

スマートメータのデータをAzure IoT に送る



概要

本資料はスマートメータから取得したデータをAzure IoT Centralで収集しblobストレージに蓄積するまでの手順を記したものです

参考資料

Azure IoT Central クイックスタート

<https://docs.microsoft.com/ja-jp/azure/iot-develop/quickstart-send-telemetry-central?pivot=programming-language-ansi-c>

Azure IoT Central デバイステンプレート

<https://docs.microsoft.com/ja-jp/azure/iot-central/core/howto-edit-device-template>

掲載内容

本資料は以下の内容に従いAzure IoT Centralとスマートメータからのデータ取得について掲載しています

1. ハードウェアのセットアップ
 - 1-1. Raspberry PiとWiSUNデバイスの接続
2. Raspberry Piのセットアップ
 - 2-1. 書き込みツールのインストール
 - 2-2. SDカードへの書き込み
 - 2-3. Raspberry Piの起動と初期設定
 - 2-4. ネットワーク経由の接続
 - 2-5. タイムゾーンの設定
 - 2-6. OSアップグレード
 - 2-7. 必須パッケージのインストール
3. スマートメータ
 - 3-1. シリアルポートの有効化
 - 3-2. ライブラリのインストール
 - 3-3. サンプルコードの実行
4. Azure IoT central
 - 4-1. Azure IoTデバイスの作成
 - 4-2. Azure IoT SDKのインストール
 - 4-3. Azure IoT centralとの接続確認
5. スマートメータアプリの実行とIoT デバイス登録
 - 5-1. スマートメータアプリの実行
 - 5-2. Azure IoT Central上でデバイス登録
 - 5-3. データの可視化
6. Blobストレージへのエクスポート
 - 6-1. Blobストレージの作成
 - 6-2. エクスポート設定
 - 6-3. Blobストレージのエクスポート

必要なもの

本資料で掲載している内容を実施するためには以下の機材を用意しておく必要があります

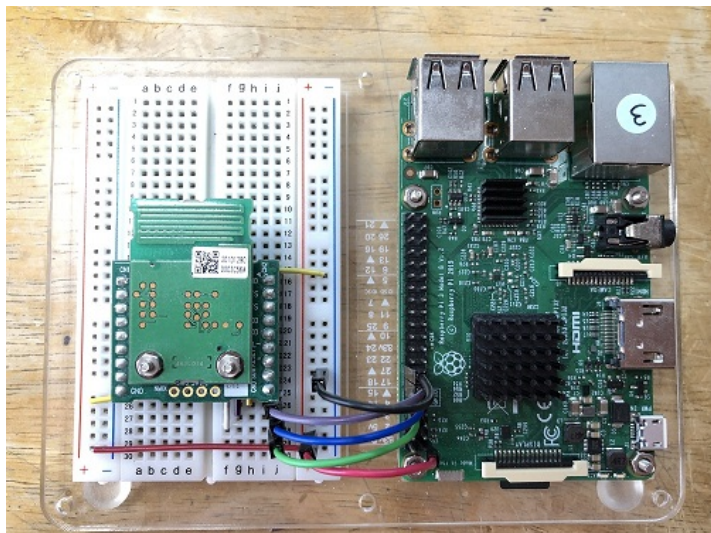
- ・スマートメータ接続認証ID
電力メーター情報発信サービス（Bルートサービス）利用申込を行ったあとに郵送される書類に入っています
申し込みは各電力会社のWebページにて行ってください
東京電力: <https://www.tepco.co.jp/pg/consignment/liberalization/smartmeter-broute.html>
- ・WiSUN通信デバイス ロームデバイスの代理店もしくはオンラインショップで購入可能です
BP35A3(+変換基板)
BP35C0(+専用アンテナ)

- ・ Raspberry Pi本体 (3B,4,ZeroW)
- ・ Raspberry Pi用 ACアダプタ
- ・ microSDカード
- ・ SDカードリーダー
- ・ HDMIモニタ(+ケーブル)
- ・ USBキーボード
- ・ PC (Windowsもしくはそれ以外)
- ・ 有線LANケーブルおよび有線ルータ
- ・ ブレッドボードもしくはWiSUN組み立てキット

<https://www.switch-science.com/products/6467>

1.ハードウェアのセットアップ

スマートメータからデータを取り出すためには次のようなハードウェアを準備します



1-1. Raspberry PiとWiSUNデバイスの接続

スマートメータと接続するためにはWiSUN通信デバイス(BP35A3もしくはBP35C0)が必要となります
これらはオンラインショップで購入可能です

(参考)キットの購入

ブレッドボードで組み立てる以外にも以下の場所で組み立てキットが販売されており購入することができます
なおキットにはWiSUN通信デバイスは付属していないため別途購入してください

