2016

Warsztat IoT, 9.3.2016

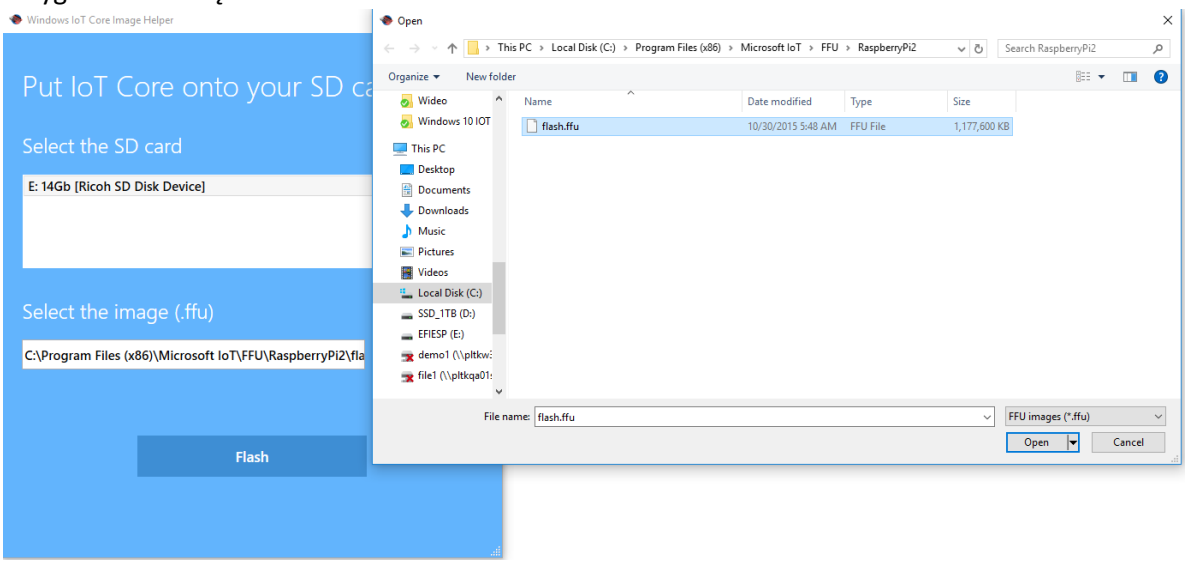
ADAFRUIT WINDOWS IOT STARTER KIT, AZURE IOT HUB

TOMASZ KOPACZ

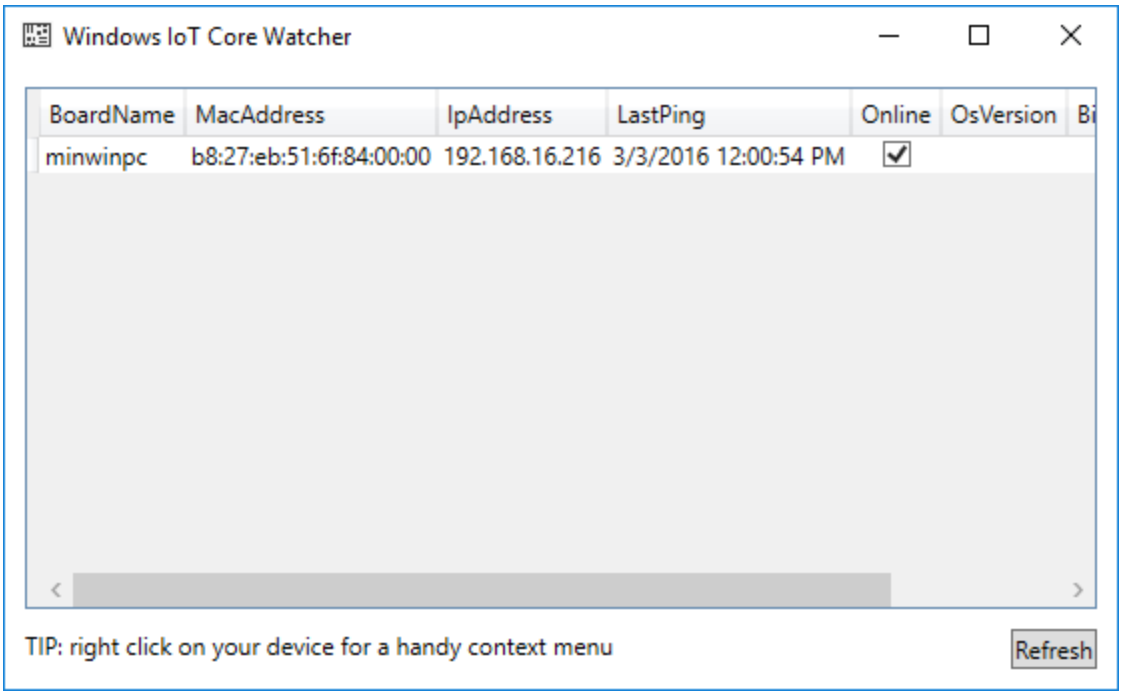
Table of Contents

Przegląd Windows for IoT – UI i procedury postępowania (kolejność ważna!)

1. Przygotować kartę microSD

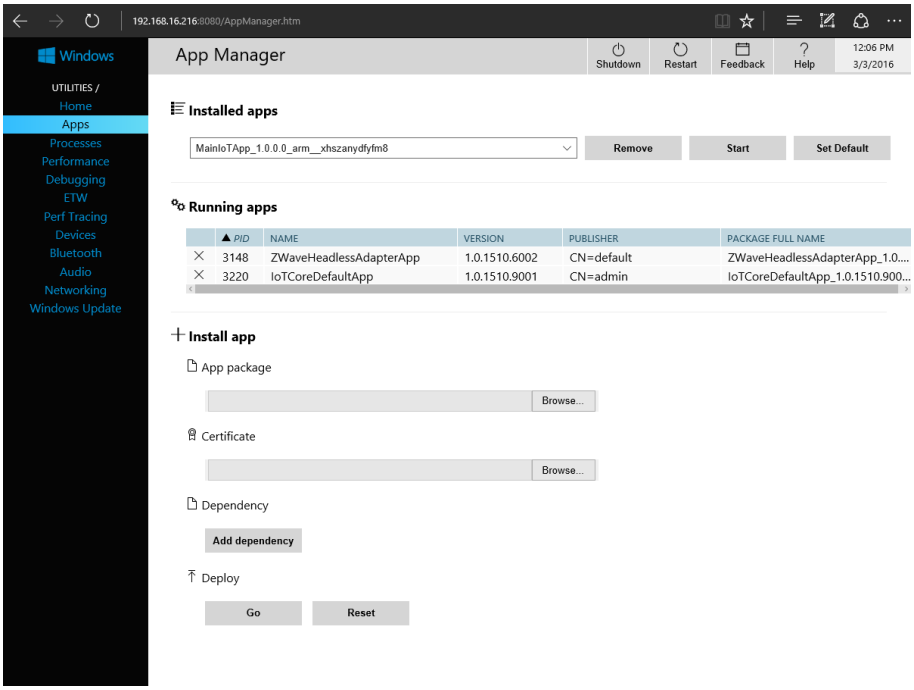


2. PO KOLEI WŁĄCZAĆ URZĄDZENIA – w kolejności wskazanej przez prowadzącego.
    a. **Jedynym jednoznacznym wyróżnikiem jest MAC Address – no i trzeba swój zapisać na kartce!**
3. Zmienić hasło na jakieś „własne” (i go nie zapomnieć – albo idź do punktu 1)
4. Zmienić nazwę urządzenia na D<numer stanowiska>
5. Obserwowanie WSZYSTKICH urządzeń z Windows IoT.



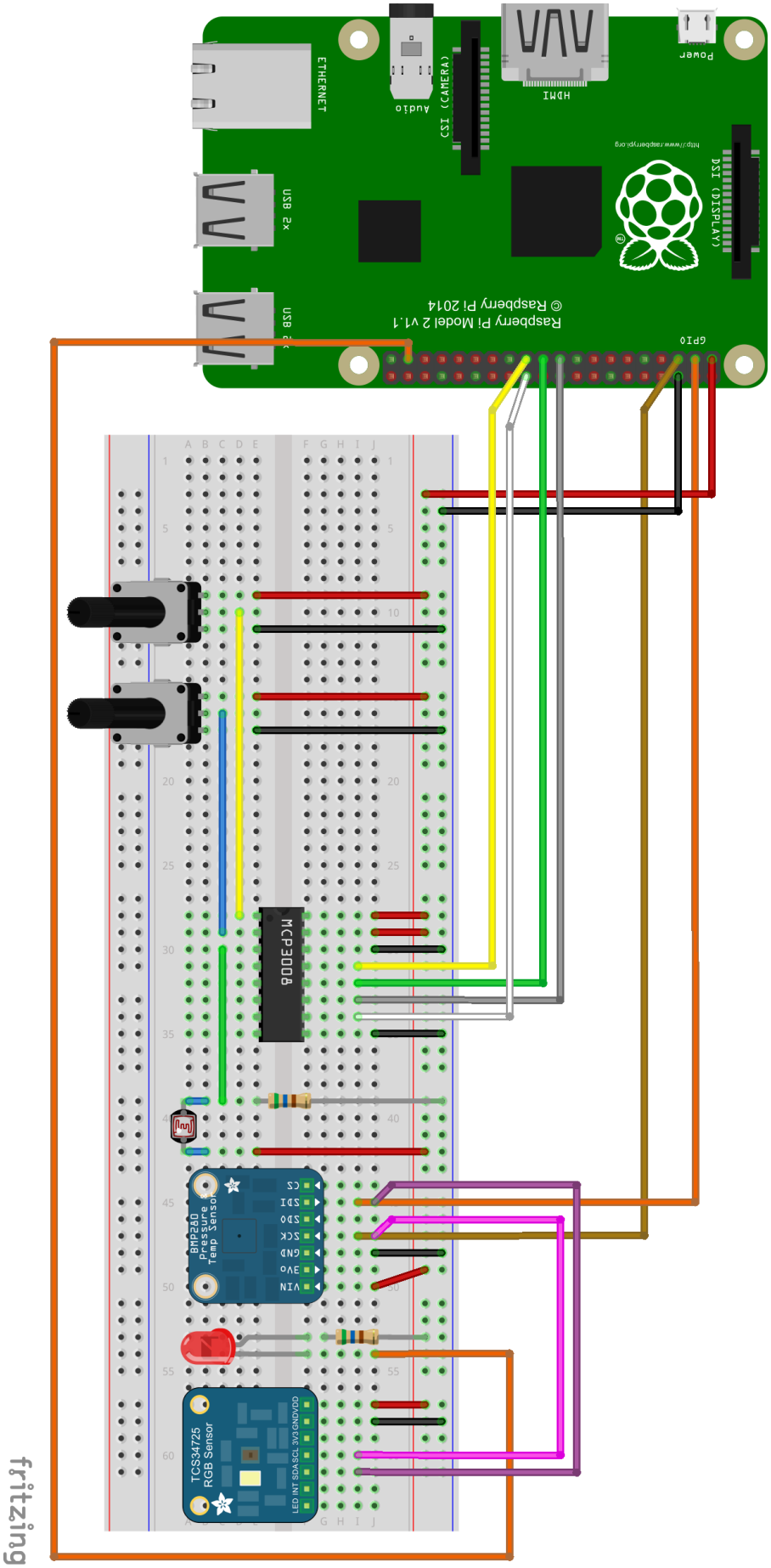
6. Wejście na udział sieciowy //<nazwa urządzenia>/c$

7. Wejście na stronę WWW do zarządzania Windows IoT (http://IP:8080)

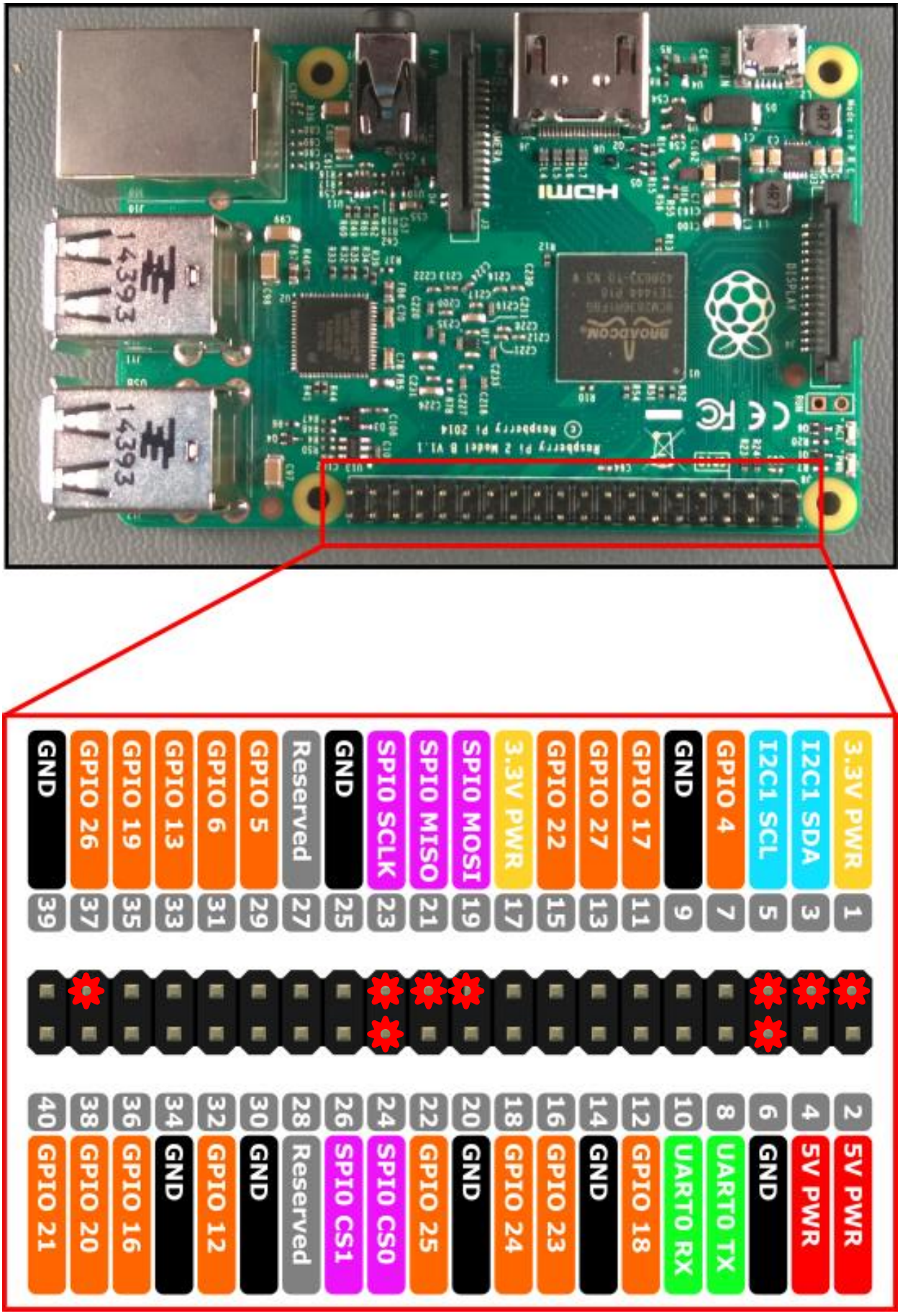


8. Chwilę posłuchać co można zrobić na portalu.

Połączenia


ETHERNET
Audio
CSI (CAMERA)
HDMI
Power
USB 2x
USB 2x
DSI (DISPLAY)
GPIO
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
MCP3008
BMP280
Temp Sensor
CS
SDI
SDO
SCK
GND
3Vo
VIN
TCS34725
RGB Sensor
LED INT SDA SCL 3V3 GND VDD
fritzing

