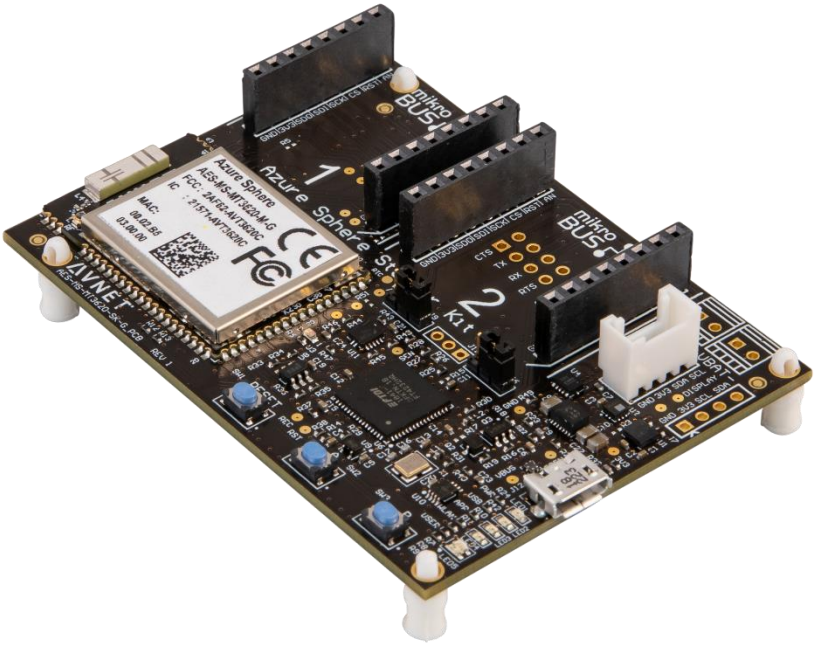
Avnet Technical Training Course

Azure Sphere: Getting Data to the Cloud

Lab 4



Azure Sphere SDK:	20.01
Training Version:	v8
Date:	19 February 2020

Introduction

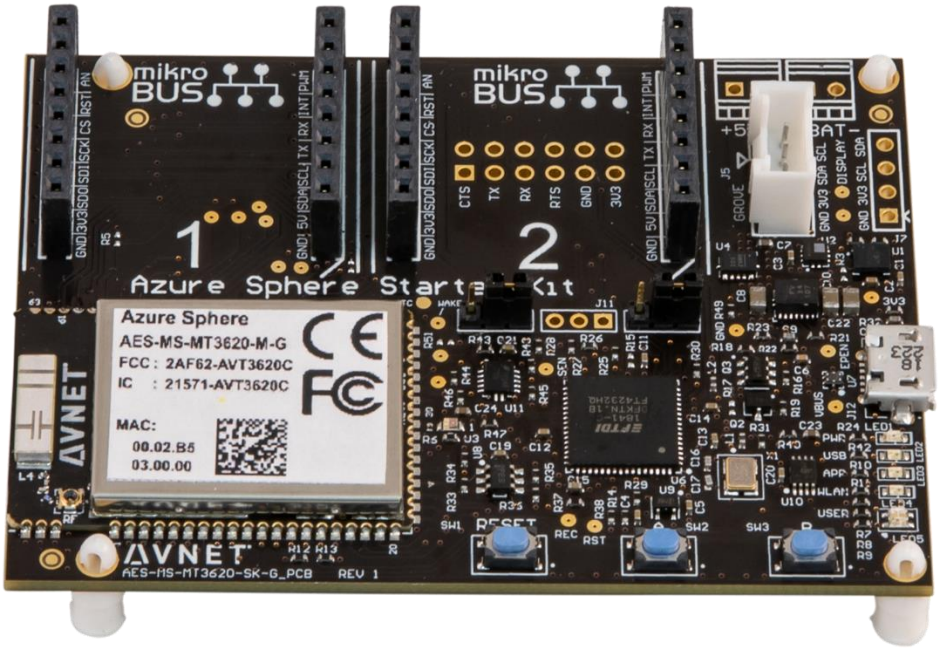
This Lab will walk the student through connecting the example application to an Azure Device Provisioning Service (DPS), then to an IoT Hub. We'll send data to our IoT Hub and then configure a Time Series Insights (TSI) environment to visualize our data.

Avnet Azure Sphere Starter Kit Overview

The Avnet Azure Sphere Starter Kit from Avnet Electronics Marketing provides engineers with a complete system for prototyping and evaluating systems based on the MT3620 Azure Sphere device.

The Avnet Azure Sphere MT3620 Starter Kit supports rapid prototyping of highly secure, end-to-end IoT implementations using Microsoft's Azure Sphere. The small form-factor carrier board includes a production-ready MT3620 Sphere module with Wi-Fi connectivity, along with multiple expansion interfaces for easy integration of off-the-shelf sensors, displays, motors, relays, and more.

The Starter Kit includes Avnet's MT3620 Module. Having the module on the Starter Kit means that you can do all your development work for your IoT project on the Starter Kit and then easily migrate your Azure Sphere Application to your custom hardware design using Avnet's MT3620 Module.



Avnet Azure Sphere Starter Kit

Lab 4: Objectives

Lab 4 teaches the student how to connect an Azure Sphere application to Azure IoT services to send telemetry data to the cloud, then use a cloud based service, Time Series Insights, to visualize the data.

- Learn how to create an IoT Hub
- Learn how to create a Device Provisioning Service (DPS)
- Configure the example application for the IoT Hub configuration
- Complete a code assignment
- Learn how to create a Time Series Insights (TSI) Environment

Lab 4 builds on the previous labs and should not be started until Labs 0-2 have been completed. Lab 3 is NOT a prerequisite for this lab.

Requirements

Hardware

- A PC running Windows 10 Anniversary Update or later (Version 1607 or greater)
- An unused USB port on the PC
- An Avnet Azure Sphere Starter Kit
- A micro USB cable to connect the Starter Kit to your PC

Software

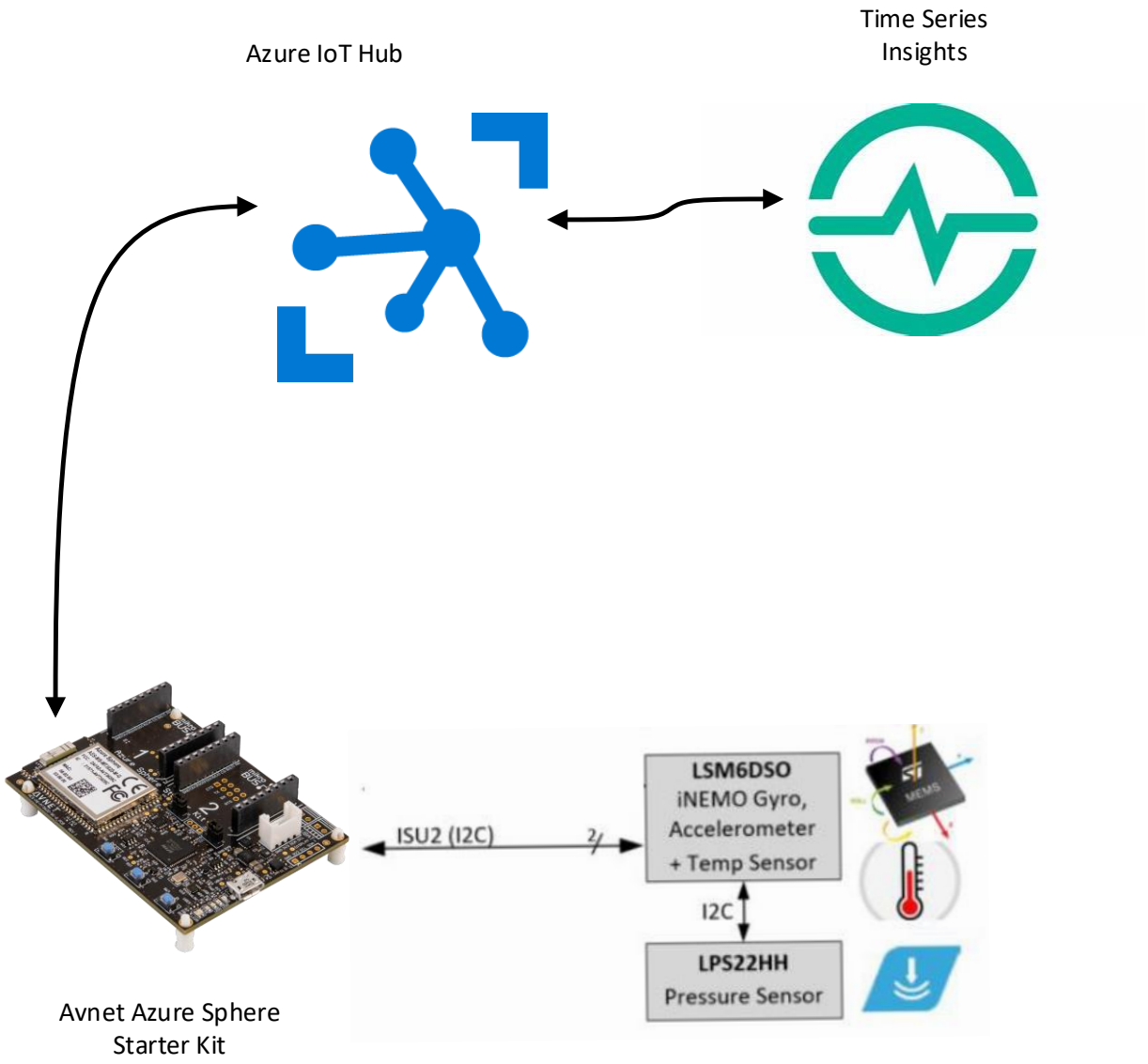
- Visual Studio 2019 version 16.4 or later (Enterprise, Professional, or Community version)
- Azure Sphere SDK0.01 or the current SDK release **installed**

Other

- Your Azure Sphere device must be connected to a Wi-Fi access point or hotspot with access to the internet.
- Labs 4 requires an Azure Account. You can sign up for a free Azure account with a $200 credit here.

The Big Picture

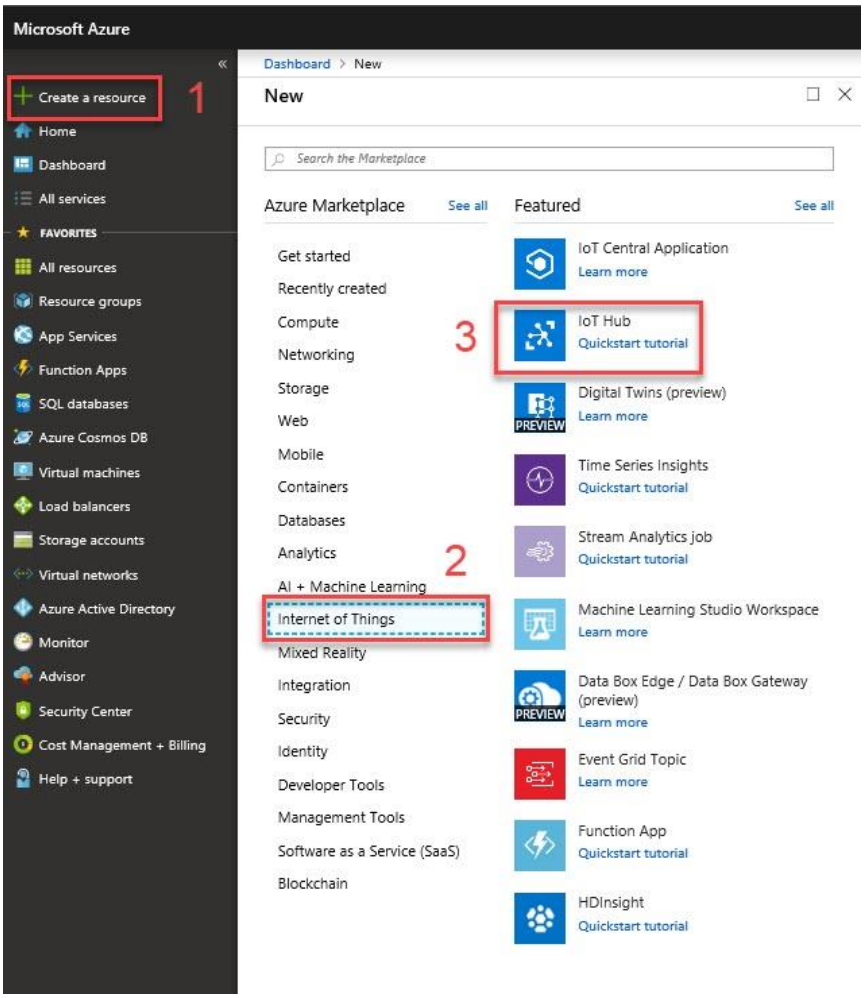
The diagram below shows the system we'll be building out in this lab. Starting at the bottom right of the graphic, we'll use the on-board I2C sensors to read accelerometer and pressure data. That data will be sent as telemetry to an IoT Hub. Next we'll connect a Time Series Insights resource to our IoT Hub so that we can visualize the sensor data in the cloud.



