arduino/the-nerd-0144f9>
- <https://create.arduino.cc/projecthub/133030/iot-cloud-getting-started-c93255>