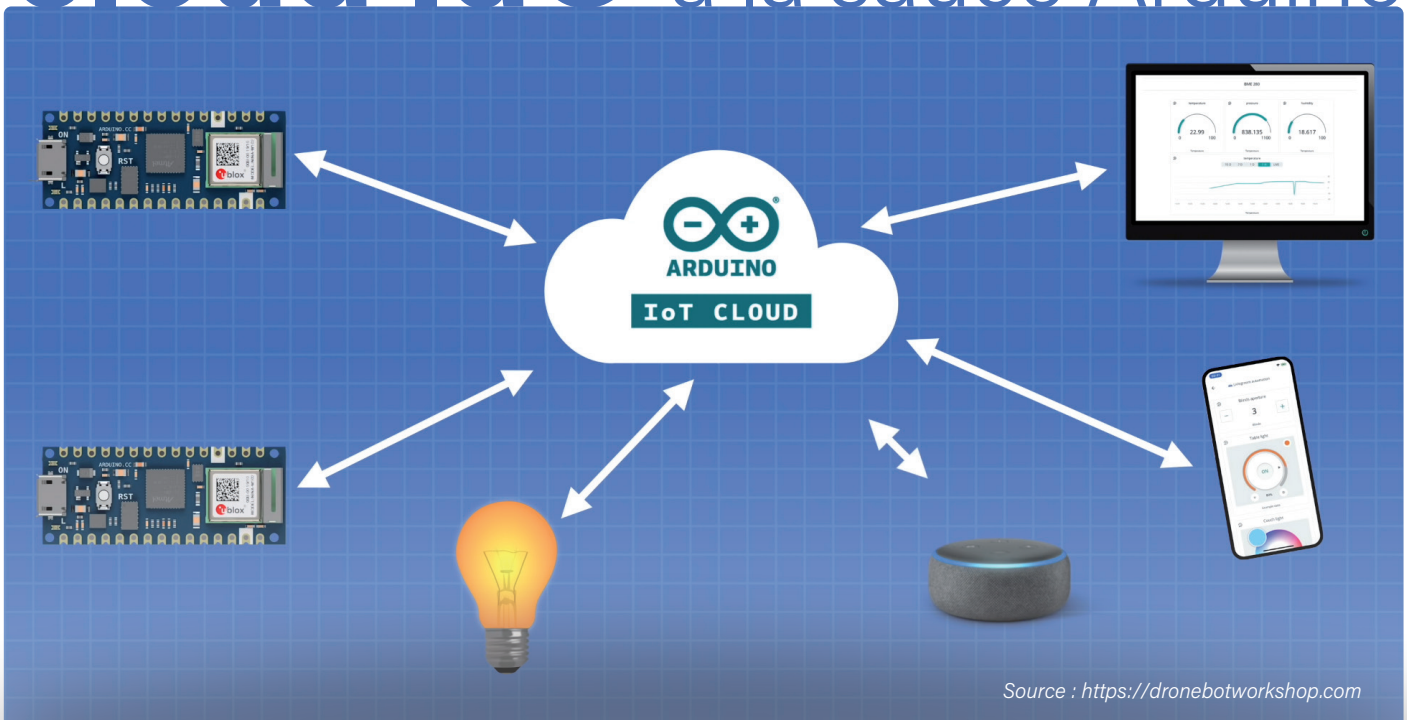
cloud IdO à la sauce Arduino



Source : https://dronebotworkshop.com

Tam Hanna (Slovaquie)

L'Arduino IoT Cloud offre aux développeurs d'applications IdO une solution pratique de mise en œuvre d'un serveur du nuage sans les inconvénients de MQTT. Intéressé ? Allons voir !

De nombreuses applications à microcontrôleurs s'articulent aujourd'hui autour de l'Internet des Objets (IdO) où des informations sont diffusées via des services du nuage IdO et un courtier MQTT. Créer ce type d'application sous un environnement de développement local comme l'EDI Arduino traditionnel peut parfois s'avérer délicat. La plateforme Arduino IoT Cloud transpose l'EDI dans le nuage de sorte que votre navigateur devient une fenêtre de l'EDI. Nous avons essayé cet environnement en envoyant une valeur variable au nuage pour faire clignoter une LED sur notre banc d'essai. Puis nous avons tenté de le prendre en défaut.

