Local API利用マニュアル

はじめに

MODEセンサーゲートウェイは、さまざまな種類のセンサーからデータを収集しMODEセンサークラウドにデータを送信し、データの可視化や監視などを行うことができます。

MODEセンサーゲートウェイを使うことで、遠隔地からのセンサーの管理、ソフトウェアの遠隔ファームウェアアップデート(OTAアップデート)、センサーデータの到着保証など、長期間の運用に耐える実世界データの収集システムに必要な機能を利用することができます。

- OTAアップデート
- デバイスやセンサーの管理機能
- コマンド受信機能
- 到達保証機能(Guarantee Delivery)による時系列データベース(TSDB)へのデータ送信
- IoTデータのイベントとTSDBへのリアルタイム送信

MODEセンサーゲートウェイで標準的にサポートされていない各種のセンサーからのデータ収集をしたい場合があります。このような用途のために、任意のプログラミング言語を使って簡単なプログラムを書くことで標準的にサポートされていないセンサーからのデータ収集を可能にする仕組みがLocal APIです。

Local APIは MODEセンサーゲートウェイ上で動作するHTTP APIです。Local APIはゲートウェイマシン上からしかアクセスできないようにすることで、APIの利用を非常に容易にしています。

新しい種類のセンサーからデータを収集するためにはゲートウェイ上で動作する次のようなプログラムを開発します。

1. これから送信するセンサーデータの種類についてLocal APIを使い宣言する
2. センサーからデータを読み出す
3. センサーデータをLocal APIを使って送る
4. 2に戻る

Local API チュートリアル

このセクションでは、MODEセンサーゲートウェイマシンのCPU温度をセンサーデータに見立て、クラウドへ5秒おきにデータを送信するプログラムを開発してみます。

ログイン

MODEセンサーゲートウェイとディスプレイをHDMIにて接続し、USBキーボードを接続しコンソール・ログイン画面を表示させます。MODE社よりあらかじめ伝えられたIDとパスワードを利用してログインします。

コーディング

簡単に作成できる Bash スクリプトで書いてみましょう。このチュートリアルではコードの修正点をスクリプトファイルに反映しながら進めていきます。コードの完成形は本章の末尾に記載しています。

シェルスクリプトファイルの作成

まずは、ホームディレクトリに`cpu_temp.sh`ファイルを作成し、起動スクリプトに`/bin/bash`を指定します。

```
$ vi cpu_temp.sh
#!/bin/bash
```

センサーの宣言

最初にやるべき事はLocal APIに対してセンサーの存在を宣言することです。宣言を行うにはアナウンスAPIを利用します。アナウンスAPIには、センサーの識別IDとセンサーデータのスキーマを知らせます。宣言はJSON形式にて記述して、このチュートリアルではcurlコマンドを用いてHTTPでAnnounce APIにPOSTします。

先ほどの`cpu_temp.sh`に下記を追記します。

```
announce_json=$(cat <<JSON
{
  "model": "CUSTOM",
  "id": "MyNUC",
  "sensors": [
    "TEMPERATURE:0"
  ]
}
JSON
)

curl -X POST -d "$announce_json" \
  --header "Content-Type: application/json" \
  http://localhost:55299/announce
```

開発版のMODEセンサーゲートウェイで設定できるセンサーの”model”は常にCUSTOMです。”id”はセンサーの固有の識別子です。MODEのHomeID内でユニークなidを指定してください。MODEのHomeIDについて学ぶ前であれば、ご自分の管理するセンサー内で重複しないようにすればよいとだけ覚えてください。
収集するデータは、温度データ1つですので、TEMPERATURE型で1つのスキーマを定義します。作られたJSONをcurlコマンドでPOSTします。

データを収集する

このMODEセンサーゲートウェイに使われているハードウェアであるNUC5CPYH上のLinuxでは、CPUの温度は次のファイルから取得することができます。

```
/sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input
```

中を開くと例えば 44000 とあります。これはCPU温度が44.000℃であることを表しています。次のようなsleep付きのループを追加し、5秒毎に変数にCPU温度を格納するようにしてみましょう。

```
while true
do
  sleep 5

  temp=$(cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input)
done
```

データをMODEセンサークラウドへLocal APIで送る

前のセクションで収集したCPUの温度データをデータ送付APIで送ってみましょう。データはJSON形式で定義して送信できます。スキーマとセンサー値を対にして送信します。”timestamp”フィールドは省略することもできますが、ここではタイムスタンプを付与して送信してみます。

