Measuring Occupancy Part 2 – Connecting a Delta Controls O3 Edge to Azure IoT

Contents

Copyright and Indemnification

Authors

- Spyros Sakellariadis, IoT/Smart Buildings consultant, spyros@spyros.com
- Maksym Mushkin, IoT/Smart Buildings architect, max.mushkin@outlook.com
- Gamal Mustapha, Director of Product Management, Delta Controls Inc.

1 Introduction

In a previous whitepaper, *Measuring Occupancy with Delta Controls O3 Sense, Azure IoT, and ICONICS* (published simultaneously on the Delta website and on ICONICS website), we described the value of monitoring the occupancy of spaces in commercial buildings, and detailed how to deploy an infrastructure to do so. This whitepaper presents a simpler and lower cost architecture for collecting and storing the data that would be appropriate for some enterprise environments.

2 Infrastructure overview

2.1 On-premises infrastructure

In the setup described in this paper, we are using a Delta Controls O3™ Edge to monitor room occupancy with a combination of temperature, humidity, motion, sound, and light sensors. The O3 Edge is the programmable version of the O3 Sense, it has a hardwired ethernet connection to a local area network with access to the Internet, and sends data using the AMQP protocol over the Internet to applications in the Microsoft Azure cloud. The on-premises configuration is shown in Figure 1:

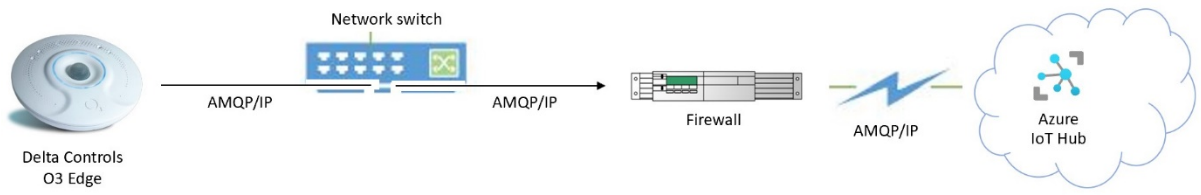


Figure 1 Physical Layout

The difference between this configuration and that described in the previous whitepaper is that here the O3 Edge pushes data directly to Azure, whereas in the previous paper we use an on-premises gateway from ICONICS to pull data from the O3 Edge and push it to Azure. The push method is simpler in that it does not require a separate local computer and application, but the pull method has the advantage that the gateway can consolidate data from many on-premises devices and run local processes to validate and analyze the data before transmitting it to Azure. The enterprise ultimately needs to decide which configuration to use. The two configurations are fully compatible, in that they both can send data to Azure using the same data schema, so the enterprise may use the direct-connect method in one set of rooms or buildings, and the gateway-connected method in another.

2.2 Cloud infrastructure

The O3 Edge connects directly to an Azure IoT Hub installed in the enterprise's Azure subscription. In the configuration described in this paper, we use Azure IoT Hub's message routing feature to route data from the IoT Hub to an Azure Event Hub based on the origin and type of data, and then use an Azure Function to read the incoming data stream and write it to an Azure Table. Once the data is in the table, we show how to view it with Azure Storage Explorer and create a report with Power BI. The overall flow is shown in Figure 2.

