

# **Dokumentacja projektu IoT**

Link do repozytorium: <https://github.com/adrian-sz/IoT2023>

## **Uruchamianie aplikacji**

W celu poprawnego działania aplikacji na platformie Microsoft Azure należy utworzyć:

- **Resource group** o dowolnej nazwie
- **IoT Hub**
- **2 device** (W przypadku mojego projektu występują one pod nazwami *Device1* i *Device2*)
- **Storage account**  
A w nim 4 kontenery typu **Blob** pod następującymi nazwami:
  - 1) **telemetry**
  - 2) **production-kpi**
  - 3) **temperature-avg-min-max**
  - 4) **device errors**
- **Stream Analytics job** z **1 inputem** wraz z łącznie **4 inputami** o nazwach takich samych jak podane wyżej kontenery

Po poprawnym pobraniu repozytorium z zamieszczonego wyżej linku prowadzącego do mojego repozytorium należy uruchomić „**ServiceSdkDemo.Console.sln**”, które otworzy projekt w Visual Studio (pod warunkiem, że jest zainstalowane). W panelu Solution Explorer należy wyszukać plik „**config.json**” i otworzyć go w kompilatorze.

Żeby aplikacja działała poprawnie musimy w pliku konfiguracyjnym wprowadzić klucze dostępu znajdujące się w platformie Azure w następujących pustych polach:

- **ServiceConnectionString**: IoT Hub → Security settings → Shared access policies → iothubowner → Primary connection string

- **StorageConnectionString**: Storage Account → Security + networking → Access keys → key1 → Connection string
- **Device1/Device2 ConnectionString**: IoT Hub → Devices → Device1/Device2 → Primary connection string

Po dodaniu kluczy dostępu należy upewnić się, że są one prawidłowe i zapisać plik „**config.json**”. Następnie należy uruchomić symulator „**IIoTSim.Desktop.application**” i dodać w nim zarówno **Device1** jak i **Device2**. Po wykonaniu tej czynności należy uruchomić „**Opc.Ua.SampleClient.application**” i w środkowym polu wkleić zawartość pola "**OpcClientConnectionString**" z pliku „**config.json**”, po czym przejść dalej aż do momentu ukazania się głównego ekranu aplikacji z widocznymi **Device1** i **Device2**.

**UWAGA:** Symulator korzysta z biblioteki, którego licencja typu trial zezwala na korzystanie z niej tylko przez 30 minut. Po tym czasie symulator wstrzymuje pracę i w celu korzystania z niego należy uruchomić proces symulacji ponownie.

Po wykonaniu powyższych kroków można uruchomić główny projekt, który automatycznie uruchamia projekty „**ServiceSdkDemo.Console**” i „**BlobStorageDemo.Desktop**”. Umożliwiają one kolejno uruchomienie urządzeń z pliku konfiguracyjnego „**config.json**”, otwieranie okienek dla agenta i urządzeń przez „**DeviceSdkDemo.Console**” i pozwalanie na pobranie danych z chmury. Projekt „**DeviceSdkDemo.Console**” łączy się z serwerem OPC UA, gdzie tworzony jest obiekt klasy VirtualDevice odświeżany co sekundę istniejący do momentu zamknięcia swojego okna.

## **Platforma Azure**

### **D2C**

Program co sekundę wysyła do platformy Azure dane za pomocą D2C message i przechowuje je w „**Blob storage**” o nazwie „**telemetry**” z przykładową zawartością:

```
12.10.2023 22:55:28
12.10.2023 22:55:28> Sending data: [{"device":"Device1","productionStatus":1,"workorderId":"b36bdf9-a
663-4d95-82d3-e8ac50b20733","goodCount":136,"badCount":14,"temperature":74.98010972207067}]
```

# Device Twin

Device twin przechowuje raporty o aktualnym tempie produkcji i jej błędach.

Mechanizm uruchamia się samoistnie jeśli nastąpi zmiana jednego z parametrów.

Przykładowa zawartość:

```
12.10.2023 23:03:11
12.10.2023 23:03:11> Sending data: [{"device":"Device1","productionStatus":0,"workorderId":"b36bdf9-a663-4d95-8
2d3-e8ac50b20733","goodCount":978,"badCount":81,"temperature":63.03866670535546}]
Invoking Twin on Device1.
12.10.2023 23:03:11> Sending updated device errors data: [{"device":"Device1","deviceErrors":"EmergencyStop"}]
12.10.2023 23:03:13
12.10.2023 23:03:13> Sending data: [{"device":"Device1","productionStatus":0,"workorderId":"b36bdf9-a663-4d95-8
2d3-e8ac50b20733","goodCount":978,"badCount":81,"temperature":25.0}]
Device is actual.
```