先ほどのループを次のように改変してみましょう。太字部分です。また、いったんここまでのコードの全体も示しておきます。

```
#!/bin/bash

announce_json=$(cat <<JSON
{
  "model": "CUSTOM",
  "id": "MyNUC",
  "sensors": [
    "TEMPERATURE:0"
  ]
}
JSON
```

```
)

curl -X POST -d "$announce_json" \
  --header "Content-Type: application/json" \
  http://localhost:55299/announce

while true
do
  sleep 5

  temp=$(cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input)
  value=$(echo $temp | awk '{print $1/1000}')

  data_json=$(cat <<JSON
    [
      {
        "sensor": "TEMPERATURE:0",
        "value": $value
      }
    ]
JSON
)

  curl -X POST -d "$data_json" \
    --header "Content-Type: application/json" \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/sensorData
done
```

確認と仕上げ

うまくできていれば、MODEセンサークラウドのUIで送信されたデータを確認することができます。
シェルスクリプトを保存し、実行権限を付けます。

```
$ chmod +x cpu_temp.sh
```

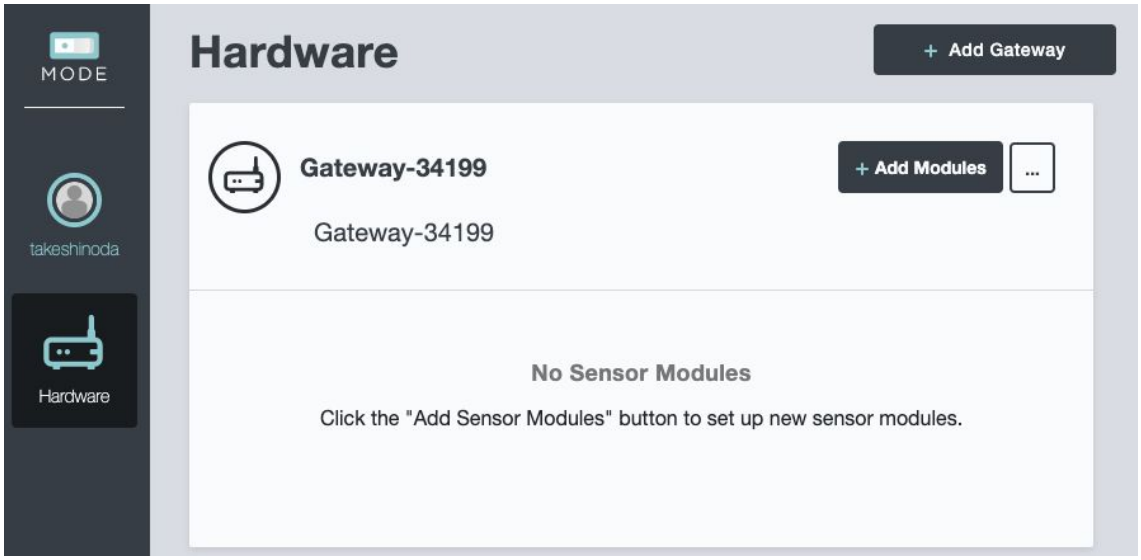
その後実行してみましょう。

```
$ ./cpu_temp.sh
```

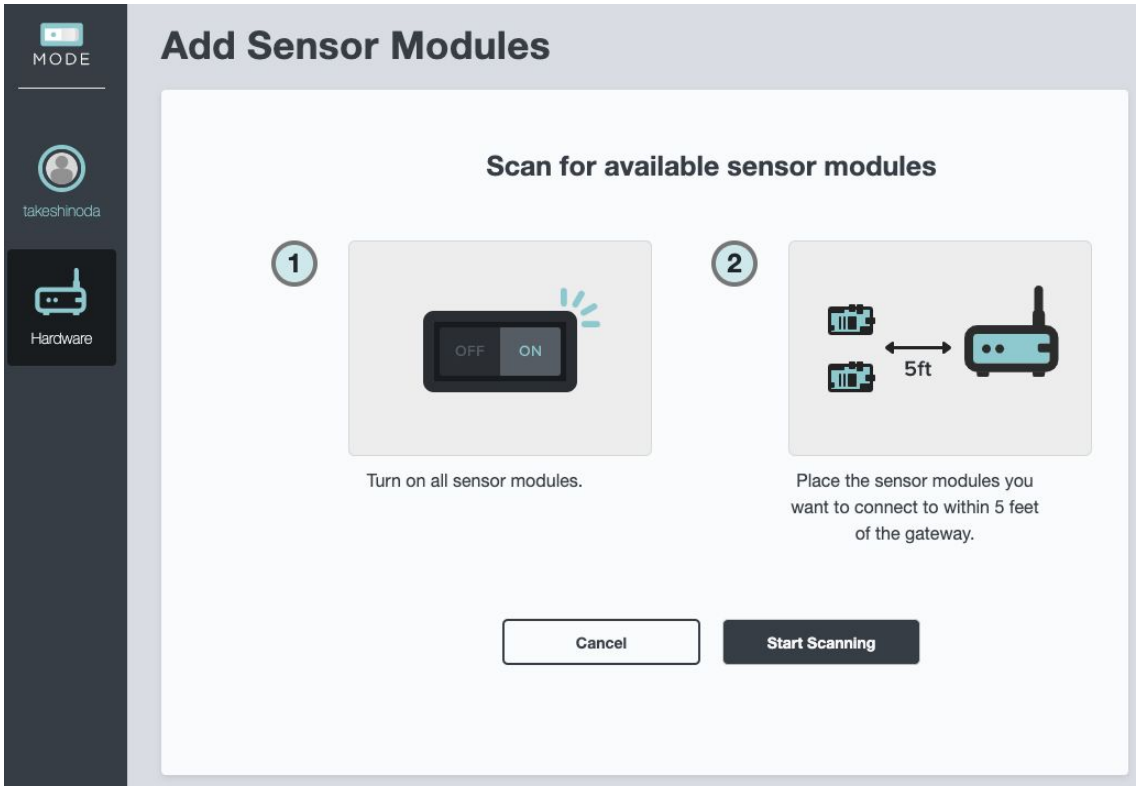
実行すると、アナウンスAPIがまず呼ばれ、データが5秒間隔で送信されます。アナウンスAPIにより、MODEセンサークラウドはMODEセンサーゲートウェイにセンサーが接続されていることを認識できます。

