IoT Development Lab 4

Azure IoT

***Abstract*:** This Azure IoT Lab is prepared to show how to configure Azure environment, upload the accelerometer data from MinnowBoard Max to Azure, and get the data form Azure.

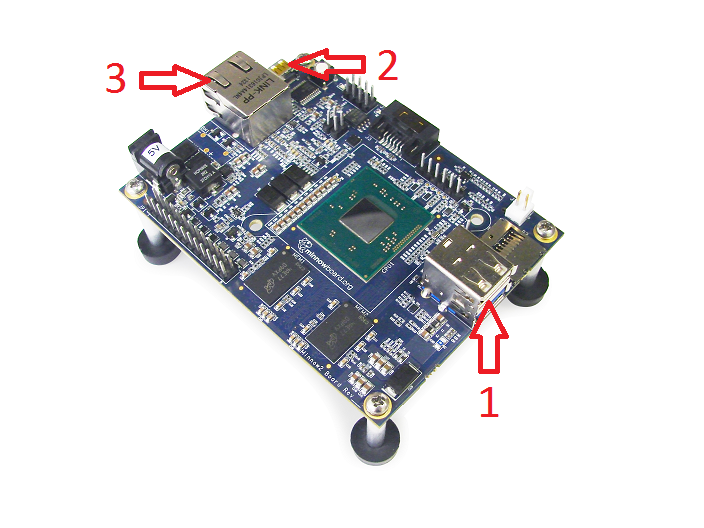
***Prerequisites*:** Azure account

***Install and configure*:**

You will need:

* A desktop computer running Windows 10
* A MinnowBoard Max running Windows 10 IoT Core
* Wired ADXL345 accelerometor
* USB Wi-Fi dongle

1. Connect a USB keyboard to one of the USB ports on the board.
2. Connect an HDMI monitor to the micro HDMI port on the board.
3. Connect a network cable to the Ethernet port on the board and the development machine. Make sure your MinnowBoard Max and the development machine are on the same network.



* The MBM should automatically boot to the card (this initial boot may take up to 2 minutes, subsequent boots should take less than 30 seconds). If it does not, it will boot to the UEFI shell, and you will have to execute the following in the UEFI shell to boot Windows:

fs1:

efi\boot\bootia32.efi

Copyright (c) 2015 Microsoft. All rights reserved.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Contents

[**Lab objective** 4](#_Toc435108569)

[**Exercise 1: Azure Setup** 4](#_Toc435108570)

[**Exercise 2: Update the Accelerometer UWP App** 9](#_Toc435108571)

[**Exercise 3: Connect MinnowBoard Max to Azure** 12](#_Toc435108572)

[**Exercise 4: Deploy the Accelerometer UWP App** 14](#_Toc435108573)

[**Exercise 5: Azure Database** 17](#_Toc435108574)

[**Exercise 6: Develop a Mobile Service UWP App** 18](#_Toc435108575)

[**Reference** 22](#_Toc435108576)

**Lab objective**

Participants will learn about the Azure environment and the process for developing UWP Apps that communicates with Azure. The lab starts from setting up Azure Mobile Services. Then, two Apps will be developed in this lab: one is running in MinnowBoard Max to send data to Azure, another is running on your development machine to get data from Azure. The tutorials for this lab will be presented as hands-on tasks that have been grouped into the sections:

* Azure Setup
* Update the Accelerometer UWP App
* Connect MinnowBoard Max to Azure
* Deploy the Accelerometer UWP App
* Azure Database
* Develop a Mobile Service UWP App

**Exercise 1: Azure Setup**

In this exercise, you will setup the Azure environment by creating Microsoft Azure Mobile Services with your Azure account that you will need for this lab.

1. Connect the Development Machine and MinnowBoard Max as the topology:

Wi-Fi

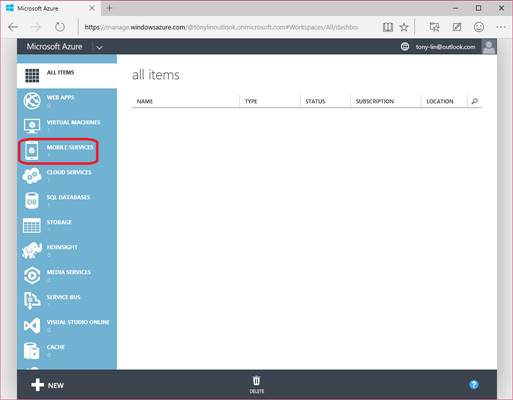
Network Cable

Development Machine

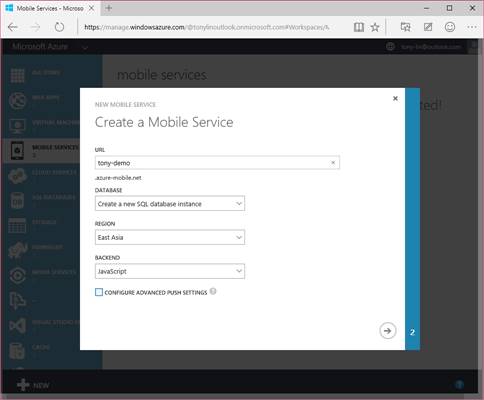
MinnowBoard Max

Internet

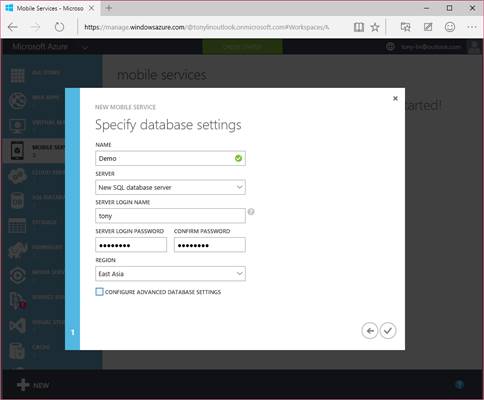
1. Login Azure portal <https://manage.windowsazure.com/> to create Microsoft Azure Mobile Services (create the database and table)
2. Select “Mobile Services” in the left side



