Dokumentacja techniczna systemu pobierania danych pogodowych

Oracle + .NET 8 (Minimal API + Worker Service)

CerbIT / System pogodowy

20 listopada 2025

1 Wprowadzenie

Celem projektu jest automatyczne pobieranie danych pogodowych z darmowego API (Open-Meteo) oraz zapisywanie ich do bazy danych Oracle w sposób bezpieczny i łatwy do monitorowania.

Aktualna architektura składa sie z:

- aplikacji .NET 8 uruchamianej jako Windows Service z wbudowanym serwerem HTTP (Kestrel) w trybie **Minimal API + Worker Service**,
- lokalnego endpointu HTTP pełniacego role **proxy pogodowego** dla bazy Oracle,
- procedur PL/SQL i joba DBMS_SCHEDULER wyzwalajacych pobranie danych i zapis do tabel,
- bazy danych Oracle (schemat `TMYSZAK`) z tabelami `WEATHER_LOCATIONS` oraz `WEATHER_MEASUREMENT`

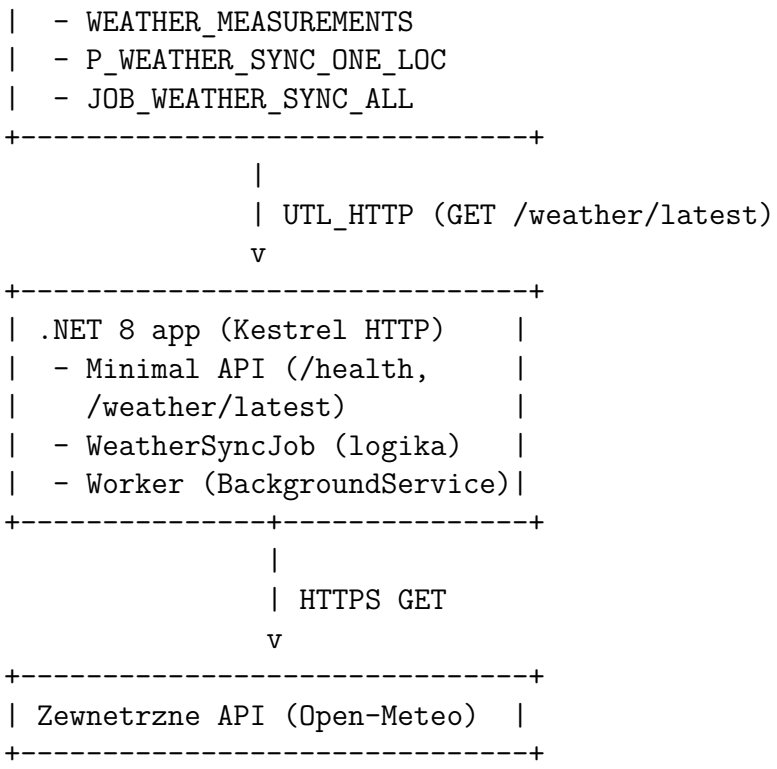
Główny przepływ produkcyjny jest sterowany po stronie Oracle:

1. Job `JOB_WEATHER_SYNC_ALL` w Oracle uruchamia cyklicznie procedure PL/SQL.
2. Procedura PL/SQL (UTL_HTTP) wywołuje lokalny endpoint HTTP w serwisie .NET: `/weather/latest?lat=...&lon=...`.
3. Serwis .NET pobiera dane z API Open-Meteo, przetwarza JSON do prostego DTO i zwraca mały JSON.
4. PL/SQL parsuje JSON (JSON_TABLE) i zapisuje pomiar do `TMYSZAK.WEATHER_MEASUREMENTS`.

2 Architektura systemu

2.1 Schemat logiczny

```
+------------------------------+
| Oracle Database (schema TMYSZAK)
|  - WEATHER_LOCATIONS
```



2.2 Opis komponentów

- **Minimal API (.NET 8)**

  Aplikacja nasłuchuje na porcie HTTP (domyślnie `http://0.0.0.0:5005`) i udostepnia:

  - `GET /health` – prosty endpoint zdrowia (test połaczenia z serwisem),
  - `GET /weather/latest?lat={lat}&lon={lon}` – główny endpoint proxy dla Oracle.