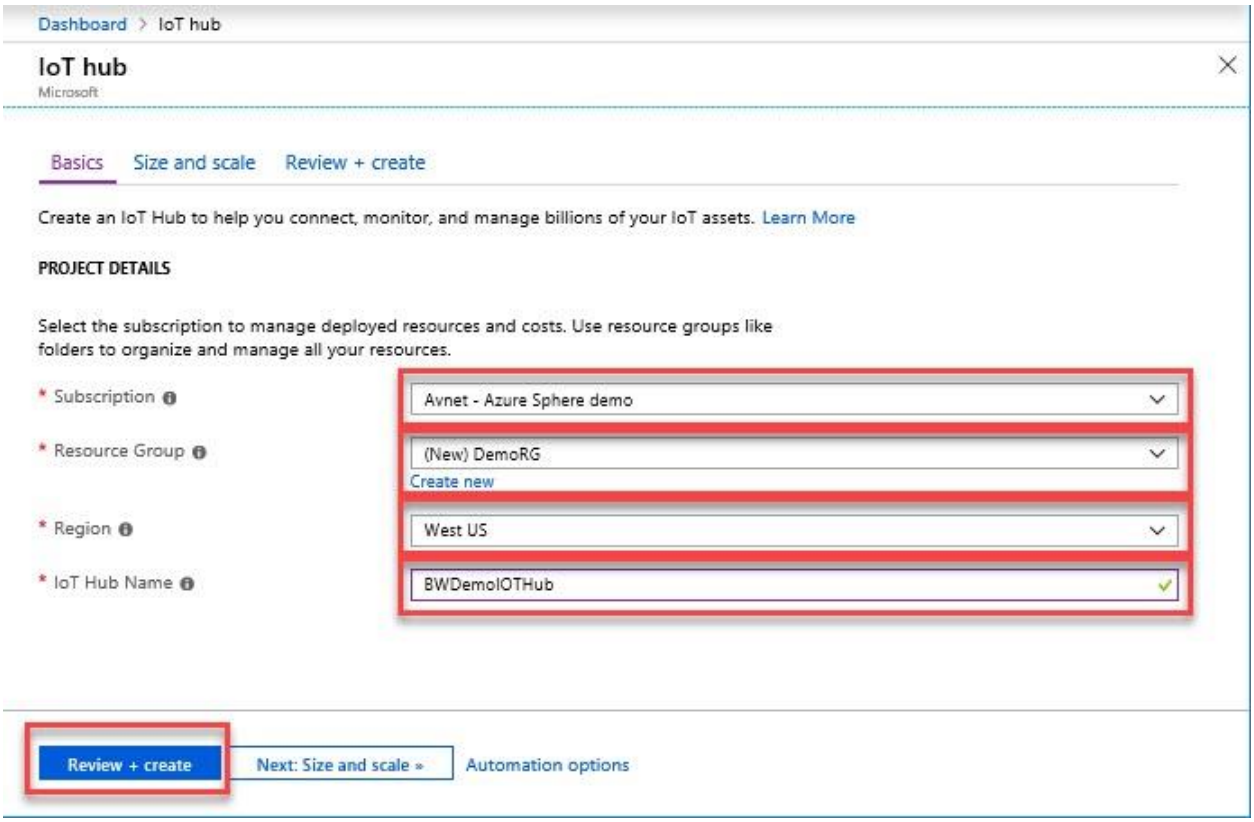
Create an IoT Hub

The first thing we need to do is create an IoT Hub. The IoT Hub is a collection point for IoT devices connecting into Azure. Once a device is connected to an IoT Hub and streaming telemetry data, other Azure services can ingest the data and do meaningful things. A single Azure IoT Hub can manage connections to hundreds of thousands of devices.

- Log into Azure https://portal.azure.com
- Click on **"+ Create a resource" → "Internet of Things" → "IoT Hub"**
  The IoT hub form will open.
    - You can also use the search bar to search for "**IoT Hub**"

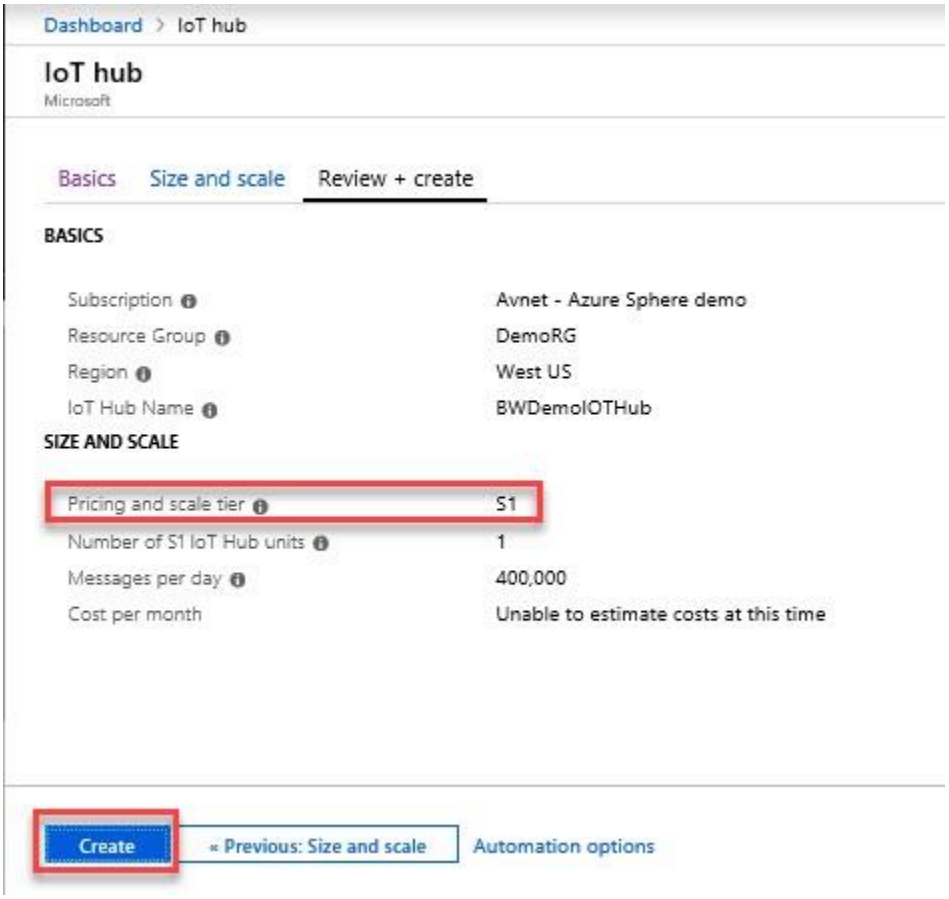


- In the IoT hub form fill in each entry
    - **Subscription:** Whatever your subscription is, it may be a “Free” or a “Pay-as-you-Go” subscription.
    - **Resource Group:** Click on the “Create new” link under the entry box and give your new Resource Group a name. For example “DemoRG”.
    - **Region:** Select the region closest to your physical location.
    - **IoT Hub Name:** Select a name for your IoT Hub. Note that the IoT Hub name must be unique across all of Azure. Your entry will be validated and if the name you used is not available, the form will display an error. Azure will generate a FQDN for your IoT Hub, so it must be unique.
- Click on the “Review + create” button

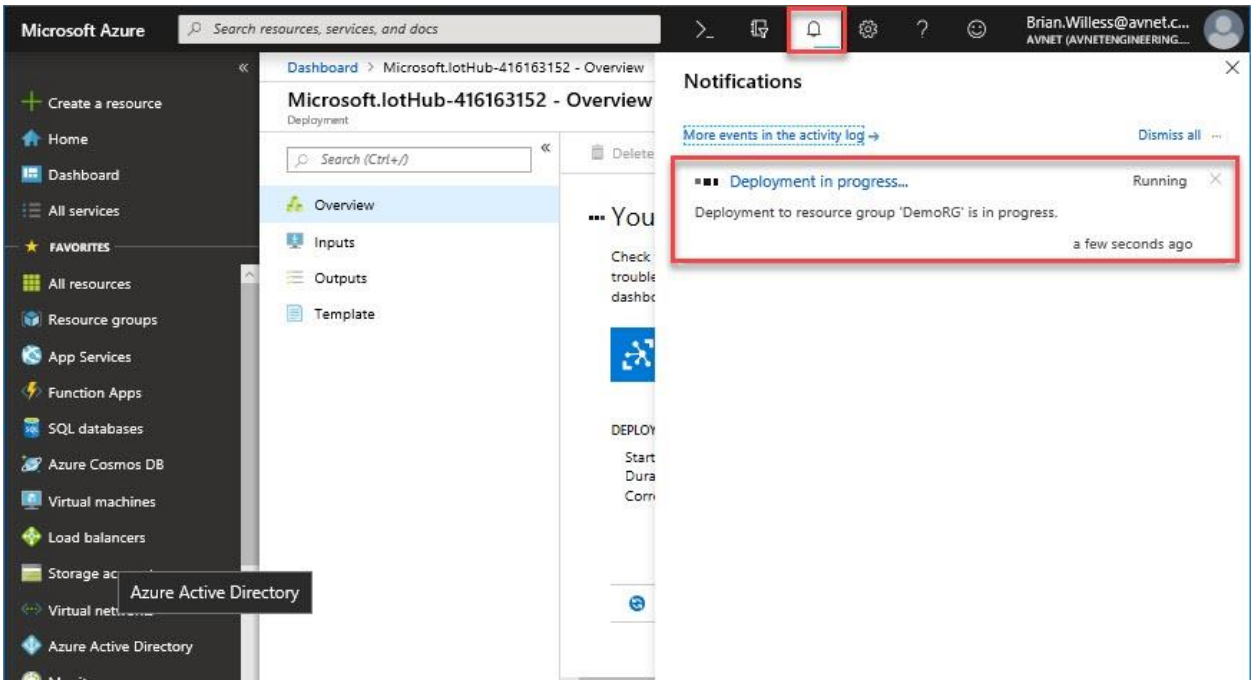


- Review the properties for your new IoT Hub. The “Pricing and scale tier” should be set to S1, the default. This tier will accommodate 400,000 data messages/day to/from your Azure Sphere device. After you’re happy with the properties, click on the “Create” button at the bottom of the form

- Note: We select the S1 tier so that we can explore some additional Azure features that are not supported by lower tiers.



- Azure will start to work on deploying your IoT Hub. This can take a few minutes. You can monitor the progress/status of your deployment by clicking on the bell icon in the header.



Create a Device Provisioning Service (DPS)

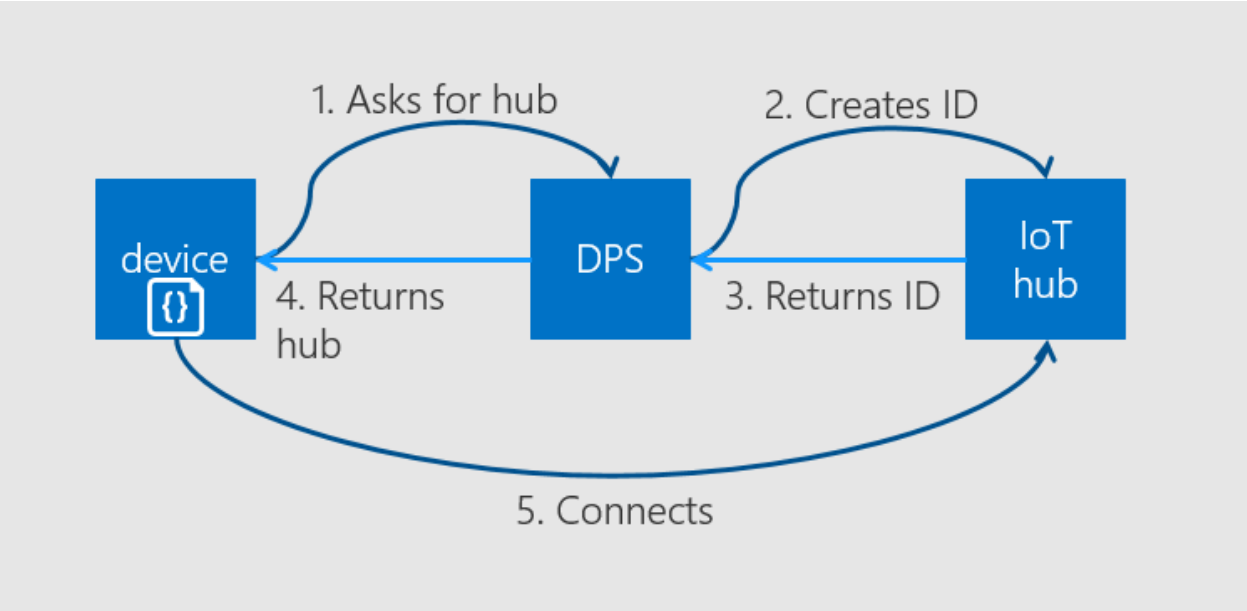
Now that we have an IoT Hub, we need to provision or add devices to our Hub. Devices must be provisioned to an IoT Hub; the IoT Hub will reject any connection attempts from un-provisioned devices.

When we're talking about IoT this usually implies that we'll have a very large number of devices. This allows us to collect large amounts of data that can be used to gain insights about our system so that we can make smart, data-driven, business decisions.

But how do we provision all these devices? One way is to manually add each device to an IoT Hub and use an IoT Hub connection string specific to that device. This works for a single device, but what about when you have 10, 100, or 100,000 devices. This approach would require some poor engineer to manually add 100 different devices to the IoT Hub, and would require 100 different application builds, each with its own specific connection string. It's easy to see this this is not a scalable approach.

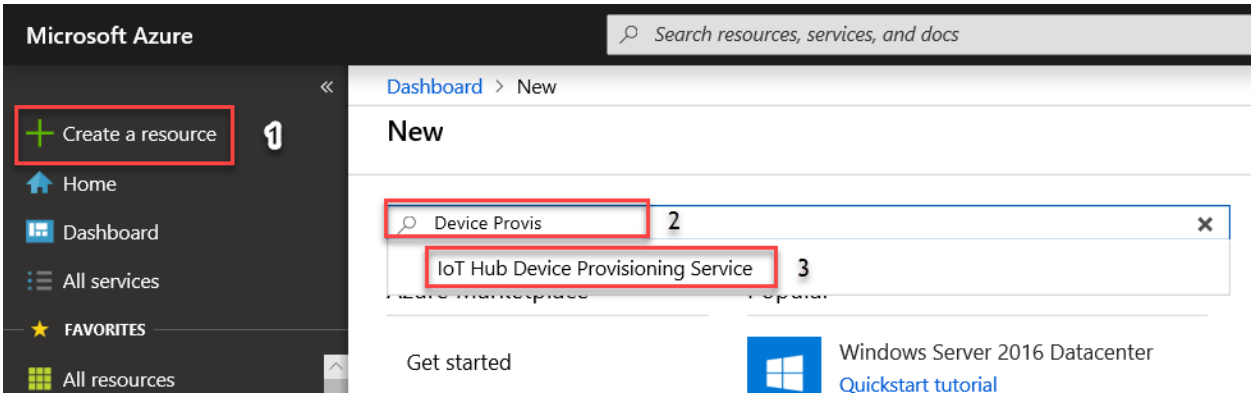
To make connecting 100, 100,000, or 1,000,000 devices to Azure easy, Microsoft provides a Device Provisioning Service (DPS). DPS allows large scale deployments without any manual provisioning steps. When you have 10 or 1,000,000 devices this is a great thing!

