AWS pour Arduino et Cie. (1)

Uutiliser AWS IoT ExpressLink en pratique

Tam Hanna (Hongrie)

Se connecter à Amazon Web Services (AWS) n'est pas toujours une tâche aisée. Bien que cela soit toujours possible avec un Raspberry Pi, les ordinateurs monocartes sont souvent trop encombrants pour de nombreuses applications IdO pratiques. C'est là qu'intervient AWS IoT ExpressLink, qui offre aux petits appareils un moyen simple de se connecter à AWS. Le module de connectivité basé sur AWS IoT ExpressLink gère les communications avec le cloud, ce qui permet au microcontrôleur hôte de réaliser des mesures et des tâches de contrôle. Dans ce premier article d'une série en deux parties, nous allons voir comment configurer un module ExpressLink avec un ESP32.

Les services en nuage (cloud) tels que Microsoft Azure et Amazon Web Services (AWS) offrent de puissantes solutions de stockage, de traitement et de visualisation de données. L'interaction avec les services cloud n'est pas vraiment simple, mais possible avec les ordinateurs monocartes tels que le Raspberry Pi ou l'Orange Pi. Cependant, ces systèmes peuvent s'avérer trop encombrants pour de nombreuses applications pratiques. Pour les cartes à microcontrôleur, l'implémentation d'une pile TCP/IP complète sur un contrôleur 32 bits est possible, mais elle consomme beaucoup de ressources. Sur des appareils plus petits et moins performants, cette tâche devient encore plus difficile.

Avec la technologie AWS IoT ExpressLink, Amazon vise à aider les concepteurs de petits appareils à surmonter cette difficulté. Dans cet article, nous allons explorer les possibilités et les limites de cette technologie avant de procéder à des essais avec un Arduino UNO R4 WiFi et un ESP32-C3-AWS-ExpressLink-DevKit d'Espressif. L'objectif est de présenter un « flux de travail de bout en bout » utile pour vos propres projets.

Protocole AT

Le protocole AT, à l'origine développé par Hayes Microcomputer Co. pour la gestion des modems, reste utile même dans une application d'accès à distance. En fait, le protocole de contrôle reste largement utilisé, notamment dans le contrôle de modules radio modernes, comme peuvent en témoigner les professionnels du domaine.

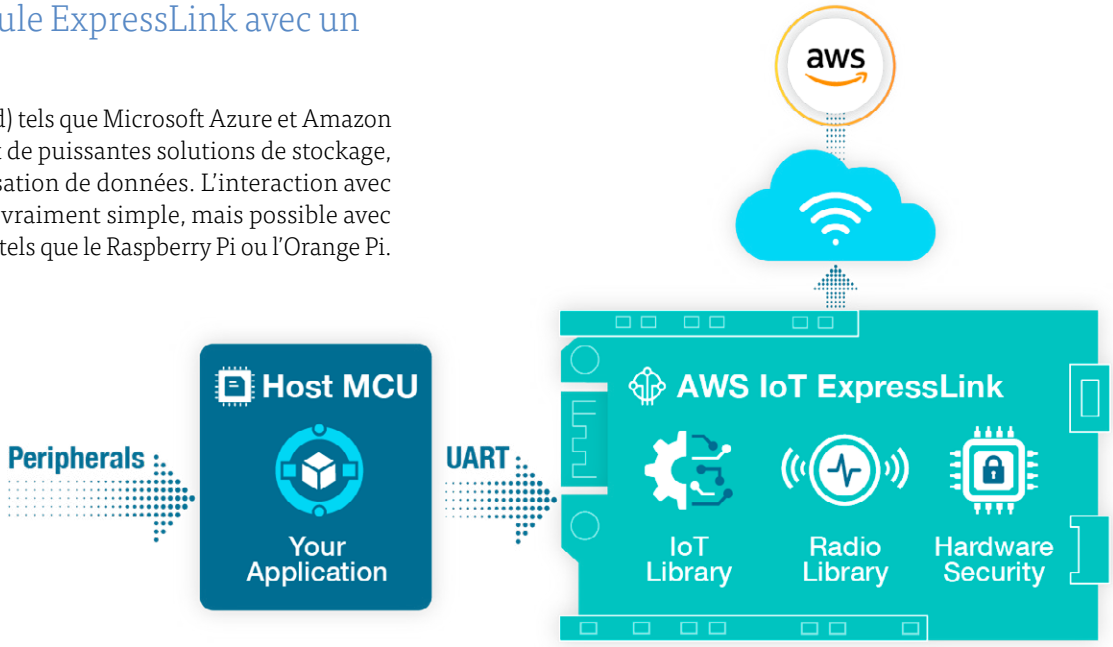

Figure 1. Certains utilisateurs appellent le module ExpressLink le "modem AWS". (Source : [1])