La base de tous les appareils IdO est bien sûr l'*Objet*. Dans l'environnement de développement Arduino IoT Cloud, l'*Objet* est un objet virtuel qui existe dans le cloud. Dans le monde réel, il s'agit d'un objet tel qu'un serveur, une carte à contrôleur ou un dispositif « intelligent » similaire [1]. Ici, votre *Objet* est construit dans le nuage avec un éditeur en ligne pour écrire un croquis qui décrit son comportement et ses réponses grâce à toute une série de *variables*.

Qui utilisera l'Arduino IoT Cloud ?

Avant de commencer, il est important de reconnaître que l'Arduino IoT Cloud n'est pas une alternative aux autres plateformes informatiques dédiées au nuage, telles qu'Amazon AWS IoT Core, Microsoft IoT Hub ou Yandex IoT Core. Avec un grand nombre de dispositifs et de données à gérer, ces services bien établis du nuage IdO sont plus indiqués.

Lors de l'introduction de la dernière mouture de l'Arduino IoT Cloud, Massimo Banzi, CTO d'Arduino, a exprimé ses ambitions pour la plateforme en disant que : « Arduino propose désormais une plateforme complète avec la famille MKR et rationalise la création des nœuds IdO locaux et des dispositifs de périphérie. Ceux-ci utilisent une gamme d'options de connectivité et de compatibilité avec du matériel tiers, des passerelles et des systèmes du nuage. L'Arduino IoT Cloud permet aux utilisateurs de gérer, configurer et connecter, non seulement le matériel Arduino, mais aussi la grande majorité des appareils basés sur Linux – démocratisant véritablement le développement IdO ».

La prise en charge de la gamme de cartes Arduino MKR dédiées à l'IdO et de quelques autres cartes tierces répandues est un plus bienvenu et devrait inciter de nombreux développeurs de systèmes IdO à reconsidérer cette plateforme de développement accessible.

Configuration du matériel

Seul le forfait le plus rudimentaire des quatre versions possibles de l'Arduino IoT Cloud est gratuit. Vous pouvez consulter les différents forfaits et leurs caractéristiques dans la **figure 1** et choisir celui qui répond le mieux à vos besoins. La communauté des makers est surtout intéressée par la prise en charge par l'Arduino IoT Cloud de cartes tierces, telles que celles des fameuses familles ESP8266 et ESP32 (**tableau 1**), et d'autres kits (voir [2]). Les bibliothèques de

