System monitorowania produkcji przemysłowej IoT

Dokumentacja projektowa

Autor:

Bartosz Gruda

May 23, 2025

Contents

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest implementacja systemu monitorowania produkcji przemysłowej opartego na technologiach Internet of Things (IoT). System umożliwia zbieranie danych telemetrycznych z urządzeń przemysłowych za pomocą protokołu OPC UA, przetwarzanie ich w chmurze Microsoft Azure oraz analizę w czasie rzeczywistym z wykorzystaniem Azure Stream Analytics.

1.2 Zakres projektu

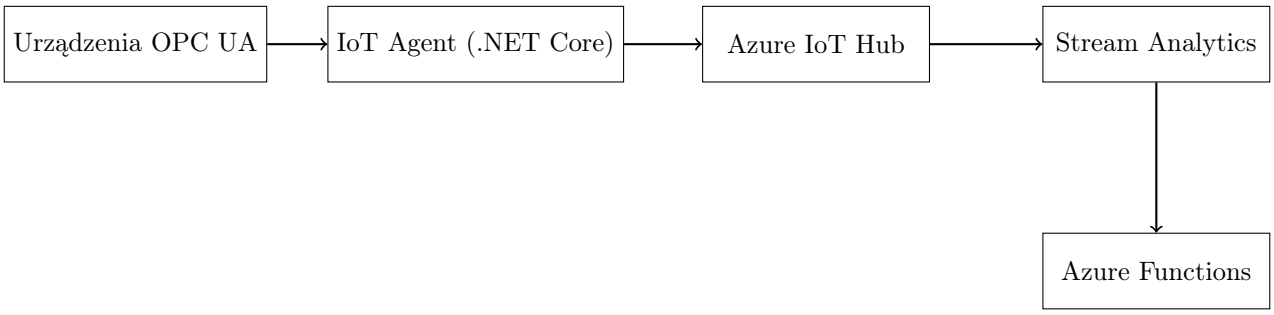
Projekt obejmuje następujące komponenty:

- Agent IoT (`IndustrialIoT.Agent`) - aplikacja .NET Core komunikująca się z urządzeniami OPC UA
- Funkcje Azure (`IndustrialIoT.Functions`) - logika biznesowa przetwarzania danych w chmurze
- Zapytania Stream Analytics - przetwarzanie danych w czasie rzeczywistym
- Integracja z Azure IoT Hub jako bramą do chmury

2 Architektura systemu

2.1 Schemat architektury

System składa się z następujących komponentów połączonych w poniższy sposób:



2.2 Opis komponentów

2.2.1 Urządzenia OPC UA

System obsługuje dowolne urządzenia przemysłowe zgodne ze standardem OPC UA. Urządzenia te mogą być fizycznymi sterownikami PLC, czujnikami, lub symulatorami OPC UA używanymi w środowisku testowym.

2.2.2 IndustrialIoT.Agent

Agent IoT to aplikacja .NET Core, która:

- Nawiązuje połączenia z urządzeniami OPC UA
- Zbiera dane telemetryczne (temperatura, liczniki produkcji, błędy)
- Przesyła dane do Azure IoT Hub w bezpieczny sposób
- Obsługuje konfigurację z pliku `AppSettings.json`

2.2.3 Azure IoT Hub

Azure IoT Hub pełni rolę centralnej bramy zarządzającej komunikacją:

- Obsługuje komunikację dwukierunkową między urządzeniami a chmurą
- Zapewnia bezpieczną komunikację za pomocą protokołu MQTT/HTTPS
- Zarządza tożsamościami urządzeń
- Umożliwia routing wiadomości do innych serwisów Azure

2.2.4 Stream Analytics

Azure Stream Analytics przetwarza dane w czasie rzeczywistym:

- Oblicza kluczowe wskaźniki wydajności (KPI) produkcji
- Monitoruje statystyki temperatury
- Wykrywa błędy urządzeń i alarmuje o anomaliach

2.2.5 IndustrialIoT.Functions

Azure Functions implementują logikę biznesową:

- Przetwarzają wiadomości z IoT Hub
- Implementują analizy danych
- Udostępniają API do pobierania zagregowanych informacji

3 Implementacja komponentów

3.1 IndustrialIoT.Agent

3.1.1 Struktura projektu

Projekt agenta składa się z następujących kluczowych plików:

- `Program.cs` - punkt wejścia aplikacji
- `IoTDevice.cs` - klasa bazowa dla urządzeń IoT
- `OpcDevice.cs` - implementacja urządzenia OPC UA
- `AppSettings.json` - plik konfiguracyjny