- **WeatherSyncJob**

  Klasa serwisowa odpowiedzialna za:

  - test połaczeń z Oracle i Open-Meteo (tryb testowy),
  - generowanie testowych danych pogodowych i zapis do pliku + bazy (`RunOnce`),
  - pobieranie rzeczywistych danych z Open-Meteo dla zadanych współrzednych,
  - parsowanie JSON z Open-Meteo do wewnetrznego modelu,
  - zwracanie uproszczonego DTO `WeatherDto` do endpointu `/weather/latest`,
  - wykonywanie INSERT do `TMYSZAK.WEATHER_MEASUREMENTS`.

- **Worker (BackgroundService)**

  Usługa tła rejestrowana w Host Builderze. W projekcie pełni role:

  - cyklicznego joba testowego (wywołanie `RunOnce` co 10 minut),
  - mechanizmu sanity-check / diagnostycznego,

– komponentu reagujacego na sygnały zatrzymania i logujacego stan serwisu.

W scenariuszu produkcyjnym głównym sterownikiem pobierania danych jest Oracle (PL/SQL + UTL_HTTP), a Worker może być dodatkowym mechanizmem testowym.

- **PL/SQL (procedury i job)**
  - `P_WEATHER_SYNC_ONE_LOC(p_id_location)` – dla jednej lokalizacji: odczytuje LAT/LON, woła `/weather/latest`, parsuje JSON, wstawia wiersz do `WEATHER_MEASUREMENTS`
  - `JOB_WEATHER_SYNC_ALL` – job DBMS_SCHEDULER, który co 10 minut iteruje po aktywnych lokalizacjach i dla każdej wywołuje `P_WEATHER_SYNC_ONE_LOC`.
  - `P_TEST_WEATHER_PROXY` – procedura testowa wywołujaca `/health` w celu sprawdzenia dostepności serwisu .NET.

3 Struktura bazy danych Oracle

3.1 Tabela WEATHER_LOCATIONS

```
CREATE TABLE TMYSZAK.WEATHER_LOCATIONS (
    ID_LOCATION       NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY
       ,
    COUNTRY_CODE      VARCHAR2(2) NOT NULL,
    CITY_NAME         VARCHAR2(100) NOT NULL,
    LATITUDE          NUMBER(9,6) NOT NULL,
    LONGITUDE         NUMBER(9,6) NOT NULL,
    ACTIVE_FLAG       CHAR(1) DEFAULT 'Y' NOT NULL
);
```

3.2 Tabela WEATHER_MEASUREMENTS

```
CREATE TABLE TMYSZAK.WEATHER_MEASUREMENTS (
    ID_MEASUREMENT    NUMBER GENERATED AS IDENTITY PRIMARY KEY,
    ID_LOCATION       NUMBER NOT NULL,
    MEASURED_AT       DATE NOT NULL,
    TEMP_C            NUMBER(5,2),
    IS_RAIN           CHAR(1),
    HUMIDITY          NUMBER(5,2),
    WIND_SPEED_MS     NUMBER(6,2),
    RAW_JSON          CLOB,
    INSERTED_AT       DATE DEFAULT SYSDATE NOT NULL,
    CONSTRAINT FK_LOC FOREIGN KEY (ID_LOCATION)
       REFERENCES TMYSZAK.WEATHER_LOCATIONS(ID_LOCATION)
);
```

4 Kontrakt JSON pomiedzy .NET a Oracle

Endpoint `GET /weather/latest` zwraca mały JSON (DTO) w formacie:

```
{
  "measuredAt": "2025-11-20T09:00:00Z",
  "tempC": 4.5,
  "humidity": 73.2,
  "windSpeedMs": 1.8,
  "isRain": false
}
```

Pole `isRain` jest konwertowane w PL/SQL na `'Y'` / `'N'` i zapisywane w kolumnie `IS_RAIN`. Cały JSON lub jego fragment może być również przechowywany w kolumnie `RAW_JSON`.