プログラムの確認のため、MODEセンサークラウドにアクセスし、センサーモジュールのスキャンをしてみましょう。

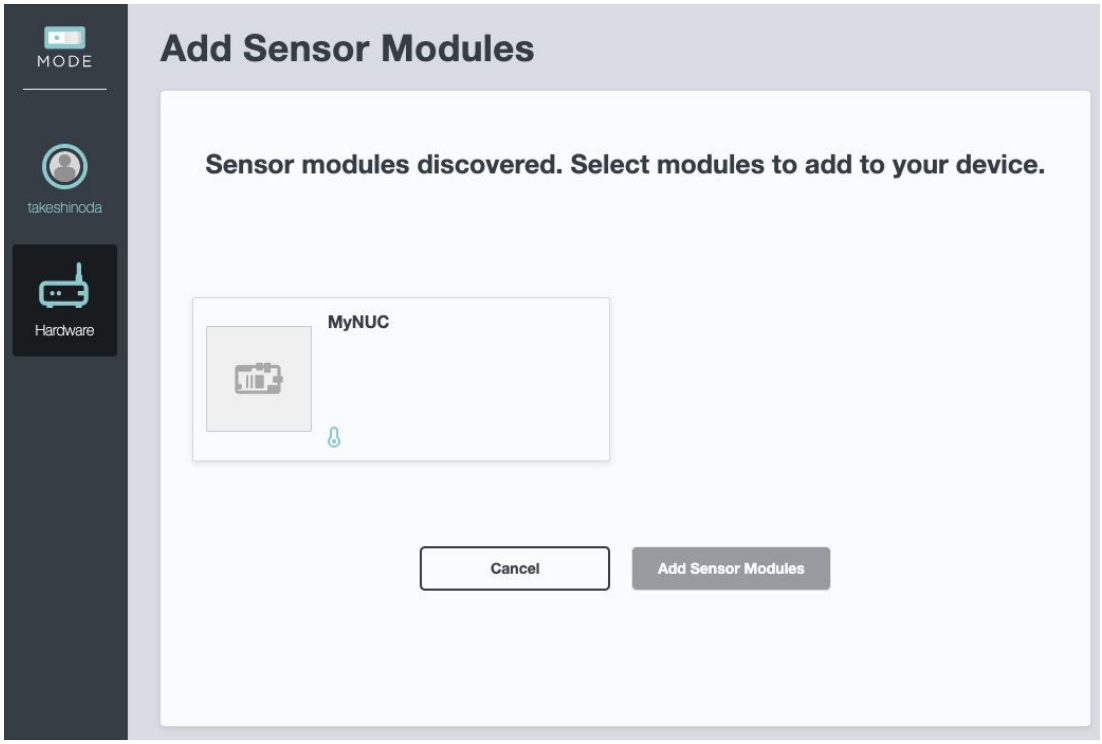
右上の [Add Modules] ボタンを押下します。



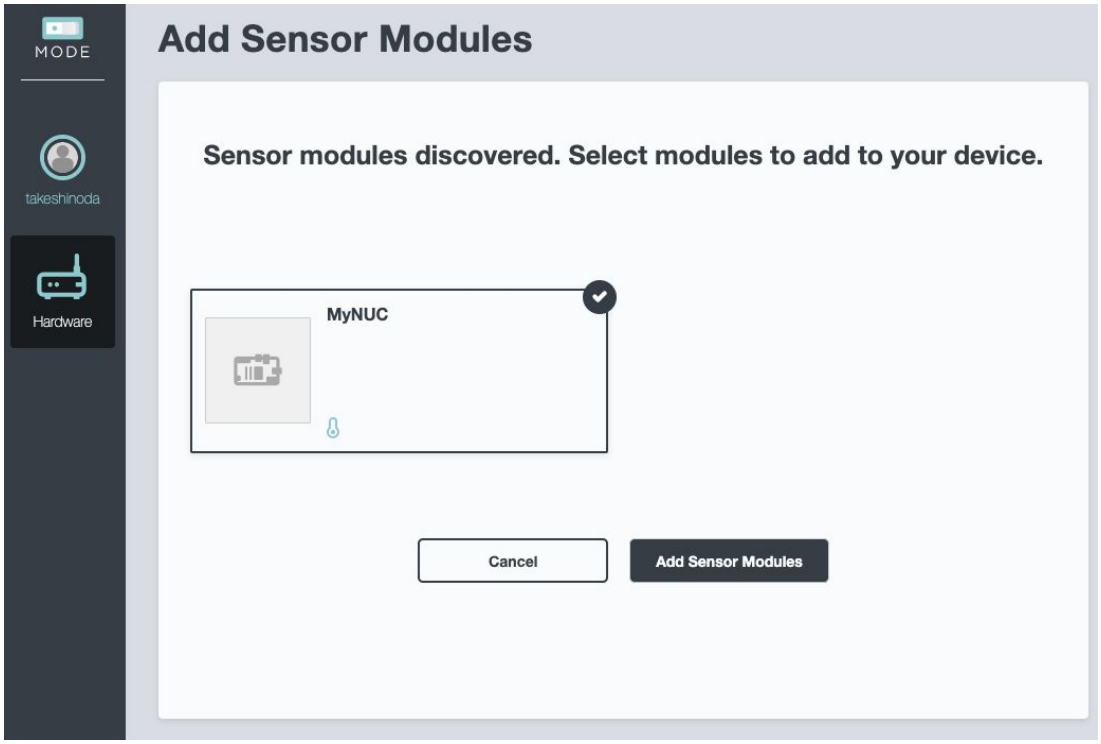
[Start Scanning] ボタンを押下し、センサーのスキャンをします。