3.1.2 Klasa IoTDevice

Klasa `IoTDevice` jest klasą bazową dla wszystkich urządzeń IoT w systemie. Implementuje podstawową funkcjonalność:

```
public abstract class IoTDevice
{
    protected string DeviceId { get; set; }
    protected string ConnectionString { get; set; }
    protected DeviceClient DeviceClient { get; set; }

    public IoTDevice(string deviceId, string connectionString)
    {
        DeviceId = deviceId;
        ConnectionString = connectionString;
        DeviceClient = DeviceClient.CreateFromConnectionString(ConnectionString);
    }

    public abstract Task Initialize();
    public abstract Task SendTelemetryAsync();
    public abstract Task ReceiveCommands();
}
```

Listing 1: IoTDevice.cs (fragment)

3.1.3 Klasa OpcDevice

Klasa `OpcDevice` rozszerza klasę `IoTDevice` i implementuje komunikację z urządzeniami OPC UA:

```
public class OpcDevice : IoTDevice
{
    private OpcClient Client { get; set; }
    private string EndpointUrl { get; set; }
    private string NodeId { get; set; }

```

```
7     public OpcDevice(string deviceId, string connectionString,
8                      string endpointUrl, string nodeId)
9         : base(deviceId, connectionString)
10    {
11        EndpointUrl = endpointUrl;
12        NodeId = nodeId;
13    }
14
15    public override async Task Initialize()
16    {
17        Client = new OpcClient(EndpointUrl);
18        await Client.Connect();
19        // Inicjalizacja urzadzenia OPC
20    }
21
22    public override async Task SendTelemetryAsync()
23    {
24        // Odczyt danych z OPC UA
25        var temperatureValue = await Client.ReadNodeAsync(NodeId + "/Temperature");
26        var goodCountValue = await Client.ReadNodeAsync(NodeId + "/GoodCount");
27        var badCountValue = await Client.ReadNodeAsync(NodeId + "/BadCount");
28        var statusValue = await Client.ReadNodeAsync(NodeId + "/Status");
29
30        // Przygotowanie telemetrii
31        var telemetryData = new
32        {
33            temperature = (double)temperatureValue,
34            goodCount = (int)goodCountValue,
35            badCount = (int)badCountValue,
36            productionStatus = statusValue.ToString(),
37            deviceErrors = GetDeviceErrors()
38        };
39
40        // Wyslanie telemetrii do IoT Hub
41        var messageString = JsonConvert.SerializeObject(telemetryData);
42        var message = new Message(Encoding.UTF8.GetBytes(messageString));
43        await DeviceClient.SendEventAsync(message);
44    }
45
46    // Pozostale metody implementacyjne
47 }
```

Listing 2: OpcDevice.cs (fragment)

3.2 IndustrialIoT.Functions

3.2.1 Struktura projektu

Projekt Azure Functions zawiera następujące kluczowe pliki:

- `Models.cs` - modele danych dla przetwarzania wiadomości IoT
- `AzureIoTService.cs` - usługa integracji z Azure IoT Hub
- `BusinessLogicFunctions.cs` - funkcje zawierające logikę biznesową
- `Startup.cs` - konfiguracja aplikacji
- `host.json` - konfiguracja hosta Functions
- `local.settings.json` - lokalne ustawienia dla środowiska deweloperskiego

3.2.2 Modele danych

Plik `Models.cs` definiuje struktury danych używane w systemie:

```
1 namespace IndustrialIoT.Functions
2 {
3     public class TelemetryMessage
4     {
```

```
        public string DeviceId { get; set; }
        public DateTime Timestamp { get; set; }
        public double Temperature { get; set; }
        public int GoodCount { get; set; }
        public int BadCount { get; set; }
        public string ProductionStatus { get; set; }
        public string DeviceErrors { get; set; }
    }

    public class ProductionKpi
    {
        public string DeviceId { get; set; }
        public double QualityPercentage { get; set; }
        public int TotalGood { get; set; }
        public int TotalBad { get; set; }
        public DateTime WindowEnd { get; set; }
    }

    public class TemperatureStats
    {
        public string DeviceId { get; set; }
        public double AverageTemperature { get; set; }
        public double MinTemperature { get; set; }
        public double MaxTemperature { get; set; }
        public DateTime WindowEnd { get; set; }
    }

    public class DeviceError
    {
        public string DeviceId { get; set; }
        public int ErrorCount { get; set; }
        public DateTime WindowEnd { get; set; }
    }
}
```

Listing 3: Models.cs (fragment)