Wyjścia Raspberry PI 2 / 3



Komentarz do połączeń

Numery znajdują się na boku płytki prototypowej (mała czcionka, kolor niebieski). Urządzenia należy połączyć w sposób pokazany na diagramie. Dodatkowo poniżej główne numery „pinów”. Dodatkowo zaznaczone są wyjścia na RPi2 które będą używane w naszym przykładzie.

1. Potencjometr 1
    a. 9,10,11
2. Potencjometr 2
    a. 15,16,17
3. MCP3008
    a. 28,29,30,31,32,33,34,35
4. Foto
    a. 39, 42 (obojętne w którą stronę)
5. BMP280
    a. 44,45,46,47,48,49,50
6. Dioda
    a. 53 (dłuższa noga), 54 (krótsza) (uwaga! On - Low, Off - High)
7. Rezystor1
    a. 39, do - (niebieskie)
8. Rezystor2
    a. 53, do + (czerwone)
9. Podłączyć zasilania i masy
10. Podłączyć inne sygnały
11. Sprawdzić podłączenia. Szczególną uwagę należy poświęcić zasilaniu (czerwony przewód, +) oraz masy (niebieski, -) przed wszystkim!

JAK SIĘ ROBI CIEPŁE I CUCHNIE – SZYBKO WYŁĄCZYĆ ZASILANIE (szansa uszkodzeń mała - podłączamy wszystko pod 3.3V)

Kod

Test 1 LED

1. Nowy projekt UWP
2. Dodać IoT Extension
3. Podłączona dioda do 37 (GPIO 26), za pośrednictwem płytki prototypowej.

Istotne fragmenty kodu w MainPage.cs

```
  public MainPage()
  {
   this.InitializeComponent();
   setup();
  }

  DispatcherTimer m_t;
  GpioPin m_blink;
  GpioPinValue m_blinkValue;
  private async void setup()
  {
   var gpio = GpioController.GetDefault();
   if (gpio!=null)
   {
    m_blink = gpio.OpenPin(26); //See board
    m_blinkValue = GpioPinValue.High;
    m_blink.Write(m_blinkValue);
    m_blink.SetDriveMode(GpioPinDriveMode.Output);
    m_t = new DispatcherTimer();
    m_t.Interval = TimeSpan.FromSeconds(1);
    m_t.Tick += M_t_Tick;
    m_t.Start();
   }
  }

  private void M_t_Tick(object sender, object e)
  {
   if (m_blinkValue == GpioPinValue.High) m_blinkValue = GpioPinValue.Low; else
m_blinkValue = GpioPinValue.High;
   m_blink.Write(m_blinkValue);
  }
```

Kod

Test 2 BMP280

1. Nowy projekt UWP
2. Dodać IoT Extension
3. Podłączony BMP280, za pośrednictwem płytki prototypowej.
4. Pobrać plik z driverem – nieznacznie zmodyfikowany: https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test2_BMP280/Test2_BMP280/BMP280.cs

Istotne fragmenty kodu w MainPage.cs

```
    public MainPage()
    {
      this.InitializeComponent();
      setup();
    }

    DispatcherTimer m_t;
    BMP280 m_bmp280;
    private async void setup()
    {
      m_bmp280 = new BMP280();
      await m_bmp280.Initialize();
      m_t = new DispatcherTimer();
      m_t.Interval = TimeSpan.FromSeconds(5);
      m_t.Tick += M_t_Tick;
      m_t.Start();
    }

    const float seaLevelPressure = 1013.25f;
    private async void M_t_Tick(object sender, object e)
    {
      var altitude = await m_bmp280.ReadAltitudeAsync(seaLevelPressure);
      var pressure = await m_bmp280.ReadPreasureAsync();
      var temperature = await m_bmp280.ReadTemperatureAsync();
      Debug.WriteLine($"Alt:{altitude} m, Press:{pressure} Pa, Temp:{temperature} deg
C");
    }
```

Kod

Test 3 MPC3008

1. Nowy projekt UWP (Test3_MCP3008)
2. Dodać IoT Extension
3. Podłączony BMP280, za pośrednictwem płytki prototypowej.
4. Pobrać ADC Providers z: https://github.com/ms-iot/BusProviders.git
5. Dodać projekt ms-iot\ADC\AdcMcp3008\AdcMcp3008.csproj do rozwiązania
6. Dodać referencję nowo dodanego projektu z poziomu Test3_MCP3008

Istotne fragmenty kodu w MainPage.cs

```
public MainPage()
{
  this.InitializeComponent();
  setup();
}

private AdcController m_adc;
private AdcChannel[] m_adcChannel;
DispatcherTimer m_t;
private async void setup()
{
  m_adc = (await AdcController.GetControllersAsync(AdcMcp3008Provider.GetAdcProvider()))[0];
  m_adcChannel = new AdcChannel[m_adc.ChannelCount];
  for (int i = 0; i < m_adc.ChannelCount; i++)
  {
    m_adcChannel[i] = m_adc.OpenChannel(i);
  }
  m_t = new DispatcherTimer();
  m_t.Interval = TimeSpan.FromSeconds(5);
  m_t.Tick += M_t_Tick;
  m_t.Start();
}

private void M_t_Tick(object sender, object e)
{
  for (int i = 0; i < m_adc.ChannelCount; i++)
  {
    Debug.Write($"{i}:{m_adcChannel[i].ReadRatio():F4}, ");
  }
  Debug.WriteLine("");
}
```

Kod

Test 4 TCS34725

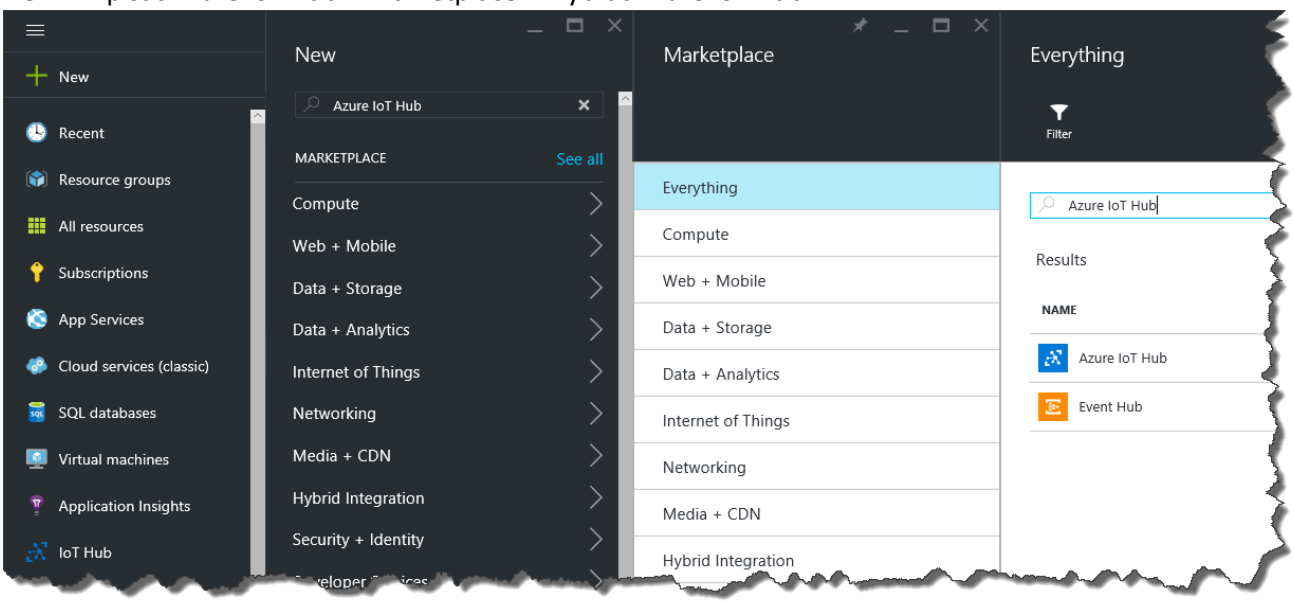
1. Nowy projekt UWP (Test3_MCP3008)
2. Dodać IoT Extension
3. Podłączony TCS34725, za pośrednictwem płytki prototypowej.
4. Pobrać lekko zmodyfikowaną bibliotekę: https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test4_TCS34725/Test4_TCS34725/TCS34725.cs

Istotne fragmenty kodu w MainPage.cs

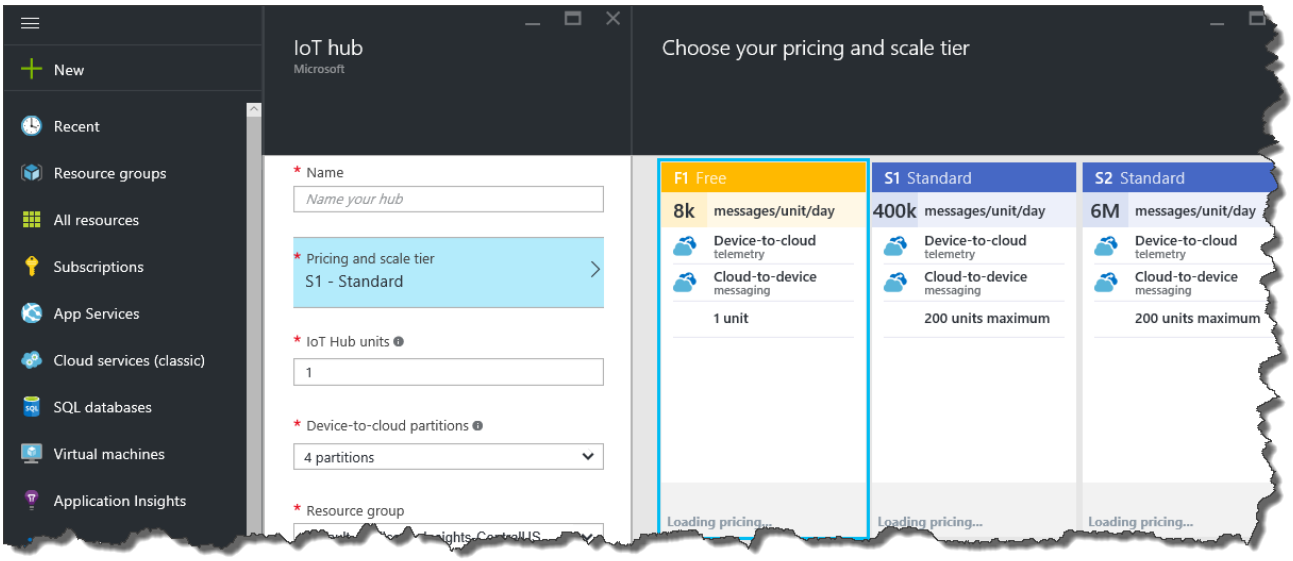
```
public MainPage()
{
  this.InitializeComponent();
  setup();
}
TCS34725 m_tcs;
DispatcherTimer m_t;

private async void setup()
{
  m_tcs = new TCS34725();
  await m_tcs.Initialize();
  m_t = new DispatcherTimer();
  m_t.Interval = TimeSpan.FromSeconds(1);
  m_t.Tick += M_t_Tick;
  m_t.Start();
}

private async void M_t_Tick(object sender, object e)
{
  var color = await m_tcs.GetRgbDataAsync();
  Debug.WriteLine($"{color.Red},{color.Green},{color.Blue}");
}
```

