Lab: Building an IoT application on Windows 10

Contents

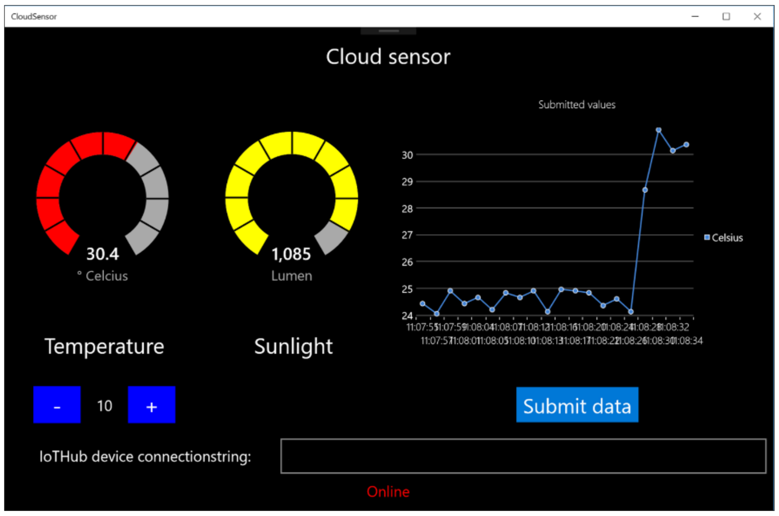
Overview

Summary

In this lab you will build a Windows 10 application (UWP) that can run on a Raspberry Pi, connect to sensors and submit data messages to Azure IoT Hub. The lab will take about 30 to 45 minutes to complete. The lab will walk through an existing application that will be extended with a shield containing sensors (a FEZHAT Shield by GHI Electronics). Cloud connectivity will be added to the code which will submit data that can be used for other labs as well.

Scenario

The scenario behind the app is about monitoring climate with a temperature and light sensors. Many IoT projects involve similar data feeds and this lab aims to give some insight in how data can be generated, submitted and stored for further analysis as is showcased also in the data centric labs Microsoft provides.



Screen 1: CloudSensor, a multiplatform UWP App.

Learning Objectives

Upon completing this lab, you will have hands-on experience with the following functions and concepts related to Windows 10 IoT Core:

- Creating Building and deploying UWP to a remote (Raspberry Pi) device
- Using the FEZHAT shield library to connect sensors and actuators to the device
- Setting up an Azure IoT Hub instance using the Azure Portal
- Using the Azure IoT Hub SDK (C#) to submit values to the cloud
- Managing the Windows 10 IoT Core device through the web management interface
- Using the SDK dev tools to monitor traffic going from the device to IoT Hub

Lab Requirements/Prerequisites

A Microsoft account is required to create and access an Azure subscription. If you have a MSDN subscription it already includes an Azure Subscription with monthly credits. If you don't have a Microsoft account, you can obtain one for free by following the link below:

https://www.microsoft.com/en-us/account

You can create a free Azure trial that only needs a credit card for identification purposes but will not be billed unless explicitly configured otherwise.

https://azure.microsoft.com/en-us/pricing/free-trial/

An alternative is to use Azure passes. These passes contain a code to create a temporary Azure subscription with a finite amount of credits, ideal for workshops and hackathon when you don't have a subscription available. This approach doesn't involve a credit card registration.

Step 1: Setting up your machine and development tools

To get started, you will need to setup your Windows 10 machine with Visual Studio 2015.

1. Make sure you have a machine with Window 10 build 10240 or newer.

2. Install Visual Studio 2015 or above. Community Edition is sufficient.

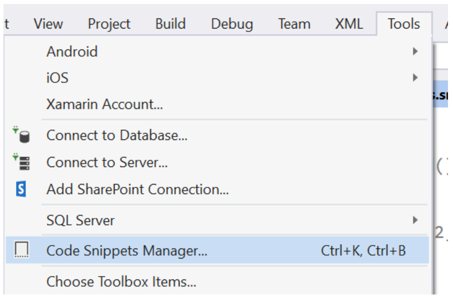
   http://www.visualstudio.com/downloads/download-visual-studio-vs

3. If you choose to install a different edition of VS 2015, make sure to do a **Custom** install and select the checkbox **Universal Windows App Development Tools** -> **Tools** and **Windows SDK**.

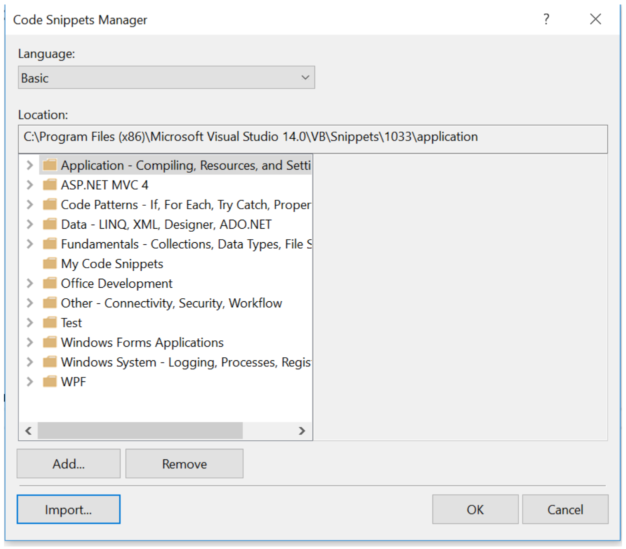
4. You can download Windows IoT Core Project Templates from **http://aka.ms/iotcoretemplates**. Alternatively, the templates can be found by searching for Windows IoT Core Project Templates in the Visual Studio Gallery (**https://visualstudiogallery.msdn.microsoft.com**) or directly from Visual Studio in the Extension and Updates dialog (**Tools > Extensions and Updates > Online**).

