Industrial IoT

Case Study – Azure

Project Documentation

Project repository:
https://github.com/LiLiTS96/IoT-Azure-UL
AzureDeviceSdkDemo.Device - class with functions to control the methods of the IoT device client.
FunctionAppTriggers - source code of Azure Function App.
ProjektZaliczeniowy – main class.
asa-dhyrenko – exported code of Azure Stream Analitycs.

Main class logic

1. To connect to the **Opc Client**, a new instance of the **OpcClient** class is initialized using the specified server address stored in the Resources object. A connection to the OPC client server is established using the **Connect()** method. The **BrowseNode()** method of the opcClient object is used to view the **OpcObjectTypes.ObjectsFolder** node.
2. Initialized **RegistryManager** object is used the connection string stored in the Resources object. Then it uses the **GetDevicesAsync** method of the **RegistryManager** object to get the list of devices registered in the IoT Hub, limited to the maximum number specified in the Resources object. The code then goes through the list of devices and checks if the device ID contains the certain string. If so, the device ID is added to the list of **deviceIds**.
3. Initialized a new dictionary, which has a key of type **VirtualDevice** and a value of type **MachineData**, which is a public class used to store data read from the Opc client's in a readable, fast-access way. Then in loop is created a new **DeviceClient** object using the connection string and the current element in the **deviceIds** list. The device client is opens asynchronously. Then it initialized a new **VirtualDevice** object using the **device client** and the **opcClient** object. Handlers are initialized for the **virtual device** using the **machineId** property of the current item in the **machineDataList**. The **virtual device** and its corresponding **machineData** object are then added to the dictionary. Finally, the property **iotHubDeviceId** of the object **MachineData** is set to the current item in the list **deviceIds**.
4. For each key-value pair in the dictionary, the code does the following:
   Calls the **readNode** function and passes the value (of type **MachineData**) and **opcClient** as arguments.
   Calls the **SetTwinAsync** function on the key (of type **VirtualDevice**) and passes the values of **deviceErrors** and **productRate** values (of type **MachineData**) as arguments.
   If the **deviceErrors** value (of type **MachineData**) is greater than 0, which means that there are errors on the device. The function **reportNewError** is called for set error status at **Device Twin** and send it to the **IoT Hub**.
5. Then the fourth point is repeated every 5 seconds by reading the data from the device, if the device production status is active, the telemetry data are sent to the cloud with as a message, in addition, if new errors occur or their status changes, an error message is also sent to the cloud and the data are updated in the device twin.

Connection to the device (Opc UA server)

The connection method is described in the previous section.

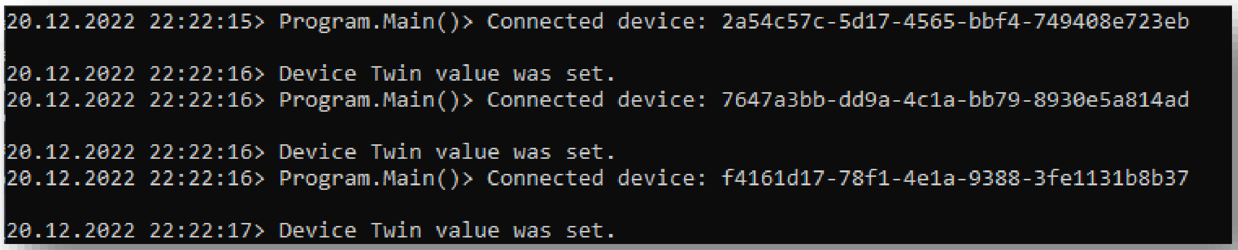
The Agent reads data from the device using a function **readNode** that takes as arguments an object of class **MachineData**(was described) and **OpcClient** and then writes the data in the current MachineData object by reading them from certain nodes on the device.

```
static void readNode(MachineData machineData, OpcClient client)
        {
            machineData.productStatus = (int)client.ReadNode(machineData.machineId +NODE_PRODUCT_STATUS).Value;
            machineData.workorderId = (string)client.ReadNode(machineData.machineId + NODE_WORKORDER_ID).Value;
            machineData.goodCount = (int)(long)client.ReadNode(machineData.machineId + NODE_GOOD_COUNT).Value;
            machineData.badCount = (int)(long)client.ReadNode(machineData.machineId + NODE_BAD_COUNT).Value;
            machineData.temperature = (double)client.ReadNode(machineData.machineId + NODE_TEMPERATURE).Value;
            machineData.deviceErrors = (int)client.ReadNode(machineData.machineId + NODE_DEVICE_ERROR).Value;
            machineData.productRate = (int)client.ReadNode(machineData.machineId + NODE_PRODUCT_RATE).Value;
            machineData.readTimeStamp = DateTime.Now;
        }
```

