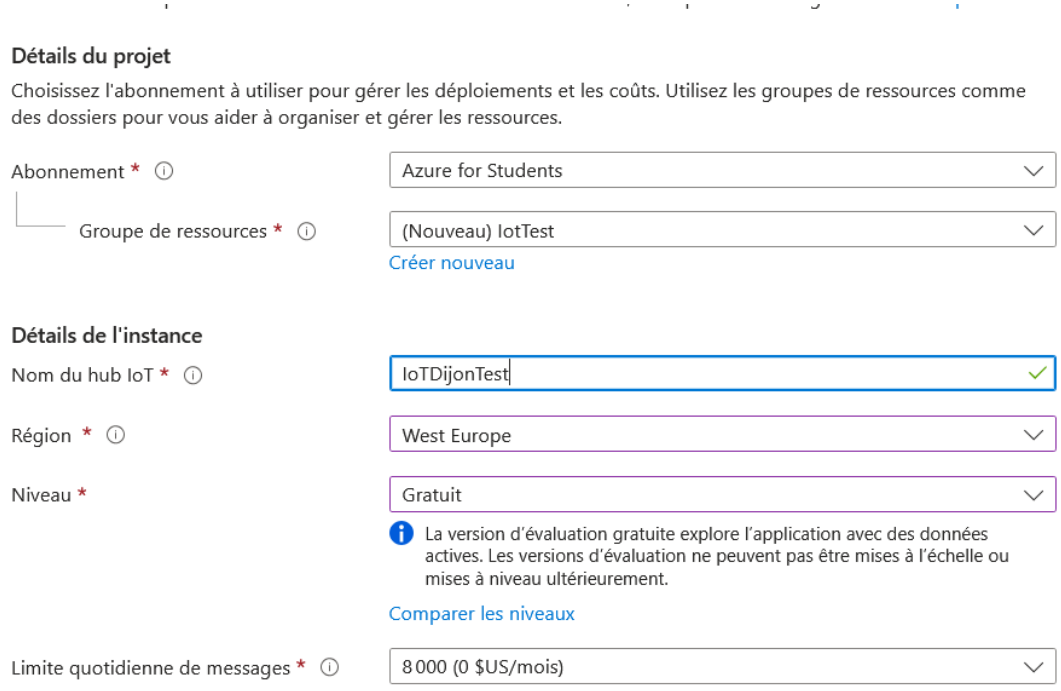
Azure IoT Hub

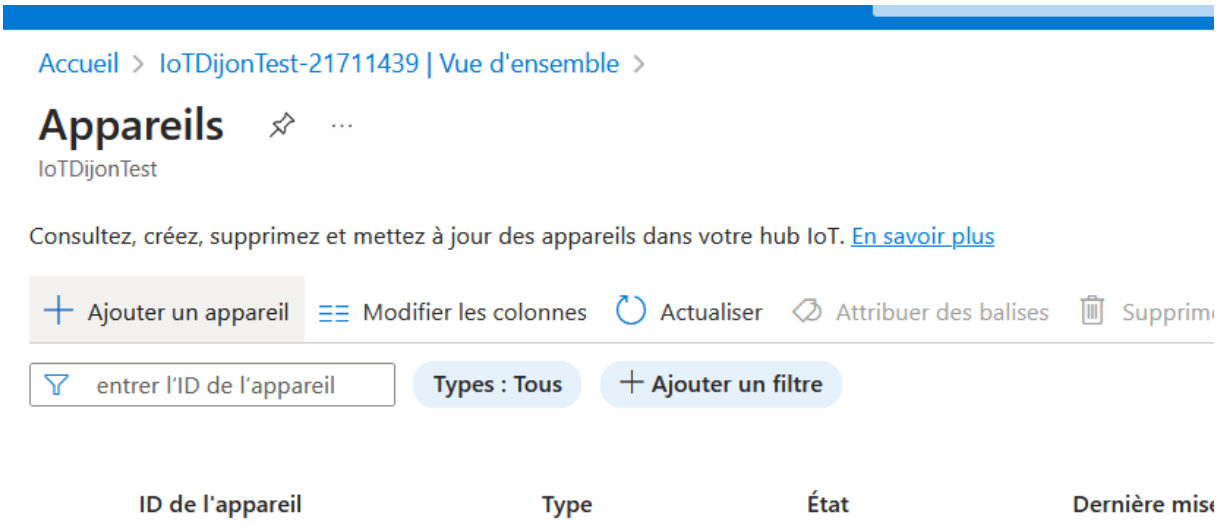
Table des matières

Azure Hub IoT :

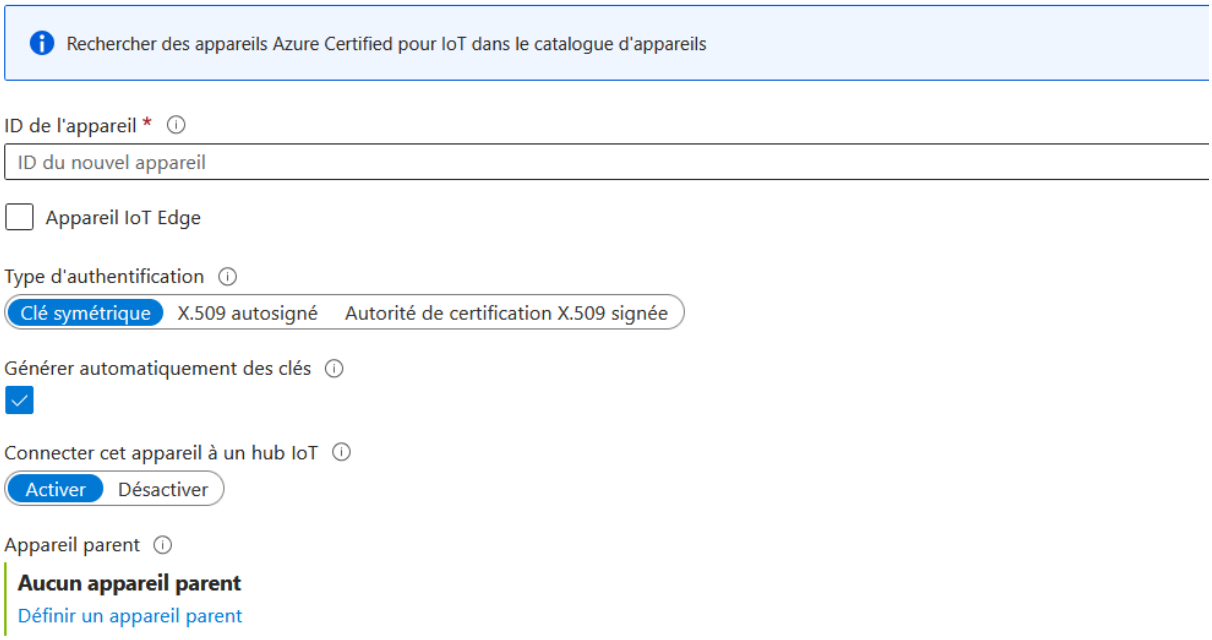
Tout d'abord on créer le hub IoT sur le portail azure



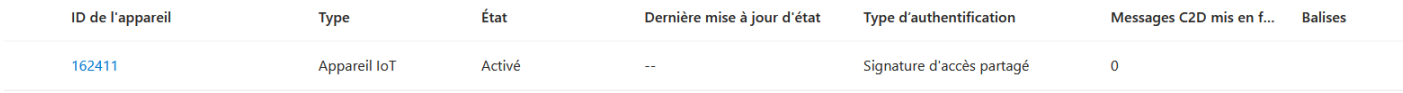
Une fois le hub créer nous devons créer dans ce hub un profil pour un appareil connecté :



Il faut définir a l'appareil un ID et le type d'authentification que l'on veut utiliser l'appareil se connectera grâce à cette **authentification (AppKey)** au hub azure permettant d'envoyé ses donnée dans le Hub IoT.



