

DOKUMENTACJA DO PROJEKTU Z IOT

Wykonanie : Barbara Kosior

Spis treści

1. Repozytorium.....	3
2. Połączenie.....	3
2. Komunikacja D2C (device to cloud).....	5
3. Device Twin.....	8
4. Direct Methods – ResetErrorStatus.....	10
5. Direct Methods – EmergencyStop.....	12
6. Obliczenia.....	13
6.1. Production KPIs.....	13
6.2. Temperature.....	14
6.3. Device errors.....	15
7. Logika biznesowa.....	16
7.1. Emergency Stop.....	17
7.2. Production Decrease.....	18
7.3. Send an Email.....	21
7.4. Użyte usługi w Azure.....	21

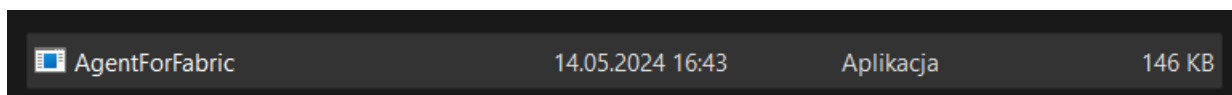
1. Repozytorium

Link do repozytorium :

https://github.com/barbara-k97/IOT_projekt_zaliczeniowy_v1

2. Połączenie

Aby uruchomić aplikację Agent, należy odpalić aplikację „AgentForFabric” z folderu „bin → Debug → net6.0” .



Gdy program się uruchomi zostaniemy poproszeni o wprowadzenie informacji, które będą potrzebne do prawidłowego działania aplikacji. Oto przykładowe prawidłowe uzupełnienie danych. Po podaniu każdej z danych należy nacisnąć Enter a aplikacja poprosi o podanie następnej z wartości. Po podaniu wszystkich danych konfiguracyjnych pojawi się komunikat o tym informujący.

```
D:\Uczelnia\3 Magisterka\ 4  X  +  v

Witaj w AgentForFabric !

-----
!!!      --- łącznie z Azure !
!!!      Wpisz string do połączenia z Azure :
HostName=hubZajecia.azure-devices.net;DeviceId=test_device;SharedAccessKey=x8bzG9iX+bK00aTd/e8X0eR67UI0ud5
!!!      łącznie z Azure zakończone sukcesem !
-----

!!!      Podaj sciezke URL do serwera OPC UA :
opc.tcp://localhost:4840/
-----

!!!      Podaj string do ServisBus:
Endpoint=sb://servicebusme.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAcc
=
-----

!!!      Podaj nazwę kolejki do obsługi produkcji:
kolejka-produkcja
-----

!!!      Podaj nazwę kolejki do obsługi błędów:
kolejka-3errors
-----

!!!      Podaj String do Registry Manager :
HostName=hubZajecia.azure-devices.net;SharedAccessKeyName=iouthubowner;SharedAccessKey=EKjQ0KY0BK0WdNaXFfXy
-----

----- DZIĘKUJĘ ZA PODANIE WSZYSTKICH DANYCH KONFIGURACYJNYCH -----
OPC UA łącznie zakończone sukcesem !
```

Zebrane dane są zapisywane i wykorzystywane do połączenia z Azure Iot Hub, OPC UA, Azure Service Bus.

```
Console.WriteLine("                Witaj w AgentForFabric ! ");
Console.WriteLine("-----");
Console.WriteLine(" !!!          ---  łączenie z Azure !");
// String do połączenia z Azure wpisane
Console.WriteLine("!!!!          Wpisz string do połączenia z Azure : ");
string deviceConnectionString = Console.ReadLine() ?? string.Empty;
using var deviceClient = DeviceClient.CreateFromConnectionString(deviceConnectionString, Microsoft.Azure.Devices.
await deviceClient.OpenAsync();
Console.WriteLine(" !!!          łączenie z Azure zakończone sukcesem !");
Console.WriteLine("-----");

// ŁĄCZENIE Z OPC UA
// prośba o podanie ścieżki URL do serwera OPC UA
// opc.tcp://localhost:4840/
Console.WriteLine("!!!!          Podaj sciezke URL do serwera OPC UA : ");
string adresServerOPC = Console.ReadLine() ?? string.Empty;
Console.WriteLine("-----");

// Ścieżka do Servicebus
Console.WriteLine("!!!!          Podaj string do ServiceBus: ");
string sbConnectionString = Console.ReadLine() ?? string.Empty;
Console.WriteLine("-----");
// const string sbConnectionString = "Endpoint=sb://servicebusme.servicebus.windows.net/;SharedAccessKeyName=Root

Console.WriteLine("!!!!          Podaj nazwę kolejki do obsługi produkcji: ");
string queueName2 = Console.ReadLine() ?? string.Empty;
// const string queueName2 = "kolejka-produkcja";
Console.WriteLine("-----");
Console.WriteLine("!!!!          Podaj nazwę kolejki do obsługi błędów: ");
string queueName = Console.ReadLine() ?? string.Empty;

using (var client = new OpcClient(adresServerOPC))
{
    client.Connect();
    Console.WriteLine("OPC UA łączenie zakończone sukcesem !");
    Console.WriteLine();
    Console.WriteLine();
    Console.WriteLine();

    var node = client.BrowseNode(OpcObjectTypes.ObjectsFolder);
    List<String> devicesList = ReadDeviceFromSimulator(node);

    using var registryManager = RegistryManager.CreateFromConnectionString(registryString);

    var device = new Class1(deviceClient, client , registryManager);
    await device.InitializeHandlers();

    // SERVISBUS wywołanie
    await using ServiceBusClient client_servisbus = new ServiceBusClient(sbConnectionString);
    await using ServiceBusProcessor processor = client_servisbus.CreateProcessor(queueName);
    processor.ProcessMessageAsync += device.Processor_ProcessMessageAsync;
    processor.ProcessErrorAsync += device.Processor_ProcessErrorAsync;
    await using ServiceBusProcessor processor2 = client_servisbus.CreateProcessor(queueName2);
    processor2.ProcessMessageAsync += device.Processor_ProcessMessageAsync2;
    processor2.ProcessErrorAsync += device.Processor_ProcessErrorAsync2;
}
```

Gdy prawidłowo połączymy się to na konsoli zaczną wyświetlać się dane zebrane z urządzeń z serwera OPC. Na początku wypiszą się nam nazwy wszystkich urządzeń jakie zostały odnalezione, a następnie odczytane wartości.

```
##### Device:Device 1
```

Przykład dla Device 1:

```
*****
DEVICE Device 1
ProductionStatus
1
ProductionRate
30
WorkorderId
026dfe38-de04-4ccb-a57b-fab1daaf0f4a
Temperature
72,88145241790812
GoodCount
130
BadCount
20
DeviceError
0
*****
```

2. Komunikacja D2C (device to cloud)

Aby odczytać dane stworzona została lista z nazwami urządzeń które są w symulatorze. Dzięki niej znamy liczbę urządzeń oraz ich nazwy. Wykorzystane to zostało do tego aby na podstawie wczytanych nazw stworzyć listy węzłów OPC dla każdego urządzenia i zaczytać ich wartość. Pobrane dane przekazywane są do metody SendTelemetry znajdującej się w Class1. Dane przesyłane są do platformy IoT za pomocą komunikatu D2C. Dane wysyłane są co 10 sekund.