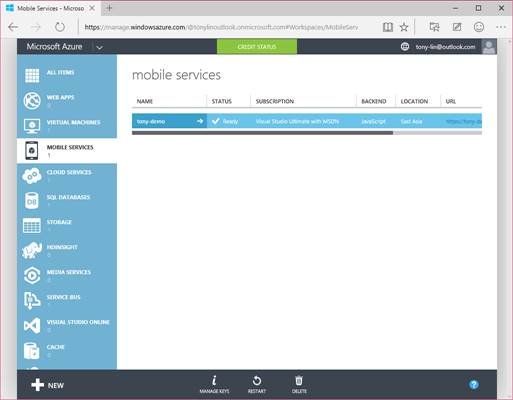
1. Create a mobile service by filling in the fields to add the SQL server database from Azure



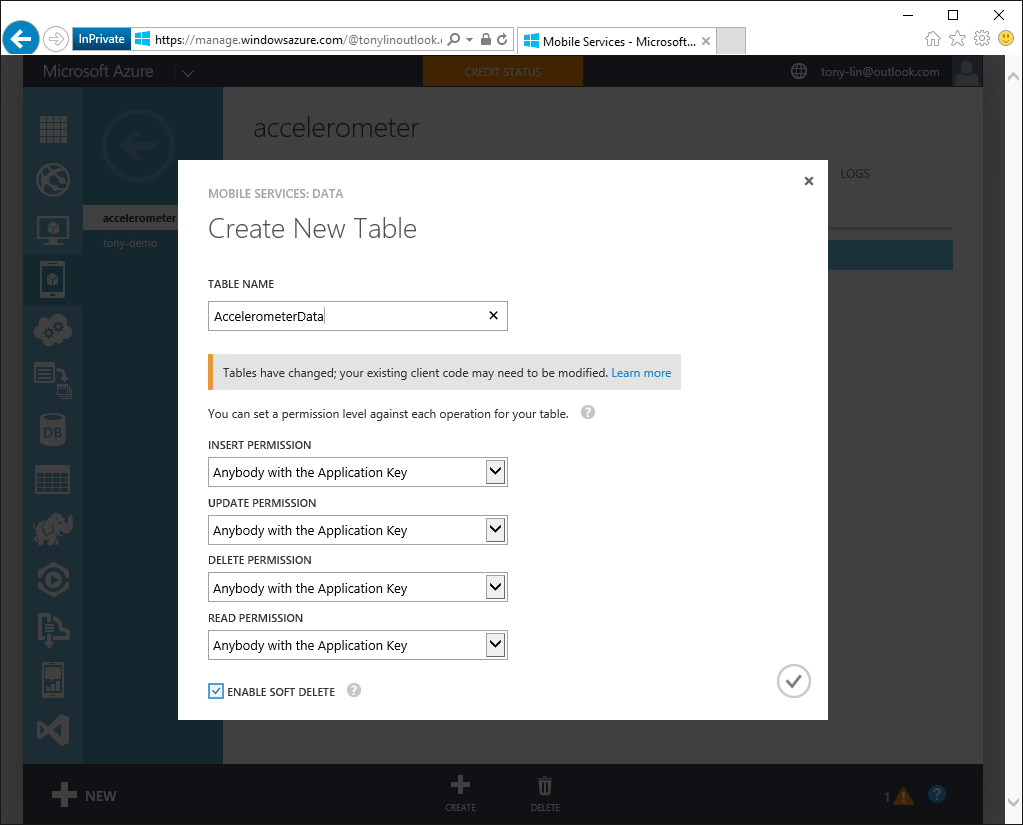
1. Specify the database, the server login name and password



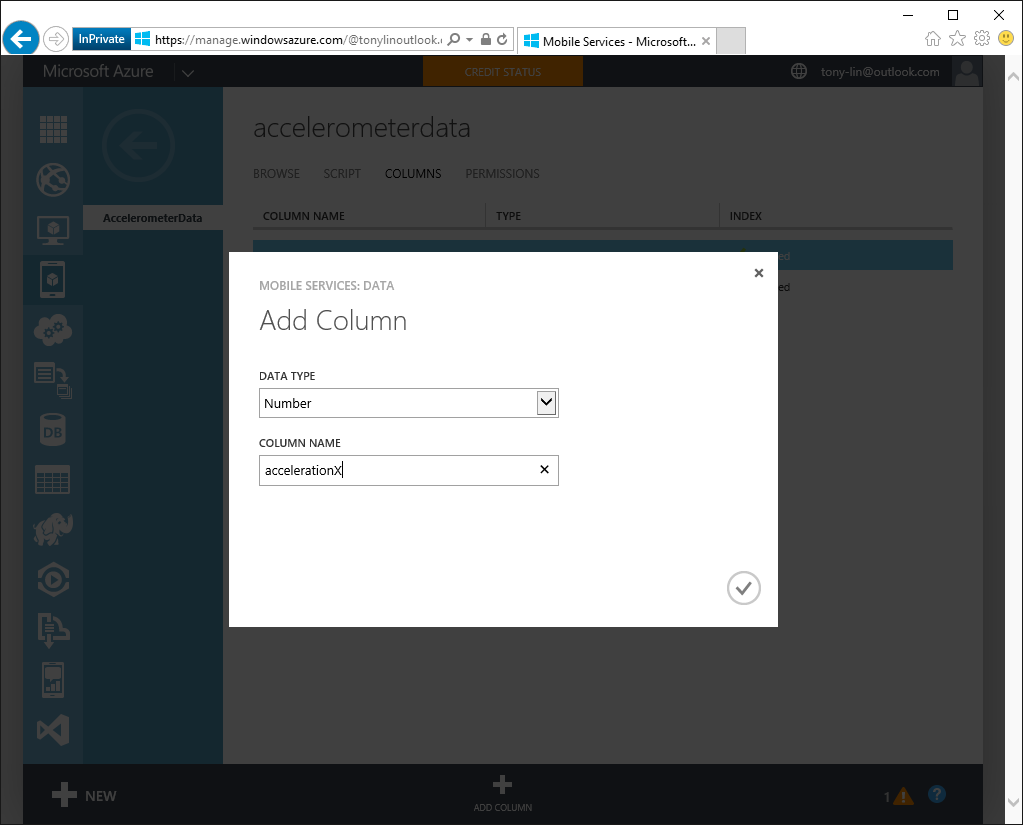
1. You’ll see the mobile service just created