5. To respect your valuable time, we included a **code snippet file** in the project that makes it easy and effortless to insert the correct code by just typing ‘snip1’ of ‘snip2’ and then hitting the TAB key where ‘snip1’ is the name of a specific piece of code. Import the snippet file in Visual Studio like this:

   a Copy the file location of the *iotlabsnippets.snippet* file in the project (right click and ‘Copy path’ is the quickest way to do so) and open the snippet manager.


t View Project Build Debug Team XML Tools
Android
iOS
Xamarin Account...
Connect to Database...
Connect to Server...
Add SharePoint Connection...
SQL Server
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...

   b Click the ‘Import’ button.



Language:
Basic

Location:
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VB\Snippets\1033\application

- Application - Compiling, Resources, and Setti
- ASP.NET MVC 4
- Code Patterns - If, For Each, Try Catch, Proper
- Data - LINQ, XML, Designer, ADO.NET
- Fundamentals - Collections, Data Types, File S
- My Code Snippets
- Office Development
- Other - Connectivity, Security, Workflow
- Test
- Windows Forms Applications
- Windows System - Logging, Processes, Regis
- WPF

   c Navigate to the file *iotlabsnippets.snippet* at the root of the project to import it and add it to a category (f.e. My Code Snippets).

   d Type the snippet name at the correct location in the code editor and press TAB to paste the snippet code. All steps in this lab will state the name of the snippet to use.

   e If you have issues setting up the snippets you can always copy the code by opening the snippet file.

6. Make sure you have enabled **Developer mode** in Windows 10 by following these instructions: http://aka.ms/enabledevmode.

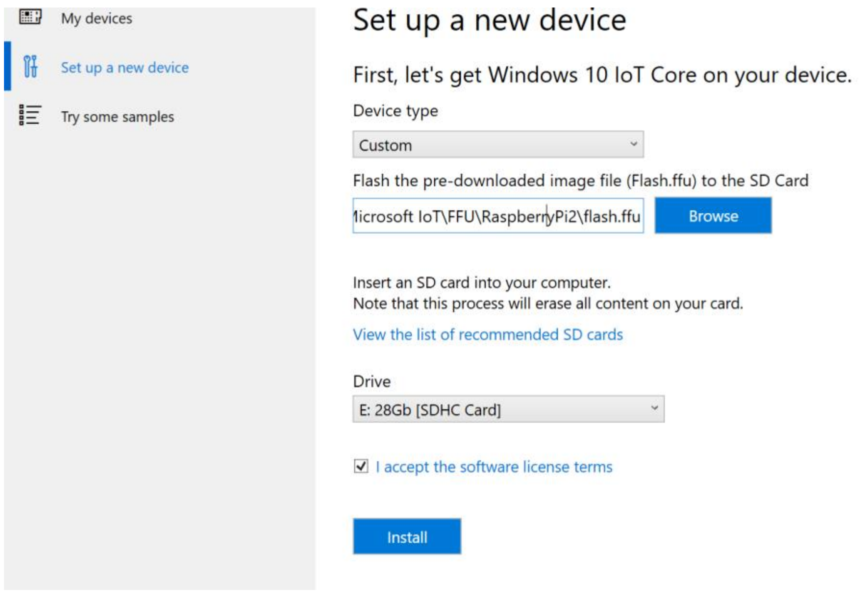
7. To register your devices in the Azure IoT Hub Service and to monitor the communication between them manually you can install the Azure Device Explorer: http://aka.ms/deviceexplorer.

8. There is a very helpful tool to manage your W10 IoT Core devices called the **Windows 10 IoT Core Dashboard** that lets us monitor the network for online devices and makes it easy to flash new SD-cards and do basic device configuration. You can download it at http://aka.ms/iotdashboard.

Step 2: Setting up your device

Now we will flash an SD card with a recent build of Windows 10 IoT Core. We use the IoT Dashboard to make this a very simple procedure. We will use a .FFU image file to flash the SD card that includes the Windows 10 IoT Core bits plus a few pre-installed UWP apps. Eventually you can create your own .FFU images to include your configuration, UI and background apps but we will be using Visual Studio (or the Device Portal) to configure the device and install apps on it.

1. Open the IoT Dashboard tool and head over to the second tab to **setup a new device**. **Insert you SD-card** and instead of using the default Raspberry Pi 2 option in the dropdown you can also select 'Custom' which will let us **select a specific .FFU** file.



Set up a new device

First, let's get Windows 10 IoT Core on your device.

Device type: Custom

Flash the pre-downloaded image file (Flash.ffu) to the SD Card: licrosoft IoT\FFU\RaspberryPi2\flash.ffu

Insert an SD card into your computer.
Note that this process will erase all content on your card.

Drive: E: 28Gb [SDHC Card]

☑ I accept the software license terms

By using a specific image, you can flash the card with the latest version of Windows 10 IoT Core and have a look at the most recent additions to the platform.

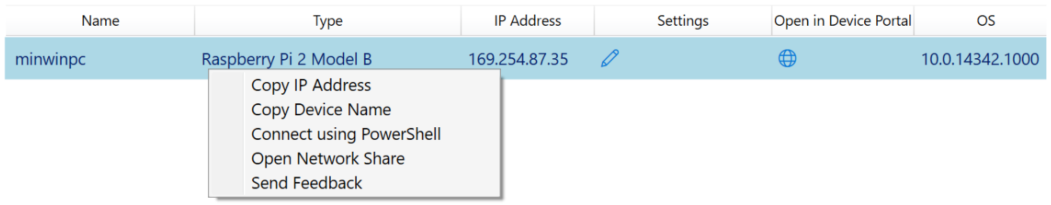
2. Press install and the flash process will commence taking only a minute or so to finish (depending on card speed).