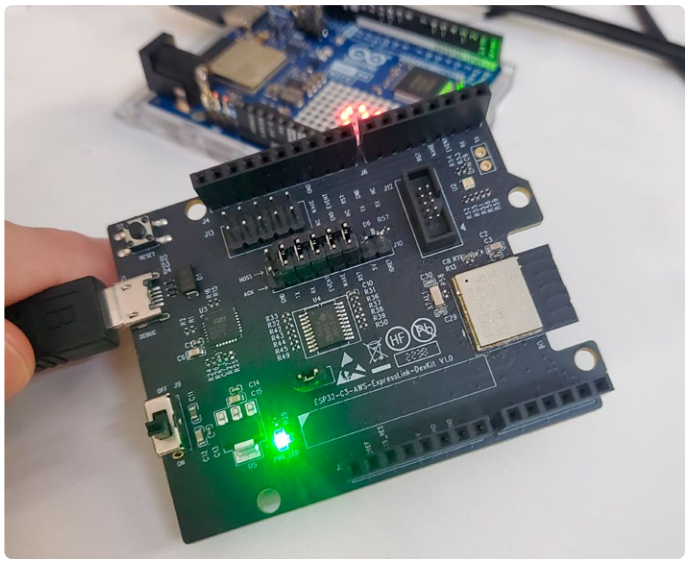

Figure 2. Avec ce module, cela fonctionne !

Le fonctionnement d'un module ExpressLink est expliqué dans la documentation d'Espressif. La **figure 1** montre comment les différents éléments fonctionnent ensemble. Le module ExpressLink communique avec le microcontrôleur hôte via une interface simple, dédié uniquement à la gestion de l'interaction avec les capteurs et les actionneurs. Le logiciel du microcontrôleur hôte se charge d'envoyer des commandes correctement formatées au module, qui prend en charge la communication avec le cloud.
Il convient de noter que le module ExpressLink gère aussi l'interface radio. Ceux qui souhaitent combiner un tel module avec un SoC radio autonome, tel que le GigaDevice GD32W515 ou une autre carte ESP32, doivent s'assurer que la communication dans le cloud soit effectuée via l'émetteur-récepteur du module ExpressLink.
L'un des avantages de ce choix est que la plupart des modules ExpressLink sont dotés d'une antenne. Cela nous permet d'éviter la complexité liée à la conception de l'antenne, de l'analyse du réseau vectoriel, etc.
L'ESP32-C3-AWS-ExpressLink-DevKit présenté dans la **figure 2** est un kit de démarrage intéressant et compatible avec diverses cartes Arduino. Selon *oemsecrets.com*, il est disponible au prix approximatif de 28 euros (voir [2]). Il constitue un outil abordable pour commencer à utiliser AWS IoT ExpressLink. Dans les étapes suivantes, nous utiliserons un Arduino UNO R4 WiFi comme carte de base, sans utiliser son module Wifi (voir ci-dessus).

Configuration du matériel

En théorie, la configuration de la carte et son utilisation en ligne sont simples — Espressif propose le module ExpressLink sous la forme d'un shield Arduino, il suffit donc de connecter les deux cartes.
Cependant, la figure 1 peut prêter à confusion, car l'interface de communication entre le microcontrôleur hôte et le module radio ne se limite pas à une simple liaison série. La **figure 3** montre quelques connexions supplémentaires nécessaires pour transmettre l'état opérationnel de certains signaux de contrôle appropriés.

ExpressLink Pin	ESP32-C3 GPIO Pin	ESP32-C3-MINI-1-N4-A Module Pin
TX	IO19	27
RX	IO18	26
EVENT	IO10	16
WAKE	IO3	6
RESET	EN	8

Figure 3. Certaines broches GPIO sont nécessaires pour signaler des états opérationnels critiques ou importants (Source : [14])

Comme nous nous concentrons sur l'utilisation d'Arduino et du shield Arduino d'Espressif dans les étapes suivantes, nous n'aborderons pas ces broches ici. Nous les mentionnons surtout parce qu'Espressif souligne leur importance dans la documentation - leur absence limitera les capacités du circuit à une simple communication « Hello World », rendant toute communication pratique avec AWS impossible.
Avant de connecter l'Arduino et la carte Espressif, il est utile de mettre à jour le micrologiciel du module. Nous ne savons pas depuis combien de temps le module a été en stock chez le distributeur et nous ne savons pas si la dernière version du logiciel est préchargée. En théorie, il est possible d'effectuer cette mise à jour à distance, mais cela nécessiterait une connexion préétablie entre le module et les services AWS IoT cloud.
En alternative, Espressif nous propose un script Python qui nécessite Python 3 installé sur un PC. Dans mon cas, un poste de travail fonctionnant sous Ubuntu 20.04 LTS, indique la version comme suit :

```
tamhan@TAMHAN18:~$ python3 --version
Python 3.8.10
```

Pour établir une communication, vous devez exécuter la commande `pip3 install pyserial==3.5`, qui installe une bibliothèque qui assure le lien entre l'environnement d'exécution Python et les ports série du PC.
Ensuite, vous devrez télécharger l'outil nécessaire au chargement du micrologiciel, disponible sous la forme d'un fichier *.py*. sur GitHub [3]. Cherchez le bouton indiqué dans la **figure 4**, qui permet de télécharger directement l'outil sans avoir recours au processus habituel de copier-coller.

