Projekt IoT

Połączenie z urządzeniem (serwerem OPC UA)

Aby połączyć się z serwerem OPC UA korzystamy z biblioteki (klienta OPC UA) **asyncua**. Tworzymy instancje klienta podając adres z pliku konfiguracyjnego *config.ini*.

Przykładowy fragment `config.ini`:

```
[opcua]
url = opc.tcp://localhost:4840/
```

Gdy połączenie zostanie nawiązane, pobieramy informacje na temat urządzeń, po czym tworzymy instancje klasy `Agent` dla każdego urządzenia. Klasa ta przyjmuje jako parametry obiekt `Node` odpowiadający danemu urządzeniu, klienta OPC UA (wszystko to z biblioteki **asyncua**) oraz `connection_string` potrzebny do połaczenia się z IoTHubem.

Na końcu dla wszystkich agentów w nieskończonej pętli wykonujemy wszystkie zakolejkowane zadania, które zostały ustawione przez metody klasy.

Konfiguracja agenta

Proces konfiguracji agenta polega na pobraniu jego ustawień z pliku konfiguracyjnego `config.ini`. W pliku tym znajduje się zapytany `connection_string` dla każdego urządzenia. W przypadku kiedy dla urządzenia w pliku konfiguracyjnym nie ma `connection_string`, użytkownik zostaje o niego zapytany, a odpowiedź zostaje zapisana.

Przykładowa konfiguracja:

```
[devices]
device 1 = <connection_string>
```

Tworzenie instancji Agenta:

```
agent = Agent(
  device=obj,
  opcua_client=client,
  connection_string=device_connection_string
)
```

Komunikacja D2C

Agent wysyła informacje do IoTHuba co jedną sekundę w przypadku telemetrii i w przypadku wystąpienia błędu z maksymalnie 200 milisekundowym opóźnieniem (subskrypcja sprawdza obserwowane parametry co 200ms).

Przykładowe dane telemetryczne:

```
{
  "body": {
    "WorkorderId": "62248319-3a00-46f0-b09c-193ad201d6c8",
    "ProductionStatus": 1,
    "GoodCount": 31,
    "BadCount": 2,
    "Temperature": 80.99897903327059,
    "message_type": "telemetry"
  },
  "enqueuedTime": "Sat Dec 31 2022 00:10:25 GMT+0100 (Central European Standard Time)",
  "properties": {}
}
```

Przykładowe dane na temat występującego błędu:

```
{
  "body": {
    "DeviceError": "Unknown",
    "message_type": "event"
  },
  "enqueuedTime": "Sat Dec 31 2022 00:10:25 GMT+0100 (Central European Standard Time)",
  "properties": {}
}
```

Device Twin

W device twin przechowywane są informacje na temat aktualnego tempa produkcji, aktualnie występujących błędów, czasu w jakim wystąpił ostatni błąd oraz czasu kiedy to został on naprawiony.

Dodatkowo w obiekcie `desired` możemy przekazać informacje na temat pożądanego tempa produkcji (pod kluczem `ProductionRate`), informacja ta zostanie pobrana przez urządzenie oraz zastosowana.

Przykładowy Device Twin:

```
{
    "deviceId": "device-1",
    "etag": "AAAAAAAAAAQ=",
    "deviceEtag": "MTA3MzM4Mjk1MQ==",
    "status": "enabled",
    "statusUpdateTime": "0001-01-01T00:00:00Z",
    "connectionState": "Disconnected",
    "lastActivityTime": "2022-12-30T17:40:20.8998558Z",
    "cloudToDeviceMessageCount": 0,
    "authenticationType": "sas",
    "x509Thumbprint": {
        "primaryThumbprint": null,
        "secondaryThumbprint": null
    },
    "modelId": "",
    "version": 28,
    "properties": {
        "desired": {
            "ProductionRate": 25,
            "$metadata": {
                "$lastUpdated": "2022-12-30T17:14:37.7163161Z",
                "$lastUpdatedVersion": 4,
                "ProductionRate": {
                    "$lastUpdated": "2022-12-30T17:14:37.7163161Z",
                    "$lastUpdatedVersion": 4
                }
            },
            "$version": 4
        },
        "reported": {
            "MaintenanceDone": "2022-12-30T18:20:40.772299",
            "ProductionRate": 40,
            "DeviceError": [
                "Sensor Failure",
                "Unknown"
            ],
            "LastErrorDate": "2022-12-31T00:10:28.887106",
            "$metadata": {
                "$lastUpdated": "2022-12-30T23:10:29.0472654Z",
                "MaintenanceDone": {
                    "$lastUpdated": "2022-12-30T17:20:40.8860736Z"
                },
                "ProductionRate": {
                    "$lastUpdated": "2022-12-30T23:10:19.9377722Z"
                },
                "DeviceError": {
                    "$lastUpdated": "2022-12-30T23:10:29.0472654Z"
                },
                "LastErrorDate": {
                    "$lastUpdated": "2022-12-30T23:10:28.9066495Z"
                }
            },
            "$version": 24
        }
    },
    "capabilities": {
        "iotEdge": false
    }
}
```