Le profil de l'appareil est créer :



Installation Azure CLI :

CMD : winget install -e --id Microsoft.AzureCLI

```
PS C:\WINDOWS\system32> winget install -e --id Microsoft.AzureCLI
La source 'msstore' nécessite que vous consultiez les contrats suivants avant de l'utiliser.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
La source nécessite que la région géographique à 2 lettres de l'ordinateur actuel soit envoyée au service principal pour
 fonctionner correctement (par exemple, « ÉTATS-UNIS »).

Acceptez-vous toutes les conditions des contrats sources ?
[Y] Oui  [N] Non: y
Trouvé Microsoft Azure CLI [Microsoft.AzureCLI] Version 2.69.0
La licence d'utilisation de cette application vous est octroyée par son propriétaire.
Microsoft n'est pas responsable des paquets tiers et n'accorde pas de licences à ceux-ci.
Téléchargement en cours https://github.com/Azure/azure-cli/releases/download/azure-cli-2.69.0/azure-cli-2.69.0-x64.msi
                                   68.7 MB / 68.7 MB
Le code de hachage de l'installation a été vérifié avec succès
Démarrage du package d'installation... Merci de patienter.
```

Après l'installation, fermez et rouvrez toute fenêtre de terminal active.
Connecter vous avec : az login. Vérifier connexion avec : az account show

Rajouter l'extension azure-iot : az extension add –name azure-iot

```
PS C:\WINDOWS\system32>
>> az extension add --name azure-iot
  - Installing ..
```

Test d'envoie message au profil de l'appareil :
az iot device send-d2c-message --hub-name MON_IOTHUB --device-id MON_APPAREIL --data "Hello IoT Hub !"

```
PS C:\WINDOWS\system32> az iot device send-d2c-message --hub-name IoTDijonTest --device-id 162411 --data "Hello IoT Hub
!"
```

Pour voir résultat : az iot hub monitor-events --hub-name MON_IOTHUB --

device-id MON_APPAREIL

```
PS C:\WINDOWS\system32>
>> az iot hub monitor-events --hub-name IoTDijonTest --device-id 162411
Dependency update (uamqp 1.2) required for IoT extension version: 0.25.0.
Continue? (y/n) -> y
Updating required dependency...
Update complete. Executing command...
Starting event monitor, filtering on device: 162411, use ctrl-c to stop...
{
    "event": {
        "origin": "162411",
        "module": "",
        "interface": "",
        "component": "",
        "payload": "Hello IoT Hub !"
    }
}
```

Test envoie de donnée d'un appareil connecté :

Nous allons mettre en place un appareil connecté simuler et envoyer des données sur le Hub IoT. Il faut faire un script python qui vas simuler un appareil connecté :

```
  GNU nano 7.2                                  simulateur.py
import time
import random
from azure.iot.device import IoTHubDeviceClient, Message

# Remplace par ta chaîne de connexion IoT Hub
CONNECTION_STRING = "HostName=IoTDijonTest.azure-devices.net;DeviceId=162411;SharedAccessKey=lnfiNsszOu>

# Création du client IoT
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

print("Connexion à Azure IoT Hub...")
client.connect()

try:
    while True:
        temperature = round(random.uniform(20.0, 30.0), 2)
        humidity = round(random.uniform(40.0, 60.0), 2)

        # Création du message JSON
        msg = Message(f'{{"temperature": {temperature}, "humidity": {humidity}}}')
        msg.content_encoding = "utf-8"
        msg.content_type = "application/json"

        print(f"Envoi du message : {msg}")
        client.send_message(msg)

        time.sleep(5)  # Attendre 5 secondes avant le prochain envoi

except KeyboardInterrupt:
    print("Déconnexion...")
    client.disconnect()
```

On précise dans la variable CONNECTION_STRING **l'ID de l'appareil** que

l'on a défini plus tôt, **le lien permettant d'accéder au hub IoT** et la **clé lié à l'appareil**. On réutilise cette variable pour se connecter a azure et récupérer le profil pour l'appareil connecté avec l'ID 162411 que l'on a créé.

