# AWS pour Arduino et cie. (2)

## transmission de données avec AWS IoT ExpressLink

**Tam Hanna (Hongrie)**

Dans le premier volet de cette série, nous avons exploré AWS IoT ExpressLink, qui simplifie l'envoi de données vers Amazon Web Services (AWS) à partir de microcontrôleurs. Un module intermédiaire gère les communications, permettant aux microcontrôleurs et aux développeurs de micrologiciels de manipuler ce « module de connectivité basé sur AWS IoT ExpressLink » via des commandes AT, sans se soucier des détails techniques. Nous avons déjà vu comment configurer un module ExpressLink basé sur l'ESP32, créer une« Thing » sur AWS et obtenir les certificats nécessaires. Cet article se concentre sur l'envoi de données depuis un Arduino vers AWS, exploitant les fonctionnalités avancées de la plateforme pour gérer et visualiser les données.

Tout d'abord, il est essentiel de souligner que l'utilisation d'AWS n'est pas une fin en soi. L'intégration d'un module AWS IoT ExpressLink dans votre projet doit viser à exploiter les nombreux services offerts par le portefeuille AWS IoT.
AWS propose une multitude de fonctionnalités (**figure 1**), tandis que la référence [1] offre un aperçu complet de ses capacités. Amazon fournit des détails sur les coûts associés à l'utilisation d'AWS IoT Core dans la référence [2].

### Envoi de données

Pour tirer parti de ces fonctionnalités, il est nécessaire de transférer des données vers le cloud AWS. La méthode choisie pour acheminer ces données, souvent désignée sous le terme d'*ingestion* dans le domaine de la science des données, repose sur l'utilisation du protocole de messagerie MQTT.
Dans la première partie de cet article [3], nous avons utilisé un croquis de test d'Espressif et d'AWS pour configurer le module ExpressLink avec des commandes AT, ce qui permet de le rendre accessible par AWS (provisionnement). Il est essentiel que le module soit configuré avec le SSID du routeur pour établir une connexion. Comme mentionné précédemment, vous pouvez facilement saisir les détails de la connexion depuis un smartphone qui communique avec le module via Bluetooth.

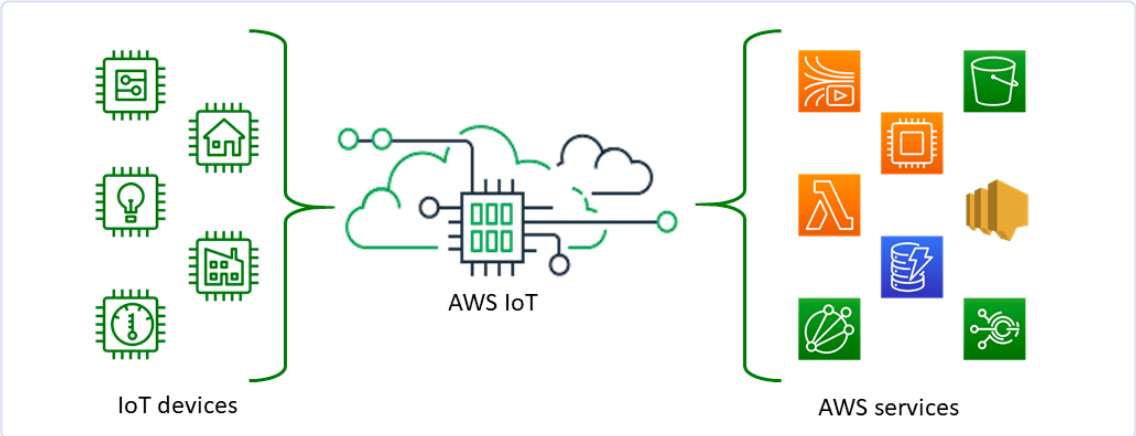


*Figure 1. Un large éventail de services est disponible via le portefeuille AWS. (Source : AWS [1])*

Si tout fonctionne correctement, le croquis de test enverra périodiquement des données au serveur AWS :

```
        case STATE_CONNECTED:
        if (event == EVENT_CONLOST) {
            state = STATE_PROVISIONED;
            break;
        }
        static unsigned long last_send_time;
        if (millis() - last_send_time >= 10000) {
            response =
      execute_command("AT+SEND1 Hello World", 5000);
            last_send_time = millis();
        }
        break;
```

La commande `execute_command("AT+SEND1...)` envoie le message au hub AWS IoT. Toute personne familière avec le protocole MQTT pourrait se demander comment le sujet MQTT utilisé par chaque appareil est déterminé.
La réponse se trouve un peu plus haut dans le listage du code :

```
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);
```

Étant donné que la définition d'un sujet MQTT est sans coût supplémentaire, il est recommandé de modifier le programme d'exemple pour inclure cette commande avant chaque transmission de données au module.

```
        if (millis() - last_send_time >= 10000) {
            response = execute_command
      ("AT+CONF Topic1=TEST", 3000);
            response = execute_command
      ("AT+SEND1 Hello World", 5000);
            last_send_time = millis();
        }
```

Après avoir apporté ces modifications, le programme sera opérationnel. Il est important de noter qu'une fois alimenté, le micrologiciel de l'Arduino enverra continuellement des messages au serveur, ce qui pourra engendre des coûts supplémentaires. Par conséquent, il est crucial de ne jamais laisser l'Arduino sans surveillance lorsqu'il est alimenté dans cette configuration !
Pour vérifier la fonctionnalité, nous pouvons utiliser le moniteur série de l'EDI Arduino. Dans la première partie de ce projet, nous avons décrit comment modifier le programme de test pour utiliser le second UART matériel de l'Arduino UNO R4. Ainsi, la console (**figure 2)** vous informera de la réussite de la transmission des paquets à envoyer au cloud AWS.