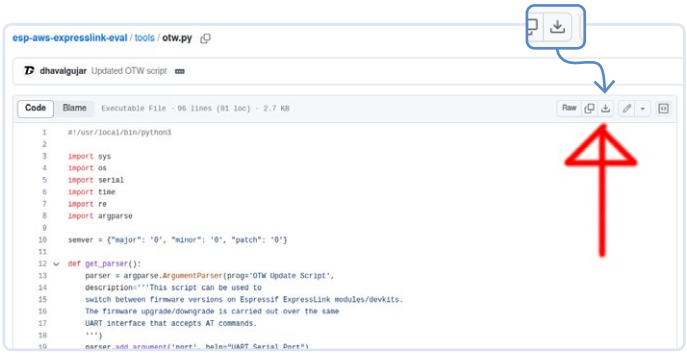

Figure 4. Ce bouton permet de gagner du temps.

Figure 5. Les deux adaptateurs USB-série apparaissent dans le système de fichiers /dev du poste de travail.

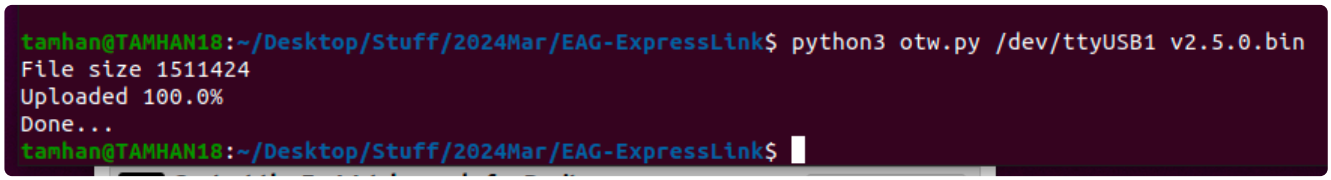

Figure 6. Assurez-vous que le module est à jour. Dans le cas contraire, OTW.py affiche des messages d'erreur.

Vous devrez ensuite télécharger le micrologiciel - pour ce faire, rendez-vous à nouveau sur GitHub [4]. Nous avons réussi à télécharger la version *v2.5.0.bin* du micrologiciel - assurez-vous simplement de ne pas télécharger le fichier *hash* par erreur.
La mise à jour du logiciel d'exploitation est impossible via l'interface USB intégrée à la carte car celle-ci se connecte à un « UART d'application » et ne permet pas la programmation de l'ESP32 qui sert ici de module radio. Pour résoudre ce problème, nous avons utilisé un mini-module FTDI, plus précisément le FT2232H MINI MODULE, qui est également efficace pour programmer les microcontrôleurs ESP32 dans votre labo.
Il convient de noter que la connexion série et l'affectation des broches ne sont pas standards. Pin BD1 est connecté au TX du module ESP32, tandis que Pin BD0 est connecté à la broche RX. Les broches TX et RX sont situées au milieu du module et ne sont pas conformes à la disposition habituelle des broches Arduino.
Enfin, nous avons besoin d'un fil de connexion pour relier les deux potentiels de masse ensemble. Pour le montage, nous commençons par connecter le module FTDI à l'ordinateur, puis nous connectons l'alimentation de la carte ExpressLink. Cette configuration de connexion est séquencée dans dmesg comme le montre la **figure 5**.
Il est important de noter que les ports ttyUSB0 et ttyUSB1 sont responsables de la communication PC-FTDI. Si vous utilisez les broches mentionnées, le chargement de programme s'effectue comme suit :

```
tamhan@TAMHAN18:~/Desktop/Stuff/2024Mar/EAG-ExpressLink$
python3 otw.py /dev/ttyUSB1 v2.5.0.bin
```

Le succès est visible, comme le montre la **figure 6**. Une fois cette étape terminée, nous n'aurons plus besoin d'utiliser le module FTDI. Le module ExpressLink sera connecté uniquement à l'Arduino.

Configuration de la connexion au cloud

Après la mise à jour du micrologiciel du module, nous sommes maintenant prêts à le mettre en service. Dans les étapes suivantes, je suppose que vous avez créé et configuré votre propre compte AWS et que vous vous connectez en tant qu'utilisateur root à l'adresse *https://aws.amazon.com*. Pour des raisons de sécurité, nous n'aborderons pas l'utilisation recommandée d'IAM dans les étapes suivantes. Si vous n'avez pas encore de compte AWS, consultez le guide de démarrage rapide [5]. Veuillez noter que la création d'un compte nécessite une carte de crédit.
Une fois mis à jour avec le dernier micrologiciel, il est possible d'initialiser le module via le cloud ou en envoyant des commandes AT avec des paramètres de configuration spécifiques. Pour les étapes suivantes, vous aurez besoin de la console AWS IoT, disponible sur *http://console.aws.amazon.com/iot*. Assurez-vous de la lancer sur votre poste de travail. Vous pouvez la garder ouverte dans une fenêtre de navigateur.
Dans les étapes suivantes, nous utilisons l'EDI Arduino 2 comme