Azure - założenie IoT Hub i okolic

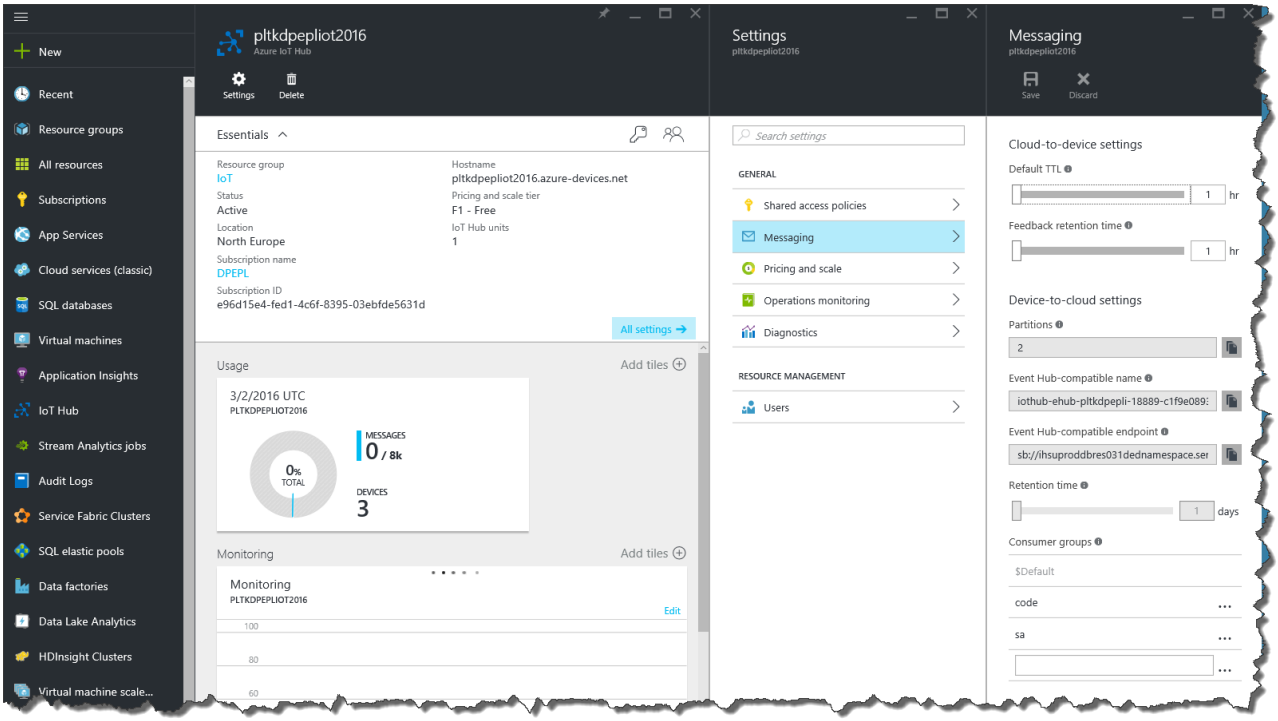
1. https://portal.azure.com
2. New – wpisać *Azure IoT Hub* – Marketplace – wybrać *Azure IoT Hub*



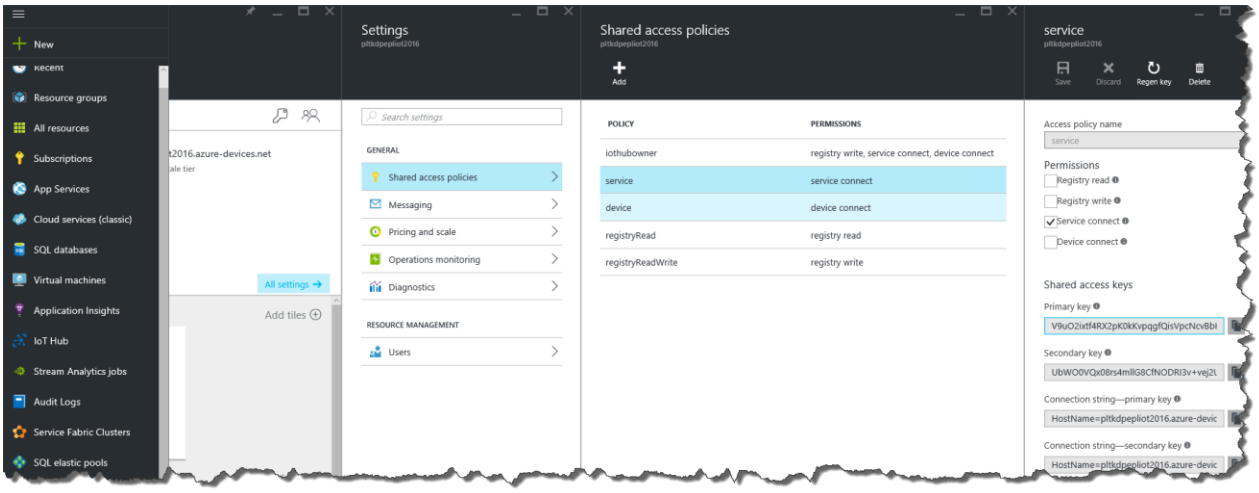
3. Kliknąc *Create*
4. Podać nazwę – sugerowane – iot*inicjałyDataEventu*
5. Wybrać wersję darmową IoT Hub (wystarczy do ćwiczeń).



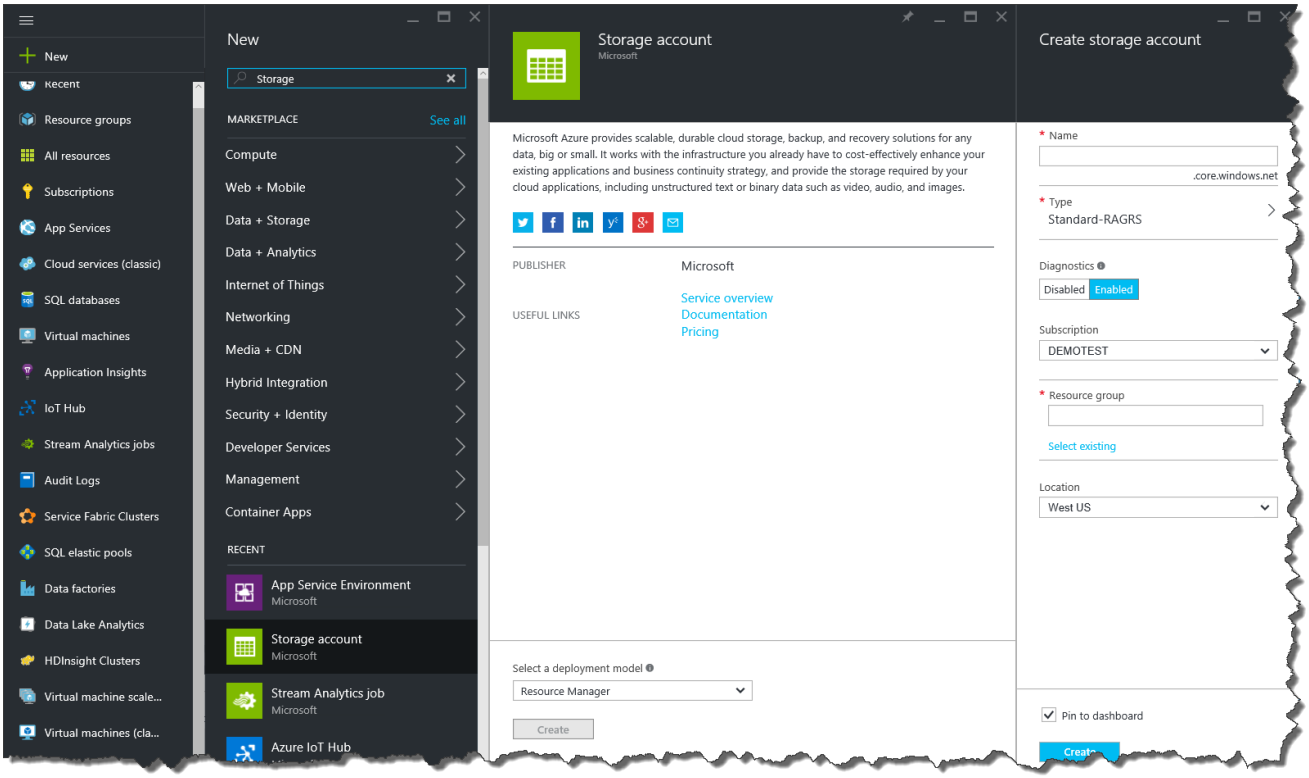
6. Po utworzeniu IoT Hub, dodać 2 Consumer Group: *sa* oraz *code*



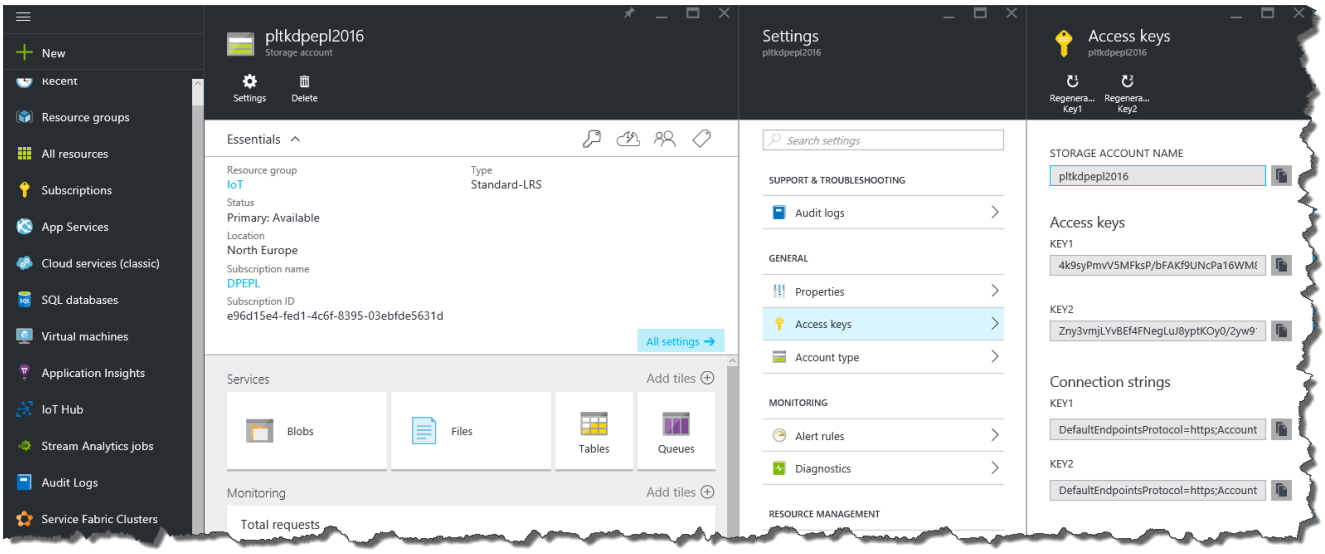
7. Skopiować na bok (przyda się potem):
   Shared access policies : *iothubowner*, *service*, *device*



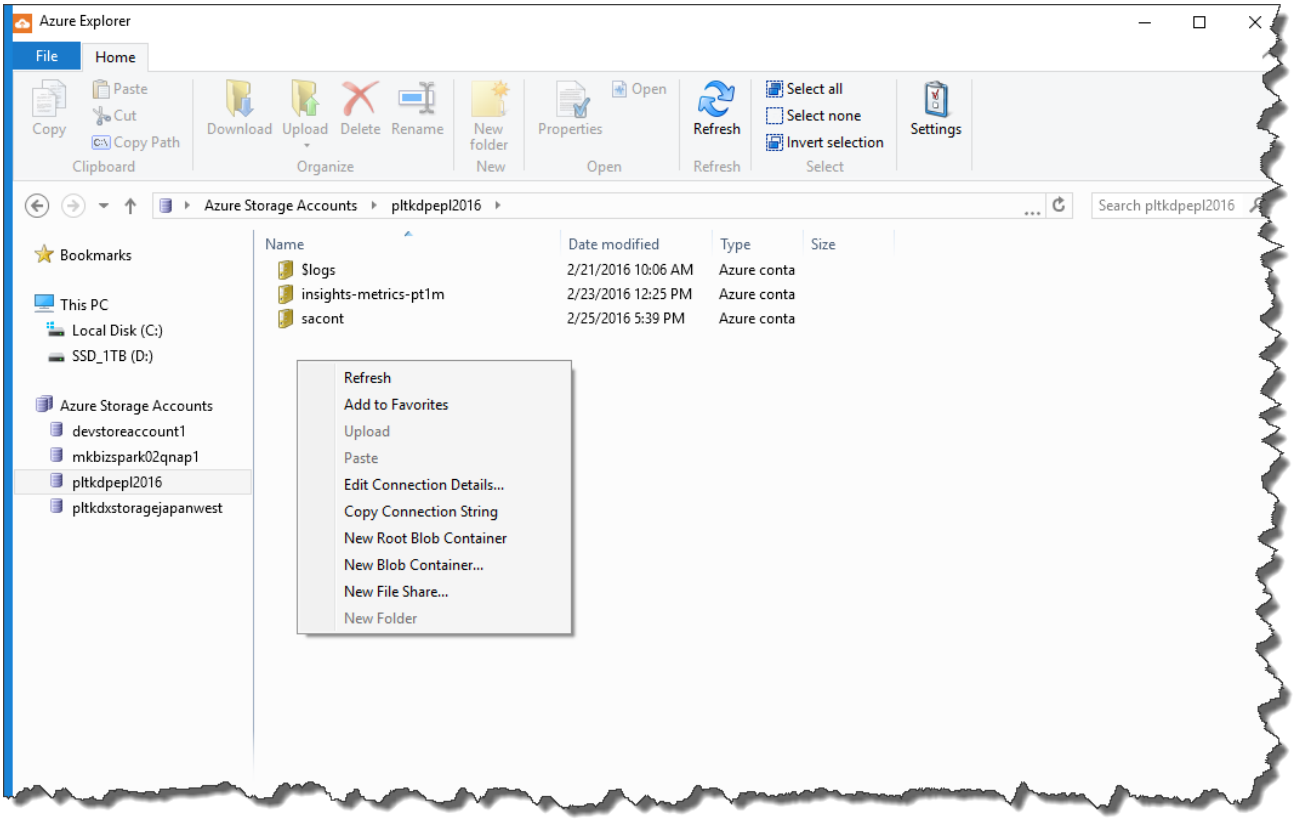
8. Założyć konto Azure Storage (o nazwie st*inicjałyDataEventu*), może być deployment type Resource Manager