The data updating on the device takes place in a class **OnDesiredPropertyChanged** , which serves to track information in case of changes desired properties on Device Twin, to reduces the Machine production Rate and report new property to Device Twin.

```
private async Task OnDesiredPropertyChanged(TwinCollection desiredProperties, object userContext)
        {
            Console.WriteLine($"\t{DateTime.Now}> Device Twin. Desired property
                                                 change:\n\t{JsonConvert.SerializeObject(desiredProperties)}");
            string nodeId = (string)userContext;
            int newProdRate = desiredProperties["ProductionRate"];
            string node = nodeId + "/ProductionRate";

            OpcStatus result = opcClient.WriteNode(node, newProdRate);
            Console.WriteLine($"\t{DateTime.Now}> opcClient.WriteNode is result good: " +
                                                                    result.IsGood.ToString());
            TwinCollection reportedProperties = new TwinCollection();
            reportedProperties["DateTimeLastDesiredPropertyChangeReceived"] = DateTime.Now;
            reportedProperties["ProductionRate"] = desiredProperties["ProductionRate"];

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

Example of logs. the Agent connected three devices from the Opc Server to three devices of the IoT Hub and updated their device twins.



```
20.12.2022 22:22:15> Program.Main()> Connected device: 2a54c57c-5d17-4565-bbf4-749408e723eb

20.12.2022 22:22:16> Device Twin value was set.
20.12.2022 22:22:16> Program.Main()> Connected device: 7647a3bb-dd9a-4c1a-bb79-8930e5a814ad

20.12.2022 22:22:16> Device Twin value was set.
20.12.2022 22:22:16> Program.Main()> Connected device: f4161d17-78f1-4e1a-9388-3fe1131b8b37

20.12.2022 22:22:17> Device Twin value was set.
```

Agent Configuration

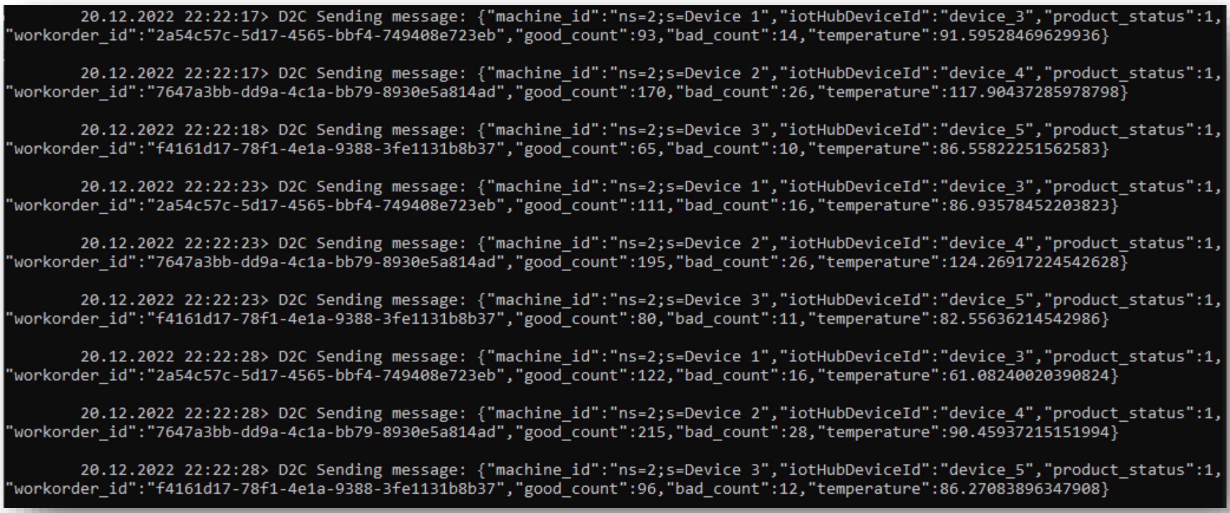
ProjektZaliczeniowy configuration file is **ProjektZaliczeniowy\Properties\Resources.resx.**

deviceConnectionString	<deviceConnectionString from IoT Hub>
ownerConnectionString	<ownerConnectionString from IoT Hub >
opcClientServer	opc.tcp://localhost:4840/
iotDevicesMaxCount	100

The way in which the Agent connects to the Opc Server and to the IoT Hub is described in the first section.

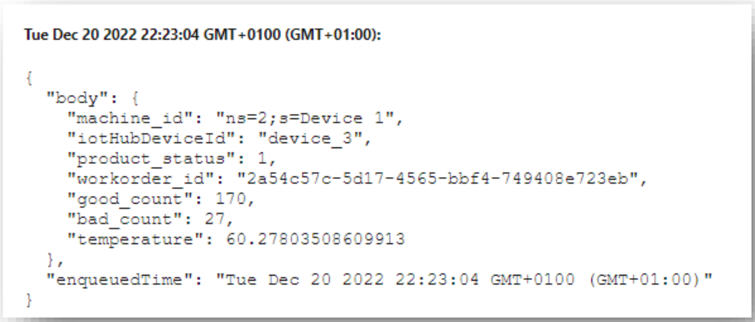
D2C Messages

The Agent reads data from machines every 5 seconds. It converts them into json with telemetry data, which it sends immediately to the IoT Hub using **SendEventAsync**() Device Client method.
Log send messages to IoT Hub:



```
        20.12.2022 22:22:17> D2C Sending message: {"machine_id":"ns=2;s=Device 1","iotHubDeviceId":"device_3","product_status":1,
"workorder_id":"2a54c57c-5d17-4565-bbf4-749408e723eb","good_count":93,"bad_count":14,"temperature":91.59528469629936}

        20.12.2022 22:22:17> D2C Sending message: {"machine_id":"ns=2;s=Device 2","iotHubDeviceId":"device_4","product_status":1,
"workorder_id":"7647a3bb-dd9a-4c1a-bb79-8930e5a814ad","good_count":170,"bad_count":26,"temperature":117.90437285978798}

        20.12.2022 22:22:18> D2C Sending message: {"machine_id":"ns=2;s=Device 3","iotHubDeviceId":"device_5","product_status":1,
"workorder_id":"f4161d17-78f1-4e1a-9388-3fe1131b8b37","good_count":65,"bad_count":10,"temperature":86.55822251562583}

        20.12.2022 22:22:23> D2C Sending message: {"machine_id":"ns=2;s=Device 1","iotHubDeviceId":"device_3","product_status":1,
"workorder_id":"2a54c57c-5d17-4565-bbf4-749408e723eb","good_count":111,"bad_count":16,"temperature":86.93578452203823}

        20.12.2022 22:22:23> D2C Sending message: {"machine_id":"ns=2;s=Device 2","iotHubDeviceId":"device_4","product_status":1,
"workorder_id":"7647a3bb-dd9a-4c1a-bb79-8930e5a814ad","good_count":195,"bad_count":26,"temperature":124.26917224542628}

        20.12.2022 22:22:23> D2C Sending message: {"machine_id":"ns=2;s=Device 3","iotHubDeviceId":"device_5","product_status":1,
"workorder_id":"f4161d17-78f1-4e1a-9388-3fe1131b8b37","good_count":80,"bad_count":11,"temperature":82.55636214542986}

        20.12.2022 22:22:28> D2C Sending message: {"machine_id":"ns=2;s=Device 1","iotHubDeviceId":"device_3","product_status":1,
"workorder_id":"2a54c57c-5d17-4565-bbf4-749408e723eb","good_count":122,"bad_count":16,"temperature":61.08240020390824}

        20.12.2022 22:22:28> D2C Sending message: {"machine_id":"ns=2;s=Device 2","iotHubDeviceId":"device_4","product_status":1,
"workorder_id":"7647a3bb-dd9a-4c1a-bb79-8930e5a814ad","good_count":215,"bad_count":28,"temperature":90.45937215151994}

        20.12.2022 22:22:28> D2C Sending message: {"machine_id":"ns=2;s=Device 3","iotHubDeviceId":"device_5","product_status":1,
"workorder_id":"f4161d17-78f1-4e1a-9388-3fe1131b8b37","good_count":96,"bad_count":12,"temperature":86.27083896347908}
```

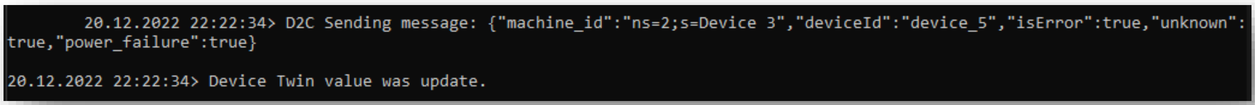
Log received messages in IoT Hub:



Tue Dec 20 2022 22:23:04 GMT+0100 (GMT+01:00):

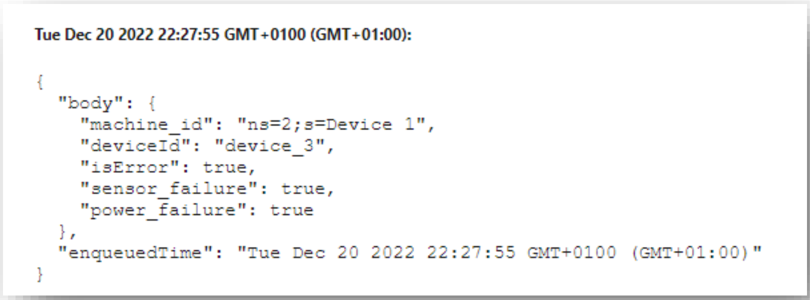
```
{
  "body": {
    "machine_id": "ns=2;s=Device 1",
    "iotHubDeviceId": "device_3",
    "product_status": 1,
    "workorder_id": "2a54c57c-5d17-4565-bbf4-749408e723eb",
    "good_count": 170,
    "bad_count": 27,
    "temperature": 60.27803508609913
  },
  "enqueuedTime": "Tue Dec 20 2022 22:23:04 GMT+0100 (GMT+01:00)"
}
```

If the error status changes, it also sends a separate message to the IoT Hub with information about the errors using **SendEventAsync**() Device Client method.
Log send messages to IoT Hub:



```
        20.12.2022 22:22:34> D2C Sending message: {"machine_id":"ns=2;s=Device 3","deviceId":"device_5","isError":true,"unknown":
true,"power_failure":true}

20.12.2022 22:22:34> Device Twin value was update.
```

Log received messages in IoT Hub:



Tue Dec 20 2022 22:27:55 GMT+0100 (GMT+01:00):

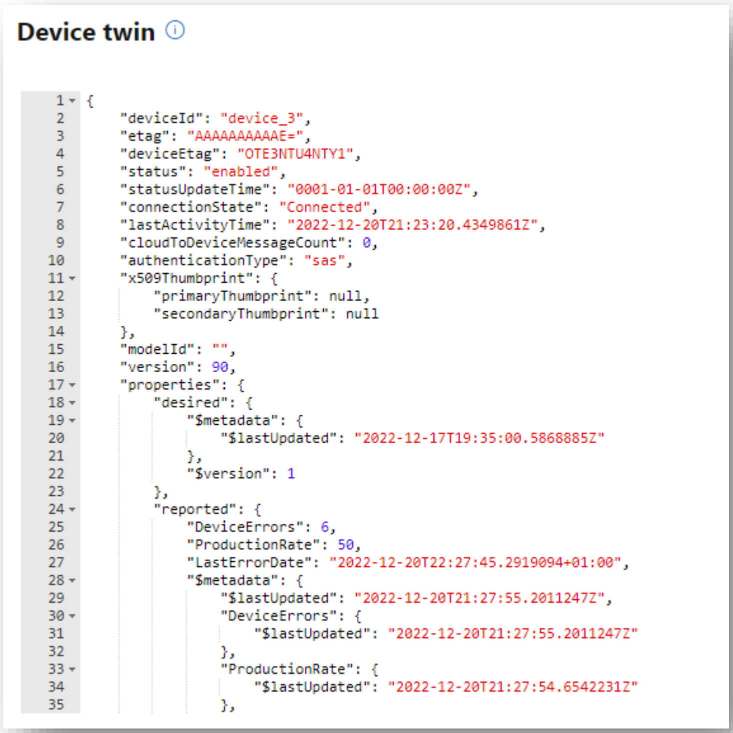
```
{
  "body": {
    "machine_id": "ns=2;s=Device 1",
    "deviceId": "device_3",
    "isError": true,
    "sensor_failure": true,
    "power_failure": true
  },
  "enqueuedTime": "Tue Dec 20 2022 22:27:55 GMT+0100 (GMT+01:00)"
}
```

Device Twin

The new data of Production Rate, Device Errors ant theirs last occur date is reported to Device Twin in **SetTwinAsync** or **UpdateTwinAsync** methods.

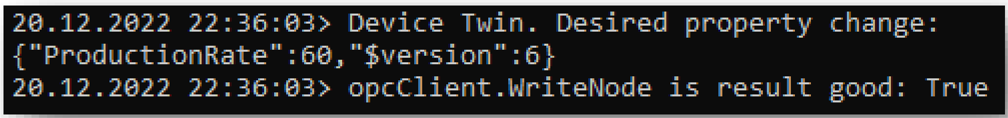
```
public async Task UpdateTwinAsync(int deviceError)
        {
            var twin = await client.GetTwinAsync();
            Console.WriteLine($"{DateTime.Now}> Device Twin value was update.");
            Console.WriteLine();

            var reportedProperties = new TwinCollection();
            reportedProperties["DeviceErrors"] = deviceError;
            reportedProperties["LastErrorDate"] = DateTime.Now;