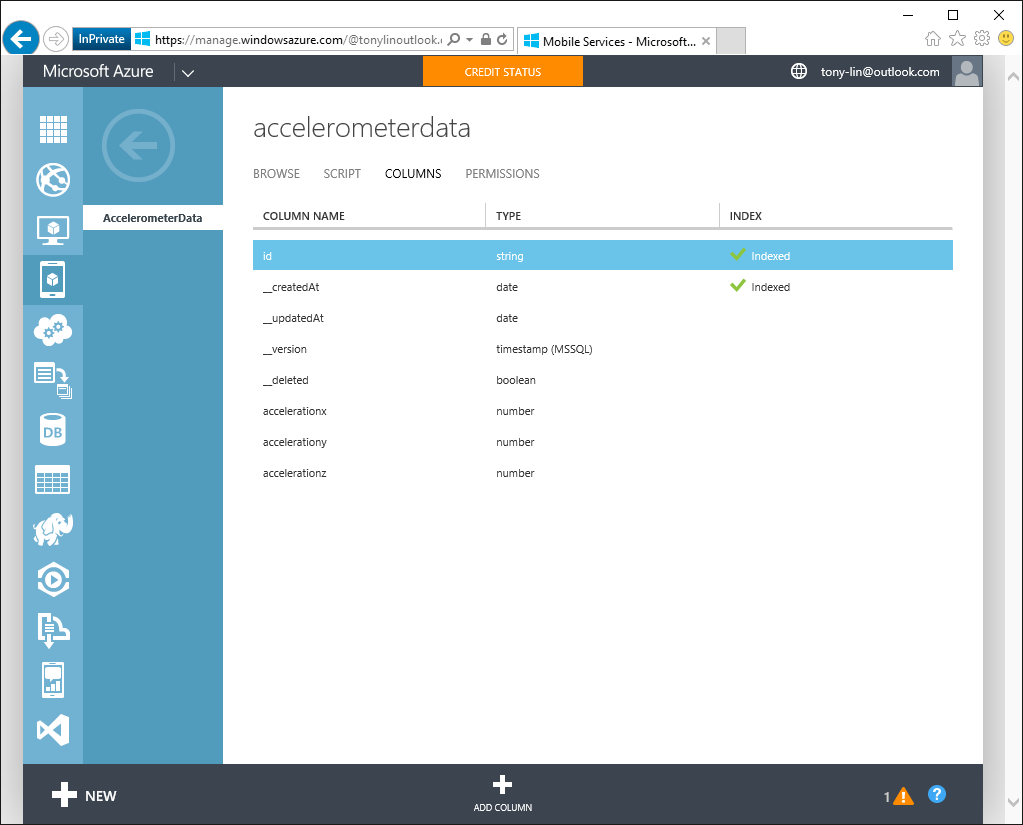
1. Create a table by clicking the mobile service just created > Data > Create a table (e.g. Create a table named AccelerometerData)



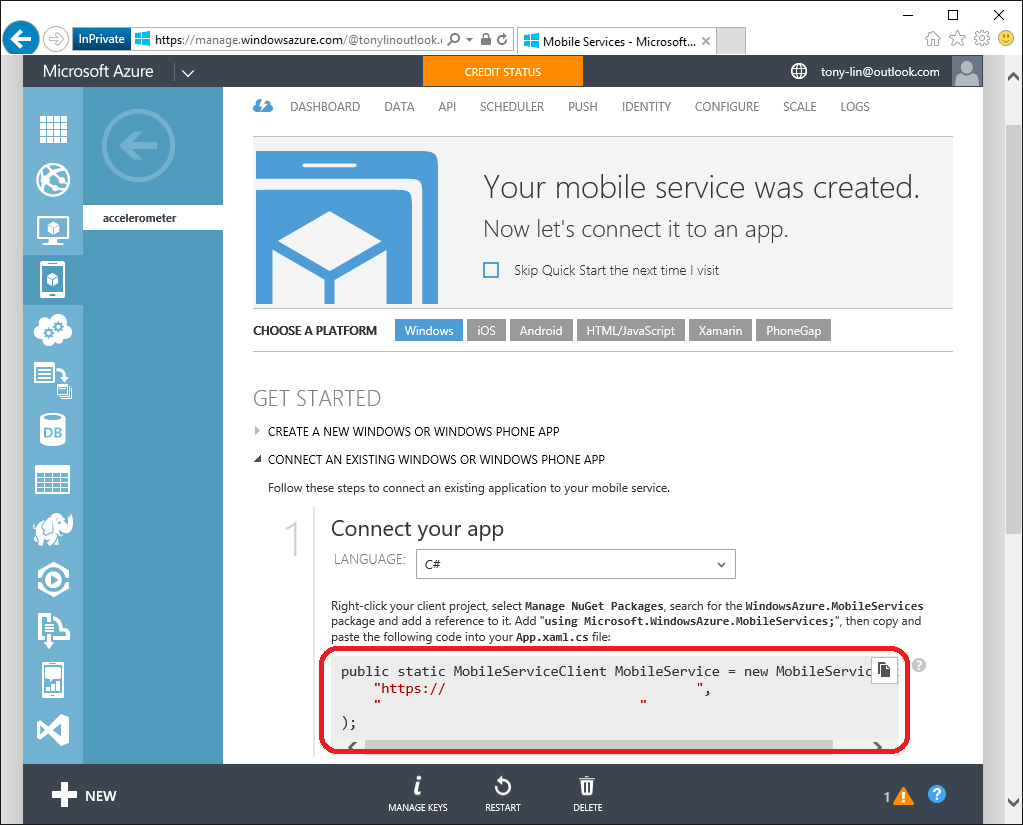
1. Click columns to add columns (accelerationX, accelerationY, accelerationZ)