Tworzenie listy z istniejącymi Device (liniami produkcyjnymi) :

```
// LISTA DO POBRANIA NAZW DEVICE W SYMULATORZE I ICH LISTY
Odwolania: 2
static List<String> ReadDeviceFromSimulator(OpcNodeInfo node, int numberDevice = 0)
{
    List<String> deviceNames = new List<String>();
    numberDevice++;
    //Console.WriteLine(" ! Lista urzadzen:" );
    foreach (var childNode in node.Children())
    {
        if (childNode.DisplayName.Value.Contains("Device "))
        {
            Console.WriteLine("##### Device:" + childNode.DisplayName.Value);
            deviceNames.Add(childNode.DisplayName.Value);

            ReadDeviceFromSimulator(childNode, numberDevice);
        }
    }
    return deviceNames;
}

var node = client.BrowseNode(OpcObjectTypes.ObjectsFolder);
List<String> devicesList = ReadDeviceFromSimulator(node);
```

Odczytywanie danych z urządzeń :

```
// Tworzenie listy węzłów OPC na podstawie wczytanych nazw urządzeń
foreach (string deviceName in devicesList)
{
    OpcValue name = deviceName;
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/ProductionStatus", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/ProductionStatus"));
    OpcValue ProductionS = client.ReadNode("ns=2;s=" + deviceName + "/ProductionStatus");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/ProductionRate", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/ProductionRate"));
    OpcValue ProductionRate = client.ReadNode("ns=2;s=" + deviceName + "/ProductionRate");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/WorkorderId", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/WorkorderId"));
    OpcValue WorkorderId = client.ReadNode("ns=2;s=" + deviceName + "/WorkorderId");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/Temperature", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/Temperature"));
    OpcValue Temperature = client.ReadNode("ns=2;s=" + deviceName + "/Temperature");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/GoodCount", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/GoodCount"));
    OpcValue GoodCount = client.ReadNode("ns=2;s=" + deviceName + "/GoodCount");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/BadCount", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/BadCount"));
    OpcValue BadCount = client.ReadNode("ns=2;s=" + deviceName + "/BadCount");
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/DeviceError", OpcAttribute.DisplayName));
    commands.Add(new OpcReadNode("ns=2;s=" + deviceName + "/DeviceError"));
    OpcValue DeviceErrors = client.ReadNode("ns=2;s=" + deviceName + "/DeviceError");
}
```

Zebrane dane wysyłane są do metody SendTelemetry :

```
await device.SendTelemetry(deviceName, WorkorderId.Value, ProductionS.Value, Temperature.Value, ProductionRate.Value,
    GoodCount.Value, BadCount.Value, DeviceErrors.Value);
```

SendTelemetry :

```

#region D2C - Sending telemetry
1 odwołanie
public async Task SendTelemetry(string DeviceName, object WorkorderId, object ProductionStatus, object Temperature, object ProductionRate,
    object GoodCount, object BadCount, object DeviceErrors)
{
    var twin = await client.GetTwinAsync();
    var reportedProperties = twin.Properties.Reported;
    var nameDevice = DeviceName.Replace(" ", "");
    var device_error = nameDevice + "_numer_bledu";
    var errorStatus = DeviceErrors;
    bool DataNoChange = false;
    //DeviceError wysylamy tylko gdy sie zmini

    if (reportedProperties.Contains(device_error))
    {
        var currentError = reportedProperties[device_error];
        DataNoChange = (currentError == errorStatus);
    }
    if (DataNoChange)
    {
        // błąd się nei zmienił wiec nie wysyłamy wartosci error
        var selectedData = new
        {
            DeviceName = DeviceName,
            WorkorderId = WorkorderId,
            ProductionStatus = ProductionStatus,
            Temperature = Temperature,
            ProductionRate = ProductionRate,
            GoodCount = GoodCount,
            BadCount = BadCount,
            DeviceErrors = DeviceErrors,
        };
        await SendMessageToIoT(selectedData);
        Console.WriteLine(selectedData);
    }
    else
    {
        var selectedData = new
        {
            DeviceName = DeviceName,
            WorkorderId = WorkorderId,
            ProductionStatus = ProductionStatus,
            Temperature = Temperature,
            ProductionRate = ProductionRate,
            GoodCount = GoodCount,
            BadCount = BadCount,
            DeviceErrors = DeviceErrors,
        };
        // W przypadku zmiany wartość należy wysłać pojedynczy komunikat D2C do platformy IoT ( punkt 2.7)
        Console.WriteLine(selectedData);
        await SendMessageToIoT(selectedData);
    }
}

await UpdateTwinAsync(nameDevice, errorStatus, ProductionRate);

```

W `SendTelemetry()` odbierane są wartości z urządzenia oraz przygotowywane do `SendMessageToIOT()`. Dane są sprawdzane, jeśli zmienił się stan wartości "DeviceErrors", czyli zmienił się stan błędu to wysyłana jest wiadomość zawierająca nową wartość błędu. W przeciwnym wypadku nie zostaje ta dana (wartość błędu) wysyłana.

Przykład :

Wiadomość gdy na urządzeniu nie zarejestrowano błędu :

```
-----
{ DeviceName = Device 1, WorkorderId = 9a20f74d-f0ca-4aa1-aa72-32bd5f9665cd, ProductionStatus = 1, Temperature = 60,37991044052
8356, ProductionRate = 30, GoodCount = 80, BadCount = 10 }
Brak zmiany błędu - nie wykonano zmian
Brak zmiany błędu - nie wykonano zmian
-----

{ DeviceName = Device 2, WorkorderId = d8cc42b0-9a1a-4cd0-b666-709597ce0085, ProductionStatus = 1, Temperature = 68,67701117318
937, ProductionRate = 60, GoodCount = 158, BadCount = 12 }
Brak zmiany błędu - nie wykonano zmian
Brak zmiany błędu - nie wykonano zmian
-----
```

Fri May 10 2024 19:43:14 GMT+0200 (czas środkowoeuropejski letni):

```
{
  "body": {
    "DeviceName": "Device 2",
    "WorkorderId": "d8cc42b0-9a1a-4cd0-b666-709597ce0085",
    "ProductionStatus": 1,
    "Temperature": 68.67701117318937,
    "ProductionRate": 60,
    "GoodCount": 158,
    "BadCount": 12
  },
  "enqueuedTime": "Fri May 10 2024 19:43:14 GMT+0200 (czas środkowoeuropejski letni)"
}
```