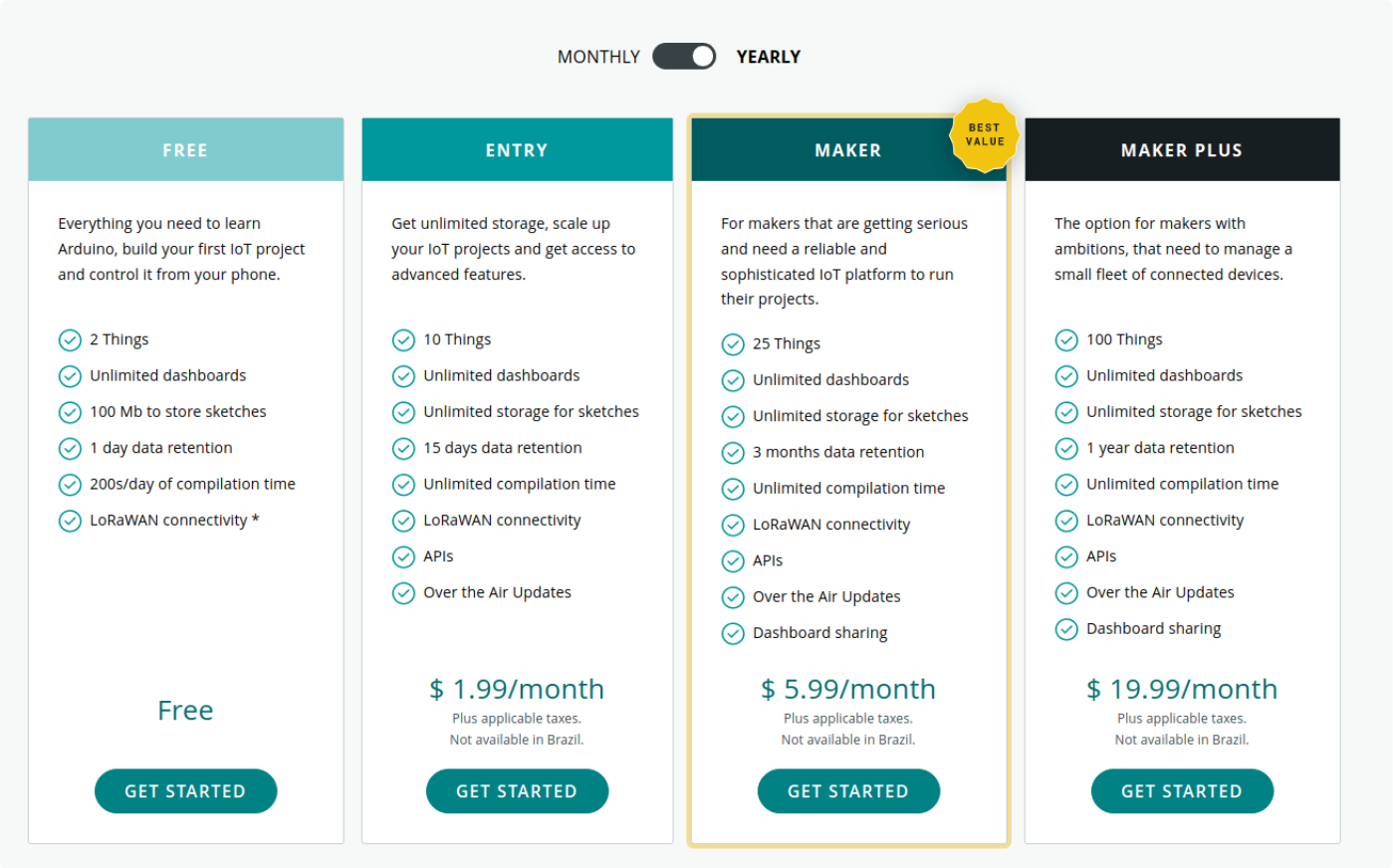


Figure 1. Quel est le meilleur forfait pour vous ? (Prix des forfaits en date du 20/01/2022).

pilotes permettent aussi à divers systèmes basés sur Linux d'envoyer et de récupérer des informations sur le nuage Arduino.

Pour tester les fonctions de l'Arduino IoT Cloud, on utilise ici comme carte cible un module Arduino Nano RP2040 Connect. Cette carte est basée sur le microcontrôleur RP2040 de la fondation Raspberry Pi et comprend un module Wi-Fi u-blox. Avant de configurer le nuage Arduino, la carte est connectée à un ordinateur via un câble USB.

Visitons d'abord le site web [3] pour créer un nouveau compte Arduino. Pour nos besoins, nous nous en tiendrons au forfait de base gratuit qui est explicitement destiné aux nouveaux venus dans le système. Commençons par cliquer sur le bouton *Create Thing* pour créer un nouvel objet.

L'aperçu de la **figure 2** montre alors la configuration des trois composants de base de l'environnement de développement. J'ai supprimé un précédent compte Arduino et tous les paramètres de configuration afin de commencer avec une configuration de périphérique vierge. Les étapes suivantes sont réalisées sur une machine sous Windows 10, mais le processus est identique sous Ubuntu (Linux), et j'ai constaté que la détection du matériel fonctionne en général mieux sous Unix.

Nous pouvons maintenant cliquer sur l'icône de raccourci dans la section *Device* et choisir *Set Up an Arduino Device*. Quelques secondes après avoir cliqué sur cette option, le serveur signalera l'absence du composant appelé *Arduino Create Agent*. Cliquez sur le bouton *Download* pour télécharger le logiciel et l'installer de la manière habituelle.

Notez que *Create Agent* est spécifique au navigateur : si vous l'installez sous Chrome, vous devrez le réinstaller. Vous devez tenir

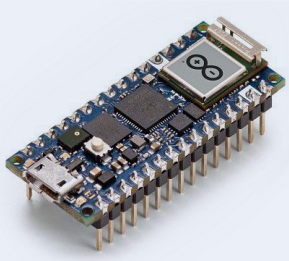
Tableau 1.
Cartes prises en charge par l'Arduino IoT Cloud.