3. Put the SD card in the Raspberry Pi and make sure, when fondling with it, you do not accidently press it as it can pop out again.

4. Plug your Raspberry Pi into a free USB port to power it up.

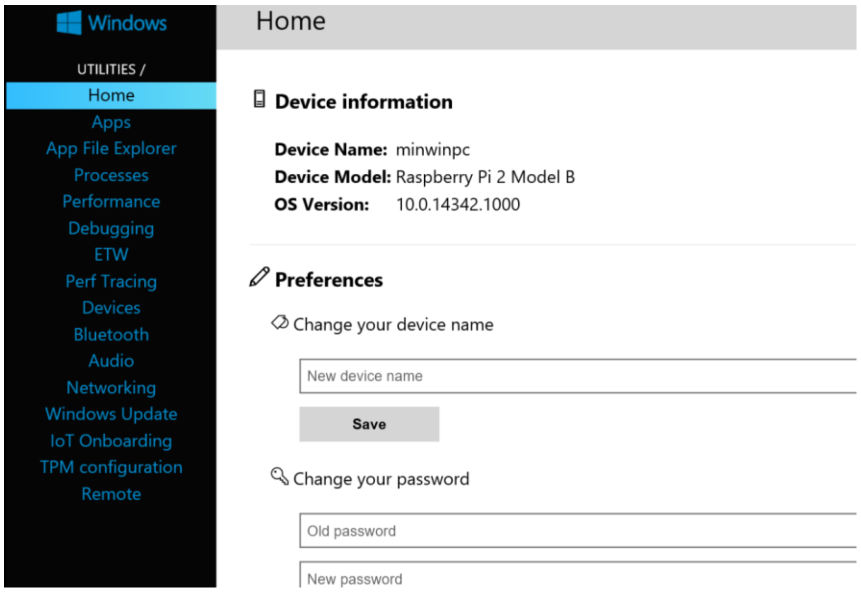
Some ports cannot deliver enough power so keep an USB power adapter at hand in case your device doesn't boot or keeps rebooting. Generally, 500mA or more should be sufficient, depending on the type of device and the added hardware in use.

5. Connect the UTP port with your machine so you can start with a direct connection. The IoT Dashboard tool should find your device (first tab) and the corresponding IP address.



Name	Type	IP Address	Settings	Open in Device Portal	OS
minwinpc	Raspberry Pi 2 Model B	169.254.87.35			10.0.14342.1000

6. Click on the 'Globe icon' to open up the web based **Device Portal** and take a look at the options available and their default settings.



Device information

Device Name: minwinpc
Device Model: Raspberry Pi 2 Model B
OS Version: 10.0.14342.1000

Preferences

Change your device name

Change your password

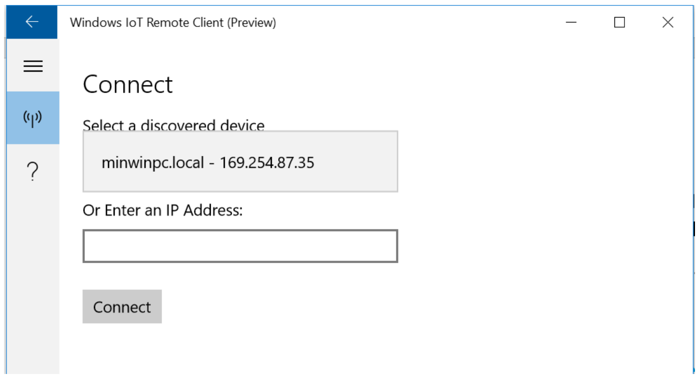
7. It would make sense to rename the device because you might need to connect your device to Wi-Fi to get to the internet (depending on your machine's configuration) and then the IoT Dashboard tool will see all devices within the same subnet.

   To simplify the setup for this lab we encourage you to use a tool called the **Windows IoT Remote Client**. This tool can remote into your device interface using the **NanoRDP service** that is preinstalled in Windows 10 IoT Core eliminating the need to hookup monitors or keyboards to your device to show UI based apps.

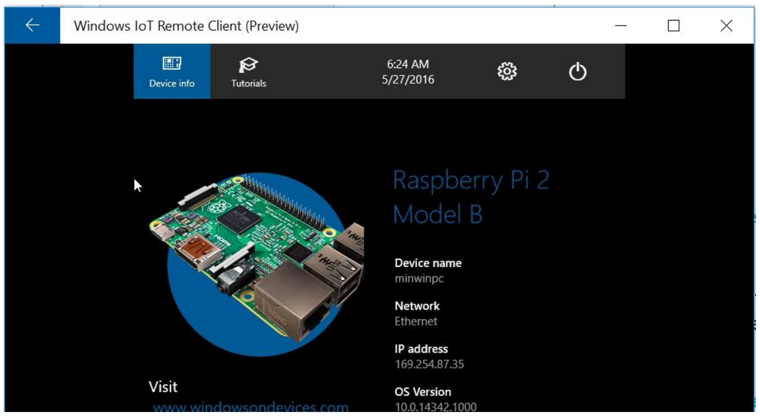
8. To see your device's UI, using an RDP connection, head over to the **Remote** tab in the Device Portal and enable **Windows IoT Remote Server.**

9. Head over to the Home tab in the Device Explorer and increase the Display Resolution to improve the visibility of the interfaces. The device will restart.

10. Start **Windows IoT Remote Client** tool and select the device from the dropdown (match the IP Address in case you see multiple devices).


Windows IoT Remote Client (Preview)
Connect
Select a discovered device
minwinpc.local - 169.254.87.35
Or Enter an IP Address:
Connect

11. Now you see one of the apps running on your device. There is a bit of latency on the connection but the screens, mouse moves and keystrokes from the host machine should be visible.