<https://www.switch-science.com/products/6467>

信号線の結線

Raspberry PiとWiSUNデバイスを接続するためには次のような結線が必要です

Raspberry Pi WiSUNデバイス

3.3V ----- VCC

GND -----+----- GND

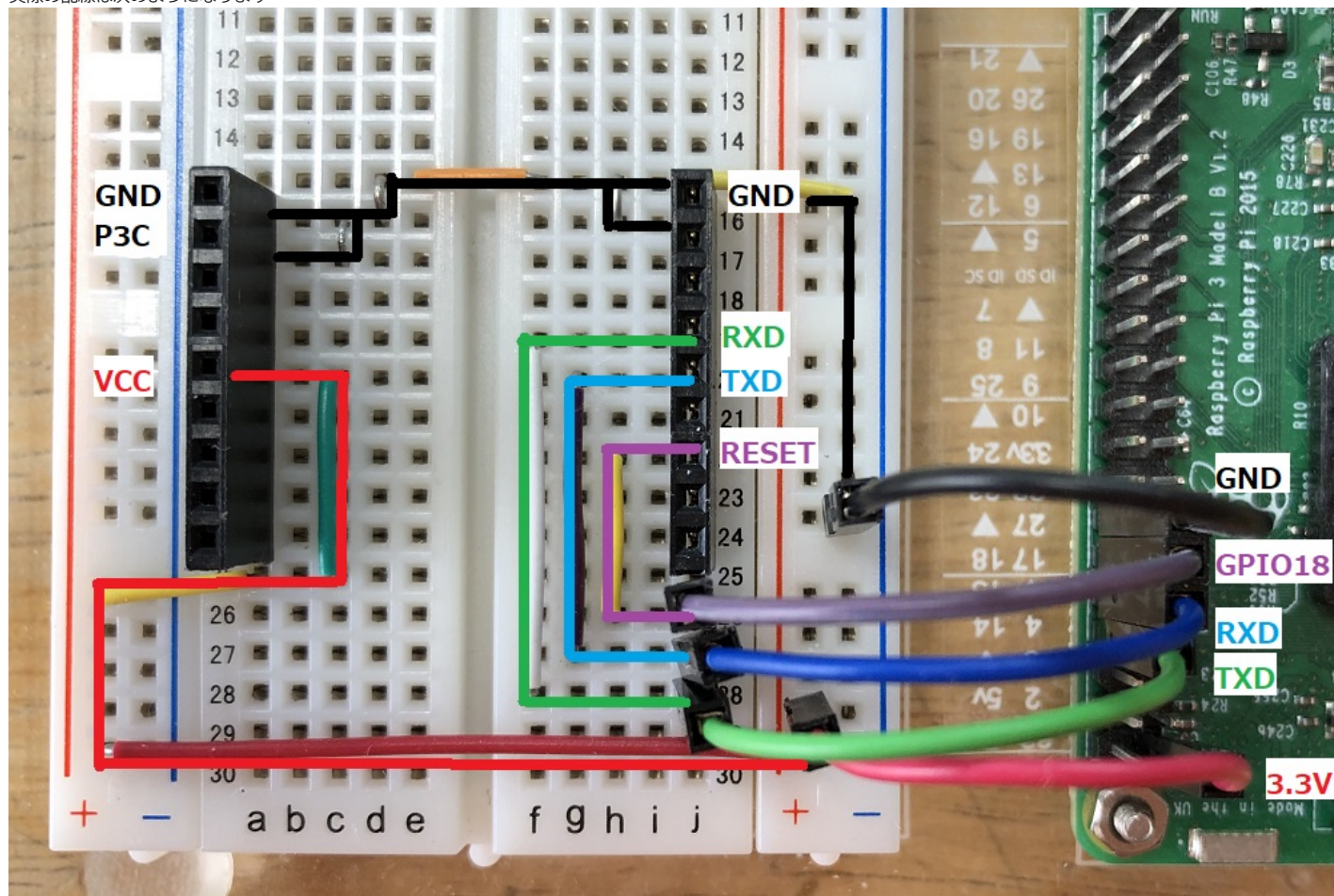
+----- P3C

TXD ----- RXD

RXD ----- TXD

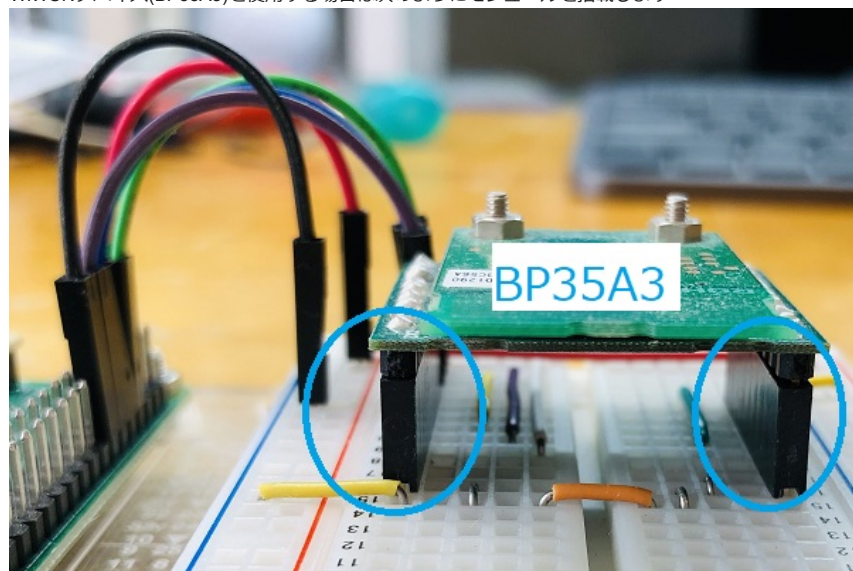
GPIO18 ----- RESET

実際の配線は次のようになります



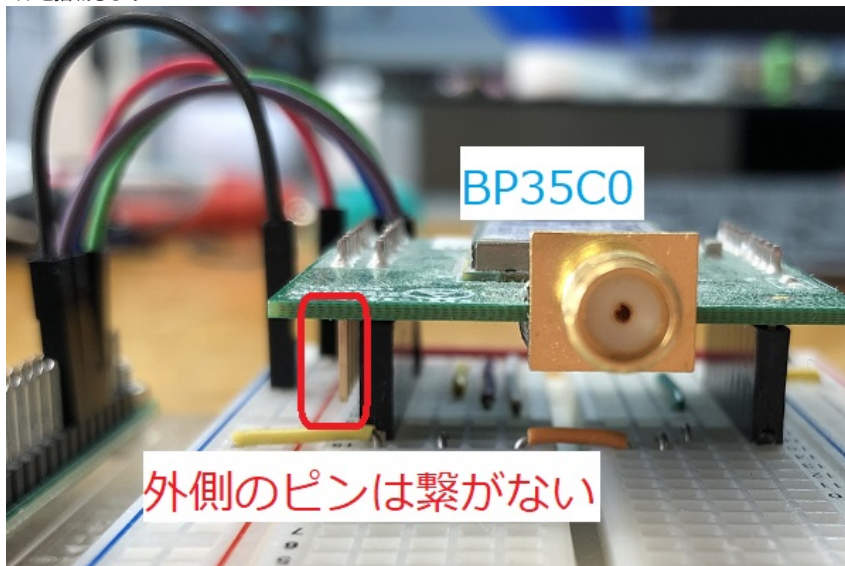
BP35A3を使用

WiWUNデバイス(BP35A3)を使用する場合は次のようにモジュールを搭載します



BP35C0を使用

WiWUNデバイス(BP35C0)を使用する場合は次のようにモジュールを搭載します



モジュールの端子のうち内側のみ接続するようにしてください

2.Raspberry Piのセットアップ

Raspberry Piを使うためにはボードにOSを入れる必要があります
本資料では 2022年 1 1月時点で最新のRaspberrypi OSをインストールします
OSは以下の場所からダウンロードしてください

<https://www.raspberrypi.com/software/operating-systems/>

今回はデスクトップ環境は不要であるためLite版(Raspberry Pi OS Lite)をダウンロードし使用します
ダウンロードのリンクを開くと次のファイルがダウンロードされます

2022-09-22-raspbios-bullseye-armhf-lite.img.xz

また同サイトからOSイメージを書き込むツール(Raspberry Pi Imager)もダウンロードしておいてください

<https://www.raspberrypi.com/software/>

2-1.書き込みツールのインストール

PCに前項でダウンロードしたRaspberry Pi Imagerをインストールを行います
winddows版の場合.exeフォーマットのファイルとなっているためダブルクリックしてインストールを開始します
インストールが成功すると Raspberry Pi Imagerという名称のアイコンが追加されます

2-2.SDカードへの書き込み

microSDカードに前項で取得したファイルを書き込みを行います
書き込みツールであるRaspberry Pi Imagerを起動
起動すると次のような画面が表示されます



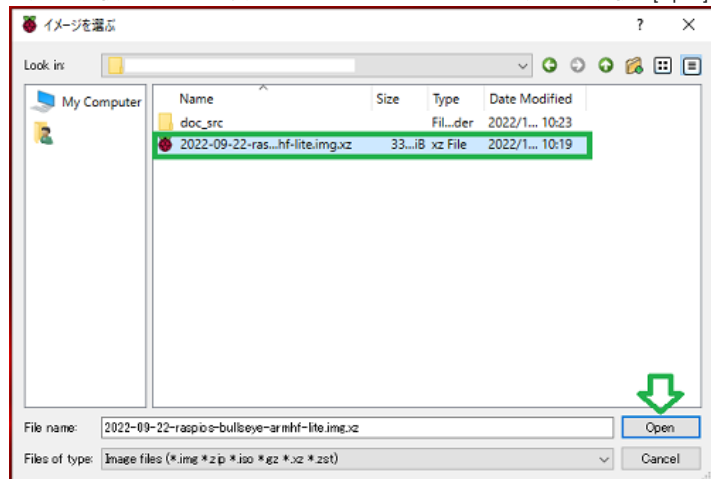
まずインストールするOSを選択します



今回は前項でダウンロードしたOSイメージを利用することから「カスタムイメージを使う」を選択



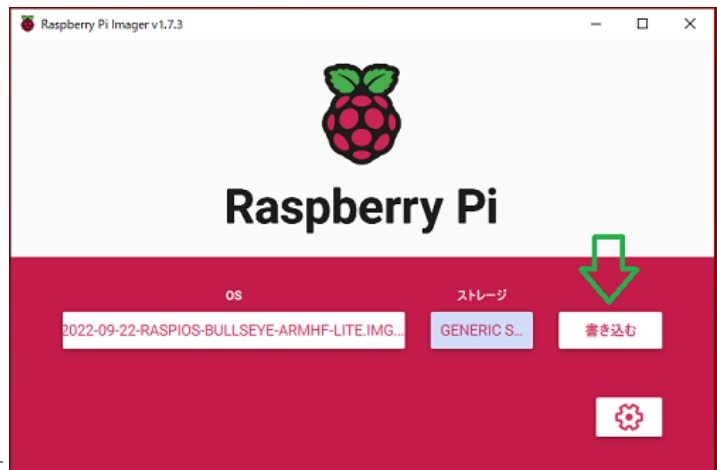
ファイル選択ダイアログが表示されるのでダウンロードしたファイルを選択し[Open]を押下します