3.2.3 Integracja z Azure IoT Hub

Plik `AzureIoTService.cs` zawiera implementację usługi do komunikacji z Azure IoT Hub:

```
public class AzureIoTService
{
    private readonly ServiceClient _serviceClient;
    private readonly RegistryManager _registryManager;

    public AzureIoTService(string connectionString)
    {
        _serviceClient = ServiceClient.CreateFromConnectionString(connectionString);
        _registryManager = RegistryManager.CreateFromConnectionString(connectionString);
    }

    public async Task<Device> GetDeviceAsync(string deviceId)
    {
        return await _registryManager.GetDeviceAsync(deviceId);
    }

    public async Task SendCommandAsync(string deviceId, string commandName, object payload
    )
    {
        var commandMessage = new Message(Encoding.UTF8.GetBytes(JsonConvert.
    SerializeObject(payload)));
        await _serviceClient.SendAsync(deviceId, commandMessage);
    }

    // Pozostale metody do zarzadzania urzadzeniami i wiadomosciami
}
```

Listing 4: AzureIoTService.cs (fragment)

3.2.4 Funkcje biznesowe

Plik `BusinessLogicFunctions.cs` zawiera implementację funkcji Azure obsługujących logikę biznesową:

```
public class BusinessLogicFunctions
{
    private readonly AzureIoTService _ioTService;

    public BusinessLogicFunctions(AzureIoTService ioTService)
    {
        _ioTService = ioTService;
    }

    [FunctionName("ProcessTelemetry")]
    public async Task ProcessTelemetry(
        [IoTHubTrigger("messages/events", Connection = "IoTHubConnection")]
        EventData message,
        ILogger log)
    {
        try
        {
            string messageBody = Encoding.UTF8.GetString(message.Body.Array);
            var telemetry = JsonConvert.DeserializeObject<TelemetryMessage>(messageBody);

            log.LogInformation($"Otrzymano telemetrie z urzadzenia: {telemetry.DeviceId}")
    ;

            // Przetwarzanie danych telemetrycznych
            // ...

            // Zapisanie danych w bazie danych
            // ...
        }
        catch (Exception ex)
        {
            log.LogError($"Blad przetwarzania telemetrii: {ex.Message}");
        }
    }

    [FunctionName("GetDeviceStatus")]
    public async Task<IActionResult> GetDeviceStatus(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = "devices/{deviceId}/
    status")]
        HttpRequest req,
        string deviceId,
        ILogger log)
    {
        try
        {
            var device = await _ioTService.GetDeviceAsync(deviceId);
            if (device == null)
            {
                return new NotFoundResult();
            }

            // Pobranie statusu urzadzenia
            // ...

            return new OkObjectResult(deviceStatus);
        }
        catch (Exception ex)
        {
            log.LogError($"Blad pobierania statusu urzadzenia: {ex.Message}");
            return new StatusCodeResult(StatusCodes.Status500InternalServerError);
        }
    }

    // Pozostale funkcje API i obslugi zdarzen
```

```
63 }
```

Listing 5: BusinessLogicFunctions.cs (fragment)

3.3 Stream Analytics

3.3.1 Zapytania

Plik `stream_analitics_kwerendy.txt` zawiera zapytania używane w Azure Stream Analytics do przetwarzania danych w czasie rzeczywistym:

Listing 6: Zapytanie obliczające KPI produkcji

```
-- Production KPIs
SELECT
    deviceId,
    CASE
        WHEN SUM(CAST(goodCount AS float)) + SUM(CAST(badCount AS float)) > 0
        THEN (SUM(CAST(goodCount AS float)) * 100.0 / (SUM(CAST(goodCount AS float)) + SUM(CA
        ELSE 100.0
    END AS prod_kpi,
    SUM(CAST(goodCount AS float)) AS total_good,
    SUM(CAST(badCount AS float)) AS total_bad,
    System.Timestamp() AS windowEnd
INTO
    kpi-output
FROM
    "iot-hub"
WHERE
    productionStatus = 'Running' OR productionStatus = '1'
GROUP BY
    deviceId,
    TumblingWindow(minute, 5)
HAVING
    SUM(CAST(goodCount AS float)) + SUM(CAST(badCount AS float)) > 0;
```

Listing 7: Zapytanie monitorujące statystyki temperatury

```
-- Temperature Stats
SELECT
    deviceId,
    AVG(temperature) AS avg_t,
    MIN(temperature) AS min_t,
    MAX(temperature) AS max_t,
    System.Timestamp() AS windowEnd
INTO
    temperature-stats
FROM
    "iot-hub"
GROUP BY
    deviceId,
    SlidingWindow(minute, 5)
HAVING
    COUNT(*) > 0;
```

Listing 8: Zapytanie wykrywające błędy urządzeń

```
-- Device Errors
SELECT
    deviceId,
    COUNT(*) AS errorCount,
    System.Timestamp() AS windowEnd
INTO
    error-data
FROM
    "iot-hub"
```

```
WHERE
    deviceErrors IS NOT NULL AND deviceErrors <> '0' AND deviceErrors <> 'None'
    OR (errorCode IS NOT NULL AND errorCode > 0)
GROUP BY
    deviceId,
    TumblingWindow(minute, 1)
HAVING
    COUNT(*) > 3;
```