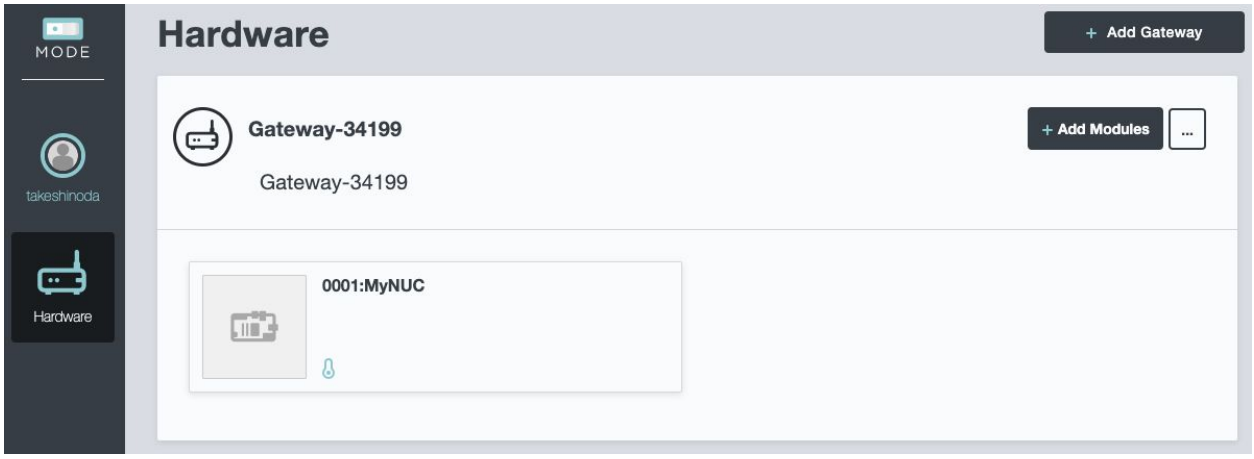
アナウンスAPIで宣言した、”MyNUC”のセンサーが表示されれば成功です。



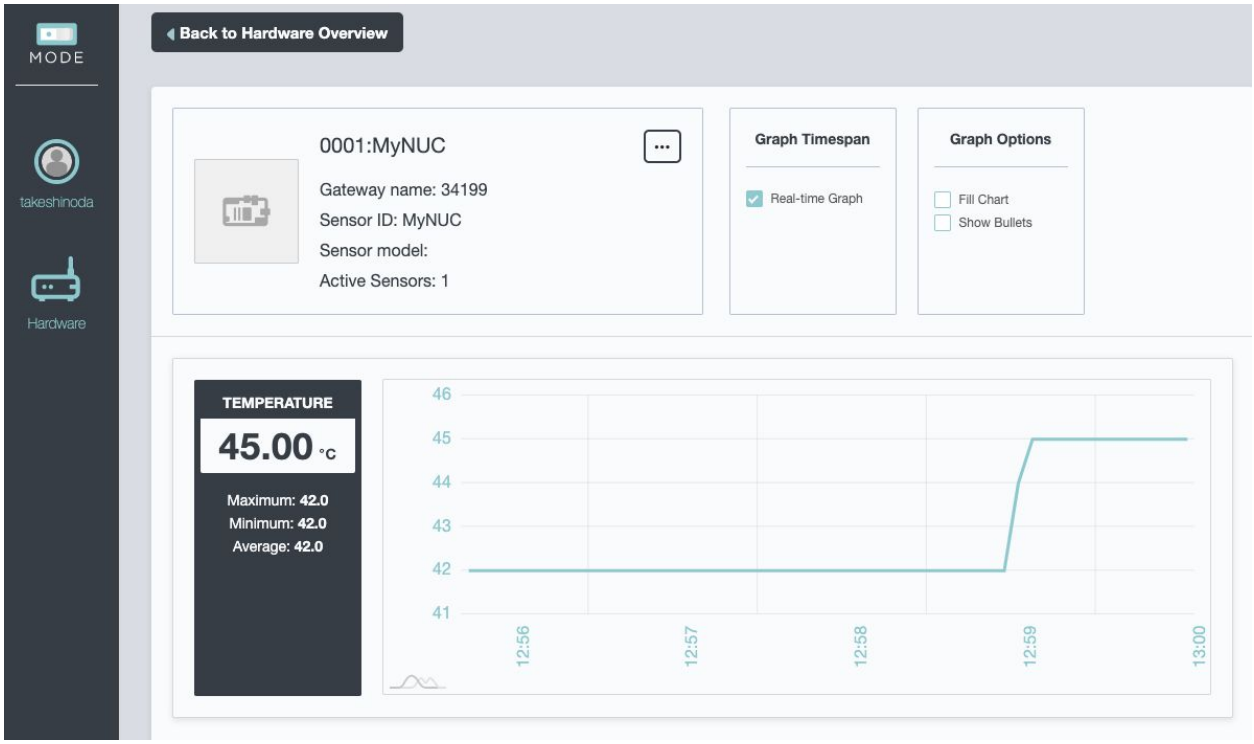
”MyNUC”を選択し、 [Add Sensor Modules] ボタンを押下しましょう。



次のように”0001:MyNUC”があれば、センサーデータの収集がはじまっています。



センサーを選択して、データを確認してみましょう。



CPU温度に変化が無くてつまらない場合は、別途下記のようなシェルスクリプトをいくつかバックグラウンドで実行してCPUを負荷をかけ、温度を上昇させててみましょう。テストのあとはプロセスを止め忘れないようにしてください。

```
#!/bin/bash
while true
do
  date +%s | sha256sum > /dev/null
done
```

