

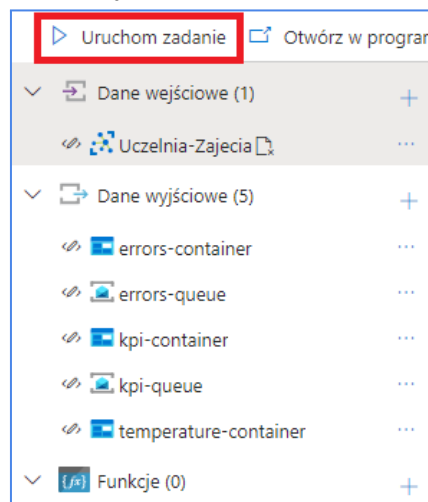
# Dokumentacja projektowa IoT

## Eliza Wielocha

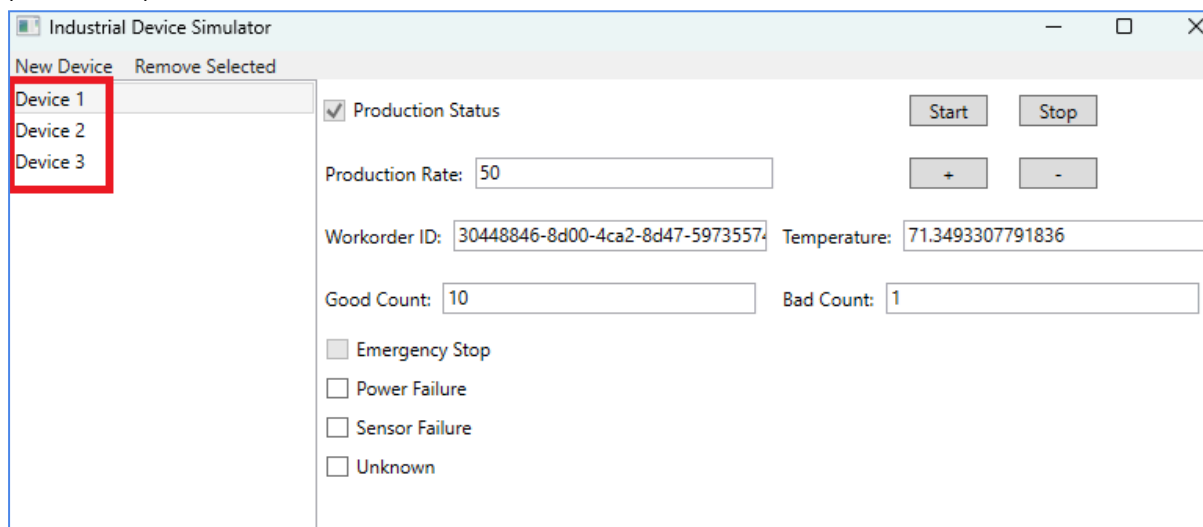
## 1. Połączenie z urządzeniem

### Uruchomienie aplikacji

Zanim uruchomiony zostanie projekt, należy najpierw w *Azure* na stronie zapytania dla Zadania usługi *Stream Analytics* wystartować zadanie:



Następnie należy uruchomić symulator i dodać do niego dowolną liczbę urządzeń (*device'ów*):



Dla każdego *device'a* należy ustawić *Production Rate* (przycisk '+' i '-') i je uruchomić za pomocą przycisku *Start*.

Projekt został wykonany w formie aplikacji konsolowej. Uruchamiany jest on poprzez uruchomienie pliku *Projekt/IOT.exe* znajdującego się w folderze dla projektu. Dokładna ścieżka do pliku *Projekt/IOT.exe* z folderu projektowego to *Projekt/IOT\bin\Debug\net6.0\*.

Kiedy aplikacja zostanie uruchomiona pokaże się okno terminala. Program od razu łączy się z serwerem i zaczyna odczytywać z niego parametry dla poszczególnych urządzeń.

## Połączenie z serwerem OPC UA

Łączenie się z symulatorem odbywa się za pomocą biblioteki *Opc.UaFx.Client*.

---

```
using (var client = new OpcClient(ConfigurationManager.AppSettings.Get("OPCclient")))
client.Connect();
```

---

Tutaj *OPCclient* jest kluczem z pliku konfiguracyjnego a jego wartość równa się *opc.tcp://localhost:4840/*. Dzięki klauzuli *client.connect()* aplikacja łączy się z serwerem *OPC UA*.