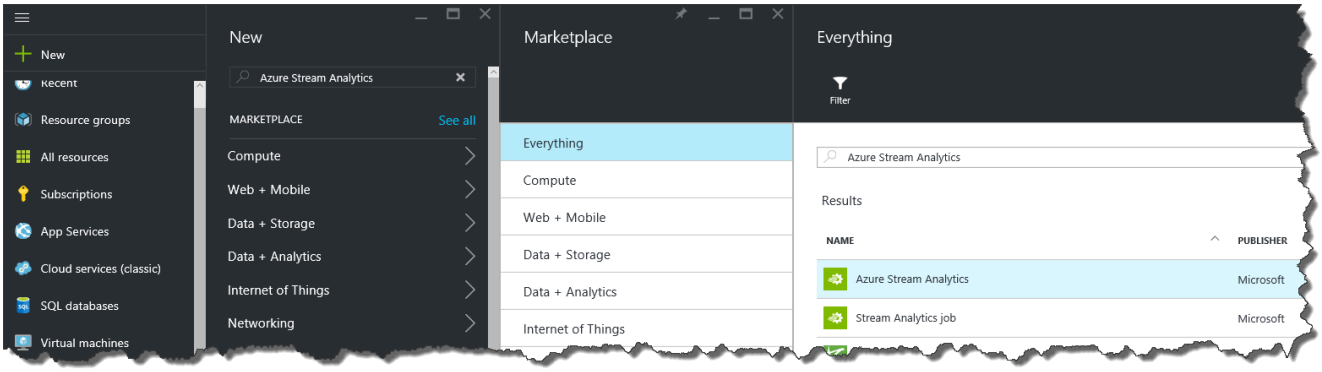
9. Skopiować na bok: Storage Account Name i Account Keys – KEY1 (przyda się do exploratora i stream analytics)



10. Podłączyć się Azure Explorer (lub innym narzędziem) do Storage:

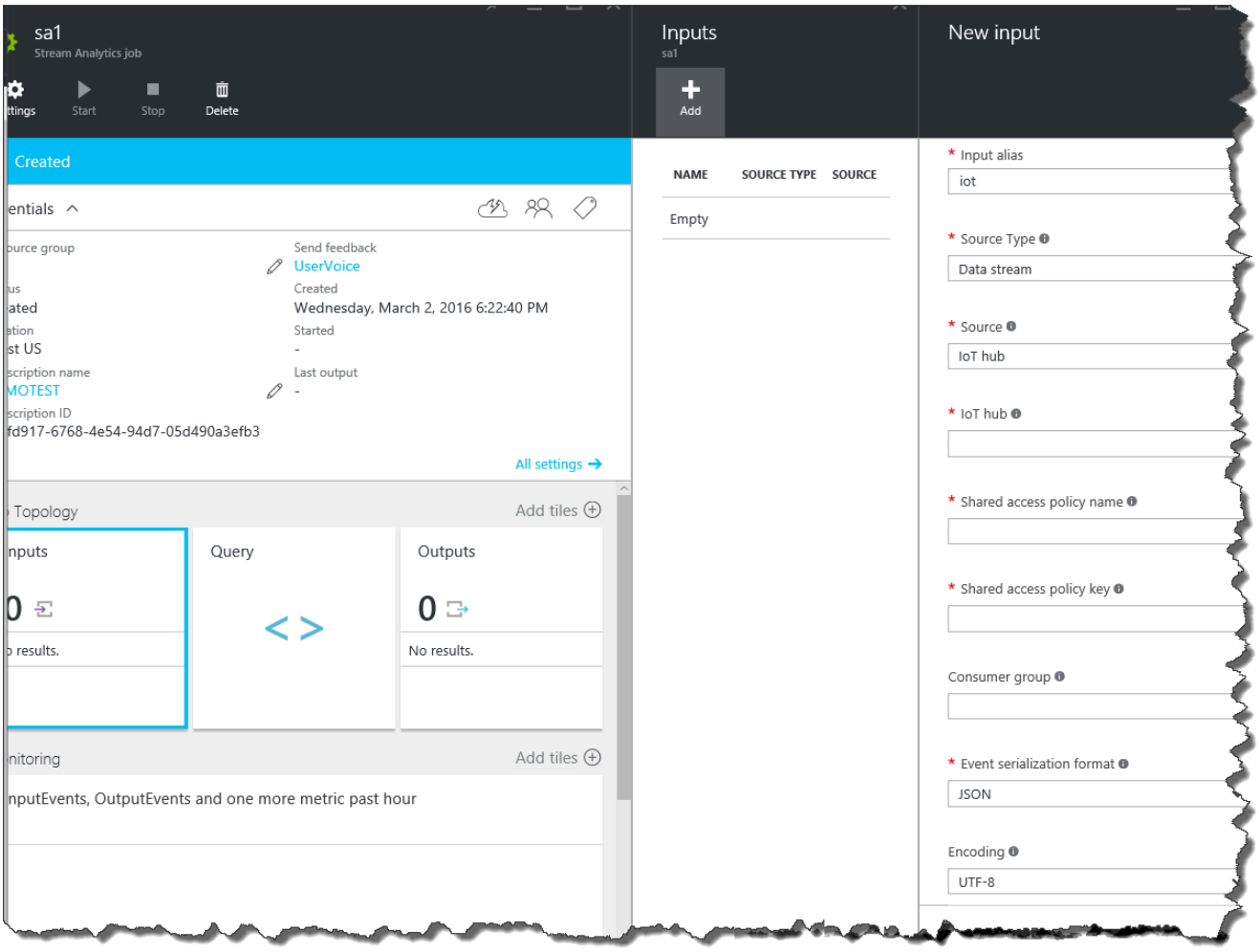


11. Utworzyć nowe *Azure Stream Analytics* (lub *Stream Analytics job*)

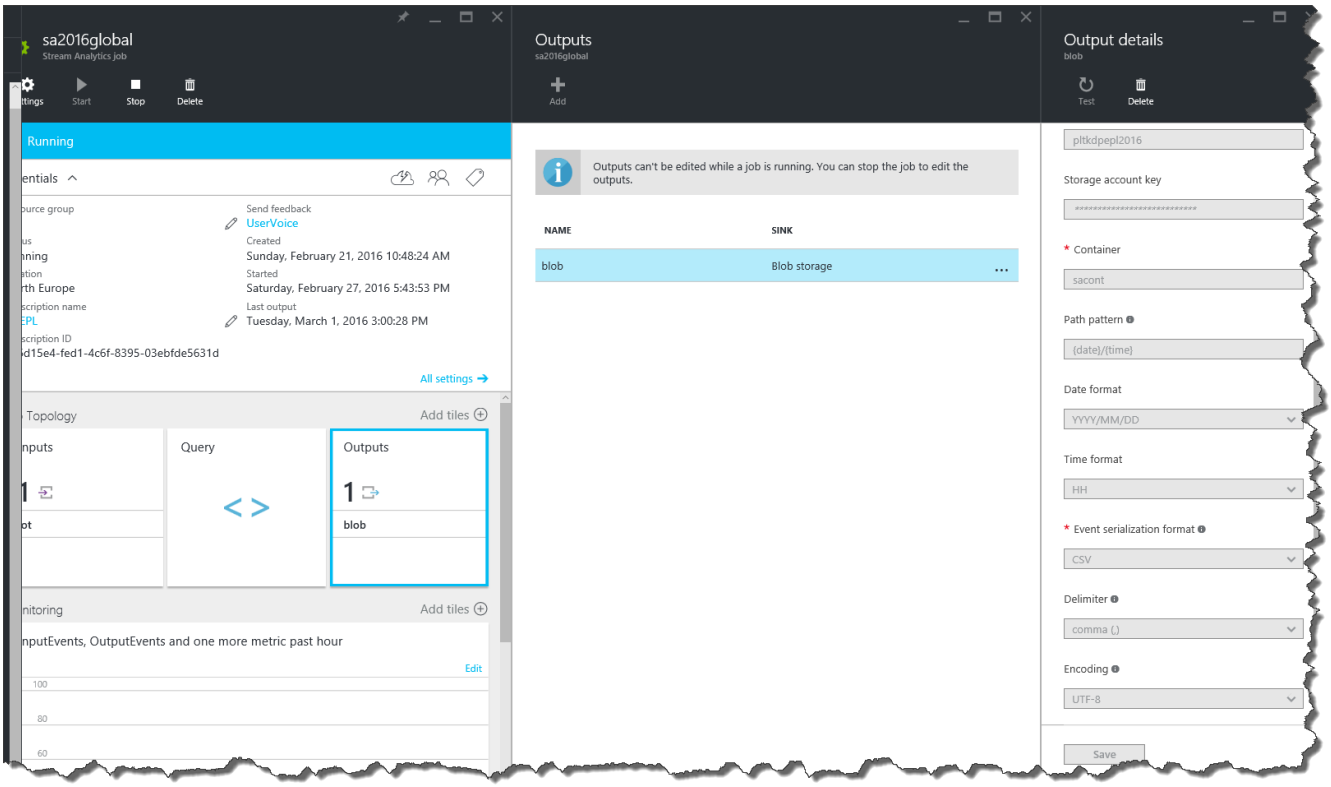


12. Nazwa to na przykład: sa*inicjałyDataEventu*

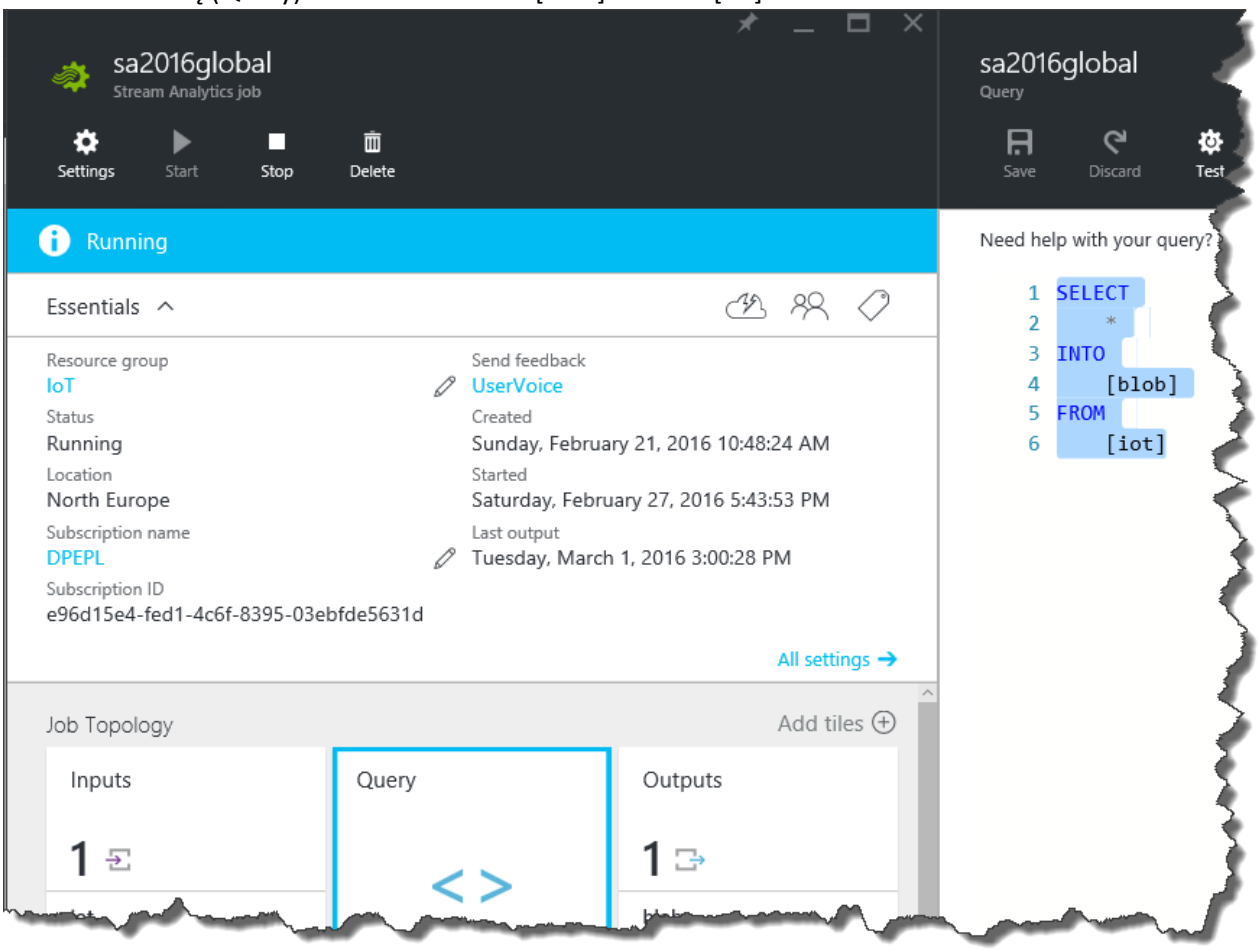
13. Dodać źródło o nazwie iot, wskazujące na Shared Access Policy naszego IoT Hub. Należy podać też Consumer group (utworzone *sa*) oraz wybrać serializację JSON w UTF-8.