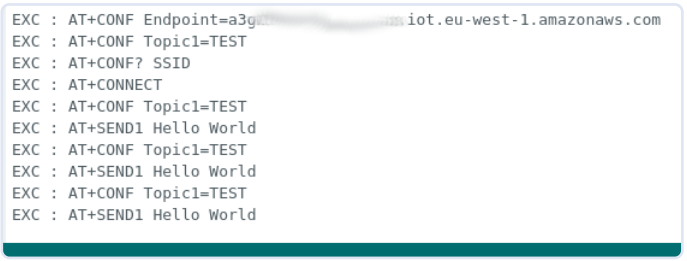


*Figure 2. Des messages apparaissent dans le moniteur série de l'EDI Arduino.*

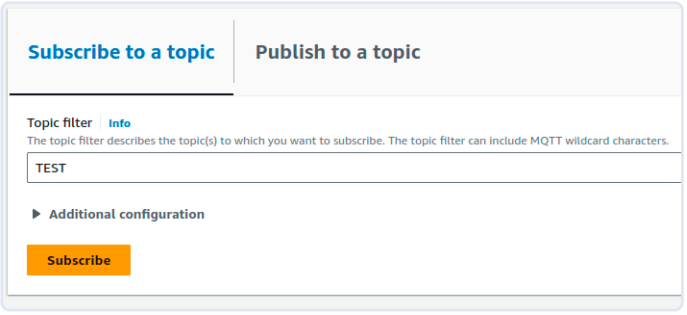


*Figure 3. Utilisez le filtre de topic pour vous inscrire*

Il est également important de confirmer que les messages sont bien reçus par AWS. Pour ce faire, sélectionnez *Test → MQTT Test Client dans* le menu du backend AWS, ce qui vous redirigera vers le client de test AWS. Comme mentionné dans l'article précédent, l'URL dépend de la localisation géographique de vos ressources AWS. Dans ma configuration, URL est la suivante *https://eu-west-1.console.aws.amazon.com/iot/home?region=eu-west-1#/test*.
Après le chargement de la page, il est important que la section *Connection Details* affiche un état *connected* — il arrive que le client fourni par Amazon ait des problèmes de démarrage.
Pour établir la connexion MQTT, l'étape suivante consiste à la configurer en utilisant *Subscribe to a Topic* et le formulaire présenté dans la **figure 3**. Une fois que vous avez cliqué sur *Subscribe*, vous devriez maintenant voir les messages entrants comme le montre la **figure 4.**

## Transmission de commandes avancées

Un des avantages majeurs des courtiers MQTT opérés depuis le cloud est leur capacité à répondre directement aux informations reçues, sans nécessité de développer extensivement la logique applicative. Pour tirer profit de cette fonctionnalité utile, les développeurs doivent veiller à ce que les données envoyées soient dans un format compréhensible par le backend. Généralement, cela implique l'utilisation d'une structure similaire à JSON. En plus de la charge utile, telles que les valeurs de mesure, ces messages peuvent contenir des attributs spécifiques, comme l'indication de la température critique d'un équipement dans un format interprétable par AWS, déclenchant une alarme dès que le seuil est dépassé.

Pour implémenter de telles fonctionnalités, il est généralement nécessaire d'utiliser les fonctions avancées du module ExpressLink. La documentation [4] fournit un aperçu du jeu de commandes.