## Odczyt/zapis danych oraz wywoływanie węzłów-metod

Po połączeniu się z serwerem program tworzy listę urządzeń, które znajdują się na serwerze. Przechodzi po kolei po węzłach-dzieciach dla danego węzła i jeśli natrafi na węzeł rozpoczynający się od słowa "Device", zapisuje pełną nazwę tego węzła do listy urządzeń, która później jest używana w programie.

---

```
// get the list of devices in simulation
public async Task Browse(OpcNodeInfo node, List<String> Devices, int level = 0)
{
    level++;
    foreach (var childNode in node.Children())
    {
        if (childNode.DisplayName.Value.Contains("Device "))
        {
            Devices.Add(childNode.DisplayName.Value);
        }
        Browse(childNode, Devices, level);
    }
}
```

---

Implementacja została napisana na podstawie dokumentacji [Client Development Guide & Tutorial - TRAEGER Docs](#).

Mając listę urządzeń, program przechodzi po urządzeniach z tej listy nieskończoną ilość razy. W każdym przejściu dla urządzenia odczytywane są wartości jego parametrów i zapisywane w jeden obiekt data:

---

```
string pre = "ns=2;s=" + deviceName + "/";
var data = new
{
    DeviceName = deviceName,
    ProductionStatus = client.ReadNode(pre + "ProductionStatus").Value,
    ProductionRate = client.ReadNode(pre + "ProductionRate").Value,
    WorkorderId = client.ReadNode(pre + "WorkorderId").Value,
    Temperature = client.ReadNode(pre + "Temperature").Value,
    GoodCount = client.ReadNode(pre + "GoodCount").Value,
    BadCount = client.ReadNode(pre + "BadCount").Value,
    DeviceError = client.ReadNode(pre + "DeviceError").Value
};
```

---

Jedno przejście listy urządzeń, tj. odczyt i zapis danych dla każdego z urządzeń na serwerze, odbywa się co 2 sekundy.

## 2. Konfiguracja agenta

Do projektu dodany został plik konfiguracyjny o nazwie *App.config* w formacie *.xml*. Użyto do tego biblioteki *System.Configuration*. Plik zawiera klucze i ich wartości odpowiadające różnym parametrom potrzebnym do komunikacji agenta z serwerem i *IoTHubem*. Prezentuje się następująco:

---

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="deviceConnectionString"
value="HostName=Uczelnia-Zajecia.azure-devices.net;DeviceId=Device_test;SharedAccessKey=NpE3SJirdSphmNeEzyU5ZNIGh6hG0tn3oAIoTGnPtco=" />
    <add key="OPCclient" value="opc.tcp://localhost:4840/" />
    <add key="registryManager"
value="HostName=Uczelnia-Zajecia.azure-devices.net;SharedAccessKeyName=iiothubowner;SharedAccessKey=gEkKZ4eq2jMwDecCXFbtAqfjh49wQRhDAIoTFmWzsQ=" />
    <add key="sbConnectionString"
value="Endpoint=sb://servicebusproject.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=Uvfl+IFASe542tBJI3lNRFw0jJRe/qUnl+ASbPabUYE=" />
    <add key="IoTHubName" value="Device_test" />
  </appSettings>
</configuration>
```

---

Pobieranie z pliku konfiguracyjnego odpowiedniej wartości odbywa się poprzez wywołanie wartości klucza z *AppSettings*:

---

```
var deviceConnectionString =
ConfigurationManager.AppSettings.Get("deviceConnectionString");
```

---

## 3. Rodzaj, format i częstotliwość wiadomości (D2C) wysyłanych przez agenta do IoT Hub

Po odczycie i zapisie danych do obiektu *data*, wiadomość wysyłana jest do *IoTHub* w formacie *.json*. Jeśli wartość dla parametru *DeviceError* zmieniła się od poprzedniego odczytu parametrów dla danego urządzenia, to włączany jest dodatkowo event *errorAlert*. Poniżej znajduje się implementacja wysyłania wiadomości:

---