Une fois connecté au hub azure, notre appareil connecté (simulé) vas envoyer des données vers ce hub **device to cloud (D2C).**

Pour voir ce que l'appareil envoie au hub :

```
PS C:\WINDOWS\system32>
>> az iot hub monitor-events --hub-name IoTDijonTest --device-id 162411
Starting event monitor, filtering on device: 162411, use ctrl-c to stop...
```

TEST :

```
(azure-iot-env) urtic@THINKPAD-E16:~$ python simulateur.py
Connexion à Azure IoT Hub...
Envoi du message : {"temperature": 26.77, "humidity": 46.93}
Envoi du message : {"temperature": 26.62, "humidity": 57.28}
Envoi du message : {"temperature": 28.43, "humidity": 42.41}
```

```
PS C:\WINDOWS\system32>
>> az iot hub monitor-events --hub-name IoTDijonTest --device-id 162411
Starting event monitor, filtering on device: 162411, use ctrl-c to stop...
{
    "event": {
        "origin": "162411",
        "module": "",
        "interface": "",
        "component": "",
        "payload": {
            "temperature": 26.77,
            "humidity": 46.93
        }
    }
}
{
    "event": {
        "origin": "162411",
        "module": "",
        "interface": "",
        "component": "",
        "payload": {
            "temperature": 26.62,
            "humidity": 57.28
        }
    }
}
{
    "event": {
        "origin": "162411",
        "module": "",
        "interface": "",
        "component": "",
        "payload": {
            "temperature": 28.43,
            "humidity": 42.41
        }
    }
}
```

Comment recevoir des commandes Cloud To Device (C2D) :

Code python pour simuler un appareil IoT prêt à recevoir des données :

```python
from azure.iot.device import IoTHubDeviceClient

# Remplace par la chaîne de connexion de ton appareil IoT dans Azure
CONNECTION_STRING = "HostName=IoTDijon.azure-devices.net;DeviceId=162411;SharedAccessKey=lnfiNsszOuKxeexb+dUUdLCFEjkRACoh8QzStSi1e/Q="

# Création du client IoT
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

# Fonction de gestion des commandes reçues
def message_handler(message):
    print(f"📩 Commande reçue : {message.data.decode()}")

    # Vérifier s'il y a des propriétés dans le message
    if message.custom_properties:
        print("📌 Propriétés :", message.custom_properties)

# Associer le handler au client
client.on_message_received = message_handler

print("☑ En attente de commandes... (Ctrl+C pour arrêter)")
client.connect()

try:
    while True:
        pass  # Boucle infinie pour écouter les commandes
except KeyboardInterrupt:
    print("🔴 Arrêt du script")
    client.disconnect()
```

Ce que fait ce script :

- Se connecte à Azure IoT Hub avec la **chaîne de connexion de ton appareil**.
- Attend une commande (message Cloud-to-Device).
- Affiche le message et ses propriétés si elles existent.

Commande envoyée :

```
PS C:\WINDOWS\system32>
>> az iot device c2d-message send --hub-name IoTDijonTest --device-id 162411 --data "Allume la LED"
PS C:\WINDOWS\system32>
>> az iot device c2d-message send --hub-name IoTDijonTest --device-id 162411 --data "Eteint la LED"
PS C:\WINDOWS\system32> _
```

Lancer le script et voir si l'appareil connecté simulé reçois les commandes :

```
(azure-iot-env) urtic@THINKPAD-E16:~$ python iot_receive_command.py
☑ En attente de commandes... (Ctrl+C pour arrêter)
📩 Commande reçue : Allume la LED
📩 Commande reçue : Eteint la LED
```