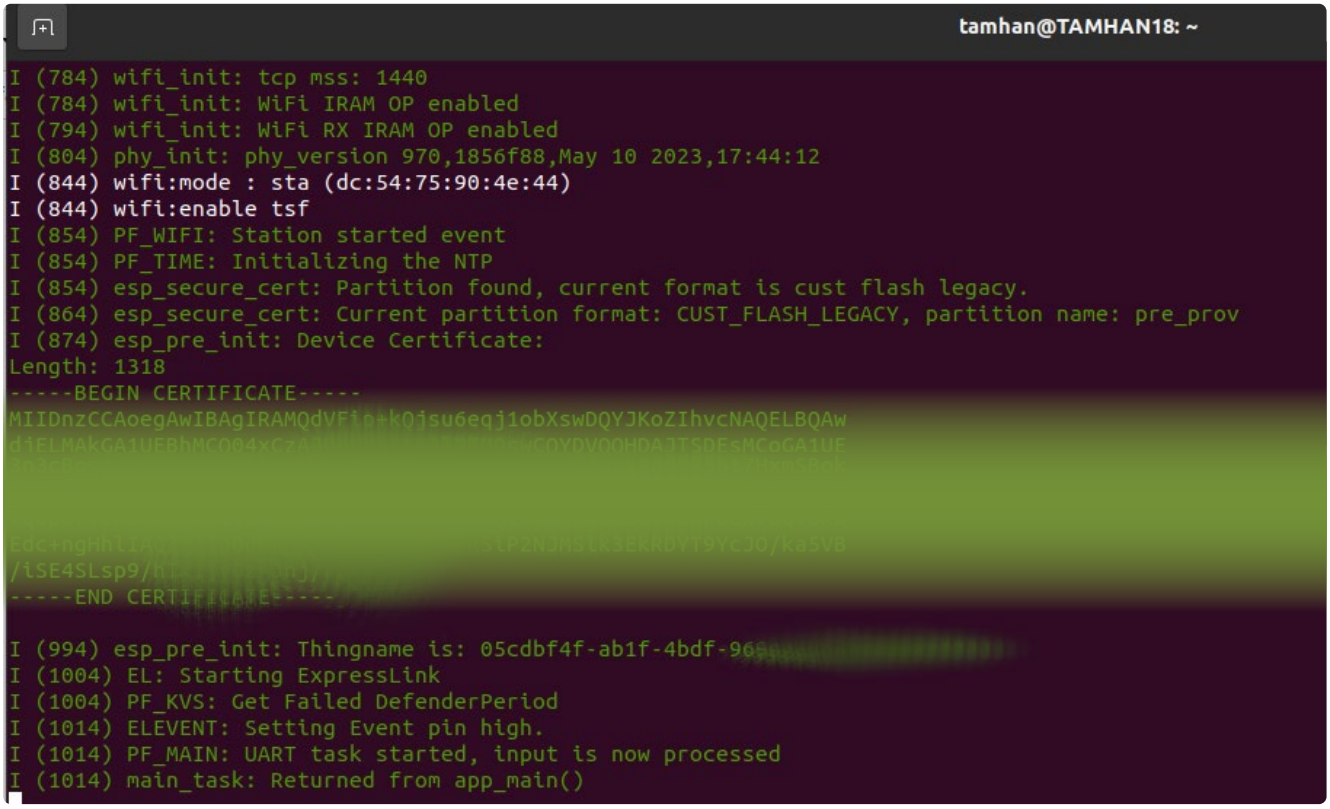

Figure 7. Le module ExpressLink est communicatif au démarrage.

environnement de développement pour écrire le croquis Arduino - nous ne couvrirons pas ici son installation sous Linux avec un fichier AppImage.

Il est important de noter que pendant cette opération, seul l'Arduino doit être connecté à l'ordinateur - le module est alimenté par l'Arduino lui-même. Si le PC doit communiquer directement avec le module en envoyant des commandes AT, vous devez d'abord déconnecter l'Arduino et utiliser un câble Micro-USB pour relier le module.

Le matériel communiquant avec AWS est représenté sur le serveur sous la forme d'une Thing. Pour l'authentification de notre Thing avec le serveur, un nom (*Thing Name*) et un certificat cryptographique sont requis. Ces deux éléments se trouvent dans la mémoire du module ExpressLink et nous devons maintenant les déterminer. Bien qu'Amazon propose dans sa documentation l'utilisation des commandes `AT+CONF? ThingName` et `AT+CONF? Certificate pem`, en pratique, il est plus simple de connecter directement le module à l'ordinateur, de lancer une application de terminal, puis d'appuyer sur le bouton de réinitialisation. Le micrologiciel implémenté par Espressif fournit dans ce cas les informations nécessaires - comme le montre la **figure 7** - que nous pouvons ensuite copier directement dans le clipboard.