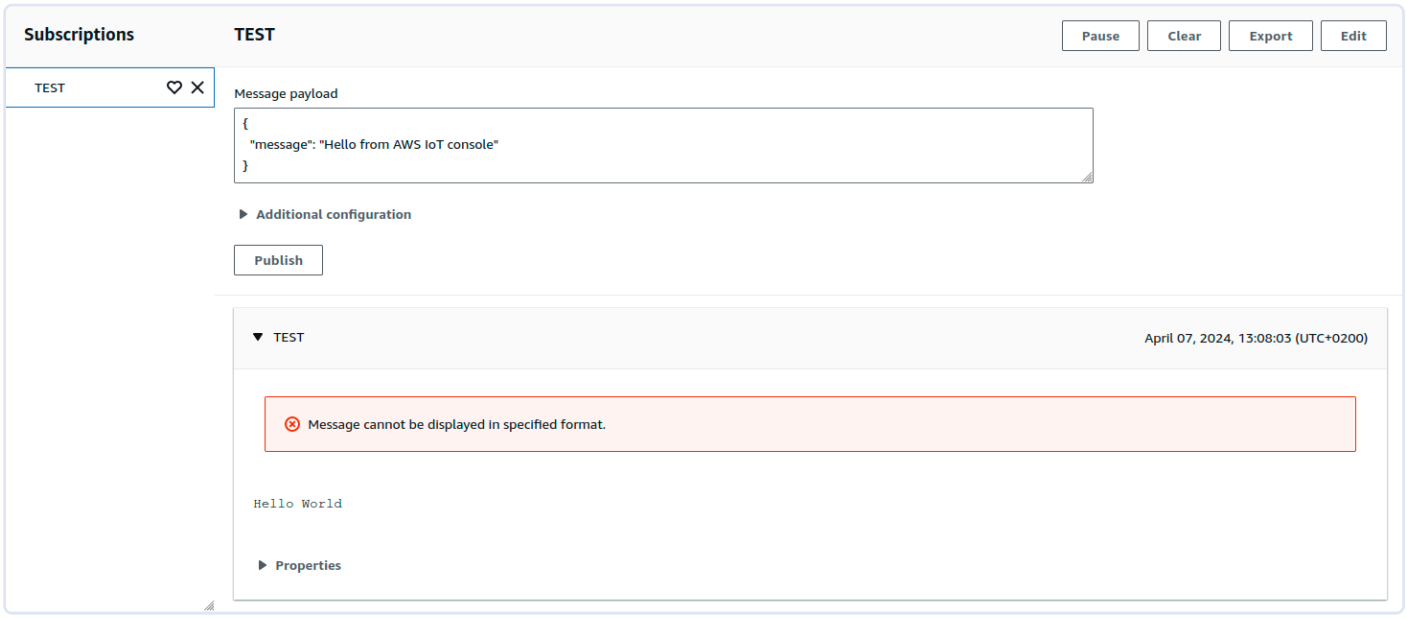


*Figure 4. Les messages envoyés par l'Arduino apparaissent dans la console AWS.*

**Table 3 - Configuration dictionary non-persistent keys**

| Configuration Parameter | Type | Initial Value | Buff Size | Description |
|---|---|---|---|---|
| QoS | R/W | 0 | 1 | QoS level selected for SEND commands |
| Topic1 | R/W | {Empty} | ≥128 | Custom defined topic 1 |
| Topic2 | R/W | {Empty} | | Custom defined topic 2 |
| ... | | | | |
| Topic<Max Topic> | R/W | {Empty} | | Custom defined topic MaxTopic |
| EnableShadow | R/W | 0 | 1 | 0 - disabled, or 1 - enabled |
| Shadow configuration parameters (required only by modules that support the Shadow feature, see 9.2 AWS IoT Device Shadow) | | | | |
| Shadow1 | R/W | {Empty} | 64 | Custom defined named shadow |
| ... | | | | |
| Shadow<MaxShadow> | R/W | {Empty} | | Custom defined named shadow |
| BLE configuration parameters (required only by modules that support BLE host control, see 12 Bluetooth Low Energy) | | | | |
| BLECentral1 | R/W | {Empty} | ≥ 128 | GAP Central discovery/connect configurations. |
| BLECentral2 | R/W | {Empty} | ≥ 128 | |
| ... | | | | |
| BLECentral<MaxBLECentral> | R/W | {Empty} | ≥ 128 | |
| BLEGATT1 | R/W | {Empty} | ≥ 128 | GATT Characteristic definitions (JSON). |
| BLEGATT2 | R/W | {Empty} | ≥ 128 | |
| ... | | | | |
| BLEGATT<MaxBLEGatt> | R/W | {Empty} | ≥ 128 | |
| BLEPeripheral | R/W | {Empty} | ≥ 128 | GAP Peripheral advertising configuration. |

*Figure 5. La mémoire de configuration du module ExpressLink comporte de nombreux paramètres. (Source : AWS [7])*

### Syntaxe SQL

À l'instar d'autres systèmes de bases de données relationnelles, AWS utilise sa propre syntaxe SQL spécifique. Pour en savoir plus sur cette variante spécifique à Amazon, veuillez consulter l'URL *https://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html.*

Une analyse détaillée de la commande d'envoi révèle qu'elle suit le format `SEND[#]` (comme illustré par `SEND1` ci-dessus). Cette structure permet de stocker plusieurs canaux, chacun destiné à être utilisé, dans la mémoire de configuration du module. Chaque canal contient plusieurs dizaines d'attributs, que Amazon décrit de manière exhaustive. La figure 5 présente un extrait du dictionnaire de configuration, illustrant certaines de ces options.

Le format idéal pour la transmission de la charge utile dépend fortement de l'utilisateur. Nous nous concentrerons d'abord sur les messages « facilement analysables » et nous nous conformerons donc à la norme JSON [5].

L'environnement Arduino propose en théorie une variété de bibliothèques facilitant l'assemblage dynamique des charges utiles JSON sans nécessiter une manipulation complexe des chaînes de caractères. Pour nos tests, l'utilisation de telles bibliothèques n'est pas nécessaire. Nous allons simplement adapter la routine d'envoi de notre programme d'exemple, comme indiqué ici :

```
if (millis() - last_send_time >= 10000) {
    response = execute_command
  ("AT+CONF Topic1=TEST", 3000);
    response = execute_command
  ("AT+SEND1 { \"messdaten\":
   { \"temperatur\": 22, \"kaefer\": 0 }}", 5000);
    last_send_time = millis();
}
```

Pour les développeurs moins familiers avec le langage C, il est important de noter que la séquence `\"` (connue sous le nom de séquence d'échappement en C) garantit que le guillemet est interprété comme faisant partie de la chaîne de caractères et non comme le terminateur de la chaîne.

À ce stade, la première version du programme est prête à être exécutée. L'analyseur MQTT mentionné précédemment, servira à confirmer la réception des charges utiles transmises.

## Règles avec SQL

Dans cette étape, nous plongeons dans un univers de fantaisie en concevant un système de surveillance qui mesure l'humidité et détecte la présence de coléoptères du tabac dans une cave à cigares. Ce système nous permettra d'émettre des alertes régulières pour signaler la présence de ces coléoptères du tabac, appelés « kaefer » en allemand :

```
        static unsigned long last_send_time;
        static int laeufer = 0;
        if (millis() - last_send_time >= 10000)
        {
            response = execute_command
               ("AT+CONF Topic1=TEST", 3000);
            if(laeufer++<10)
            {
             response = execute_command
               ("AT+SEND1 {  \"messdaten\":
               {    \"temperatur\": 22,
                   \"kaefer\": 0  }}", 5000);
            }
            else
            {
              laeufer=0;
              response = execute_command
                ("AT+SEND1 {  \"messdaten\":
                {    \"temperatur\": 22,
                   \"kaefer\": 1  }}", 5000);
            }
            last_send_time = millis();
        }
```

L'apparition de messages indiquant divers nombres de coléoptères dans le backend AWS confirme le bon fonctionnement du système. La prochaine étape consiste à signaler au backend AWS que la présence de coléoptères dans la cave à cigares constitue une donnée significative. Pour cela, nous devons établir une règle spécifique. Le gestionnaire de règles est accessible dans le backend AWS à l'adresse suivante *Manage* → *Message routing Rules*.
Lors de la création d'un compte AWS, un assistant vous guide à travers le processus de création de nouvelles règles, illustré par la **figure 6**. Dans la première étape, l'assistant de configuration vous demande de saisir le nom de la règle. Pour cet exemple, j'utiliserai *TamsKaeferRegel*. Aucune description ou texte n'est nécessaire. Lors de la deuxième étape, nous introduisons une requête SQL qui initie un événement. Pour simplifier, Amazon emploie une variante de SQL spécifique pour définir la règle (comme expliqué dans l'encadré "Syntaxe SQL"). Pour notre première expérimentation, nous utiliserons la requête suivante `SELECT messdaten.kaefer FROM 'TEST' WHERE kaefer>0`. Ensuite, nous procéderons à l'étape suivante.