14. Dodać wyjście o nazwie blob wskazujące na utworzony kontener. Warto wybrać format CSV, z przecinkiem i w UTF-8. Można też określić ścieżkę (*Path pattern*)



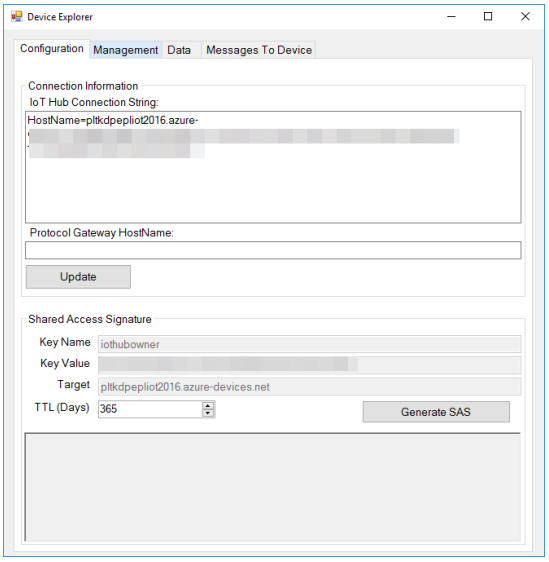
15. Dodać kwerendę (Query): SELECT * INTO [blob] FROM [iot]



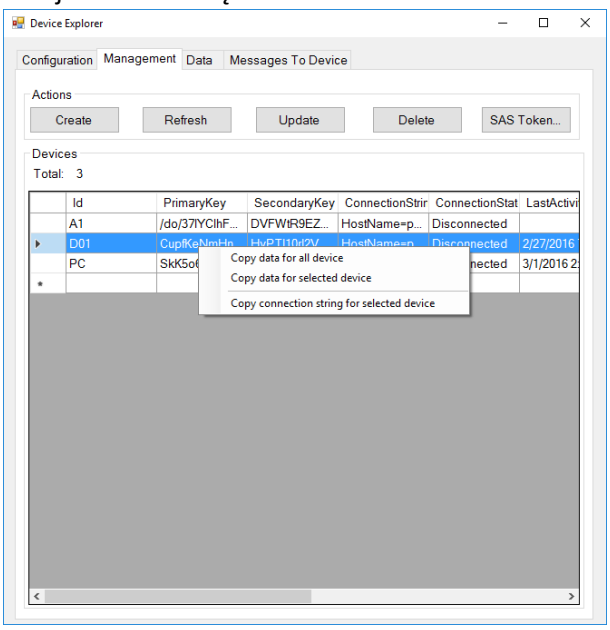
16. Koniec konfiguracji (na razie) Azure.

Rejestracja urządzenia – DeviceExplorer

1. Git clone https://github.com/tkopacz/2016windowsiot-iothub-genericsender
2. Uruchomić READY_TO_RUN\DeviceExplorer.exe
3. Zalogować się Connection string dla iothubowner



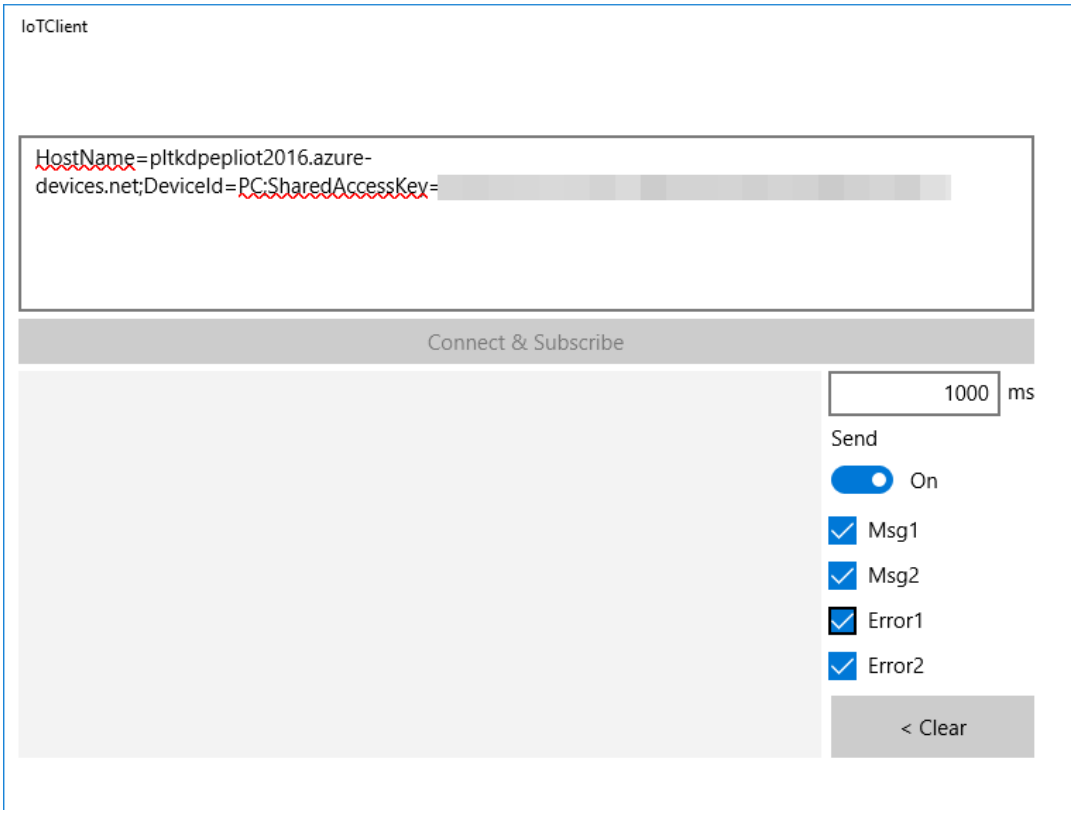
4. Zarejestrować urządzenie o nazwie *D<numer stanowiska>* oraz PC



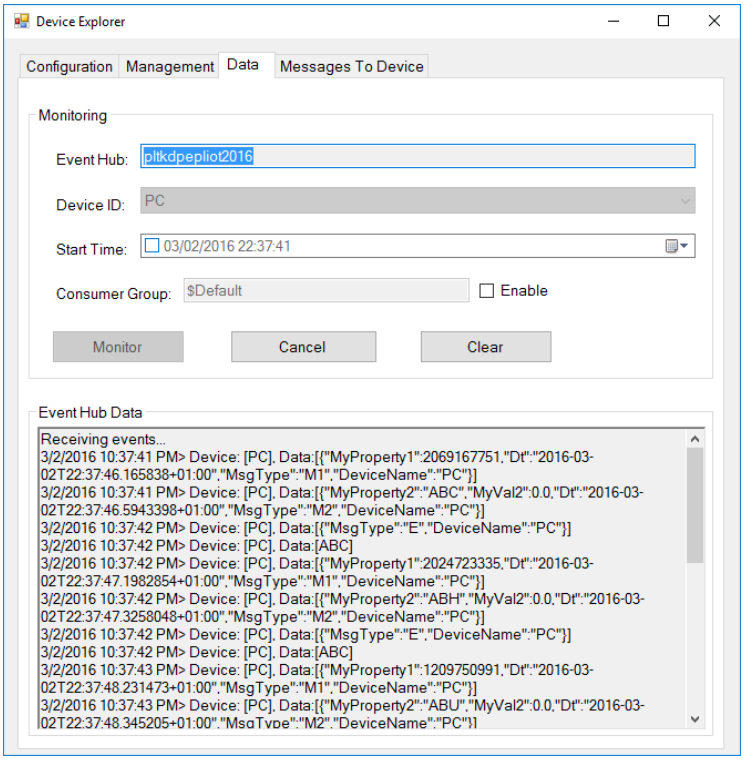
5. Skopiować Connection String dla obu urządzeń

Azure IoT Hub – czy to w ogóle działa?

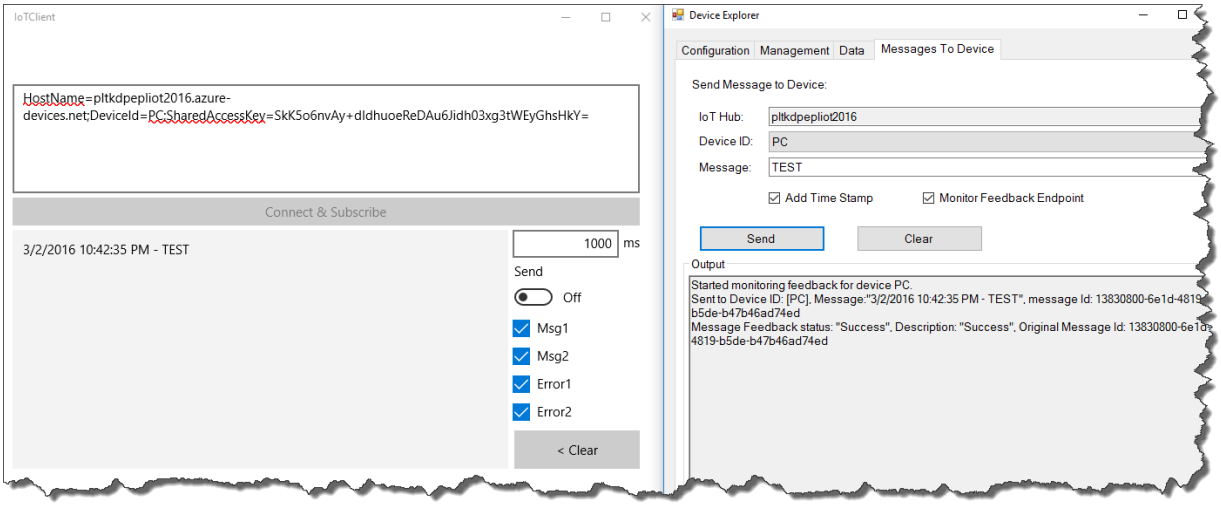
1. Git clone https://github.com/tkopacz/2016windowsiot-iothub-genericsender
2. Otworzyć projekt **SamplePCClient\IoTClient\IoTClient.sln**
3. Uruchomić SamplePCClient\IoTClient\IoTClient\IoTClient.csproj
4. Podać właściwy łańcuch połączeń (dla PC) – skopiowany przed chwilą po zarejestrowaniu urządzenia.



5. Zobaczyć czy zdarzenia się odbierają w DeviceExplorer

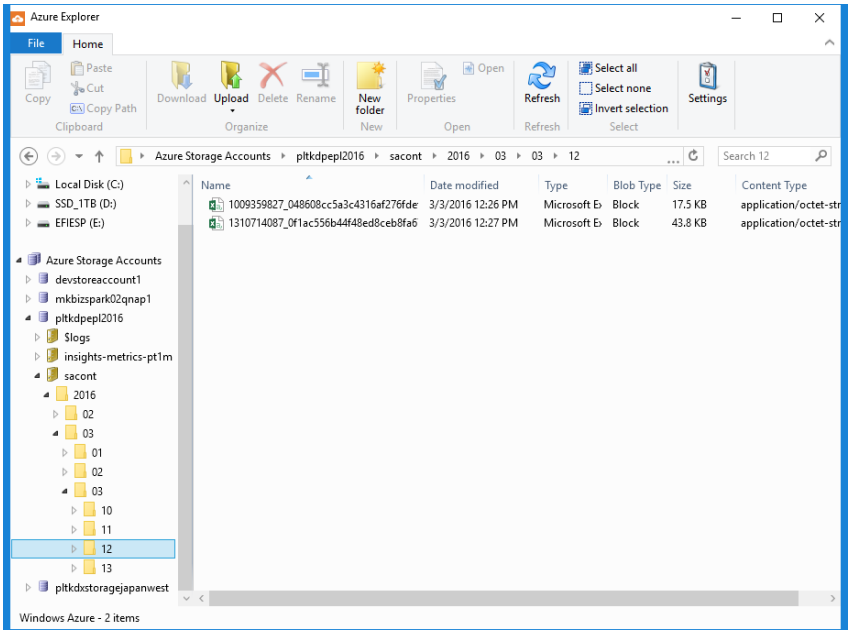


6. Przetestować wysyłanie wiadomości (zakładka Message To Device)



7. Oczywiście – podejrzeć w debuggerze jak działa aplikacja.

8. Podejrzeć czy prawidłowo się generują blob-y (za pośrednictwem Stream Analytics Job). Dowolnym narzędziem (Visual Studio, Portal – tu – RedGate Azure Explorer, z Tools\setupAzureStorageExplorerCopy20160226.exe



Aplikacja UWP do wysyłania komunikatów do IoT Hub

1. Nowa aplikacja UWP
2. Dodać Windows Iot Extension
3. Dodać (NuGet): **Newtonsoft.Json**
4. Dodać (NuGet): **Microsoft.Azure.Devices.Client**
5. Dodać referencję z ADC Providers do ms-iot\ADC\AdcMcp3008\AdcMcp3008.csproj
   Źródło: https://github.com/ms-iot/BusProviders.git
6. Pobrać plik z driverem dla BMP280 – nieznacznie zmodyfikowany (poprawić przestrzenie nazw itp.)
   https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test2_BMP280/Test2_BMP280/BMP280.cs
5. Pobrać plik z driverem do TCS34725 (poprawić przestrzenie nazw itp.):
   https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test4_TCS34725/Test4_TCS34725/TCS34725.cs
7. Dodać **Model.cs**, klasę opisującą komunikat wysyłany do IoT Hub

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MainIoTApp
{
    public class MIoTBase
    {
        public DateTime Dt { get; set; }
        public string MsgType { get; set; }
        public string DeviceName { get; set; }
    }
    public class MSPI:MIoTBase
    {
        public double Potentiometer1 { get; set; }
        public double Potentiometer2 { get; set; }
        public double Light { get; set; }
    }
    public class MAll : MSPI
    {
        public double ADC3 { get; internal set; }
        public double ADC4 { get; internal set; }
        public double ADC5 { get; internal set; }
        public double ADC6 { get; internal set; }
        public double ADC7 { get; internal set; }
        public float Altitude { get; internal set; }
        public string ColorName { get; internal set; }
        public ColorData ColorRaw { get; internal set; }
        public RgbData ColorRgb { get; internal set; }
        public float Pressure { get; internal set; }
        public float Temperature { get; internal set; }
    }
}
```

---

8. W pliku MainPage.xaml dodać “UI”. Składa się z:
   TextBox **txtSPI** – określa jak często będzie odpytywany przetwornik ADC i wysyłany „mały” komunikat)
   TextBox **txtAll** – określa jak często będzie odpytywany cały układ i wysyłany „duży” komunikat)
   ToggleSwitch **tgSend** – włącza i wyłącza wysyłanie)
   TextBlock **txtState** – pokazuje na ekranie stan działania (ostatnio wysłany komunikat, wyjątek itp.)