1. Created columns accelerationX, accelerationY, accelerationZ, and some default columns



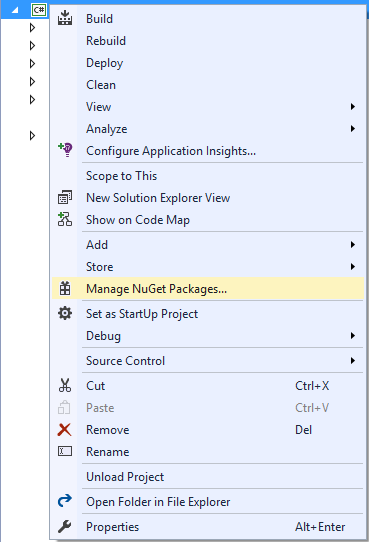
1. Go back to the mobile service just created and get the connection string by expand the section “Connect an existing Windows or Windows phone app”

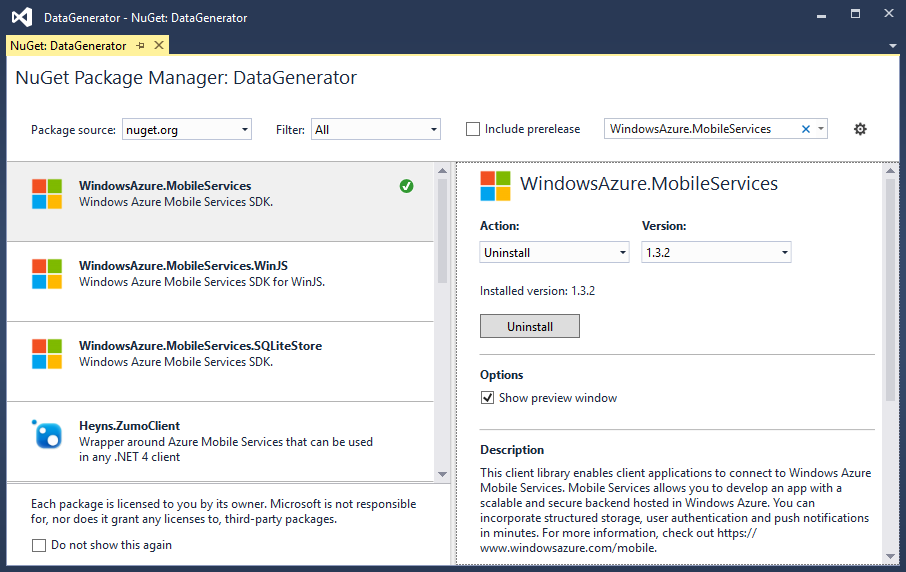


**Exercise 2: Update the Accelerometer UWP App**

In this exercise, you will update the existing Accelerometer UWP (<https://ms-iot.github.io/content/en-US/win10/samples/I2CAccelerometer.htm>) and use WindowsAzure.MobileServices to upload sensor data to Azure.

After mobile service and a table are created from Azure, add NuGet package reference of WindowsAzure.MobileServices in your app along with using statement.





Let’s add some content to the MainPage. From Solution Explorer, select the MainPage.xaml file. We want to add a TextBox to show some interaction. So we will edit the XAML file to add these elements. Locate the <Grid> tag in the XAML section of the designer, and add the following markup.

|  |
| --- |
| <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  <TextBlock x:Name="TextBlockData"  HorizontalAlignment="Center"  VerticalAlignment="Center"/>  </Grid> |

Open the MainPage.xaml.cs file in the project. Add the using as follows:

|  |
| --- |
| using Microsoft.WindowsAzure.MobileServices; |

Creates the MobileServiceClient object that is used to access your mobile service. You can find the AppUrl and AppKey from Azure:

|  |
| --- |
| MobileServiceClient \_mobileService = new MobileServiceClient("AppUrl", "AppKey"); |

Assume the table is named AccelerometerData, and it has following columns: id, accelerationX, accelerationY, accelerationZ, \_\_updatedAt. Create a corresponding typed class in the client-side .NET code as given in the following:

|  |
| --- |
| public class AccelerometerData  {  public string id { get; set; }  public double accelerationX { get; set; }  public double accelerationY { get; set; }  public double accelerationZ { get; set; }  public DateTime \_\_updatedAt { get; set; }  } |

Create a DispatcherTimer and a callback function that will be called for a certain interval in the conductor (public MainPage()). You may need to add “using Windows.UI.Xaml” for DispatcherTimer. The callback function will be called every 1 second in the sample below:

|  |
| --- |
| DispatcherTimer timer = new DispatcherTimer();  timer.Tick += Timer\_Tick;  timer.Interval = new TimeSpan(0, 0, 1);  timer.Start(); |

Now, implement the timer callback function as given in the following, where x, y, and z are the sensor readings from accelerometer. You may set the variables to global variables so that you can access the data in the timer callback function.

|  |
| --- |
| private async void Timer\_Tick(object sender, object e)  {  AccelerometerData data = new AccelerometerData { id = Guid.NewGuid().ToString(),  accelerationX = x,  accelerationY = y,  accelerationZ = z };  await \_mobileService.GetTable<AccelerometerData>().InsertAsync(data);  TextBlockData.Text = String.Format("Sent: X {0}, Y {1}, Z {2}", x, y, z);  } |

**Exercise 3: Connect MinnowBoard Max to Azure**

In this exercise, you will use USB Wi-Fi dongle and configure the MinnowBoard Max to Connect to the Internet (<https://ms-iot.github.io/content/en-US/win10/SetupWiFi.htm>).