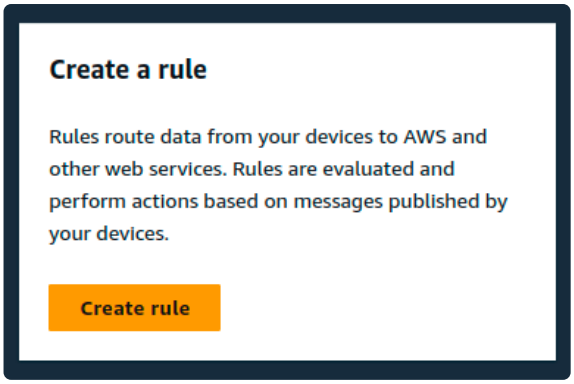


*Figure 6. Cet outil peut maintenant être utilisé pour créer les règles.*

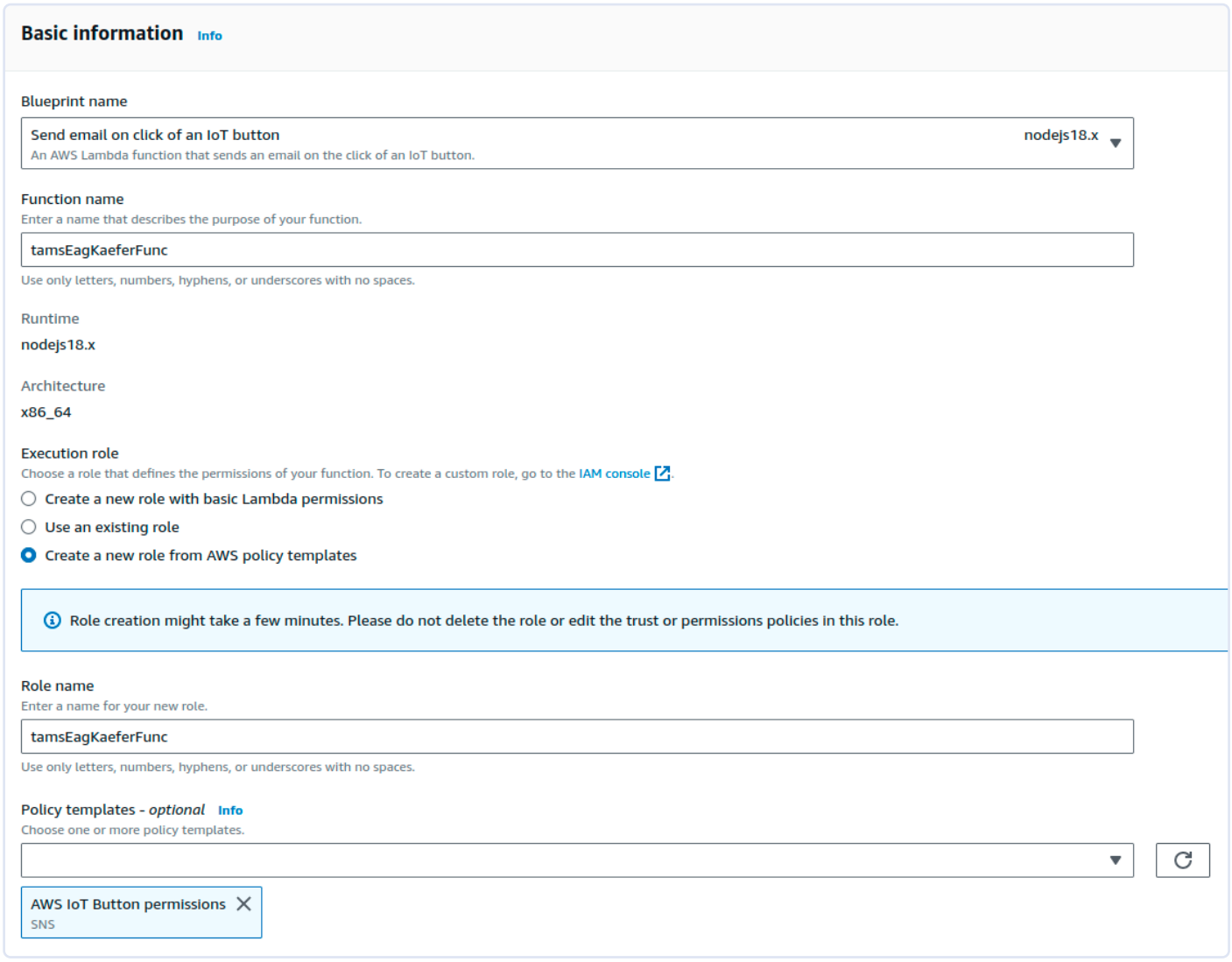


*Figure 7. Les paramètres doivent être conformes au générateur de règles AWS.*

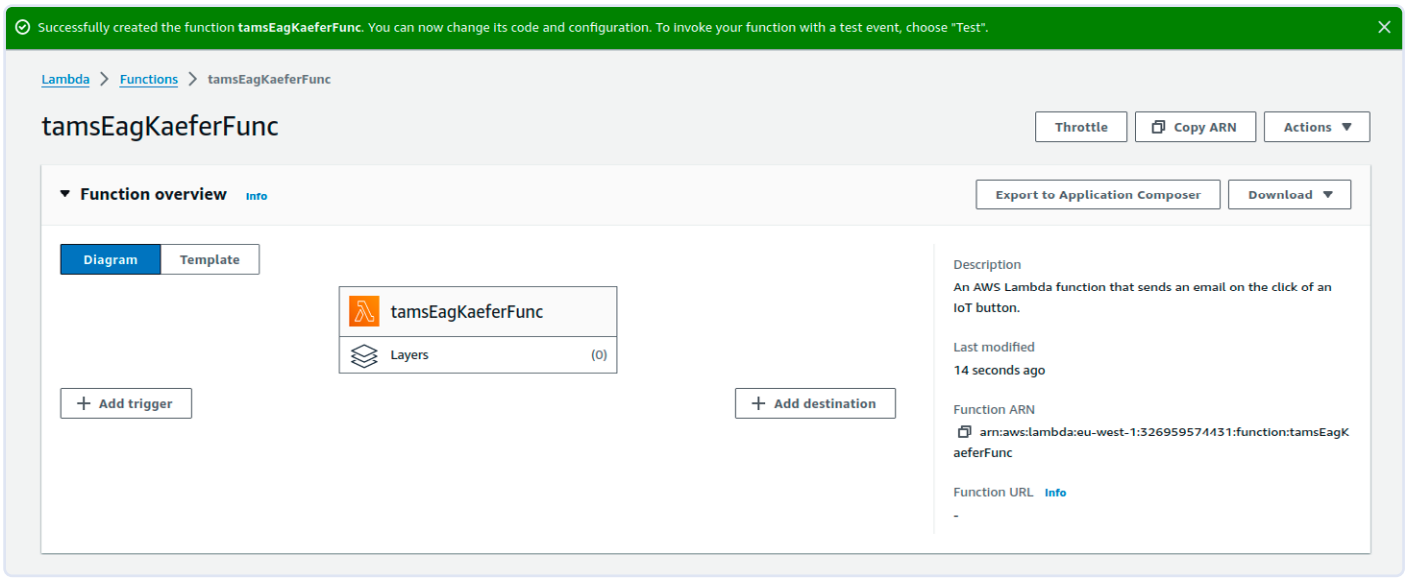


*Figure 8. La bannière verte située en haut de la page nous informe que la fonction cloud AWS a été générée avec succès.*

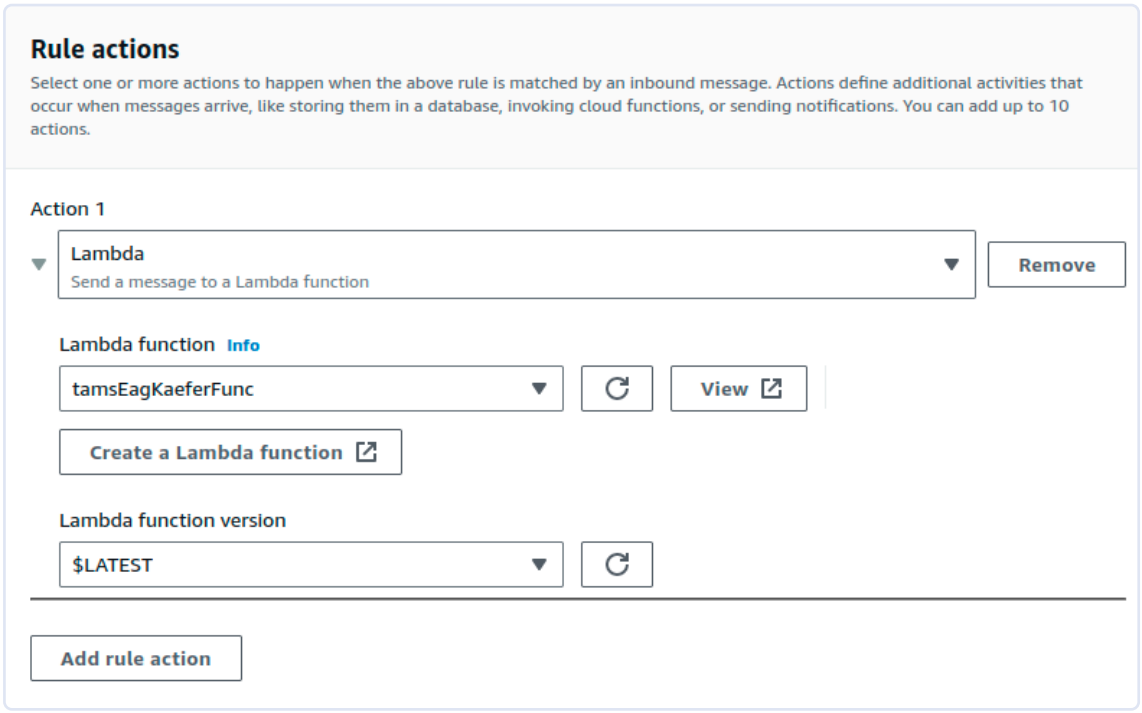


*Figure 9. Définition de l'action de la règle sur les messages reçus.*

À l'étape 3, Amazon vous invite à définir les actions à effectuer par la règle (*Rule Actions*), c'est-à-dire les opérations que le backend AWS doit exécuter lorsqu'un événement déclencheur est détecté. À ce stade, nous sélectionnons l'action *Lambda* : il s'agit de sections de code hébergées dans le cloud que le backend Amazon peut exécuter sans nécessiter de machine virtuelle préconfigurée par le développeur. Amazon prend en charge plus d'une douzaine de langages de programmation différents pour ces fonctions Lambda - la documentation est disponible à l'adresse [6].

Ensuite, cliquez sur le bouton *Create a Lambda Function*, ce qui lancera un assistant dans une nouvelle fenêtre de navigateur pour la création de la fonction dans le cloud. À l'étape suivante, choisissez l'option *Use a Blueprint* pour générer différents modèles de projet. Le modèle que nous utiliserons ici est *IT Automation → Send E-Mail on Click of IoT Button*.

Après avoir choisi le modèle de projet souhaité, défilez vers le bas et saisissez un nom dans le champ *Function Name*. Puisque notre fonction va jouer un rôle actif, il convient de lui attribuer les autorisations nécessaires. Dans la section *Execution Role*, sélectionnez *Create a new Role from AWS Policy Templates* et donnez un nom à ce nouveau rôle.