WLAN
- MKR 1000 WiFi
- MKR WiFi 1010
- Nano RP2040 Connect
- Nano 33 IoT
- Portenta H7

LoRaWAN
- MKR WAN 1300
- MKR WAN 1310

GSM/NB-IoT
- MKR GSM 1400
- MKR NB 1500

ESP32/ESP8266
- Large gamme de cartes tierces



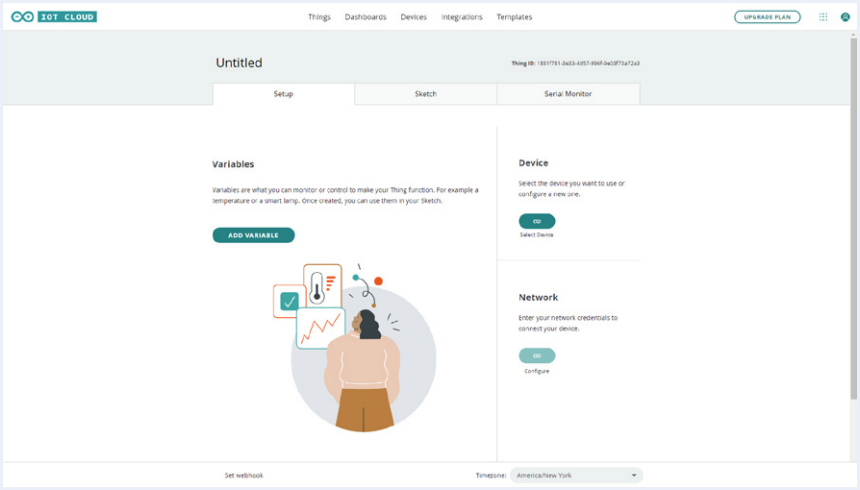


Figure 2. L'Arduino IoT Cloud guide le développeur pas à pas jusqu'à l'objectif.

Figure 3. Cet Arduino est connecté au nuage.

compte des avertissements du pare-feu qui s'affichent. Assurez-vous que vous autorisez l'accès aux réseaux privés et publics. *Arduino Create Agent* réside désormais dans votre barre de tâches – dans certains cas, il peut être nécessaire de le rappeler à partir du menu de démarrage, mais ainsi se termine l'installation du pilote.

Pour continuer, nous allons rafraîchir la vue jusqu'à ce que le nuage Arduino nous informe que notre carte Nano RP2040 Connect a été reconnue. Ensuite cliquez sur le bouton *Configure* pour lancer l'assistant de configuration – il vous demandera d'entrer un nom « convivial », puis il initialisera l'élément sécurisé du système cible avec le logiciel de communication de base.

Des problèmes surviennent parfois lors de l'affectation du réseau sous Windows. La méthode la plus fiable et la plus efficace est de le faire sous Linux. Par ailleurs, la section *Network* n'est pas automatiquement activée par Arduino IoT Cloud. Elle n'est disponible que lorsque vous créez l'une des variables destinées à l'échange de données.

Cliquons maintenant sur *Variables*, ce qui ouvre une fenêtre pour ajouter une nouvelle variable. Nous commençons par la nommer `ledIntenBool` en précisant *Boolean* dans le champ du type de données. Il est amusant de constater que l'Arduino IoT Cloud ne prend pas seulement en charge les unités de programmation en C, mais implémente également des encapsulations autour des variables du monde réel.

Si vous souhaitez vous limiter au C, nous vous recommandons de sélectionner l'option *Basic Types*. En théorie, nous pourrions alors définir des paramètres dans les champs *Variable Permission* et *Variable Update Policy*, mais les paramètres par défaut seront suffisants pour nos besoins, c'est pourquoi nous fermons maintenant la boîte de dialogue. En suivant, nous créons un champ de type *Integer Number*, que nous nommons `ledIntenInt`.

Préparation du code

Après avoir créé les variables, des points rouges dans l'onglet *Sketch* indiquent les changements apportés à la structure du programme. Il est maintenant possible de cliquer sur l'icône de raccourci dans la section *Network* pour saisir les paramètres Wi-Fi. Ma préférence est de saisir les valeurs à l'aide d'un outil en ligne de commande tel que *iwlist* sur une machine Linux, puis de les copier dans le presse-papiers.