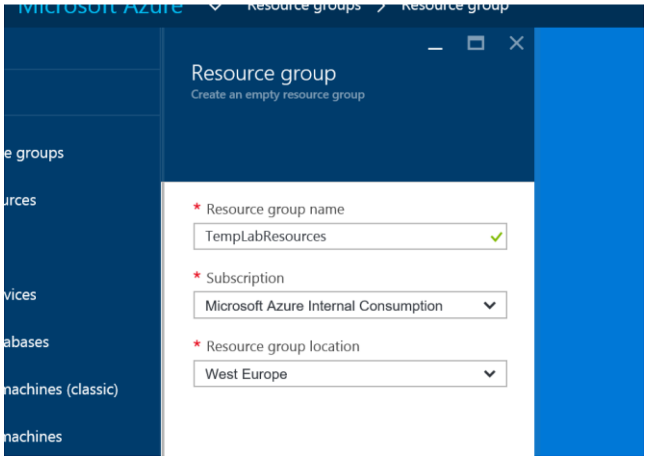
Depending on your network settings (and possibly policy restrictions) your device might not yet have access to the internet. On the device there is a default App called **IoTCoreDefaultApp** (screenshot above) that lets us configure network settings but this can also be done using the Device Portal in the 'Networking' tab.

Step 3: Provisioning an Azure IoT Hub

In this step you will use the Azure Portal to create an instance of an IoT Hub. We will extract a connection string from its configuration and use another dev tool called **Device Explorer** that lets us quickly register a device and extract another, device specific, connection string that will be used in the UWP app.

1. Head over to http://portal.azure.com

2. Create a resource group so you can group all resources you create in the context of this lab or project.



Resource group
Create an empty resource group

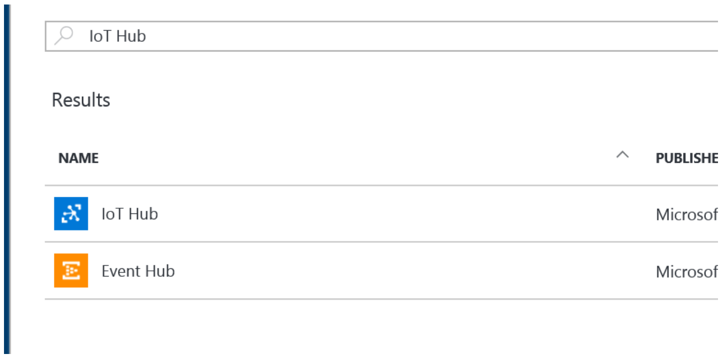
* Resource group name
TempLabResources

* Subscription
Microsoft Azure Internal Consumption

* Resource group location
West Europe

TIP! Give resource groups a recognizable name (like Temp, LabResource, ProjectXYZ, etc.) so you can delete obsolete resources in a single step later on.

3. Click on the 'resource group' and click the '+Add button' and enter 'IoT Hub' as a search term and select IoT Hub. Or select it from the IoT category under the + button at the left of the portal screen.

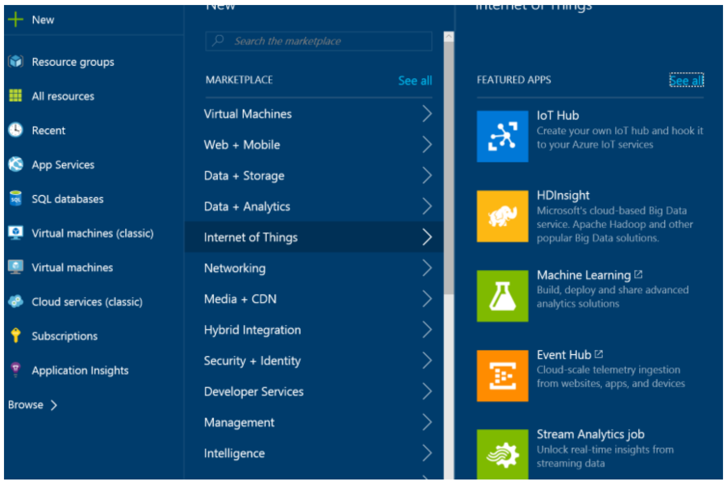


IoT Hub

Results

NAME	PUBLISHE
IoT Hub	Microsof
Event Hub	Microsof

Screen 2: Search option including market place offerings by partners and Microsoft



IoT Hub
Create your own IoT hub and hook it to your Azure IoT services

HDInsight
Microsoft's cloud-based Big Data service. Apache Hadoop and other popular Big Data solutions.

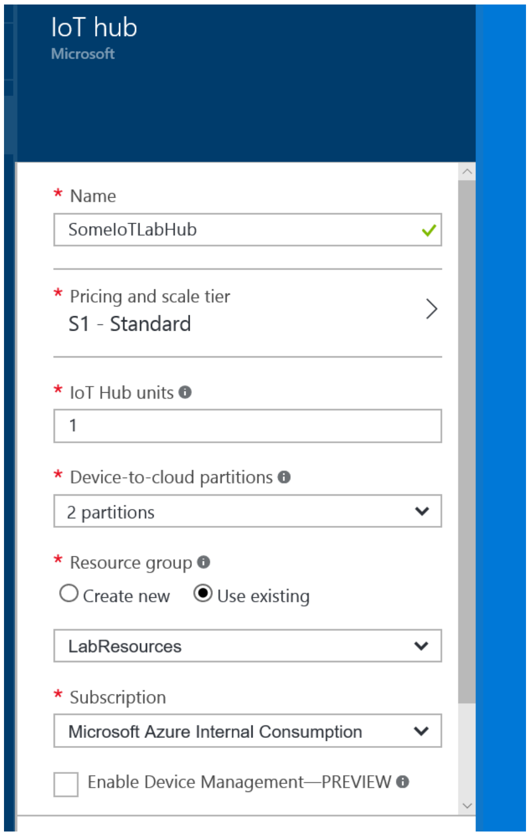
Machine Learning
Build, deploy and share advanced analytics solutions

Event Hub
Cloud-scale telemetry ingestion from websites, apps, and devices

Stream Analytics job
Unlock real-time insights from streaming data

Screen 3: Category selection option showing multiple IoT related services.