Plug the USB Wi-Fi dongle to MinnowBoard Max. Connect the Development Machine and MinnowBoard Max as the topology below:

Wi-Fi

Wi-Fi

Network Cable

Development Machine

MinnowBoard Max

Internet

There are two ways to get MinnowBoard Max connected to Azure:

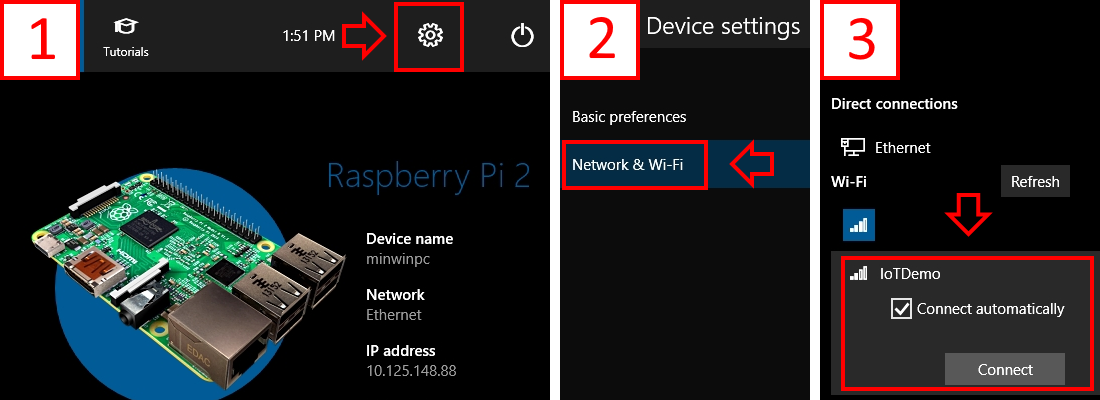
1. Prerequisite: The Windows 10 IoT core device needs a mouse, keyboard, display, and USB WiFi Adapter plugged in.

You can use this to configure or modify WiFi settings after the device has booted.

Click on the gear-shaped settings icon on the homepage

Select Network & Wi-Fi in the left pane

Click on the WiFi network you want to connect to. Supply the password if prompted, and click Connect

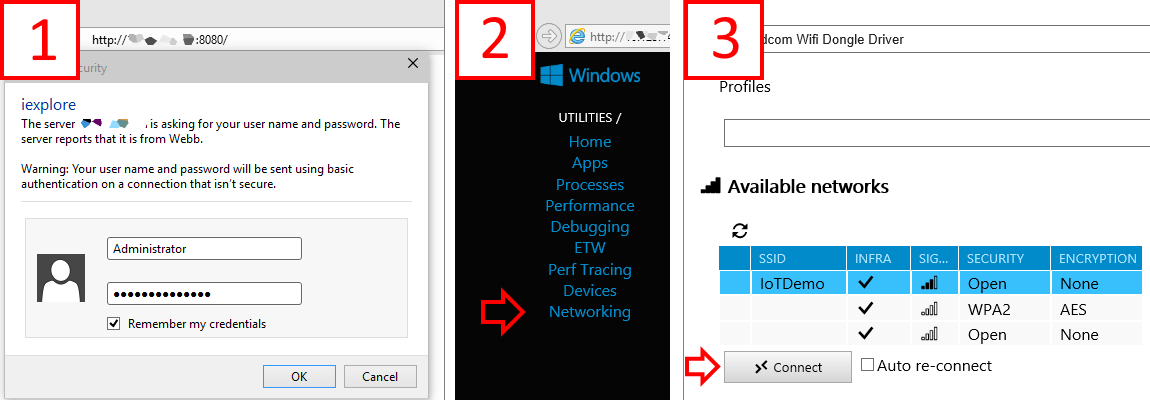


1. If you have device a with no UI, display, or input devices, you can still configure it through web-based management. In Windows IoT Core Watcher, Right Click on your Raspberry Pi, then select Web Browser Here.

Enter Administrator for the username, and supply your password (p@ssw0rd by default)

Click on Networking in the left-hand pane

Under Available networks, select network you would like to connect to and supply the connection credentials. Click Connect

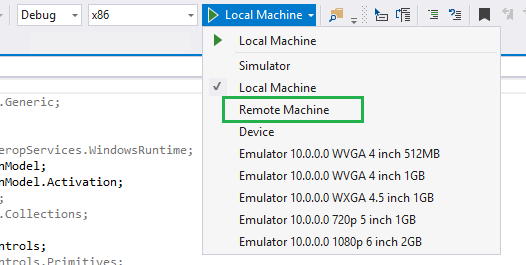


**Exercise 4: Deploy the Accelerometer UWP App**

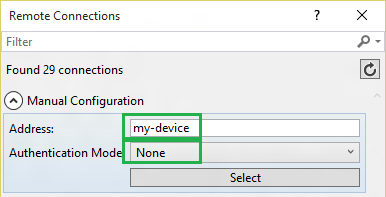
In this exercise, you will deploy the updated UWP App to the MinnowBoard Max (<https://ms-iot.github.io/content/en-US/win10/AppDeployment.htm>).

