Intel® Edison – Linux Yocto 3.5

Rozwijanie aplikacji MQTT

|  |  |
| --- | --- |
| Autor | Bartłomiej Krasoń |
| Wersja | 1 |
| Data modyfikacji | 28-11-2018 |

Spis treści

[Wprowadzenie 3](#_Toc531196130)

[Dane techniczne stanowiska 3](#_Toc531196131)

[Protokół MQTT 4](#_Toc531196132)

[Broker MQTT 4](#_Toc531196133)

[Client MQTT 4](#_Toc531196134)

[Instalacja Mosquitto 5](#_Toc531196135)

[Uruchomienie Brokera 5](#_Toc531196136)

[Klient subskrybent 6](#_Toc531196137)

[Klient publikant 6](#_Toc531196138)

[Program w Pythonie 6](#_Toc531196139)

[Klient MQTT 7](#_Toc531196140)

# Wprowadzenie

Niniejszy dokument prezentuje w jaki sposób rozwijać aplikacje obsługujące protokół MQTT przy wykorzystaniu modułu obliczeniowego jakim jest **Intel® Edison** z wbudowanym systemem **Linux Yocto 3.5**. Dokument opisuje w jaki sposób, na Edisonie uruchomić *Brokera MQTT* oraz jak go wykorzystać za pomocą prostych przykładowych klientów. Dla tego przykładu wykorzystamy **Eclipse Mosquitto™**, ktróy jest open-sourcowym *brokerem wiadomości* implementującym protokół MQTT.

# Dane techniczne stanowiska

Rozdział ten stanowi zbiór wszystkich używanych programów, systemów itp. oraz ich wersji jakie stanowią stanowisko pracy nad zadanych tematem. Udostępniane są również odnośniki umożliwiające pozyskanie programów lub wgląd do dokumentacji technicznych, dodatkowe poradniki. Objaśniane są również ważniejsze pojęcia techniczne.

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa / Wersja | Typ | Opis | Linki |
| Intel® Edison | moduł | moduł obliczeniowy, zawierający w sobie mikroprocesor Intel® Atom™ (dual-core 500MHz) oraz moduły do obsługi Wi-Fi, Bluetooth, USB itp. | [Hardware Guide](https://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison-module-hardware-guide.pdf)  [Product Brief](https://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison_pb_331179002.pdf) |
| Intel® Edison Breakout Board | płytka deweloperska | płytka do której wpina się moduł Edisona, posiada potrzebne złącza, umożliwiające mu komunikację z systemem hostem w celu jego zaprogramowania | [Hardware Guide](https://www.intel.com/content/dam/support/us/en/documents/edison/sb/edisonbreakout_hg_331190006.pdf) |
| Windows / 10 | host system | system operacyjny na którym będziemy pracować, na nim piszemy oprogramowanie, które później umieszczamy na systemie targetowym (docelowym) |  |
| Linux Yocto / 3.5 | target system | system operacyjny, który zainstalowany jest w module Edisona, na nim docelowo będzie działać napisana przez nas aplikacja | [Latest Yocto Image](https://downloadcenter.intel.com/download/27074/Intel-Edison-Yocto-Poky-image) |
| PuTTY / 0.70 | program | program służący do komunikacji między systemem hostowym z systemem targetowym zainstalowanym na Edisonie | [Pobierz](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) |
| WinSCP / 5.13.4 | program | program służący do przenoszenia plików miedzy systemami hostowym - targetowym | [Pobierz](https://winscp.net/eng/download.php) |
| Eclipse Mosquitto™ | Messager broker | Open-sourcowy pośrednik wiadomości (message broker) implementujący protokół MQTT. | [Download page](https://mosquitto.org/download/) |
| Eclipse Paho | biblioteka | open-sourcowa biblioteka implementująca interfejsy klientów MQTT. Dostępna w szerokiej gamie języków programowania. | [Download page](https://www.eclipse.org/paho/downloads.php) |
| MQTT Lens | aplikacja webowa | prosta aplikacja przeglądarkowa, umożliwiając komunikację z własnym *message brokerem* spoza lokalnej sieci | [Install (Chrome)](https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm) |

# Protokół MQTT

**MQ Telemetry Transport** (MQTT) – oparty o wzorzec publikacja/subskrypcja, ekstremalnie prosty, lekki [protokół transmisji danych](https://pl.wikipedia.org/wiki/Protok%C3%B3%C5%82_komunikacyjny). Przeznaczony jest do transmisji dla urządzeń niewymagających dużej przepustowości.

## https://s3.amazonaws.com/www.appcelerator.com.images/MQTT_1.pngBroker MQTT

**Message Broker** (*pośrednik wiadomości*) pełni rolę serwera, z którym łączą się klienci, aby za jego pośrednictwem publikować informacje. Najpopularniejsze obecnie brokery, to: Mosquitto[[1]](https://pl.wikipedia.org/wiki/MQTT#cite_note-1), RabbitMQ[[2]](https://pl.wikipedia.org/wiki/MQTT#cite_note-2), HiveMQ[[3]](https://pl.wikipedia.org/wiki/MQTT#cite_note-3), IBM MessageSight[[4]](https://pl.wikipedia.org/wiki/MQTT#cite_note-4), VerneMQ[[5]](https://pl.wikipedia.org/wiki/MQTT#cite_note-5). W tym projekcie wykorzystamy brokera **Mosquitto™**, który jest udostępniony na zasadach open-sourcowych przez organizację **Eclipse**.