From the developer's point of view, I can create a single application build that can be deployed on all my devices. When my devices first connect to the Internet they will contact the global DPS server that will use some specific information in my application to provision my devices onto my IoT Hub(s). The diagram below illustrates how DPS works. After step #5 the device is provisioned and connected to the IoT Hub!

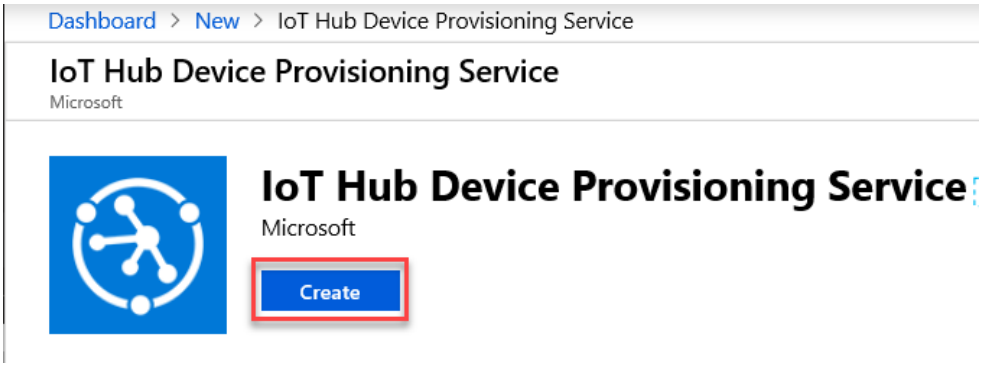


Create a new DPS

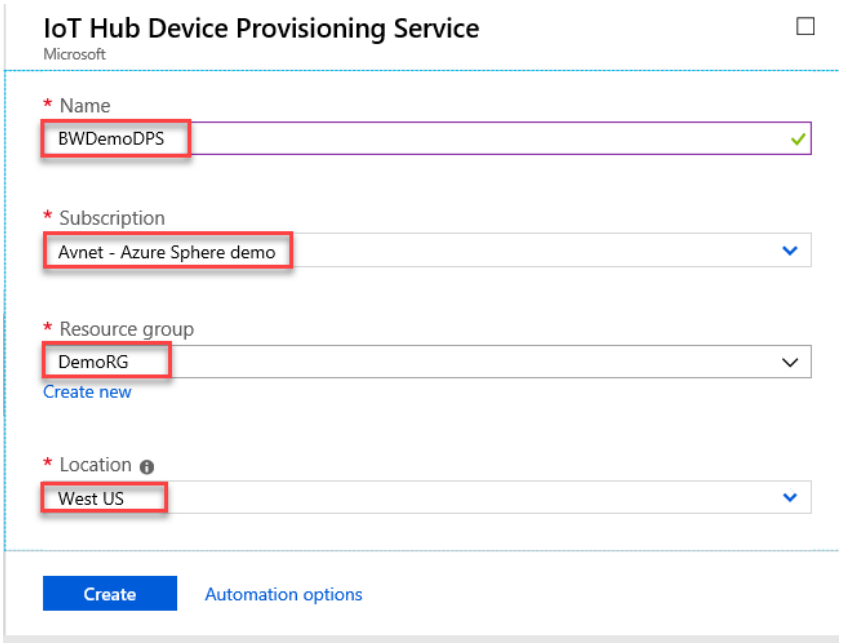
- In your Azure portal click on the “Create a resource” button
- Search the marketplace for “Device Provisioning Service”
- Select IoT Hub Device Provisioning Service



- Click Create



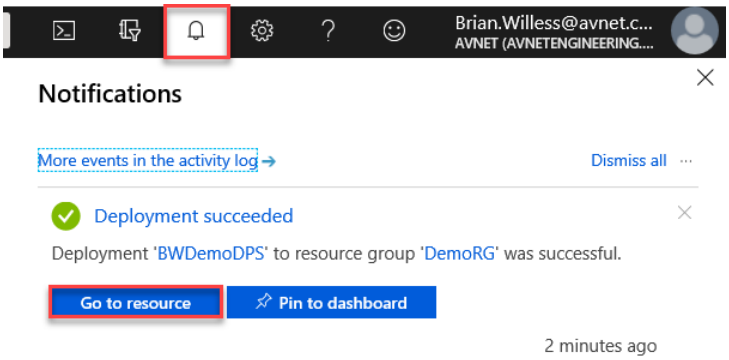
- In the **IoT Hub Device Provisioning Service** form fill in each entry
  - **Name:** Select a name for your DPS.
  - **Subscription:** Select your subscription, it may be a “Free” or a “Pay-as-you-Go” subscription.
  - **Resource Group:** Select the same resource group that you created when you created your IoT Hub
  - **Region:** Select the region closest to your physical location.
- Click on the “Create” button



- Wait for your DPS to be deployed

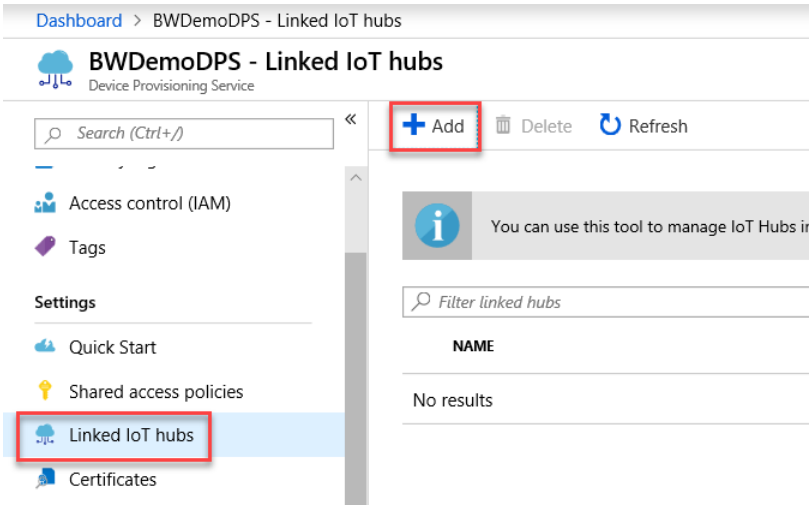
Next we need to configure our DPS. Find your new DPS resource. One way to find your new resource is from the notification icon at the top of the Azure Portal (the bell), click on this icon to see the resources you recently created.

- Click on the “Go to resource” link



Next we need to associate our DPS with our IoT Hub. This way when a device connects to the DPS, the DPS will know which IoT Hub to provision the device.

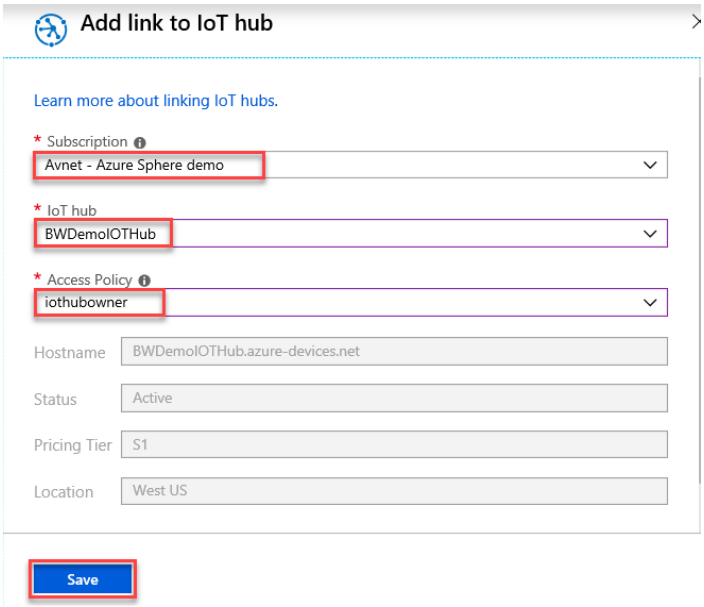
- From the DPS resource find the “Linked IoT hubs” blade (In Azure these configuration categories are called blades)
- Click on the “+ Add” link



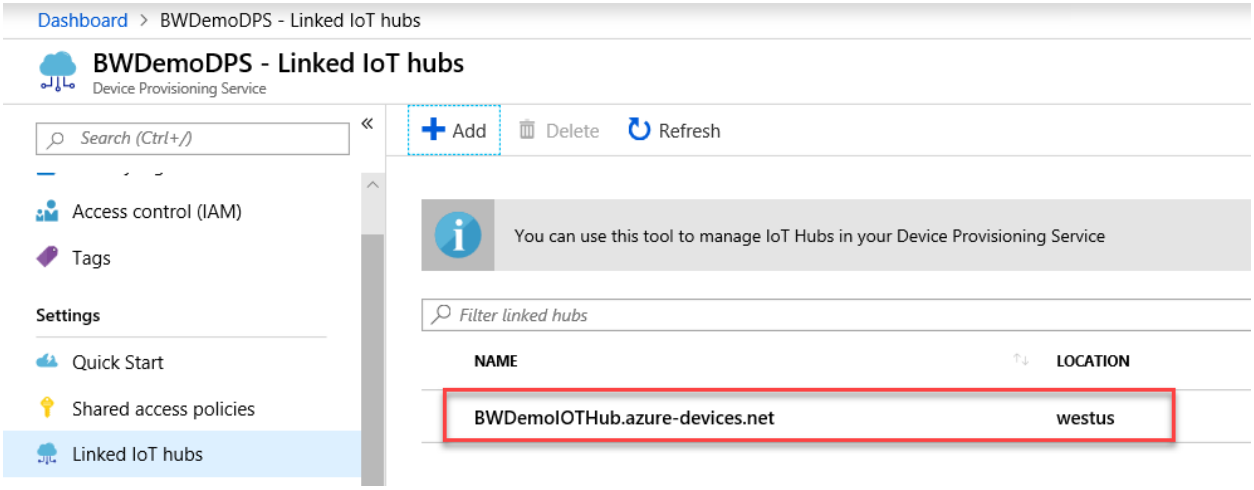
In the **Add link to IoT hub** form fill in each entry

- **Subscription:** Select your subscription, it may be a “Free” or a “Pay-as-you-Go” subscription.
- **IoT hub:** Select the IoT Hub we created earlier
- **Access Policy:** Select iothubowner from the drop down list

- Click on the “Save” button



- You will see your IoT Hub listed by its FQDN
- If you don't see your IoT Hub, click on the refresh button at the top of the form



Prove to DPS that we own the tenant

The next thing we need to do is to prove to the DPS that we own the Azure Sphere tenant that our devices are claimed to. This is another slick security feature of the Azure Sphere system. After everything is setup, only devices in your tenant will be able to use your DPS to connect to your IoT Hub. That means that if someone where to get hold of your application and side load it onto their Azure Sphere device, that device would not be allowed to connect to your DPS or your IoT Hub. DPS will reject the connection based on an incorrect tenant certificate. Only devices claimed to your Azure Sphere Tenant will be allowed to connect to your DPS and IoT Hub because they will have the correct tenant certificate.

The steps to setup this trust relationship are listed below. This is a one-time setup task. Once this is all setup and configured, you'll only have to do this again if you setup a new DPS with a new IoT Hub.

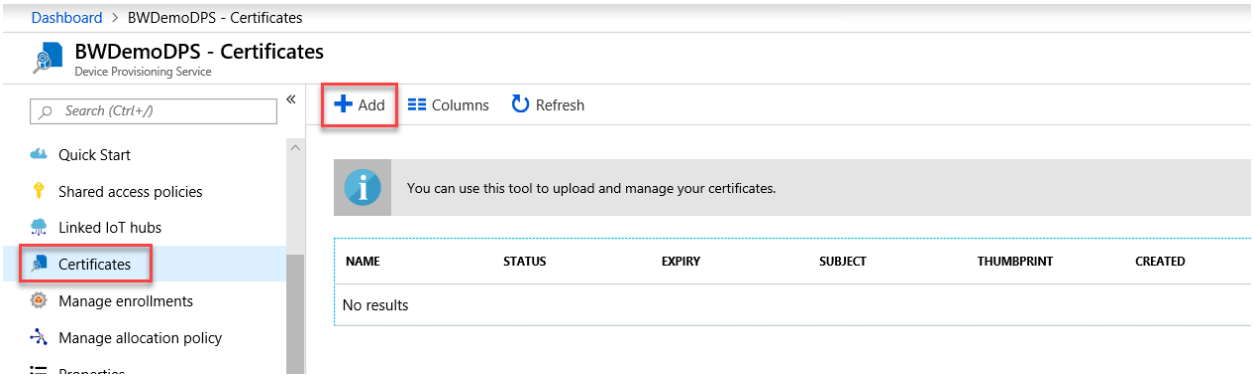
1. Download the authentication CA certificate for your Azure Sphere tenant from the Azure Sphere Security Service.
2. Upload the CA certificate to DPS to tell it that you own all devices whose certificates are signed by this CA. In return, the DPS presents a challenge code.
3. Generate and download a validation certificate from the Azure Sphere Security Service, which signs the challenge code. Upload the validation certificate to prove to DPS that you own the CA.
4. Create a device enrollment group, which will enroll any newly claimed Azure Sphere device whose certificate is signed by the validated tenant CA.