Passons ensuite à l'onglet *Sketch* et cliquons sur le bouton *Verify and Upload*. L'Arduino IoT Cloud commence alors à compiler le code et l'envoie à la carte RP2040 connectée en utilisant l'*Arduino Cloud Agent*. Si le code compilé est chargé avec succès, le message « Untitled_dec25a uploaded successfully on board Arduino Nano RP2040 Connect (/dev/ttyACM0) » est affiché.

Après la nécessaire réinitialisation, le RP2040 va se connecter via son émetteur Wi-Fi. Après un certain temps et en appuyant plusieurs fois sur *F5*, l'appareil apparaîtra avec « Status: online » (**fig. 3**).

Si vous choisissez de vous abonner à l'un des forfaits Arduino IoT Cloud les plus complets, vous pouvez recevoir les mises à jour logicielles directement via le Wi-Fi ; mais pour nos expériences simples avec le forfait gratuit, il faut une connexion filaire.

Plus en détails

L'outil d'édition de base de l'onglet *Sketch* n'est pas très utile, mais en cliquant sur le bouton *Open full editor*, nous obtenons un EDI complet basé sur le nuage qui nous permet d'éditer de petits projets plus confortablement. Examinons d'abord le contenu du fichier *thingProperties.h*, qui contient les éléments structurels du croquis. Au début, nous voyons les déclarations suivantes qui fournissent les éléments requis pour l'accès Wi-Fi :

```
const char SSID[] = SECRET_SSID; // SSID du réseau (nom)
const char PASS[] = SECRET_PASS; // Mot de passe réseau
// (à utiliser pour WPA, ou comme clé pour WEP)
```

Arduino IoT Cloud se charge de saisir le nom et le mot de passe en utilisant les paramètres déjà saisis dans la section *Network*. Viennent ensuite les deux variables suivantes dont les noms devraient vous être familiers depuis leur déclaration dans la section *Variables* :

```
int ledIntenInt;
bool ledIntenBool;
```

Arduino IoT Cloud implémente les variables dans le *back-end* comme des variables C standard dotées de propriétés supplémentaires. Ces propriétés se trouvent, avec d'autres, dans la méthode `initProperties`, qui assure la mise en place des primitives et des structures nécessaires à la communication avec le nuage selon le schéma suivant :

```
void initProperties() {
ArduinoCloud.setThingId(THING_ID);
ArduinoCloud.addProperty(ledIntenInt, READWRITE,
ON_CHANGE, onLedIntenIntChange);
ArduinoCloud.addProperty(ledIntenBool, READWRITE,
  ON_CHANGE, onLedIntenBoolChange);
}
```

Il est intéressant de noter que la méthode `addProperty` se charge « d'enregistrer » l'attribut. Remarquez le passage des pointeurs de fonction `onLedIntenIntChange` et `onLedIntenBoolChange` – ils joueront un rôle important par la suite.
La fonction de pilotage de l'application est décrite dans le croquis, qui commence par l'inclusion de l'en-tête (non représenté ici). Elle est suivie par l'initialisation du croquis, qui se déroule selon le schéma suivant :

```
void setup() {
 Serial.begin(9600);
 delay(1500);

 initProperties();

 ArduinoCloud.begin(ArduinoIoTPreferredConnection);
 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}
```

Du point de vue de l'environnement de programmation Arduino, l'Arduino IoT Cloud est un pilote matériel comme un autre. L'objet global `ArduinoCloud` expose un ensemble de fonctions que votre code utilise pour communiquer avec le pilote du nuage. L'appel à `setDebugMessageLevel`, qui définit la « verbosité » du pilote, est particulièrement important. Plus la valeur est élevée, plus le pilote du nuage émet d'informations de débogage sur le port série de la carte. Dans le cas de pilotes « compliqués », la question se pose toujours de savoir comment la puissance de calcul est allouée. Le squelette de projet créé pour nous par l'Arduino IoT Cloud peut répondre à cette question dans la méthode `loop`, qui se charge de l'allocation de la puissance de calcul selon le schéma suivant :

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

L'Arduino IoT Cloud nous offre par défaut les trois méthodes d'écoute suivantes :

```
void onTestScheduleChange() {
}
void onLedIntenBoolChange() {
}
void onLedIntenIntChange() {
}
```