5 Konfiguracja systemu (.NET)

5.1 Plik appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "Oracle": "User Id=TMYSZAK;Password=***;Data Source=HOST:1521/
       SERVICE;"
  },
  "WeatherApi": {
    "BaseUrl": "https://api.open-meteo.com/v1/forecast",
    "Params": "hourly=temperature_2m,relativehumidity_2m,precipitation,
       windspeed_10m&forecast_days=1&timezone=auto"
  }
}
```

6 Implementacja aplikacji .NET 8 (Minimal API + Worker)

6.1 Program.cs – konfiguracja hosta i endpointów

```
var builder = WebApplication.CreateBuilder(args);

// Nas uch na wszystkich interfejsach, port 5005
builder.WebHost.UseUrls("http://0.0.0.0:5005");

// Logowanie na konsol
builder.Logging.ClearProviders();
builder.Logging.AddConsole();
builder.Logging.SetMinimumLevel(LogLevel.Information);

// Rejestracja serwis w
builder.Services.AddSingleton<WeatherSyncJob>();
```

```
builder.Services.AddHostedService<Worker>();

var app = builder.Build();

// /health
app.MapGet("/health", () => Results.Ok("OK"));

// /weather/latest?lat=..&lon=..
app.MapGet("/weather/latest",
    async (double lat,
           double lon,
           WeatherSyncJob job,
           CancellationToken token) =>
{
    // walidacja parametr w
    if (lat is < -90 or > 90)
        return Results.BadRequest("Nieprawid owa␣szeroko ␣
            geograficzna.");
    if (lon is < -180 or > 180)
        return Results.BadRequest("Nieprawid owa␣d ugo ␣
            geograficzna.");

    var dto = await job.GetLatestMeasurementDtoAsync(lat, lon, token);
    if (dto is null)
    {
        return Results.Problem(
            detail: "Brak␣danych␣z␣serwisu␣pogodowego.",
            statusCode: StatusCodes.Status502BadGateway,
            title: "B   d␣proxy␣pogodowego");
    }

    return Results.Ok(dto);
});

app.Run();
```

6.2 Worker.cs – cykliczny job testowy

```
public sealed class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;
    private readonly WeatherSyncJob _weatherSyncJob;

    public Worker(ILogger<Worker> logger, WeatherSyncJob weatherSyncJob)
    {
        _logger = logger;
        _weatherSyncJob = weatherSyncJob;
    }

    protected override async Task ExecuteAsync(CancellationToken
        stoppingToken)
    {
        _logger.LogInformation("WeatherService␣Worker:␣started␣at␣{Time}
            ", DateTime.Now);

        while (!stoppingToken.IsCancellationRequested)
```

```
        {
            try
            {
                await _weatherSyncJob.RunOnce(stoppingToken);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "B    d podczas wykonywania
                    WeatherSyncJob.");
            }

            _logger.LogInformation(
                "WeatherService Worker: oczekiwanie 10 minut do
                    kolejnego cyklu.");

            try
            {
                await Task.Delay(TimeSpan.FromMinutes(10), stoppingToken
                    );
            }
            catch (TaskCanceledException)
            {
                break;
            }
        }

        _logger.LogInformation("WeatherService Worker: stopped at {Time}
            ", DateTime.Now);
    }
}
```

7 Logika synchronizacji – WeatherSyncJob

7.1 Pobieranie i parsowanie danych z Open-Meteo

```
public async Task<WeatherDto?> GetLatestMeasurementDtoAsync(
    double latitude,
    double longitude,
    CancellationToken cancellationToken)
{
    string? baseUrl = _configuration["WeatherApi:BaseUrl"];
    string? commonParms = _configuration["WeatherApi:Params"];

    if (string.IsNullOrWhiteSpace(baseUrl))
    {
        _logger.LogError("Brak konfiguracji WeatherApi:BaseUrl");
        return null;
    }

    var sb = new StringBuilder();
    sb.Append(baseUrl);
    sb.Append("?latitude=");
    sb.Append(latitude.ToString(CultureInfo.InvariantCulture));
    sb.Append("&longitude=");
    sb.Append(longitude.ToString(CultureInfo.InvariantCulture));
```

```
    if (!string.IsNullOrWhiteSpace(commonParms))
    {
        sb.Append('&');
        sb.Append(commonParms);
    }

    using var http = new HttpClient { Timeout = TimeSpan.FromSeconds(10)
        };