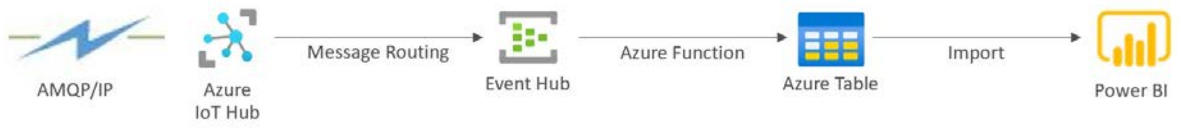


Figure 2 Cloud components

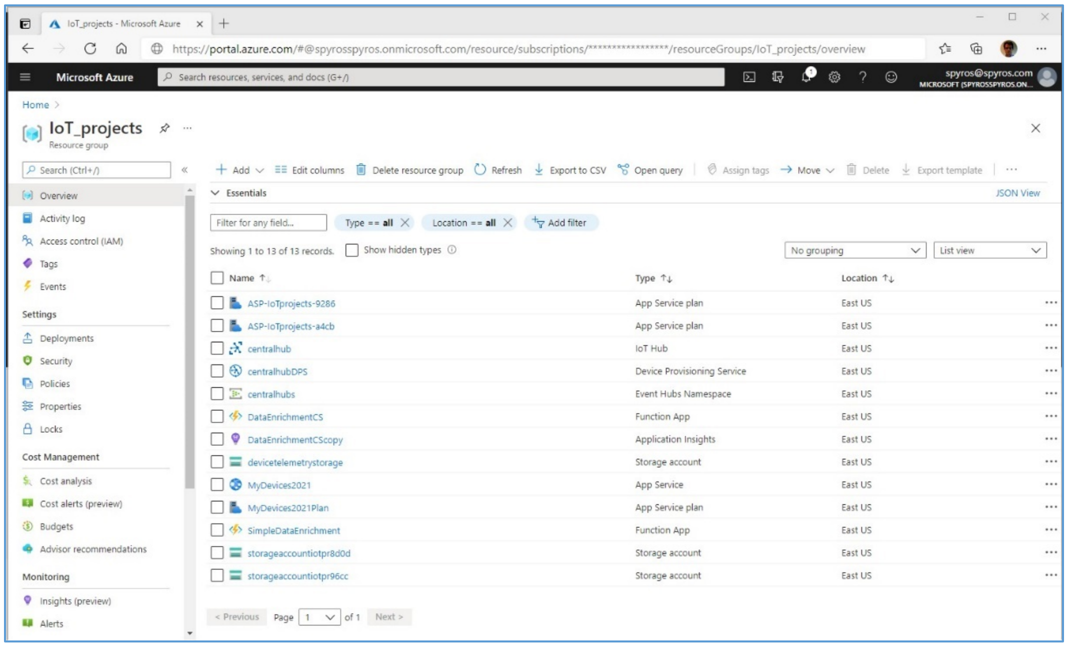
The following sections contain a description of how to configure the O3 Edge and the Azure components to monitor the occupancy and other elements detected by the O3 Edge.

3 Configuring Azure prerequisites

3.1 Azure Resource Group

This article assumes the reader has basic knowledge of Microsoft cloud products and services and understands how to create and configure resources. Consequently, only descriptions or diagrams of the final configuration will be included, not step-by-step instructions.

The example described here uses various Azure services, deployed in a single resource group shown below. We called the resource group **IoT_projects** when creating this configuration. The final set of services is shown below in Figure 3 :



Showing 1 to 13 of 13 records.

Name	Type	Location
ASP-IoTprojects-9286	App Service plan	East US
ASP-IoTprojects-a4cb	App Service plan	East US
centralhub	IoT Hub	East US
centralhubDPS	Device Provisioning Service	East US
centralhubs	Event Hubs Namespace	East US
DataEnrichmentCS	Function App	East US
DataEnrichmentCScopy	Application Insights	East US
devicetelemetrystorage	Storage account	East US
MyDevices2021	App Service	East US
MyDevices2021Plan	App Service plan	East US
SimpleDataEnrichment	Function App	East US
storageaccountiotpr8d0d	Storage account	East US
storageaccountiotpr96cc	Storage account	East US

Figure 3: Azure Resource Group

The key services we will use in this solution are the following:

Resource	Type	Function
centralhub	IoT Hub	Receive data from the O3 Edge
centralhubs	Event Hubs Namespace	Holds multiple Event Hubs
Storageaccountiotpr96cc	Storage account	Stores telemetry from the O3 Edge
DataEnrichmentCS	Function App	Writes data from Event Hub to Storage

3.2 Azure IoT Hub

The first task after creating the empty resource group is to create an Azure IoT Hub to receive the data from the O3 Edge. In the Azure portal select **+ Create a resource,** select the **Internet of Things** category, and click on **IoT Hub**. To create the environment used in this example, set the parameters as follows:

Settings	Value
Subscription	Enter your Azure IoT subscription name. In our example, this is **Subscription-1**.
Resource Group	Enter **IoT_projects**.
Region	Select the region where you have created the IoT Hub. In our example, this is **East US**.
IoT Hub Name	Enter **centralhub.**

Next, select the **Built-in endpoints** category, and create a couple of consumer groups for use by different readers of the data:

- deltao3hub
- o3

Next, from the left menu select **IoT Devices**, then select **+ New** at the top of the page to create a new device. Add the following:

Name	Value
Device ID	Enter **DeltaO3**.