## Metody

Aplikacja pozwala na wykorzystanie następujących metod:

- **C2D**

Wysła wiadomość Client to Device:

```
$>
Type your message (confirm with enter):
$>This is a test message
Type your device id (confirm with enter):
$>Device1
1 - C2D
2 - Direct Method
3 - Device Twin
4 - 'Business Logic'
0 - Exit
$>
Device is actual.
12.10.2023 23:07:40
12.10.2023 23:07:40> Sending data: [{"device":"Device1","productionStatus":0,"workorderId":"b36bdf9-a663-4d95-8
2d3-e8ac50b20733","goodCount":978,"badCount":81,"temperature":24.3825428089977}]
Device is actual.
12.10.2023 23:07:41
12.10.2023 23:07:41> Sending data: [{"device":"Device1","productionStatus":0,"workorderId":"b36bdf9-a663-4d95-8
2d3-e8ac50b20733","goodCount":978,"badCount":81,"temperature":25.0}]
12.10.2023 23:07:41> C2D message callback - message received with Id=933ee796-65ec-49a9-ab45-e321498e43ab
Received message: {"text":"This is a test message"}
12.10.2023 23:07:41> Completed C2D message with Id=933ee796-65ec-49a9-ab45-e321498e43ab.
Device is actual.
12.10.2023 23:07:42
```

- **Direct Method**

Pozwala na zastosowanie jednej z metod wpływającej na produkcję podanego urządzenia:

- 1) EmergencyStop – Zatrzymuje produkcję
- 2) ResetErrorStatus – Usuwa błędy
- 3) ReduceProductionRate – Zmniejsza tempo produkcji o 10%
- 4) SendTelemetry – Wysła dane telemetryczne

```

1 - C2D
2 - Direct Method
3 - Device Twin
4 - 'Business Logic'
0 - Exit

$>2
Type your device id (confirm with enter):
$>Device2
Choose Method (Emergency Stop is default, hehe):
1 - Emergency Stop
2 - Reset Error Status
3 - Reduce Production Rate
4 - Send Telemetry

$>_

```

- **Device Twin**

Przypisuje do urządzenia pseudolosową wartość podanej nazwy właściwości.

```

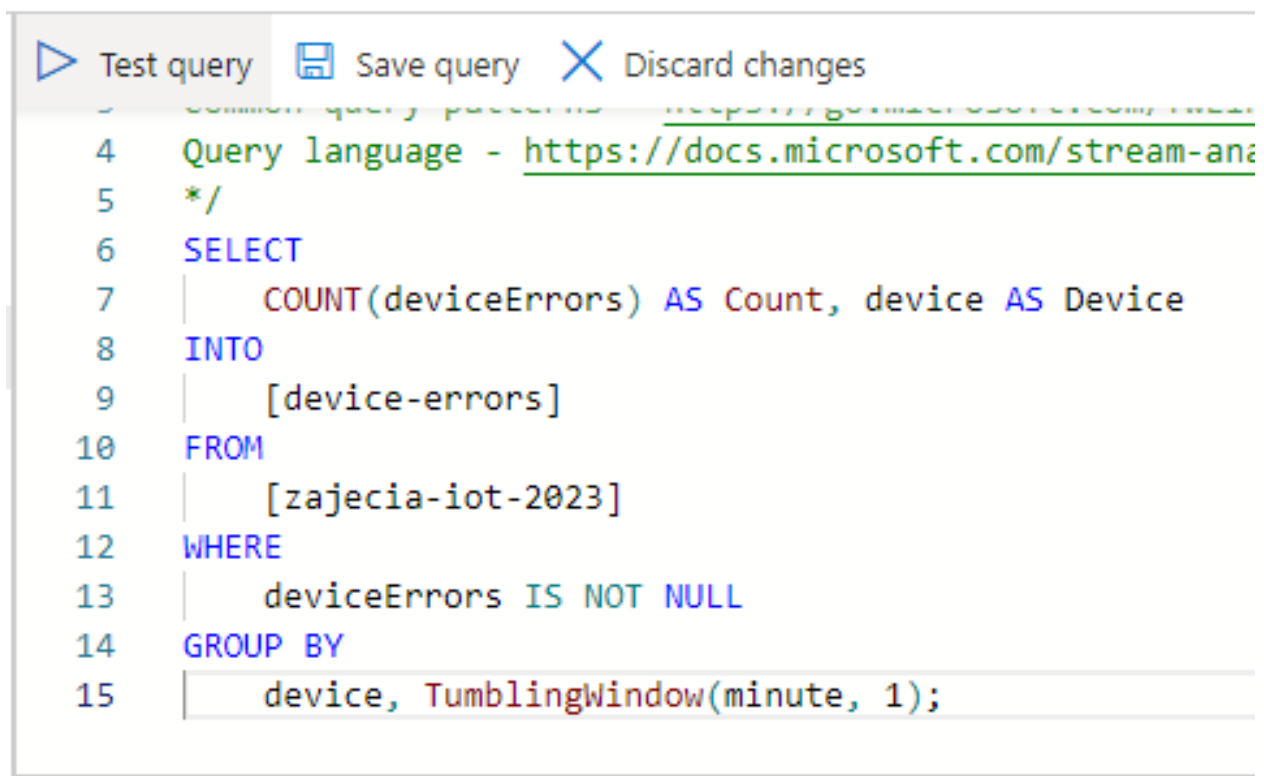
Desired property change:
{"Test":685579661,"$version":3}
Sending current time as reported property
Device1 is actual.