La **figure 7** montre la configuration prévue pour les étapes suivantes. Dans la section *AWS IoT Trigger*, sélectionnez l'option *Custom IoT Rule*.Ensuite, saisissez votre adresse e-mail dans le champ *Environment Variables*. Cliquez ensuite sur *Create Rules*. À ce stade, un problème peut survenir si le champ *Existing Rules* requiert une entrée et que celle-ci n'est pas fournie. Un message d'erreur `This field is required` s'affiche. Pour résoudre ce problème, cliquez sur le bouton *Remove* dans la section *Triggers*, puis revenez à la section *Environment Variables*. Ensuite, après avoir cliqué sur *Create Function*, le processus de création de la fonction cloud commencera. Ce processus peut prendre un peu de temps en raison des latences du backend. La **figure 8** montre la création réussie de la fonction.

Dans l'étape suivante, retournez à l'autre fenêtre du navigateur pour configurer l'action de la règle, comme illustré par la **figure 9**. Cette action est déclenchée lorsqu'une règle correspond à un message entrant. Cliquez sur *Next* pour revoir les paramètres, et enfin, cliquez sur *Create* pour configurer la nouvelle règle dans le backend AWS.


*Figure 10. Les messages de test sont reçus avec succès.*

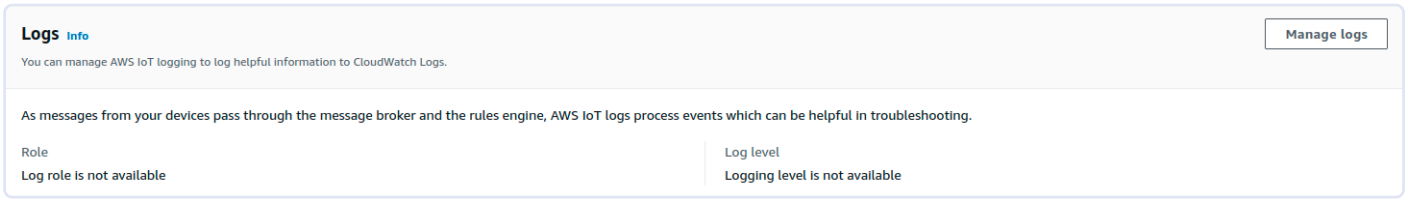


*Figure 11. La possibilité d'enregistrer des événements n'est pas disponible ici.*

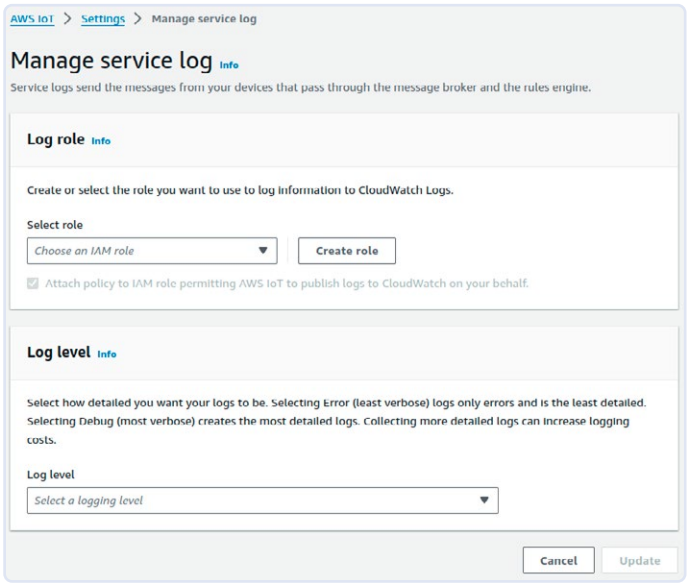


*Figure 12. Ces deux paramètres permettent de définir la configuration de la journalisation.*

Tout devrait désormais être en place dans le backend, en particulier la nouvelle règle dans la section *Message Routing* → *Rules*, qui décrit le « traitement ». Il est temps maintenant de tester le logiciel Arduino. La **figure 10** montre comment les messages envoyés par l'Arduino apparaissent dans le client de test MQTT pour l'évaluation.
Le problème est qu'aucun message n'est reçu à l'e-mail spécifié. Il semble qu'un dépannage soit nécessaire.

## Mise en place d'un Log Watcher

Dans le domaine du cloud computing, les fournisseurs doivent trouver un équilibre entre l'enrichissement de leur offre de services et la prévention de configurations par défaut trop exigeantes, susceptibles d'engendrer des coûts supplémentaires et de décevoir les clients, résultant ainsi en une expérience utilisateur négative. C'est pourquoi de nombreuses fonctionnalités sont initialement désactivées par les fournisseurs. Par exemple, la fonction de journalisation est couramment désactivée dans AWS.

Pour activer la journalisation, retournez à la console AWS IoT, cliquez sur *Settings*, puis faites défiler vers le bas jusqu'à la section *Logs*. Lorsque vous accédez à une console AWS fraîchement configurée, il est possible de voir un avertissement semblable à celui illustré par la **figure 11**, indiquant qu'aucun rôle n'est configuré.
Une fois les tâches initiales accomplies, vous accéderez à l'interface principale de configuration de la journalisation, comme montré dans la **figure 12**. Ici, vous allez configurer le rôle IAM nécessaire pour accéder aux informations et définir le niveau de détail des informations enregistrées. Vous pouvez maintenant cliquer sur le bouton *Create Role* dans le champ *Log Role* pour lancer l'assistant de génération d'un nouveau rôle IAM. Vous avez une certaine liberté pour nommer ce rôle ; dans cet exemple, j'ai choisi *tamsiamlogrole*.
Après avoir choisi et confirmé le nom du rôle, une bannière verte s'affiche en haut de l'écran, signalant que la création du rôle a été réussie. Le message « Policy attached » apparaît alors sous la combo-box, confirmant que le rôle a bien été créé et que les autorisations nécessaires lui ont été correctement attribuées.
Nous pouvons à présent nous rendre dans la section *Log Level*, où nouss sélectionnerons l'option *Debug* pour recueillir le maximum d'informations de journalisation. Une fois l'option *Debug* sélectionnée, cliquez sur *Update* pour appliquer les règles de journalisation.
Il convient de noter que le backend peut parfois signaler une règle de journalisation manquante, comme le montre la **figure 13**.
Dans cette situation, le problème pourrait être lié à la carte de crédit enregistrée. Dans ce cas, accédez à la console générale AWS et vérifiez la validité des informations de votre carte de crédit. Si les problèmes persistaient, il pourrait être nécessaire de créer un nouveau rôle. Une règle correctement configurée est présentée dans la **figure 14**.
À l'étape suivante, il est recommandé de revenir au client de test MQTT et de redémarrer l'Arduino. Prévoyez un peu de temps pour la réception d'une douzaine de messages, car les fonctions de journalisation des fournisseurs de cloud, y compris AWS, ne sont généralement pas immédiates. Avant de procéder aux étapes suivantes, il peut être nécessaire de prévoir jusqu'à une demi-heure de temps d'arrêt - la mise à jour de la liste des sujets contenus dans *Logs* → *Log Groups* se fait très lentement.
L'étape suivante consiste à ouvrir une nouvelle fenêtre de navigateur et à accéder à *https://console.aws.amazon.com/cloudwatch*. Le backend vous redirigera automatiquement vers la console où