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

Device Twin example:



```
{
    "deviceId": "device_3",
    "etag": "AAAAAAAAAAE=",
    "deviceEtag": "OTE3NTU4NTY1",
    "status": "enabled",
    "statusUpdateTime": "0001-01-01T00:00:00Z",
    "connectionState": "Connected",
    "lastActivityTime": "2022-12-20T21:23:20.4349861Z",
    "cloudToDeviceMessageCount": 0,
    "authenticationType": "sas",
    "x509Thumbprint": {
        "primaryThumbprint": null,
        "secondaryThumbprint": null
    },
    "modelId": "",
    "version": 90,
    "properties": {
        "desired": {
            "$metadata": {
                "$lastUpdated": "2022-12-17T19:35:00.5868885Z"
            },
            "$version": 1
        },
        "reported": {
            "DeviceErrors": 6,
            "ProductionRate": 50,
            "LastErrorDate": "2022-12-20T22:27:45.2919094+01:00",
            "$metadata": {
                "$lastUpdated": "2022-12-20T21:27:55.2011247Z",
                "DeviceErrors": {
                    "$lastUpdated": "2022-12-20T21:27:55.2011247Z"
                },
                "ProductionRate": {
                    "$lastUpdated": "2022-12-20T21:27:54.6542231Z"
                },
```

New data retrieved in desired property is update data in Machine, method **OnDesiredPropertyChanged** is described at Connection to the device (Opc UA server) section
Log device twin desired property:



```
20.12.2022 22:36:03> Device Twin. Desired property change:
{"ProductionRate":60,"$version":6}
20.12.2022 22:36:03> opcClient.WriteNode is result good: True
```

Direct Methods

There are five Direct Methods implemented in the Agent.

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

The first two methods serve to call the method on the machine.
An example of one of them:

```
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);
        }
```

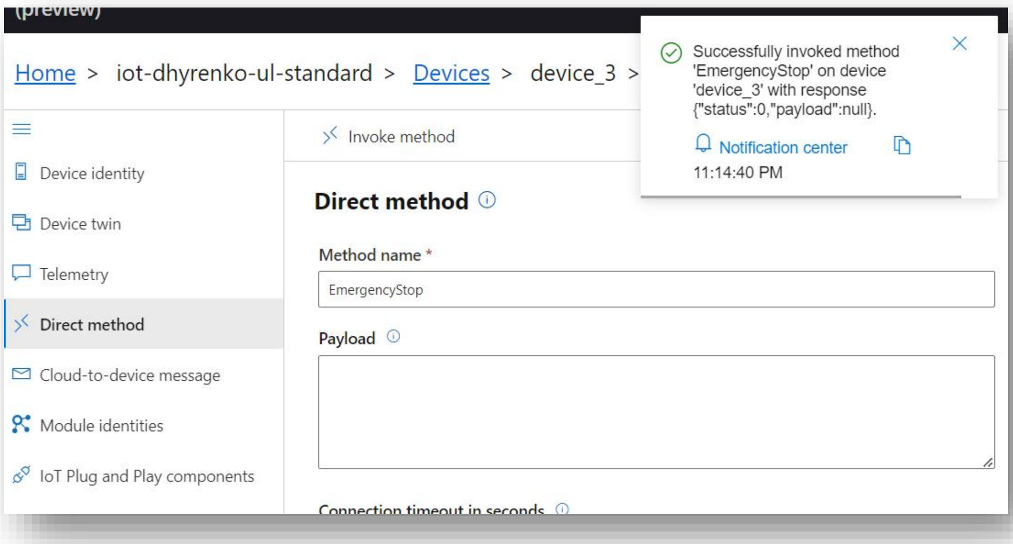
In userContext the device id that was initialized for each device in the main class:

```
await device.InitializeHandlers(machineDataList[i].machineId);
```

DecreaseProductRate and MaintenanceDone serve to update reported property in Device Twin.

```
private async Task<MethodResponse> DecreaseProductRateHandler(MethodRequest methodRequest, object userContext)
        {
            string productionRate = "/ProductionRate";
            string deviceError = "/DeviceError";
            Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
            string nodeId = (string)userContext;
            int rate = (int)opcClient.ReadNode(nodeId + productionRate).Value;
            int error = (int)opcClient.ReadNode(nodeId + deviceError).Value;
            OpcStatus result = opcClient.WriteNode(nodeId + productionRate, rate - 10);
            Console.WriteLine(result.ToString());
            await SetTwinAsync(error, rate - 10);
            return new MethodResponse(0);
        }

private async Task<MethodResponse> MaintenanceDoneHandler(MethodRequest methodRequest, object userContext)
        {
            Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
            var twin = await client.GetTwinAsync();
            var reportedProperties = new TwinCollection();
            reportedProperties["LastMainTenanceDone"] = DateTime.Now;
            await client.UpdateReportedPropertiesAsync(reportedProperties);
            Console.WriteLine($"\n{DateTime.Now}> Device Twin Maintenance Done.");

            return new MethodResponse(0);
        }
```