つぎにUSB-SDカードリーダーにmicroSDカード(16GB以上)を装着しPCに接続し「ストレージを選ぶ」を押下



対象デバイスのリストが表示されるのでmicroSDカードの近似サイズのUSBストレージを選択



全ての選択が終わったら「書き込む」ボタンを押下し、「はい」を押下します



microSDへの書き込みが開始されるので終了まで待つ
書き込みが完了したらUSBカードリーダーを外し、microSDを抜きます



2-3. Raspberry Piの起動と初期設定

前項で書き込んだ microSDカードを Raspberry Piに装着し起動させます

この際 HDMIコネクタにはディスプレイ、USBコネクタにはキーボードを接続しておいてください またネットワークは有線ケーブル接続とします

※ ルータ設定etcでリモート接続できない場合が多いことから有線接続を推奨しています

今回作成するシステムではディスプレイを使用しないヘッドレス運用になりますのでネットワーク経由でリモートログインできるようにします

```
pi@raspberrypi:~$ sudo raspi-config
3 Interface Option
-> I2 SSH
--> <Yes>
<finish>
```

リモートログイン設定が終わったら Raspberry PiのIPアドレスを調べておきます
次のようなコマンドを入力する inet行に IPアドレスが表示されます
ここで表示されたIPアドレスは後で使用するためメモしておいてください(メモ①)

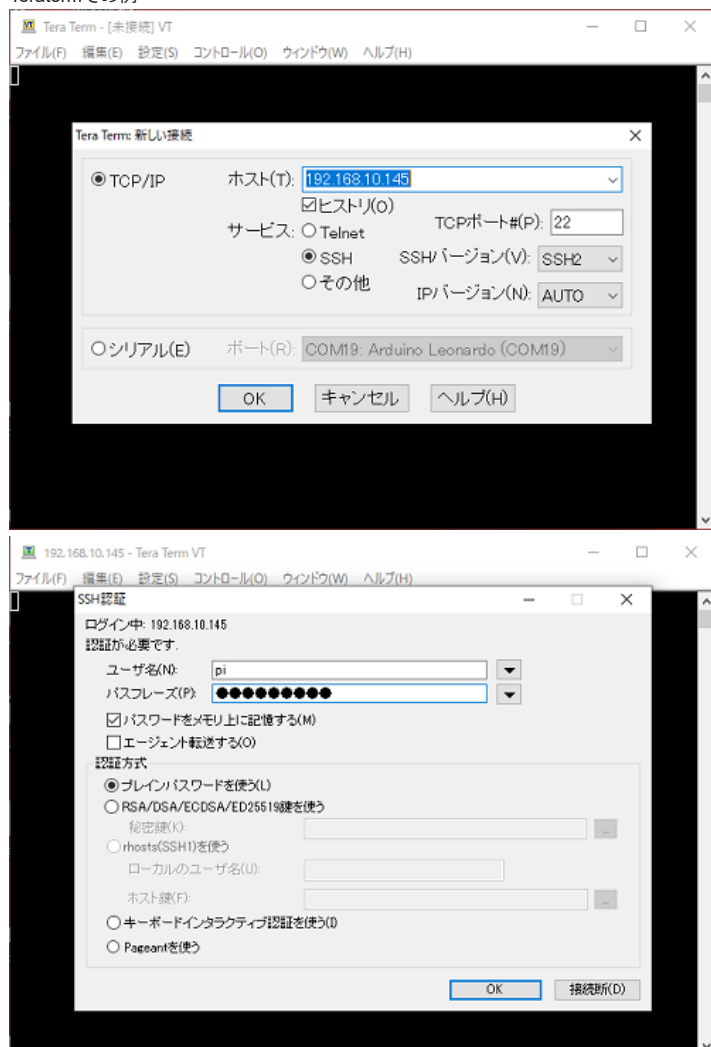
```
pi@raspberrypi:~$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.10.145  netmask 255.255.255.0  broadcast 192.168.10.255
    ...
```

2-4.ネットワーク経由の接続

以降はディスプレイとキーボードによる操作は難しくなるので SSHによるリモート接続で操作を行います
TeratermなどのSSHクライアントを用い先ほどメモ①したIPアドレスで接続します

ユーザ名 : pi
パスワード : raspberry

Teratermでの例



SSH接続が成功すると次のような画面となる

```
192.168.10.145 - pi@raspberrypi ~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Linux raspberrypi 5.15.61-v7+ #1579 SMP Fri Aug 26 11:10:59 BST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Nov 13 23:47:10 2022

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$
```

Teratermでは次のような操作でコマンドラインのコピー/ペースト、ファイル送信ができます

[ALT]+[C] 選択領域のコピー

[ALT]+[V] コマンドラインへのペースト

画面内へのファイルドラッグ ファイル送信

2-5. タイムゾーンの設定

デフォルトではGMT(グリニッジ標準時間)となっているため日本標準時にセットします

```
$ sudo timedatectl set-timezone Asia/Tokyo
```

2-6. OSアップグレード

ダウンロードしたOSが最新とは限らないためオンラインで更新を行います

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

2-7. 必須パッケージのインストール

AzureSDKとpython動作に必要なパッケージをインストールします

```
$ sudo apt-get install -y git cmake build-essential curl libcurl4-openssl-dev libssl-dev uuid-dev
$ sudo apt-get install python3-pip
$ pip3 install pyserial
```

インストールされたパッケージのバージョンを確認する

cmake のバージョンが 2.8.12 より大きく、GCC のバージョンが 4.4.7 より大きいことを確認

```
$ cmake --version
cmake version 3.18.4

CMake suite maintained and supported by Kitware (kitware.com/cmake).

$ gcc --version
gcc (Raspbian 10.2.1-6+rpil) 10.2.1 20210110
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3. スマートメータ

この章ではスマートメータ接続に必要な機能をセットアップします

3-1. シリアルポートの有効化

WiSUN通信デバイスを使うためには Raspberry Piのシリアルポートが必要となります デフォルトでは無効化されていることから次の操作を行いシリアルポートを有効化させます

```
$ sudo raspi-config

Raspberry Pi Software Configuration Tool (raspi-config)
3 Interface Options
<Select>
I6 Serial Port
<Select>
Would you like a login shell to be accessible over serial?
<No>
```

```
Would you like the serial port hardware to be enabled?
<Yes>
The serial login shell is disabled
The serial interface is enabled
<Ok>
<Finish>
Would you like to reboot now?
<Yes>
```

設定を有効化するためRaspberry Piが再起動されます
再起動が終わったら再度SSHでログインしてください

3-2. スマートメータライブラリのインストール

日本向けスマートメータのライブラリとしてgithub上で公開されているものを利用します <https://github.com/kazunyanya/pi-smartmeter>

ライブラリは次のコマンドでインストールします

```
$ cd ~/
$ git clone https://github.com/kazunyanya/pi-smartmeter.git
$ cd pi-smartmeter
$ pip3 install .
```

3-3. サンプルコードの実行

github配布ライブラリのフォルダにはサンプルコード sample.py が入っています
実行前に以下の部分をviエディタ等で3か所編集を行います

使用しているWiSUNデバイスの行にのみコメントアウトを外します