4 Bezpieczeństwo systemu

4.1 Mechanizmy uwierzytelniania

System wykorzystuje następujące mechanizmy bezpieczeństwa do uwierzytelniania urządzeń i komunikacji:

- Sygnatury SAS (Shared Access Signatures) do bezpiecznego uwierzytelniania urządzeń w Azure IoT Hub
- Certyfikaty X.509 dla komunikacji z urządzeniami OPC UA
- Azure Active Directory do uwierzytelniania użytkowników w aplikacjach zaplecza

4.2 Zabezpieczenie transmisji danych

Komunikacja między komponentami systemu jest zabezpieczona następującymi metodami:

- Szyfrowanie TLS 1.2 lub nowsze dla wszystkich połączeń sieciowych
- Bezpieczne tunele komunikacyjne dla urządzeń za zaporami sieciowymi
- Szyfrowanie przechowywanych danych w Azure Storage i Cosmos DB

4.3 Zarządzanie tożsamościami urządzeń

Azure IoT Hub zapewnia:

- Unikalną tożsamość dla każdego urządzenia
- Możliwość zdalnego wyłączenia skompromitowanych urządzeń
- Rotację kluczy dostępowych dla zwiększenia bezpieczeństwa

5 Konfiguracja i wdrożenie

5.1 Wymagania wstępne

Do uruchomienia systemu potrzebne są:

- .NET 6.0 SDK lub nowszy
- Subskrypcja Azure
- Instancja Azure IoT Hub
- Zadanie Stream Analytics
- Urządzenia zgodne z OPC UA lub symulatory

5.2 Konfiguracja lokalnego środowiska

5.2.1 Konfiguracja agenta IoT

Aby skonfigurować agenta IoT, należy zaktualizować plik `AppSettings.json`:

```
{
  "IoTHubConnectionString": "HostName=your-iot-hub.azure-devices.net;DeviceId=your-device-id;SharedAccessKey=your-access-key",
  "DeviceId": "device1",
  "OpcUaEndpoint": "opc.tcp://localhost:4840",
  "OpcUaNodeId": "ns=2;s=Machine1",
  "TelemetryInterval": 1000,
  "LogLevel": "Information"
}
```

Listing 9: AppSettings.json

5.2.2 Konfiguracja Azure Functions

Aby skonfigurować Azure Functions, należy zaktualizować plik `local.settings.json`:

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
    "IoTHubConnection": "HostName=your-iot-hub.azure-devices.net;SharedAccessKeyName=service;SharedAccessKey=your-access-key",
    "CosmosDBConnection": "AccountEndpoint=https://your-account.documents.azure.com:443/;AccountKey=your-account-key;"
  }
}
```

Listing 10: local.settings.json

5.3 Uruchomienie lokalne

5.3.1 Uruchomienie agenta IoT

Aby uruchomić agenta IoT lokalnie:

```
cd IndustrialIoT.Agent
dotnet run
```

5.3.2 Uruchomienie Azure Functions

Aby uruchomić Azure Functions lokalnie:

```
cd IndustrialIoT.Functions
func start
```

5.4 Wdrożenie na Azure

5.4.1 Wdrożenie Azure Functions

Aby wdrożyć Azure Functions:

```
cd IndustrialIoT.Functions
func azure functionapp publish <nazwa-aplikacji-funkcji>
```

5.4.2 Konfiguracja Stream Analytics

Aby skonfigurować Stream Analytics:

1. Utwórz nowe zadanie Stream Analytics w portalu Azure
2. Skonfiguruj wejście ze źródła IoT Hub
3. Skonfiguruj wyjścia dla danych KPI, statystyk temperatury i błędów urządzeń
4. Dodaj zapytania z pliku `stream_analitics_kwerendy.txt`
5. Uruchom zadanie

5.4.3 Wdrożenie agenta IoT

Aby wdrożyć agenta IoT na urządzenie brzegowe:

1. Opublikuj aplikację jako samodzielny plik wykonywalny
2. Skopiuj pliki na urządzenie brzegowe
3. Zaktualizuj `AppSettings.json` odpowiednimi wartościami konfiguracyjnymi
4. Uruchom aplikację

6 Bibliografia

1. Microsoft Azure IoT Hub Documentation, `https://docs.microsoft.com/en-us/azure/iot-hub/`
2. OPC Foundation, OPC UA Specification, `https://opcfoundation.org/about/opc-technologies/opc-ua/`
3. Microsoft Azure Stream Analytics Documentation, `https://docs.microsoft.com/en-us/azure/stream-analytics/`
4. Microsoft Azure Functions Documentation, `https://docs.microsoft.com/en-us/azure/azure-functions/`