4. Give the hub a unique name (as it will get a public endpoint) and configure the number of units (1 is more than enough for this lab) to scale on the ingress side and define how many partitions messages should be placed into (for scaling on the egress side). Select your **Resource Group** and in the **Pricing and scale tier** tab select the free tier to start with 8000 messages per day and press the Create button.



IoT hub
Microsoft

* Name
SomeIoTLabHub ✓

* Pricing and scale tier
S1 - Standard

* IoT Hub units
1

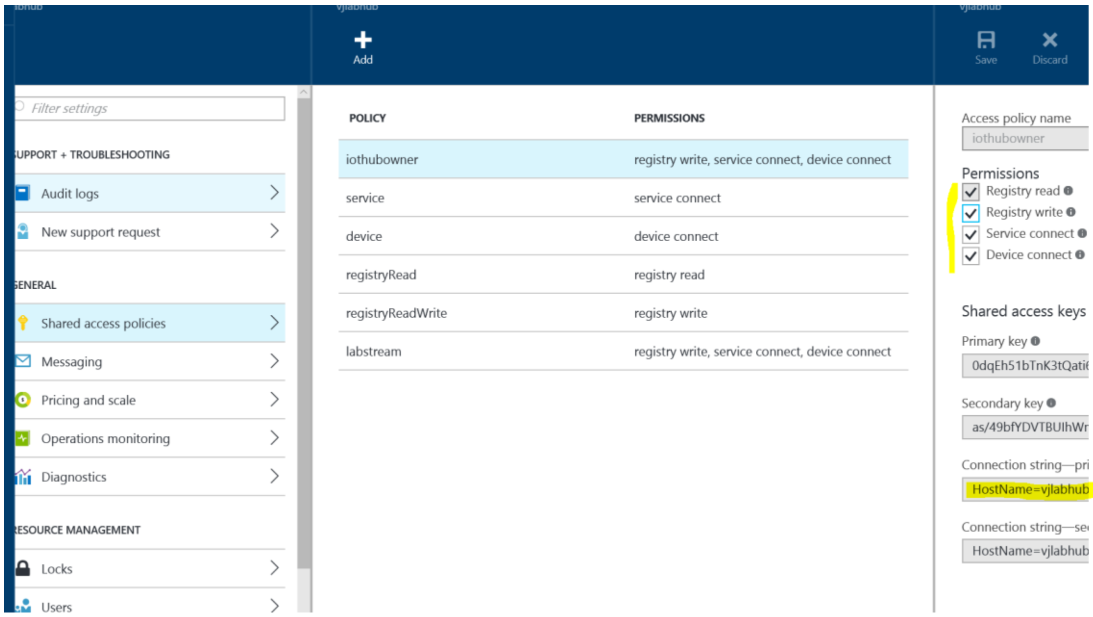
* Device-to-cloud partitions
2 partitions

* Resource group
Create new [✓ Use existing]
LabResources

* Subscription
Microsoft Azure Internal Consumption

☐ Enable Device Management—PREVIEW

5. After the hub is created, head over to its settings blade and select the 'Shared access policies' tab so we can copy the connection string that is based on the **iothubowner** policy. This will give all the permissions needed to register and manage devices.



POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write
labstream	registry write, service connect, device connect

Access policy name: iothubowner

Permissions
- ✓ Registry read
- ✓ Registry write
- ✓ Service connect
- ✓ Device connect

Shared access keys

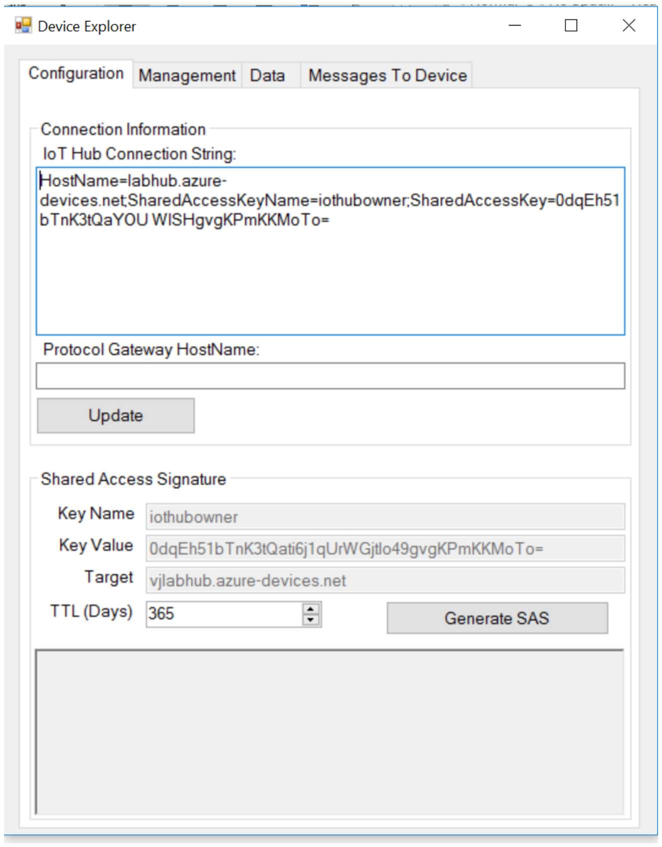
Primary key: 0dqEh51bTnK3tQati

Secondary key: as/49bfYDVTBUIhWr

Connection string—pri: HostName=vjlabhub

Connection string—se: HostName=vjlabhub

6. Run the **Device Explorer** tool you installed during the setup steps and paste the *iothubowner* connection string in the first text box.