```
動作設定 - 使用するデバイスのみコメントアウトを外す
DEVICE = "BP35A3"
# DEVICE = "BP35C0"
```

スマートメータ接続認証IDとパスワードを設定します
この情報は電力会社から郵送される書類に入っているものを設定してください

```
# ブルート認証ID
ID_WISUN = "00000000000000000000000000000000"
# ブルート認証パスワード
PWD_WISUN = "SSSSSSSSSSSS"
```

サンプルコードの修正が終わったらテストのため実行してみます
数分ごとにスマートメータの値が表示されていれば正常に動作しています

```
$ cd ~/pi-smartmeter
$ python3 sample.py

##### main() start
connecting SmartMeter
<INFO>init bp35c0x
<INFO>BP35A3 version 1.2.10
<INFO>finding echonet device
<INFO>echonet device found
<INFO>connecting echonet device
<INFO>echonet device connected
SmartMeter connected
request for SmartMeter
dousa 0x30

sekisan tanni 0.1[kWh]

sekisan 3871.2000000000003[kWh]

denryoku 226.0[W]

R=0.0[A] T=3.0[A]

##### main() sleep
{以降繰り返し}
```

4. Azure IoT central

本章ではAzure IoT centralを利用するための登録を行います

4-1. Azure IoTデバイスの作成

Azure IoT centralの画面にてデバイス作成を行います
画面上部にある[+作成]ボタンを押下します



デバイス作成画面となる

デバイス名は任意の文字列を入力

今回は新規デバイスであるため「デバイステンプレート」項目は未割り当てを行います

入力が終わったら[作成]ボタンを押下します

新しいデバイスの作成

新しいデバイスを作成するには、デバイステンプレート、名前、一意の ID を選択してください。 [詳細情報](#)

デバイス名 * ①

raspi_iot_1111

デバイス ID * ①

ac7hs9q2re

組織 * ①

iotapp-20220514

デバイステンプレート *

未割り当て

このデバイスをシミュレートしますか?

シミュレートされたデバイスでは、実際のデバイスに接続する前にアプリケーションの動作をテストできるテレメトリが生成されます。

☐ いいえ

Azure IoT Edge デバイスですか?

Azure IoT Edge では、クラウド分析とカスタム ビジネス ロジックをクラウドからデバイスに移動します。

☐ いいえ

作成

キャンセル

作成が成功するとリスト上に作成したデバイス名が表示されるようになる

次に作成したデバイス名のリンクを開く



デバイス情報のページが開きます

まだデバイスとの接続が行われていないのでデータは空欄のままである

ここで画面上部の[接続]ボタンを押下します



デバイス接続情報が表示されるのでメモしておきます

例：

ID スコープ: 0ne005EC119
 デバイス ID: ac7hs9q2re
 認証の種類: SAS
 主キー: bnRdgAa+3FsFqmJCoNXZN9yfjwqIlo/eUNMaWOLg8Og=
 セカンダリ キー: cvo2+upGUMvtYXh41wMIDcWFUSRSm7j2NJKDjXmWJWI=

デバイス接続のグループ

×

ID スコープ ①

011E0000000000000000000000000000

デバイス ID ①

011E0000000000000000000000000000

このデバイスの接続の種類を選択します。これは、必要に応じて後で変更できます。

認証の種類

Shared Access Signature (SAS) ▾

キー QR コード

Shared Access Signature (SAS) は、セキュリティ トークンとキーを使用して IoT Central に接続します。下に表示されている既定の登録グループの SAS キーを使用して、デバイスを登録してください。[詳細情報](#)

主キー ①

011E0000000000000000000000000000

セカンダリ キー ①

011E0000000000000000000000000000

閉じる

4-2. Azure IoT SDKのインストール

Azure IoT Centralの登録が終わったら Raspberry Piに SDKをインストールします
今回はpython3を使用しているため python版SDKをダウンロードします

```
$ git clone https://github.com/Azure/azure-iot-sdk-python
```

ダウンロードしたSDKをインストールする

```
$ pip3 install azure-iot-device
```

4-3. Azure IoT centralとの接続確認

AzureクラウドとRaspberry Piの設定が正しいか確認するため簡単なテストアプリを動作します
今回は公開したリポジトリからazure-cent-sample.pyのサンプルコードを実行します

実行する前にソースコードを編集し認証データをセットします

```
以下の{}部分を書き換えます
# Azure
id_scope = "{ID スコープ}"
registration_id = "{デバイス ID}"
symmetric_key = "{主キー}"
```

サンプルコードの実行

サンプルコードを実行を行うと次のようなログが出力されます

```
$ python3 azure-cent01.py
##### main() start
<INFO>Creating client for connecting using MQTT over TCP
<INFO>Registering with Provisioning Service...
<INFO>Enabling reception of response from Device Provisioning Service...
<INFO>Connect using port 8883 (TCP)
<INFO>connected with result code: 0
<INFO>_on_mqtt_connected called
<INFO>subscribing to $dps/registrations/res/# with qos 1
<INFO>suback received for 1
<INFO>Successfully subscribed to Device Provisioning Service to receive responses
<INFO>publishing on $dps/registrations/PUT/iotdps-register/?$rid=5acc20d5-8f00-42b0-b743-e7fb041d675d
<INFO>payload published for 2
<INFO>message received on $dps/registrations/res/202/?$rid=5acc20d5-8f00-42b0-b743-e7fb041d675d&retry-after=3
<INFO>RegistrationStage(RequestAndResponseOperation): polling
<INFO>publishing on $dps/registrations/GET/iotdps-get-operationstatus/?$rid=07147be3-7917-4455-af80-a87bd0ff53c2&operationId=5.0111a09938d21874.7b0682c2-2bd5-4858-9d9a-a75acd062712
<INFO>payload published for 3
<INFO>message received on $dps/registrations/res/200/?$rid=07147be3-7917-4455-af80-a87bd0ff53c2
<INFO>Successfully registered with Provisioning Service
<INFO>Forcing paho disconnect to prevent it from automatically reconnecting
```

```
@@@ regist:ac7hs9q2re
iotc-7e352a34-21c1-46f2-81c9-2e6ebf46ee59.azure-devices.net
initialAssignment
null
assigned
Will send telemetry from the provisioned device
<INFO>Creating client for connecting using MQTT over TCP
@@@@ connect IoT central
<INFO>Connecting to Hub...
<INFO>Connect using port 8883 (TCP)
<INFO>connected with result code: 0
<INFO>_on_mqtt_connected called
<INFO>Connection State - Connected
<INFO>Successfully connected to Hub
<INFO>Enabling feature:c2d...
<INFO>subscribing to devices/ac7hs9q2re/messages/devicebound/# with qos 1
<INFO>suback received for 1
<INFO>Successfully enabled feature:c2d
@@@@ connected
@@@@ send{'TEST': 1.01, 'MsgId': 1}
<INFO>Sending message to Hub...
<INFO>publishing on devices/ac7hs9q2re/messages/events/
<INFO>payload published for 2
<INFO>Successfully sent message to Hub
@@@@ send{'TEST': 2.02, 'MsgId': 2}
<INFO>Sending message to Hub...
<INFO>publishing on devices/ac7hs9q2re/messages/events/
<INFO>payload published for 3
<INFO>Successfully sent message to Hub
...
```

前項でデバイス登録した Azureクラウドを開くとテレメトリデータが入っていれば正常にデータ送信されています

デバイス > raspi_iot_1111



raspi_iot_1111
● 接続済み | データの最終受信日: 2022/11/15 8:53:34 | 状態: プロビジョニング済み | 組織: iotapp-20220514

生データ マップされたエリアス ファイル

①	>	2022/11/15 8:51:41	テレメトリ	({"TEST":2.02,"MsgId":2})
①	>	2022/11/15 8:51:38	テレメトリ	({"TEST":1.01,"MsgId":1})
	>	2022/11/15 8:51:38	デバイスが接続済み	

5. スマートメータアプリの実行とIoT デバイス登録

5-1. スマートメータアプリの実行

今回は公開したリポジトリからwisun_smartmeter_azure.py のサンプルコードを実行します

まずサンプルコードをWISUNライブラリの中にコピーします