```
var dataString = JsonConvert.SerializeObject(Data);
Microsoft.Azure.Devices.Client.Message eventMessage = new
Microsoft.Azure.Devices.Client.Message(Encoding.UTF8.GetBytes(dataString));

eventMessage.Properties.Add("errorAlert", (!sameData && Data.DeviceError != 0) ?
"true" : "false");
eventMessage.ContentType = MediaTypeNames.Application.Json;
eventMessage.ContentEncoding = "utf-8";

await client.SendEventAsync(eventMessage);
```

---

Można wyróżnić 3 rodzaje wiadomości, które agent wysyła do *IoTHub*:

1. Wartość parametru *DeviceError* nie zmieniła się od poprzedniego wczytania parametrów. Wtedy wysyłana jest wiadomość nie zawierająca tego parametru wraz z alertem ustawionym na *false*:

```
Wed May 15 2024 22:21:19 GMT+0200 (czas środkowoeuropejski letni):

{
  "body": {
    "DeviceName": "Device 2",
    "ProductionStatus": 1,
    "WorkorderId": "0eefdfd3-52e7-4d7f-8042-0268f3dcc3b5",
    "GoodCount": 451,
    "BadCount": 31,
    "Temperature": 68.58267400596796,
    "ProductionRate": 30
  },
  "enqueuedTime": "Wed May 15 2024 22:21:19 GMT+0200 (czas środkowoeuropejski letni)",
  "properties": {
    "errorAlert": "false"
  }
}
```

2. Wartość parametru *DeviceError* zmieniła się od poprzedniego wczytania parametrów i jest ona różna od 0. Wtedy wysyłana jest wiadomość zawierająca wartość tego parametru wraz z alertem ustawionym na *true*:

```
Wed May 15 2024 22:21:58 GMT+0200 (czas środkowoeuropejski letni):

{
  "body": {
    "DeviceName": "Device 2",
    "ProductionStatus": 1,
    "WorkorderId": "0eefdfd3-52e7-4d7f-8042-0268f3dcc3b5",
    "GoodCount": 506,
    "BadCount": 35,
    "Temperature": 227,
    "ProductionRate": 30,
    "DeviceError": 4
  },
  "enqueuedTime": "Wed May 15 2024 22:21:58 GMT+0200 (czas środkowoeuropejski letni)",
  "properties": {
    "errorAlert": "true"
  }
}
```

3. Wartość parametru *DeviceError* zmieniła się od poprzedniego wczytania parametrów, ale jest ona równa 0. Wtedy wysyłana jest wiadomość zawierająca wartość tego parametru (czyli 0), natomiast alert ustawiony jest na *false*:

```
Wed May 15 2024 22:22:23 GMT+0200 (czas środkowoeuropejski letni):

{
  "body": {
    "DeviceName": "Device 2",
    "ProductionStatus": 1,
    "WorkorderId": "0eefdfd3-52e7-4d7f-8042-0268f3dcc3b5",
    "GoodCount": 541,
    "BadCount": 38,
    "Temperature": 68.16000484416814,
    "ProductionRate": 30,
    "DeviceError": 0
  },
  "enqueuedTime": "Wed May 15 2024 22:22:23 GMT+0200 (czas środkowoeuropejski letni)",
  "properties": {
    "errorAlert": "false"
  }
}
```

## 4. Rodzaj i format danych przechowywanych w *Device Twin*

W *Device Twin* przechowywane są dane o błędach i wskaźnikach produkcji w urządzeniach symulowanych jako:

- *Device[nr]\_errors*
- *Device[nr]\_production\_rate*

Podczas działania agenta aktualizują się lub tworzą właściwości reprezentujące powyższe dane w *Device Twin*. W *Reported Properties* są to zarówno *ProductionRate* jak i *Errors*, które na bieżąco zmieniają swoje wartości. Natomiast w *Desired Properties* jest tylko *ProductionRate*, który zmienia się w 2 przypadkach:

- 1) Podczas działania programu można ręcznie umieścić nową właściwość w *Desired Properties* dla *ProductionRate* dla konkretnego urządzenia w symulacji, np.  
`"Device1_production_rate": 50`, i ją zapisać
- 2) Gdy program wykryje więcej niż 3 błędy (tj. wartości dla *DeviceError* różne od 0) dla urządzenia w ciągu 1 minuty to automatycznie zapisuje w *Desired Properties* wartość dotychczasowego *ProductionRate* obniżone o 10.

Warto dodać, że na początku działania agenta usuwane są z *Reported Properties* wszelkie właściwości pozostałe z poprzedniego jego uruchomienia. W przypadku *Desired Properties* - należy usunąć stare właściwości ręcznie podając wartość *null* przy właściwości i zapisać *device twin* zanim jeszcze uruchomi się nową instancję agenta.

Przykładowa zawartość *device twin*:

---