Je préfère utiliser Screen comme application de terminal sous Linux - bien que ce soit un peu difficile, les commandes de configuration de la connexion sont les suivantes :

```
tamhan@TAMHAN18:~$ screen /dev/ttyUSB0 115200
```

Pour quitter Screen, ouvrez une deuxième fenêtre de terminal et entrez la commande suivante (cette méthode permet de conserver le contenu de la boîte de dialogue Screen dans l'autre fenêtre de terminal de sorte que vous pouvez y accéder si nécessaire) :

```
tamhan@TAMHAN18:~$ sudo killall screen -s9
```

Comme prochaine étape dans notre configuration, nous devons retourner à la console AWS IoT mentionnée précédemment. L'URL exacte dépend de la région géographique que vous avez choisi comme « base » pour votre système IdO - sur mon poste de travail, le chemin est le suivant *https://eu-west-1.console.aws.amazon.com/iot/home?region=eu-west-1#/home*.

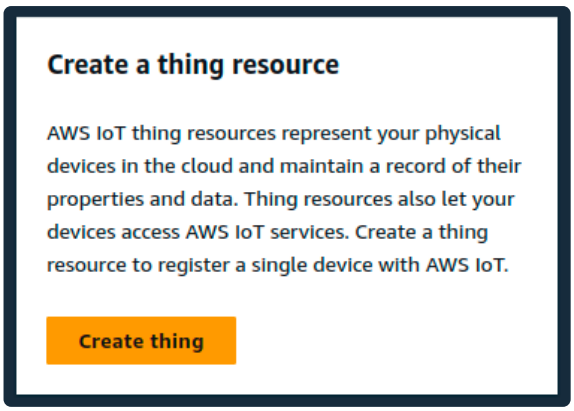

Figure 8. Seule cette option permet de créer de nouvelles Thing.

Cartes de démarrage

Dans cet article, nous avons utilisé le module Espressif pour sa disponibilité. Amazon a signé des contrats avec divers autres partenaires, tels que U-blox, Infineon, Realtek et Telit - vous trouverez une liste actualisée des cartes de démarrage de tous les fabricants sous [15].

La navigation est un peu délicate, car l'interface de navigation de gauche n'est pas toujours complète et peut défiler. Néanmoins, la prochaine étape consiste à cliquer sur le bouton *Link Manage* (Gérer les liens) -. Vous devrez alors cliquer sur le bouton *Create Thing* (Créer une Thing) dans l'écran qui apparait ensuite, comme le montre la **figure 8**.
Après tous ces efforts, nous accédons à l'assistant de configuration, qui va maintenant nous guider à travers les trois étapes permettant d'intégrer une nouvelle Thing dans le service AWS IoT.
Dans la première étape, le seul champ à remplir est *Thing Name*, où vous entrerez la chaîne de caractères obtenue plus tôt. Ensuite, il suffit de cliquer sur *Next Step*, où Amazon nous invitera à « télécharger » les fichiers cryptographiques.
Dans la première étape, nous optons pour l'option *Use my certificate* et ensuite nous sélectionnons *CA is not registered with AWS IoT*.
Au fil des ans, le format de fichier *.pem* est devenu un conteneur quasi-standard pour divers types de clés cryptographiques. Pour ceux qui souhaitent en savoir plus, une description détaillée de ce format de fichier est disponible sur [6]. Cependant, nous avons juste besoin de générer un fichier *.pem* valide :

```
tamhan@TAMHAN18:~/Desktop/ExpressLink$ touch tamscert.pem
tamhan@TAMHAN18:~/Desktop/ExpressLink$ gedit tamscert.pem
```

Après avoir entré la commande `gedit-`, un nouveau fichier s'ouvre dans l'éditeur de texte, dans lequel nous collons le texte du certificat que nous avons copié plus tôt. Il est important d'inclure les délimiteurs `-----BEGIN CERTIFICATE-----` et `-----END CERTIFICATE-----` dans le fichier texte, sans quoi l'analyseur ne pourra pas récupérer les informations du certificat.
Une fois que vous avez sauvegardé le fichier *.pem*, vous pouvez le télécharger lors de la dernière étape. Dans la section *Certificate Status*, sélectionnez l'option *Actif*. Cliquez ensuite sur *Next Step* pour passer à la troisième étape, puis sur *Attach policies to certificate* (Attacher des politiques au certificat).
AWS IoT met en œuvre un système d'autorisation basé sur des certificats. Cela signifie que les certificats sont associés à des documents de politique, qui dictent ensuite les actions que la ressource respective dans le réseau AWS est autorisée à effectuer.
Dans les clusters AWS nouvellement configurés, les politiques prédéfinies ne sont plus utilisées depuis un certain temps. Amazon adopte probablement cette approche pour éviter toute responsabilité en cas d'incidents de sécurité causés par des configurations non sécurisées.
Pour cette raison, dans la première étape, nous pouvons cliquer sur le bouton *Create Policy*, qui ouvre une nouvelle fenêtre de navigateur montrant le *Policy Editor*. Dans cette première étape, nous donnons à la politique un nom par lequel elle sera connue par la suite.
En dessous, une deuxième option permet de définir les autorisations accordées par la politique. Ici, nous cliquons sur l'option *JSON* pour activer la saisie directe de texte. Le code déjà présent est généralement prêt à l'emploi - il suffit de veiller à insérer les caractères génériques suivants dans les champs *Action* et *Resource* :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