```
$ cp wisun_smartmeter_azure.py ~/pi-smartmeter/
```

次にサンプルコードを編集します

編集する内容な前項で変更したパラメータと同一になります

```
$ cd ~/pi-smartmeter
$ vi wisun_smartmeter_azure.py

Azure
id_scope = "{ID スコープ}"
registration_id = "{デバイス ID}"
symmetric_key = "{主キー}"
provisioning_host = "global.azure-devices-provisioning.net"

# 動作設定
DEVICE = "BP35A3"
# DEVICE = "BP35C0"
# ブルート認証ID
ID_WISUN = "00000000000000000000000000000000"
# ブルート認証パスワード
PWD_WISUN = "SSSSSSSSSSSS"
```

次のようなコマンドを実行するとスマートメータアプリが起動し電力量がAzrue IoT Centralに送信されます

```
$ cd ~/pi-smartmeter
$ python3 sample.py

@@@@@ main() start
/home/pi/pi-smartmeter/wisun_smartmeter_azure.py:75: RuntimeWarning: This channel is already in use, continuing anyway. Use
GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(18, GPIO.OUT)
```

```

@@@ main() connect Azure
<INFO>Creating client for connecting using MQTT over TCP
<INFO>Registering with Provisioning Service...
<INFO>Enabling reception of response from Device Provisioning Service...
<INFO>Connect using port 8883 (TCP)
....
martMeter connected
request for SmartMeter
dousa 0x30

sekisan tanni 0.1[kWh]

sekisan 3871.3[kWh]

denryoku 175.0[W]

R=0.0[A] T=2.0[A]

{'sekisan': 3871.3, 'denryoku': 175.0, 'R': 0.0, 'T': 2.0, 'MsgId': 1}
<INFO>Sending message to Hub...
<INFO>publishing on devices/ac7hs9q2re/messages/events/
<INFO>payload published for 1
<INFO>Successfully sent message to Hub

```

5-2. Azure IoT Central上でデバイス登録

Raspberry側からAzure IoT Centralへデータ送信ができるようになったら Azureクラウド上でデバイス登録を行います
Azure IoT Central画面上で前項で作成したデバイスを開きます

デバイス > raspi_iot_1111

 **raspi_iot_1111**
 接続済み | データの最終受信日: 2022/11/15 8:53:34 | 状態: プロビジョニング済み | 組織: iotapp-20220514

生データ マップされたエイリアス ファイル

① >	2022/11/15 8:51:41	テレメトリ	("TEST":2.02,"MsgId":2)
① >	2022/11/15 8:51:38	テレメトリ	("TEST":1.01,"MsgId":1)
>	2022/11/15 8:51:38	デバイスが接続済み	

デバイスを開くと Raspberry Piが送信したデータが表示されているはずです

デバイス > raspi_iot_1111

 **raspi_iot_1111**
 接続済み | データの最終受信日: 2022/11/22 15:52:20 | 状態: プロビジョニング済み | 組織: iotapp-20220514

生データ マップされたエイリアス ファイル

タイムスタンプ ↓	メッセージの種類	イベントの作成時刻	モデル化されていないデータ	エラー
① >	2022/11/22 15:52:20	テレメトリ	("sekisan":3871.3,"denryoku...	
>	2022/11/22 15:51:19	デバイスが接続済み		

画面上部にある「テンプレートの管理」から「テンプレートの自動作成」を選択します

接続 **テンプレートの管理** デバイスの管理

デバイス

テンプレートの自動作成

テンプレートの割り当て

 **raspi_iot_1111**
 接続済み | データの最終受信日: 2022/11/22 15:52:20 | 状態: プロビジョニング済み | 組織: iotapp-20220514

生データ マップされたエイリアス ファイル

タイムスタンプ ↓	メッセージの種類	イベントの作成時刻
① >	2022/11/22 15:52:20	テレメトリ
>	2022/11/22 15:51:19	デバイスが接続済み

Azure IoT Centralへ送信したデータからテンプレートが自動作成されました
この画面上で緑枠の部分の項目を編集し[保存]ボタンを押下します

積算電力 (W) - sekisan
 瞬間電力 (W) - denryoku

R相 (A) - R
T相 (A) - T

デバイス テンプレート > raspi_iot_1111 > モデル > raspi_iot_1111

デバイス テンプレート > raspi_iot_1111 > モデル > raspi_iot_1111

raspi_iot_1111 ルート 発行済み
このデバイス モデルに固有の機能を追加します。詳細情報

保存 + 機能の追加 ID の編集 エクスポート 削除 ... DTDL の編集

表示名	名前 *	機能の種類 *	セマンティックの種類 *		
積算電力(W)	seisan	デレトリ	なし	X	✓
瞬間電力(W)	denryoku	デレトリ	なし	X	✓
R相(A)	R	デレトリ	なし	X	✓
T相(A)	T	デレトリ	なし	X	✓

+ 機能の追加

テンプレートの修正/保存が終わったらテンプレートを公開します
公開することでAzure IoT Central上でデータが扱えるようになります

バージョン テストデバイスの管理 公開 名前の変更 削除

デバイス テンプレート > raspi_iot_1111 > モデル > raspi_iot_1111

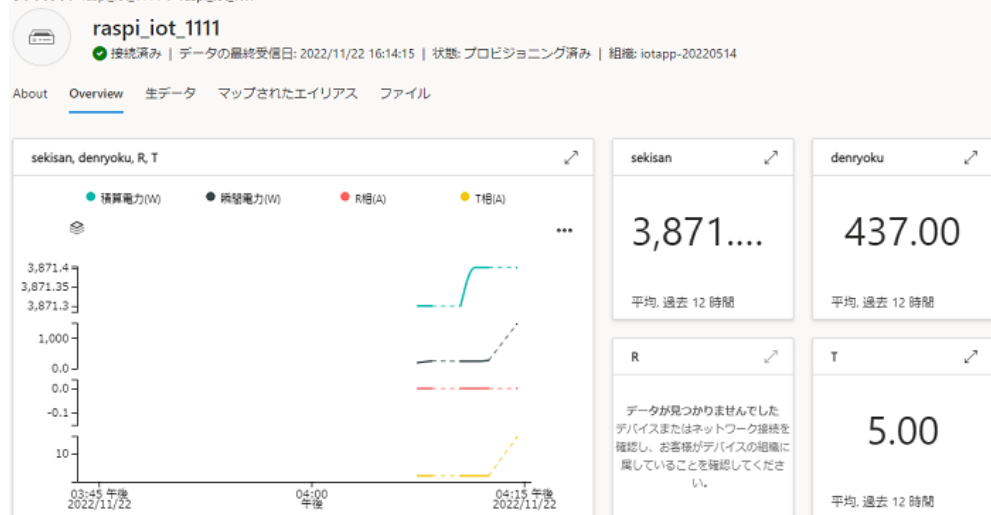
raspi_iot_1111 ルート ドラフト
このデバイス モデルに固有の機能を追加します

保存 + 機能の追加 ID の編集

5-3. データの可視化

前項で登録したテンプレートを用いて可視化を行います
デバイス画面を開くとAzure IoT Centralに保存されたスマートメータのテレメトリ情報が可視化されます

デバイス > raspi_iot_1111 > raspi_iot_1111



6. Blobストレージへのエクスポート

スマートメータ情報がAzure IoT Centralに取り込めるようになったらテレメトリ情報をストレージに保存します

6-1. Blobストレージの作成

Azure ホームからストレージアカウントを作成します
例: azstorege1213

ホーム > ストレージ アカウント >

ストレージ アカウントを作成する

基本 詳細設定 ネットワーク データ保護 暗号化 タグ 確認および作成

Azure Storage は、高可用性、セキュリティ、耐久性、スケーラビリティ、冗長性を備えたクラウド ストレージを提供する Microsoft が管理するサービスです。Azure Storage には、Azure BLOB (オブジェクト)、Azure Data Lake Storage Gen2、Azure Files、Azure Queues、Azure Tables が含まれます。ストレージ アカウントのコストは、使用量と、下で選ぶオプションに応じて決まります。 [Azure ストレージ アカウントの詳細](#)

プロジェクトの詳細

新しいストレージ アカウントを作成するサブスクリプションを選択します。ストレージ アカウントを他のリソースと一緒に整理して管理するには、新規または既存のリソース グループを選択します。

サブスクリプション *

sub_kazunyan20220514

リソース グループ *

IOTC

新規作成

インスタンスの詳細

レガシストレージ アカウントの種類を作成する必要がある場合は、[こちら](#) をクリックしてください。

ストレージ アカウント名 *

azstorege1213

地域 *

(Asia Pacific) Japan East

パフォーマンス *

☒ Standard: ほとんどのシナリオに対して推奨される (汎用 v2 アカウント)

☐ Premium: 低遅延が必要なシナリオにお勧めします。

冗長性 *

geo 冗長ストレージ (GRS)

☒ リージョンが利用できなくなった場合に、データへの読み取りアクセスを行えるようにします。

Review

< 前へ

次へ: 詳細設定 >

ストレージアカウントが作成できたら認証情報を取得しメモしておきます
例：