As you’re building for MinnowBoard Max, select x86 in the Visual Studio toolbar architecture dropdown.

Next, in the Visual Studio toolbar, click on the Local Machine dropdown and select Remote Machine

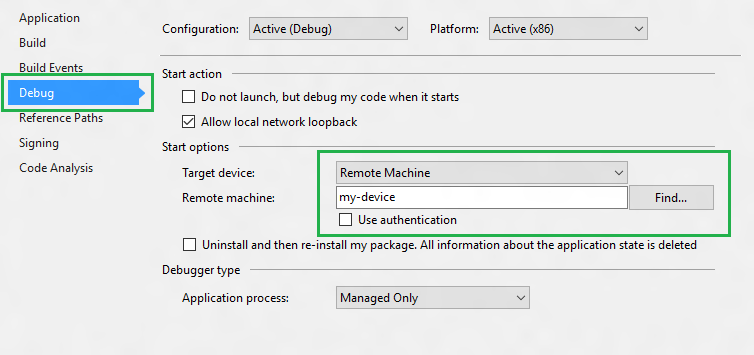


At this point, Visual Studio will present the ‘Remote Connections’ dialog. Put the IP address or name of your Windows IoT Core device (in this example, we’re using ‘my-device’) and select None for Windows Authentication. Then click ‘Select’.



Couple of notes:

* You can use the IP address instead of the Windows IoT Core device name.
* You can verify and/or modify these values navigating to the project properties (select ‘Properties’ in the Solution Explorer) and choose the ‘Debug’ tab on the left:



Now we’re ready to deploy to the remote Windows IoT Core device. Simply press F5 (or select Debug > Start Debugging) to start debugging our app. You should see the app come up in Windows IoT Core device screen, and you should be able to click on the button.

If you see an error message in Visual Studio when deploying that says “Unable to connect to the Microsoft Visual Studio Remote Debugger named ‘XXXX’. The Visual Studio 2015 Remote Debugger (MSVSMON.EXE) does not appear to be running on the remote computer.”, the Remote Debugger may have timed out. Connect to your device using PowerShell and query the active processes by running tlist. If at least one msvsmon.exe is not present in that list, you’ll need to run this command to restart the Remote Debugger (or you can reboot your device): schtasks /run /tn StartMsvsmon.

You can set breakpoints, see variable values, etc. To stop the app, press on the ‘Stop Debugging’ button (or select Debug > Stop Debugging).

Having successfully deployed and debugged your first UWP application, create a Release version by simply changing the Visual Studio toolbar configuration dropdown from Debug to Release. You can now build and deploy your app to your device by selecting Build > Rebuild Solution and Build > Deploy Solution.

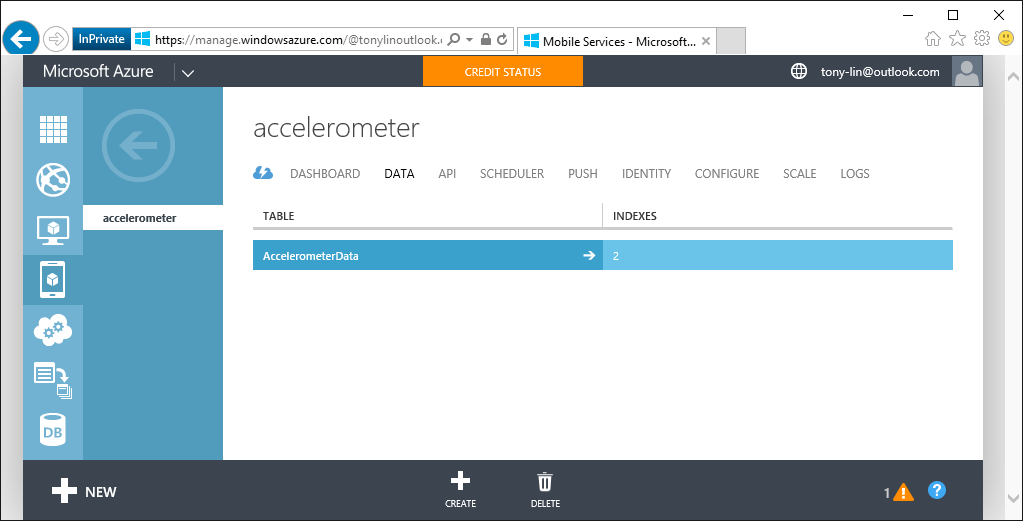
Check if the App is running and sending data to Azure:



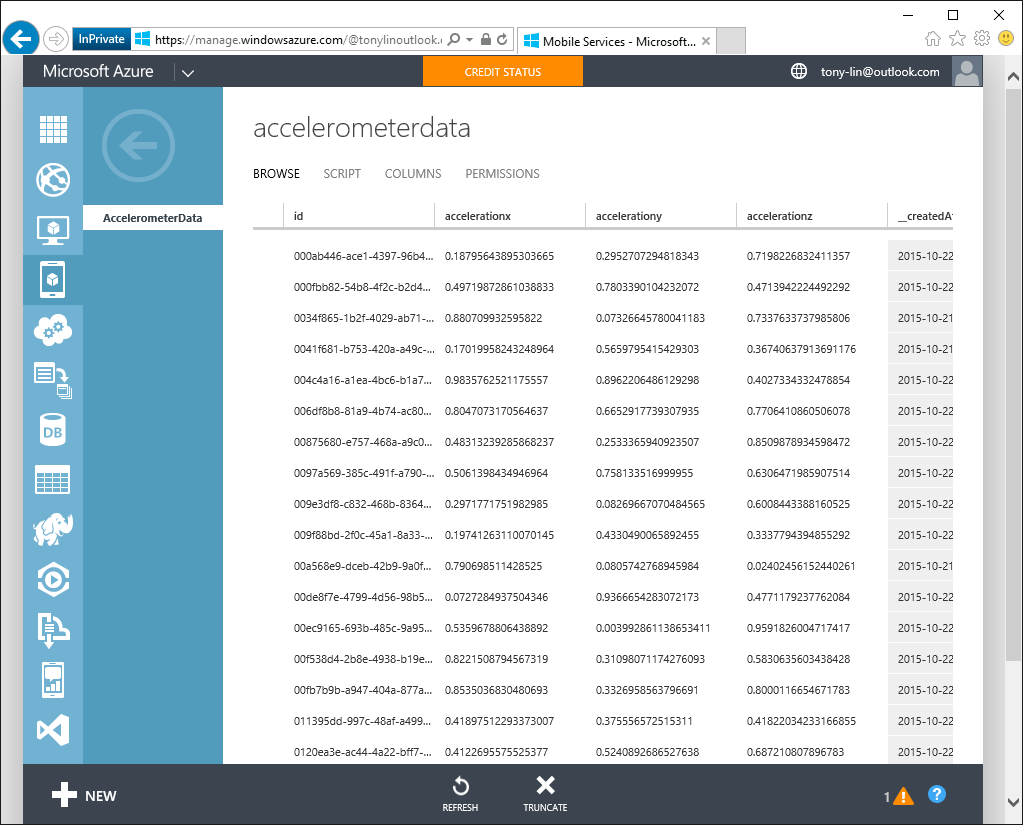
**Exercise 5: Azure Database**

In this exercise, you will connect to Azure database and check if the sensor data from MinnowBoard Max is uploaded to Azure by the App.

Connected to Azure and navigate to mobile services > Data > the table:



Check the table to see if the sensor data is uploaded:



**Exercise 6: Develop a Mobile Service UWP App**

In this exercise, you will develop and deploy another UWP App that gets the data from Azure.

Connect the Development Machine and MinnowBoard Max as the topology below:

Wi-Fi

Wi-Fi

Development Machine

MinnowBoard Max

Internet

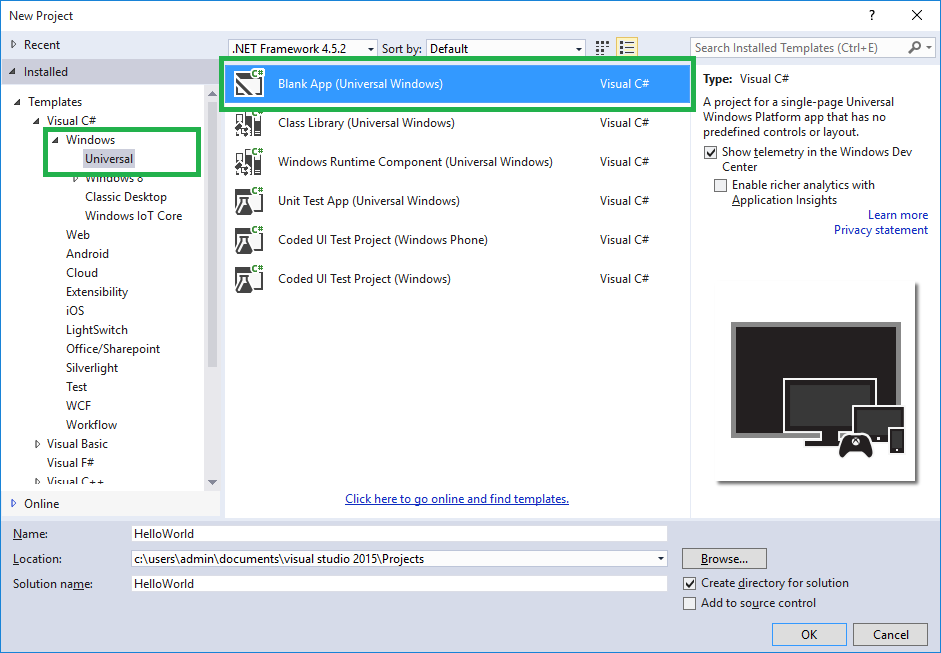
Start Visual Studio 2015.

Create a new project (File > New Project...)

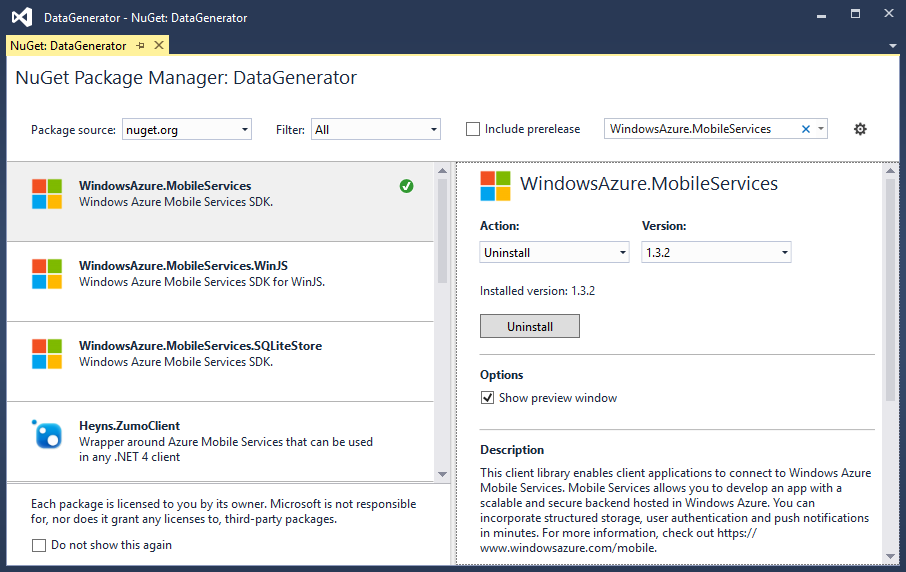
In the New Project dialog, navigate to Universal as shown below (in the left pane in the dialog: Templates | Visual C# | Windows | Universal).