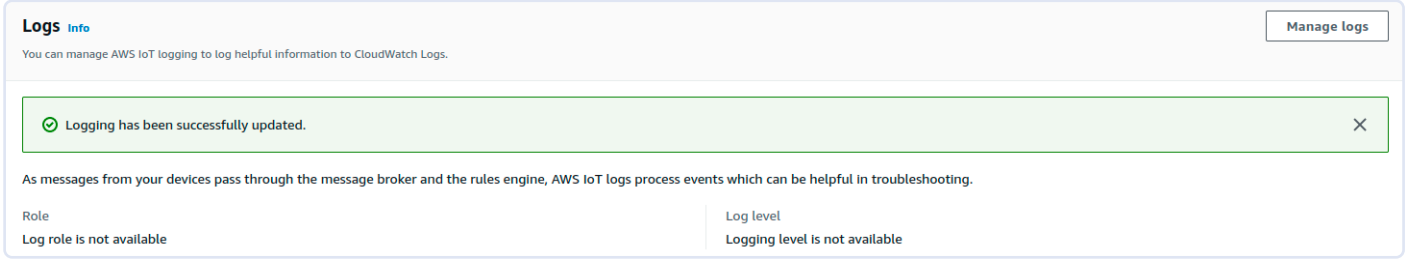


*Figure 13. Message d'erreur indiquant que l'enregistrement n'est pas disponible.*

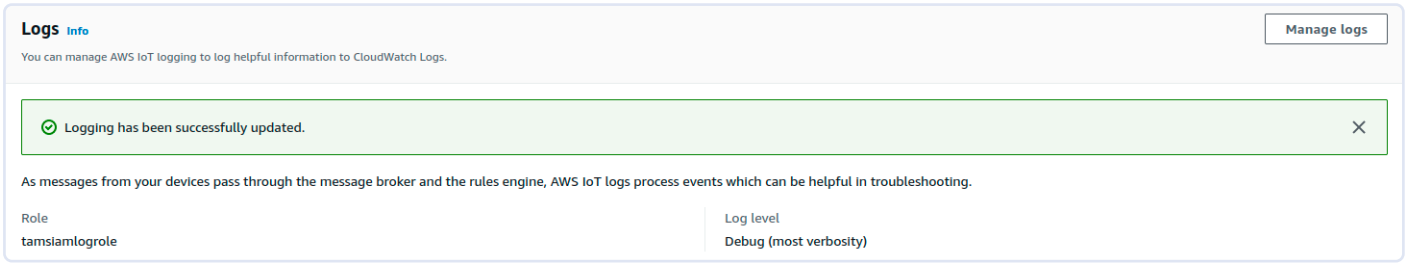


*Figure 14. Aucun message d'erreur, nous pouvons donc passer à l'étape suivante.*


*Figure 15. Les journaux de TheCloudWatch contiennent de nombreuses informations utiles.*

les informations de journalisation *CloudWatch* de votre compte sont disponibles. Ensuite, sélectionnez le groupe de journalisation *AWSIotLogsV2* utilisé par AWS IoT Core pour stocker les données de journalisation liées à l'IoT. Cliquez sur l'un des flux de données pour afficher les informations de journalisation qu'il contient.

Si vous avez utilisé la même requête que moi, vous observerez probablement des messages d'erreur semblables à ceux présentés dans la **figure 15**, qui signalent des problèmes lors du traitement de l'opérateur. Plus précisément, vous pourriez rencontrer une erreur *OperatorEval*, indiquant qu'une expression non définie a été évaluée dans la condition de la règle. Cela entraîne logiquement l'interruption du traitement de la règle, et les avertissements supplémentaires concernant la non-satisfaction de la condition *Where* sont redondants et peuvent être ignorés.

Pour résoudre le problème, retournez aux propriétés de la règle et mettez à jour la requête SQL en utilisant `SELECT messdaten FROM 'TEST' WHERE messdaten.kaefer>0`. Après avoir enregistré la règle, attendez quelques minutes supplémentaires pour qu'elle s'exécute à nouveau, puis ouvrez les métriques disponibles dans le backend Lambda. La **figure 16** montre des invocations occasionnelles de la fonction Lambda qui deviendront visibles.