```
azstorege1213 | アクセス キー
key1
キー   : sKL5KQ3KXzQOjmfTbMtW//pzWRUfYdVT66Z+GDKOr4MnPQngxWrEKerMz6OukBmaenqu8kDBNy8b+AstVPF1Fw==
接続文字列:
DefaultEndpointsProtocol=https;AccountName=azstorege1213;AccountKey=sKL5KQ3KXzQOjmfTbMtW//pzWRUfYdVT66Z+GDKOr4MnPQngxWrEKerMz6OukBmaenqu8kDBNy8b+AstVPF1Fw==;EndpointSuffix=core.windows.net
```

概要

アクティビティ ログ

タグ

問題の診断と解決

アクセス制御 (IAM)

データ移行

イベント

ストレージ ブラウザー (プレビュー)

データ ストレージ

コンテナー

ファイル共有




キュー

テーブル

セキュリティとネットワーク

ネットワーク

Azure CDN



アクセス キー
 交換リマインダーの設定
  最新の情報に更新

アクセス キーは、このストレージ アカウントに対するアプリケーションの要求を認証します。これらのキーは、Azure Key Vault のような安全な場所に保管し、頻繁に新しいキーで置き換えてください。2 つのキーを用意すれば、一方を使用したまま他方で置き換えることができます。

このストレージ アカウントを使用するすべての Azure リソースとアプリでキーを更新してください。
[ストレージ アカウントのアクセス キーの管理に関する詳細情報](#)

ストレージ アカウント名

azstorage1213

key1  キーの交換


最終交換: 2022/12/13 (24 日前)

キー

表示

接続文字列

表示

key2  キーの交換

最終交換: 2022/12/13 (24 日前)

キー

表示

接続文字列

表示

次にblobストレージのコンテナを作成します

前項で作成したblobストレージを開き「新しいコンテナ」を押下し名前を付けます

例: cont1213



The screenshot shows the Microsoft Azure portal interface. On the left, the navigation pane is open, and 'アクセス キー' (Access Keys) is selected. The main content area shows the 'azstorage1213' storage account details. On the right, the '新しいコンテナ' (New Container) dialog box is open, showing the container name 'cont1213' and the access level 'プライベート (匿名アクセスはありません)' (Private (no anonymous access)).

6-2. エクスポート設定

保存するblobストレージの準備ができれば IoT Centralからエクスポート設定を行います

左側のリストから「データのエクスポート」を選択し、名前を付けます

エクスポートするデータの種別は「テレメトリ」を選択

宛先はまだ登録していませんので「新しく作成」を押下します

≡

接続

② デバイス

📁 デバイス グループ

📄 デバイス テンプレ...

📄 エッジ マニフェスト

分析

📊 データ エクスプロ...

📊 ダッシュボード

管理

📄 ジョブ

拡張

🔗 規則

📄 データのエクスボ...

セキュリティ

📄 監査ログ

🔍 アクセス許可

設定

📄 アプリケーション

📄 カスタマイズ

📄 保存 ✕ キャンセル 📄 名前の変更

エクスポート > power_export1213

power_export1213

🔴 有効

データ

絞り込むためにフィルターを追加しない限り、すべてのデバイスからデータがエクスポートされます。[詳細情報](#)

エクスポートするデータの種類 *

テレメトリ

+ フィルター + メッセージプロパティのフィ...

Enrichments

エクスポートに追加情報を追加します。これは、エクスポートされたメッセージにキーと値のペアとして表示されます。[詳細情報](#)

+ カスタム文字列 + プロパティ

宛先

エクスポートする宛先を選択します。宛先が見つからない場合は、[新しく作成](#)してください。

+ 宛先

新しい宛先ダイアログが開きますので次のように設定し「作成」を押下します

宛先: 任意 (例: export1213)
ターゲットの種類: Azure Blob Storage
接続文字列: (ストレージアカウント作成時にメモした文字列)
コンテナ: (前項で作成したコンテナ 例: cont1213)

新しい宛先

✕

新しい宛先を作成するには、宛先の種類を選択し、接続情報を入力します。[詳細情報](#)

宛先名 *

export1213

ターゲットの種類 *

Azure Blob Storage

認可

接続文字列

接続文字列 *

DefaultEndpointsProtocol=https;AccountName=azstorege1...

コンテナ * ⓘ

cont1213

作成

キャンセル

宛先が登録されましたので次はデータ変換を登録します

エクスポート > power_export1213

power_export1213

☒ 有効

データ

絞り込むためにフィルターを追加しない限り、すべてのデバイスからデータがエクスポートされます。[詳細情報](#)

エクスポートするデータの種類 *

テレメトリ

+ フィルター + メッセージプロパティのフィ...

Enrichments

エクスポートに追加情報を追加します。これは、エクスポートされたメッセージにキーと値のペアとして表示されます。[詳細情報](#)

+ カスタム文字列 + プロパティ

宛先

エクスポートする宛先を選択します。宛先が見つからない場合は、[新しく作成](#)してください。

宛先 *

export1213

データ変換

+ 変換

×

+ 宛先

データ変換の設定画面が表示されます

まず1. の項目でデバイス名を指定

2. の変換クエリを次のものに置き換え

入力が終わったら[保存]を押下して終了します