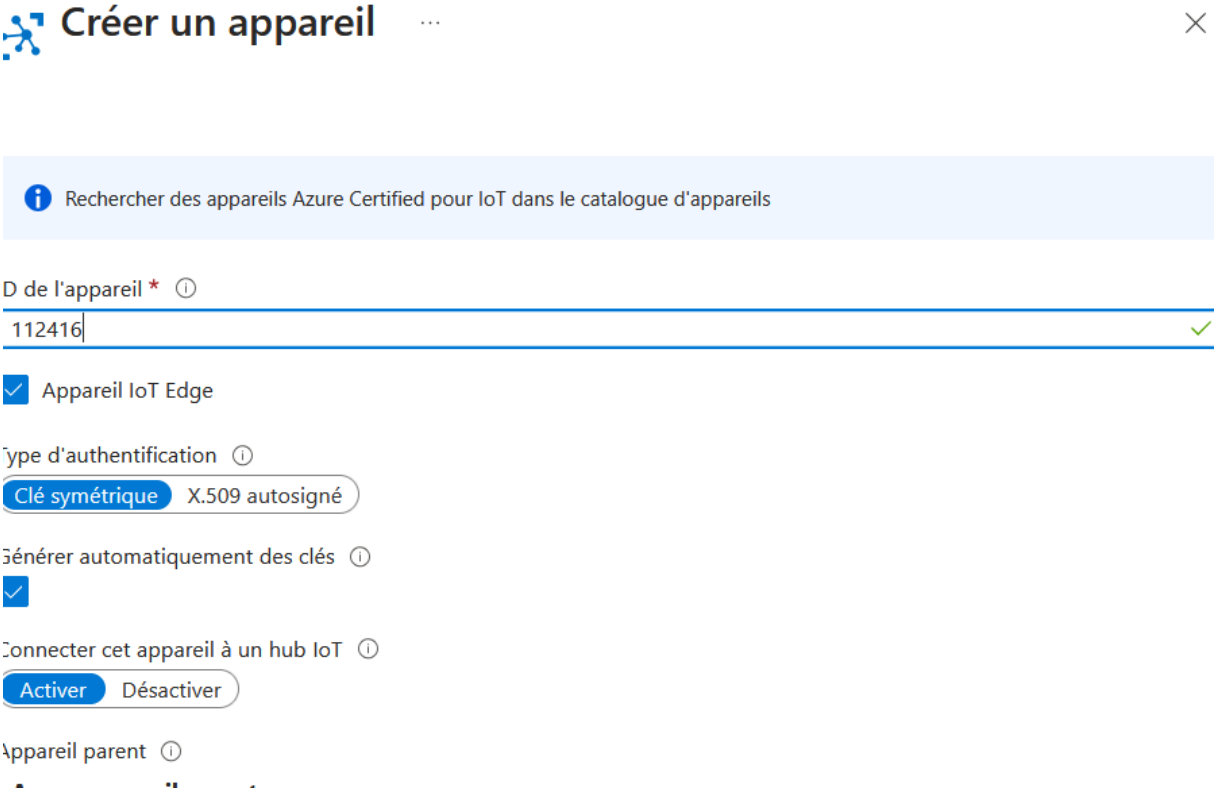
IoT edge :

Un appareil IoT edge permet de traiter les données avant de les envoyés vers un serveur applicatif.

Exemple de scénario dans le cadre de LoRaWAN et Azure IoT Edge :

- Un capteur LoRaWAN génère des données de température toutes les minutes.
- IoT Edge reçoit ces données, mais vous avez configuré un module pour filtrer et ne transmettre que les valeurs de température supérieures à 30°C, car c'est la température seuil au-delà de laquelle une action est requise.
- Une fois filtrées, les données sont envoyées à Azure IoT Hub ou directement à un serveur applicatif pour un traitement ultérieur

Créer un profil pour l'appareil IoT edge :



Maintenant nous allons créer l'appareil IoT edge sur une VM Ubuntu.