`onLedIntenBoolChange` et `onLedIntenIntChange` sont responsables des variables créées à distance dans le *back-end*, tandis que `onTestScheduleChange` aide à mettre en œuvre les fonctions « internes » du nuage Arduino.
Nous pouvons nous en occuper à l'étape suivante, en utilisant les capacités intégrées « intelligentes » de l'Arduino IoT Cloud. La carte Arduino utilisée ici est dotée d'une LED standard (rouge) sur la broche 13 et d'une LED RVB (canaux LEDR, LEDG et LEDB), qui peut être commandée par trois signaux PWM pour en modifier la couleur.
Revenons à la fonction `setup`, et initialisons les broches nécessaires :

```
void setup() {
...
 ArduinoCloud.printDebugInfo();

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
}
```

Les changements déclenchés dans le nuage activent l'écouteur d'évènement, qui écrit les valeurs entrantes sur le matériel :

```
void onLedIntenBoolChange() {
 digitalWrite(LED_BUILTIN, ledIntenBool);
}
void onLedIntenIntChange() {
 analogWrite(LEDB, ledIntenInt);
}
```

À ce stade, vous pouvez à nouveau transférer le croquis sur la carte. La LED RVB utilise des connexions à anode commune, de sorte que les couleurs individuelles sont contrôlées par la sortie d'un 0 pour allumer la LED correspondante. Les variables sont initialisées comme indiqué à la **figure 4,** pour que la diode bleue de la LED RVB s'allume après une initialisation réussie.

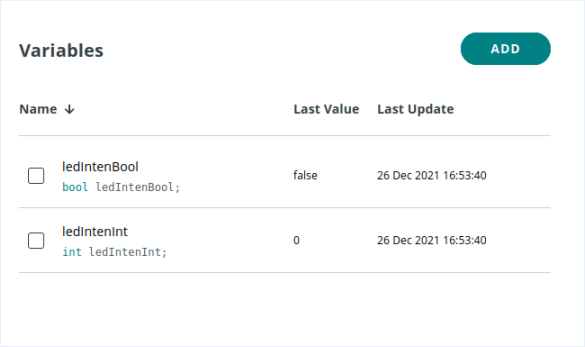


Figure 4. L'Arduino IoT Cloud fournit des informations détaillées sur l'état actuel des informations stockées dans une variable.

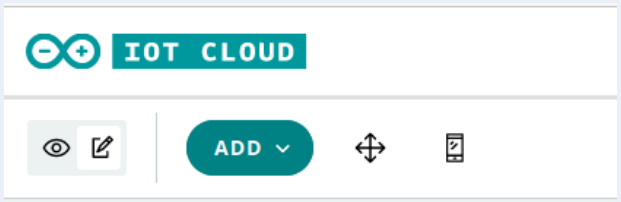


Figure 5. Passez le tableau de bord en mode édition en cliquant sur l'icône crayon en haut à gauche.

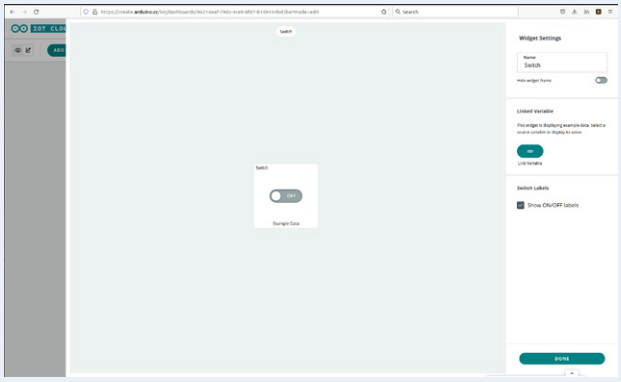


Figure 6. L'interface d'édition des commandes est « modale ».