Select the template Blank App (Windows Universal).

Remember to give a good name to your first app! In this example, we called the project “MobileService”



Add NuGet package reference of WindowsAzure.MobileServices in your app along with using statement.



Let’s add some content to the MainPage. From Solution Explorer, select the MainPage.xaml file. We want to add a TextBox to show some interaction. So we will edit the XAML file to add these elements. Locate the <Grid> tag in the XAML section of the designer, and add the following markup.

|  |
| --- |
| <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  <TextBlock x:Name="TextBlockData"  HorizontalAlignment="Center"  VerticalAlignment="Center"/>  </Grid> |

Open the MainPage.xaml.cs file in the project. Add the using as follows:

|  |
| --- |
| using Microsoft.WindowsAzure.MobileServices; |

Creates the MobileServiceClient object that is used to access your mobile service. You can find the AppUrl and AppKey from Azure. Besides, create the IMobileServiceTable object:

|  |
| --- |
| MobileServiceClient \_mobileService = new MobileServiceClient("AppUrl", "AppKey");  IMobileServiceTable<AccelerometerData> \_mobileServiceTable; |

Assume the table is named AccelerometerData, and it has following columns: id, accelerationX, accelerationY, accelerationZ, \_\_updatedAt. Create a corresponding typed class in the client-side .NET code as given in the following:

|  |
| --- |
| public class AccelerometerData  {  public string id { get; set; }  public double accelerationX { get; set; }  public double accelerationY { get; set; }  public double accelerationZ { get; set; }  public DateTime \_\_updatedAt { get; set; }  } |

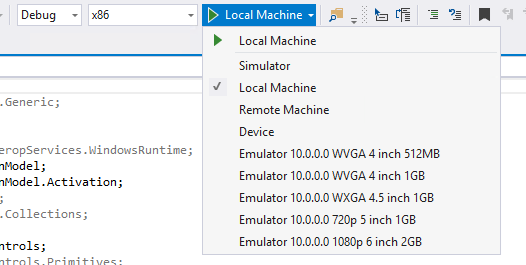
Initialize IMobileServiceTable with the table information. Create a DispatcherTimer and a callback function that will be called for a certain interval in the conductor (public MainPage()). You may need to add “using Windows.UI.Xaml” for DispatcherTimer. The callback function will be called every 1 second in the sample below:

|  |
| --- |
| \_mobileServiceTable = \_mobileService.GetTable<AccelerometerData>();  DispatcherTimer timer = new DispatcherTimer();  timer.Tick += Timer\_Tick;  timer.Interval = new TimeSpan(0, 0, 1);  timer.Start(); |

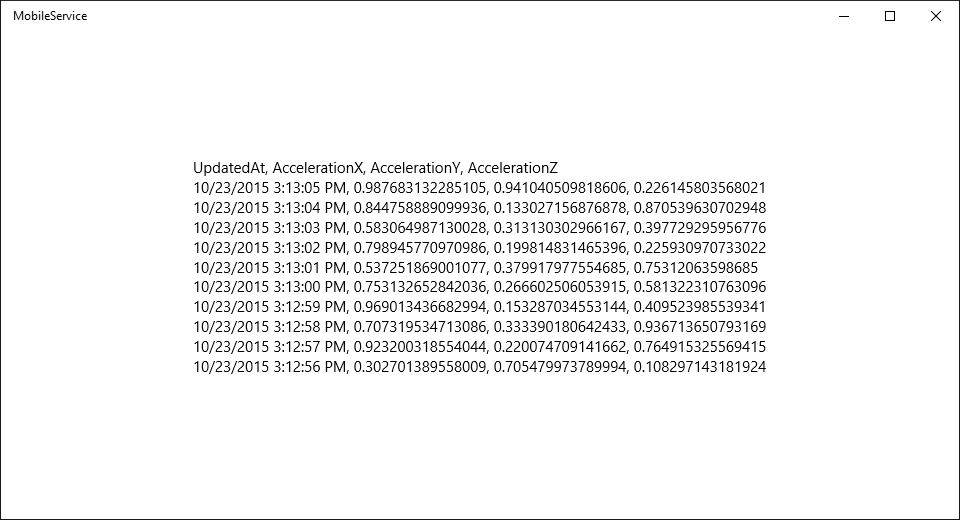
Now, implement the timer callback function as given in the following:

|  |
| --- |
| private async void Timer\_Tick(object sender, object e)  {  List<AccelerometerData> data = new List<AccelerometerData>();  data = await \_mobileServiceTable.OrderByDescending(item => item.\_\_updatedAt)  .Take(10)  .ToListAsync();  TextBlockData.Text = "UpdatedAt, AccelerationX, AccelerationY," +  "AccelerationZ\r\n";  for (int i = 0; i < data.Count; i++)  {  TextBlockData.Text += data[i].\_\_updatedAt + ", " +  data[i].accelerationX + ", " +  data[i].accelerationY + ", " +  data[i].accelerationZ + "\r\n";  }  } |

Deploy the App to the local machine:



Check if the App is running and getting data form Azure:



**Reference**

* <https://azure.microsoft.com/en-us/services/app-service/mobile/>
* <https://azure.microsoft.com/en-us/documentation/articles/mobile-services-dotnet-backend-windows-store-dotnet-get-started/>
* <https://channel9.msdn.com/Series/Windows-Azure-Mobile-Services>