```
"properties": {
  "desired": {
    "Device3_production_rate": 50,
    "Device2_production_rate": 70,
    "Device1_production_rate": 70,
    "$metadata": {
      "$lastUpdated": "2024-05-16T17:49:34.4654712Z",
      "$lastUpdatedVersion": 76,
      "Device3_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:34.4654712Z",
        "$lastUpdatedVersion": 76
      },
      "Device2_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:34.4654712Z",
        "$lastUpdatedVersion": 76
      },
      "Device1_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:34.4654712Z",
        "$lastUpdatedVersion": 76
      }
    },
    "$version": 76
  },
  "reported": {
    "DateTimeLastAppLaunch": "2024-04-03T19:41:57.2027555+02:00",
    "DateTimeLastDesiredPropertyChangeReceived":
"2024-04-03T20:12:13.2970386+02:00",
    "Device1_errors": 2,
    "Device1_production_rate": 70,
    "Device2_errors": 0,
```

```

    "Device2_production_rate": 70,
    "Device3_errors": 12,
    "Device3_production_rate": 50,
    "$metadata": {
      "$lastUpdated": "2024-05-16T17:49:37.8780194Z",
      "DateTimeLastAppLaunch": {
        "$lastUpdated": "2024-04-03T17:41:55.7066538Z"
      },
      "DateTimeLastDesiredPropertyChangeReceived": {
        "$lastUpdated": "2024-04-03T18:12:11.7151366Z"
      },
      "Device1_errors": {
        "$lastUpdated": "2024-05-16T17:49:06.4026938Z"
      },
      "Device1_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:36.8936729Z"
      },
      "Device2_errors": {
        "$lastUpdated": "2024-05-16T17:48:14.3237426Z"
      },
      "Device2_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:37.8780194Z"
      },
      "Device3_errors": {
        "$lastUpdated": "2024-05-16T17:48:54.9758078Z"
      },
      "Device3_production_rate": {
        "$lastUpdated": "2024-05-16T17:49:33.9792506Z"
      }
    },
    "$version": 2865
  },
}

```

---

## 5. Dokumentacja *Direct Methods* zaimplementowanych w agencie

Agent 'nasłuchuje' za pomocą metody *Handlers*, czy *Direct metody* zostały wywołane w *Azure IoT/IoT Explorer*:

```

public async Task InitializeHandlers()
{
    // wait for running direct methods
    await client.SetMethodHandlerAsync("EmergencyStop", EmergencyStop, client);
    await client.SetMethodHandlerAsync("ResetErrorStatus", ResetErrorStatus,
client);
}

```

---

W projekcie zostały zaimplementowane 2 *Direct Metody*:

- 1) **EmergencyStop** - Metoda, która na wejściu przyjmuje nazwę urządzenia i wywołuje opcję zatrzymania danego urządzenia. Jej implementacja znajduje się poniżej:

```
public async Task<MethodResponse> EmergencyStop(MethodRequest methodRequest,
object userContext)
{
    var payload =
    JsonConvert.DeserializeAnonymousType(methodRequest.DataAsJson, new {
        deviceName = default(string) });
    if (payload != null)
    {
        Console.WriteLine("Emergency stop for " + payload.deviceName);

        await Task.Run(() => OPC.CallMethod("ns=2;s=" + payload.deviceName,
"ns=2;s=" + payload.deviceName + "/EmergencyStop"));
    }
    else
    {
        Console.WriteLine("Emergency stop for " + payload.deviceName +
"done!");
    }
    return new MethodResponse(0);
}
```

Po jej wywołaniu przez *IoT Explorer* dla konkretnego urządzenia w symulacji, następuje zatrzymanie awaryjne tego urządzenia a opcja *EmergencyStop* jest zaznaczona. Aby wywołać tą metodę należy wykonać kroki od 1 do 3 znajdując się w zakładce *Direct Method* w *IoT Explorer*:

1. Wpisać w pole *Method name* nazwę metody: *EmergencyStop*
2. Wpisać w pole *Payload* nazwę urządzenia w formacie *.json*: *{"DeviceName": "Device [nr]"}*
3. Kliknąć opcję *Invoke method*

Home > Uczelnia-Zajecia > Devices > Device\_test > Direct method

Invoke method

**Direct method** You can send direct methods to a device. Direct methods have a name, payload, and configurable connection and method timeouts.

Method name \*  
EmergencyStop

Payload ⓘ  
{"DeviceName": "Device 2"}

Jeśli metoda zostanie wywołana poprawnie, wyświetli się komunikat w prawym górnym rogu ze statusem 0. Wynik w symulatorze wygląda następująco:

The screenshot shows the 'New Device' configuration window in the IoT Explorer simulator. On the left, a list of devices includes 'Device 1', 'Device 2' (selected), and 'Device 3'. The main configuration area for 'Device 2' includes: 'Production Status' (unchecked), 'Start' and 'Stop' buttons, 'Production Rate' (60), '+' and '-' buttons, 'Workorder ID' (9c319823-04d7-4fdc-8d76-c0f16203), 'Temperature' (24.792449169980838), 'Good Count' (75), 'Bad Count' (7), and a list of error types: 'Emergency Stop' (checked and highlighted with a red box), 'Power Failure' (unchecked), 'Sensor Failure' (unchecked), and 'Unknown' (unchecked).

Urządzenie zatrzymało się.

- 2) **ResetErrorStatus** - Metoda, która przyjmuje na wejściu nazwę urządzenia i wywołuje opcję zresetowania wszystkich błędów na danym urządzeniu. Jej implementacja znajduje się poniżej:

```
public async Task<MethodResponse> ResetErrorStatus(MethodRequest
methodRequest, object userContext)
{
    var payload =
    JsonConvert.DeserializeAnonymousType(methodRequest.DataAsJson, new {
deviceName = default(string) });
    if (payload != null)
    {
        Console.WriteLine("Reset error status for " + payload.deviceName);

        await Task.Run(() => OPC.CallMethod("ns=2;s=" + payload.deviceName,
"ns=2;s=" + payload.deviceName + "/ResetErrorStatus"));
    }
    else
    {
        Console.WriteLine("Reset error status for " + payload.deviceName +
"done!");
    }
    return new MethodResponse(0);
}
```

Po jej wywołaniu przez *IoT Explorer* dla konkretnego urządzenia w symulacji, następuje zresetowanie błędów na tym urządzeniu, więc wszystkie 4 opcje błędów:

- *Emergency Stop*
- *Power Failure*
- *Sensor Failure*
- *Unknown*

zostają odznaczone .



Poniżej znajduje się przykład działającego urządzenia 1, na którym pojawiły się błędy:

New Device Remove Selected

Device 1  
Device 2  
Device 3

☒ Production Status Start Stop

Production Rate: 50 + -

Workorder ID: bdd994ea-c880-4a90-8624-90f8bfb1 Temperature: -370

Good Count: 28 Bad Count: 4

☐ Emergency Stop  
☒ Power Failure  
☒ Sensor Failure  
☒ Unknown

Aby wywołać metodę *ResetErrorStatus* należy wykonać kroki od 1 do 3 znajdując się w zakładce *Direct Method* w *IoT Explorer*:

1. Wpisać w pole "Method name" nazwę metody: *ResetErrorStatus*
2. Wpisać w pole "Payload" nazwę urządzenia w formacie *.json: {"DeviceName": "Device [nr]}"*
3. Kliknąć opcję "Invoke method"

Home > Uczelnia-Zajecia > Devices > Device\_test > Direct method

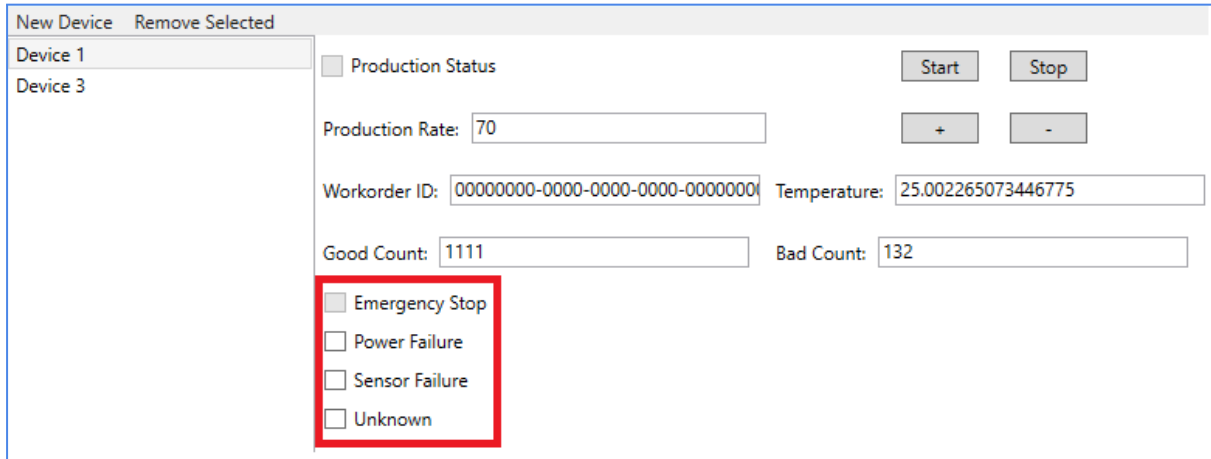
< Invoke method <sup>3</sup>

**Direct method** *You can send direct methods to a device. Direct methods have a name, payload, and configurable connection and method timeouts.*

Method name \* <sup>1</sup>  
ResetErrorStatus

Payload ⓘ <sup>2</sup>  
{"DeviceName": "Device 1"}

Jeśli metoda zostanie wywołana poprawnie, wyświetli się komunikat w prawym górnym rogu ze statusem 0. Wynik w symulatorze wygląda następująco:



Błędy zostały zresetowane.

Uwaga: W przypadku gdy jednym z aktywnych błędów na urządzeniu jest *Emergency Stop* i zostanie wywołany *ResetErrorStatus*, błąd ten zostanie odznaczony w symulatorze ale urządzenie będzie nadal wstrzymane aż do naciśnięcia przycisku start.

## 6. Kalkulacje i logika biznesowa

### Dostępne są kalkulacje:

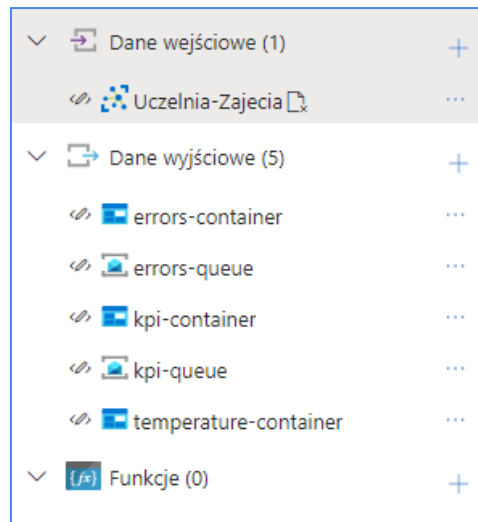
- Zapis *KPI* (% dobrej produkcji) do kontenera *kpi-container* w 5 minutowych oknach
- Zapis średniej, minimalnej i maksymalnej temperatury z ostatnich 5 minut do kontenera *temperature-container* co 1 minutę
- Zapis wystąpienia więcej niż 3 błędów w ciągu 1 minuty na urządzeniu do kontenera *errors-container*

### Dostępna jest logika biznesowa:

- Wywołanie *EmergencyStop* jeśli na urządzeniu występują więcej niż 3 błędy w ciągu 1 minuty
- Zmniejszenie wartości *Production Rate* o 10 punktów jeśli zapisywane *KPI* w 5 minutowych oknach jest mniejsze niż 90%

### Na potrzeby projektu korzystano z:

- *Konto Magazynu* - Utworzono 3 kontenery:
  - *kpi-container*
  - *temperature-container*
  - *errors-container*
- *ServiseBus* - Utworzono 2 kolejki:
  - *errors-queue*
  - *kpi-queue*
- *Azure Stream Analytics* - Dodano dane wejściowe w postaci *IoTHuba* oraz dane wyjściowe w postaci kontenerów i kolejek z *konta magazynu* i *serviceBusa*.



Poniżej znajduje się tabela przedstawiająca poszczególne dane wyjściowe w *Azure Stream Analytics*, co jest do nich zapisywane oraz implementacja zapytań:

<b>kpi-container</b>	<p>Kontener do którego zapisują się dane o KPI dla poszczególnych urządzeń w 5 minutowych oknach</p> <pre> SELECT     DeviceName,     round(100.0 * SUM(GoodCount) / SUM(GoodCount + BadCount), 2) AS KPI INTO     [kpi-container] FROM     [Uczelnia-Zajecia] GROUP BY     DeviceName, TumblingWindow(minute, 5) </pre>
<b>temperature-container</b>	<p>Kontener do którego co minutę zapisują się informacje o temperaturze z ostatnich 5 minut</p> <pre> SELECT     DeviceName,     AVG(Temperature) AS avg_Temperature,     MIN(Temperature) AS min_Temperature,     MAX(Temperature) AS max_Temperature INTO     [temperature-container] FROM     [Uczelnia-Zajecia] GROUP BY     DeviceName, HoppingWindow(minute, 5 , 1); </pre>
<b>errors-container</b>	<p>Kontener do którego zapisuje się liczba wystąpień błędów z danego urządzenia w ciągu minuty</p> <pre> SELECT     DeviceName, COUNT(*) AS Count INTO     [errors-container] FROM     [Uczelnia-Zajecia] WHERE </pre>

	<pre> DeviceError is not null and DeviceError != 0 GROUP BY DeviceName, SlidingWindow(minute, 1) HAVING COUNT(DeviceError) &gt; 3; </pre>
<b>errors-queue</b>	<p>Kolejka do której zapisują się zapisuje się liczba wystąpień błędów z danego urządzenia w ciągu minuty</p> <pre> SELECT     DeviceName, COUNT(*) AS Count INTO     [errors-queue] FROM     [Uczelnia-Zajecia] WHERE     DeviceError is not null and     DeviceError != 0 GROUP BY DeviceName, SlidingWindow(minute, 1) HAVING COUNT(DeviceError) &gt; 3; </pre>
<b>kpi-queue</b>	<p>Kolejka do której zapisuje się KPI o wartości poniżej 90% dla poszczególnych urządzeń w 5 minutowych oknach</p> <pre> SELECT     DeviceName,     round(100.0 * SUM(GoodCount)/SUM(GoodCount + BadCount), 2) AS KPI INTO     [kpi-queue] FROM     [Uczelnia-Zajecia] GROUP BY     DeviceName, TumblingWindow(minute,1) HAVING     KPI &lt; 90; </pre>

Kolejki *ServiceBusa* są procesowane przez agenta, który wykonuje odpowiednie kroki w zależności od której kolejki zostanie komunikat:

- Komunikat od *errors-queue*: Wywołanie *EmergencyStop* dla odpowiedniego urządzenia.
- Komunikat od *kpi-queue*: Obniżenie wartości *Production Rate* jako właściwości w *Desired Properties* w *Device Twin* o 10 punktów. W wyniku zmiany *Desired Production Rate*, zmieni się również *Reported Production Rate*, a co za tym idzie, wartość *Production Rate* w symulatorze.

### Przykładowe wyniki działania kalkulacji i logiki biznesowej:

*kpi-container*-----

```

{"DeviceName":"Device 2","KPI":88.79}
{"DeviceName":"Device 3","KPI":83.03}
{"DeviceName":"Device 1","KPI":88.8}
{"DeviceName":"Device 2","KPI":91.74}
{"DeviceName":"Device 3","KPI":89.62}
{"DeviceName":"Device 1","KPI":91.78}
{"DeviceName":"Device 3","KPI":88.26}

```

```
{"DeviceName":"Device 1","KPI":89.24}
{"DeviceName":"Device 2","KPI":90.52}
{"DeviceName":"Device 3","KPI":83.82}
```

#### ***temperature-container***-----

```
{"DeviceName":"Device
1","avg_Temperature":91.24077257937105,"min_Temperature":62.00473912714558,"m
ax_Temperature":112.71826689249485}
{"DeviceName":"Device
2","avg_Temperature":89.44042957840556,"min_Temperature":63.100191102535895,"
max_Temperature":122.38753893252567}
{"DeviceName":"Device
3","avg_Temperature":154.1539774903633,"min_Temperature":65.14535556669742,"m
ax_Temperature":546.0}
{"DeviceName":"Device
1","avg_Temperature":84.52162122704259,"min_Temperature":62.00473912714558,"m
ax_Temperature":112.71826689249485}
{"DeviceName":"Device
2","avg_Temperature":87.57990264141111,"min_Temperature":60.46962922851354,"m
ax_Temperature":122.38753893252567}
{"DeviceName":"Device
3","avg_Temperature":158.51019749442085,"min_Temperature":-758.0,"max_Tempera
ture":724.0}
{"DeviceName":"Device
3","avg_Temperature":158.51019749442085,"min_Temperature":-758.0,"max_Tempera
ture":724.0}
{"DeviceName":"Device
2","avg_Temperature":87.57990264141111,"min_Temperature":60.46962922851354,"m
ax_Temperature":122.38753893252567}
{"DeviceName":"Device
1","avg_Temperature":84.52162122704259,"min_Temperature":62.00473912714558,"m
ax_Temperature":112.71826689249485}
{"DeviceName":"Device
3","avg_Temperature":158.51019749442085,"min_Temperature":-758.0,"max_Tempera
ture":724.0}
{"DeviceName":"Device
2","avg_Temperature":87.57990264141111,"min_Temperature":60.46962922851354,"m
ax_Temperature":122.38753893252567}
```

#### ***errors-container***-----

```
{"DeviceName":"Device 6","Count":4}
{"DeviceName":"Device 6","Count":5}
{"DeviceName":"Device 6","Count":4}
{"DeviceName":"Device 1","Count":4}
{"DeviceName":"Device 1","Count":5}
{"DeviceName":"Device 1","Count":6}
{"DeviceName":"Device 1","Count":5}
{"DeviceName":"Device 1","Count":4}
```

### errors-queue-----

sequenceNumber	messageId	enqueuedTimeUtc	deliveryCo	state	subject	bodySize	body	sessionId
81	ed5e239a0a2d4b5690293ecc9d119982	16.05.2024	0	active		35 B	{"DeviceName":"Device 1","Count":4}	216dc
82	e9ae3c3cf8af452496246cff6c8d4b13	16.05.2024	0	active		35 B	{"DeviceName":"Device 1","Count":5}	f6a75
83	8aa20284dc03494997488c8b1b3ca620	16.05.2024	0	active		35 B	{"DeviceName":"Device 1","Count":6}	361f1
84	e5d01b5daacd4f77bca353aabb0ffd0a	16.05.2024	0	active		35 B	{"DeviceName":"Device 3","Count":4}	f9abf
85	837494fc6fe74de79b0108317f74baa9	16.05.2024	0	active		35 B	{"DeviceName":"Device 3","Count":5}	7326c
86	668e8f76696c4297b655c2290579f589	16.05.2024	0	active		35 B	{"DeviceName":"Device 3","Count":6}	1936c

Po przyjęciu jednego z komunikatów od *errors-queue* agent wywołuje *EmergencyStop* na urządzeniu.

### kpi-queue-----

sequenceNumber	messageId	enqueuedTimeUtc	deliveryCo	state	subject	bodySize	body	sessionId
101	7e15449058b64363a057cef5bb0defb5	16.05.2024	0	active		37 B	{"DeviceName":"Device 2","KPI":86.42}	67def
102	721fad4db7044036aeb63f87436db0d8	16.05.2024	0	active		37 B	{"DeviceName":"Device 3","KPI":88.06}	67def
103	f2a60648d19041bf9658e0b7b5d5d4c3	16.05.2024	0	active		37 B	{"DeviceName":"Device 1","KPI":83.77}	67def

Po przyjęciu jednego z komunikatów od *kpi-queue* agent obniża *Production Rate* dla wybranego urządzenia o 10 punktów i aktualizuje *Desired Property* dla *Production Rate*.