La politique présentée permet à l'appareil d'avoir un accès universel à toutes les ressources du backend AWS. Bien entendu, cette approche n'est pas recommandée dans les déploiements pratiques. Il convient de noter que, pour des raisons juridiques, vous devez supprimer ces politiques dès que possible après vos tests. L'éditeur et l'auteur ne peuvent être tenus responsables des conséquences des erreurs commises.
Après avoir cliqué sur l'option *Create*, la fenêtre *Policy Editor* affiche une liste de politiques - il suffit maintenant de fermer la fenêtre et de revenir au processus de génération d'une Thing. La politique nouvellement créée est maintenant prête à être sélectionnée.
Ensuite, il est nécessaire de cliquer sur le bouton *Create* pour compléter le processus de saisie dans le backend.
Pour terminer la configuration, il faudra défiler tout en bas dans l'interface de navigation de gauche de la console AWS IoT et cliquer sur l'option *Settings*.

Dans la section *Device Data Endpoints*, vous trouverez une chaîne au format suivant : *a3gw111abcdefgdl-ats.iot.eu-west-1.amazonaws.com*. Vous devez enregistrer cette chaîne, car elle détermine le point d'entrée par lequel le module se connectera au cloud.

Établissement de la liaison Wifi

La configuration correcte du côté du cloud AWS ne représente que la moitié de la tâche, car le module ESP32 doit encore réussir à communiquer avec votre réseau local. Pour ce faire, vous devez entrer vos informations d'identification Wifi dans le *Secure Element*.
Dans l'ensemble de commandes AT spécifié par Amazon, vous trouverez les commandes `AT+CONF SSID=<replace-with-your-router-ssid>` et `AT+CONF Passphrase=<replace-with-your-router-passphrase>`, qui nous permettent de transférer les informations d'identification Wifi appropriées via une liaison série.
Espressif a toutefois doté ses modules ExpressLink d'une fonction utile et pratique : des applications Android et iOS, qui permettent une configuration plus « graphique » des modules, sont disponibles sur [6] et [7].
Le seul problème dans ce contexte est que l'exposition de l'interface Bluetooth nécessaire nécessite d'abord la transmission de la commande `AT+CONFMODE`. Pour cela, nous pouvons utiliser le croquis d'exemple fourni par Amazon [8].
Normalement, l'Arduino utilise le port série matériel pour afficher des messages `printf()` et d'autres informations d'état. Après

l'installation d'un module ExpressLink, ce port n'est (logiquement) plus disponible. L'équipe de développement d'AWS s'est donc appuyée sur la bibliothèque *SoftwareSerial*. Au moment de la rédaction de cet article, cette bibliothèque ne fonctionne pas bien avec l'Arduino R4 (voir [10]), mais elle n'est de toute façon pas nécessaire. Notre première tâche consiste donc à nettoyer le croquis en supprimant tous les appels qui y sont liés. Il est très utile que le Renesas Arduino offre deux interfaces UART matérielles utilisant des broches différentes [11]. Serial1 envoie des données au module radio, tandis que Serial peut être utilisé pour la communication via le moniteur série de l'EDI Arduino fonctionnant sur le PC. Cela signifie que la méthode responsable de la communication avec le module ESP32 doit être modifiée avec les commandes suivantes :

```
String execute_command(String command,
    unsigned long timeout_ms) {
    Serial.print("EXC : ");
    Serial.println(command);

    unsigned long saved_timeout_value_ms =
       Serial1.getTimeout();
    Serial1.setTimeout(timeout_ms);
    Serial1.println(command);
    String s = Serial1.readStringUntil('\n');
    s.trim();
    Serial1.setTimeout(saved_timeout_value_ms);
    return s;
}
```

Veillez à modifier la configuration conformément aux définitions suivantes avant d'exécuter le programme :

```
#define EVENT_PIN 2
#define RESET_PIN 4

#define MY_AWS_IOT_ENDPOINT "change_me-ats.iot.eu-west-1.
    amazonaws.com"
```

En raison d'un conflit rapporté par l'auteur entre les parties Renesas et Arduino de la bibliothèque, comme documenté sous [12], une modification de la fonction `process_event()` est nécessaire. Spécifiquement, la valeur retournée par `c_str()` ne doit pas être passée directement à `scanf()` :

```
event_t process_event()
{
  String response;

    int val = 0;
    event_t event_number = EVENT_NONE;
    val = digitalRead(EVENT_PIN);
    if(val)
    {
        response = execute_command("AT+EVENT?",
           3000);
        if (response.equals("OK"))
        {
            return EVENT_NONE;
        }
        else {
            char ok_string[3];
            int topic_index;
            int total_read;
            char someResponse[300];
            strcpy(someResponse, response.c_str());
            total_read = sscanf(someResponse,
               "%s %d %d %*s", ok_string,
               &event_number, &topic_index);
            return event_number;
        }
    }
}
```

