Bartłomiej Lipiński

402694

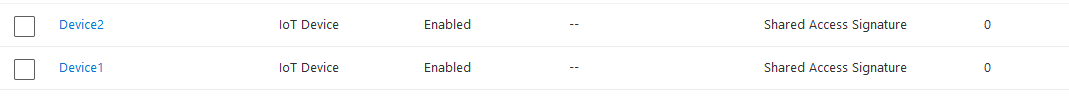
<https://github.com/GGGeralt/IoT-Project>

**Functions** – Projekt w którym są napisane funkcje zdeployowane jako Azure Function App

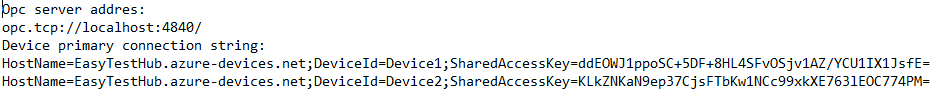
**OPCUAAGENT** – Agent

**ServiceSdkDemo (Ex1)** – Ulepszony kod z zajęć używany do testowania

Jeden Agent obsługuje jedno urządzenie, a nie całą linię produkcyjną jak było wcześniej. Aby agent osbugiwał np**. Device 3**, należy utworzyć w **Azure** kolejny **IotHub\_Device**, i wkleić primary connection string na końcu pliku z kluczami. Muszą być one po kolei, czyli klucz do **Device 4**, nie może być wcześniej niż **Device 5**.



Połączenie do IotHuba odbywa się przez klucz w pliku „Settings.txt”, który wygląda tak:



Obraz zawierający tekst

Opis wygenerowany automatycznie

Połączenie z serwerem OPC jest zrobione za pomocą biblioteki Opc.uafx, a dane odczytywanie raz na sekundę w funkcji **OneDeviceMagic**.

Metody można wywołać po prostu wpisując ich nazwę.

Obraz zawierający tekst

Opis wygenerowany automatycznie

**EmergencyStop** zatrzymuje produkcję na aktualnie połączonym device, włączając też flagę błędu **EmergencyStop**.

**ResetErrorStatus** usuwa wszystkie informacje o błędach.

**Maintenance** ustawia aktualną datę w **DeviceTwin** w polu **LastMaintenanceDate**.

**ReduceProductionRate** zmniejsza produkcję device o 10.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Agent wysyła telemetrię w formacie



Raz na sekundę.

Dane są przechowywane w **DeviceTwin** za pomocą funkcji **CheckIfUpdateTwin**. Najpierw sprawdzam, czy są odpowiednie pola. Próba odczytania pola, którego nie ma kończyła się dziwnym zachowaniem programu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Następnie, jeżeli takie pola istnieją, porównujemy wartości (DeviceErrors i ProductionRate), czy jest w ogóle potrzeba aktualizacji **DeviceTwina**.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeżeli wykryło, że należy zrobić aktualizacje, odpowiednio są wpisywane odpowiednie wartości.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Zapisuję tam

**desiredProperties**:

**ProductionRate** – **ProductionRate**, na które chcemy żeby device zmienił wartość

**reportedProperties**:

**DeviceErrors** - jako błędy po przecinku(jako napis),

**ProductionRate** - odczytane z Opc Device,

**LastMaintenanceDate** – aktualizowane za każdym razem, jak tylko Metoda **Maintenance** zostanie wywołana

**LastErrorDate** – aktualizowane za każdym razem, jak tylko aktualizowane są **DeviceErrors**.

Przykładowa zawartość **DeviceTwina** znajduje się w pliku **DeviceTwinExample.txt**.

Punkt **Data calculations** został wykonany przy pomocy Azure Stream Analytics, kod:

SELECT

    workorderid,

    AVG(temperature)

    INTO

        [asa-average-temperature-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(minute, 5);

SELECT

    workorderid,

    SUM(badCount) as badCoundSum

    INTO

        [asa-bad-count-sum-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(hour, 1);

SELECT

    workorderid,

    SUM(goodCount) as goodCountSum

    INTO

        [asa-good-count-sum-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(hour, 1);

SELECT

    workorderid,

    ((SUM(goodCount) / (SUM(goodCount) + SUM(badCount))) \* 100) as percentage

    INTO

        [asa-good-percentage-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(minute, 15);

SELECT

    workorderid,

    MAX(temperature)

    INTO

        [asa-max-temp-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(minute, 5);

SELECT

    workorderid,

    MIN(temperature)

    INTO

        [asa-min-temp-out]

    FROM

        [asa-in-blip]

    GROUP BY workorderid, TumblingWindow(minute, 5);

SELECT \* FROM [asa-in-blip];