Download the authentication CA certificate for your Azure Sphere tenant from the Azure Sphere Security Service.

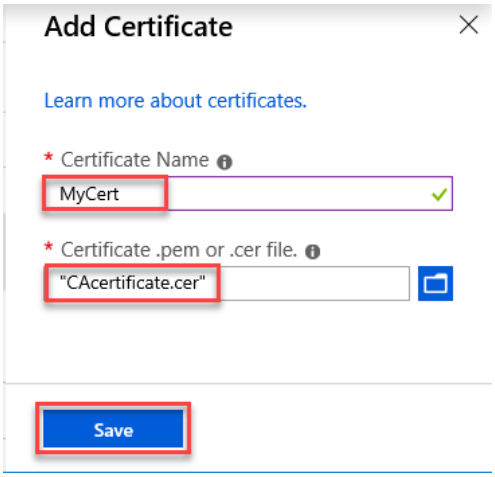
- Go back to your "Azure Sphere Developer Command Prompt Preview" application
    - **Start → Azure Sphere → Azure Sphere Developer Command Prompt Preview**
- Make sure you're logged into your tenant:
    - **azsphere login**
- Copy and paste in the command:
    - **azsphere tenant download-CA-certificate --output CAcertificate.cer**
    - Note the output file must have the .cer extension

Upload the CA certificate to DPS to tell it that you own all devices whose certificates are signed by this CA. In return, the DPS presents a challenge code.

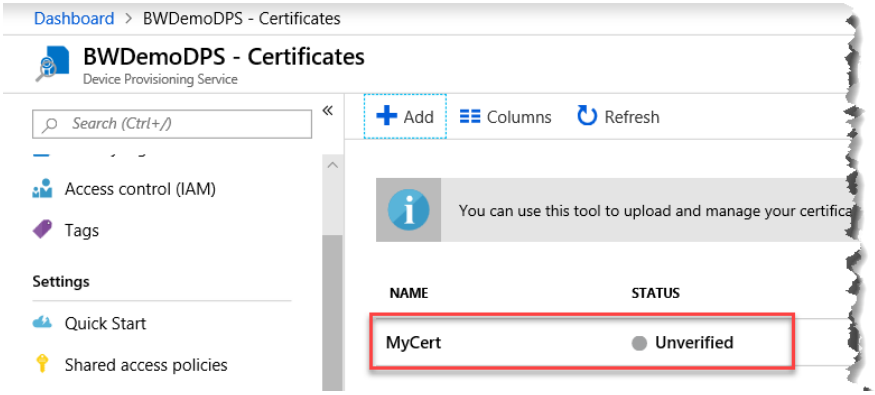
- Back in your Azure Portal navigate to the DPS that you created
- Open the Certificates blade from the list
- Click on the “+ Add” link at the top of the form



- In the **Add Certificate** form fill in each entry
  - **Certificate Name:** Create a name for your certificate
  - **Certificate *:** Browse to the certificate file we just downloaded
- Click on the “Save” button

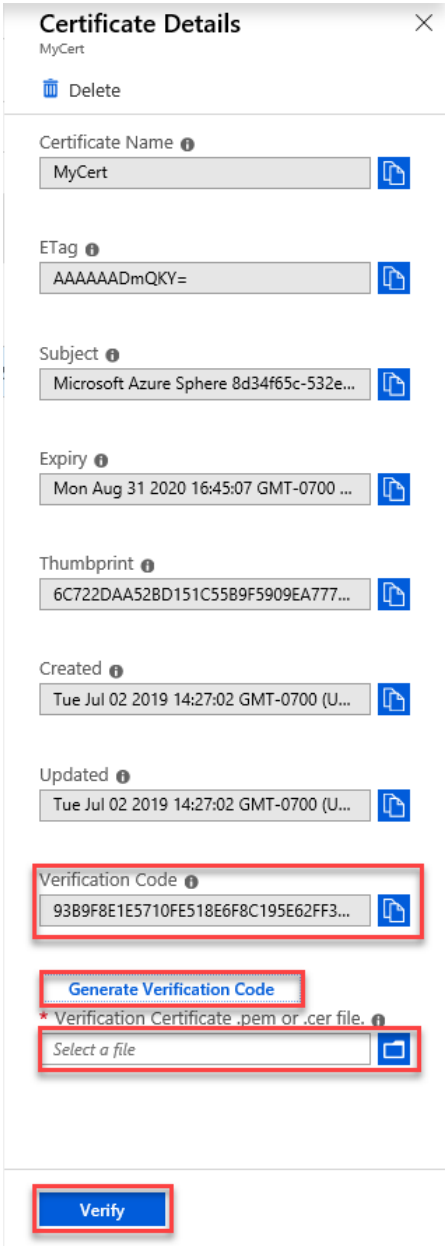


Your certificate will be added to the list and its state will be “Unverified.” Technically, we could have gained access to someone’s tenant ca. Just because we have this public certificate it does not prove that we own the tenant. Next we need to verify the certificate.



Generate and download a validation certificate from the Azure Sphere Security Service, which signs the challenge code. Upload the validation certificate to prove to DPS that you own the CA.

- Click on your certificate in the list, this brings up the Certificates Details form

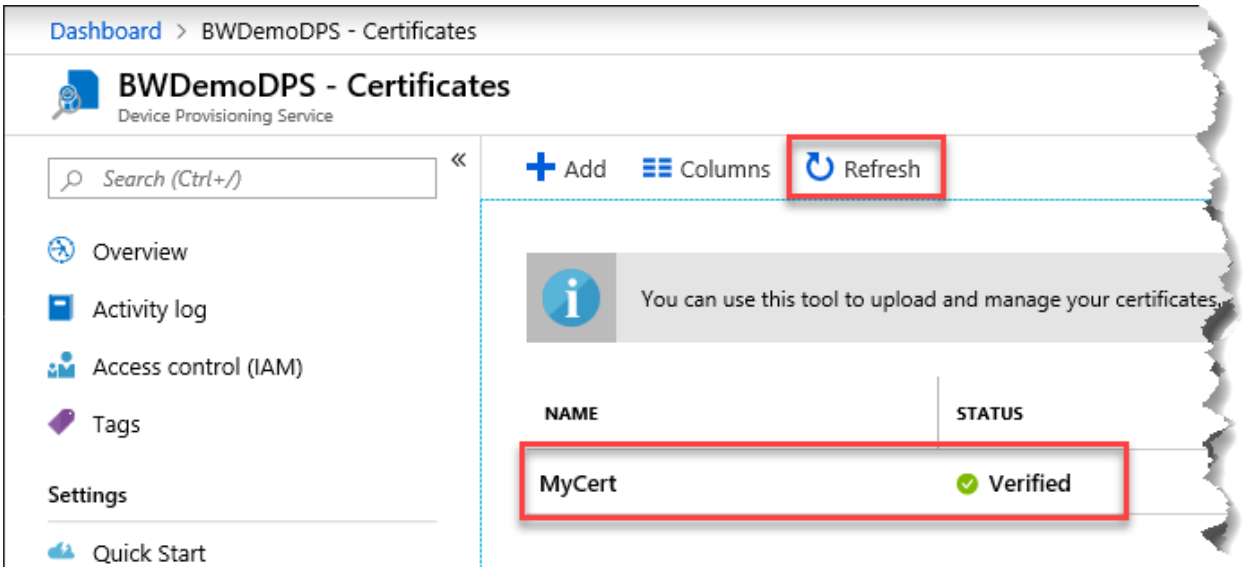


- Return to the Azure Sphere Developer Command Prompt
- Copy and paste the following command into the application but do not execute the command yet, as a verification code still needs to be added:
  azsphere tenant download-validation-certificate –output validation.cer --verificationcode <code>
- Click on the "**Generate Verification Code**" link towards bottom of the form, a Verification code is generated and displayed in the "Verification Code" box.
- Copy the verification code, there's a copy link to right of the box
- Back in the command window, replace the <code> text with your verification code and execute the command

The Azure Sphere Security Service signs the validation certificate with the verification code to prove that you own the CA and downloads a Verification certificate.

- Back in your Azure portal, upload the new Verification Certificate by browsing to the file.
- Click the "**Verify**" link at the bottom of the form

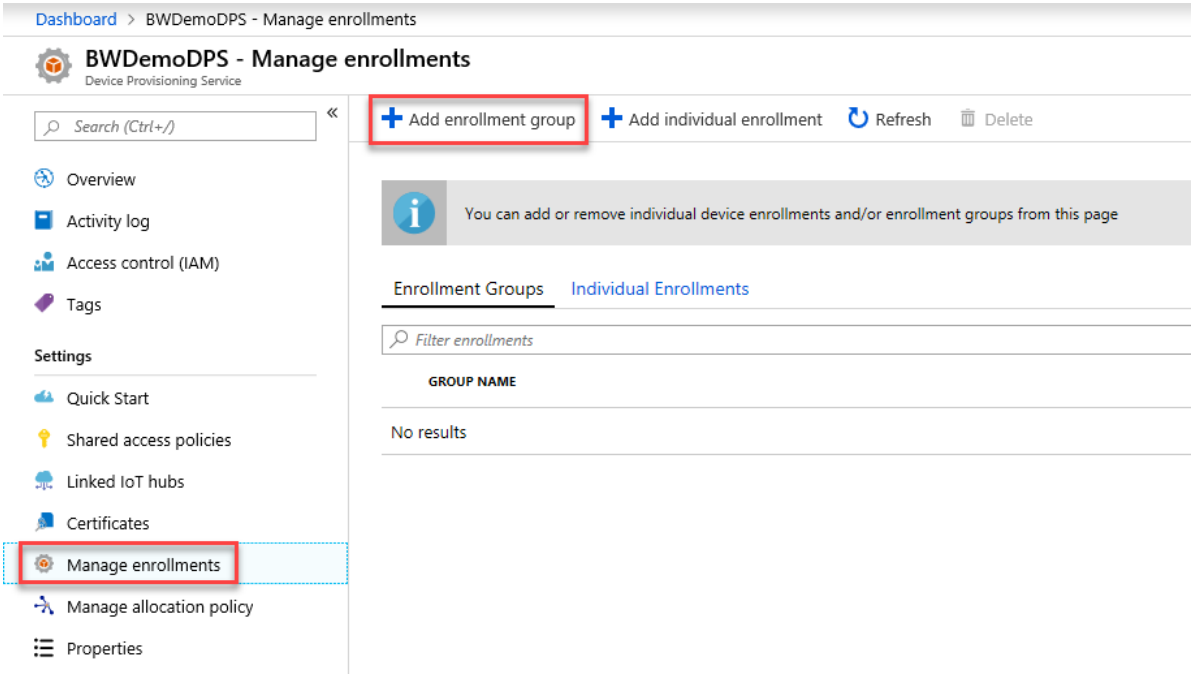
You should see that your certificate is now shown as Verified. If not, click on the “Refresh” link at the top of the form.



Create a device enrollment group, which will enroll any newly claimed Azure Sphere device whose certificate is signed by the validated tenant CA.

We're almost done!

- Back in your Azure Portal, navigate to your DPS resource
- Find and click on the "Manage enrollments" blade
- Click on the "Add enrollment group" link at the top of the form



- In the **Add Enrollment Group** form fill in each entry
  - **Group Name:** Create a name for your enrollment group
  - **Primary Certificate:** Select the certificate that we just uploaded and validated
  - Leave all other fields at the default selections

- Update **Initial Device Twin State** by adding three userLed entries as shown below, ie.
  "userLedRed": false,
  "userLedGreen": false,
  "userLedBlue": true

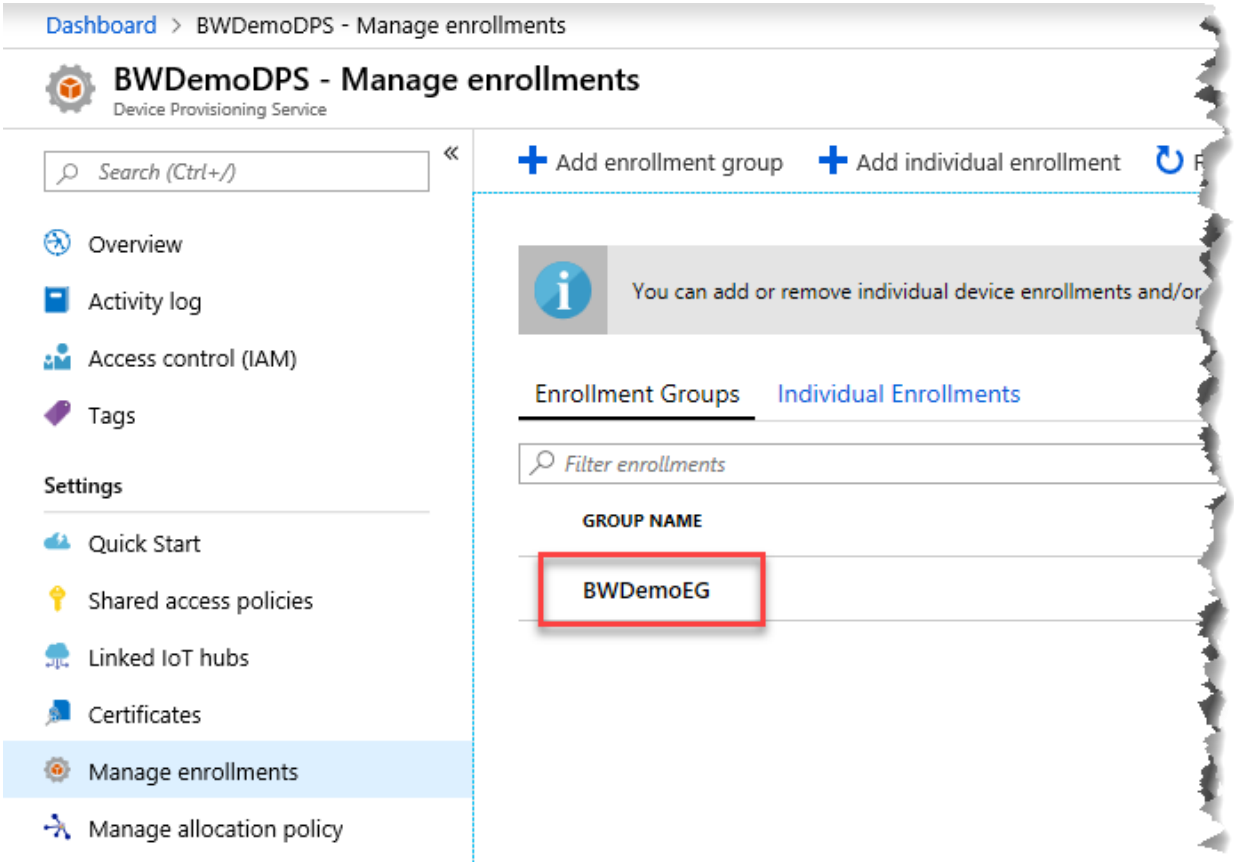
Initial Device Twin State

```
{
  "tags": {},
  "properties": {
    "desired": {
      "userLedRed": false,
      "userLedGreen": false,
      "userLedBlue": true
    }
  }
}
```

- Click on the "**Save**" button at the top of the form

The Initial Device Twin State is a cool feature and the reason why we selected the S1 IoT Hub tier. When devices are provisioned by this DPS, they will get these initial device twin properties. By setting one or more of the userLed properties to true we'll be able to see when our device connects to our IoT Hub. When the device connects to the IoT Hub it will receive this update and turn on any LED set to true!