Connection Information

IoT Hub Connection String:

```
HostName=labhub.azure-
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=0dqEh51
bTnK3tQaYOU WISHgvgKPmKKMoTo=
```

Protocol Gateway HostName:

Shared Access Signature

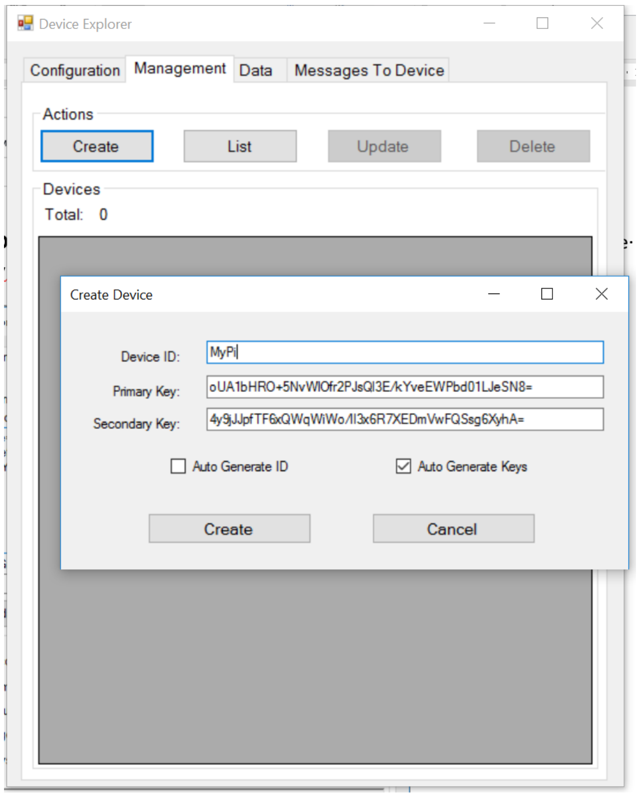
Key Name: iothubowner

Key Value: 0dqEh51bTnK3tQati6j1qUrWGjtlo49gvgKPmKKMoTo=

Target: vjlabhub.azure-devices.net

TTL (Days): 365

7. On the second tab, create a new device registration and right click on it in the table to copy the device specific connection string. This contains an ID and key specific to that device so it can be traced, disabled and invoked from the cloud.



Total: 0

Device ID: MyPi

Primary Key: oUA1bHRO+5NvWlOfr2PJsQl3E/kYveEWPbd01LJeSN8=

Secondary Key: 4y9jJJpfTF6xQWqWiWo/Il3x6R7XEDmVwFQSsg6XyhA=

[ ] Auto Generate ID [✓] Auto Generate Keys

8. On the third tab you can monitor the stream of data coming into the hub from a specified device. We will come back to this later on in the lab.

9. You now have an active Hub, a device registration and all configuration items needed to build a solution.

Step 4: Running the starter solution on an IoT device

In this step we will be using a starter solution that contains the framework for our IoT App. The app is going to be deployed to the device to test the configuration.

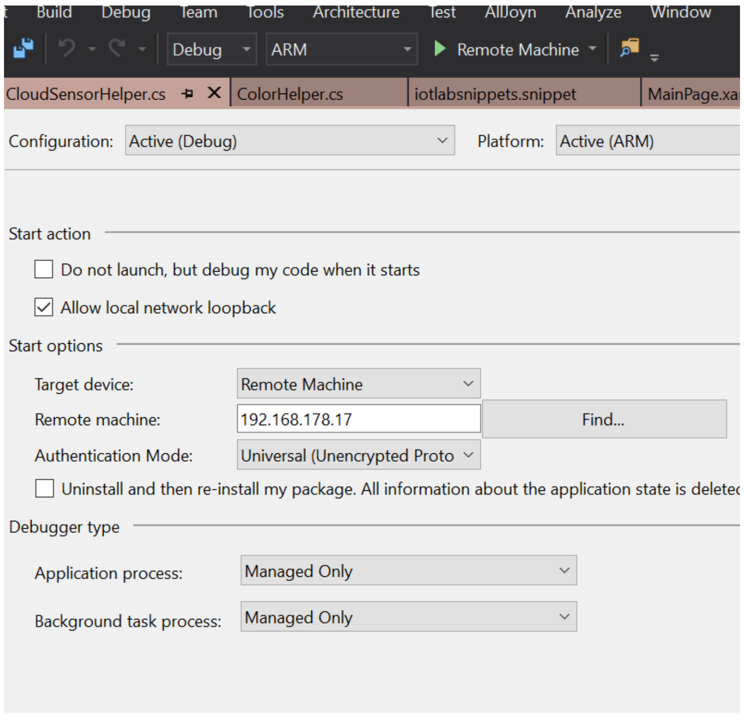
1. Download the starter solution from https://github.com/valeryjacobs/AzureCloudSensor

2. Run the App locally to check for a flawless build and to see the starter app in its current state. Not too much will happen as it is still under construction at this point.

   The app will show randomized values. We will extend this with real values coming from the shield but in case you do not have one, you can still use this app for the purpose of sending data messages to the cloud. Use the on-screen button to offset the temperature to simulate varying environmental conditions.