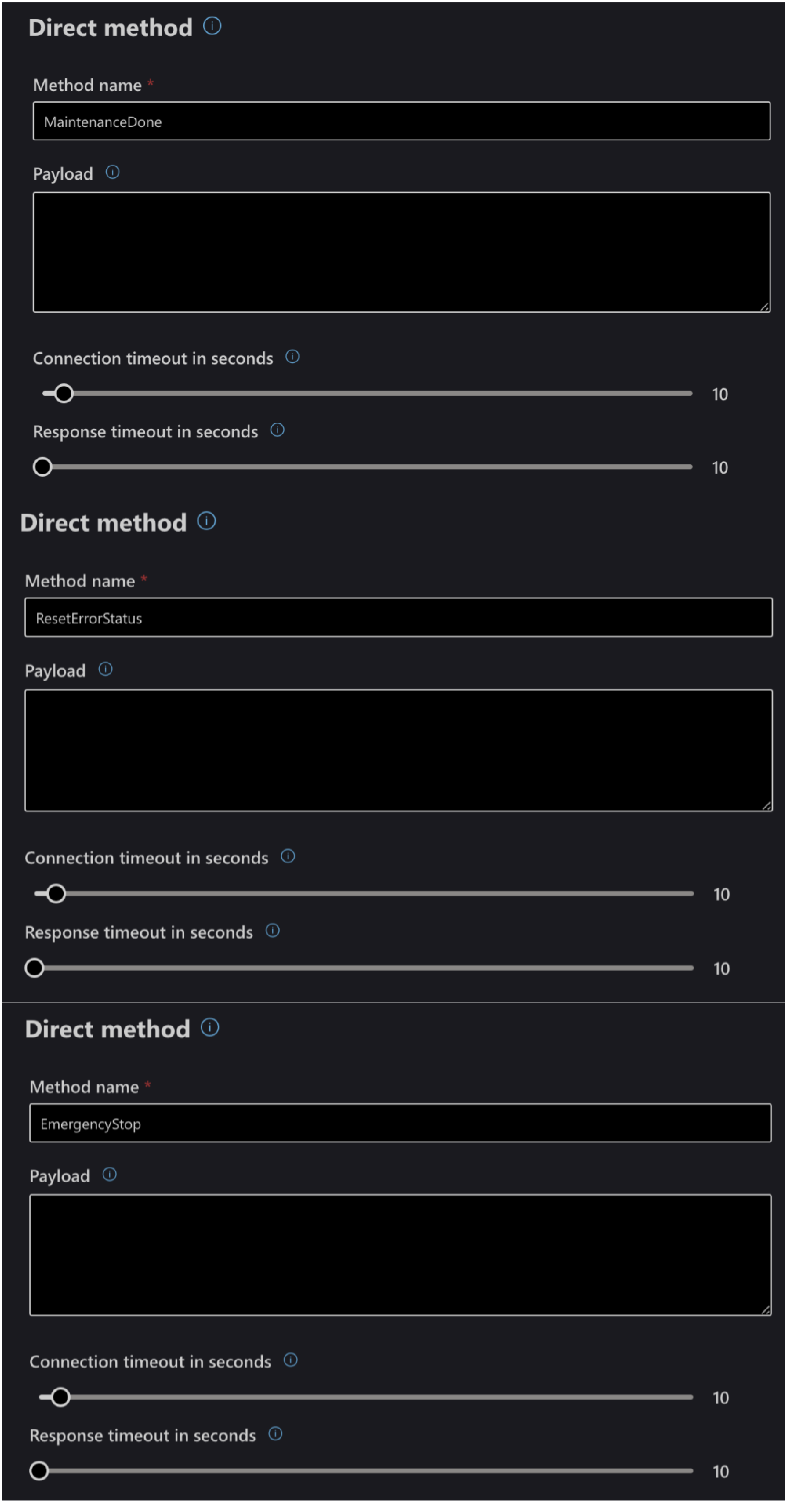
Direct Methods

Agent posiada zaimplementowane 3 metody:

- EmergencyStop
- ResetErrorStatus
- MaintenanceDone

Metody te nie potrzebują podawania żadnych argumentów oraz zwracają tylko wiadomość "OK" ze statusem 200. W przypadku próby wywołania nieistniejącej metody otrzymamy wiadomość "Nieznana metoda" z statusem 404.

Przykłady wywołania metod:



Direct method

Method name: MaintenanceDone

Connection timeout in seconds: 10

Response timeout in seconds: 10

Direct method

Method name: ResetErrorStatus

Connection timeout in seconds: 10

Response timeout in seconds: 10

Direct method

Method name: EmergencyStop

Connection timeout in seconds: 10

Response timeout in seconds: 10

Kalkulacje danych i logika biznesowa

Kalkulacja danych

Do kalkulacji wykorzystane zostały kwerendy z Azure Stream Analytics. Przy ich pomocy przetwarzamy dane pochodzące z danych telemetrycznych wysyłanych do IoT Huba przez Agenta.

Kwerendy użyte do kalkulacji:

```
-- produkcja per workorderId
SELECT
    WorkorderId,
    SUM(GoodCount) AS GoodCountSum,
    SUM(BadCount) AS BadCountSum
INTO [asa-out-production-counts]
FROM [asa-in] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    WorkorderId, TumblingWindow(minute , 15)

-- production kpi
SELECT
    (SUM(GoodCount) / (SUM(GoodCount) + SUM(BadCount))) AS ProductionKPI
INTO [asa-out-kpi]
FROM [asa-in] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    TumblingWindow(minute , 15)

-- minimalna, maksymalna i średnia temperatura
SELECT
    WorkorderId,