You should see your new Enrollment Group appear in the list

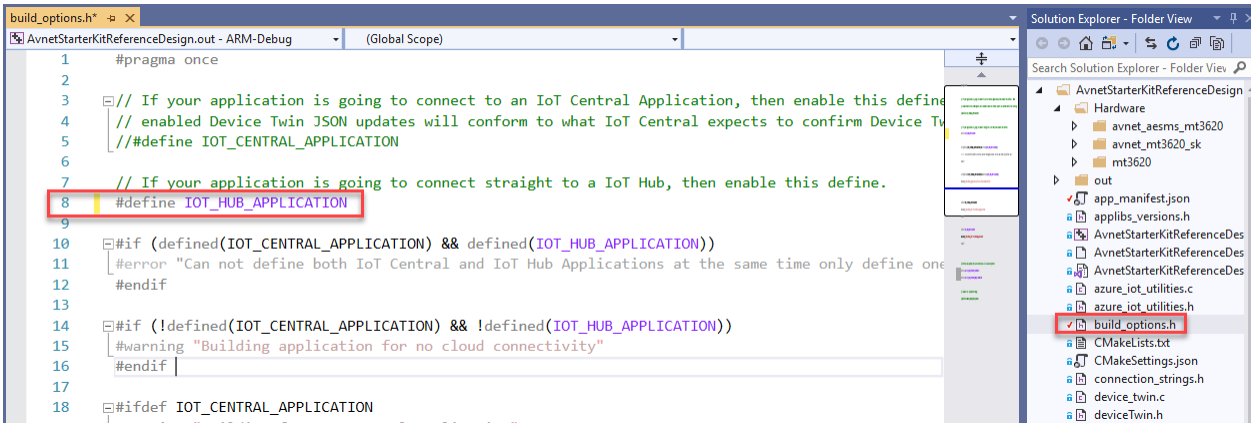


Configure the example application for the IoT Hub configuration

Now that we have the Azure side ready, let's go back to our application code and configure the application to connect to use our newly created DPS and IoT Hub.

Modify the Azure Sphere source code

- Launch the Visual Studio application and open the "**AzureSphereHacksterTTC**" project. Visual Studio keeps a list of recent projects, your project should be visible at top of that list.
- Open the **build_options.h** file
- On line #8 remove the "//"s to enable the IOT_HUB_APPLICATION build option
- Confirm line #5 is commented out, the press Ctl+S to save this file



Next we need to add DPS and IoT Hub details to our project.

Updating the app_manifest.json file

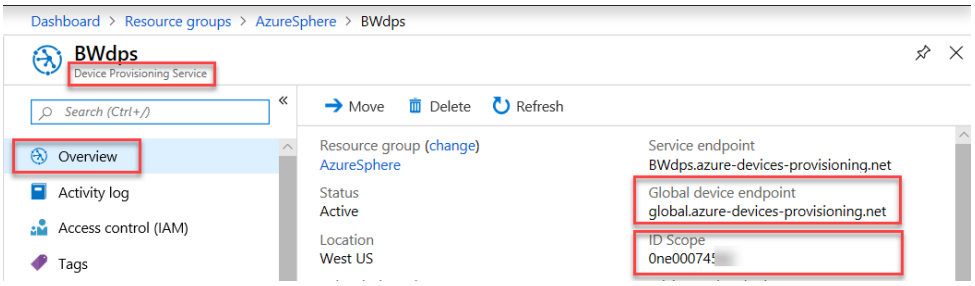
Open the **app_manifest.json** file this application. There are four items we need to include in order for our application to use DPS and connect to our IoT Hub:

- **DPS Scope ID**: This is used when our application connects to the global DPS, this allows the global DPS to route our request to our DPS.
- **AllowedConnections (2)**: The "AllowedConnections" parameter is discussed in an earlier section For our application to talk with any server, that server's FQDN IP address must be listed here. We will locate and add the following detail:
  - the DPS global device endpoint FQDN and
  - the IoT Hub's FQDN.
- **DeviceAuthentication:** This is the GUID for my Azure Sphere Tenant. This is included so that this application will only work on devices claimed to our Azure Sphere Tenant. So if someone was able to get our application and side load it onto an Azure Sphere device, that device would not be able to connect to our Azure services, unless it's in the same Azure Sphere Tenant.

DPS Scope ID and DPS global device endpoint FQDN

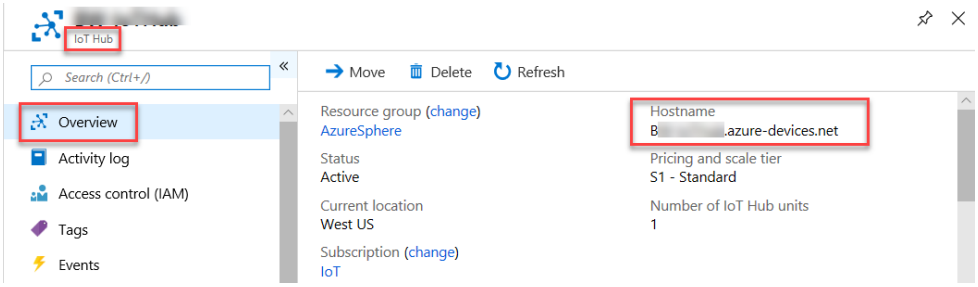
Open your DPS resource in Azure and on the "**Overview**" blade locate:

- **ID Scope**: Copy + paste the DPS Scope ID into the "**CmdArgs**" line in the **app_manifest.json** file.
- **Global device endpoint**: Copy + paste this FQDN into the "**AllowedConnections**" line of the **app_manifest.json** file



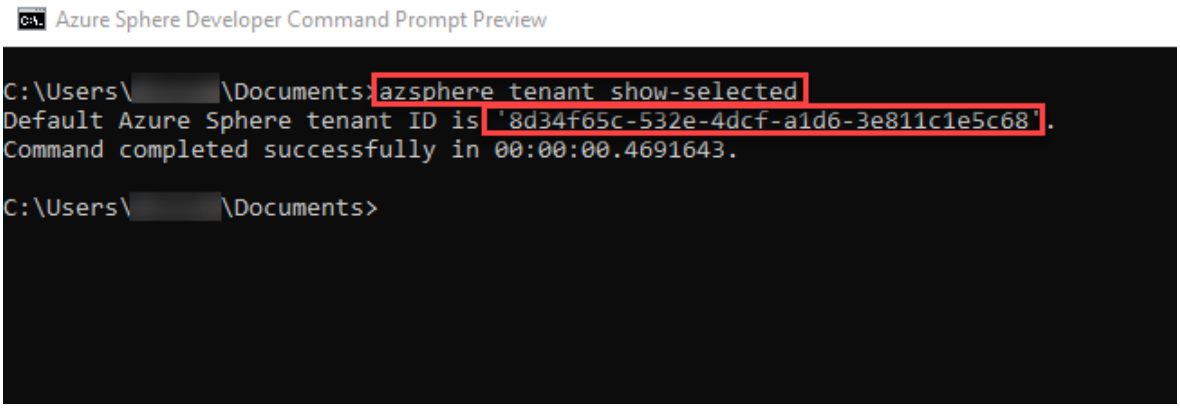
IoT Hub Hostname

From Azure also locate and copy the IoT Hub Hostname. Navigate to your IoT Hub resource and in the "**Overview**" blade find and copy your IoT Hub hostname. Add the hostname to your "**AllowedConnections**" line in your **app_manifest.json** file.

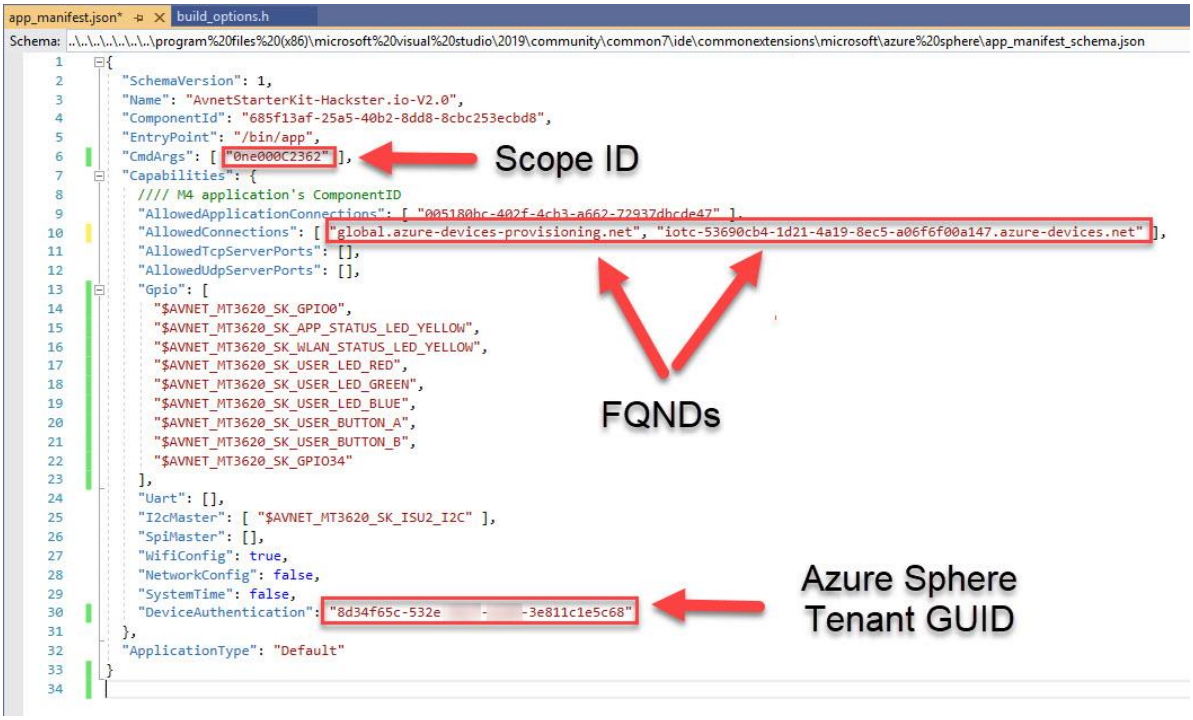


DeviceAuthentication (Azure Sphere Tenant ID)

The "**DeviceAuthentication**" GUID is really just your Azure Sphere Tenant GUID.
Get this by using the command line interface and the command **azsphere tenant show-selected** (You may be asked to login to your tenant to do this). Copy the reported GUID and paste this in the "**DeviceAuthentication**" line of your **app_manifest.json** file.



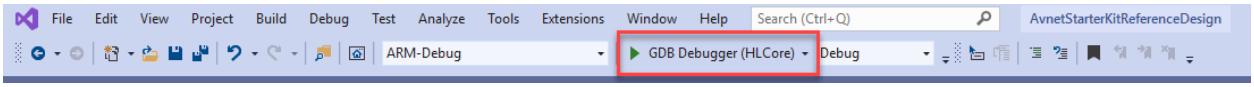
Below is a capture of my app_manifest.json file with the changes identified.



Build and Run the Application

Now let's build and run our application!

- Make sure your device is connected to your PC
- Ensure your device has a Wi-Fi connection
    - **azsphere device wifi show-status**
- Click on the "**GDB Debugger (HLCore)**" button at top of the screen in Visual Studio. Your application will build, link, side-load, and run. If this selection is not visible, use the pull down control to select it from the list.



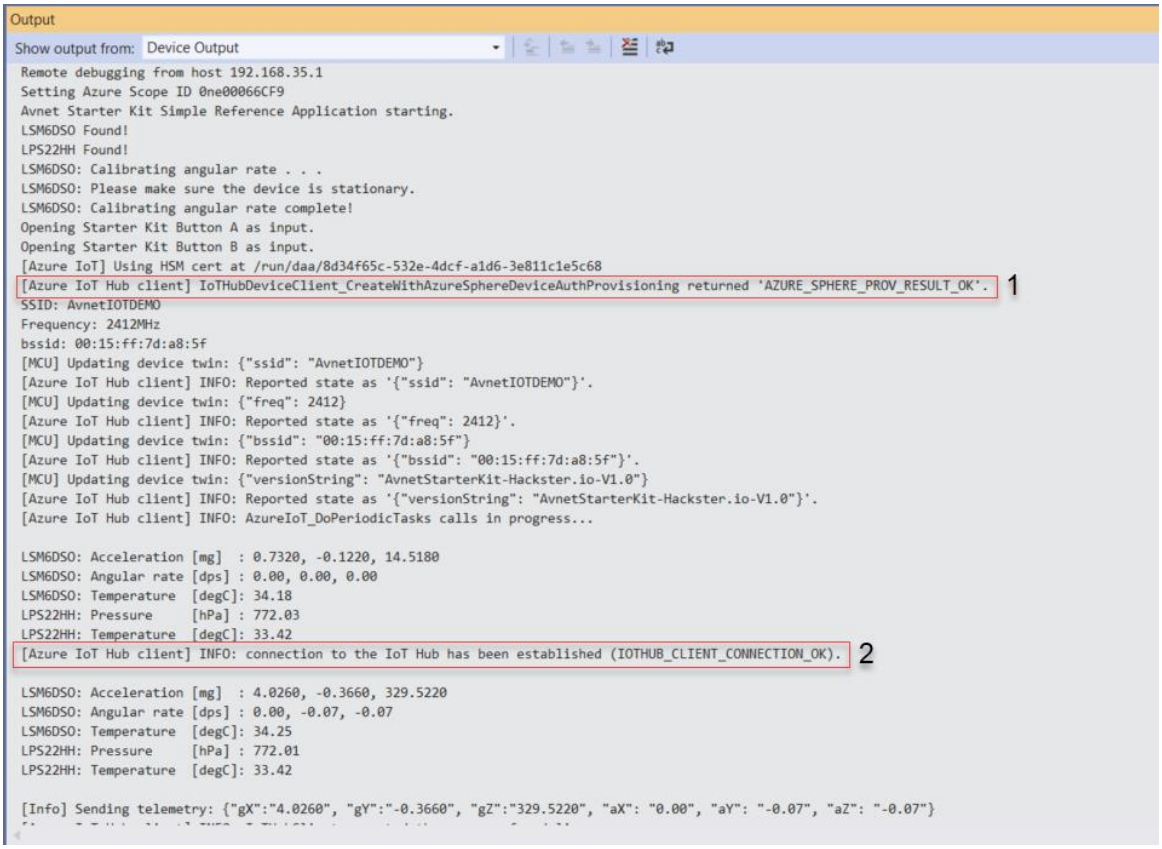
Your output should look similar to the screenshot below. The two [Azure IoT Hub Client] messages highlighted in the screenshot should be visible at start of the debug session.
If these not seen, do a **Build → Clean All**, then again click on **GDB Debugger (HLCore)**

[Azure IoT Hub Client] …'AZURE_SPHERE_PROV_RESULT_OK': Confirms that our device connected to our DPS and that the DPS successfully provisioned our device to our IoT Hub

[Azure IoT Hub Client] … IOTHUB_CLIENT_CONNECTION_OK: Confirms that our application has successfully connected to our IoT Hub

If you have errors . . .

- Confirm your device is connected to a Wi-Fi access point or hot spot that has internet connectivity
- Review the Azure configuration steps to make sure you did not miss a step



Code assignment

The assignment is to add and additional telemetry item to report the pressure reading from the LPS22HH sensor. The key for the {“key”: value} pair should be called “**pressure**.”

Hints:

- The pressure is read into the variable **`pressure_hPa`** in the **i2c.c** file around line #172
- You could modify the existing code that sends the telemetry data up to Azure
    - **i2c.c** file around line #200
- You could send up the pressure telemetry as a standalone {“key”: value} pair
- Press **Ctl+S** to save i2C.c when finished editing, the relaunch the debugger

What does success look like?

After rebuild with the new telemetry item added, you’ll see pressure transmitted to the Azure IoT Hub. Confirm that you see pressure data in the **Sending Telemetry** message in Azure Sphere debug output.

After setting up Time Series Insights (the next section) you will also be able to see this pressure data graphed in your TSI Environment.

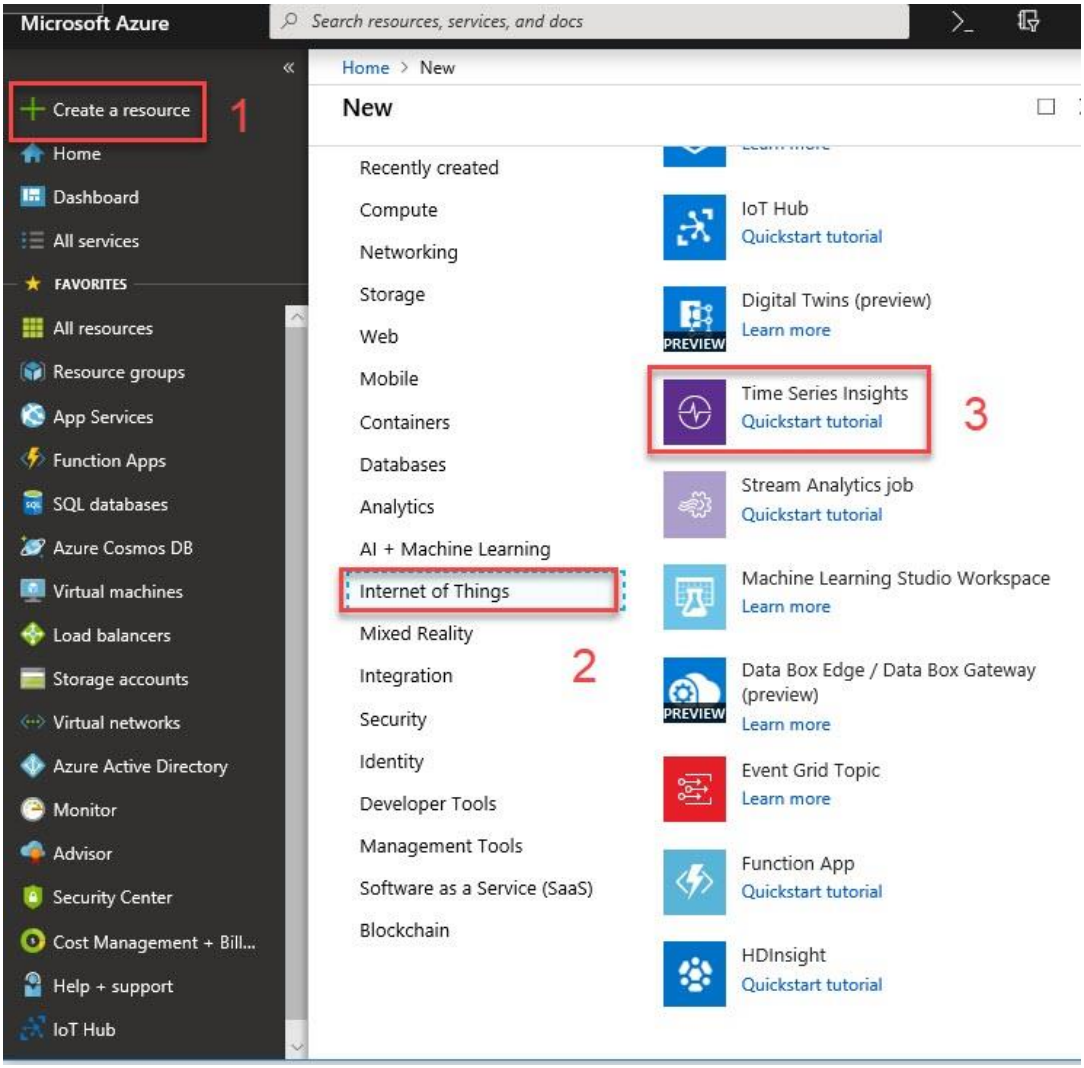
Create a Time Series Insights (TSI) Environment

Our application is streaming lots of telemetry data. What can we do in Azure to view this data? One answer is to setup a Time Series Insights (TSI) environment. TSI documentation is here. There are other services available in Azure to visualize your data, TSI is easy to setup and is a powerful tool.

> "Azure Time Series Insights provides powerful data exploration and telemetry tools to help you refine operational analysis."

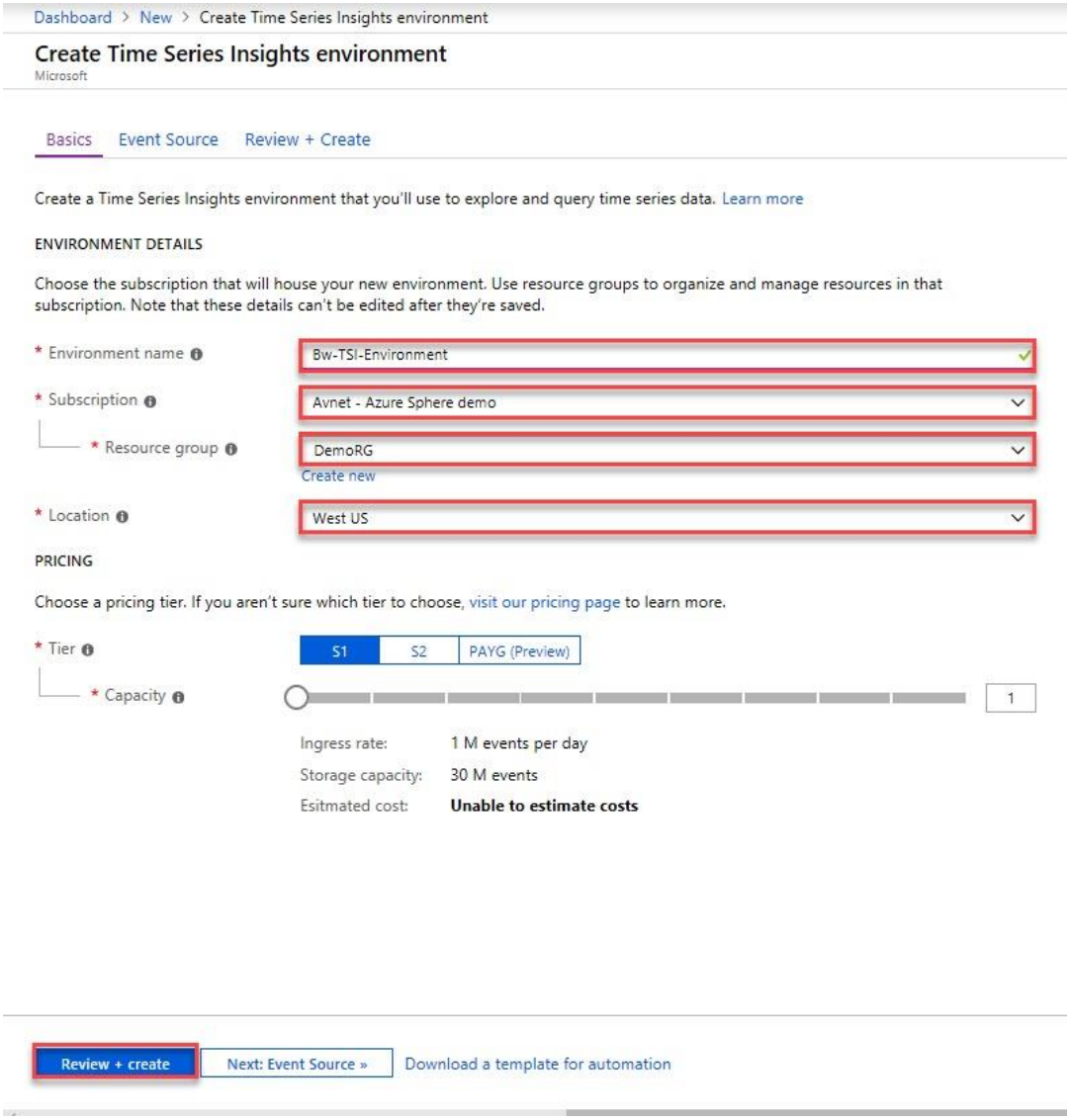
One note about the TSI resource. This resource is one of the more expensive Azure resources. I recommend deleting this resource when you've completed this lab.