コマンドの受信

MODE クラウドからコマンドを受け取れるようにしてみましょう。ここのチュートリアルではデータ送信の停止と再開をできるようにします。

先ほどのデータ収集のループにコマンドを受け取ることができるコマンド取得 APIを追記します。今までのコードのループに対して、下記太字部分を追記します。

```
while true
do
  sleep 5

  curl -s -X GET \
```

```
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/command

  temp=$(cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input)
  value=$(echo $temp | awk '{print $1/1000}')

  data_json=$(cat <<JSON
    [
      {
        "sensor": "TEMPERATURE:0",
        "value": $value
      }
    ]
JSON
)

  curl -X POST -d "$data_json" \
    --header "Content-Type: application/json" \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/sensorData
done
```

実行すると、数回次のようなJSONが出力されます。アナウンス時や接続時にセンサーの起動コマンドが自動的に送信されてきたものです。

```
$ ./cpu_temp.sh
{"action":"startSensors","interval":30,"sensors":["TEMPERATURE:0"]}
```

MODEセンサークラウドからセンサーのストップの指示が来た時はセンサーデータを送らないように、スタートの指示の時はデータを送るようにループを改修してみましょう。
修正と追記する箇所は下記の太字部分です。ループに入る直前に変数の初期化が追加されているところに注意してください。

```
sensing=true
while true
do
  sleep 5

  cmd=$(curl -s -X GET -w '%{http_code}' \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/command |\
  grep 200)

  echo $cmd | grep startSensor && sensing=true
  echo $cmd | grep stopSensor && sensing=false

  ! $sensing && continue
```

```
  temp=$(cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input)
  value=$(echo $temp | awk '{print $1/1000}')

  data_json=$(cat <<JSON
    [
      {
        "sensor": "TEMPERATURE:0",
        "value": $value
      }
    ]
JSON
)

  curl -X POST -d "$data_json" \
    --header "Content-Type: application/json" \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/sensorData
done
```

収集の設定

最後にMODEセンサーゲートウェイの電源が切れ、再起動された場合にもデータ収集が自動的に開始するような設定をしてみましょう。cronにて先ほど書いたスクリプトを起動するよう実行コマンドを追記します。crontab -e コマンドを使ってエディタを起動して設定ファイルを開いてください。

```
$ crontab -e
```

下記を最下行に追記し、保存します。

```
@reboot nohup /home/ubuntu/cpu_temp.sh > /dev/null &
```

再起動し、データがMODEセンサークラウドで確認できていれば成功です。再起動は、MODEセンサーゲートウェイの電源ボタンを押して電源を切り(電源ボタンの青いLEDが消えます)、もう一度押すことでできます。もしくは、下記コマンドで再起動できます。

```
$ sudo shutdown -r now
```

チュートリアルのコード全体

完成したコード全体をここに示します。

```
#!/bin/bash

announce_json=$(cat <<JSON
{
  "model": "CUSTOM",
```

```
  "id": "MyNUC",
  "sensors": [
    "TEMPERATURE:0"
  ]
}
JSON
)

curl -X POST -d "$announce_json" \
  --header "Content-Type: application/json" \
  http://localhost:55299/announce

sensing=true
while true
do
  sleep 5

  cmd=$(curl -s -X GET -w '%{http_code}' \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/command |\
  grep 200)

  echo $cmd | grep startSensor && sensing=true
  echo $cmd | grep stopSensor && sensing=false

  ! $sensing && continue

  temp=$(cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp2_input)
  value=$(echo $temp | awk '{print $1/1000}')

  data_json=$(cat <<JSON
    [
      {
        "sensor": "TEMPERATURE:0",
        "value": $value
      }
    ]
JSON
)

  curl -X POST -d "$data_json" \
    --header "Content-Type: application/json" \
    http://localhost:55299/sensorModules/CUSTOM:MyNUC/sensorData
done
```