```

## Zapytania query

Dzięki zastosowaniu „**Stream analytics job**” mamy możliwość dokonywania kalkulacji na 4 outputach:

- **device errors**





The screenshot shows a query editor interface with a toolbar at the top containing icons for running a query (play button), saving a query (floppy disk), and discarding changes (X button). Below the toolbar, there is a list of common query patterns with a link to the Microsoft documentation. The main area displays a SQL query that counts device errors for a specific device over a one-minute tumbling window.

```

4 Query language - https://docs.microsoft.com/stream-analytics-query
5 */
6 SELECT
7     COUNT(deviceErrors) AS Count, device AS Device
8 INTO
9     [device-errors]
10 FROM
11     [zajecia-iot-2023]
12 WHERE
13     deviceErrors IS NOT NULL
14 GROUP BY
15     device, TumblingWindow(minute, 1);

```

- **production-kpi** - pokazuje KPI na urządzeniach

▶ Test query  Save query  Discard changes


Common query patterns <https://github.com/microsoft/stream-analytics-query-language-elements-azure>

Query language - <https://docs.microsoft.com/stream-analytics-query/query-language-elements-azure>

\*/

```
6 SELECT
7     device AS Device, (( SUM(goodCount) / (SUM(goodCount) + SUM(badCount))) * 100) AS Kpi
8 INTO
9     [production-kpi]
10 FROM
11     [zajecia-iot-2023]
12 WHERE
13     device IS NOT NULL
14 GROUP BY
15     device, TumblingWindow(minute, 5)
```

Input preview Test results Job simulation (preview)

 Download results

Device	Kpi
<i>string</i>	<i>float</i>
"Device1"	90.08931143762604
"Device2"	82.54211332312404

- **telemetry** – przechowuje wiadomości C2D

Test query Save query Discard changes

Query language - <https://docs.microsoft.com/stream-analytics-query/query-language-azure-stream>

```

4  */
5
6  SELECT
7      *
8  INTO
9      [telemetry]
10 FROM
11     [zajecia-iot-2023]
12

```

Input preview Test results Job simulation (preview)

Download results

device string	productionStatus bigint	workorderId string	goodCount bigint	badCount bigint	temperature float
"Device1"	1	"b36bdf9e9-a663-4d95...	95	10	63.107
"Device2"	1	"d498d9e2-bcbd-48f8...	16	3	64.586
"Device1"	1	"b36bdf9e9-a663-4d95...	97	10	63.728

- **temperature-avg-min-max** – przechowuje średnią, minimalną i maksymalną temperaturę na urządzenie z podziałem 5 minutowym

Test query Save query Discard changes

```

6  SELECT
7      device, AVG(temperature) AS temperatureAvg, MIN(temperature) AS temperatureMin,
8      MAX(temperature) AS temperatureMax
9  INTO
10     [temperature-avg-min-max]
11 FROM
12     [zajecia-iot-2023]
13 WHERE
14     device IS NOT NULL
15 GROUP BY
16     device, TumblingWindow(minute, 5);

```

## Business Logic

Niestety w mojej aplikacji „**Business logic**” nie jest obsługiwane w chmurze zatem funkcjonalność nie jest taka sama jak w kryteriach projektu. Dane logiki pobierane są z kontenerów „**device-errors**”, „**production-kpi**”, „**telemetry**”, „**temperature-avg-min-max**” i pobierane do folderu „**Blob**”. Pliki należy pobrać z kontenerów „**device-errors**” i „**production-kpi**”. W menu głównym należy wybrać opcję 4 „**Download selected blob**” i na podstawie kalkulacji z podanych wcześniej kontenerów program

zdecyduje, czy należy uruchomić metody „***EmergencyStop***” lub „***ReduceProductionRate***”.