- Login to your Azure account: https://portal.azure.com
- Click on the **"+ Create a resource**" link (1)
- Click on the "**Internet of Things**" category (2)
- Click on the "**Time Series Insights**" link (3)

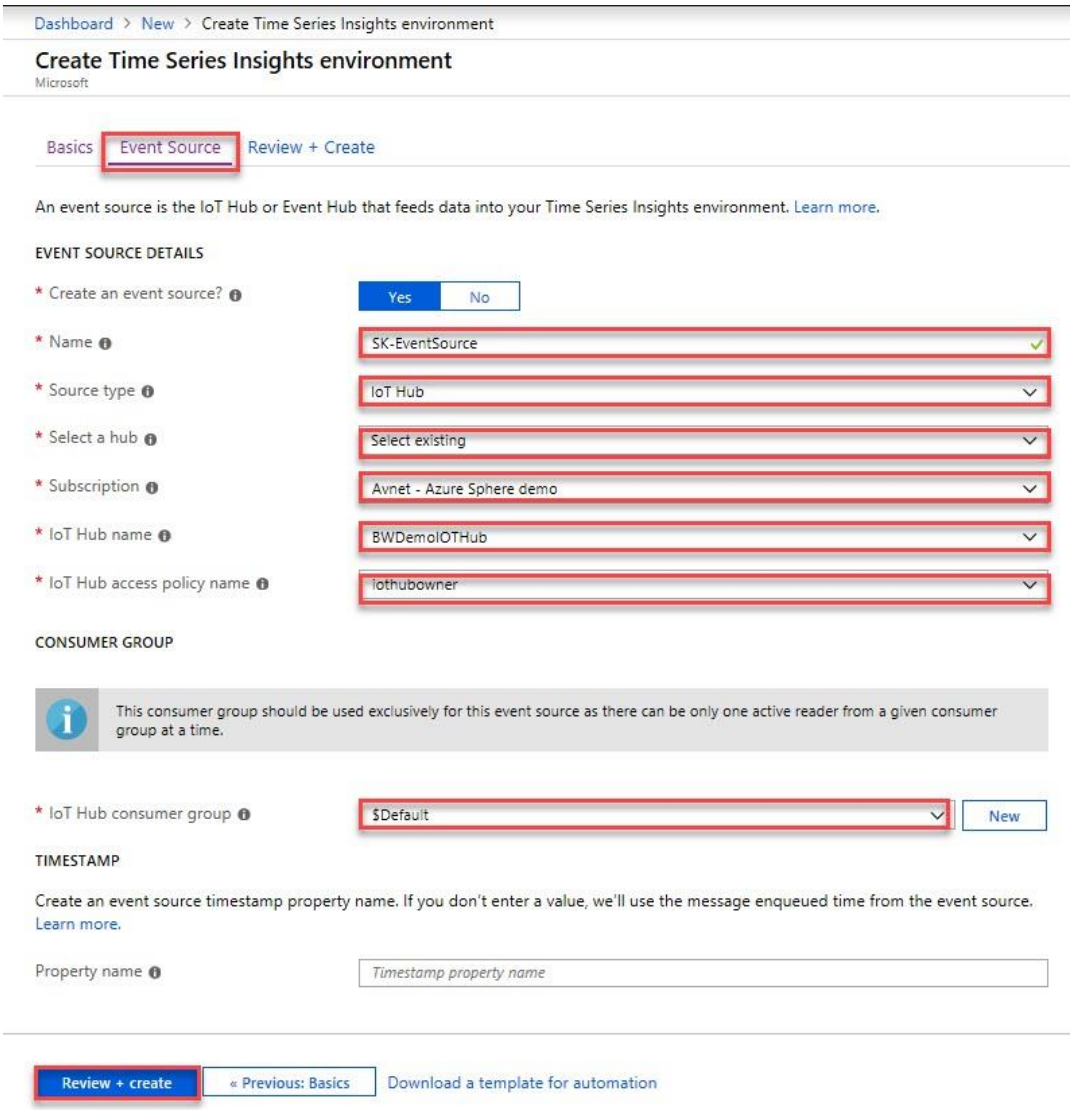


On the “Basics” Tab fill in the form

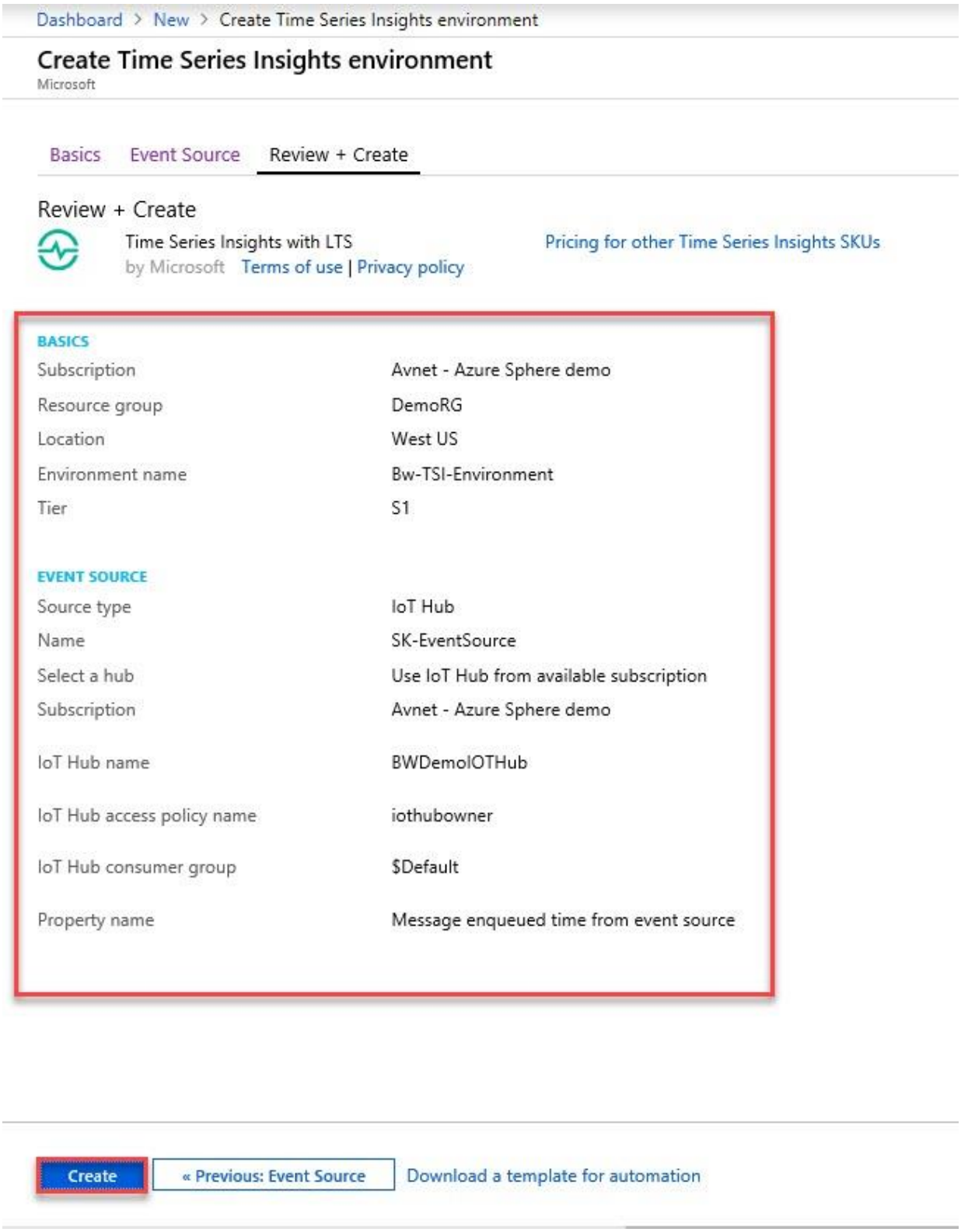
- **Environment Name**: Enter a name for your TSI environment
- **Subscription**: Select your Subscription
- **Resource group**: Select the same Resource Group you used for the IoT Hub
- **Location**: Select the Location closest to your physical location



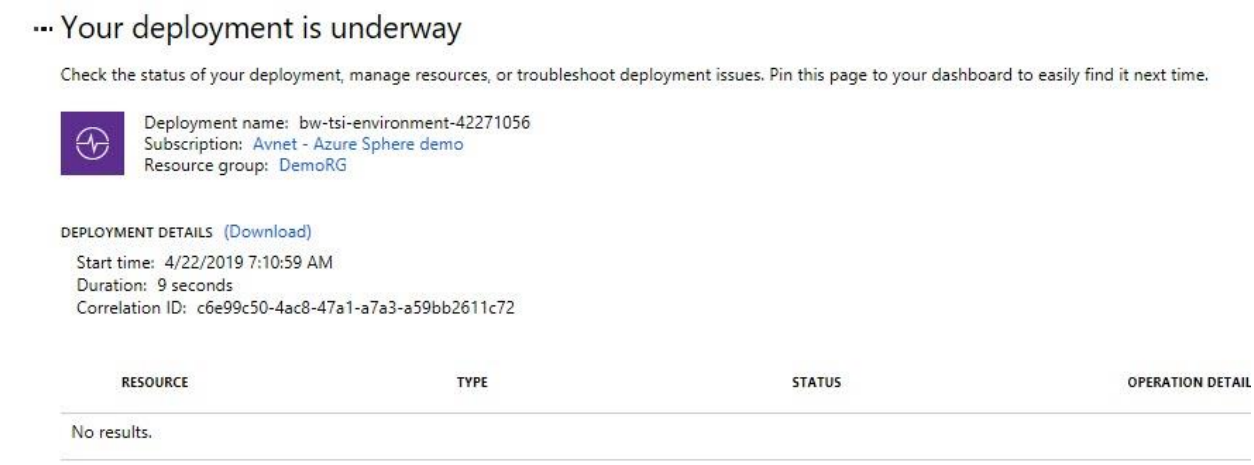
- Next click on the “Event Source” tab.
  - **Name**: Select a name for your event
  - **Source Type**: Select “IoT Hub”
  - **Select a Hub**: Select “Select existing”
  - **Subscription**: Select your subscription
  - **IoT Hub Name**: Select the name of your IoT Hub from the list
  - **IoT Hub access policy name**: Select “iothubowner”
  - **IoT Hub consumer group**: Select “$Default”
  - Click on “Review + create”



On the “Review + Create” screen, review your settings and if they look good, click on the “Create” button at the bottom of the form.

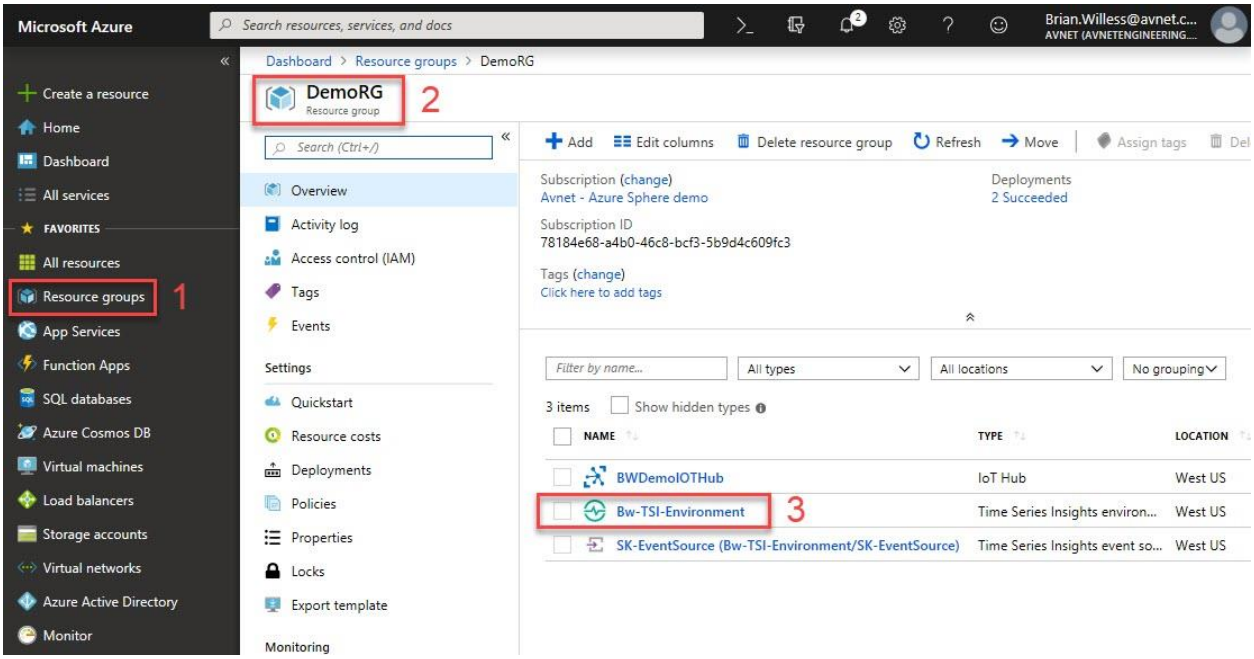


You'll see your TSI environment being provisioned:

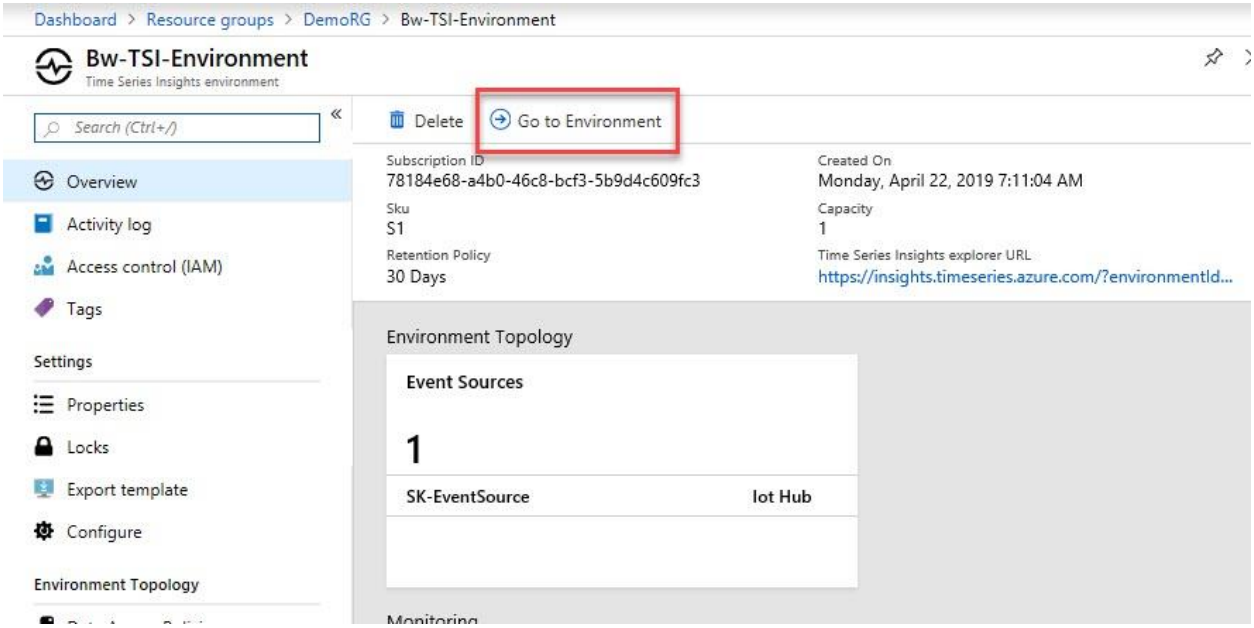


Once your deployment is complete (mine took 42 seconds), navigate to your new TSI resource.