Le module ESP32 est connecté à l'Arduino, alors que seul l'Arduino utilise la liaison USB. L'alimentation électrique du kit ExpressLink doit être éteinte. Une double pression rapide sur le bouton *RST* de l'Arduino UNO R4 est nécessaire pour permettre au bootloader d'accepter de nouvelles compilations.
Après avoir modifié et flashé avec succès le croquis de test [13], le module reçoit la commande nécessaire s'il n'est pas déjà dans un état provisionné. Vous pouvez vérifier ceci en essayant de relire le SSID stocké dans le *Secure Element* :

```
void loop() {
 event_t event = EVENT_NONE;
 String response;
 static state_t state = STATE_INIT;
 if (state != STATE_INIT) {
    event = process_event();
 }
 switch (state) {
   …
   case STATE_EL_READY:
      response = execute_command(
         "AT+CONF Endpoint="MY_AWS_IOT_ENDPOINT"",
         3000);
      response = execute_command(
         "AT+CONF Topic1=TEST", 3000);
      response = execute_command(
         "AT+CONF? SSID", 3000);
      state = process_ssid(response);
      break;
   case STATE_UNPROVISIONED:
       response = execute_command(
         "AT+CONFMODE", 5000);
      if (response.equals("OK CONFMODE ENABLED"))
      {
        state = STATE_PROVISIONING;
      }
      break;
```

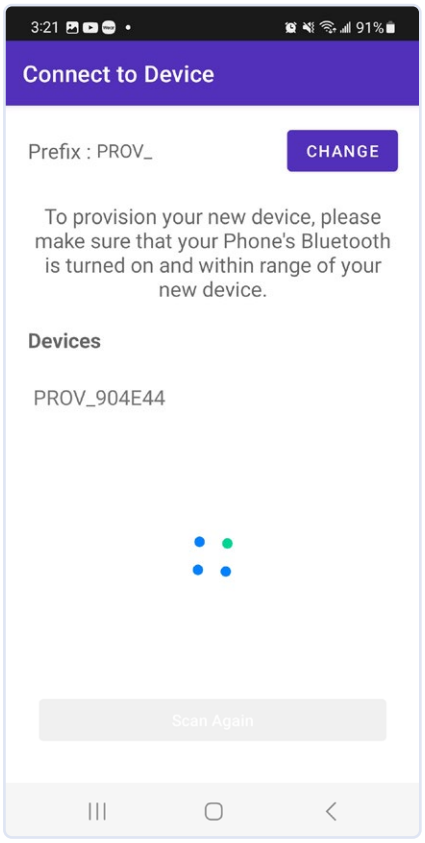

Figure 9. Le scanner Bluetooth LE a détecté une nouvelle cible !

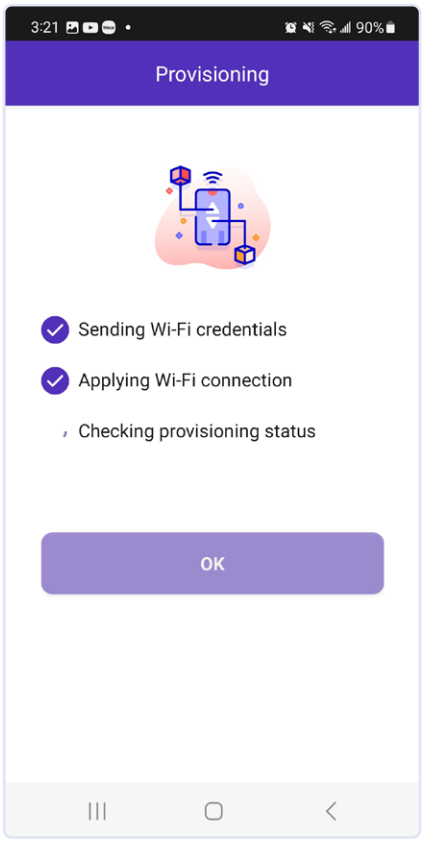

Figure 10. L'approvisionnement peut prendre jusqu'à 2 minutes...