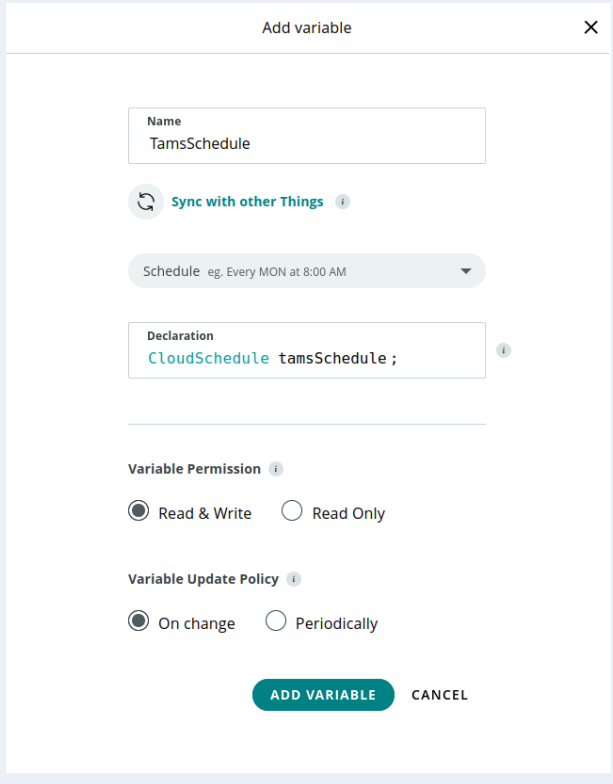


Figure 7. CloudScheduler est un type de données comme tous les autres.

Modification du contenu des variables

Si nous revenons au panneau principal d'Arduino IoT Cloud, nous pouvons cliquer sur l'onglet *Dashboards*, qui, s'il s'agit d'un nouveau compte, vous invitera à créer un nouveau tableau de bord. Cliquez maintenant sur le bouton *Build Dashboard* pour lancer l'éditeur, ce qui prendra un bon moment même si vous avez une connexion internet rapide.

Pour modifier et ajouter des éléments au tableau de bord, cliquez sur l'icône d'édition (le crayon) en haut à gauche de l'écran (**fig. 5).** Une fois ce mode sélectionné, le bouton bleu *ADD* apparaît, que vous pouvez utiliser pour afficher la liste déroulante des widgets disponibles. Nous sélectionnons d'abord un widget *Switch*, qui apparaît alors dans l'interface d'édition présentée dans la **figure 6**. On a maintenant à droite de l'écran un champ *Linked Variable* avec un bouton de liaison. Cliquez dessus pour activer une liste de tous les objets et variables contenus dans le compte du nuage. Ici, nous pouvons choisir la variable `ledIntenBool` et la lier en utilisant le bouton *Link Variable*. L'état de `ledIntenBool` sera maintenant associé à l'état de l'interrupteur. Un clic sur le bouton *DONE* ferme l'interface d'édition et l'interrupteur est maintenant incorporé dans le tableau de bord. Puis nous pouvons cliquer sur l'icône en forme d'œil pour relâcher l'interrupteur et l'activer. La manœuvre de l'interrupteur commande maintenant la LED rouge située à côté de la prise micro-USB.

Afin de pouvoir régler la luminosité de la LED bleue, il faut remettre l'éditeur du tableau de bord en mode édition et ajouter une nouvelle commande avec *Add -> Widgets*. Cette fois, j'ai choisi le type *Slider*. Dans son interface d'édition, nous définissons sa *Value range* de 0 à 255. Le lien est fait par la variable `letIntenInt`, qui représente la « commande de luminosité » de la LED RVB. Nous repassons enfin en mode d'activation et constatons que les changements de position du curseur affectent désormais la luminosité de la LED bleue.

Utilisation du planificateur

Au moment où j'écris cet article, une nouvelle fonction permet de configurer des tâches planifiées ou *web cron-jobs* : elle utilise un type de variable appelé *CloudSchedule* que vous pouvez définir pour qu'elle soit vraie ou fausse à un moment précis et pour une durée donnée. Il n'est pas nécessaire d'invoquer une fonction de minuterie, car cette variable est définie ou réinitialisée automatiquement dans l'Arduino IoT Cloud, selon la façon dont vous la configurez. Les tâches peuvent ensuite être déclenchées en vérifiant l'état de cette variable. Pour en démontrer les possibilités, créons une nouvelle variable de type *Schedule*. La **figure 7** montre la configuration souhaitée. Nous pouvons maintenant voir le nouveau type de variable dans le code :

```
CloudSchedule tamsSchedule;
```

Les paramètres temporels de cette variable sont configurés via le tableau de bord. La page de configuration présentée dans la **figure 8** montre les paramètres de réglage à fournir.

