Projekt IoT – dokumentacja

Link do repozytorium: <https://github.com/WiktoriaSzkudlarek/IoT-project>

# Uruchamianie aplikacji

Przed pobraniem aplikacji, na platformie Azure należy utworzyć:

* **Resource group**,
* **IoT Hub i 2 Device’y** (u mnie są o nazwach: Device1, Device2),
* **Storage account z 4 kontenerami typu Blob** (u mnie są o nazwach: telemetry, production-kpi, temperature-avg-min-max, device-errors),
* **Stream Analytics job z 1 Inputem oraz zapytania kierujące wyniki do 4 Outputów** (u mnie mają takie same nazwy jak kontenery).

Po pobraniu projektu z repozytorium należy wejść do folderu **ServiceSdkDemo.Console**, a następnie kliknąć na plik **ServiceSdkDemo.Console.sln**, który uruchomi projekt w Visual Studio (o ile jest zainstalowany).

W panelu Solution Explorer trzeba wyszukać i wcisnąć plik **config.json**, który ma następującą strukturę:

{

"ServiceConnectionString": "",

"OpcClientConnectionString": "opc.tcp://localhost:4840/",

"StorageConnectionString": "",

"Devices": [

{

"DeviceId": "1",

"ConnectionString": ""

},

{

"DeviceId": "2",

"ConnectionString": ""

}

],

"Blobs": [

{

"BlobName": "production-kpi.json",

"Method": "ReduceProductionRate"

},

{

"BlobName": "device-errors.json",

"Method": "EmergencyStop"

}

]

}

Parametry będące pustymi znakami należy skopiować z portalu Azure.

* **ServiceConnectionString** znajduje się w: IoT Hub: Security settings > iothubowner > Primary connection string.
* **ConnectionString** znajduje się w: IoT Hub: Device management > Devices > [nazwa urządzenia] > Primary connection string.
* **StorageConnectionString** znajduje się w: Storage account: [nazwa kontenera] > Security + networking > Access keys > key1 > Connection string

Po wpisaniu connectionStringów, zapisaniu pliku config.json oraz **włączeniu symulatora i dodaniu odpowiedniej ilości urządzeń** można uruchomić projekt, który powinien włączyć dwa mniejsze projekty:

* ServiceSdkDemo.Console otwierające okienka dla agenta i urządzeń działających przez DeviceSdkDemo.Console
* oraz BlobStorageDemo.Desktop pozwalające na pobranie danych z chmury.

Projekt ServiceSdkDemo.Console uruchamia tyle okienek dla urządzeń, ile jest ich wpisanych w pliku konfiguracynym.

Połączenie z serwerem OPC UA odbywa się w projekcie DeviceSdkDemo.Console, gdzie tworzony jest obiekt klasy VirtualDevice i istnieje ono dla danego urządzenia do momentu zamknięcia okienka swojego lub agenta. Dane są pobierane co około 1 sekundę.

# Komunikacja z platformą Azure

## Wiadomości D2C

Aplikacja wysyła do platformy dane telemetryczne za pomocą D2C message, którego sekcja Body ma następujące parametry z przykładową zawartością:

"Body": {

"device": "Device2",

"productionStatus": 1,

"workorderId": "26525968-b224-46eb-9de1-fe270f993039",

"goodCount": 6,

"badCount": 1,

"temperature": 61.38280472731428

}

Ten mechanizm zostaje uruchamiany co ok. 1 sekundę i przechowywany jest w Blob Storage’u o nazwie **telemetry**.

Oprócz danych telemetrycznych, wysyłane są również raporty o zmianie błędów urządzeń, które przechowywane są w Blob Storage’u o nazwie **device-errors**.

"Body": {

"Count": 12,

"Device": "Device2"

}

## Device Twin

Za pomocą Device Twin przechowywane są raporty o aktualnym tempie produkcji oraz błędach urządzenia. Mechanizm się uruchamia jeśli zostanie odnotowana zmiana jednego z tych parametrów.