Fri May 10 2024 19:43:13 GMT+0200 (czas środkowoeuropejski letni):

```
{
  "body": {
    "DeviceName": "Device 1",
    "WorkorderId": "9a20f74d-f0ca-4aa1-aa72-32bd5f9665cd",
    "ProductionStatus": 1,
    "Temperature": 60.379910440528356,
    "ProductionRate": 30,
    "GoodCount": 80,
    "BadCount": 10
  },
  "enqueuedTime": "Fri May 10 2024 19:43:13 GMT+0200 (czas środkowoeuropejski letni)"
}
```

Wiadomość gdy wykryto błąd :

```
-----
{ DeviceName = Device 1, WorkorderId = 9a20f74d-f0ca-4aa1-aa72-32bd5f9665cd, ProductionStatus = 1, Temperature = -27,
ProductionRate = 30, GoodCount = 489, BadCount = 53, DeviceErrors = 4 }
Zaktualizowano liczbę błędów dla :
10.05.2024 19:50:47> Device Twin was update.
-----

{ DeviceName = Device 2, WorkorderId = d8cc42b0-9a1a-4cd0-b666-709597ce0085, ProductionStatus = 1, Temperature = 669,
ProductionRate = 60, GoodCount = 1027, BadCount = 110, DeviceErrors = 6 }
Zaktualizowano liczbę błędów dla :
10.05.2024 19:50:47> Device Twin was update.
```



```
{
  "body": {
    "DeviceName": "Device 2",
    "WorkorderId": "d8cc42b0-9a1a-4cd0-b666-709597ce0085",
    "ProductionStatus": 1,
    "Temperature": 669,
    "ProductionRate": 60,
    "GoodCount": 1027,
    "BadCount": 110,
    "DeviceErrors": 6
  },
  "enqueuedTime": "Fri May 10 2024 19:50:46 GMT+0200 (czas środkowoeuropejski letni)"
}
```

Fri May 10 2024 19:50:45 GMT+0200 (czas środkowoeuropejski letni):

```
{
  "body": {
    "DeviceName": "Device 1",
    "WorkorderId": "9a20f74d-f0ca-4aa1-aa72-32bd5f9665cd",
    "ProductionStatus": 1,
    "Temperature": -27,
    "ProductionRate": 30,
    "GoodCount": 489,
    "BadCount": 53,
    "DeviceErrors": 4
  },
  "enqueuedTime": "Fri May 10 2024 19:50:45 GMT+0200 (czas środkowoeuropejski letni)"
}
```

3. Device Twin

W Device Twin możemy raportować jaki jest aktualny stan błędów oraz szybkości produkcji a także zlecać maszynie zmianę prędkości produkcji. Do Device Twin zgłaszane są informacje dotyczące szybkości produkcji (production rate) i błędów występujących na maszynie. Dla każdej maszyny wartości zapisywane są w osobnych liniach podpisanych numerem urządzenia i rodzajem przechowywanej wartości np. "Device1_numer_bledu": 2".

Przykładowa zawartość zawartość Device twin :


```

"properties": {
  "desired": {
    "restartCount": "true",
    "test_device": 641475592,
    "Device1_production_procent": 30,
    "$metadata": {
      "$lastUpdated": "2024-05-16T20:30:34.0253603Z",
      "$lastUpdatedVersion": 29,
      "restartCount": {
        "$lastUpdated": "2024-05-16T20:30:34.0253603Z",
        "$lastUpdatedVersion": 29
      },
      "test_device": {
        "$lastUpdated": "2024-05-16T20:30:34.0253603Z",
        "$lastUpdatedVersion": 29
      },
      "Device1_production_procent": {
        "$lastUpdated": "2024-05-16T20:30:34.0253603Z",
        "$lastUpdatedVersion": 29
      }
    },
    "$version": 29
  },
  "reported": {
    "DateTimeLastAppLaunch": "2024-04-08T20:54:47.8807559+02:00",
    "DateTimeLastDesiredPropertyChangeReceived": "2024-04-08T20:55:46.818316+02:00",
    "DeviceErrors": 20,
    "LastErrorDate": "2024-05-05T11:04:27.7278945+02:00",
    "ProductionRate": 0,
    "Device1_numer_bledu": 0,
    "Device1_production_procent": 40,
    "Device2_production_procent": 0,
    "Device2_numer_bledu": 0,
    "Device3_production_procent": 100,
    "Device3_numer_bledu": 10,
  }
}

```

Gdy zostanie zmieniona prędkość produkcji użytkownik zostanie o tym poinformowany i wyświetli się stosowna informacja.

```

{ DeviceName = Device 1, WorkorderId = d655bb6b-7cda-443f-a02e-260da41ad1d4, ProductionStatus = 1, Temperature = 80,27496280794159, ProductionRate = 80, GoodCount = 396, BadCount = 39 }
Zaktualowano % produkcji dla :
14.05.2024 17:07:02> Device Twin was update.

```

Również w przypadku zmiany błędu, na konsoli pojawi się informacja :

```

{ DeviceName = Device 1, WorkorderId = d655bb6b-7cda-443f-a02e-260da41ad1d4, ProductionStatus = 1, Temperature = 67,87637777413241, ProductionRate = 80, GoodCount = 519, BadCount = 51, DeviceErrors = 2 }
Zaktualowano liczbe bledow dla :
14.05.2024 17:07:34> Device Twin was update.