Finally, note the following parameters for the IoT Hub, which will be needed later:

Parameter	Value
Host name	From **Overview tab**
IoT Hub primary connection string	From **Shared Access policies** à **iothubowner**
Device primary connection string	From **IoT devices** à **DeltaO3**

3.3 Event Hub

Next, we need an Event Hub to which we will route a subset of the data coming into IoT Hub. In the Azure portal select **+ Create a resource,** enter **Event Hubs** in the search category, click on **Event Hubs** and **Create**. To create the environment used in this example, set the parameters as follows:

Settings	Value
Subscription	Enter your Azure IoT subscription name. In our example, this is **Subscription-1**.
Resource Group	Enter **IoT_projects**.
Namespace name	Enter **centralhubs**
Location	Select the region where you have created the IoT Hub. In our example, this is **East US**.
Pricing tier	Select **Standard.** Do not select Basic, as Basic allows only one consumer group and we need two to use Visual Studio to view data coming into the Event Hub.

Click **Review + create.** Once the Event Hub is created, go to the resource. From the left menu, select **Event Hubs** and click **+ Event Hub** at the top of the page. To create the environment used in this example, set the parameters as follows:

Settings	Value
Name	Enter **deltao3**.

Next, select the **Consumer groups** category, and several consumer groups for use by different readers of the data:

- monitoring
- monitoringwithvscode
- EventHub2Table

4 Configuring the O3 Edge

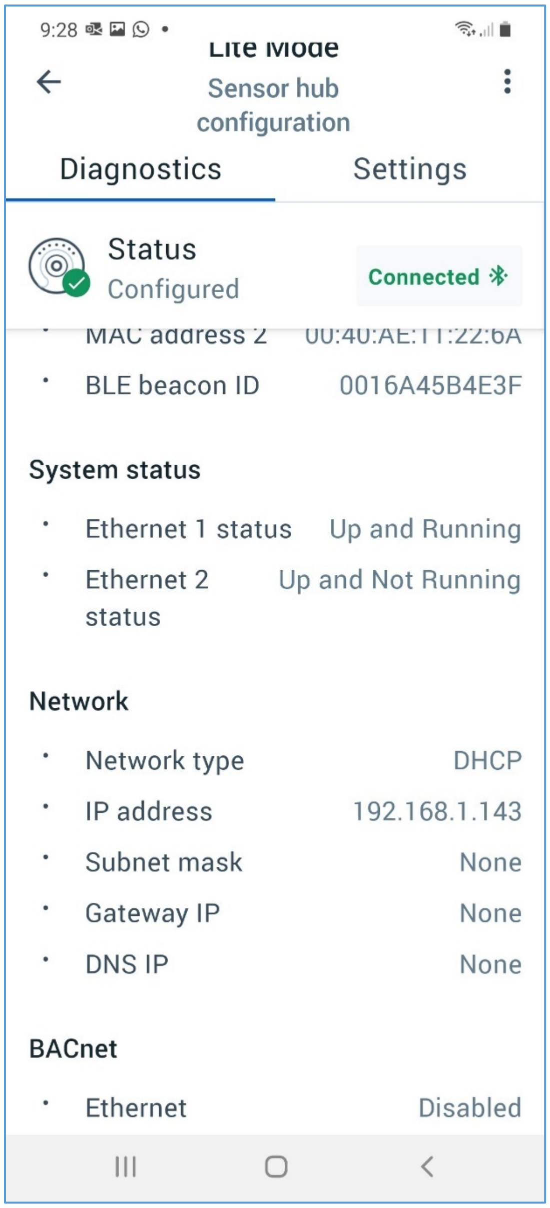
4.1 Setting up the O3 Edge

This guide from Delta Controls describes how to install and set up the O3 Edge. To set up the O3, you will need an Android or iOS device with the O3 Setup app installed. You can get the app from Google Play or the App Store.

Key steps to configure the O3 are as follows:

1. Open the O3 Setup app and select Continue to enter Lite Mode.
2. In the lower right corner of the screen, select Connect.
3. Select your O3 to initiate a connection over Bluetooth, O3 units are displayed in the order of signal strength.
4. Once the connection is initiated, select Verify. The O3 should play a sound and the light ring flashes blue.
5. Select *Yes, connect to this hub*. Data loads from the hub and the status changes to Connected.
6. You can now view device information and sensor data from the hub in the Diagnostics tab.
7. After connecting to the hub, select the Settings tab.
8. By default, the O3 is set to DHCP. If you want to assign a static IP address to the O3, select the pencil icon next to Network, select Static, enter the IP settings, then select Save.