```
# ここで指定した変換クエリを使用して、エクスポートされた各メッセージを
# 別の形式に変更します。次の例の使用を開始できます。
# 言語の詳細については、ドキュメントを参照してください：
# https://aka.ms/dataexporttransformation
# 以下に、'RangeOfMotion' という名前の機能の値を検索して割り当てるサンプル クエリを示します'
# import 'iotc' as iotc;
# { RangeOfMotion: .telemetry | iotc::find(.name == 'RangeOfMotion').value }
import "iotc" as iotc;
# ここで指定した変換クエリを使用して、エクスポートされた各メッセージを
# 別の形式に変更します。次の例の使用を開始できます。
# 言語の詳細については、ドキュメントを参照してください：
# https://aka.ms/dataexporttransformation
# 以下に、'RangeOfMotion' という名前の機能の値を検索して割り当てるサンプル クエリを示します'
# import 'iotc' as iotc;
# { RangeOfMotion: .telemetry | iotc::find(.name == 'RangeOfMotion').value }
import "iotc" as iotc;
{
  schema: "default@v1",
  applicationId: .applicationId,
  deviceId: .device.id,
  messageSource: .messageSource,
  enqueuedTime: .enqueuedTime | split(".")[0],
  sekisan: .telemetry | iotc::find(.name == "sekisan").value,
  denryoku: .telemetry | iotc::find(.name == "denryoku").value,
  module: .module
}
```


データ変換

これは高度な機能であり、まずデータ変換クエリにアクセスすることをお勧めします。変換を実行すると、エクスポートされたメッセージの形が新しい形式に変更されます。

1. 入力メッセージを追加する

aspiot_1111

1

{

2

"applicationId": "3faad2c7-083c-48aa-44d9-7f3363e0b6b7",

3

"enqueueTime": "1939-06-11T09:23:25.738976019Z",

4

"messageSource": "telemetry",

5

"telemetry": {

6

{

7

"id": "dtmi:iotapp20220514:raspi_iot_1111:sekisan;1",

8

"name": "sekisan",

9

"value": 8.98574632895673e+307

10

},

11

{

12

"id": "dtmi:iotapp20220514:raspi_iot_1111:denryoku;1",

13

"name": "denryoku",

14

"value": 1.2476425009335365e+308

15

},

16

{

17

"id": "dtmi:iotapp20220514:raspi_iot_1111:R;1",

18

"name": "R",

19

"value": 3.581065551632079e+307

20

},

21

{

22

"id": "dtmi:iotapp20220514:raspi_iot_1111:F;1",

23

"name": "F",

24

"value": 1.0349597990191937e+308

25

}

26

},

27

"device": {

28

"id": "vgzumaz4kj3d",

29

"name": "Virtual feed",

30

"templateId": "dtmi:modelDefinition:iotapp20220514:raspi_iot_1111;1",

31

"templateName": "raspi_iot_1111",

32

"properties": {

33

"reported": []

34

},

35

"cloudProperties": [],

36

"simulated": true,

37

"approved": false,

38

"blocked": true,

39

"provisioned": false,

40

"organizations": [

41

"leaf-organization"

42

],

43

"organizationPaths": [

44

[

45

{

46

"id": "root-organization",

47

"displayName": "Root Organization"

48

}

49

]

50

}

51

}

52

}

53

2. 変換クエリを作成する

1

ここで指定した変換クエリを使用して、エクスポートされた各メッセージを

2

別の形式に変換します。次の例の使用を開始できます。

3

各語句の詳細については、ドキュメントを参照してください:

4

https://aka.ms/dataserviceexporttransformation

5

以下に、"RangeOfMotion" という名前の機能の値を格納して割り当てするサンプル クエリを示しま

6

import "iotc" as iotc;

7

{ RangeOfMotion: .telemetry | iotc::find(.name == "RangeOfMotion").value }

8

import "iotc" as iotc;

9

{

10

schema: "default@v1",

11

applicationId: .applicationId,

12

deviceId: .deviceId,

13

messageSource: .messageSource,

14

enqueueTime: .enqueueTime,

15

sekisan: .telemetry | iotc::find(.name == "sekisan").value,

16

denryoku: .telemetry | iotc::find(.name == "denryoku").value,

17

module: .module

18

}

19

