IoT Projekt

Michał Poliwczak

402673

Repozytorium :

https://github.com/Michalo02/IOT-Projekt

Agent:

```
using IotHubDevice;
using Microsoft.Azure.Devices.Client;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Agent
{
    Odwołania: 3
    public class Program
    {
        static string[] Links = File.ReadAllLines("Logins.txt");
        static string OpcConnectionString = Links[1];
        static string DeviceConnectionString = Links[3];

        public static DateTime maintenanceDate = DateTime.MinValue;

        Odwołania: 0
        static async Task Main(string[] arguments)
        {
            using var deviceClient = DeviceClient.CreateFromConnectionString(DeviceConnectionString, TransportType.Mqtt);
            await deviceClient.OpenAsync();
            var device = new IoTDevice(deviceClient);

            await device.InitializeHandlers();

            DeviceOPC opcDevice = new DeviceOPC();

            DeviceOPC.StartConnection();

            var timer = new PeriodicTimer(TimeSpan.FromSeconds(1));
            while (await timer.WaitForNextTickAsync())
            {
                Console.WriteLine(DateTime.Now);
                await IoTDevice.SendTelemetry(DeviceOPC.client);
            }

            DeviceOPC.EndConnection();
            Console.ReadLine();
        }
```

Agent łączy się z Hubem przy użyciu kluczy znajdujących się w pliku „Logins.txt”. Dzięki funkcji InitializeHandlers obsługiwane są wywoływania do DirectMetods i RecevinigMessages.

Działanie agenta:

```
Connection Succes
15.02.2023 02:59:28
1
82f61759-aa34-4426-9a79-0485d0e609a8
6
1
63,384688554918306
        15.02.2023 02:59:29> Data: [{"device":1,"productionStatus":1,"workorderId":"82f61759-aa34-4426-9a79-0485d0e609a8
","goodCount":6,"badCount":1,"temperature":63.384688554918306}]
Device errors =
Production rate = 10
1
791a09a9-ae7b-4f89-99b2-69ff2936620e
8
1
62,2478131808749
        15.02.2023 02:59:30> Data: [{"device":2,"productionStatus":1,"workorderId":"791a09a9-ae7b-4f89-99b2-69ff2936620e
","goodCount":8,"badCount":1,"temperature":62.2478131808749}]
Device errors =
Production rate = 20
```

Telemetria:

```
#region Sending telemetry
1 odwołanie
public static async Task SendTelemetry(OpcClient opcClient)
{
    var node = opcClient.BrowseNode(OpcObjectTypes.ObjectsFolder);

    if (node.Children().Count() > 1)
    {
        foreach (var childNode in node.Children())
        {
            if (!childNode.DisplayName.Value.Contains("Server"))
            {
                var device = Convert.ToInt32(childNode.DisplayName.Value.Split(" ")[1]);

                var productionStatus = opcClient.ReadNode($"ns=2;s=Device {device}/ProductionStatus").Value;
                Console.WriteLine(productionStatus);
                var workorderId = opcClient.ReadNode($"ns=2;s=Device {device}/WorkorderId").Value;
                Console.WriteLine(workorderId);
                var goodCount = opcClient.ReadNode($"ns=2;s=Device {device}/GoodCount").Value;
                Console.WriteLine(goodCount);
                var badCount = opcClient.ReadNode($"ns=2;s=Device {device}/BadCount").Value;
                Console.WriteLine(badCount);
                var temperature = opcClient.ReadNode($"ns=2;s=Device {device}/Temperature").Value;
                Console.WriteLine(temperature);


                var telemetryData = new
                {
                    device = device,
                    productionStatus = productionStatus,
                    workorderId = workorderId,
                    goodCount = goodCount,
                    badCount = badCount,
                    temperature = temperature,

                };

                await SendTelemetryMessage(telemetryData, client);

                var deviceErrors = opcClient.ReadNode($"ns=2;s=Device {device}/deviceErrors").Value;
                Console.WriteLine($"Device errors = {deviceErrors}");
                var productionRate = opcClient.ReadNode($"ns=2;s=Device {device}/ProductionRate").Value;
                Console.WriteLine($"Production rate = {productionRate}");

                await TwinAsync(deviceErrors,  productionRate);
            }
        }
    }
```

```
1 odwołanie
public static async Task SendTelemetryMessage(dynamic telemetryData, DeviceClient client)
{
    var dataString = JsonConvert.SerializeObject(telemetryData);

    Microsoft.Azure.Devices.Client.Message eventMessage = new Microsoft.Azure.Devices.Client.Message(Encoding.UTF8.GetBytes(dataString));
    eventMessage.ContentType = MediaTypeNames.Application.Json;
    eventMessage.ContentEncoding = "utf-8";
    Console.WriteLine($"\t{DateTime.Now.ToLocalTime()}> Data: [{dataString}]");

    await client.SendEventAsync(eventMessage);
}
```

Telemetria jest wysyłana co sekundę do Huba, której wiadomości można podejrzeć w Azure albo w IoT Explorerze

Przykładowe wiadomości:

Receiving events...

Wed Feb 15 2023 04:02:21 GMT+0100 (czas środkowoeuropejski standardowy):


```
{
  "body": {
    "device": 1,
    "productionStatus": 1,
    "workorderId": "2545d3be-ec1e-4b50-a997-c6d08e5c8bfc",
    "goodCount": 34,
    "badCount": 4,
    "temperature": 65.11228835388349
  },
  "enqueuedTime": "Wed Feb 15 2023 04:02:21 GMT+0100 (czas środkowoeuropejski standardowy)",
  "properties": {}
}
```


Wed Feb 15 2023 04:02:19 GMT+0100 (czas środkowoeuropejski standardowy):


```
{
  "body": {
    "device": 2,
    "productionStatus": 1,
    "workorderId": "f0eca966-b43a-4cd6-afec-cbdc6e5ea1a1",
    "goodCount": 18,
    "badCount": 11,
    "temperature": 62.837414510245
  },
  "enqueuedTime": "Wed Feb 15 2023 04:02:19 GMT+0100 (czas środkowoeuropejski standardowy)",
  "properties": {}
}
```


Direct Methods:

```
1 odwołanie
public static async Task Emergency_Stop(string deviceId)
{
    Console.WriteLine($"\tDevice shut down {deviceId}\n");
    client.CallMethod($"ns=2;s=Device {deviceId}", $"ns=2;s=Device {deviceId}/EmergencyStop");
    client.WriteNode($"ns=2;s=Device {deviceId}/ProductionRate", OpcAttribute.Value, 0);
    await Task.Delay(1000);
}

1 odwołanie
public static async Task Reset_Errors(string deviceId)
{
    client.CallMethod($"ns=2;s=Device {deviceId}", $"ns=2;s=Device {deviceId}/ResetErrorStatus");
    await Task.Delay(1000);
}

1 odwołanie
public static async Task Maintenance()
{
    Program.maintenanceDate = DateTime.Now;
    Console.WriteLine($"Device Last Maintenace Date set to: {Program.maintenanceDate}\n");
    await IoTDevice.UpdateTwinValueAsync("LastMaintenanceDate", Program.maintenanceDate);
}
```

Metody wpływają na działanie urządzeń. Wywołuje się je wpisując ich nazwy w IoT Explorerze lub na platformie Azure, do których jest potrzebne kluczy z pliku „Logi.txt”:

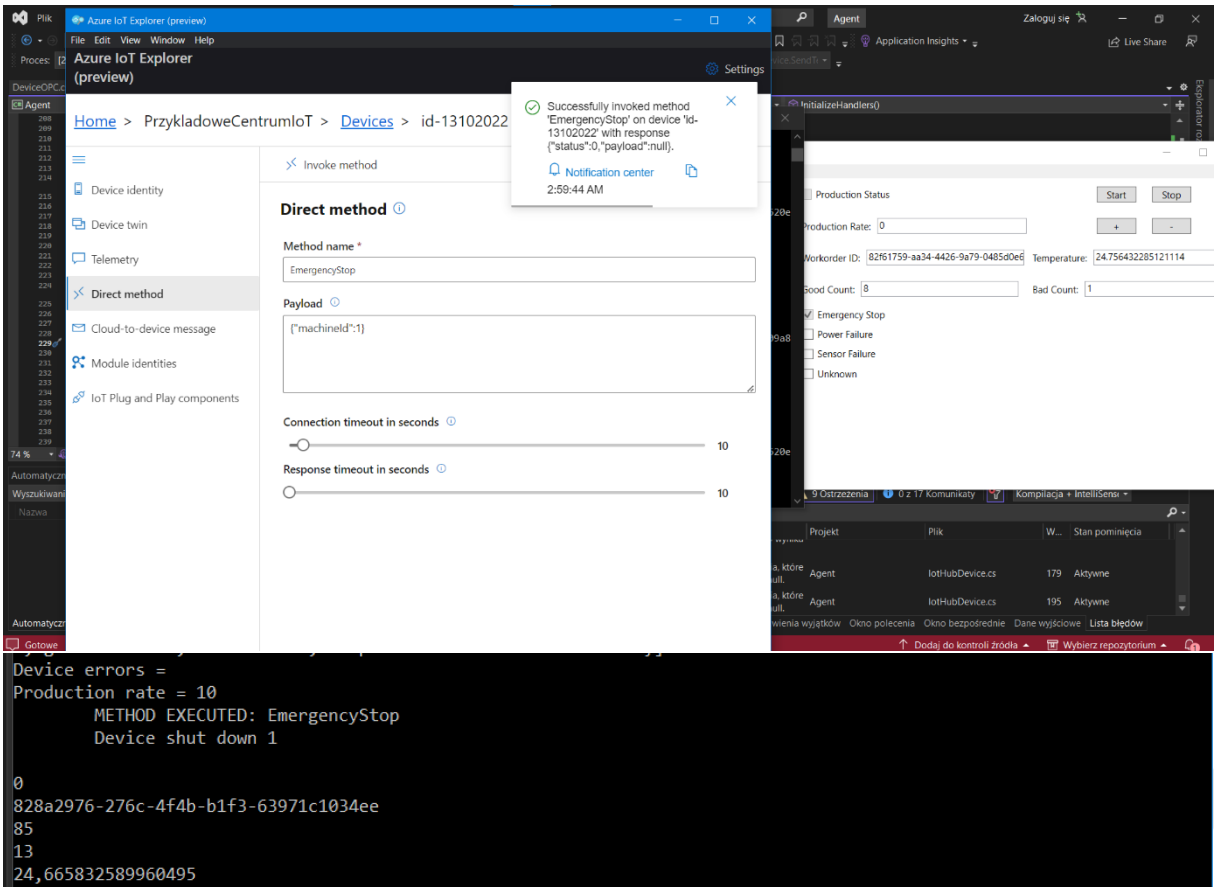
Są trzy główne funkcje:

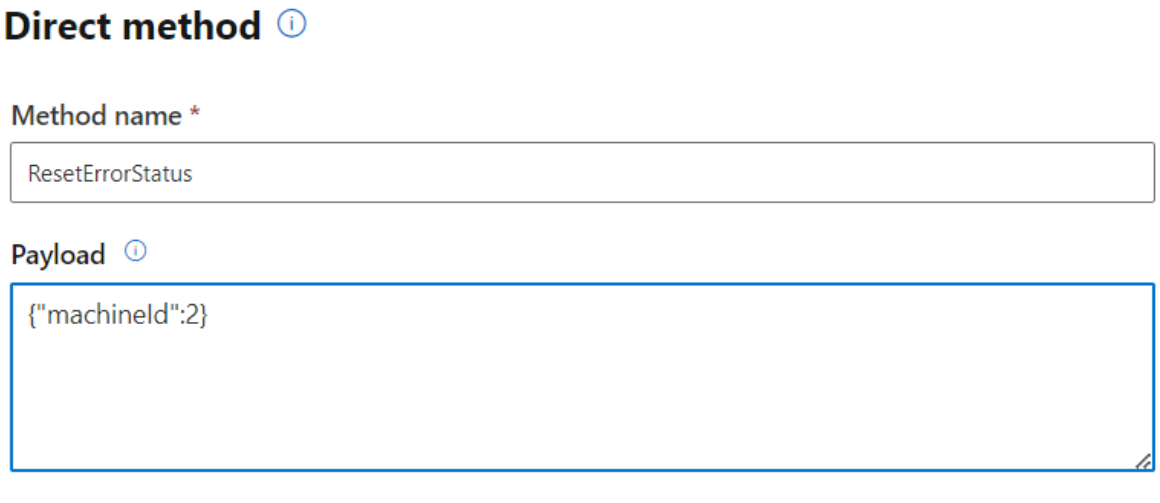
*EmergencyStop powodujące całkowite zatrzymanie produkcji w aktualnie połączonym urządzeniem aktywując przy tym flagę błędu;

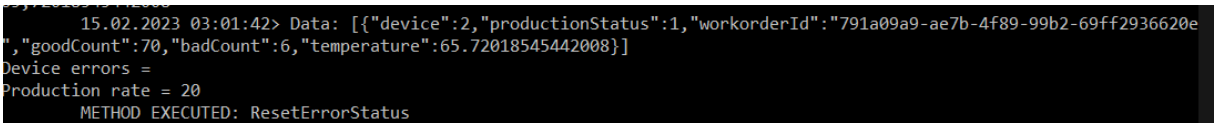
*ResetErrorStatus kasujący wszelkie błędy i informacje o nich;

*Maintanance ustawiający aktualną datę w DeviceTwinie.

Przykład działania Metod:


Azure IoT Explorer (preview)
Azure IoT Explorer (preview)
Settings
Home > PrzykladoweCentrumIoT > Devices > id-13102022
Successfully invoked method 'EmergencyStop' on device 'id-13102022' with response {"status":0,"payload":null}.
Notification center
2:59:44 AM
Device identity
Device twin
Telemetry
Direct method
Cloud-to-device message
Module identities
IoT Plug and Play components
Invoke method
Direct method
Method name *
EmergencyStop
Payload
{"machineId":1}
Connection timeout in seconds
10
Response timeout in seconds
10
Production Status
Start
Stop
Production Rate: 0
Workorder ID: 82f61759-aa34-4426-9a79-0485d0e6
Temperature: 24.756432285121114
Good Count: 8
Bad Count: 1
Emergency Stop
Power Failure
Sensor Failure
Unknown
Device errors =
Production rate = 10
METHOD EXECUTED: EmergencyStop
Device shut down 1
0
828a2976-276c-4f4b-b1f3-63971c1034ee
85
13
24,665832589960495


Direct method
Method name *
ResetErrorStatus
Payload
{"machineId":2}


15.02.2023 03:01:42> Data: [{"device":2,"productionStatus":1,"workorderId":"791a09a9-ae7b-4f89-99b2-69ff2936620e","goodCount":70,"badCount":6,"temperature":65.72018545442008}]
Device errors =
Production rate = 20
METHOD EXECUTED: ResetErrorStatus

Device Twin:

```
1 odwołanie
public static async Task TwinAsync(dynamic deviceErrors, dynamic productionRate)
{

    string errors = string.Empty;
    await UpdateTwinValueAsync("deviceErrors", deviceErrors);
    if ((deviceErrors & Convert.ToInt32(Errors.Unknown)) != 0)
    {
        errors += "Unknown, ";
    }
    if ((deviceErrors & Convert.ToInt32(Errors.SensorFailue)) != 0)
    {
        errors += "SensorFailure, ";
    }
    if ((deviceErrors & Convert.ToInt32(Errors.PowerFailure)) != 0)
    {
        errors += "PowerFailure, ";
    }
    if ((deviceErrors & Convert.ToInt32(Errors.EmergencyStop)) != 0)
    {
        errors += "Emergency stop";
    }

    await UpdateTwinValueAsync("productionRate", productionRate);
}

Odwołania: 2
public static async Task UpdateTwinValueAsync(string valueName, dynamic value)
{
    var twin = await client.GetTwinAsync();

    var reportedProperties = new TwinCollection();
    reportedProperties[valueName] = value;

    await client.UpdateReportedPropertiesAsync(reportedProperties);
}
```

```
Odwołania: 3
public static async Task UpdateTwinValueAsync(string valueName, DateTime value)
{
    var twin = await client.GetTwinAsync();

    var reportedProperties = new TwinCollection();
    reportedProperties[valueName] = value;

    await client.UpdateReportedPropertiesAsync(reportedProperties);
}

1 odwołanie
private async Task OnDesiredPropertyChanged(TwinCollection desiredProperties, object userContext)
{
    Console.WriteLine($"\tDesired property change:\n\t{JsonConvert.SerializeObject(desiredProperties)}");
    Console.WriteLine("\tSending current time as reported property");
    TwinCollection reportedProperties = new TwinCollection();
    reportedProperties["DateTimeLastDesiredPropertyChangeReceived"] = DateTime.Now;

    await client.UpdateReportedPropertiesAsync(reportedProperties).ConfigureAwait(false);
}
```


```
        "secondaryThumbprint": null
    },
    "modelId": "",
    "version": 5063,
    "properties": {
        "desired": {
            "$metadata": {
                "$lastUpdated": "2022-10-13T18:49:39.4572036Z"
            },
            "$version": 1
        },
        "reported": {
            "DateTimeLastAppLaunch": "2022-12-18T14:57:42.1947338+01:00",
            "productionRate": 10,
            "LastMaintenanceDate": "2023-02-14T12:56:36.1599064+01:00",
            "$metadata": {
                "$lastUpdated": "2023-02-15T03:00:56.6258524Z",
                "DateTimeLastAppLaunch": {
                    "$lastUpdated": "2022-12-18T13:57:42.6196846Z"
                },
                "productionRate": {
                    "$lastUpdated": "2023-02-15T03:00:56.6258524Z"
                },
                "LastMaintenanceDate": {
                    "$lastUpdated": "2023-02-14T11:56:35.8409634Z"
                }
            },
```

Device Twin odpowiada za zapis danych pobranych przez Agenta. Wśród nich znajdują się:

DeviceErrors – błędy urządzeń

ProductionRate – produkcja wykonywana przez urządzenia

Data Calculation:

Wykorzystując Azure Stream Analitics wykonywane są kalkulacje danych, które są zapisywane w kontenerach. Dane te można pobrać z portalu Azure albo przy pomocy Blob Storage.

Przykładowe kwerendy i wyniki:

```
SELECT
    MAX(temperature) AS max_temperature, MIN(temperature) As min_temperature, AVG(temperature) AS avg_temperature,
     device AS deviceID, workorderId
INTO
    [out-asa-temp]
FROM
    [in-asa]
GROUP BY TumblingWindow(minute, 5), device, workorderId;
```

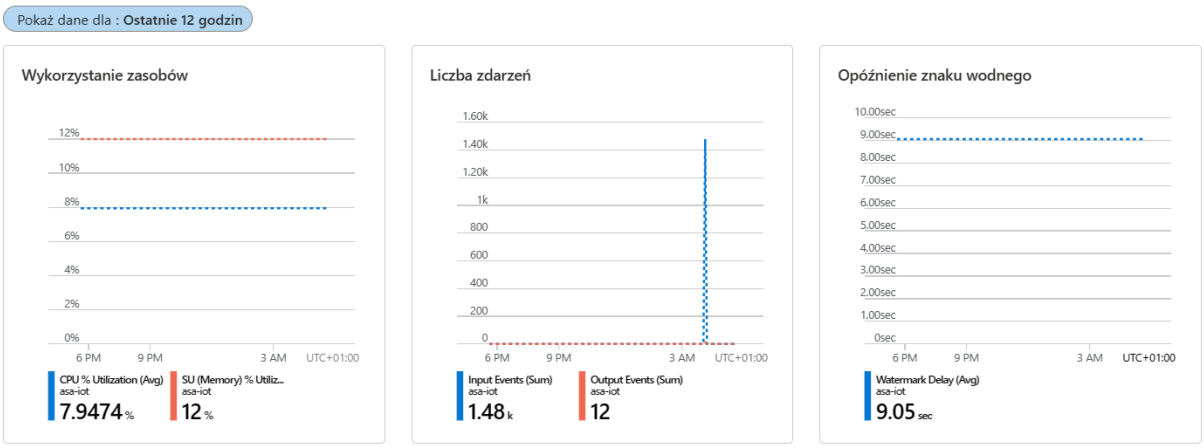
```
SELECT
    COUNT(goodCount) AS goodCount, COUNT(badCount) AS badCount
INTO
    [out-asa-prod]
FROM
    [in-asa]
GROUP BY TumblingWindow(minute, 5), device, workorderId;
```

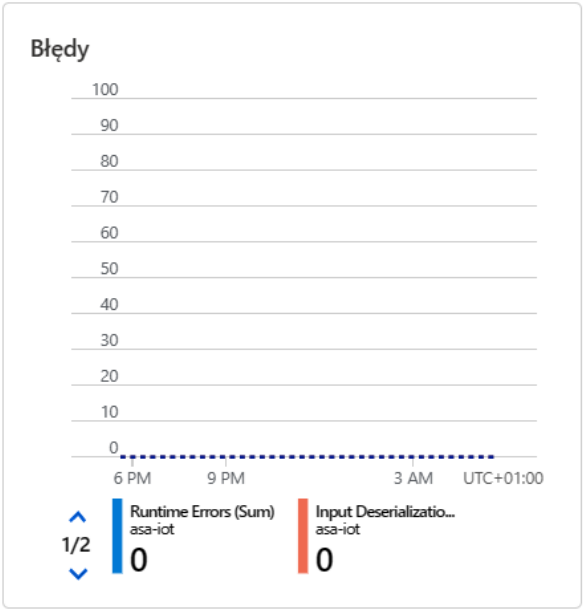
```
SELECT
    COUNT(goodCount)*100/(COUNT(goodCount)+COUNT(badCount)) productionKPI
INTO
    [out-asa-prod-KPI]
FROM
    [in-asa]
GROUP BY TumblingWindow(minute, 15), device, workorderId;
```

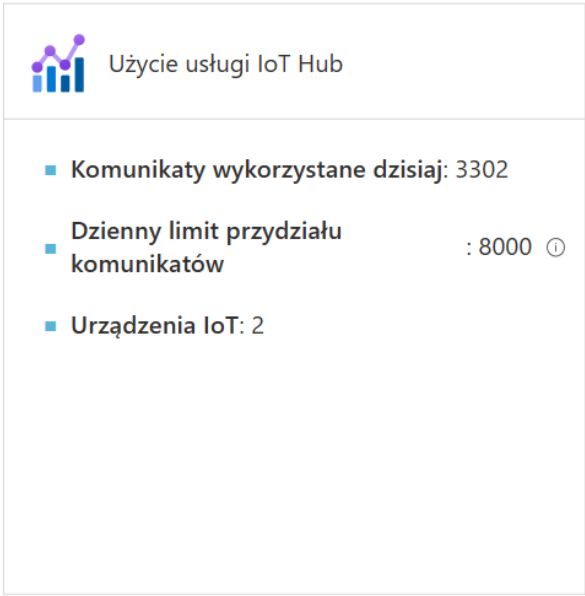
```
SELECT
    COUNT (EmergencyStop) AS Emergency_Stop_Counter,COUNT (PowerFailure) AS Power_Failure_Counter,
    COUNT (SensorFailure) AS Sensor_Failure_Counter,COUNT(Unknown) AS Unknown_Error_Counter
INTO
    [out-asa-err]
FROM
    [in-asa]
GROUP BY TumblingWindow(minute, 5), device, workorderId;
```

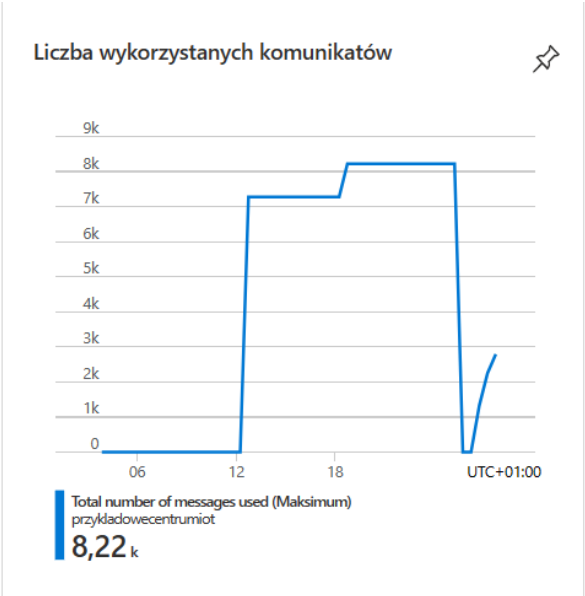
Przykładowe rezultaty kwerend znajdują się w folderze Asa results.

Pozostałe screeny:









Home > PrzykladoweCentrumIoT > Devices

New Refresh Delete

Query by device ID... Add query parameter

Device ID	Status	Connection st...	Authenticatio...	Last status up...	IoT Plug and ...	Edge device
id-13102022	Enabled	Disconnected	Sas	--		
dev2	Enabled	Disconnected	Sas	--		

Blob Storage Demo

Refresh lists Create container Upload to selected container Download selected blob

59f99d9e-d77b-4543-9764
errors
iot1
iot2
produkcja
temperatura

0_196d40e95f8d4789b2b5dd1f98acfe29_1.json
0_fcd16813afe14ab2870996d405b09b7b_1.json