    HttpResponseMessage resp;
    try
    {
        resp = await http.GetAsync(sb.ToString(), cancellationToken);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "B    d przy wywo aniu serwisu pogodowego.
            ");
        return null;
    }

    if (!resp.IsSuccessStatusCode)
    {
        _logger.LogError(
            "Weather API zwr ci status {Code} {Reason}",
            (int)resp.StatusCode,
            resp.ReasonPhrase);
        return null;
    }

    var rawJson = await resp.Content.ReadAsStringAsync(cancellationToken
        );
    var measurement = ParseLatestMeasurement(rawJson);
    if (measurement is null)
    {
        _logger.LogWarning("Nie uda o si sparsowa JSON z Open-Meteo
            .");
        return null;
    }

    return new WeatherDto(
        MeasuredAt: measurement.MeasuredAt,
        TempC: measurement.TempC,
        Humidity: measurement.Humidity,
        WindSpeedMs: measurement.WindSpeedMs,
        IsRain: measurement.IsRain);
}
```

7.2 Insert do TMYSZAK.WEATHER_MEASUREMENTS

```
INSERT INTO TMYSZAK.WEATHER_MEASUREMENTS
    (ID_LOCATION, MEASURED_AT, TEMP_C, IS_RAIN,
     HUMIDITY, WIND_SPEED_MS, RAW_JSON)
VALUES
    (:p_loc, :p_time, :p_temp, :p_rain,
```

```
    :p_hum, :p_wind, :p_json);
```

8 Procedury PL/SQL i scheduler

8.1 P_WEATHER_SYNC_ONE_LOC

Procedura:

- odczytuje `LATITUDE`, `LONGITUDE` z `TMYSZAK.WEATHER_LOCATIONS`,
- woła `http://127.0.0.1:5005/weather/latest?lat=...&lon=...` za pomoca UTL_HTTP,
- odbiera JSON do CLOB,
- parsuje JSON_TABLE do zmiennych PL/SQL,
- mapuje `isRain` → `'Y'/'N'`,
- wykonuje INSERT do `TMYSZAK.WEATHER_MEASUREMENTS`,
- na końcu wykonuje `COMMIT`.

8.2 JOB_WEATHER_SYNC_ALL

Job DBMS_SCHEDULER:

- uruchamiany co 10 minut,
- iteruje po `WEATHER_LOCATIONS` z `ACTIVE_FLAG = 'Y'`,
- dla każdej lokalizacji wywołuje `P_WEATHER_SYNC_ONE_LOC(id_location)`.

8.3 P_TEST_WEATHER_PROXY i ACL

- `P_TEST_WEATHER_PROXY` wywołuje `/health` i wypisuje odpowiedź w `DBMS_OUTPUT`.
- ACL z użyciem `DBMS_NETWORK_ACL_ADMIN.append_host_ace` pozwala schematowi `TMYSZAK` na HTTP do hosta `127.0.0.1`.

9 Instalacja aplikacji .NET jako usługa Windows

9.1 Publikacja projektu

```
dotnet publish -c Release -r win-x64 --self-contained false -o publish
```

9.2 Rejestracja usługi

```
sc create WeatherService binPath= "C:\Weather\WeatherService.exe"
sc start WeatherService
```

Aplikacja po uruchomieniu wystawia port HTTP (np. 5005) dla wywołań z Oracle.

10 Podsumowanie

Przygotowany system umożliwia:

- automatyczne i cykliczne pobieranie danych pogodowych poprzez DBMS_SCHEDULER w Oracle,
- wykorzystanie lekkiego serwisu .NET jako proxy HTTP do zewnetrznego API pogodowego,
- stabilne działanie jako usługa Windows z logowaniem i Worker Service,
- pełna integracje z Oracle (ODP.NET, UTL_HTTP, JSON_TABLE),
- łatwe dodawanie nowych lokalizacji (tabela `WEATHER_LOCATIONS`) i rozszerzeń logiki po obu stronach (C# / PL/SQL).

System jest gotowy do użycia produkcyjnego po uzupełnieniu:

- spójnego logowania (np. Serilog, logi do plików / tabel),
- polityki retry/fallback przy błedach API i sieci,
- monitoringu endpointów (`/health`, `/weather/latest`) oraz jobów DBMS_SCHEDULER.