IoT Edge a des dépendances sur Docker et le service d'identité IoT. Installez les dépendances en utilisant les commandes suivantes :

sudo snap install docker

sudo snap install azure-iot-identity

Installez IoT Edge à partir du Snap Store :

sudo snap install azure-iot-edge

« Par défaut, les snaps n'ont pas de dépendances, ne sont pas approuvés et sont strictement restreints. Par conséquent, les snaps doivent être connectés à d'autres snaps et à des ressources système après l'installation. Utilisez les commandes suivantes pour connecter le service d'identité IoT et les snaps IoT Edge entre eux et aux ressources système. »

#----------

IoT Identity Service

#----------

Connect the Identity Service snap to the logging system

and grant permission to query system info

sudo snap connect azure-iot-identity:log-observe

sudo snap connect azure-iot-identity:mount-observe

sudo snap connect azure-iot-identity:system-observe

sudo snap connect azure-iot-identity:hostname-control

If using a TPM, enable TPM access

sudo snap connect azure-iot-identity:tpm

#----------

IoT Edge

#----------

Connect to your /home directory to enable writing support bundles

sudo snap connect azure-iot-edge:home

Connect to logging and grant permission to query system info

sudo snap connect azure-iot-edge:log-observe

sudo snap connect azure-iot-edge:mount-observe

sudo snap connect azure-iot-edge:system-observe

sudo snap connect azure-iot-edge:hostname-control

Allow IoT Edge to connect to the /var/run/iotedge folder and use sockets

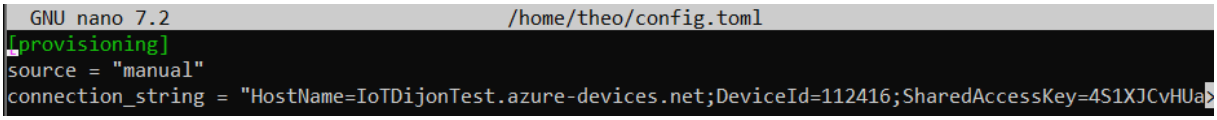
sudo snap connect azure-iot-edge:run-iotedge

Connect IoT Edge to Docker

sudo snap connect azure-iot-edge:docker docker:docker-daemon

Une fois ceci réaliser nous devons créer un fichier de configuration pour préciser à l'appareil IoT Edge son id le lien vers le IoT HUB et la clé symétrique créer plus haut.

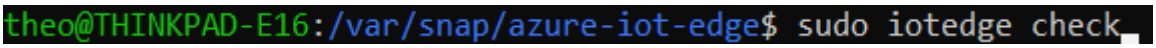
sudo nano ~/config.toml



Ensuite on définit le fichier de configuration que IoT Edge doit utiliser :

sudo snap set azure-iot-edge raw-config="$(cat ~/config.toml)"

On vérifie ensuite que tout est fonctionnel en faisant cette commande :



et le résultat devrai ressembler à cela :