MODEセンサーゲートウェイのソフトウェア構成

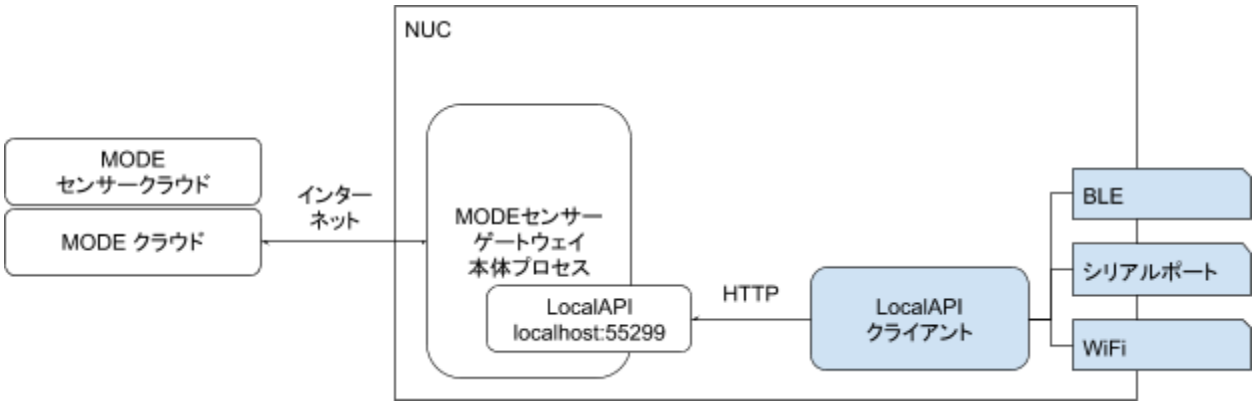
OS

OSはUbuntu 18.04 LTSを採用しています。ローカルのコンソールからMODE社より伝えられるアカウントとパスワードによりログインできます。

MODEセンサーゲートウェイのソフトウェア構成

MODEセンサーゲートウェイ内では、MODEセンサーゲートウェイの本体プロセスがsystemdにより起動され、稼働しています。実行ファイルは `/home/mode/gateway/sensorgw` です。
Local APIは、MODEセンサーゲートウェイの本体プロセス上に実装されており、ポート55299のHTTPサーバーとして動作します。

ある特定のセンサーのデータを収集し、Local APIを利用してMODEセンサークラウドに送信するプログラムが Local APIクライアントです。
下図は構成の概要図です。着色部分が開発対象にあたり、新しいセンサをサポートするために開発が必要になる部分です。



ディレクトリ構造

MODEセンサーゲートウェイのディレクトリ構造は次のような構造となっています。

/home/mode

MODEが提供し、MODEセンサーゲートウェイを構成する本体および各種コンポーネントが配置されています。OTAUクライアントによりソフトウェアアップデートがおこなわれた場合、設定ファイルを除き、このディレクトリのファイルが書き換えられる可能性があります。そのため、永続化が必要なファイルはこのディレクトリに配置しないでください。

/var/log

MODEセンサーゲートウェイの各種コンポーネントからのログは、syslog経由でこのディレクトリに収集されます。

/var/gd

センサーデータの到達保証機能のためのバッファ用ディレクトリです。

Local APIデータ形式

センサーの識別子

モデル

センサーモジュールの型番を識別するための文字列です。モデル名には、半角英数・アンダースコア(_)・ハイフン(-)が使用でき、大文字小文字は区別しません。Local APIクライアントが利用できるモデル名は常に"CUSTOM"です。

例: “CUSTOM”

センサーモジュール識別ID

センサーの個体を示すIDです。MODEプラットフォーム上の"ホーム"で使用されるセンサーの特定のモデル内のひとつひとつの個体に対して、ユニークな文字列を割り当てる必要があります。センサーモジュール識別IDには、半角英数・アンダースコア(_)・ハイフン(-)が使えます、大文字小文字は区別しません。

例: “custom-sensor-1”

センサーモジュールID

モデル名とセンサーモジュール識別IDをコロン(:)でつないだものです。MODEセンサークラウドは、センサーモジュールIDをセンサーモジュールの識別に用います。

例: “CUSTOM:custom-sensor-1”

センサースキーマ

各モデルが持っているセンサーのデータ形式とその番号です。同じ形式が複数ある場合、ゼロからはじまる連番です。
形式と番号はコロン(:)でつながれます。半角英数・アンダースコア(_)・ハイフン(-)で構成されます。大文字小文字は区別しません。

例えば、あるセンサーが2箇所の温度と1つのCO2濃度を測ることができるならば、次のような3つのセンサースキーマを持ちます。

- “TEMPERATURE:0”
- “TEMPERATURE:1”
- “CO2:0”

センサーデータ

センサーデータは数値であり、64bit 浮動小数点数のデータとして取り扱われます

タイムスタンプ

センサーの値が測定された時刻です。これは省略可能です。採取タイムスタンプを省略した場合は、MODEセンサーゲートウェイの本体プロセスがデータを受信した時刻をタイムスタンプとして自動的に採用します。

例: "2019-07-23T12:23:45.000Z"

API リファレンス

アナウンスAPI

概要

MODEセンサーゲートウェイに対して、Local APIクライアントが利用するセンサーのスキーマを送付します。センサーデータを送信する前に最低1回は実行する必要があります。

インターフェース

エンドポイント

`http://localhost:55299/announce`

メソッド

`POST`

コンテントタイプ

`application/json`

パラメータ

```
{
  "model": "CUSTOM",
  "id": <string>,
  "sensors": [<string>, ...]
}
```