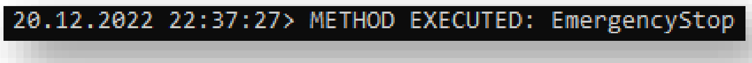
In the case of a call to a method with a uninitialized name, a default response is initiated that does nothing.
In case of successful execution the methods return 0


Home > iot-dhyrenko-ul-standard > Devices > device_3 >

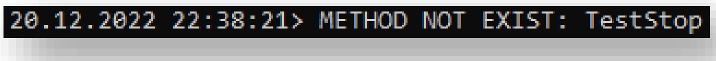
Successfully invoked method 'EmergencyStop' on device 'device_3' with response {"status":0,"payload":null}.
Notification center
11:14:40 PM

Direct method
Method name * EmergencyStop
Payload

Log direct method:



```
20.12.2022 22:37:27> METHOD EXECUTED: EmergencyStop
```



```
20.12.2022 22:38:21> METHOD NOT EXIST: TestStop
```

Data calculation

All data calculation take place in Azure Stream Analytics.

Stream input is IoT Hub

Outputs are six Blob storages and two Service Bus queue.

Queries used in ASA:

Situations, when there are more than 3 errors within 15 minutes (per machine):

```
SELECT
    System.Timestamp() emergency_stop_time,
    machine_id,
    SUM(COALESCE(emergency_stop,0)) + SUM(COALESCE(power_failure,0)) + SUM(COALESCE(sensor_failure,0)) + SUM(COALESCE(unknown
,0)) as error_sum
INTO
    [asa-out-emerg-stops]
FROM
    [asa-in-dhyrenko]
GROUP BY
    machine_id,
    TumblingWindow(minute, 15)
HAVING
    error_sum > 3
```

Temperature per machine (maximum, minimum, average):

```
SELECT
    System.Timestamp() log_time,
    machine_id,
    MAX(temperature) as temperature_max,
    MIN(temperature) as temperature_min,
    AVG(temperature) as temperature_avg
INTO
    [asa-out-temp]
FROM
    [asa-in-dhyrenko]
GROUP BY
    machine_id,
    TumblingWindow(minute, 5)
```

Production per workorder (sum of good/bad counts):

```
SELECT
    System.Timestamp() log_time,
    workorder_id,
    MAX(good_count) as good_count_sum,
    MAX(bad_count) as bad_count_sum
INTO
    [asa-out-aggreg-counts]
FROM
    [asa-in-dhyrenko]
WHERE
```

```
    workorder_id IS NOT NULL
GROUP BY
    workorder_id,
    TumblingWindow(minute, 30)
```

% of good production (vs total production) in 15-minute windows:

```
SELECT
    System.Timestamp() log_time,
    machine_id,
    (MAX(good_count)*100)/(MAX(good_count)+MAX(bad_count)) as proc_of_good_production
INTO
    [asa-out-proc-good-count]
FROM
    [asa-in-dhyrenko]
GROUP BY
    machine_id,
    TumblingWindow(minute, 15)
```

Number of individual errors per machine in 30-minute windows:

```
SELECT
    System.Timestamp() log_time,
    machine_id,
    SUM(emergency_stop) as emergency_stop_count,
    SUM(power_failure) as power_failure_count,
    SUM(sensor_failure) as sensor_failure_count,
    SUM(unknown) as unknown_error_count
INTO
    [asa-out-errors]
FROM
    [asa-in-dhyrenko]
GROUP BY
    machine_id,
    TumblingWindow(minute, 30)
```

If there are more than 3 errors within 15 minutes (per machine):
Send Message to Service Bus Queue