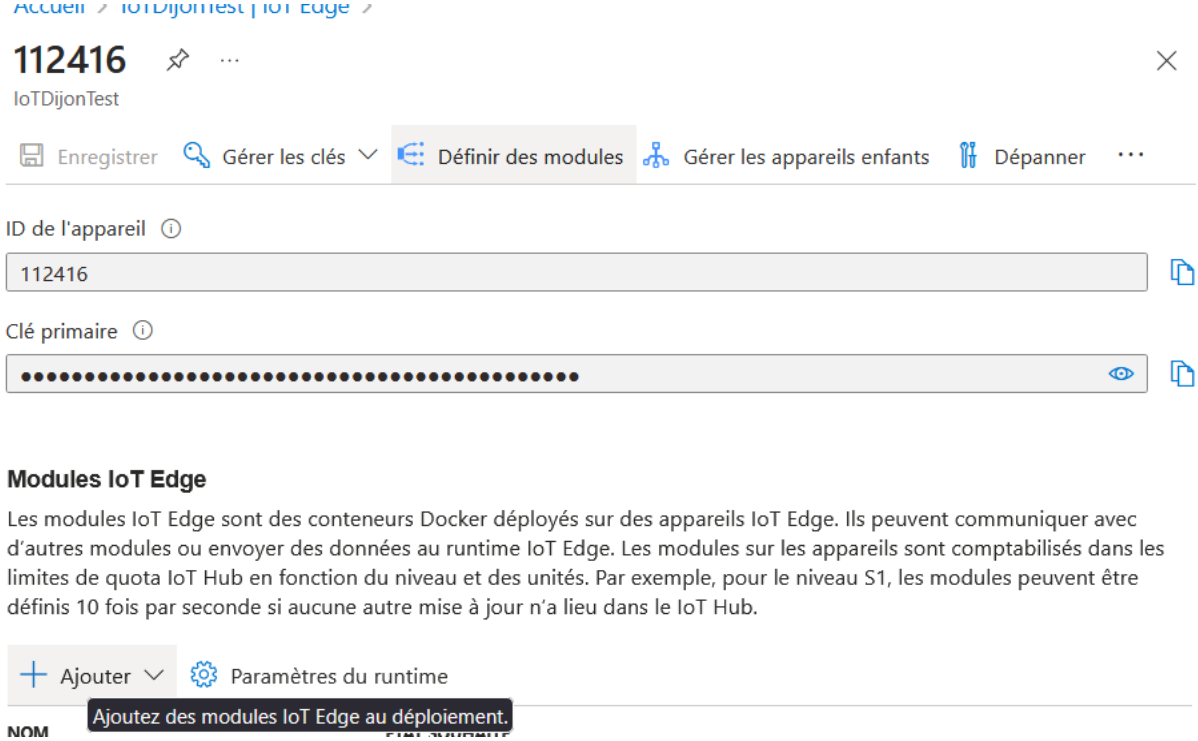
```
Configuration checks
--------------------
√ aziot-edged configuration is well-formed - OK
√ configuration up-to-date with config.toml - OK
√ container engine is installed and functional - OK
√ configuration has correct URIs for daemon mgmt endpoint - OK
√ aziot-edge package is up-to-date - OK
√ container time is close to host time - OK
‼ DNS server - Warning
    Container engine is not configured with DNS server setting, which may impact connectivity to IoT Hub
.
    Please see https://aka.ms/iotedge-prod-checklist-dns for best practices.
    You can ignore this warning if you are setting DNS server per module in the Edge deployment.
‼ production readiness: logs policy - Warning
    Container engine is not configured to rotate module logs which may cause it run out of disk space.
    Please see https://aka.ms/iotedge-prod-checklist-logs for best practices.
    You can ignore this warning if you are setting log policy per module in the Edge deployment.
‼ production readiness: Edge Agent's storage directory is persisted on the host filesystem - Warning
    The edgeAgent module is not configured to persist its /tmp/edgeAgent directory on the host filesyste
m.
    Data might be lost if the module is deleted or updated.
    Please see https://aka.ms/iotedge-storage-host for best practices.
× production readiness: Edge Hub's storage directory is persisted on the host filesystem - Error
    Could not check current state of edgeHub container
√ Agent image is valid and can be pulled from upstream - OK
√ proxy settings are consistent in aziot-edged, aziot-identityd, moby daemon and config.toml - OK

Connectivity checks
-------------------
√ container on the default network can connect to upstream AMQP port - OK
√ container on the default network can connect to upstream HTTPS / WebSockets port - OK
√ container on the IoT Edge module network can connect to upstream AMQP port - OK
√ container on the IoT Edge module network can connect to upstream HTTPS / WebSockets port - OK
31 check(s) succeeded.
3 check(s) raised warnings. Re-run with --verbose for more details.
1 check(s) raised errors. Re-run with --verbose for more details.
2 check(s) were skipped due to errors from other checks. Re-run with --verbose for more details.
```

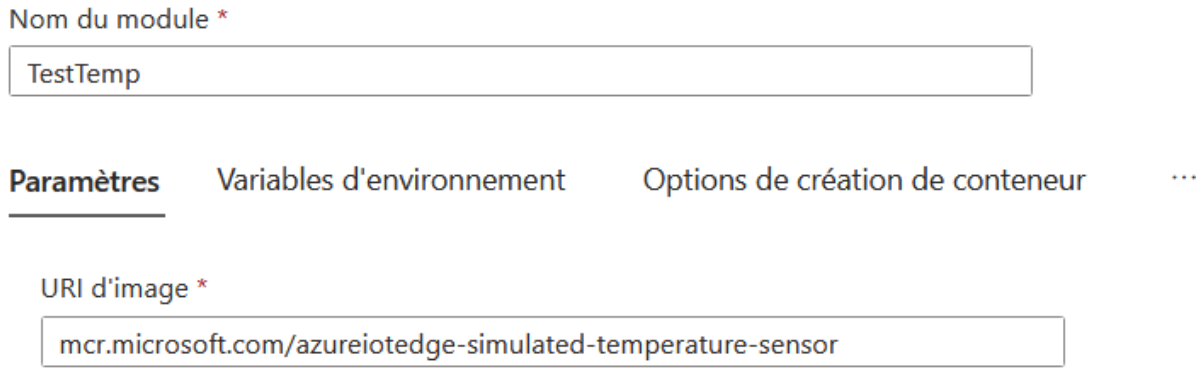
L’erreur est normal car nous devons maintenant définir un module pour l’appareil IoT edge.

On retourne dans les options de notre appareil

Et on lui définit un module :

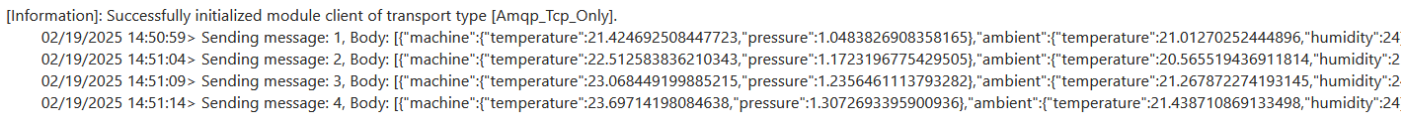


Ici nous allons prendre le module **Simulated-temperature-sensor** car nous n'avons aucun appareil connecté donc il va simuler l'envoie de donnée d'un appareil connecté



Nom	Type	Spécifié dans le déploiement	Signalé par l'appareil	État du runtime	Code de sortie
$edgeAgent	Module système IoT Edge	✓ Oui	✓ Oui	running	NA
$edgeHub	Module système IoT Edge	✓ Oui	✓ Oui	running	NA
TestTemp	Module personnalisé IoT Edge	✓ Oui	✓ Oui	running	NA

En cliquant sur running nous avons plus de détail sur les modules :



On voit ici l'envoie de donnée, nous pouvons ensuite rajouter un module Azure Stream Analytics qui vas permettre par exemple de ne prendre que les données qui donne une température dépassant les 30°.

Conclusion :

Un **appareil IoT Edge** permet de **traiter, analyser et filtrer** les données localement avant de les envoyer au cloud. Il réduit la **latence**, optimise la **bande passante** et permet des **décisions en temps réel** sans dépendre d'Internet. Grâce à des **modules personnalisés**, il peut exécuter des algorithmes d'IA, appliquer des règles métiers ou déclencher des actions locales. C'est une solution idéale pour les environnements où la **réactivité**, la **sécurité** et l'**efficacité des communications** sont essentielles, comme l'industrie, la santé ou **les villes intelligentes**.