## Client MQTT

Każdy z klientów łączy się z brokerem, a następnie subskrybuje dany temat/tematy, może również publikować informacje w danym temacie. Kiedy klient opublikuje jakieś informacje, każdy klient, który subskrybuje ten sam temat, otrzyma tę informację. Tematy nie muszą być wcześniej tworzone, oraz mogą mieć dowolną nazwę. Istnieje szereg bibliotek dla całe gamy języków programowania implementujących interfejsy klientów MQTT. W tym projekcie używamy biblioteki **Eclipse   
Paho**. Jest jedną z popularniejszych i bardziej rozbudowanych bibliotek. Dostępna w 10 językach programowania, m. In. w C++, Javie czy Pythonie.

# Instalacja Mosquitto

Należy nawiązać połączenie sesyjne z Edisonem, zalecane jest połączenie **ssh**. Opis jak go dokonać w [poprzednim dokumencie](Edison_Developing_C++_application_guide.pdf). Po udanym połączeniu i zalogowaniu do Edisona jako **Root** wykonujemy następujące czynności:

1. Pobieramy najnowsze **Mosquitto.** Najnowszą wersję moża sprawdzić na stronie <https://mosquitto.org/download/> poleceniem:

# wget <http://mosquitto.org/files/source/mosquitto-1.5.4.tar.gz>

1. Rozpakowujemy plik:

# tar xzf mosquitto–1.5.4

# cd mosquitto–1.5.4

1. Buildujemy:

# make

# make install *(opcjonalnie)*

1. Instalujemy binarki:

# cp client/mosquitto-pub /usr/bin

# cp client/mosquitto-sub /usr/bin

# cp lib/libmosquitto.so.1 /usr/lib

# cp src/mosquitto /usr/bin

Gdyby jakaś operacja była niemożliwa, należy użyć **sudo.**

## Uruchomienie Brokera

Na tak przygotowanym Edisonie, jesteśmy w stanie uruchomić lokalnego *brokera* na wybranym przez nas porcie. System Edisona jest skonfigurowany w taki sposób, że automatycznie uruchamia lokalny RSMB MQTT serwer. Aby umożliwić mu nasłuchiwanie na wybranym porcie, wystarczy uruchomić komendę:

# mosquito –p 1883

Jeżeli chcemy natomiast skonfigurować działanie brokera w jakiś sposób, należy edytować plik

[*installation\_dir]*/mosquitto/mosquitto.conf

Przykład konfiguracji *brokera* jako mostu, przepuszczającego wszystkie tematy:

#conncetion <name>

connection bridge-1

address 192.168.1.25:1883

topic # out 0

topic # In 0

## Klient subskrybent

Wraz z pakietem, na Edisona zainstalowaliśmy przykładowego najprostszego klienta jako binarkę **mosquitto\_sub**. Aby przetestowac jego działanie można podać komendę:

# mosquitto\_sub -h localhost –p 1883 -t test -v

Po czym klient w nieskończonej pętli w tej sesji będzie nasłuchiwał na wszystkie widomości w lokalnym hoście na porcie 1883 w temacie test i jego pochodnych (np. test/1, test/test itp.)

## Klient publikant

Z pakietem mosquitto zainstalowaliśmy przykładowego klienta publikacyjnego, którego możemy użyć w następujący sposób.

Odpalamy **nową sesję** np. ssh, logujemy się jako **root** i podajemy komendę:

# mosquitto\_pub -h localhost –p 1883 -t test/1 –m “Hello world!”

Co owocuje opublikowaniem wiadomości: “*Hello World!*” w temacie „*test/1*” na *lokalnym hoście* na porcie *1883*.

# Program w Pythonie

Aby rozwijać aplikację stanowiącą klienta MQTT posłużymy się językiem skryptowym jakim jest **Python** oraz bibliotekę **paho-mqtt**, która implementuje przykładowy interfejs klienta protokołu MQTT w języku Python. Aby umożliwić działanie biblioteki na Edisonie musimy zainstalować jej pakiet komendą:

# pip install paho-mqtt

Rozwijać aplikację niekoniecznie musimy na systemie targetowym Linuxa, Jako iż jest to język skryptowy możemy pisać te program na hostowym systemie (tj. Windows). Następnie aby wgrać program skryptowy na Edisona należy przekopiować jego pilk np. programem **WinSCP** ([instrukcja](Edison_Developing_C++_application_guide.pdf)). Następnie:

*PPM na pliku* -> *Edytuj*

U góry pliku dopisujemy:

*#! usr/bin/pyton*

Zapisujemy plik, i jest on możliwy do uruchomienia na Edisonie

Aby uruchomić plik jako skrypt, Pier należy przyznać mu odpowiednie prawa:

# chmod u+x *filename*.py

Uruchomienie skryptu realizujemy komendą:

# ./*filename*.py

## Klient MQTT

Kod przykładowego klienta napisany w języku **Python** przy wykorzystaniu biblioteki **paho-mqtt**:

import paho.mqtt.client as mqtt

*# callback gdy klient otrzyma odpowiedź CONNACK z serwera.*

def on\_connect(client, userdata, rc):

print("Connected with result code " + str(rc))

*# subskrybuj wszytskie tematy pochodne $SYS*

client.subscribe("$SYS/#")

*# callback gdy wiadomość PUBLISH jest otrzymana przez serwer.*

def on\_message(client, userdata, msg):

*# wypisz temat i wiadomość na ekran*

print(msg.topic + " " + str(msg.payload))

*# inicjalizacja klienta*

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

*# połączenie z publicznym serwerem “iot.eclipse.org”*

client.connect("iot.eclipse.org", 1883, 60)

*# działaj w pętli*

client.loop\_forever()