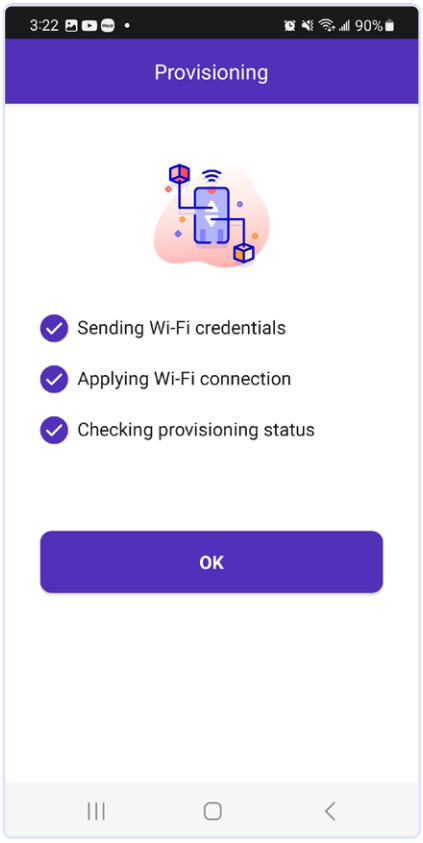

Figure 11. ...et n'est terminé que lorsque la fenêtre ci-contre apparaît.

Amazon met en œuvre le programme présenté ici sous la forme d'une machine à états colossale. La méthode responsable du traitement du SSID définit ensuite l'état correct comme suit, si le SSID renvoyé est inutilisable :

```
state_t process_ssid(String response)
{
  int event_number = 0;
  char ok_string[3];
  char ssid_string[33] = {0};
  int total_read;

  total_read = sscanf(response.c_str(),
    "%s %s" , ok_string, ssid_string);

  if(total_read > 1) {
    return STATE_PROVISIONED;
  }
  else {
    return STATE_UNPROVISIONED;
  }
}
```

Une fois que le module atteint le statut de provisionnement, il devient visible pour un scanner Bluetooth LE. Une méthode plus pratique consiste à utiliser l'application pour smartphone mentionnée précédemment. En raison des particularités du système de permissions d'Android, le système d'exploitation vous demandera de nombreuses autorisations lors du premier lancement. Si vous les accordez, vous accéderez à l'interface de l'appareil photo - Espressif utilise la même application non seulement pour le provisionnement Bluetooth LE, mais aussi pour capturer les informations des modules ExpressLink qui fournissent leur nom de module via un code QR.

La **figure 9** montre comment le module est détecté. L'exécution proprement dite du programme est ensuite réalisée par un assistant qui vous invite à saisir le mot de passe après avoir sélectionné le nom du Wifi. Notez que la transition entre les captures d'écran présentées dans les **figures 10** et **11** peut prendre un peu de temps.

Conclusion

Dans cet article, nous avons principalement démontré les étapes de configuration d'un module AWS IoT ExpressLink. Dans la deuxième partie de cette série, nous approfondirons la transmission des données. Les avantages de cette configuration particulière du système devraient déjà être évidents. Avec un investissement matériel aussi minime, il serait autrement assez difficile de se connecter à un grand fournisseur de services cloud. Le coût supplémentaire des modules radio peut être justifié par les promotions occasionnelles offertes en collaboration avec Amazon, ce qui peut être intéressant.

240240-04

À propos de l'auteur

Avec plus de 20 ans d'expérience, Tam Hanna est un ingénieur spécialisé dans l'électronique, l'informatique et les logiciels. Il est designer indépendant, auteur de plusieurs ouvrages et journaliste — *instagram.com/tam.hanna*. Durant ses moments libres, il se consacre à la conception et à la production de solutions imprimées en 3D. Il nourrit également une passion pour le commerce et la dégustation de cigares haut de gamme.

Questions ou commentaires ?

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

Produits

> **Arduino Uno R4 WiFi**
> www.elektor.fr/20528

LIENS

[1] Documentation d'ExpressLink par Espressif : https://espressif.com/en/solutions/device-connectivity/esp-aws-iot-expresslink
[2] Distributeurs de l'ESP32-C3-AWS-ExpressLink-DevKit : https://oemsecrets.com/compare/ESP32-C3-AWS-ExpressLink-DevKit
[3] Télécharger l'outil : https://github.com/espressif/esp-aws-expresslink-eval/blob/main/tools/otw.py
[4] Micrologiciel du module : https://github.com/espressif/esp-aws-expresslink-eval/releases
[5] AWS getting started guide : https://tinyurl.com/elgsg-set-up
[6] Description du format de fichier .pem : https://tinyurl.com/PEM-file
[7] Outil de configuration pour Android : https://play.google.com/store/apps/details?id=com.espressif.provble
[8] Outil de configuration pour iOS : https://apps.apple.com/app/esp-ble-provisioning/id1473590141
[9] Croquis Arduino pour l'activation du Bluetooth : https://tinyurl.com/sketches-arduino
[10] Problèmes liés à l'UART logiciel : https://github.com/arduino/uno-r4-library-compatibility/issues/12
[11] Two Hardware UARTs : https://forum.arduino.cc/t/uno-r4-and-serial-rx-tx/1146022/2
[12] Questions concernant les bibliothèques : https://github.com/espressif/esp-aws-expresslink-eval/issues/23
[13] Croquis de l'auteur : https://elektormagazine.fr/240240-0
[14] Getting Started Guide ExpressLink Evaluation Kit : https://github.com/espressif/esp-aws-expresslink-eval
[15] ExpressLink Starter Boards : https://tinyurl.com/ExpressLink-Starter