Accès distant à un Raspberry Pi via Modem LTE (IoT)

Accès distant à un Raspberry Pi via Modem LTE (IoT)

---

1. Module LTE utilisé : SIM8230G-M2 RedCap

Le modem utilisé dans ce projet est le module **SIM8230G-M2**, un module 5G RedCap basé sur la norme 3GPP Release 17.

Compatibilité avec d'autres modems

Le script reste compatible avec la majorité des modems LTE/5G utilisant des commandes AT standards.

Installation du module

Documentation officielle : https://www.waveshare.com/wiki/SIM8230G-M2

---

2. Objectif

Ce projet vise à établir un accès distant fiable à un Raspberry Pi connecté via modem LTE, en contournant les contraintes NAT habituelles des réseaux mobiles. Les points couverts sont :

- connexion LTE via APN dédié
- tunnel SSH inverse vers un serveur intermédiaire
- accès distant malgré le NAT opérateur

- reconnexion automatique en cas de perte réseau

---

3. Contexte réseau

Dans un réseau LTE, le Raspberry Pi est placé derrière un **NAT opérateur** : le modem obtient une adresse IP côté réseau, mais le Raspberry est en réalité derrière une adresse privée non accessible depuis l'extérieur.

Conséquences directes :

- impossible de se connecter directement en SSH depuis l'extérieur
- impossible de joindre le Raspberry depuis Internet

Solution retenue : reverse SSH (tunnel SSH inverse).

---

4. Accès via serveur de logs

L'accès au Raspberry ne se fait **pas directement**, mais via un serveur intermédiaire appelé serveur de logs. Ce serveur est accessible à la fois depuis le Raspberry (via le réseau LTE) et depuis les postes d'administration.

Le principe est le suivant :

- le Raspberry initie lui-même une connexion SSH vers le serveur de logs
- il expose un port local redirigé sur le serveur
- l'administrateur se connecte ensuite à ce port depuis le serveur

```
ssh localhost -p 2222
```

Cette approche présente plusieurs avantages :

- tous les membres de l'équipe ont accès au serveur de logs
- les connexions directes vers les équipements sont évitées
- l'accès aux équipements IoT est centralisé
- la gestion des accès SSH est simplifiée

> **autossh** est utilisé pour maintenir le tunnel actif : il relance automatiquement la connexion SSH en cas de déconnexion, sans intervention manuelle.

5. Préparation du Raspberry Pi

Ces étapes sont à réaliser en amont, directement sur le Raspberry Pi, avant de déployer le script.

Création d'un utilisateur dédié

```
# Création d'un utilisateur dédié
sudo adduser iot

# Ajout aux droits sudo
sudo usermod -aG sudo iot
```

Génération de la clé SSH

```
ssh-keygen -t ed25519 -f ~/.ssh/modem_tunnel -N ""
```

6. Script complet

```
#!/bin/bash

# Arrêt du script en cas d'erreur
set -euo pipefail

# Port série du modem
PORT="/dev/ttyUSB2"

# Nombre max d'essais modem
max_tries=100
try=0

# IP de test Internet
ping_ip="8.8.8.8"

# Interface réseau modem
iface="usb0"
```

```
# Nombre max d'essais réseau
max_ping_tries=100

# Paramètres SSH
SSH_HOST="192.108.119.79"
SSH_USER="procom"
SSH_KEY="/home/iot/.ssh/modem_tunnel"

# Demande une IP via DHCP
sudo dhclient -4 -v usb0 || true

# Ajout des IDs USB modem
bind_ids() {
  sudo sh -c 'echo "1e0e 9073" > /sys/bus/usb-serial/drivers/option1/new_id'
  sudo sh -c 'echo "1e0e 9071" > /sys/bus/usb-serial/drivers/option1/new_id'
  sudo sh -c 'echo "1e0e 9072" > /sys/bus/usb-serial/drivers/option1/new_id'
  sudo sh -c 'echo "1e0e 9078" > /sys/bus/usb-serial/drivers/option1/new_id'
  sudo sh -c 'echo "1e0e 907b" > /sys/bus/usb-serial/drivers/option1/new_id'
}

# Attente du modem
wait_for_port() {
  while [ ! -e "$PORT" ]; do
    try=$((try+1))

    # Reboot si modem absent trop longtemps
    if [ "$try" -ge "$max_tries" ]; then
      sudo reboot
    fi

    bind_ids
    sleep 5
  done
}

# Envoi d'une commande AT au modem
send_at() {
  local cmd="$1"
  echo -e "${cmd}\r" | sudo tee "$PORT" >/dev/null
```

```
}

wait_for_port

# Initialisation modem
send_at "AT"
send_at "AT+CFUN=0"
sleep 5
send_at "AT+CFUN=1"
sleep 5

# Configuration APN
send_at 'AT+CGDCONT=1,"IP","iot"'
sleep 2

# Sélection opérateur
send_at 'AT+COPS=1,2,"99970"'
sleep 5

# Test réseau
ping_ok() {
  ping -I "$iface" -c 1 -W 2 "$ping_ip" >/dev/null 2>&1
}

# Reset modem
restart_module() {
  send_at "AT+CFUN=0"
  sleep 10
  send_at "AT+CFUN=1"
  sleep 5
  send_at 'AT+CGDCONT=1,"IP","iot"'
  sleep 2
  send_at 'AT+COPS=1,2,"99970"'
  sleep 10
}

# Attente de l'attachement réseau
sleep 20

# Boucle de connexion
```

```
for i in $(seq 1 "$max_ping_tries"); do
  if ping_ok; then
    break
  fi
  restart_module
done

# Reboot si pas de réseau après toutes les tentatives
if ! ping_ok; then
  sudo reboot
fi

# Tunnel SSH permanent
while true; do

  autossh -M 0 \
    -i "$SSH_KEY" \
    -o StrictHostKeyChecking=no \
    -o ServerAliveInterval=30 \
    -o ServerAliveCountMax=3 \
    -N \
    -R 2222:localhost:22 \
    ${SSH_USER}@${SSH_HOST}

  # Si le tunnel tombe et que le réseau est absent → reboot
  if ! ping_ok; then
    sudo reboot
  fi

  sleep 10
done
```

---

7. Démarrage automatique

Le script est déclaré comme un service systemd pour être lancé automatiquement au démarrage du Raspberry Pi.

Créer le fichier `/etc/systemd/system/modem.service` :

```
[Unit]
Description=Modem LTE launcher
After=network-online.target

[Service]
ExecStart=/usr/local/bin/modem-launch.sh
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Puis activer le service :

```
sudo systemctl daemon-reload
sudo systemctl enable modem.service
```

---

8. Paramètres à adapter

Paramètre	Valeur par défaut	Description
`PORT`	`/dev/ttyUSB2`	Port série du modem
`APN`	`iot`	APN déclaré dans la configuration réseau
Opérateur	`99970`	Code PLMN de l'opérateur
`iface`	`usb0`	Interface réseau du modem
`SSH_USER`	`procom`	Utilisateur SSH sur le serveur de logs
`SSH_HOST`	`192.108.119.79`	IP du serveur de logs
Port tunnel	`2222`	Port exposé sur le serveur pour rebondir
`SSH_KEY`	`/home/iot/.ssh/modem_tunnel`	Chemin vers la clé SSH privée

Revision #2
Created 2026-04-29 21:14:09 UTC by Maxime Boison
Updated 2026-04-29 21:19:26 UTC by Maxime Boison