```
    <Page
x:Class="MainIoTApp.MainPage"
```

```
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MainIoTApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <TextBlock x:Name="txtState" HorizontalAlignment="Left" Margin="0,133,0,0" TextWrapping="Wrap" Text="TextBlock"
VerticalAlignment="Top" Height="200" Width="700" FontSize="13.333"/>
        <TextBox x:Name="txtSPI" HorizontalAlignment="Left" Margin="266,73,0,0" TextWrapping="Wrap" Text="1000"
VerticalAlignment="Top" FontSize="32" Width="166" TextChanged="txtSPI_TextChanged" IsEnabled="False"/>
        <ToggleSwitch x:Name="tgSend" Header="Send" HorizontalAlignment="Left" Margin="10,59,0,0" VerticalAlignment="Top"
FontSize="21.333" Toggled="tgSend_Toggled" IsEnabled="False" />
        <TextBox x:Name="txtAll" HorizontalAlignment="Left" Margin="534,73,0,0" TextWrapping="Wrap" Text="5000"
VerticalAlignment="Top" FontSize="32" Width="166" TextChanged="txtAll_TextChanged" IsEnabled="False"/>
        <TextBlock x:Name="textBlock" HorizontalAlignment="Left" Margin="474,77,0,0" TextWrapping="Wrap" Text="All :"
VerticalAlignment="Top" FontSize="32"/>
        <TextBlock x:Name="textBlock_Copy" HorizontalAlignment="Left" Margin="198,77,0,0" TextWrapping="Wrap" Text="SPI :"
VerticalAlignment="Top" FontSize="32"/>

    </Grid>
</Page>
```

---

9. Utworzyć plik **mainIothubex.cs** z łańcuchami połączeń do IoT Hub (partial class dla MainPage). Zmienić oczywiście parametry:
   DeviceId – nazwa urządzenia (musi pasować) do SAS dla MQTT
   TKConnectionString – łańcuch połączeń do IoT Hub
   TKConnectionMqtt – nazwa DNS wskazująca na IoT Hub
   TKConnectionMqttPassword – SAS wygenerowany dla DANEGO urządzenia używając DeviceExplorer

```
    public sealed partial class MainPage : Page

{
    const string DeviceId = "D01";

    const string TKConnectionString = "HostName=pltkdpepliot2016.azure-
devices.net;DeviceId=D01;SharedAccessKey=<własne>";
    const string TKConnectionMqtt = "pltkdpepliot2016.azure-devices.net";
    const string TKConnectionMqttUsername = TKConnectionMqtt + "/" + DeviceId;
    const string TKConnectionMqttPassword = "SharedAccessSignature sr=pltkdpepliot2016.azure-
devices.net&sig=….3d&se=1488102293&skn=device";
    //const string TKConnectionMqttPassword = "SharedAccessSignature sr=pltkdpepliot2016.azure-
devices.net&sig=…&se=1488062147&skn=iothubowner";
    const string TKMqttTopicSend = "devices/"+ DeviceId + "/messages/events";
    const string TKMqttTopicReceive = "devices/" + DeviceId + "/messages/devicebound";

}
```

---

10. Do pliku MainPage.xaml.cs i klasy MainPage dodać zmienne pomocnicze:
    Uwaga! **MaxMsgCount** określa maksymalną liczbę komunikatów (małych i dużych) wysyłanych w danej sesji. Pozwala to uniknąć za szybkiego wyczerpania puli darmowych komunikatów w IoT Hub.

```
     DispatcherTimer m_t;
    DispatcherTimer m_tSPI;

    GpioPin m_blink;
    GpioPinValue m_blinkValue;

    const float seaLevelPressure = 1013.25f;

    BMP280 m_bmp280;
    AdcController m_adc;
    AdcChannel[] m_adcChannel;
    TCS34725 m_tcs;

    DeviceClient m_clt;
    MSPI m_mSPI;
    int m_msgSpiCount = 0;
    int m_msgCount = 0;
    int MaxMsgCount = 1000;
```

11. Inicjalizacja urządzeń:

```
        public MainPage()
    {
        this.InitializeComponent();
        setup();
    }
    private async void setup()
    {
        try
        {
            //0.IoTHub client
            m_clt = DeviceClient.CreateFromConnectionString(TKConnectionString, TransportType.Http1);
            await m_clt.SendEventAsync(new Message(new byte[] { 1, 2, 3 }));
            Task.Run(() => ReceiveDataFromAzure()); //Loop.

            //0. Cache for message
            m_mSPI = new MSPI();
            m_mSPI.DeviceName = DeviceId;
            m_mSPI.MsgType = "SPI";

            //1. LED
            var gpio = GpioController.GetDefault();
            if (gpio != null)
            {
                m_blink = gpio.OpenPin(26); //See board, connected to pin 19,
                m_blinkValue = GpioPinValue.High;
                m_blink.Write(m_blinkValue);
                m_blink.SetDriveMode(GpioPinDriveMode.Output);
            }
            //2. BMP280
            m_bmp280 = new BMP280();
            await m_bmp280.Initialize();
            //3. ADC
            m_adc = (await AdcController.GetControllersAsync(AdcMcp3008Provider.GetAdcProvider()))[0];
            m_adcChannel = new AdcChannel[m_adc.ChannelCount];
            for (int i = 0; i < m_adc.ChannelCount; i++)
            {
                m_adcChannel[i] = m_adc.OpenChannel(i);
            }
            //4. TCS34725
            m_tcs = new TCS34725();
            await m_tcs.Initialize();


            m_t = new DispatcherTimer();
            m_t.Interval = TimeSpan.FromMilliseconds(5000);
            m_t.Tick += M_t_Tick;

            m_tSPI = new DispatcherTimer();
            m_tSPI.Interval = TimeSpan.FromMilliseconds(1000); //Caution - we have 8 000 messages / day. So - LIMIT
RATE
            m_tSPI.Tick += M_tSPI_Tick;

            //Can enable UI
            txtAll.IsEnabled = txtSPI.IsEnabled = tgSend.IsEnabled = true;
            tgSend.IsOn = true;
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex.ToString());
            txtState.Text = ex.ToString();
        }


    }
```

12. Obsługa zegara wysyłającego stan przetwornika SPI

```
    private async void M_tSPI_Tick(object sender, object e)
    {
        if (m_msgSpiCount >= MaxMsgCount && MaxMsgCount != -1)
        {
            //No more than MaxMsgCount messages / run
            m_tSPI.Stop(); return;
        }
```

```
        m_mSPI.Potentiometer1 = m_adcChannel[0].ReadRatio();
        m_mSPI.Potentiometer2 = m_adcChannel[1].ReadRatio();
        m_mSPI.Light = m_adcChannel[2].ReadRatio();
        m_mSPI.Dt = DateTime.UtcNow;
        var obj = JsonConvert.SerializeObject(m_mSPI);
        try
        {
            if (m_clt!=null)
            {
                await m_clt.SendEventAsync(new Message(System.Text.Encoding.UTF8.GetBytes(obj)));
                m_msgSpiCount++;
            }
        }
        catch (Exception ex)
        {
            txtState.Text = ex.ToString();
        }

    }
```

---

13. Obsługa zegara wysyłającego komplet stanu odczytanego z dołączonych urządzeń

```
     private async void M_t_Tick(object sender, object e)
    {
        if (m_msgCount >= MaxMsgCount && MaxMsgCount != -1)
        {
            //No more than MaxMsgCount messages / run
            m_t.Stop(); return;
        }
        MAll m = new MAll();
        m.DeviceName = DeviceId;
        m.MsgType = "ALL";

        m.Altitude = await m_bmp280.ReadAltitudeAsync(seaLevelPressure);
        m.Pressure = await m_bmp280.ReadPreasureAsync();
        m.Temperature = await m_bmp280.ReadTemperatureAsync();

        m.Potentiometer1 = m_adcChannel[0].ReadRatio();
        m.Potentiometer2 = m_adcChannel[1].ReadRatio();
        m.Light = m_adcChannel[2].ReadRatio();
        m.ADC3 = m_adcChannel[3].ReadRatio();
        m.ADC4 = m_adcChannel[3].ReadRatio();
        m.ADC5 = m_adcChannel[3].ReadRatio();
        m.ADC6 = m_adcChannel[3].ReadRatio();
        m.ADC7 = m_adcChannel[3].ReadRatio();

        m.ColorRgb = await m_tcs.GetRgbDataAsync();
        m.ColorRaw = await m_tcs.GetRawDataAsync();
        m.ColorName = await m_tcs.GetClosestColorAsync();

        m.Dt = DateTime.UtcNow;
        var obj = JsonConvert.SerializeObject(m);
        try
        {   if (m_clt != null)
            {
                await m_clt.SendEventAsync(new Message(System.Text.Encoding.UTF8.GetBytes(obj)));
                m_msgCount++;
            }
            txtState.Text = obj + $", MSG:{m_msgCount}, MSGSPI:{m_msgSpiCount}";
        }
        catch (Exception ex)
        {
            txtState.Text = ex.ToString();
        }
    }
```

---

14. Obsługa UI (przyciski, zmiana częstotliwości działania timera itp.)

```
     private void tgSend_Toggled(object sender, RoutedEventArgs e)
    {
        if (tgSend.IsOn)
        {
            m_t.Start();
            m_tSPI.Start();
        }
        else
        {
```

```
            m_t.Stop();
            m_t.Stop();
        }
    }

    private void txtSPI_TextChanged(object sender, TextChangedEventArgs e)
    {
        double val;
        if (double.TryParse(txtSPI.Text, out val) && val > 0)
        {
            m_tSPI.Interval = TimeSpan.FromMilliseconds(val);
        }
    }

    private void txtAll_TextChanged(object sender, TextChangedEventArgs e)
    {
        double val;
        if (double.TryParse(txtAll.Text, out val) && val > 0)
        {
            m_t.Interval = TimeSpan.FromMilliseconds(val);
        }
    }
```

---

15. Obsługa zdarzeń (poleceń) odbieranych z chmury

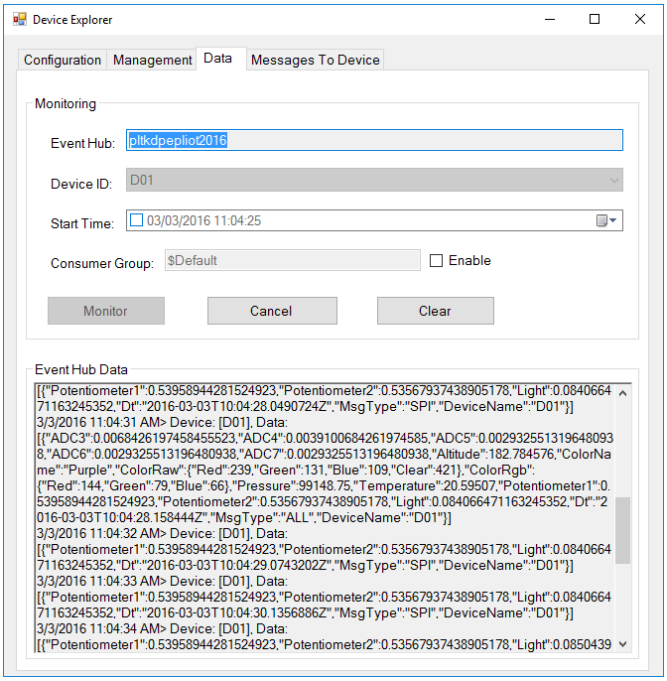
```
     public async Task ReceiveDataFromAzure()
    {

        Message receivedMessage;
        string messageData;
        if (m_clt != null)
        {
            while (true)
            {
                try
                {
                    receivedMessage = await m_clt.ReceiveAsync();

                    if (receivedMessage != null)
                    {
                        messageData = System.Text.Encoding.ASCII.GetString(receivedMessage.GetBytes());
                        //Wykonanie polecenia
                        if (messageData.Length >= 2)
                        {
                            await this.Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
                            {
                                double val;
                                switch (messageData[0])
                                {
                                    case 'L':
                                    //Light
                                    if (messageData[1] == '0')
                                            m_blinkValue = GpioPinValue.High;
                                        else
                                            m_blinkValue = GpioPinValue.Low;
                                        m_blink.Write(m_blinkValue);
                                        break;
                                    case 'A':
                                    //All messages - interval
                                    if (double.TryParse(messageData.Substring(1), out val))
                                        {
                                            txtAll.Text = val.ToString();
                                        }
                                        break;
                                    case 'S':
                                    //SPI messages - interval
                                    if (double.TryParse(messageData.Substring(1), out val))
                                        {
                                            txtSPI.Text = val.ToString();
                                        }
                                        break;
                                    case 'O':
                                    //On / Off
                                    if (messageData[1] == '0')
                                            tgSend.IsOn = false;
                                        else
                                            tgSend.IsOn = true;
```