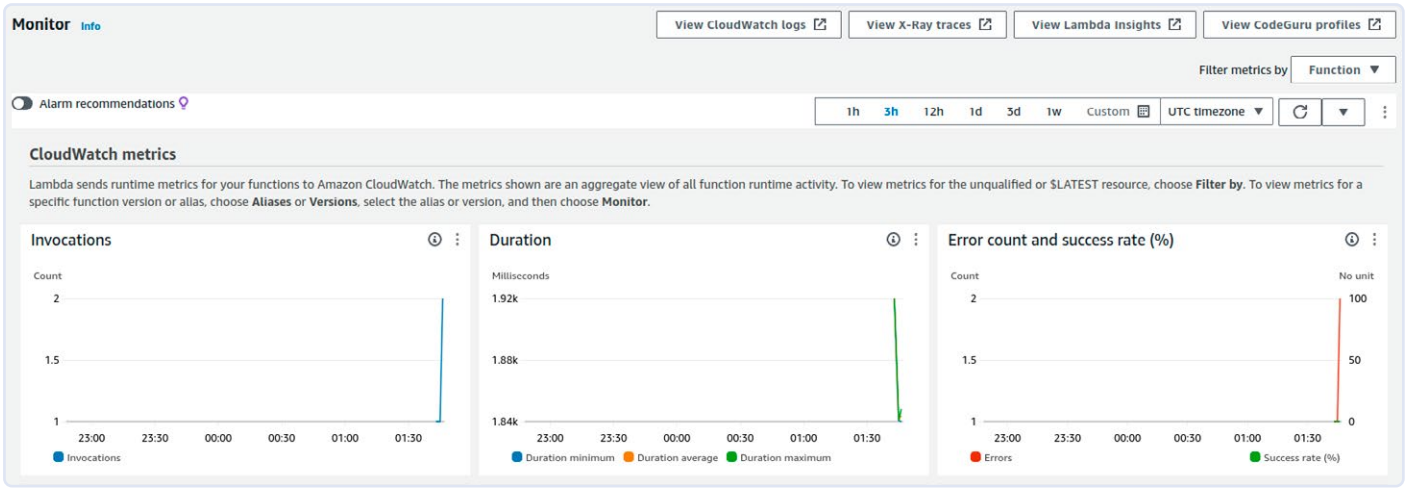


*Figure 16. La fonction Lambda démarre mais échoue en cours d'exécution.*

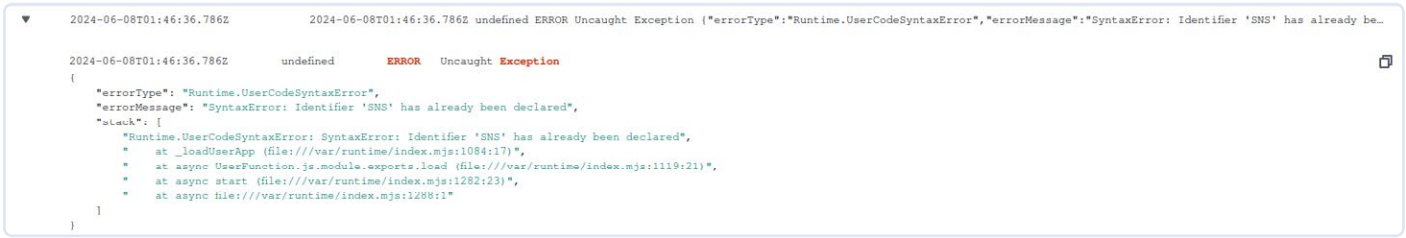


*Figure 17. Une erreur de syntaxe évidente dans la définition de la fonction.*

Pour une analyse plus détaillée, cliquez sur le bouton *View CloudWatch Logs* . Dans le flux de données du journal, vous pourriez observer des erreurs similaires à celles présentées dans la **figure 17** qui signalent des problèmes liés au code comme cette erreur de syntaxe détectée.
L'erreur peut notamment être liée à l'intégration de l'objet SNS responsable de l'envoi des courriels :

```
import { SNS } from '@aws-sdk/client-sns';

const EMAIL = 'tamhan@tamoggemon.com';
const SNS = new SNS();
```

Pour résoudre ce problème, vous pourriez envisager de renommer `SNS` en `mySNS` — bien que ce sujet dépasse le cadre de cet article.

**Important :** l'utilisation d'AWS Logging peut entraîner des coûts significatifs. Il est donc vivement conseillé de désactiver les paramètres de journalisation de niveau élevé avant de terminer une session.

## Une multitude de possibilités

En raison de contraintes d'espace, cet article n'a abordé qu'une fraction des capacités d'AWS IoT. Nous n'avons pas eu l'occasion d'explorer le processus d'envoi de messages depuis le cloud vers un point final, ni de nous pencher sur des visualisations enrichies ou l'intégration de charges utiles de machine learning (ML).
Il convient de noter que de nombreux modules ExpressLink supportent désormais le Bluetooth LE, facilitant ainsi le déploiement d'appareils compatibles Bluetooth. Cependant, cet article s'est concentré sur les principes fondamentaux de l'interaction avec AWS IoT. Nous espérons que cette introduction vous encouragera à expérimenter et à explorer par vous-même les autres fonctionnalités disponibles !

240240-B-04



## Questions ou commentaires ?

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

## À propos de l'auteur


Tam Hanna est un ingénieur spécialisé en électronique, informatique et logiciels depuis plus de 20 ans. Designer indépendant, auteur et journaliste (*instagram.com/tam.hanna*), il consacre son temps libre à l'impression 3D et à la dégustation de cigares haut de gamme.




## LIENS


[1] AWS IoT Core: Developer Guide : https://docs.aws.amazon.com/pdfs/iot/latest/developerguide/iot-dg.pdf
[2] AWS IoT Core pricing: https://aws.amazon.com/iot-core/pricing/
[3] Tam Hanna, « AWS pour Arduino et Cie. » (1) Elektor 7-8/2024 : https://www.elektormagazine.fr/magazine/elektor-349/62995
[4] AWS IoT ExpressLink - documentation : https://docs.aws.amazon.com/iot-expresslink/latest/programmersguide/elpg.html
[5] JSON extensions specification : https://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-json.html
[6] Lambda Functions documentation : https://docs.aws.amazon.com/lambda/
[7] AWS IoT ExpressLink documentation: Configuration dictionary non-persistent keys : https://docs.aws.amazon.com/iot-expresslink/latest/programmersguide/elpg-configuration-dictionary.html#elpg-table3
[8] Téléchargement du logiciel : http://www.elektormagazine.fr/240240-B-04