Poniżej znajduje się przykładowa treść dostarczana przez Device Twin oraz informacje na urządzeniu informujące o uruchomieniu się tej formy komunikacji:

"properties": {

"desired": {

"$metadata": {

"$lastUpdated": "2023-05-06T08:21:13.923281Z"

},

"$version": 1

},

"reported": {

"productionRate": 20,

"deviceErrors": "None",

"lastDeviceErrorsDate": "2023-05-06T00:00:00+02:00",

"$metadata": {

"$lastUpdated": "2023-05-06T13:21:29.5156727Z",

"productionRate": {

"$lastUpdated": "2023-05-06T13:21:29.5156727Z"

},

"deviceErrors": {

"$lastUpdated": "2023-05-06T13:21:29.5156727Z"

},

"lastDeviceErrorsDate": {

"$lastUpdated": "2023-05-06T09:39:54.80008Z"

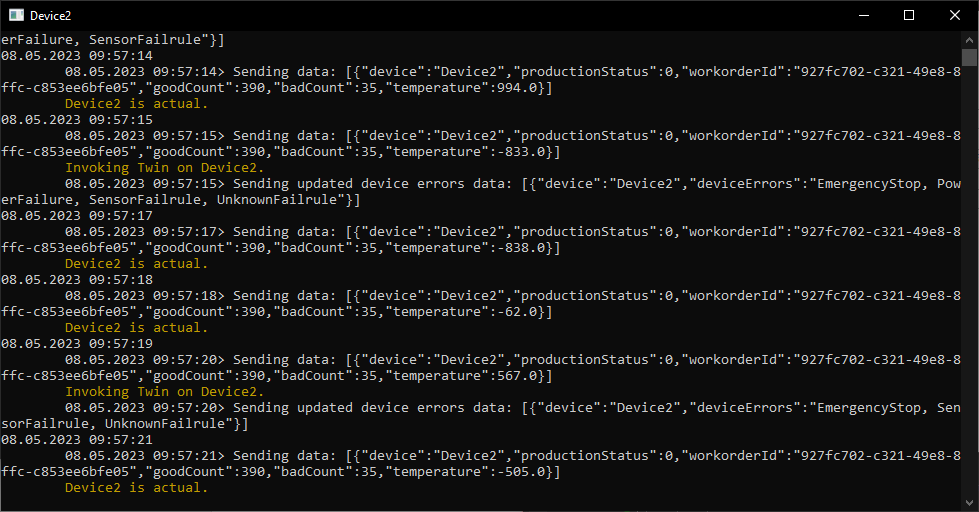
}

},

"$version": 13

}

},

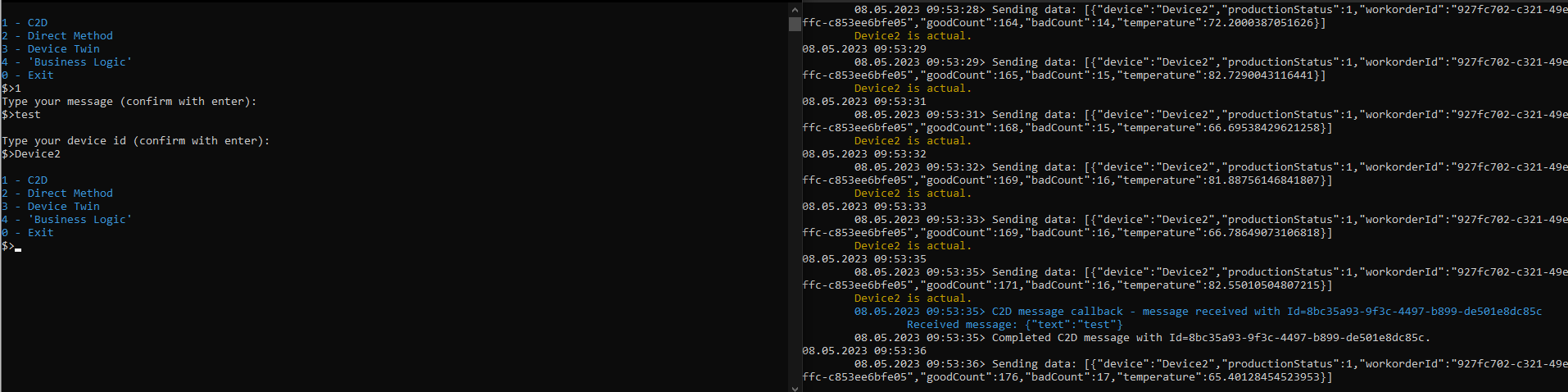


## Dostępne metody

W kliencie dostępne są następujące metody:

### C2D

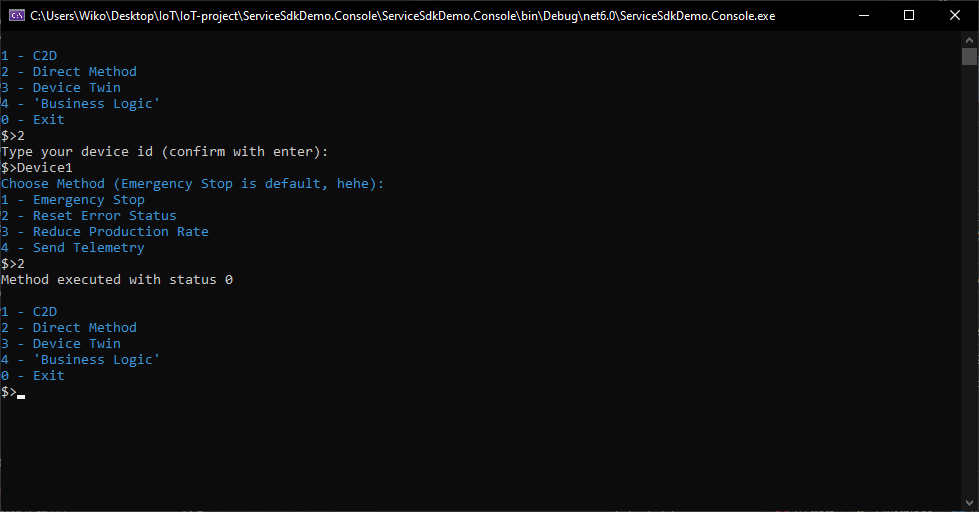
Wysyłanie wiadomości w wybraną treścią dla parametru text do wybranego urządzenia.



### Direct Method

Wykonanie wybranej metody, która wpłynie na wirtualną produkcję dla wybranego urządzenia.

* EmergencyStop – zatrzymanie produkcji na wybranym urządzeniu
* ResetErrorStatus – usuwanie błędów na wybranym urządzeniu
* ReduceProductionRate – zmniejszanie tempa produkcji o 10% na wybranym urządzeniu
* SendTelemetry – wysyłanie pojedynczej dany telemetrycznej z wybranego urządzenia



### Device Twin

Ta opcja działa tak jak na przykładowym kodzie z zajęć, czyli po podaniu nazwy urządzenia i nazwy właściwości, aktualizuje ją przypisując pseudolosową wartość.

# Zaimplementowane kalkulacje

Za pomocą Stream Analytics job wykonywane są kalkulacje dotyczące:

* Wskaźnika wydajności produkcji – pokazuje KPI na dane urządzenie z podziałem na 5 minut, którego wyniki są przechowywane w Blobie **production-kpi**.

**SELECT**

device **AS** Device, (( **SUM**(goodCount) / (**SUM**(goodCount) + **SUM**(badCount))) \* 100) **AS** Kpi

**INTO**

[production-kpi]

**FROM**

[iot]

**WHERE**

device **IS** **NOT** **NULL**

**GROUP** **BY**

device, TumblingWindow(**minute**, 5);

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

* Temperatury – przechowuje średnią, minimalną i maksymalną temperaturę na dane urządzenie z podziałem na 5 minut, którego wyniki są przechowywane w Blobie **temperature-avg-min-max**.

**SELECT**

device, **AVG**(temperature) **AS** temperatureAvg, **MIN**(temperature) **AS** temperatureMin, **MAX**(temperature) **AS** temperatureMax

**INTO**

[temperature-**avg**-**min**-**max**]

**FROM**

[iot]

**WHERE**

device **IS** **NOT** **NULL**

**GROUP** **BY**

device, TumblingWindow(**minute**, 5);

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

* Ilość zarejestrowanych błędów urządzenia – zlicza łączną liczbę napotkanych błędów w czasie 1 minuty i przechowuje wyniki w Blobie **device-errors**.

**SELECT**

**COUNT**(deviceErrors) **AS** Count, device **AS** Device

**INTO**

[device-errors]

**FROM**

[iot]

**WHERE**

deviceErrors **IS** **NOT** **NULL**

**GROUP** **BY**

device, TumblingWindow(**minute**, 1);

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

* Jest również Blob o nazwie **telemetry**, który przechowuje wszystkie wiadomości wysłane przez C2D.

**SELECT**

\*

**INTO**

[telemetry]

**FROM**

[iot];

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Przykładowe wyniki są zapisane w folderze **Blobs**.

# Zaimplementowana logika biznesowa

Niestety nie byłam w stanie wymyślić jak wywoływać metody w chmurze, więc ten warunek jest wykonywany inaczej niż był zapisany w kryteriach.

Dane są pobierane z wybranych komtenerów, które później można pobrać do folderu **Blob**.

Aby „Logika biznesowa” zadziałała, trzeba pobrać pliki z kontenerów **device-errors** i **production-kpi** (domyślną nazwą pliku będzie [nazwa blobu].json). Następnie w okienku agenta, w „menu głównym” można wybrać opcję 4. Program odczyta „najświeższą” (pierwszą linijkę pliku) kalkulację z obu plików i na jej podstawie zadecyduje czy uruchomić metody EmergencyStop lub ReduceProductionRate.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie