Case Study – Azurep

Project Documentation

Spis treści

Connecting to OPC UA server

Connecting to OPC UA server instance is made by using OpcClient instance. Connection address is stored in project resource's file.

```
using (var opcClient = new OpcClient(Resources.opcServerAddress))
```

Device Agent Description

Pre-defined **Dictionary<string, string> OpcToIoTHubIds** is used to store OPC UA device id's with their device id on IoT Hub.

Connecting to each device on OPC UA server and it's representation on IoT Hub is made by using **Dictionary<VirtualDevice, OpcDeviceData> iotHubDeviceToOpcDeviceData**, where **OpcDeviceData** it's a class, which represents OPC UA device data. **VirtualDevice** – it's a class, which has **OpcClient** for writing/reading nodes of OPC UA device and Azure **DeviceClient** to send telemetry, events, and handle incoming method calls on this device.

Class **OpcDeviceData** also has two methods which create JSON file of the current data.
getTelemetryJSON() – returns JSON representation of device's telemetry.
getErrorsJSON() – returns JSON representation of device's errors parsed from binary error code.

In order to establish connection with IoT devices and read and store data from OPC server agent goes through every pre-defined id in **OpcToIoTHubIds** where ***Key*** – it's a OPC UA node and ***Value*** – device id in IoT Hub.

For every key-value pair these actions are performed:

- Connecting to IoT Hub device using connection string stored in project resources and device id stored in ***Value***.
- Creating **VirtualDevice** instance to store both side connection.
- Creating **OpcDeviceData** instance and reading data from OPC UA server to it using ***readOpcDeviceData*** method.
- Sending data to Device Twin.
- Initializing Direct Method handlers in **VirtualDevice** instance using ***InitializeHandlers*** async method.
- Saving **VirtualDevice** and **OpcDeviceData** objects to **iotHubDeviceToOpcDeviceData**.

After initialization, every 5 seconds agent goes through every device and reads it's data using static method ***readOpcDeviceData***. In case if there are any of errors on the device, program uses method ***sendDeviceErrorReport*** to send message to IoT Hub and update device's twin data.

```
OPCUA Device "ns=2;s=Device 1" is connected to IoT device "Device_1"
Device twin was set...
OPCUA Device "ns=2;s=Device 2" is connected to IoT device "Device_2"
Device twin was set...
OPCUA Device "ns=2;s=Device 3" is connected to IoT device "Device_3"
Device twin was set...
OPCUA Device "ns=2;s=Device 4" is connected to IoT device "Device_4"
Device twin was set...
OPCUA Device "ns=2;s=Device 5" is connected to IoT device "Device_5"
Device twin was set...
OPCUA Device "ns=2;s=Device 6" is connected to IoT device "Device_6"
Device twin was set...
```

Device Twin

In case if there is a production rate change or new device error occurred, new data is sent to device twin. Device Twim example for Device_1:


```
{
  "deviceId": "Device_1",
  "etag": "AAAAAAAAAAE=",
  "deviceEtag": "Njg5MTkwMzkx",
  "status": "enabled",
  "statusUpdateTime": "0001-01-01T00:00:00Z",
  "connectionState": "Disconnected",
  "lastActivityTime": "2023-01-05T06:49:44.9351578Z",
  "cloudToDeviceMessageCount": 0,
  "authenticationType": "sas",
  "x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null
  },
  "modelId": "",
  "version": 61,
  "properties": {
    "desired": {
      "$metadata": {
        "$lastUpdated": "2022-12-21T17:37:58.6058459Z"
      },
      "$version": 1
    },
    "reported": {
      "device_errors": 0,
      "production_rate": 60,
      "last_maintenance_date": "0001-01-01T00:00:00",
      "last_error_date": "0001-01-01T00:00:00",
      "$metadata": {
        "$lastUpdated": "2023-01-05T06:49:20.404008Z",
        "device_errors": {
          "$lastUpdated": "2023-01-05T06:49:20.404008Z"
        },
        "production_rate": {
          "$lastUpdated": "2023-01-05T06:49:20.404008Z"
        },
        "last_maintenance_date": {
          "$lastUpdated": "2023-01-05T06:49:20.404008Z"
        },
        "last_error_date": {
          "$lastUpdated": "2023-01-05T06:49:20.404008Z"
        }
      },
}
```

As was described above, for every IoT Device Device Twin is created using ***SetTwinDataAsync*** async method.

```
public async Task SetTwinDataAsync(int deviceError, int productionRate, DateTime lastMaintananceDate, DateTime lastErrorDate)
{
    var deviceTwin = await _deviceClient.GetTwinAsync();

    TwinCollection reportedProperties = new TwinCollection();
    reportedProperties["device_errors"] = deviceError;
    reportedProperties["production_rate"] = productionRate;
    reportedProperties["last_maintenance_date"] = lastMaintananceDate;
    reportedProperties["last_error_date"] = lastErrorDate;

    await  _deviceClient.UpdateReportedPropertiesAsync(reportedProperties);
    Console.WriteLine("Device twin was set...");
}
```

In order to update error or production rate status there are ***UpdateTwinErrorDataAsync*** and ***UpdateTwinProductionRateAsync*** async methods. They are called every 5 seconds if there is anything to report.

```
1 odwołanie
public async Task UpdateTwinErrorDataAsync(int deviceError)
{
    var deviceTwin = await _deviceClient.GetTwinAsync();
    Console.WriteLine($"{DateTime.Now}> Device Twin value was updated.");

    TwinCollection reportedProperties = new TwinCollection();
    reportedProperties["device_errors"] = deviceError;
    reportedProperties["last_error_date"] = DateTime.Now;

    await _deviceClient.UpdateReportedPropertiesAsync(reportedProperties);
}

1 odwołanie
public async Task UpdateTwinProductionRateAsync(int deviceError, int productionRate)
{
    var deviceTwin = await _deviceClient.GetTwinAsync();
    Console.WriteLine($"{DateTime.Now}> Device Twin value was updated.");

    TwinCollection reportedProperties = new TwinCollection();
    reportedProperties["device_errors"] = deviceError;
    reportedProperties["production_rate"] = productionRate;

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

Device To Cloud Messaging

As described above, agent reads OPC UA Devices data every 5 seconds and sends it to IoT hub.

```
Sending data to IoT Hub...
05.01.2023 09:50:40> D2C Sending message: {"opc_device_id":"ns=2;s=Device 1"
,"iot_device_id":"Device_1","production_status":1,"workorder_id":"8985f4b3-9
31e-4ad7-9dac-bff281e3be03","good_count":0,"bad_count":0,"temperature":25.04
2404819579332}
Sending data to IoT Hub...
05.01.2023 09:50:41> D2C Sending message: {"opc_device_id":"ns=2;s=Device 2"
,"iot_device_id":"Device_2","production_status":1,"workorder_id":"73c3199a-6
e15-467e-bf01-ae46c6e556fc","good_count":0,"bad_count":0,"temperature":25.95
53574801430 43}
Sending data to IoT Hub...
05.01.2023 09:50:41> D2C Sending message: {"opc_device_id":"ns=2;s=Device 3"
,"iot_device_id":"Device_3","production_status":1,"workorder_id":"1d5f021a-d
f41-4a46-bae3-93067ac3c503","good_count":0,"bad_count":0,"temperature":24.48
4599200046485}
```

After sending message to IoT Hub it sees sent message from device as shown below:


```
{
  "body": {
    "opc_device_id": "ns=2;s=Device 1",
    "iot_device_id": "Device_1",
    "production_status": 1,
    "workorder_id": "630e63dc-0d0e-4111-ac34-5ba0e7236b12",
    "good_count": 659,
    "bad_count": 75,
    "temperature": 65.25320641406748
  },
  "enqueuedTime": "Thu Jan 05 2023 10:15:41 GMT+0100 (czas środkowoeuropejski
standardowy)"
}
```


In case of new error agent sends message to IoT:

```
Error from device Device_1 was reported
Sending data to IoT Hub...
05.01.2023 10:03:34> D2C Sending message: {"opc_device_id":"ns=2;s=Device 1"
,"iot_device_id":"Device_1","is_error":true,"error_unknown":false,"error_sen
sor":false,"error_power":true,"error_emergency_stop":false}

05.01.2023 10:03:34> Device Twin value was updated.
```

Errors IoT receives in very similar way:


```
{
  "body": {
    "opc_device_id": "ns=2;s=Device 1",
    "iot_device_id": "Device_1",
    "is_error": true,
    "error_unknown": false,
    "error_sensor": true,
    "error_power": true,
    "error_emergency_stop": false
  },
  "enqueuedTime": "Thu Jan 05 2023 10:18:33 GMT+0100 (czas środkowoeuropejski
standardowy)"
}
```

Direct Methods

Also it has async methods, which are simultaneously awaiting for any incoming messages to defined device.

In case if any C2D messages od methods are sent to device from IoT, **VirtualDevice** class has a ***OnC2dMessageReceivedAsync*** async method for receiving messages and ***DecreaseProductRateHandler***, ***EmergencyStopHandler***, ***ResetErrorStatusHandler***, ***MaintenanceDoneHandler***, ***DefaultServiceHandler*** for receiving direct methods.

Initialization of method handlers inside ***VirtualDevice.InitializeHandlers***:

```
public async Task InitializeHandlers(string userContext)
{
    await _deviceClient.SetReceiveMessageHandlerAsync(OnC2dMessageReceivedAsync, userContext);

    await _deviceClient.SetMethodHandlerAsync("EmergencyStop", EmergencyStopHandler, userContext);
    await _deviceClient.SetMethodHandlerAsync("ResetErrorStatus", ResetErrorStatusHandler, userContext);
    await _deviceClient.SetMethodHandlerAsync("DecreaseProductRate", DecreaseProductRateHandler, userContext);
    await _deviceClient.SetMethodHandlerAsync("MaintenanceDone", MaintenanceDoneHandler, userContext);
    await _deviceClient.SetMethodDefaultHandlerAsync(DefaultServiceHandler, userContext);

    await _deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredProductionRateChanged, userContext);
}
```

For example, call handler of emergency stop method looks like this:

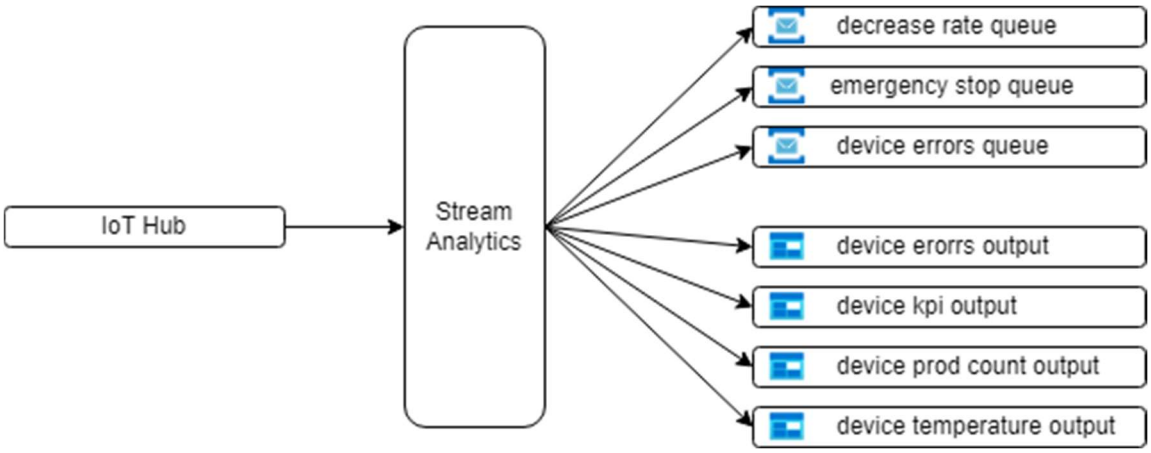
```
private async Task<MethodResponse> EmergencyStopHandler(MethodRequest methodRequest, object userContext)
{
    Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
    string nodeId = (string)userContext;
    object[] result = _opcClient.CallMethod(nodeId, nodeId + "/EmergencyStop");
    return new MethodResponse(0);
}
```

Example of calling method “`EmergencyStop`” on Device_1:

```
05.01.2023 10:20:39> METHOD EXECUTED: EmergencyStop
```

Data calculation

As for cloud data calculation was set up Azure Stream analytics. The only input stream Is an IoT Hub. On the output there are 3 service bus queues and 4 blob storage containers.



//Blob storages will store calculated data as described in section 2.2 of **Case Study – Azure**.

Storing messages in Service Bus

Service bus queues in this project are "endpoints" for triggering method calls and sending emails about device errors. The query below will filter incoming data from IoT Hub to ***device-decrease-production-rate*** queue if there is any device which has less than 90% of good product production.

```
SELECT
    System.Timestamp() time,
    iot_device_id AS deviceId
INTO [device-decrease-rate-output]
FROM [device-data-input]
GROUP BY iot_device_id, TumblingWindow(minute, 15)
HAVING (MAX(good_count)*100)/(MAX(good_count)+MAX(bad_count)) < 90
```

And this query below will send data to ***device-emergency-stop*** queue for invoking emergency stop on specified device if there is any errors occurred on it in past 15 minutes.

```
SELECT
    System.Timestamp() time,
    iot_device_id AS deviceId,
    SUM(COALESCE(error_emergency_stop,0)) + SUM(COALESCE(error_power,0)) + SUM
(COALESCE(error_sensor,0)) + SUM(COALESCE(error_unknown,0)) AS errorSum
INTO [device-emergency-stop-output]
FROM [device-data-input]
GROUP BY iot_device_id, TumblingWindow(minute, 15)
HAVING errorSum > 3
```

There 2 queries above will trigger "`DecreaseProductRate`" and "`EmergencyStop`" methods on specified devices using Function Apps.

The query below will send any error, which has occurred on any device to trigger Logic Apps functions for sending emails about these errors.

```
SELECT Collect() AS allEvents
INTO [device-errors-output]
FROM [device-data-input]
WHERE is_error = 1
GROUP BY Tumbling(second, 10)
```

Storing data in Blob storage

The queries below will save data calculations from query results into blob storages with specified name.

```
--Production per workorder count every 15 minutes
SELECT
    System.Timestamp() log_time,
    workorder_id,
    MAX(good_count) AS good_count,
    MAX(bad_count) AS bad_count
INTO [device-prod-count-output]
FROM [device-data-input]
WHERE workorder_id IS NOT NULL
GROUP BY workorder_id, TumblingWindow(minute, 15)
```

```
-- Device production KPI every 15 minutes
SELECT
    System.Timestamp() time,
    iot_device_id,
    (MAX(good_count)*100)/(MAX(good_count)+MAX(bad_count)) as production_kpi
INTO [device-kpi-output]
FROM [device-data-input]
GROUP BY iot_device_id, TumblingWindow(minute, 15)
```

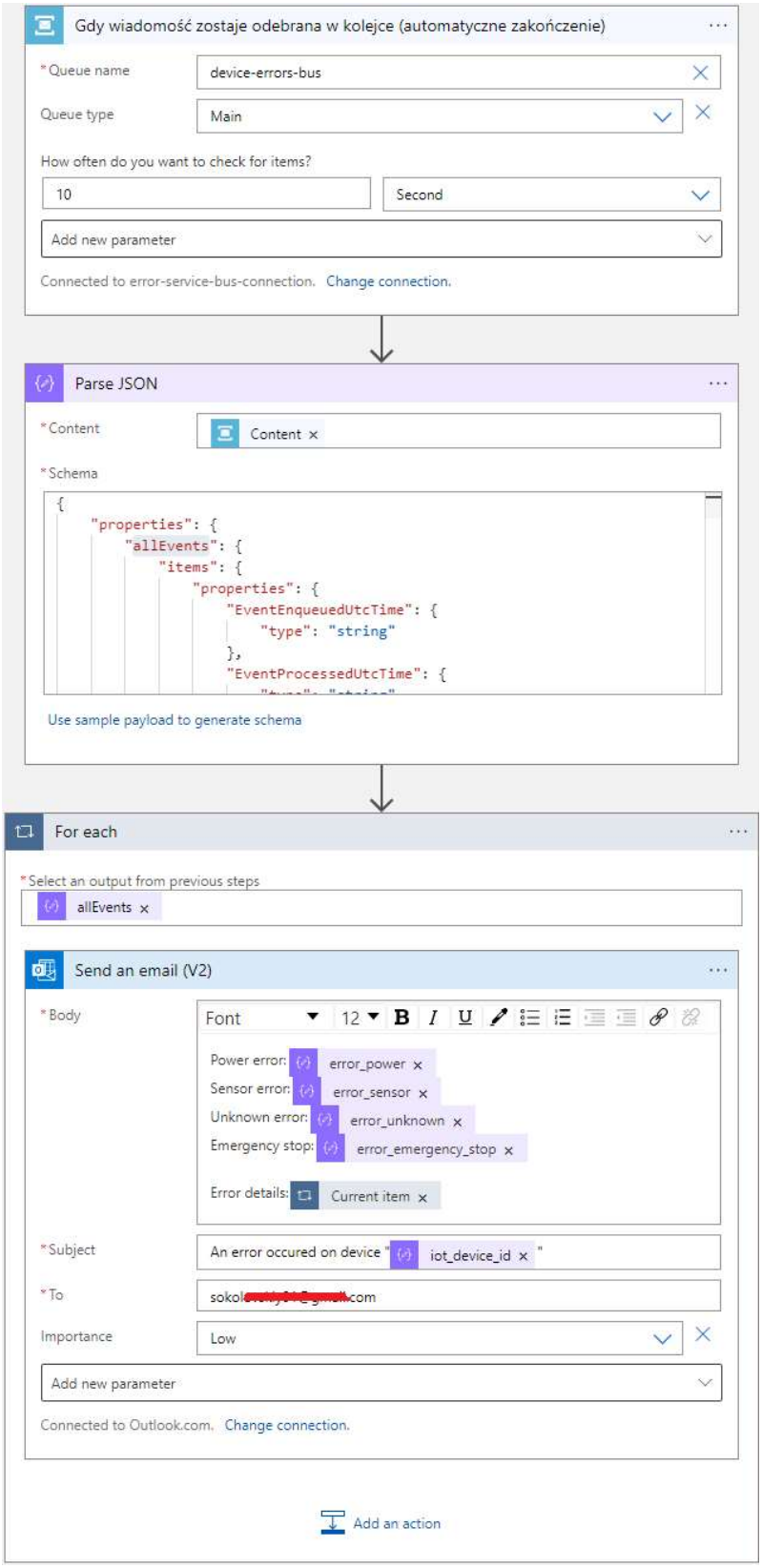
```
--Device's temperature every 5 minutes
SELECT
    System.Timestamp() log_time,
    iot_device_id,
    MAX(temperature) as temperature_max,
    MIN(temperature) as temperature_min,
    AVG(temperature) as temperature_avg
INTO [device-temperature-output]
FROM [device-data-input]
GROUP BY iot_device_id, TumblingWindow(minute, 5)
```

```
--Errors per machine every 30 minutes
SELECT
    System.Timestamp() time,
    iot_device_id,
    SUM(error_unknown) AS error_unknown_count,
    SUM(error_sensor) AS error_sensor_count,
    SUM(error_power) AS error_power_count,
    SUM(error_emergency_stop) AS error_emergency_stop_count
INTO [device-errors-reports-output]
FROM [device-data-input]
GROUP BY iot_device_id, TumblingWindow(minute, 30)
```

Example file content of ***devicetemperature*** storage container:

{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_5","temperature_max":108.61782104108727,"temperature_min":60.81257579616633,"temperature_avg":80.05718972124704}
{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_6","temperature_max":105.16631978455828,"temperature_min":67.15605294570004,"temperature_avg":86.14941037823635}
{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_1","temperature_max":129.80991171289944,"temperature_min":65.45836097878019,"temperature_avg":95.85393921010476}
{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_3","temperature_max":106.19681434759627,"temperature_min":61.739548796270995,"temperature_avg":77.17479187627058}
{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_2","temperature_max":716.0,"temperature_min":-783.0,"temperature_avg":59.733333333333334}
{"log_time":"2023-01-05T05:40:00.0000000Z","iot_device_id":"Device_4","temperature_max":114.05520472508806,"temperature_min":64.51093915421379,"temperature_avg":86.18532649209779}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_5","temperature_max":103.21234996992865,"temperature_min":60.90142293863809,"temperature_avg":79.21367753246183}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_6","temperature_max":112.502540972881,"temperature_min":60.06450333225412,"temperature_avg":83.46799387857304}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_1","temperature_max":151.2707051188939,"temperature_min":60.16909703741379,"temperature_avg":94.69628878725685}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_3","temperature_max":109.80614575062799,"temperature_min":60.23581213294332,"temperature_avg":85.00473066315637}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_2","temperature_max":905.0,"temperature_min":-909.0,"temperature_avg":77.71986856194246}
{"log_time":"2023-01-05T05:45:00.0000000Z","iot_device_id":"Device_4","temperature_max":105.28171369918957,"temperature_min":61.856772894682734,"temperature_avg":81.56312999684243}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_5","temperature_max":116.72975163525913,"temperature_min":60.0937193946087,"temperature_avg":84.56559104162947}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_6","temperature_max":112.26248799359051,"temperature_min":61.344183815844396,"temperature_avg":82.38219635941287}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_1","temperature_max":143.09682633965122,"temperature_min":61.083426411238495,"temperature_avg":99.93076125909636}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_3","temperature_max":115.19397763039939,"temperature_min":61.97951993422117,"temperature_avg":85.44169434532407}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_2","temperature_max":84.81939397397572,"temperature_min":60.085087561808464,"temperature_avg":71.51761822247109}
{"log_time":"2023-01-05T05:50:00.0000000Z","iot_device_id":"Device_4","temperature_max":108.99600229949621,"temperature_min":60.710122546698386,"temperature_avg":81.35238896208644}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_6","temperature_max":117.34602632567008,"temperature_min":60.04634293374363,"temperature_avg":82.30993623107017}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_5","temperature_max":114.16695680504527,"temperature_min":60.34146347672979,"temperature_avg":83.93873924022805}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_1","temperature_max":151.01457622888412,"temperature_min":60.242608786545105,"temperature_avg":101.49459368661802}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_3","temperature_max":107.57239265176815,"temperature_min":61.2975458571782,"temperature_avg":80.67683836997718}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_2","temperature_max":112.82349370371116,"temperature_min":-661.0,"temperature_avg":67.33770125006829}
{"log_time":"2023-01-05T05:55:00.0000000Z","iot_device_id":"Device_4","temperature_max":111.37166303323846,"temperature_min":61.11213227236784,"temperature_avg":81.7275780628295}

Logic App


Gdy wiadomość zostaje odebrana w kolejce (automatyczne zakończenie)
*Queue name
device-errors-bus
Queue type
Main
How often do you want to check for items?
10
Second
Add new parameter
Connected to error-service-bus-connection. Change connection.
Parse JSON
*Content
Content ×
*Schema
{
"properties": {
"allEvents": {
"items": {
"properties": {
"EventEnqueuedUtcTime": {
"type": "string"
},
"EventProcessedUtcTime": {
Use sample payload to generate schema
For each
*Select an output from previous steps
allEvents ×
Send an email (V2)
*Body
Font
12
Power error:
error_power ×
Sensor error:
error_sensor ×
Unknown error:
error_unknown ×
Emergency stop:
error_emergency_stop ×
Error details:
Current item ×
*Subject
An error occured on device "
iot_device_id ×
"
*To
Importance
Low
Add new parameter
Connected to Outlook.com. Change connection.
Add an action

Function App

Function Apps solution was added to this project for executing triggers when there are new messages in ***device-decrease-production-rate*** or ***device-emergency-stop*** service bus queues. These functions executing direct methods on specified device for decreasing device's production rate or executing emergency stop if there is more, than 3 error within 15 minutes on device.

This project uses ***Microsoft.Azure.Devices.CloudToDeviceMethod*** for calling methods on device. IoT Hub connection string is stored inside project's properties. Other connection data is stored in *local.settings.json*.

Example of received message

For emergency stop:

```
{"deviceId" : "Device_1", "errorSum" : 12.1, "time" : "2023-01-03T03:22:19.935745Z"}
```

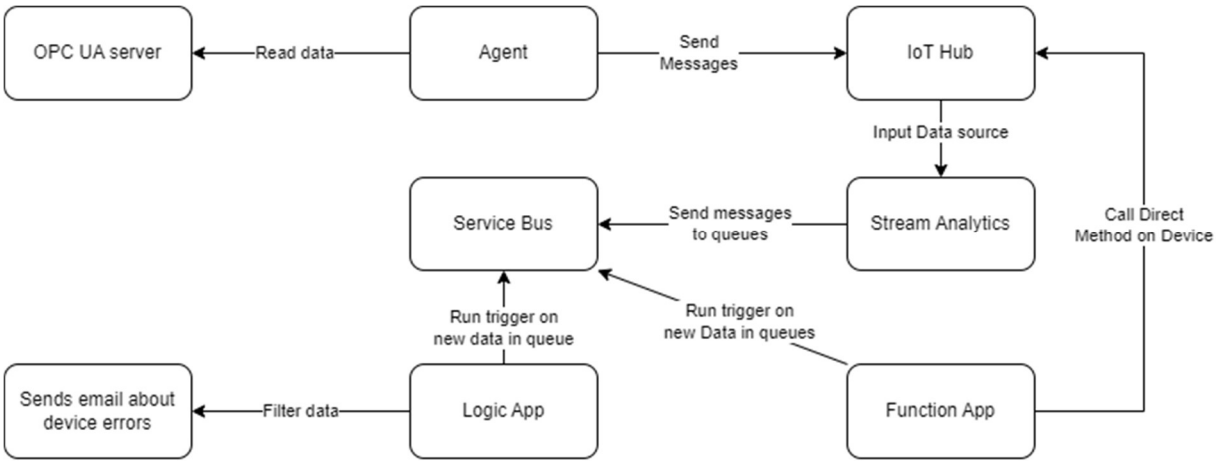
For decrease production rate:

```
{"deviceId" : "Device_1", "time" : "2023-01-06T011:43:25.754234Z"}
```

Example of executed trigger

```
[2023-01-09T19:16:09.096Z] Executing 'FunctionDecreaseRateQueue' (Reason='(null)', Id=740e805f-048c-4e84-98c8-34bff8aef649)
[2023-01-09T19:16:09.097Z] Trigger Details: MessageId: 8231e169576a4c009c5fbb54fccf0a02, SequenceNumber: 8, DeliveryCount: 29, EnqueuedTimeUtc: 2023-01-09T1
9:15:02.0290000+00:00, LockedUntilUtc: 2023-01-09T19:16:36.7950000+00:00, SessionId: c0f80
[2023-01-09T19:16:09.098Z] Recieved decrease production rate message: {"time":"2023-01-09T19:15:00.0000000Z","deviceId":"Device_1"}
[2023-01-09T19:16:09.100Z] DecreaseProductRate call result:
[2023-01-09T19:16:09.663Z] 0
[2023-01-09T19:16:09.664Z] null
[2023-01-09T19:16:09.667Z] Executed 'FunctionDecreaseRateQueue' (Succeeded, Id=740e805f-048c-4e84-98c8-34bff8aef649, Duration=570ms)
```

Business logic



Example of received email

An error occured on device "Device_1" Odebrane ×

Sokolovskyi Dmytro przez outlook.com 19:57 (0 minut temu)

do mnie

Power error:False
Sensor error:False
Unknown error:True
Emergency stop:False

Error details:{"opc_device_id":"ns=2;s=Device 1","iot_device_id":"Device_1","is_error":true,"error_unknown":true,"error_sensor":false,"error_power":false,"error_emergency_stop":false,"EventProcessedUtcTime":"2023-01-09T18:57:06.2360127Z","PartitionId":0,"EventEnqueuedUtcTime":"2023-01-09T18:57:06.0870000Z","IoTHub":{"MessageId":null,"CorrelationId":null,"ConnectionDeviceId":"Device_1","ConnectionDeviceGenerationId":"638072410786058459","EnqueuedTime":"2023-01-09T18:57:06.0510000Z"}}