À l'étape suivante, nous devons nous occuper du traitement « local » des valeurs contenues dans `tamsSchedule` :

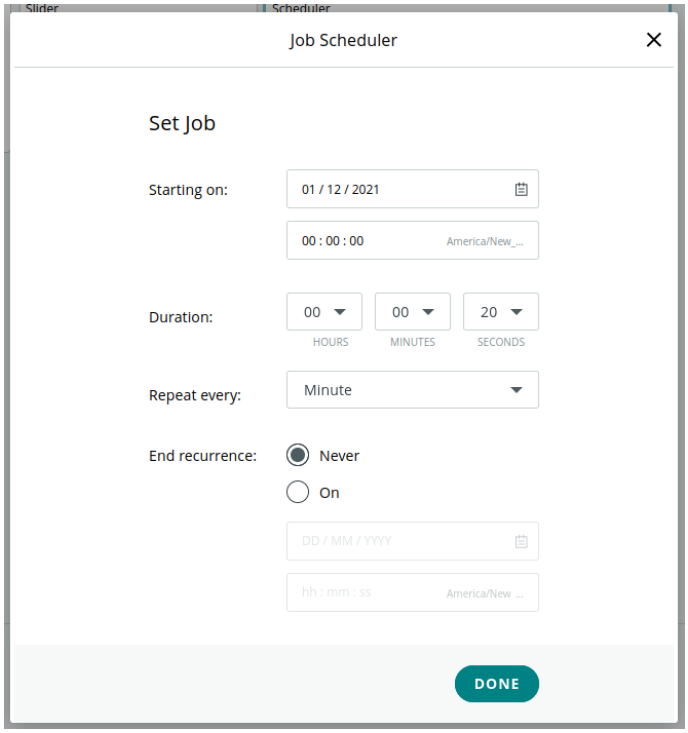


Figure 8. Le planificateur est configuré à l'aide du tableau de bord.

```
void setup() {
...

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
 pinMode(LEDR, OUTPUT);
}

void loop() {
 ArduinoCloud.update();
 // Your code here
 if(tamsSchedule.isActive()){
  digitalWrite(LEDR, HIGH);
 }
 else{
  digitalWrite(LEDR, LOW);
 }
}
```

Il est important de noter ici que la « rédaction » des informations fournies dans `tamsSchedule` est à la charge exclusive du développeur. Le nuage se limite simplement à mettre à jour périodiquement la valeur contenue dans `tamsSchedule`. La procédure d'interrogation continue présentée ici dans la structure `loop` n'est peut-être pas optimale du point de vue des ressources, mais elle fonctionne sans problème. Le programme peut maintenant être envoyé à l'Arduino où vous pouvez observer les clignotements rouges périodiques provenant de la LED RVB.
À ce stade, je n'ai pas pu résister à un autre petit test du système pour voir ce qui se passe lorsque la liaison RF tombe en panne. Le « lab Wi-Fi » a été éteint et l'Arduino a commencé à agir de manière erratique en commutant de manière aléatoire l'élément bleu de la LED RVB ainsi que la LED sur la broche 13 pour finalement effectuer un redémarrage complet après quelques secondes.

À la mise sous presse de cet article, je n'ai pas vraiment compris comment précisément l'Arduino IoT Cloud se remettait de la perte de la liaison radio entre l'appareil final et le serveur.

Une option pratique

Il est clair que l'Arduino IoT Cloud est encore en cours de développement et d'amélioration. Il a cependant un grand potentiel et offre au développeur d'applications IdO une option pratique et accessible pour mettre en œuvre un *back-end* dans le nuage, sans les inconvénients de MQTT et Cie. En dépit du petit couac, je recommande vivement ce produit !

210550-04

Contributeurs

Idée, illustrations et texte : **Tam Hanna**
Rédaction : **Rolf Gerstendorf**
Traduction : **Denis Lafourcade**
Mise en page : **Harmen Heida**

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (tamhan@tamoggemon.com) ou contactez Elektor (redaction@elektor.fr).

LIENS

[1] Jumeau numérique : https://en.wikipedia.org/wiki/Digital_twin
[2] Cartes prises en charge par l'Arduino IoT Cloud : https://bit.ly/3t8Vl3W
[3] Objets Arduino : https://create.arduino.cc/iot/things