    AVG(Temperature) AS AvgTemp,
    MIN(Temperature) AS MinTemp,
    MAX(Temperature) AS MaxTemp
INTO [asa-out-machine-temperatures]
FROM [asa-in] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    WorkorderId, TumblingWindow(minute , 5)

-- błędy w 15 minutowym okienku
SELECT ih.IoTHub.ConnectionDeviceId, COUNT(message_type) as errors
INTO [asa-out-error-per-machine]
FROM [asa-in] ih TIMESTAMP by EventEnqueuedUtcTime
WHERE message_type = 'event'
GROUP BY
    message_type, ih.IoTHub.ConnectionDeviceId, TumblingWindow(minute , 15)
HAVING count(message_type) > 3
```

Logika biznesowa

Do wykonywania logiki biznesowej zostały wykorzystane kwerendy z Azure Stream Analytics w połączeniu z funkcjami z Function App (funkcje te są wyjściem dla wyników z kwerend).

Kwerendy odpowiadające za obliczanie danych dla funkcji:

```
--- awaryjne zatrzymanie dla funkcji
SELECT ih.IoTHub.ConnectionDeviceId, COUNT(message_type) as errors
INTO [asa-out-emergency-stop-http-trigger]
FROM [asa-in] ih TIMESTAMP by EventEnqueuedUtcTime
WHERE message_type = 'event'
GROUP BY
    message_type, ih.IoTHub.ConnectionDeviceId, TumblingWindow(minute , 15)

-- production kpi dla funkcji
SELECT
    (SUM(GoodCount) / (SUM(GoodCount) + SUM(BadCount))) AS kpi,
    System.Timestamp() AS WindowEndTime
INTO [asa-out-production-kpi-http-trigger]
FROM [asa-in] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    TumblingWindow(minute , 15)
```

Funkcje po otrzymaniu danych i zajściu wyznaczonych warunków wywołują dostępne dla urządzenia metody poprzez C2D.