3. Set the target platform to ARM and select 'Remote Machine' as a deploy target.

4. Use the IoT Dashboard to copy the device's IP address and use the **Universal** setting for the authentication mode.



Debug | ARM | Remote Machine

Configuration: Active (Debug)   Platform: Active (ARM)

Start action

☐ Do not launch, but debug my code when it starts

☑ Allow local network loopback

Start options

Target device: Remote Machine

Remote machine: 192.168.178.17

Authentication Mode: Universal (Unencrypted Proto

☐ Uninstall and then re-install my package. All information about the application state is delete

Debugger type

Application process: Managed Only

Background task process: Managed Only

5. Right-click the project and start deploying to your device. Initially this can take longer as there are framework dependencies that need to be installed. This is where the UTP connection will save us some time compared to Wi-Fi.

6. If you are working on a remote development machine which has no access to your Raspberry Pi you can right-click on the project, select 'Store' then 'Build App packages' and build an application package. The resulting package file can be uploaded to the device using the Device Portal interface in the 'App' tab.

7. Use the Windows IoT Remote Client tool to see the app in action ON THE DEVICE! (Remember, there's a bit of screen delay).

Step 5: Building the UWP app's sensor & cloud connectivity

In this step we will be adding sensor connectivity to read temperature and luminosity values and make use of a library provided by the creators of the FEZHAT shield for the Raspberry Pi.

1. Note the NuGet packages (*project.json* file) called **GHIElectronics.UWP.Shields.FEZHAT** and **Microsoft.Azure.Devices.Client** that are preinstalled in this starter solution. During the lab we will be using these namespaces to add functionality to the app.

2. There is a helper class called CloudSensorHelper that we will append some code to so it will be able to connect with the FEZHAT shield and read out values.

    a. Add the snippet named 'snip1' in place of the *Init()* method of this class. If you are not sure where to put it, look for a comment line stating its location (f.e. '//SNIP1…'). This snippet instantiates an object that represents the shield and initializes the LED lights to green.

    b. Add snippet 'snip2' to the *Sense()* method in the *if* statement. This will readout the temperature and luminosity values and set the LED lights to a corresponding value in case you do not have a screen or RDP session available.

    c. Add snippet 'snip3' to the class to include a connection string property and implement the *SendMeasurement()* method which formats a message object (JSON payload) and submits it to Azure IoT Hub.

    d. Switch to *MainPage.xaml.cs* and add snippet 'snip4' to constructor. This will arrange for the client to store the device connection string so you don't need to enter it every time the app starts.

    e. Add snippet 'snip5' to the *IoTimer_Tick()* handler so the buttons on the FEZHAT shield are enabled for setting the temperature offset, just to make it easy to manipulate the temperature values. To test the lumen value, you can use your phone flashlight or put your finger on top of the sensor (the FEZHAT has prints on the board indicating the location of the sensors).

    f. Add snippet 'snip6' to the *SenseAndSubmit()* method in the *if* statement to call the *SendMeasurement()* method and update the UI.

    g. This concludes all the coding we need to do to finish the app. You can tweak the code to use additional sensors, control two motors, sense motion or connect other hardware on top of the shield (ZigBee, LoRa, and lots more…)

3. When running on a remote device it makes sense to paste the **device connection string** (copied from the Device Explorer in the device registry list) to the first line in the *MainPage* class since it is quite cumbersome to type it into the textbox of the app UI:

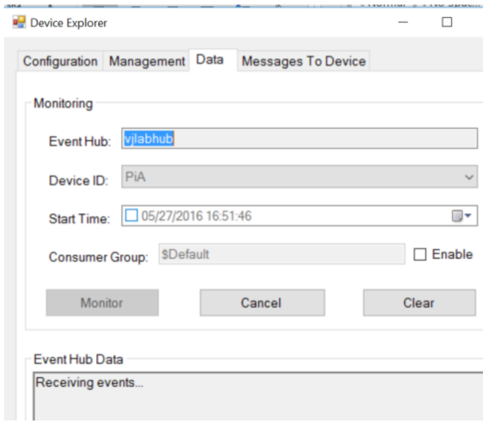
   ```
   private const string hardCodedDeviceConnectionString = "[PUT IT HERE]";
   ```

4. Run the app locally (randomized values) or on the device (real data).

5. Press the 'Submit data' button to start sending measurements to IoT Hub. All data send will be added to the chart with a time indication (lower screen resolutions limit the number of readable values on the x-axis).

6. Press the '+' and '–' buttons to chance the offset for the temperature.

7. Use the Device Explorer to monitor messages coming in from this specific device.



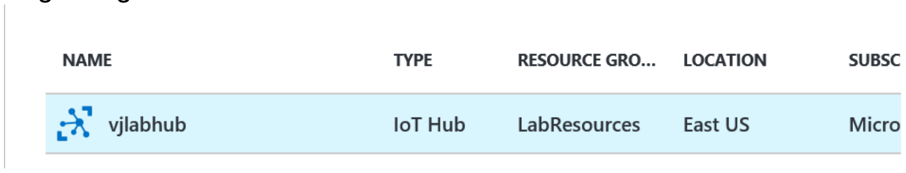
Event Hub: vjlabhub
Device ID: PiA
Start Time: 05/27/2016 16:51:46
Consumer Group: $Default
Enable
Receiving events...

8. This concludes the creation of our IoT Device and its 'gateway' application to the cloud. When finished you can set the app as the default app so it will start whenever the device starts, use the Device Portal to set this up.