```
                                break;
                            default:
                                break;
                        }
                    });
                }
                //await m_clt.RejectAsync(receivedMessage);
                //await m_clt.AbandonAsync(receivedMessage); - reject, will be redelivered
                //Confirm
                await m_clt.CompleteAsync(receivedMessage); //potwierdza odebranie
            }
        }
        catch (Exception ex)
        {
            txtState.Text = ex.ToString();
        }
    }
  }
}
```

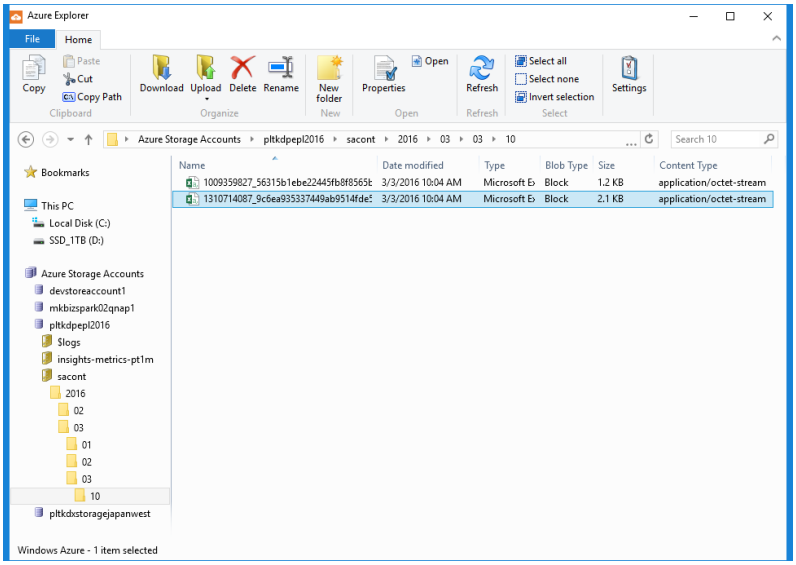
---

16. Uruchomić aplikację

17. Podejrzeć zdarzenia dla danego urządzenia używając DeviceExplorer



18. Zobaczyć że zdarzenia zostały przetworzone przez Stream Analytics

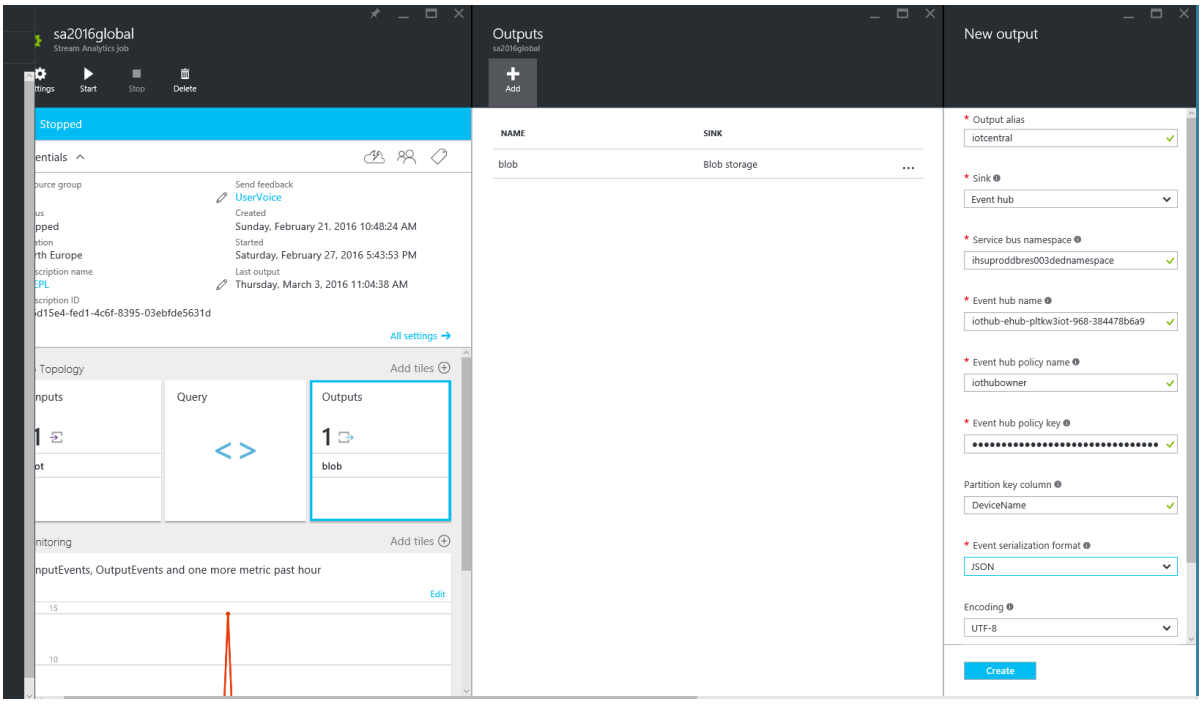


19. Sprawdzić że działa wysyłanie komunikatów do urządzenia i np. włączenie / wyłączenie wysyłania

Dodanie do Stream Analytics dodatkowego wyjścia - centralnego „Event Hub”

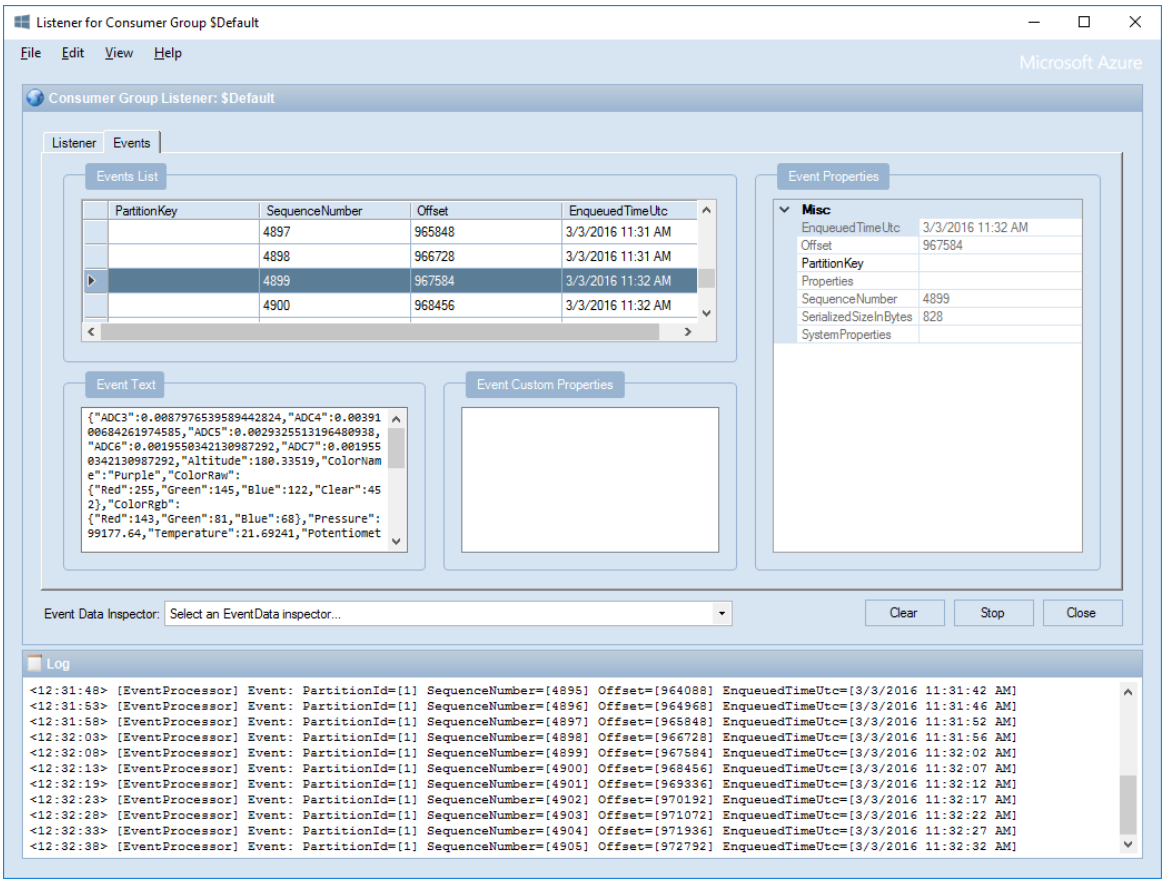
Cel – zebrać w jednym miejscu informacje ze wszystkich urządzeń.

1. Zatrzymać Stream Analytics Job
2. Dodać dodatkowe wyjście o nazwie iotcentral, typu Event Hub
   Używamy API Event Hub. Parametry podane są w innym dokumencie!



3. Zmodyfikować kwerendę, tak by były 2 wyrażenia (wysyłamy do drugiego IoT tylko wiadomości ALL):
   SELECT * INTO [blob] FROM [iot]
   SELECT * INTO [iotcentral] FROM [iot] where MsgType='ALL'

4. Uruchomić kwerendę (i wysyłanie zdarzeń). Po pewnym czasie prowadzący powinien widzieć w swoim Event Hub zdarzenia z sali.

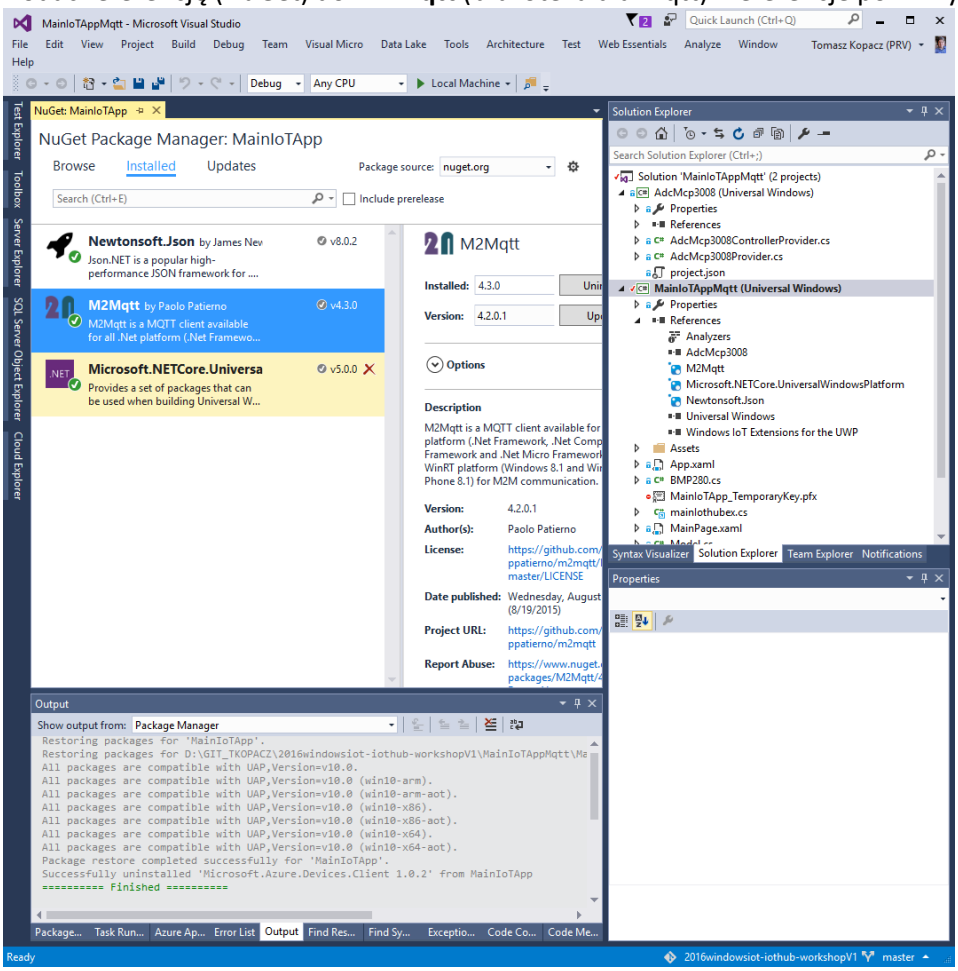


5. Dygresja: można by użyć IoT Hub – pod warunkiem ponownej rejestracji urządzeń itp.

Aplikacja UWP do wysyłania komunikatów do IoT Hub – MQTT

Uwaga! Na UWP albo używamy IoT Hub albo biblioteki MQTT – nie można tego połączyć

1. Proszę skopiować do innego foldera całe rozwiązanie które powstało w ramach Aplikacja UWP do wysyłania komunikatów do IoT Hub
2. Usunąć referencję (NuGet) do **Microsoft.Azure.Devices.Client**
3. Dodać referencję (NuGet) do **M2Mqtt** (biblioteka dla Mqtt). Referencje powinny wyglądać jak:



4. Usunąć odwołania do Azure IoT Hub SDK – m_clt, Message

```
~~DeviceClient m_clt;~~

~~using Microsoft.Azure.Devices.Client;~~
…
```

---

5. Zmienić sposób inicjowania komunikacji w setup:

```
private async void setup()
{
    try
    {

            //0. MQQT for IoT Hub, uPLibrary.Networking.M2Mqtt
            m_mqtt = new MqttClient(TKConnectionMqtt, 8883, true, MqttSslProtocols.TLSv1_2);
            m_mqtt.Connect(DeviceId, TKConnectionMqttUsername, TKConnectionMqttPassword);
            if (m_mqtt.IsConnected == false) throw new ArgumentException("Bad
username/password for MQTT");

        //0. Cache for message
        m_mSPI = new MSPI();
```

```
        m_mSPI.DeviceName = DeviceId;
        m_mSPI.MsgType = "SPI";
```

---

6. Zmienić sposób wysyłania komunikatów z ADC:

```
private async void M_tSPI_Tick(object sender, object e)
{
    if (m_msgSpiCount >= MaxMsgCount && MaxMsgCount != -1)
    {
        //No more than MaxMsgCount messages / run
        m_tSPI.Stop(); return;
    }
    m_mSPI.Potentiometer1 = m_adcChannel[0].ReadRatio();
    m_mSPI.Potentiometer2 = m_adcChannel[1].ReadRatio();
    m_mSPI.Light = m_adcChannel[2].ReadRatio();
    m_mSPI.Dt = DateTime.UtcNow;
    var obj = JsonConvert.SerializeObject(m_mSPI);
    try
    {
          if (m_mqtt != null)
          {
              //Publish to internal queue; then - background process send messages to IoT Hub
              m_mqtt.Publish(TKMqttTopicSend, System.Text.Encoding.UTF8.GetBytes(obj));
              m_msgSpiCount++;
          }
    }
    catch (Exception ex)
    {
        txtState.Text = ex.ToString();
    }
}
```

---

7. Zmienić sposób wysyłania pozostałych komunikatów:

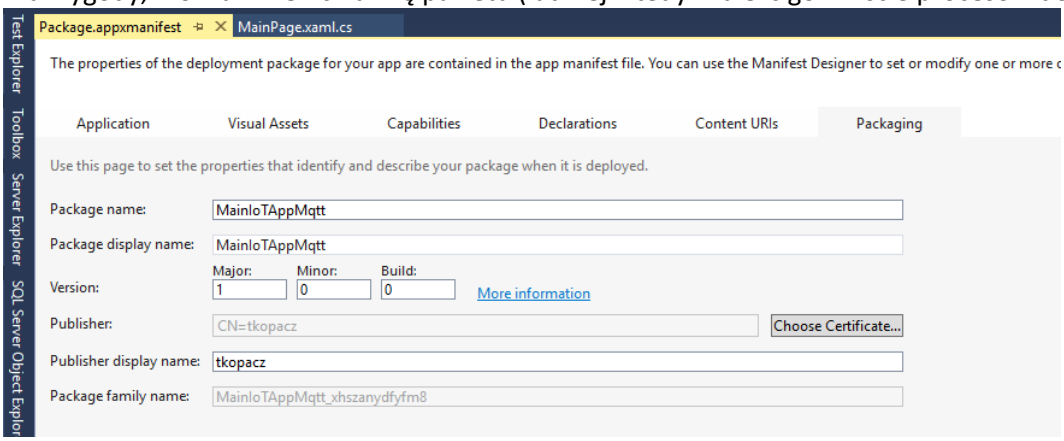
```
 private async void M_t_Tick(object sender, object e)
{
    if (m_msgCount >= MaxMsgCount && MaxMsgCount != -1)
    {
        //No more than MaxMsgCount messages / run
        m_t.Stop(); return;
    }
    MAll m = new MAll();
    m.DeviceName = DeviceId;
    m.MsgType = "ALL";

    m.Altitude = await m_bmp280.ReadAltitudeAsync(seaLevelPressure);
    m.Pressure = await m_bmp280.ReadPreasureAsync();
    m.Temperature = await m_bmp280.ReadTemperatureAsync();

    m.Potentiometer1 = m_adcChannel[0].ReadRatio();
    m.Potentiometer2 = m_adcChannel[1].ReadRatio();
    m.Light = m_adcChannel[2].ReadRatio();
    m.ADC3 = m_adcChannel[3].ReadRatio();
    m.ADC4 = m_adcChannel[3].ReadRatio();
    m.ADC5 = m_adcChannel[3].ReadRatio();
    m.ADC6 = m_adcChannel[3].ReadRatio();
    m.ADC7 = m_adcChannel[3].ReadRatio();

    m.ColorRgb = await m_tcs.GetRgbDataAsync();
    m.ColorRaw = await m_tcs.GetRawDataAsync();
    m.ColorName = await m_tcs.GetClosestColorAsync();

    m.Dt = DateTime.UtcNow;
    var obj = JsonConvert.SerializeObject(m);
    try
    {
           if (m_mqtt != null)
           {
               //Publish to internal queue; then - background process send messages to IoT
Hub
               m_mqtt.Publish(TKMqttTopicSend, System.Text.Encoding.UTF8.GetBytes(obj));
               m_msgSpiCount++;
```

```
            }
            txtState.Text = obj + $", MSG:{m_msgCount}, MSGSPI:{m_msgSpiCount}";
        }
        catch (Exception ex)
        {
            txtState.Text = ex.ToString();
        }
    }
```

---

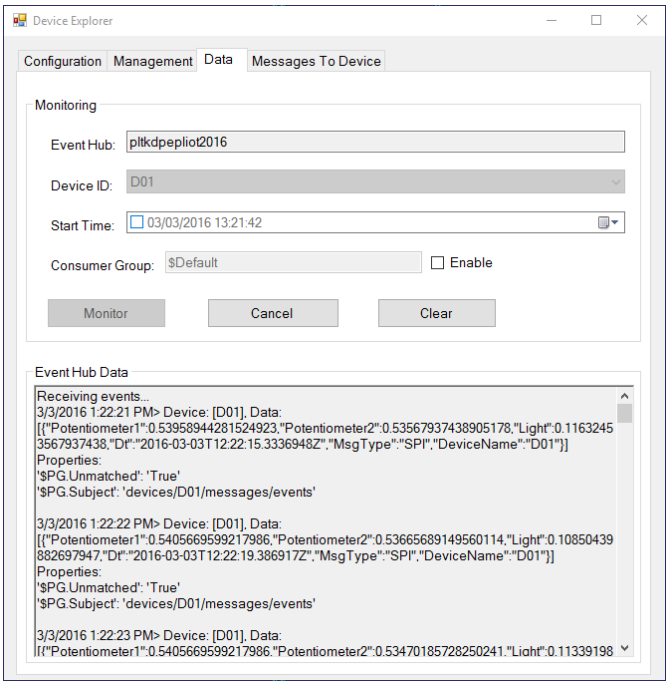
8. Usunąć metodę `public async Task ReceiveDataFromAzure()`

9. Dla wygody, można zmienić nazwę pakietu (łatwiej wtedy znaleźć go w liście procesów do uruchomienia na RPi2.



10. Wgrać rozwiązanie na RPi2

11. Zobaczyć, w DeviceExplorer / Stream Analytics, jak wysyłane są wiadomości



12. Proszę zauważyć, że tym razem – dodatkowo są pewne właściwości (wynikające z działania i implementacji protokołu MQTT

Własna aplikacja do odbierania i przetwarzania wiadomości z IoT Hub

1. Dodać nową aplikację konsolową `ConsoleMonitor`.

2. Dodac referencję (NuGet) do `Microsoft.Azure.Devices`, `Microsoft.Azure.Amqp`, `Microsoft.AspNet.WebApi.Client`

1. W funkcji Main dodać proste pobieranie parametrów z linii poleceń:

```
Console.WriteLine("Usage: \r\n" +
                  "ConsoleMonitor <iotHubOwner> [<consumerGroup>] [<device>] ");

string connection = args[0];
string consumerGroupName = "$Default";
if (args.Length>1) consumerGroupName = args[1];
string deviceName = "";
if (args.Length>2) deviceName = args[2];
```

---

2. W funkcji Main dodać nawiązywanie połączeń z IoT Hub:

```
EventHubClient eventHubClient = null;
EventHubReceiver eventHubReceiver = null;
eventHubClient = EventHubClient.CreateFromConnectionString(connection, "messages/events");
```

---

3. Pobranie informacji (między innymi) o ilości partycji:

```
var ri = eventHubClient.GetRuntimeInformation();
```

---

4. Jeżeli jako parametr została przekazana nazwa urządzenia, monitorujemy tylko tą partycję z IoT Hub/Event Hub do której komunikaty są wysyłane (ResolveToPartition zwraca nazwę partycji dla danej nazwy urządzenia):

```
if (deviceName != "")
{
  string partition = EventHubPartitionKeyResolver.ResolveToPartition(deviceName, ri.PartitionCount);
  eventHubReceiver = eventHubClient.GetConsumerGroup(consumerGroupName).CreateReceiver(partition, DateTime.Now);
  Task.Run(() => eventLoop(eventHubReceiver));
}
```

---

5. W przeciwnym wypadku, startujemy oddzielne wątki do monitorowania każdej partycji. Uwaga! Zwykle to będą oddzielne węzły klastra, oddzielne „spouts” ze Storm – albo – aktorzy z Service Fabric. Zakładamy też, że interesują nas zdarzenia od „teraz”.

```
else {
 EventHubReceiver[] eventHubReceivers = new EventHubReceiver[ri.PartitionCount];
 int i = 0;
 foreach (var partition in ri.PartitionIds) {
   eventHubReceivers[i] = eventHubClient.GetConsumerGroup(consumerGroupName).CreateReceiver(
                          partition, DateTime.Now);
   //Task.Run(() => eventLoop(eventHubReceivers[i])); <- very common bug!
   var r = eventHubReceivers[i];
   Task.Run(() => eventLoop(r));
   i++;
 }
}
```

---

6. Ponieważ aplikacja uruchamia wątki w tle, trzeba jeszcze dodać:

```
Console.ReadLine();
```

(na końcu funkcji Main)

7. Procedura przetwarzania wiadomości z danej partycji jest dosyć oczywista:

```
private static async Task eventLoop(EventHubReceiver eventHubReceiver) {
while (true) {
          var edata = await eventHubReceiver.ReceiveAsync();
          if (edata != null) {
            var data = Encoding.UTF8.GetString(edata.GetBytes());
            Console.WriteLine(data);
        }
    }
}
```

8. Proszę uruchomić plik używając linii poleceń. Na przykład: "D:\GIT_TKOPACZ\2016windowsiot-iothub-genericsender\SamplePCClient\IoTClient\ConsoleMonitor\bin\Debug\ConsoleMonitor.exe" HostName=pltkdpepliot2016.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=klucz(własny)

9. Proszę uruchomić aplikację wysyłającą sygnały z RPi.

10. Po chwili powinniśmy zobaczyć ciąg zarejestrowanych zdarzeń

```
D:\GIT_TKOPACZ\PRIVATE_PASSWORDS\MainIotApp>"D:\GIT_TKOPACZ\2016windowsiot-iothub-genericsender\SamplePCClient\IoTClient\ConsoleMonitor\bin\Debug\ConsoleMonitor.exe" HostName=pltkdpepliot2016.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=
Usage:
ConsoleMonitor <iotHubOwner> [<consumerGroup>] [<device>]
0@♥
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13098729227761485,"Dt":"2016-03-03T13:01:43.9639735Z","MsgType":"SPI","DeviceName":"D01"}
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13000977517106549,"Dt":"2016-03-03T13:01:48.0414347Z","MsgType":"SPI","DeviceName":"D01"}
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13000977517106549,"Dt":"2016-03-03T13:01:46.9792403Z","MsgType":"SPI","DeviceName":"D01"}
{"ADC3":0.011730205278592375,"ADC4":0.0048875855327468231,"ADC5":0.0029325513196480938,"ADC6":0.00195503421309872
92,"ADC7":0.00097751710654936461,"Altitude":183.823,"ColorName":"DarkSlateBlue","ColorRaw":{"Red":278,"Green":161
,"Blue":130,"Clear":501},"ColorRgb":{"Red":141,"Green":81,"Blue":66},"Pressure":99136.48,"Temperature":22.2011757
,"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13098729227761485,"Dt":"2016-03-03T13:01:48.2445572Z","MsgType":"ALL","DeviceName":"D01"}
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13098729227761485,"Dt":"2016-03-03T13:01:49.0874394Z","MsgType":"SPI","DeviceName":"D01"}
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13098729227761485,"Dt":"2016-03-03T13:01:50.1204268Z","MsgType":"SPI","DeviceName":"D01"}
{"Potentiometer1":0.53958944281524923,"Potentiometer2":0.53567937438905178,"Light":0.13098729227761485,"Dt":"2016-03-03T13:01:51.1505665Z","MsgType":"SPI","DeviceName":"D01"}
```

11. Pełny kod rozwiązania:

```
        using Microsoft.Azure.Devices.Common;
using Microsoft.ServiceBus.Messaging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleMonitor
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Usage: \r\n" +
                              "ConsoleMonitor <iotHubOwner> [<consumerGroup>] [<device>] ");
            string connection = args[0];

            string consumerGroupName = "$Default";
            if (args.Length>1) consumerGroupName = args[1];
            string deviceName = "";
            if (args.Length>2) deviceName = args[2];
            EventHubClient eventHubClient = null;
            EventHubReceiver eventHubReceiver = null;
```

```
            eventHubClient = EventHubClient.CreateFromConnectionString(connection, "messages/events");
            var ri = eventHubClient.GetRuntimeInformation();
            if (deviceName != "")
            {
                string partition = EventHubPartitionKeyResolver.ResolveToPartition(deviceName, ri.PartitionCount);
                eventHubReceiver = eventHubClient.GetConsumerGroup(consumerGroupName).CreateReceiver(partition,
DateTime.Now);
                Task.Run(() => eventLoop(eventHubReceiver));
            } else
            {
                EventHubReceiver[] eventHubReceivers = new EventHubReceiver[ri.PartitionCount];


                int i = 0;
                foreach (var partition in ri.PartitionIds)
                {
                    eventHubReceivers[i] = eventHubClient.GetConsumerGroup(consumerGroupName).CreateReceiver(partition,
DateTime.Now);
                    //Task.Run(() => eventLoop(eventHubReceivers[i])); <- very common bug!
                    var r = eventHubReceivers[i];
                    Task.Run(() => eventLoop(r));
                    i++;
                }

            }
            Console.ReadLine();
        }

        private static async Task eventLoop(EventHubReceiver eventHubReceiver)
        {
            while (true)
            {
                var edata = await eventHubReceiver.ReceiveAsync();
                if (edata != null)
                {
                    var data = Encoding.UTF8.GetString(edata.GetBytes());
                    Console.WriteLine(data);
                }
            }
        }
    }
}
```