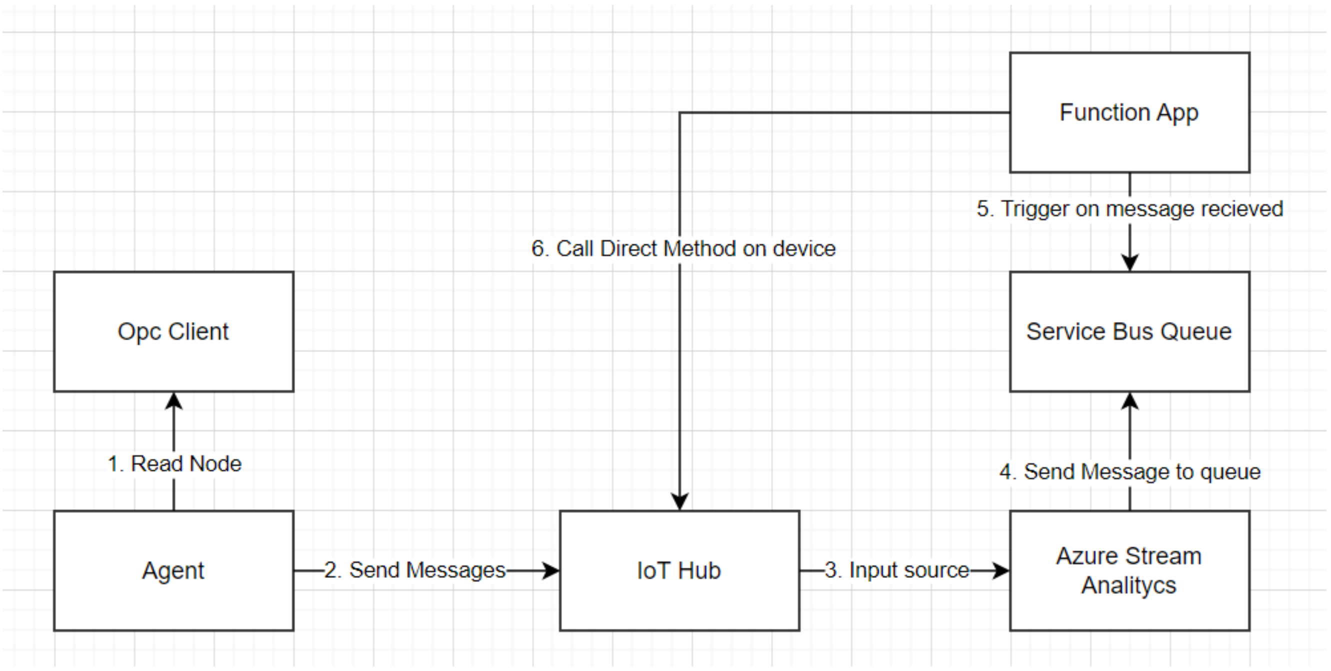
```
SELECT
    System.Timestamp() time,
    deviceId
INTO
    [asa-out-decrease-rate-queue]
FROM
    [asa-in-dhyrenko]
GROUP BY
    deviceId,
    TumblingWindow(minute, 15)
HAVING
    (MAX(good_count)*100)/(MAX(good_count)+MAX(bad_count)) < 90
```

If % of good production in 15-minute window drops below 90% Send Message to Service Bus Queue :

```
SELECT
    System.Timestamp() emergency_stop_time,
    deviceId,
    SUM(COALESCE(emergency_stop,0)) + SUM(COALESCE(power_failure,0)) + SUM(COALESCE(sensor_failure,0)) + SUM(COALESCE(unknown
,0)) as error_sum
INTO
    [asa-out-error-bus-queue]
FROM
    [asa-in-dhyrenko]
GROUP BY
    deviceId,
    TumblingWindow(minute, 15)
HAVING
    error_sum > 3
```

All Logs example added to github repo in LogExample directory.