Step 6: Preparing for the next step

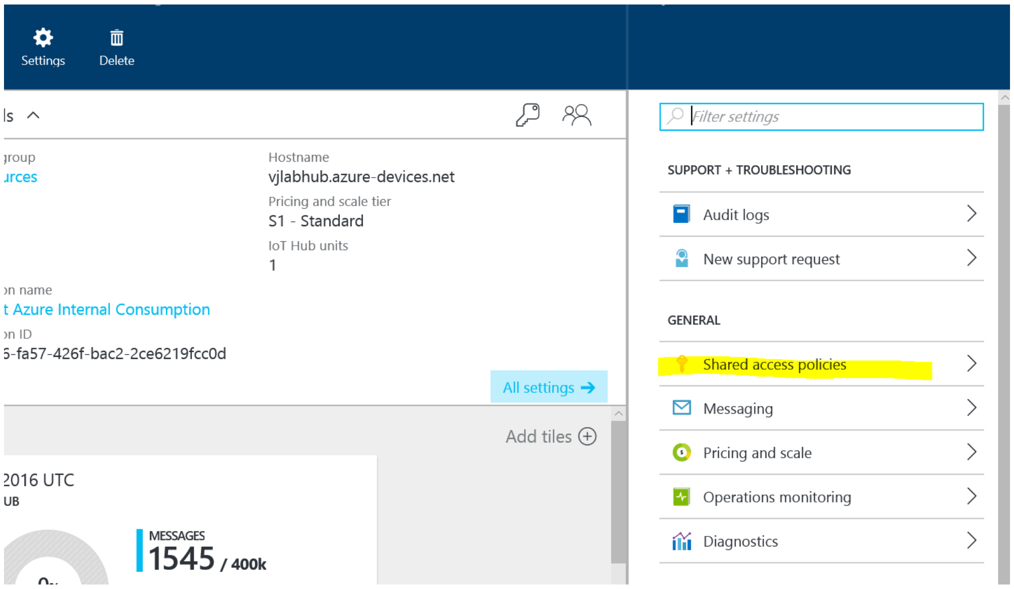
Now we have built an application that sends data to the cloud and we want to use this data in other applications like back-end services and data analytics tools. To prepare for this we need a couple of configuration items from the IoT Hub instance.

1. Head over to the Azure portal and click on the **Resource Group** you created at the beginning of the lab.

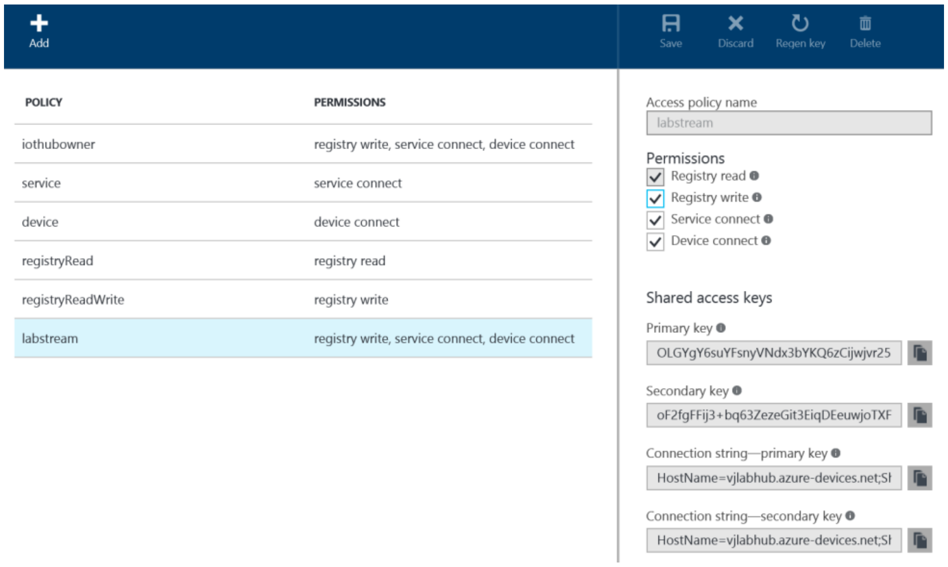


NAME	TYPE	RESOURCE GRO...	LOCATION	SUBSC
vjlabhub	IoT Hub	LabResources	East US	Micro

2. Click on the settings button to open the settings pane on the right and click on **Share Access Policies** tab.



3. Here you can add policies that result in connection credentials that correspond to specific permission on the hub. Policies are also useful for monitoring and auditing purposes in that it can separate activities on the IoT Hub.



POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write
labstream	registry write, service connect, device connect

Access policy name
labstream

Permissions
✓ Registry read
✓ Registry write
✓ Service connect
✓ Device connect

Shared access keys

Primary key
OLGYgY6suYFsnyVNdx3bYKQ6zCijwjvr25

Secondary key
oF2fgFFij3+bq63ZezeGit3EiqDEeuwjoTXF

Connection string—primary key
HostName=vjlabhub.azure-devices.net;SI

Connection string—secondary key
HostName=vjlabhub.azure-devices.net;SI

4. Mark down you policy name and Primary key as you will need it later on to connect f.e. Azure Stream Analytics to process messages coming in.

Conclusion

This concludes the *Using Windows 10 IoT Core for IoT* lab. To recap, you have successfully created a UWP app that is hooked up to sensor hardware and connected it with the cloud to submit IoT data.

There are multiple services in Azure that are designed to work with this data and transform the raw data into information that companies can use to innovate, optimize, revive and grow. Apart from being able to read from IoT Hub on the back-end with code (like the Device Explorer does), Azure Stream Analytics is designed to act like a data pump that connects to IoT Hub and, using a SQL-like query, is able to filter and manipulate the data passing through. There are multiple options for forwarding these query results like SQL Database, Blob storage, Power BI and many more.

Head over to the other labs that will show what you can do with the data to build a complete IoT scenario. Remember, raw data still has to be transformed into valuable information and that is where the challenge of IoT actually starts.