Zadanie 2:
Metody komunikacji w systemach Internetu Rzeczy

dr inż. Łukasz Falas & dr inż. Patryk Schauer

Wprowadzenie

W systemach Internetu Rzeczy możemy wyróżnić dwa główne modele komunikacji pomiędzy usługami. Pierwszym z nich jest komunikacja synchroniczna, najczęściej wykorzystywana w procesie sterowania lub pobierania danych w trybie „na żądanie”. Komunikacja synchroniczna w tego typu systemach najczęściej jest realizowana z wykorzystaniem usług budowanych zgodnie z konwencją REST, w przypadku, których komunikacja jest oparta na protokole http(s). W tym modelu jedna ze stron pełni rolę serwera, który nasłuchuje żądań i na nie odpowiada, natomiast druga strona pełni rolę klienta, który wysyła żądania do serwera. Jest to klasyczny model komunikacji *klient-serwer* (*ang. client-server*).

Drugim z modeli komunikacji jest komunikacja asynchroniczna. W tym modelu źródło danych przesyła komunikaty do kolejki (najczęściej udostępnianej przez brokera komunikatów), która najczęściej jest identyfikowana przez *temat* będący jej unikalnym identyfikatorem. W tym modelu strona, która przesyła dane nie wie, kto będzie ich odbiorcą, komunikuje się jedynie z brokerem, do którego *publikuje* komunikaty. Konsument z kolei też nie wchodzi w bezpośrednią interakcję ze stroną publikującą. Jedyne co robi to nawiązuje komunikację z brokerem, wskazuje identyfikator kolejki (*temat*) i ją subskrybuje, zostając *subskrybentem*. Ten model asynchronicznej komunikacji najczęściej jest nazywany modelem *publikuj-subskrybuj* (*ang. publish-subscribe*). Model ten jest najczęściej wykorzystywany do strumieniowego przesyłania danych (np. regularnego przesyłania danych pomiarowych o temperaturze) oraz do komunikacji, w której nie mamy pewności, że obie strony biorące udział w komunikacji (publikujący lub subskrybent) są włączone lub mają dostęp do sieci (w tym przypadku komunikaty będą czekały w kolejce, niezależnie od stanu obu stron).

Zadanie realizowane w ramach tego laboratorium ma na celu zapoznanie z tymi dwoma modelami komunikacji, poprzez ich praktyczne wdrożenie w formie programu. Ze względu na rozproszony charakter systemów Internetu Rzeczy, komponenty pełniące rolę klienta (lub subskrybenta) oraz serwera (lub publikującego) powinny być niezależnymi aplikacjami, w celu wiarygodnego odwzorowania komunikacji w tego typu systemach.

Zadanie

W ramach zadania należy przygotować następujące programy/aplikacje:

1. **Aplikacja 1** (generator) wysyłająca do brokera MQTT informację o aktualnej zajętości dysków (może być dowolny inny parametr dot. zasobu

obliczeniowego) maszyny, na której aplikacja jest uruchomiona wraz ze znacznikiem czasu (timestamp) kiedy informacja została pozyskana. Wysyłanie ma następować w przypadku zmiany wartości mierzonego parametru o określoną wartość.

2. **Aplikacja 2** nasłuchująca wybrany temat brokera MQTT (temat taki jak zdefiniowany w aplikacji 1.), a następnie wysyłająca komunikat JSON przy pomocy żądania POST do serwera HTTP.
3. **Aplikacja 3** będąca serwerem HTTP, który przyjmuje komunikaty, o których mowa w opisie aplikacji 2, a których odczytanie jest możliwe poprzez wysłanie żądania GET.
   Format komunikatu:
   Język programowania oraz wykorzystane biblioteki są dowolne.

Materiały przydatne do realizacji zadania:

- https://test.mosquitto.org/ – testowa instancja brokera MQTT.
- https://hub.docker.com/_/eclipse-mosquitto – obraz Docker z brokerem MQTT, gdyby ktoś miał ochotę na uruchomienie lokalnej instancji.
- https://mqtt.org/software/ – lista oprogramowania, bibliotek, itp. związanych z protokołem MQTT.

Pomoce programistyczne

PYTHON:

- http://www.steves-internet-guide.com/into-mqtt-python-client/ – bardzo łatwy, szybki i przyjemny tutorial dot. obsługi MQTT, u dołu strony film instruktażowy.
- https://www.codementor.io/@sagaragarwal94/building-a-basic-restful-api-in-python-58k02xsiq – prosty przykład serwera REST w Python, jeżeli ktoś nie miał wcześniej doświadczeń z czymś podobnym.
- https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask -- rozbudowany opis budowania serwera REST w Python.
- https://flask-restful.readthedocs.io/en/latest/ – jw., chyba nawet przyjemniejszy.

.NET Core/C#:

- Opracowanie usługi REST:
  - https://docs.microsoft.com/pl-pl/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio – tutorial opisujący w jaki sposób zaimplementować przykładową usługę REST w .NET Core MVC.
  - https://docs.microsoft.com/pl-pl/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client – opis wykorzystania klasy HttpClient do wywołania metody usługi REST.
  - http://zetcode.com/csharp/httpclient/ – alternatywny opis wykorzystania klasy HttpClient, być może bardziej przyjazny.
- Opracowanie usługi wykorzystującej MQTT:
  - https://github.com/chkr1011/MQTTnet – biblioteka do .NET Core wspierająca wykorzystanie protokołu MQTT do wymiany danych.
  - https://www.nuget.org/packages/MQTTnet/ – paczka Nuget do .NET Core z ww. biblioteką.

- https://github.com/chkr1011/MQTTnet/wiki/Client – tutorial opisujący podstawowe przypadki wykorzystania biblioteki MQTT do .NET.