Attribute 名	必須	説明
model	✓	モデル名です。Local APIクライアントはCUSTOM固定です。 CUSTOM以外は受け付けません。
id	✓	センサーモジュール識別IDです。
sensors	✓	センサーを示すスキーマの配列です。順番に意味はありません。

レスポンス

成功の場合はHTTPステータスコード204(Not content)が返されます。Body部はありません。

データ送付API

概要

センサーデータをMODEセンサーゲートウェイの本体プロセスに対して送付します。送付されたデータは本体プロセスによってMODEセンサークラウドに送信されます。
Local APIクライアントは配列によって、一度に複数のセンサーのデータを送付することができます。

インターフェース

エンドポイント

`http://localhost:55299/sensorModules/<センサーモジュールID>/sensorData`

メソッド

`POST`

コンテントタイプ

`application/json`

パラメータ

センサーデータの配列で送付します。

```
[{
  "sensor": <string>,
  "timestamp": <string>,
  "value": <number>
},...]
```

Attribute 名	必須	説明
sensor	✓	データのセンサーのスキーマ名です。
timestamp		センサーの値を採取した時刻のRFC3339形式です。省略した場合は、ゲートウェイがデータを受信した時刻をタイムスタンプとします。
value	✓	センサーのデータです。64bit の浮動小数点数の値。

レスポンス

成功の場合はHTTPステータスコード204(Not content)が返されます。Body部はありません。

コマンド取得API

概要

MODEセンサークラウドおよびMODEセンサーゲートウェイから送信されたセンサーに対するコマンドを、コマンド取得APIによりLocal APIクライアントが受信できます。1-10秒に1回程度の頻度でコマンド取得APIをポーリングすることで、MODEセンサークラウドから送信されるコマンドをほぼリアルタイムで受信できます。
なお、クライアントはこのコマンドに必ずしも従う必要はありません。例えば、stopSensorsの実装を省いた場合、MODEセンサークラウドからセンサーの停止を設定してもデータが送信され続けますが、MODEセンサークラウドに技術的な問題が発生することはありません。

インターフェース

エンドポイント

`http://localhost:55299/sensorModules/<センサーモジュールID>/command`

メソッド

`GET`

コンテントタイプ

`application/json`

パラメータ

無し

レスポンス

MODEセンサークラウドから送信されたコマンドが存在しない場合は、HTTPステータスコード204(No content)が返されます。
MODEセンサークラウドから送信されたコマンドが存在する場合はHTTPステータスコード200が返されます。あわせて、Body部に次のようなJSONが返されます。

```
{
  "action": <string>,
  "interval": <number>,
  "sensors": [<string>,...]
}
```

Attribute 名	必須	説明
action	✓	“startSensors”, “stopSensors” のいずれかです。 stopSensorsはセンサーを止めることが期待されます。 startSensorsはセンサーを稼働することが期待されます。
interval	✓	センサー値の採取の間隔が指定されます。単位は秒です。
sensors	✓	action対象のセンサータイプの配列。

接続状態の管理

MODEセンサークラウドおよびMODEセンサーゲートウェイはセンサーの接続状態を常に管理しています。

MODEセンサーゲートウェイとセンサーの紐付け

MODEセンサークラウド上でセンサーとMODEセンサーゲートウェイを紐付けるには、一度以上アナウンスAPIがMODEセンサーゲートウェイ内で正常に呼び出される必要があります。紐付けられたセンサーはMODEセンサークラウドのWeb画面により、TSDBに格納されたデータを参照することができます。

センサーのオフライン(タイムアウト)

コマンドAPIもしくはデータ送付APIが15秒以上呼び出され無い場合、タイムアウトとなり、センサーはオフライン状態となります。

オフライン(タイムアウト)後のデータ送信再開

データ送付APIもしくはコマンド取得APIを呼び出すことで、センサーはオンライン状態に戻ります。

開発版の利用のためのTips

ネットワーク

SSHサーバを設定すると、ご自身のPCからターミナルを通じて開発できます。aptにて ssh-serverをインストールして、enp3s0(有線NIC)を固定IPアドレスとして設定できます。

有線LANを有効化する

/etc/network/interfaces ファイルの下記部分を修正し、再起動することで、反映されます。

```
auto enp3s0
iface enp3s0 inet static
    address 192.168.100.30
    mask 255.255.255.0
```

履歴

2019-06-17: 初期版
2019-09-19: 公開