3. 出力メッセージをプレビューする

1

{

2

"applicationId": "3faad2c7-083c-48aa-44d9-7f3363e0b6b7",

3

"denryoku": 1.2476425009335365e+308,

4

"deviceId": "vgzumaz4kj3d",

5

"enqueueTime": "1939-06-11T09:23:25.738976019Z",

6

"messageSource": "telemetry",

7

"module": null,

8

"schema": "default@v1",

9

"sekisan": 8.98574632895673e+307

10

}

更新

キャンセル

エクスポート登録が終了しテレメトリデータがストレージに保存されるとエクスポートの状態が「正常」となります

エクスポート > power_export1213

power_export1213

有効

データ

絞り込むためにフィルターを追加しない限り、すべてのデバイスからデータがエクスポートされます。[詳細情報](#)

エクスポートするデータの種類

テレメトリ

+ フィルター + メッセージプロパティのフィ...

Enrichments

エクスポートに追加情報を追加します。これは、エクスポートされたメッセージにキーと値のペアとして表示されます。[詳細情報](#)

+ カスタム文字列 + プロパティ

宛先

エクスポートする宛先を選択します。宛先が見つからない場合は、[新しく作成](#)してください。

宛先 *

export1213

+ 宛先

データ変換

編集

エクスポートの状態

✓ 正常

詳細

エクスポートが正常に行われるようになったらコンテナを開いて中身を確認しましょう

ホーム > azstorege1213

azstorege1213 | コンテナ

ストレージアカウント

検索

+ コンテナ

アクセスレベルを変更します

コンテナを復元する

更新

削除

プレフィックスによるコンテナの検索

削除されたコンテナを表示する

名前	最終変更日時	パブリック アクセス レベル	リソース状態	
<input type="checkbox"/> logs	2022/12/13 13:55:35	プライベート	利用可能	...
<input type="checkbox"/> cont1213	2022/12/13 14:09:53	プライベート	利用可能	...

概要

アクティビティ ログ

タグ

問題の診断と解決

アクセス制御 (IAM)

データ移行

イベント

ストレージブロッカー (プレビュー)

データストレージ

コンテナ

コンテナを開くとテレメトリデータが入ったフォルダがされているはずですが



6-3. Blobストレージのエクスポート

blobストレージに格納されたスマートメータのデータは Azure ストレージから取り出すこともできますが
まとまったデータとして取り出すためには外部ツールである Azure Storage Explorerを使います

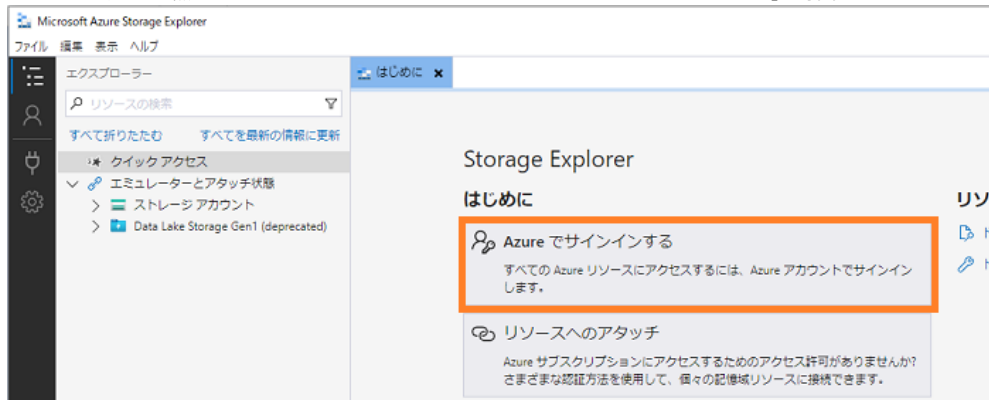
blobストレージ画面の上にある「Explorerで開く」を押下すると Azure Storage Explorer画面が表示されます

ここで「Azure Storage Explorer」ボタンを押すことにより外部ツールが起動します

初回利用時でまだAzure Storage Explorerをインストールしていない場合は下にあるリンクからツールのダウンロードとインストールを行ってください



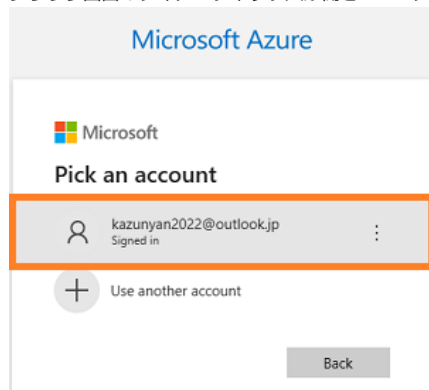
ツールを起動した時点ではAzureサインインされていないので「Azure でサインインボタン」を押下します



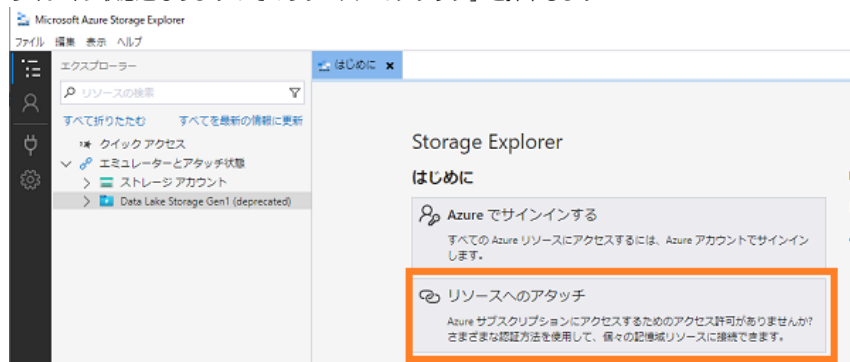
接続先のAzure選択画面ではデフォルトの"Azure"を選択したまま「次へ」を押下します



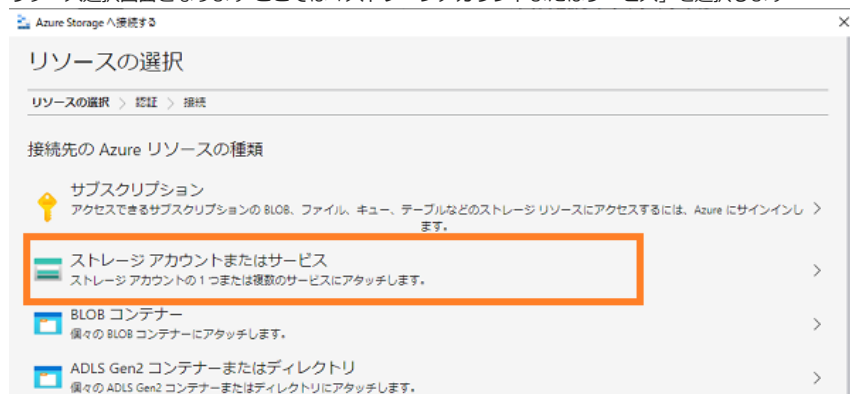
ブラウザ画面のダイアログボックスが開きAzureアカウントの選択表示となりますので blobストレージを作成したアカウントを選択しましょう



サインイン状態となりますので「リソースへのアタッチ」を押下します



リソース選択画面となります ここでは「ストレージアカウントまたはサービス」を選択します



接続方法の選択画面となります

今回はストレージアカウント作成時に取得した接続文字列を使用するため「接続文字列」を選択します

Azure Storage へ接続する

接続方法の選択

リソースの選択 > 接続方法の選択 > 接続情報の入力 > 概要

ストレージ アカウントへの接続方法

- ☒ 接続文字列 (キーまたは SAS)
- ☐ Shared Access Signature URL (SAS)
- ☐ アカウント名とキー

戻る 次へ キャンセル

接続文字列の入力画面になりましたら接続文字列のボックスにストレージアカウント作成時にメモした接続文字列をセットします
有効な文字列が入力されると表示名の欄に作成したストレージ名が表示されます 作成したストレージ名が正しいことを確認し「次へ」を押下します

Azure Storage へ接続する

接続情報の入力

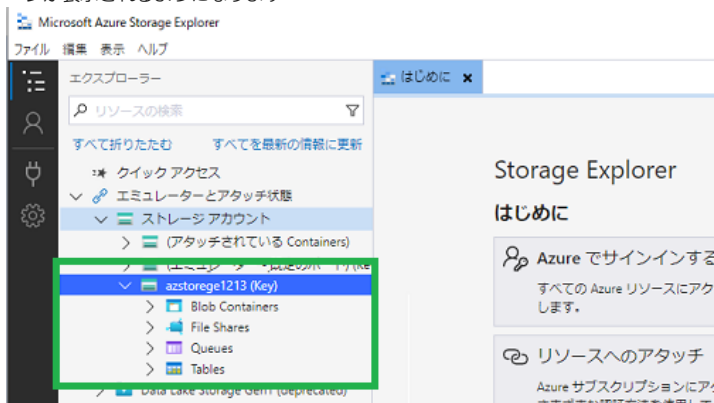
リソースの選択 > 接続方法の選択 > 接続情報の入力 > 概要

表示名:
azstorage1213 自動で表示される

接続文字列:
DefaultEndpointsProtocol=https;AccountName=azstorage1213;AccountKey=sKL5KQ3KXzQQjmfTbMtW//pzWRUfNdVT66Z-GDKO4MnPQngxWrEKeRMz6Ouk8maenqu&kD8Ny8b+ASvPFIFw=:;EndpointSuffix=core.windows.net

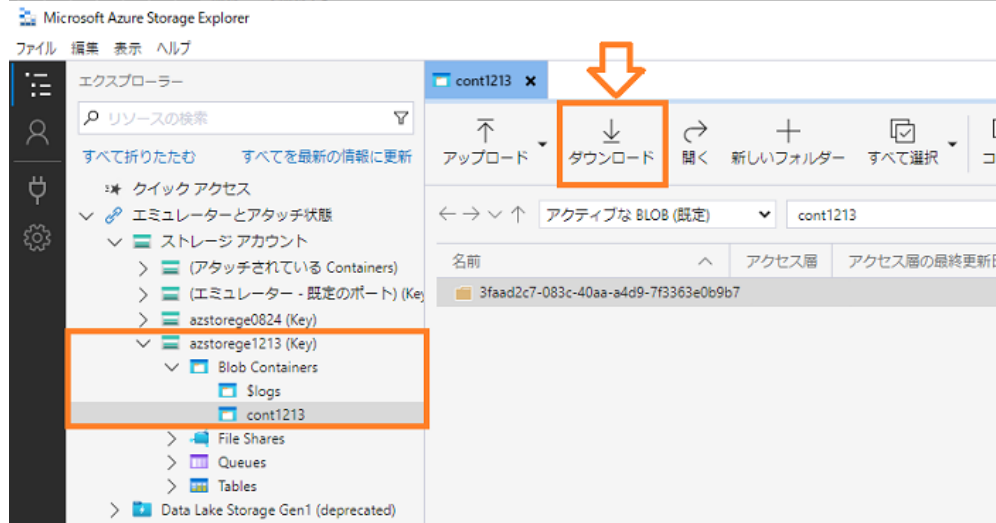
戻る 次へ キャンセル

ストレージアカウントとの接続に成功すると緑枠のようにストレージが表示されるようになります



接続できたストレージからblobコンテナを選択するとAzure上のblobストレージと同じ内容が表示されているはずです
ここで画面上にある「ダウンロード」ボタンを押下すると選択されたblobコンテナの内容がダウンロードされます

ダウンロードされたデータはblobストレージと同じフォルダ構成となっていますので必要に応じてアーカイブして使用してください



最後に

これでスマートメータの情報がAzure上で利用可能になりました