```

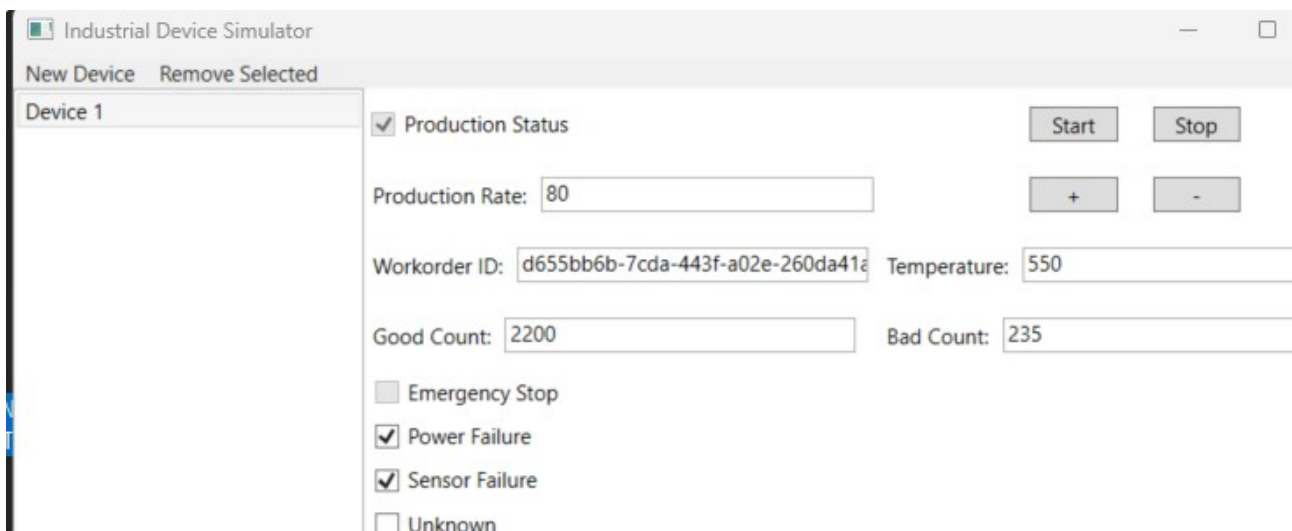
4. Direct Methods – ResetErrorStatus

Wywołanie metody `ResetErrorStatus` sprawia że zresetowany zostanie stan błędu występujący na wskazanej przez użytkownika maszynie. Metoda ta może zostać wywołana w Azure IOT Explorer. Jako wymagany parametr należy podać nazwę maszyny, na której ma ona zostać uruchomiona. Poniżej pokazany jest przykład zapytania oraz efekt (maszyna przed i po wykonaniu metody).

Jako nazwa podajemy : `ResetErrorStatus`

A jako parametr np. : `{"deviceName": "Device 1"}`

Maszyna przed wywołaniem :



Industrial Device Simulator

New Device Remove Selected

Device 1

☒ Production Status Start Stop

Production Rate: 80 + -

Workorder ID: d655bb6b-7cda-443f-a02e-260da41e Temperature: 550

Good Count: 2200 Bad Count: 235

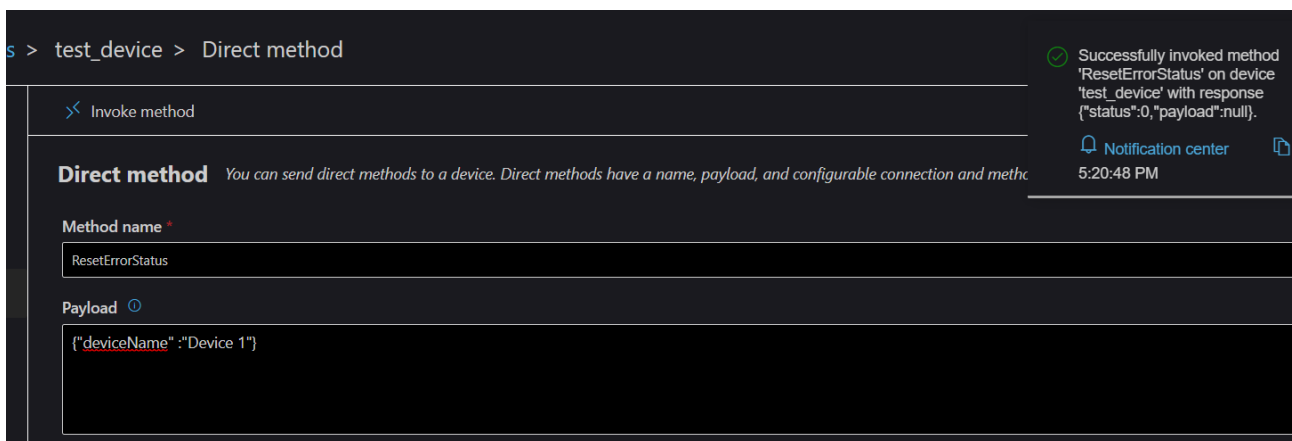
☐ Emergency Stop

☒ Power Failure

☒ Sensor Failure

☐ Unknown

Wywołanie metody:



s > test_device > Direct method

>> Invoke method

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

Method name *

ResetErrorStatus

Payload ⓘ

{ "deviceName": "Device 1" }

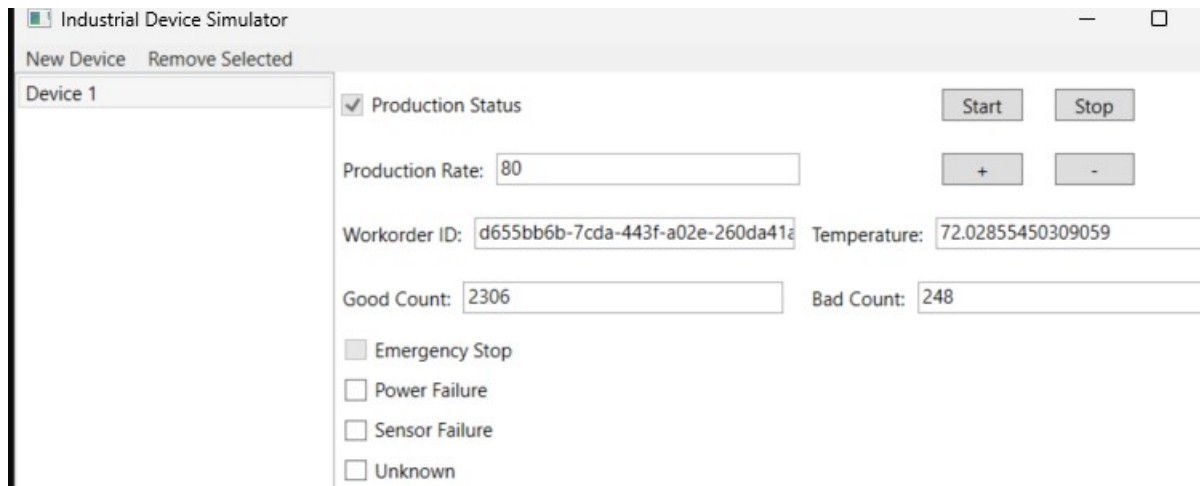
Successfully invoked method 'ResetErrorStatus' on device 'test_device' with response {'status': 0, 'payload': null}.

Notification center 5:20:48 PM

Informacja na konsoli o działaniu :

```
*****  
METHOD EXECUTED ResetErrorStatus FROM : Device 1
```

Efekt :



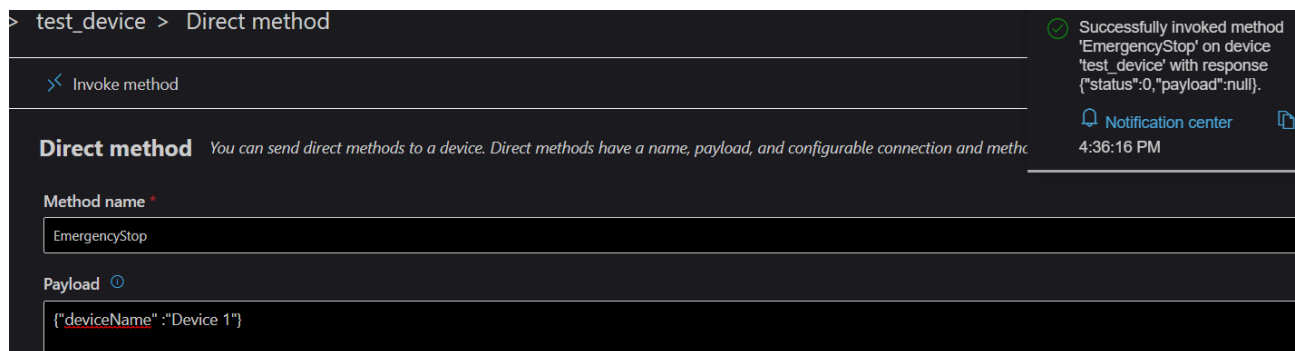
```
#region Direct Methods - ResetErrorStatus  
1 odwołanie  
public async Task ResetError(string deviceName)  
{  
    Console.WriteLine($"\\t    METHOD EXECUTED ResetErrorStatus FROM : {deviceName}");  
    OPC.CallMethod($"ns=2;s={deviceName}", $"ns=2;s={deviceName}/ResetErrorStatus");  
    await Task.Delay(1000);  
}  
  
1 odwołanie  
private async Task<MethodResponse> ResetErrorStatus(MethodRequest methodRequest, object userContext)  
{  
    var payload = JsonConvert.DeserializeAnonymousType(methodRequest.DataAsJson, new {deviceName = default(string)});  
    // Console.WriteLine($"\\t    METHOD EXECUTED: {methodRequest.Name} na {payload.deviceName}");  
    await ResetError(payload.deviceName);  
    return new MethodResponse(0);  
}  
#endregion
```

5. Direct Methods – EmergencyStop

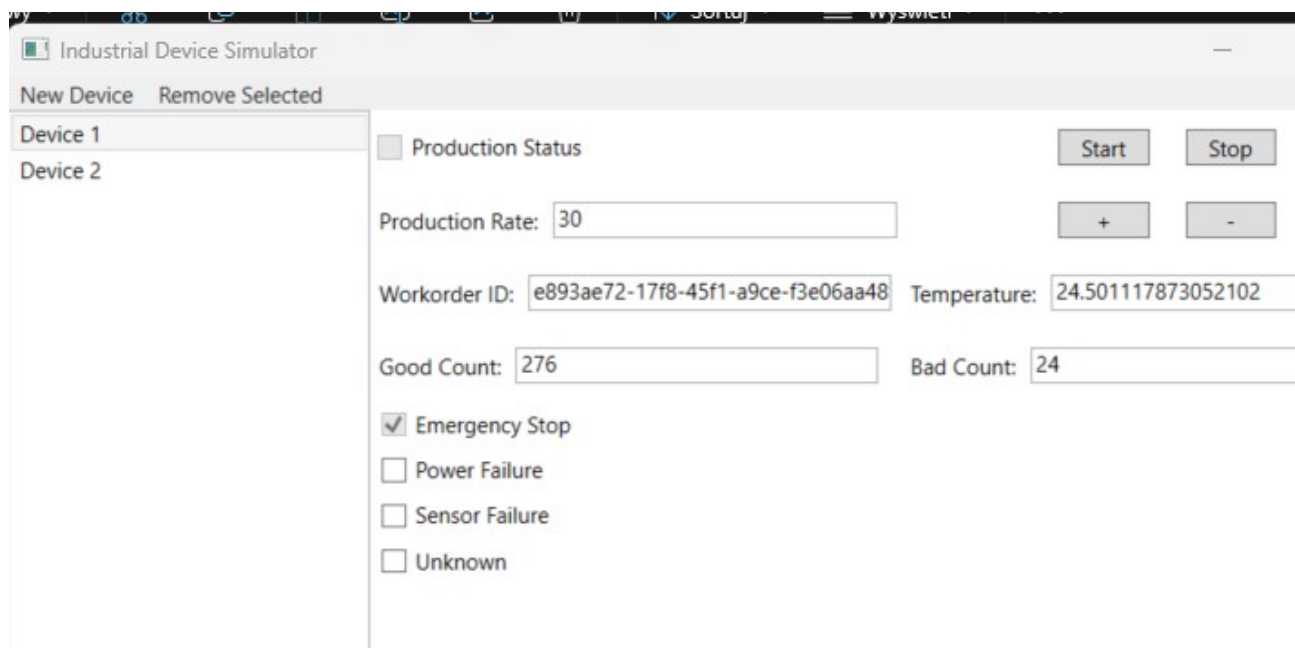
Wywołanie metody EmergencyStop sprawia że na wskazanej przez użytkownika maszynie uruchomione zostanie zatrzymanie awaryjne. Metoda ta może zostać wywołana w Azure IOT Explorer. Jako wymagany parametr należy podać nazwę maszyny, na której ma ona zostać uruchomiona.

Jako nazwa podajemy : EmergencyStop

A jako parametr np. : {"deviceName": "Device 1"}



Efekt :



Użytkownik zostanie poinformowany że metoda została wywołana :

```
*****  
METHOD EXECUTED Emergency Stop FROM : Device 1
```

```
#region Direct Methods - Emergency Stop  
1 odwołanie  
public async Task EmergencyStopStatus(string deviceName)  
{  
    Console.WriteLine($"{deviceName} METHOD EXECUTED Emergency Stop FROM : {deviceName}");  
    OPC.CallMethod($"ns=2;s={deviceName}", $"ns=2;s={deviceName}/EmergencyStop");  
    await Task.Delay(1000);  
}  
  
1 odwołanie  
private async Task<MethodResponse> EmergencyStop(MethodRequest methodRequest, object userContext)  
{  
    var payload = JsonConvert.DeserializeObjectAnonymousType(methodRequest.DataAsJson, new { deviceName = default(string) });  
    // Console.WriteLine($"{methodRequest.Name} na {payload.deviceName}");  
    await EmergencyStopStatus(payload.deviceName);  
    return new MethodResponse(0);  
}  
  
#endregion
```

6. Obliczenia

Do tej części dokumentacji dołączony został dodatkowy folder w projekcie o nazwie „Analytic” zawierający w sobie plik tekstowy „polecenia_SQL.txt” oraz foldery z poprawnymi plikami blob z przykładami działania zapytań. W pliku tekstowym umieszczono zapytania, które wykorzystano w Azure Analytics. Danymi wejściowymi do wszystkich zapytań jest IoT Hub, do którego przesyłane są dane z agenta OPC. Dane wyjściowe dla pierwszych trzech zapytań to kontenery o indywidualnych nazwach, a dla dwóch ostatnich zapytań danymi wyjściowymi jest kolejka Service Bus. Dwa ostatnie zapytania zostaną zaprezentowane i opisane w części dokumentacji dotyczącej logiki biznesowej.

6.1. Production KPIs

Procent dobrej produkcji w całkowitej objętości, pogrupowane według urządzenia w 5-minutowych oknach. Zapytanie :

```

1  /*
2  1. Production KPIs
3  Procent dobrej produkcji w 5 min odstępach pogrupowane wedle urządzenia
4  */
5  SELECT
6  DeviceName , System.Timestamp() as windowEndTime , SUM(GoodCount)*100/(SUM(GoodCount) + SUM(BadCount))
7  AS "% of good production"
8  INTO
9  [production]
10 FROM
11 [hubZajecia]
12 GROUP BY TumblingWindow(minute, 5 ) , DeviceName ;
13

```

Dane zapisywane są w kontenerze.

A to przykład otrzymanych danych z przeprowadzonych obliczeń :

0_6a88bdb7159547d097e18a0a8777205f_1.json ...

Blob

Przegląd Wersje Migawki Edytuj Generuj sygnaturę dostępu współdzielonego

```

1  [{"DeviceName": "Device 2", "windowEndTime": "2024-05-13T11:15:00.000000Z", "% of good production": 90.00470366886171}]
2  [{"DeviceName": "Device 1", "windowEndTime": "2024-05-13T11:15:00.000000Z", "% of good production": 89.754336942601}]
3  [{"DeviceName": "Device 2", "windowEndTime": "2024-05-13T11:20:00.000000Z", "% of good production": 89.8164513936098}]
4  [{"DeviceName": "Device 1", "windowEndTime": "2024-05-13T11:20:00.000000Z", "% of good production": 90.42455711667685}]

```

6.2. Temperature

Co 1 minutę podawaj mi średnią, minimalną i maksymalną temperaturę z ostatnich 5 minut (pogrupowane według urządzenia). Zapytanie :

```

/*
2. Temperatura
średnia, minimalna, maksymalna temperatura z ostatnich 5 min ,pogrupowana wedle urządzenia, co 1 minutę ,
*/
SELECT
DeviceName , System.Timestamp() as windowEndTime , MAX(Temperature) as max , MIN(Temperature) as min,
AVG(Temperature) as avg
INTO
[temperature]
FROM
[hubZajecia]
GROUP BY HoppingWindow(minute,5,1), DeviceName ;

```

Dane zapisywane są w kontenerze.

A to przykład otrzymanych danych z przeprowadzonych obliczeń :

0_47c6261d5cb4414580bf6e24a0f3097e_1.json ...

Blob

Zapisz Odrzuć Pobierz Odśwież Usuń

Przegląd Wersje Migawki Edytuj Generuj sygnaturę dostępu współdzielonego

```
1 [{"DeviceName": "Device 1", "windowEndTime": "2024-05-13T10:33:00.000000Z", "max": 65.96279440432882, "min": 24.4142220402, "avg": 45.18850820216541}
2 [{"DeviceName": "Device 2", "windowEndTime": "2024-05-13T10:33:00.000000Z", "max": 673.0, "min": -549.0, "avg": 52.22827672440, "liczba_bledow": 4}
3 [{"DeviceName": "Device 1", "windowEndTime": "2024-05-13T10:34:00.000000Z", "max": 65.96279440432882, "min": 24.1070525372017, "avg": 45.03492347076526}
4 [{"DeviceName": "Device 2", "windowEndTime": "2024-05-13T10:34:00.000000Z", "max": 871.0, "min": -549.0, "avg": 139.31473357950, "liczba_bledow": 4}
5 [{"DeviceName": "Device 1", "windowEndTime": "2024-05-13T10:35:00.000000Z", "max": 65.96279440432882, "min": 24.1070525372017, "avg": 45.03492347076526}
6 [{"DeviceName": "Device 2", "windowEndTime": "2024-05-13T10:35:00.000000Z", "max": 871.0, "min": -877.0, "avg": -3.678291512816, "liczba_bledow": 4}
```

Co minutę, dla każdego urządzenia zapisywane są wyniki.

6.3. Device errors

Sytuacje, w których na urządzeniu wystąpią więcej niż 3 błędy w czasie krótszym niż 1 minuta. W poleceniu zastosowane zostało zabezpieczenie, by nie zliczało pozycji gdy wartość DeviceError wynosi 0, ponieważ nie jest to błąd a jedynie informacja o zmienionym stanie błędów. Zapytanie :

```
/*
3. Błędy
Informacje o sytuacjach gdy na urządzeniu w ciągu minuty wystąpią więcej niż 3 błędy.
*/
SELECT
    System.Timestamp() as windowEndTime , DeviceName , COUNT(*) as liczba_bledow
INTO
    [deviceerrors]
FROM
    [hubZajecia]
WHERE
    DeviceErrors IS NOT null and DeviceErrors != 0
GROUP BY SlidingWindow(minute,1) , DeviceName , liczba
HAVING COUNT(DeviceErrors)>3 ;
```

Dane zapisywane są w kontenerze.

A to przykład otrzymanych danych z przeprowadzonych obliczeń :

0_758dfb73339c49aea3885df9b79486bc_1.json ...

Blob

Zapisz Odrzuć Pobierz Odśwież Usuń

Przegląd Wersje Migawki Edytuj Generuj sygnaturę dostępu współdzielonego

```
1 [{"windowEndTime": "2024-05-14T15:37:39.830000Z", "DeviceName": "Device 1", "liczba_bledow": 4}
2 [{"windowEndTime": "2024-05-14T15:37:40.643000Z", "DeviceName": "Device 2", "liczba_bledow": 4}
```


7. Logika biznesowa

Do obsługi logiki biznesowej wykorzystano kolejki usługi Service Bus. Podczas uruchomienia agenta użytkownik proszony zostaje o podanie adresu do połączenia z Azure Service Bus oraz nazw kolejek, które będą używane.

```
// Ścieżka do Servisbus
Console.WriteLine("!!!      Podaj string do ServisBus: ");
string sbConnectionString = Console.ReadLine() ?? string.Empty;
Console.WriteLine("-----");
// const string sbConnectionString = "Endpoint=sb://servicebusme.servicebus.windows.net/;Shar

Console.WriteLine("!!!      Podaj nazwę kolejki do obsługi produkcji: ");
string queueName2 = Console.ReadLine() ?? string.Empty;
// const string queueName2 = "kolejka-produkcja";
Console.WriteLine("-----");
Console.WriteLine("!!!      Podaj nazwę kolejki do obsługi błędów: ");
string queueName = Console.ReadLine() ?? string.Empty;
// const string queueName = "kolejka-3errors";
Console.WriteLine("-----");

Console.WriteLine("!!!      Podaj String do Registry Manager : ");
string registryString = Console.ReadLine() ?? string.Empty;

// SERVISBUS wywołanie
await using ServiceBusClient client_servisbus = new ServiceBusClient(sbConnectionString);
await using ServiceBusProcessor processor = client_servisbus.CreateProcessor(queueName);
processor.ProcessMessageAsync += device.Processor_ProcessMessageAsync;
processor.ProcessErrorAsync += device.Processor_ProcessErrorAsync;
await using ServiceBusProcessor processor2 = client_servisbus.CreateProcessor(queueName2);
processor2.ProcessMessageAsync += device.Processor_ProcessMessageAsync2;
processor2.ProcessErrorAsync += device.Processor_ProcessErrorAsync2;
```


Do obsługi logiki biznesowej utworzono dwie kolejki.


[Strona główna](#) > [serviceBusMe](#)


 **serviceBusMe | Kolejki** ★ ...
Przestrzeń nazw usługi Service Bus

<<


[+ Kolejka](#) [🔄 Odśwież](#) [🗨️ Przekaż opinię](#)

 Przegląd

 Dziennik aktywności

 Kontrola dostępu (IAM)

 Tagi

 Diagnostowanie i rozwiązywanie problemów

Nazwa ↑↓	Stan ↑↓	Liczba komunikat... 1
kolejka-3errors	Active	0
kolejka-produkcja	Active	1

7.1. Emergency Stop

W Azure Stream Analytics zostało utworzone zapytanie, które z danych dostarczonych z IOT Hub przekazuje do kolejki Service Bus, tutaj o nazwie „kolejka-3errors”, informacje o sytuacjach gdy na urządzeniu wystąpią więcej niż 3 błędy w ciągu 1 minuty. Gdy tak się stanie to na urządzeniu uruchamiane jest zatrzymanie awaryjne (Emergency Stop).

```
/*
4. Błędy do logiki biznesowej
Informacje o sytuacjach gdy na urządzeniu w ciągu minuty wystąpią więcej niż 3 błędy.
*/
SELECT
    System.Timestamp() as windowEndTime , DeviceName , COUNT(*) as liczba_bledow
INTO
    [kolejka-3errors]
FROM
    [hubZajecia]
WHERE
    DeviceErrors IS NOT null and DeviceErrors != 0
GROUP BY SlidingWindow(minute,1) , DeviceName , liczba
HAVING COUNT(DeviceErrors)>3 ;
```

Gdy dostaniemy informację o otrzymanej wiadomości najpierw odczytujemy wiadomość a następnie z odebranej wiadomości wyłuskujemy informacje, o którym Device była ta wiadomość. Następnie na podstawie tej informacji wywołana zostaje metoda EmergencyStop dla zadanej maszyny. W międzyczasie na konsoli zostaną wyświetlone informacje, które znajdowały się w wiadomości.

```
#region Logika Biznesowa - ERRORS

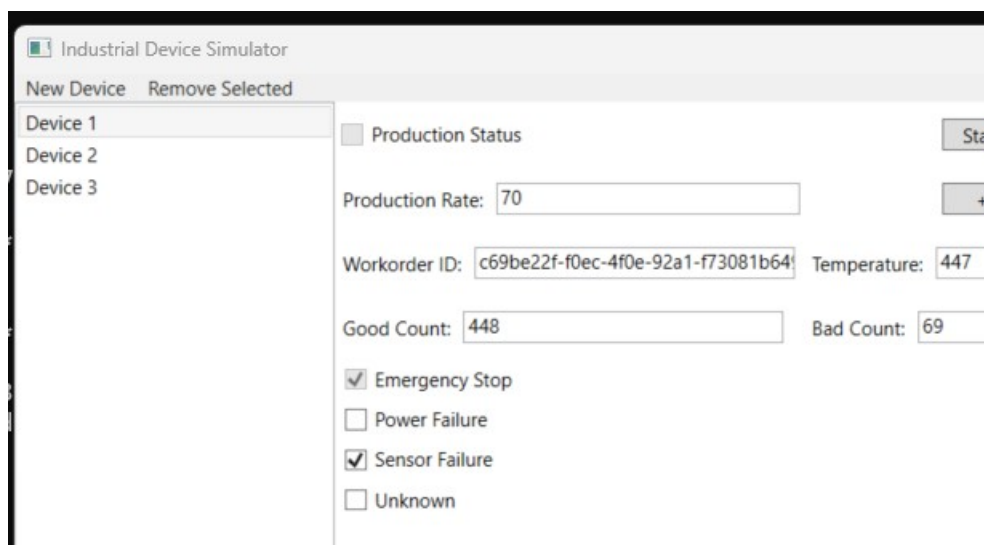
1 odwołanie
public async Task Processor_ProcessMessageAsync(ProcessMessageEventArgs arg)
{
    Console.WriteLine($"RECEIVED MESSAGE:\n\t{arg.Message.Body}");
    var message = Encoding.UTF8.GetString(arg.Message.Body);
    ReadMessage mesg = JsonConvert.DeserializeObject<ReadMessage>(message);

    string deviceId = mesg.DeviceName;
    Console.WriteLine("! ----- Zgłoszono wywołanie metody EmergencyStop  ");
    Console.WriteLine(mesg.windowEndTime);
    Console.WriteLine(mesg.DeviceName);
    Console.WriteLine(mesg.error);
    OPC.CallMethod($"ns=2;s={deviceId}", $"ns=2;s={deviceId}/EmergencyStop");
    Console.WriteLine("! ----- ");
}

1 odwołanie
public Task Processor_ProcessErrorAsync(ProcessErrorEventArgs arg)
{
    Console.WriteLine(arg.Exception.ToString());
    return Task.CompletedTask;
}

#endregion
```

Działanie:



```
RECEIVED MESSAGE:
      {"windowEndTime":"2024-05-15T12:33:49.941000Z","DeviceName":"Device 1","liczba_bledow":4}
! ----- Zgłoszono wywołanie metody EmergencyStop
15.05.2024 12:33:49
Device 1
! -----
```

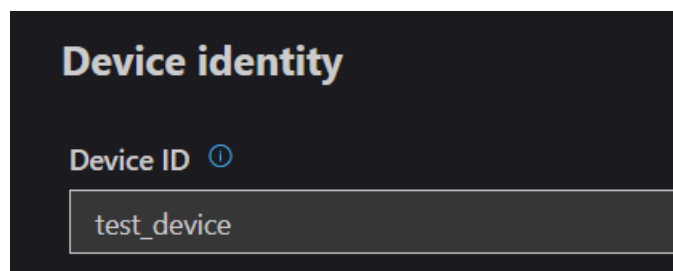
7.2. Production Decrease

Na podstawie zaprezentowanego poniżej zapytania SQL w Azure Stream Analytics do kolejki Service Bus o nazwie „kolejka-produkcja” przekazywane są informacje gdy procent dobrej produkcji na danym urządzeniu spadnie poniżej 90%.

```
/*
5. Produkcja do logiki biznesowej
jak produkcja dobra ejest mniejsza niz 90%
*/

SELECT
    DeviceName ,    System.Timestamp() as windowEndTime , SUM(GoodCount)*100/(SUM(GoodCount) + SUM(BadCount))
    AS "productionDevice"
INTO
    [kolejka-produkcja]
FROM
    [hubZajecia]
GROUP BY TumblingWindow(minute, 5 ) , DeviceName
HAVING
    SUM(GoodCount)*100/(SUM(GoodCount) + SUM(BadCount)) < 90;
```

W przypadku takiej sytuacji na konsoli pokaże się prośba o podanie Device ID, które można sprawdzić w tym miejscu :



#region Logika Biznesowa - ProductionDecrease

1 odwołanie

```
public async Task Processor_ProcessMessageAsync2(ProcessMessageEventArgs arg )
{
    Console.WriteLine("*****");
    Console.WriteLine("!!!      Podaj Device ID : ");
    string nazwaIOHub = Console.ReadLine() ?? string.Empty;

    //string nazwaIOHub = "test_device";
    Console.WriteLine("*****");

    Console.WriteLine($"RECEIVED MESSAGE:\n\t{arg.Message.Body}");
    var message = Encoding.UTF8.GetString(arg.Message.Body);
    ReadMessage mesg = JsonConvert.DeserializeObject<ReadMessage>(message);
    Console.WriteLine($"! _____Zgłoszono wywołanie metody ChangeProduction " );
    Console.WriteLine(mesg.windowEndTime);
    Console.WriteLine(mesg.DeviceName);
    Console.WriteLine(mesg.productionDevice);

    string deviceId = mesg.DeviceName;

    await ChangeProduction(nazwaIOHub , deviceId);
    Console.WriteLine($"! _____");
}
```

1 odwołanie

```
public Task Processor_ProcessErrorAsync2(ProcessErrorEventArgs arg)
{
    Console.WriteLine(arg.Exception.ToString());
    return Task.CompletedTask;
}
```

#endregion

Następnie wywołane zostanie ChangeProduction() dla wskazanego urządzenia. Z reported pobrana zostanie obecna wartość szybkości produkcji , wartość jej zostanie obniżona o 10%, a następnie wysłana do desired.

Przykład działania poniżej. Pierwszy screen to przed spadkiem produkcji poniżej 90 % :

```

*****
DEVICE Device 2
ProductionStatus
1
ProductionRate
100
WorkorderId
31d9a006-9038-470b-b035-2f06b203e291
Temperature
147,77209171700963
GoodCount
903
BadCount
97
DeviceError
0
*****

```

Na konsoli użytkownik proszony jest o podanie nazwy, pokazana zostanie otrzymana wiadomość. Produkcja spadła poniżej 90%, dlatego została zmniejszona produkcja (w tym przypadku z 100 na 90).

```

*****
!!! Podaj Device ID :
test_device
*****
RECEIVED MESSAGE:
{ "DeviceName": "Device 2", "windowEndTime": "2024-05-15T12:35:00.000000Z", "productionDevice": "89.91034390472367" }
! _____Zgłoszono wywołanie metody ChangeProduction
15.05.2024 12:35:00
Device 2
89.91034390472367
!
*****
{ DeviceName = Device 2, WorkorderId = 31d9a006-9038-470b-b035-2f06b203e291, ProductionStatus = 1, Temperature = 128,1036302581902, ProductionRate = 90, GoodCount = 998, BadCount = 108 }
Zaktualizowano % produkcji dla :
15.05.2024 14:35:28> Device Twin was update.
*****

```

Wpis w Device Twin, z nowym tempem produkcji :

```

version : 303,
"properties": {
  "desired": {
    "restartCount": "true",
    "test_device": 641475592,
    "Device2_production_procent": 90,
    "Device1_production_procent": 70,
    "$metadata": {

```

Potwierdzenie o zmianie na maszynie :

```

*****
DEVICE Device 2
ProductionStatus
1
ProductionRate
90
WorkorderId
31d9a006-9038-470b-b035-2f06b203e291
Temperature
135,721253311889
GoodCount
1051
BadCount
115
DeviceError
0
*****

```







The screenshot shows the 'Industrial Device Simulator' window. On the left, a list of devices includes 'Device 1', 'Device 2' (which is selected), and 'Device 3'. On the right, the configuration for the selected device is displayed. It includes a checked 'Production Status' checkbox, a 'Production Rate' input field set to '90', a 'Workorder ID' input field containing '31d9a006-9038-470b-b035-2f06b203e291', and a 'Good Count' input field set to '1915'. There is also a partially visible 'Bad Count' field.

7.3. Send an Email

W projekcie nie udało się zaimplementować wysyłania wiadomości e-mail w sytuacji gdy wystąpił błąd urządzenia.

7.4. Użyte usługi w Azure

Poniżej zaprezentowany został screen z Azure z listą wykorzystywanych usług.

<input type="checkbox"/>	Nazwa ↑		Typ
<input type="checkbox"/>	 zajeciaIoT	...	Grupa zasobów
<input type="checkbox"/>	 hubZajecia	...	IoT Hub
<input type="checkbox"/>	 AnalyticForProject	...	Zadanie usługi Stream Analytics
<input type="checkbox"/>	 Azure for Students	...	Subskrypcja
<input type="checkbox"/>	 storageOf	...	Konto magazynu
<input type="checkbox"/>	 serviceBusMe	...	Przestrzeń nazw usługi Service B...

W pracy wykorzystane zostały 3 kontenery oraz 2 kolejki.

Strona główna > AnalyticForProject | Zapytanie > storageOf
 **storageOf** | Kontenery ⚙️ ★ ...
Konto magazynu

Przegląd

Dziennik aktywności

Tagi

Diagnozowanie i rozwiązywanie problemów

Kontrola dostępu (Zarządzanie dostępem i tożsamościami)

Migracja danych

Wyszukaj kontenery według prefiksu

Nazwa
<input type="checkbox"/> \$logs
<input type="checkbox"/> deviceerrors
<input type="checkbox"/> production
<input type="checkbox"/> temperature

Strona główna > serviceBusMe

 **serviceBusMe** | Kolejki ★ ...
Przestrzeń nazw usługi Service Bus

Kolejka


Odśwież

Przekaż opinię


Wyszukaj, aby filtrować elementy według na

Nazwa ↑↓	Stai
kolejka-3errors	Act
kolejka-produkcja	Act

Rozpocznij Właściwości Monitorowanie Samouczki

 **Dane wejściowe (1)**
hubZajecia

<> **Zapytanie**
Edytuj zapytanie

 **Dane wyjściowe (5)**
deviceerrors
kolejka-3errors
kolejka-produkcja
production
temperature