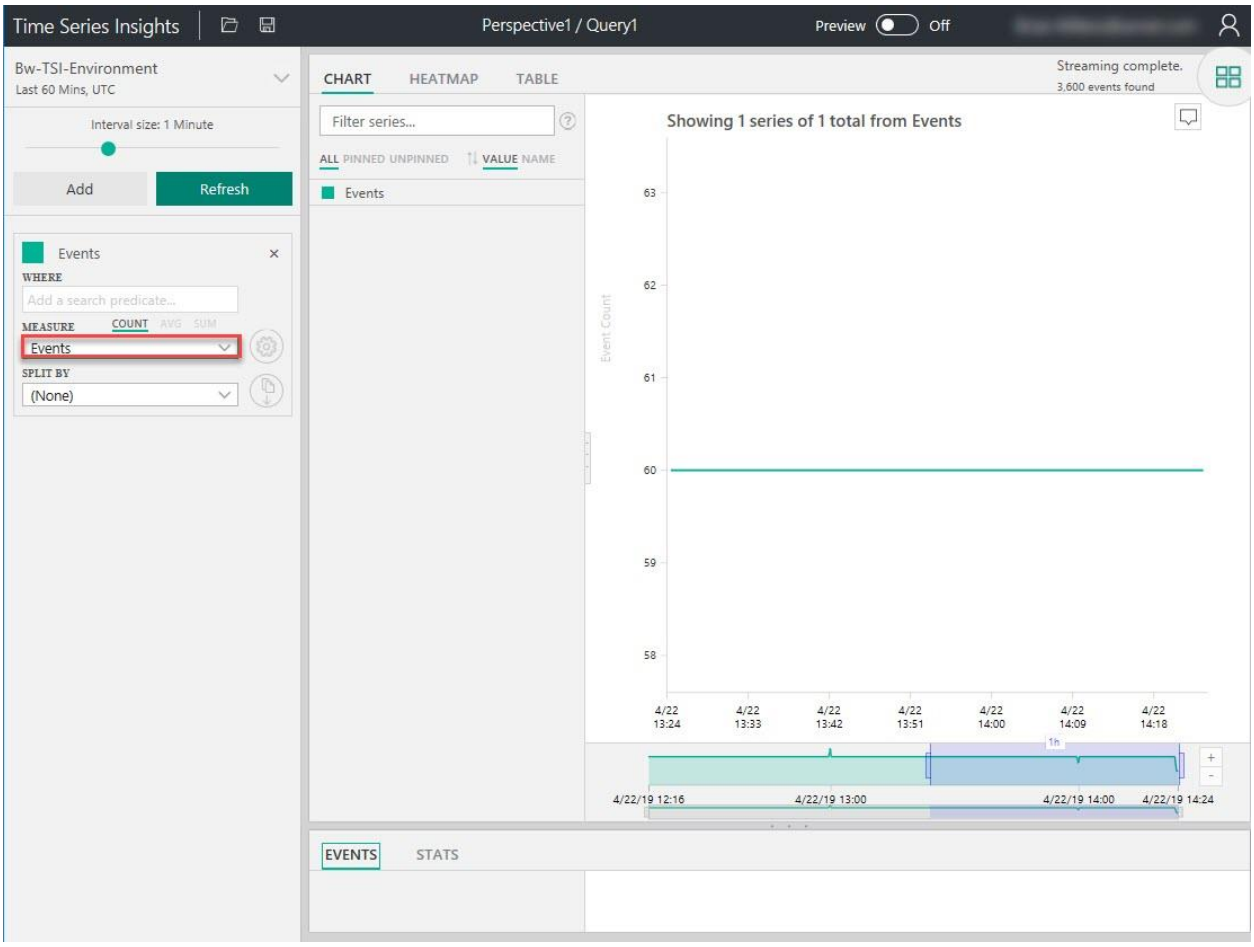
- From the left most column, click on "Resource Groups"
- Select your resource group
- Select your TSI Environment



- Click on the “Go to Environment” link



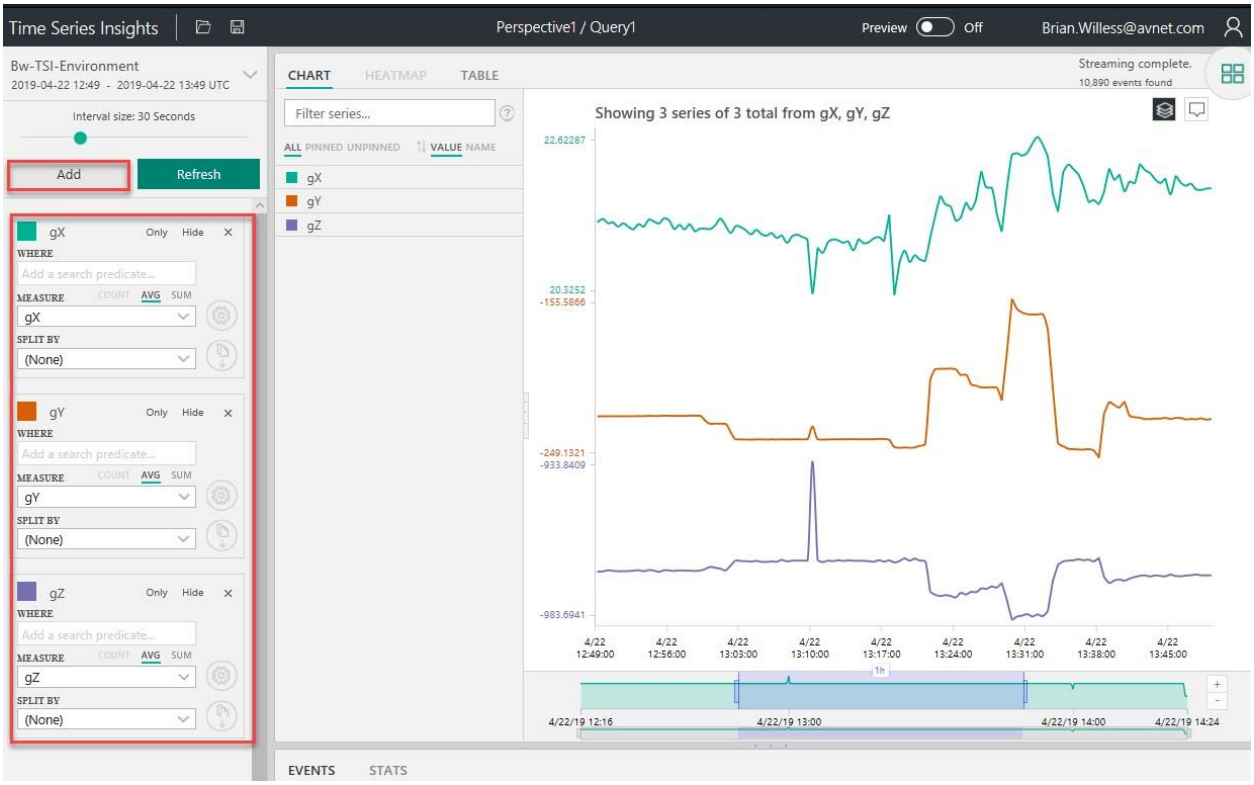
The Environment will open, but you won’t see any data



Select the “Events” pull down menu. You’ll see entries for all the different telemetry items the application sends up, select the “gX” entry. You can add additional measurements to the graph by clicking the “Add” button. Use the table below to understand what each telemetry item represents.

Display Name	Field Name	Units
X-GForce	gX	mg
Y-Gforce	gY	mg
Z-Gforce	gZ	mg
X-Angular Rate	aX	mdps
Y-Angular Rate	aY	mdps
Z-Angular Rate	aZ	mdps
Pressure	pressure	hPa

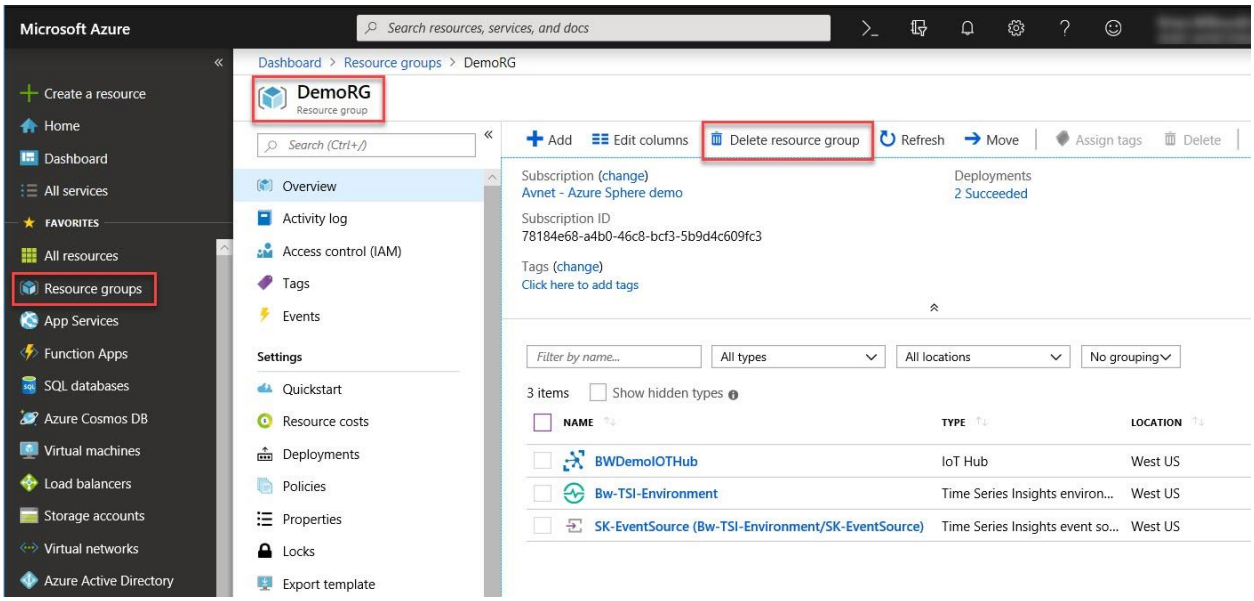
The graphic below shows the gX, gY, and gZ data all plotted. I was moving my Starter Kit around on the different axis’s to generate this data.



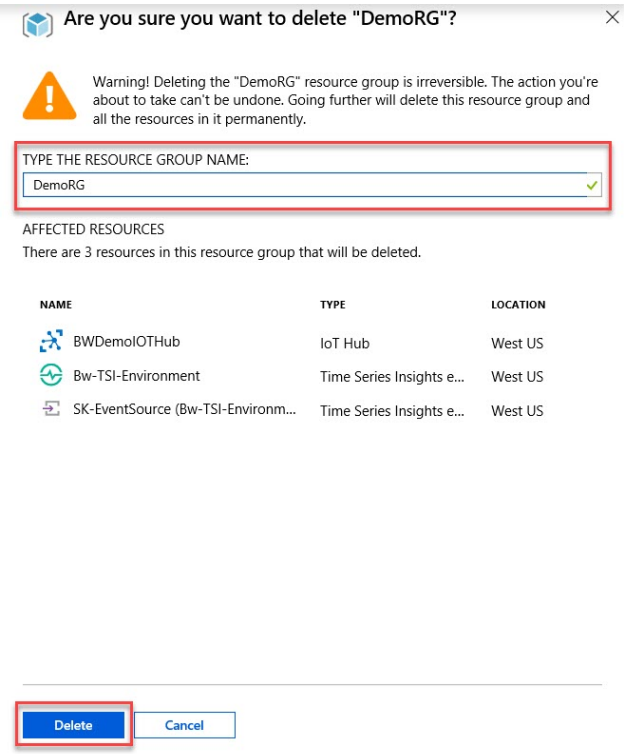
Clean up Azure Resources

When you're finished playing with the graph and the demo, you should delete the Azure resources. If you don't delete them, then you'll see monthly charges for the IoT Hub, and for the Time Series Insights resources. The quick and easy way to remove all these resources is to delete the Resource Group.

- Open the Azure Portal: https://portal.azure.com
- In the left most column, select "Resource Groups"
- Select your resource group
- At the top select "Delete resource group"



- Type the name of your resource group into the form
- Click on the "**Delete**" button at the bottom of the form.



Useful Links

- Microsoft Azure Sphere Documentation
- Microsoft Azure Sphere GitHub Examples
- Azure Sphere Developer Resources

Wrap Up

In this Lab we learned about Azure and how to create the Azure resources required to connect IoT devices.

- How to create an IoT Hub
- How to create a Device Provisioning Service (DPS)
- How to configure the example application for the IoT Hub configuration and the Connected Service utility
- How to create a Time Series Insights (TSI) Environment

Revision History

Date	Version	Revision
01 Jul 2019	01	Preliminary release
05 Sep 2019	02	Added documentation to work around a Visual Studio 2019 issue where the “Add Connected Service” utility does not work.
08 Nov 2019	03	Added table showing telemetry data for TSI exercise
14 Nov 2019	04	Updates for 19.10 release Added details to use default device twin properties in deployment group
24 Dec 2019	05	Updates for the 19.11 release Reworked the section to update the app_manifest.json file Removed references to defunct “add connected service” feature Added details around the new CMAKE build process
19 Jan 2020	06	Corrected the validation-certificate download command Updated the VS project name to AzureSphereHacksterTTC Removed OTA from the listed objectives for this lab Miscellaneous clean-up and readability edits
19 Jan 2020	07	Updated initial device twin details to enable the blue LED to on so students can see when device is connected to IoT Huyb
19 Feb 2020	08	Corrected a couple of typos