Business logic

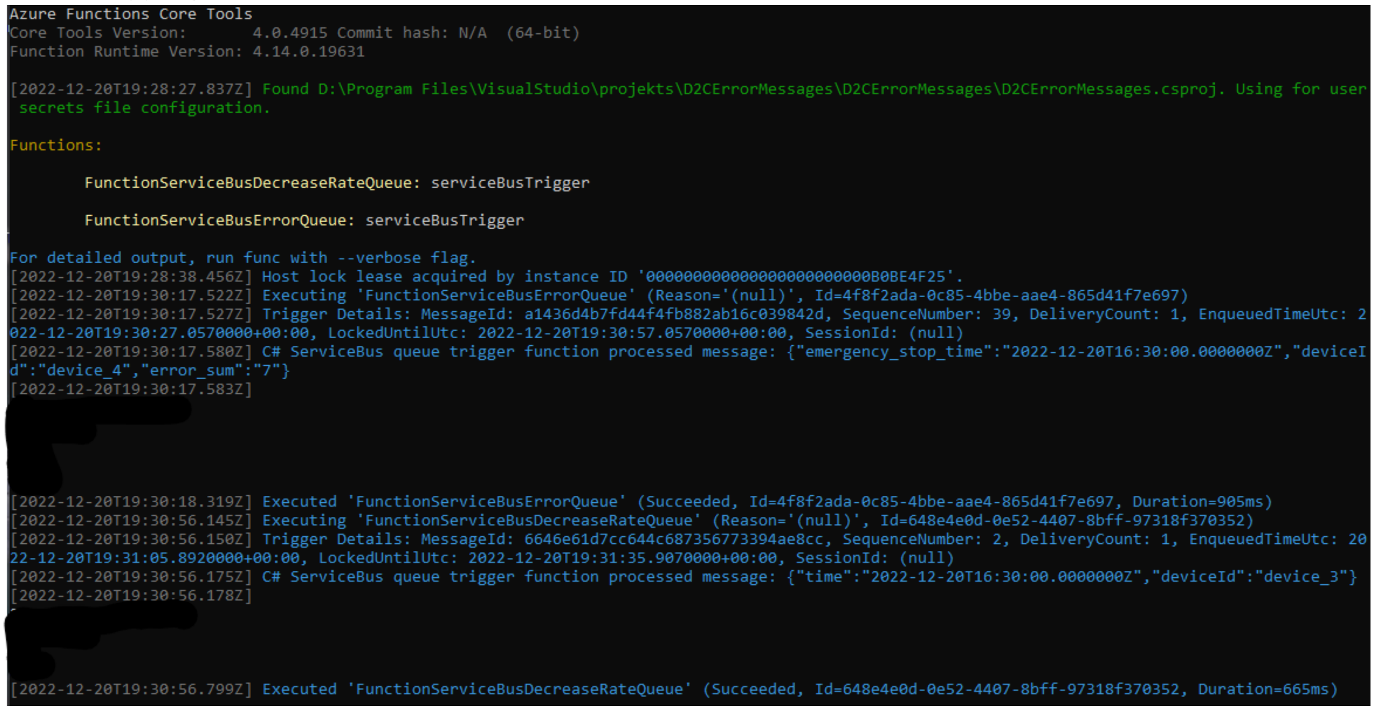


ASA sends a message in case the conditions are met, which in turn releases the trigger that reacts to the new message and immediately calls the direct method on the given device.
Messages sent to the queue contain the id of the IoT device on which you want to call the method.

Message example send to **error-queue** Service Bus Queue:
{"emergency_stop_time":"2022-12-20T16:30:00.0000000Z","deviceId":"device_4","error_sum":"7"}

Message example send to **decrease-rate-queue** Service Bus Queue:
{"time":"2022-12-20T16:30:00.0000000Z","deviceId":"device_3"}

Function app logs:



```
Azure Functions Core Tools
Core Tools Version:       4.0.4915 Commit hash: N/A  (64-bit)
Function Runtime Version: 4.14.0.19631

[2022-12-20T19:28:27.837Z] Found D:\Program Files\VisualStudio\projekts\D2CErrorMessages\D2CErrorMessages\D2CErrorMessages.csproj. Using for user
 secrets file configuration.

Functions:

        FunctionServiceBusDecreaseRateQueue: serviceBusTrigger

        FunctionServiceBusErrorQueue: serviceBusTrigger

For detailed output, run func with --verbose flag.
[2022-12-20T19:28:38.456Z] Host lock lease acquired by instance ID '0000000000000000000000000B0BE4F25'.
[2022-12-20T19:30:17.522Z] Executing 'FunctionServiceBusErrorQueue' (Reason='(null)', Id=4f8f2ada-0c85-4bbe-aae4-865d41f7e697)
[2022-12-20T19:30:17.527Z] Trigger Details: MessageId: a1436d4b7fd44f4fb882ab16c039842d, SequenceNumber: 39, DeliveryCount: 1, EnqueuedTimeUtc: 2
022-12-20T19:30:27.0570000+00:00, LockedUntilUtc: 2022-12-20T19:30:57.0570000+00:00, SessionId: (null)
[2022-12-20T19:30:17.580Z] C# ServiceBus queue trigger function processed message: {"emergency_stop_time":"2022-12-20T16:30:00.0000000Z","deviceI
d":"device_4","error_sum":"7"}
[2022-12-20T19:30:17.583Z]
[redacted]

[2022-12-20T19:30:18.319Z] Executed 'FunctionServiceBusErrorQueue' (Succeeded, Id=4f8f2ada-0c85-4bbe-aae4-865d41f7e697, Duration=905ms)
[2022-12-20T19:30:56.145Z] Executing 'FunctionServiceBusDecreaseRateQueue' (Reason='(null)', Id=648e4e0d-0e52-4407-8bff-97318f370352)
[2022-12-20T19:30:56.150Z] Trigger Details: MessageId: 6646e61d7cc644c687356773394ae8cc, SequenceNumber: 2, DeliveryCount: 1, EnqueuedTimeUtc: 20
22-12-20T19:31:05.8920000+00:00, LockedUntilUtc: 2022-12-20T19:31:35.9070000+00:00, SessionId: (null)
[2022-12-20T19:30:56.175Z] C# ServiceBus queue trigger function processed message: {"time":"2022-12-20T16:30:00.0000000Z","deviceId":"device_3"}
[2022-12-20T19:30:56.178Z]
[redacted]

[2022-12-20T19:30:56.799Z] Executed 'FunctionServiceBusDecreaseRateQueue' (Succeeded, Id=648e4e0d-0e52-4407-8bff-97318f370352, Duration=665ms)
```