Click **Apply settings to hub**, then click on the **Diagnostics** tab to see additional information:



9:28

Lite Mode

Sensor hub configuration

Diagnostics | Settings

Status
Configured
Connected

MAC address 2	00:40:AE:11:22:6A
BLE beacon ID	0016A45B4E3F

System status

Ethernet 1 status	Up and Running
Ethernet 2 status	Up and Not Running

Network

Network type	DHCP
IP address	192.168.1.143
Subnet mask	None
Gateway IP	None
DNS IP	None

BACnet

Ethernet	Disabled

Figure 4: Diagnostics for the O3

Confirm that the O3 Edge is connected to the local LAN and has received an IP address, so that it can communicate outbound.

4.2 Programming the O3 Edge to send data to Azure

4.2.1 Preparing the environment

The O3 Edge makes use of a development tool called Node-RED that has proven to be an effective tool to facilitate interoperability between different systems. Node-RED is a programming tool geared towards wiring together hardware devices, APIs, and online services. A developer version of the O3 Edge has a built-in web server that allows access to its browser-based editor giving developers direct access to the MQTT topics and hardware interfaces in the O3. The available browser-based editor makes it easy to wire together flows using the wide range of pre-built nodes in the palette, in our case we take advantage of the pre-built Azure-IoT Hub node.

In addition to using pre-built nodes, JavaScript functions can be created using the rich text editor that can be easily linked to third party nodes. Node-RED is built on Node.js and takes full advantage of its event-driven, non-blocking model making it an ideal solution to process event-based requests at the edge of the network.

To access the Node-RED webserver, use any web browser to connect to the device's URL on port 1880: http://<ip address>:1880. The IP address given to the hub over DHCP can be obtained via the Diagnostics tab on the O3 Setup app as shown in Figure 4.

Before we can create a flow to push data to Azure IoT Hub, we need to install the Azure IoT Hub palette with the appropriate nodes required for accessing Azure. On the opening screen, click on the three bars at the top right and select Manage palette:

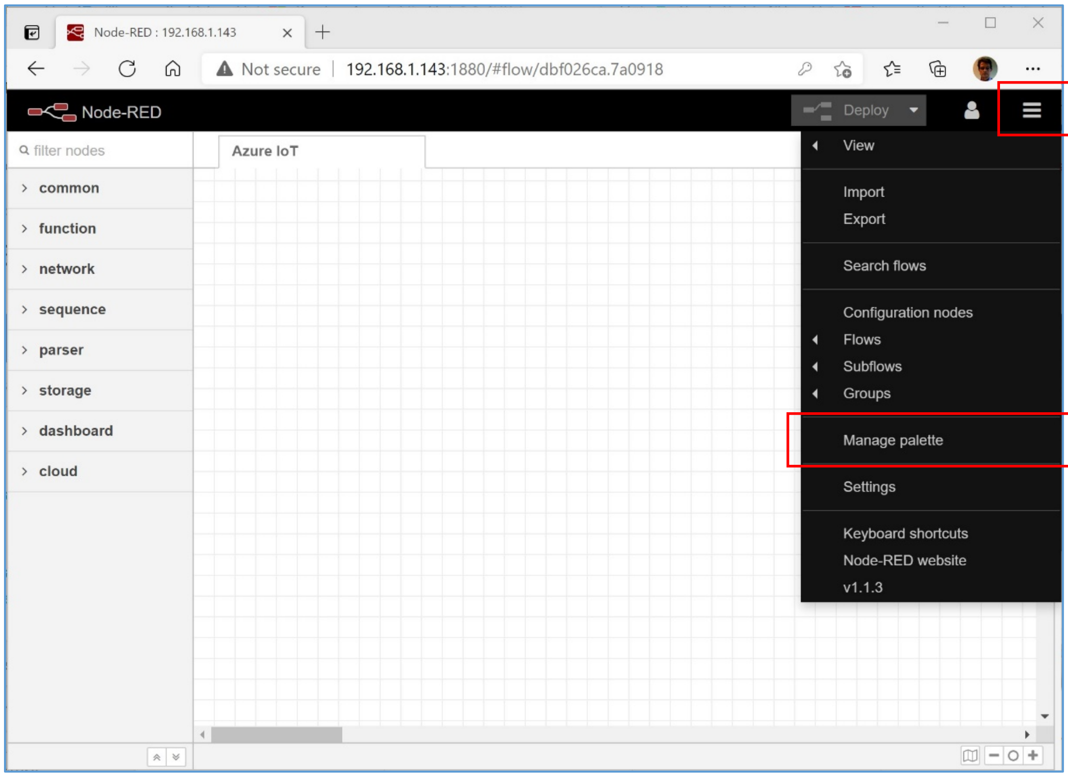


Figure 5: Node-RED opening screen

In the next screen, select the **Install** tab to search for **node-red-contrib-azure-iot-hub**:

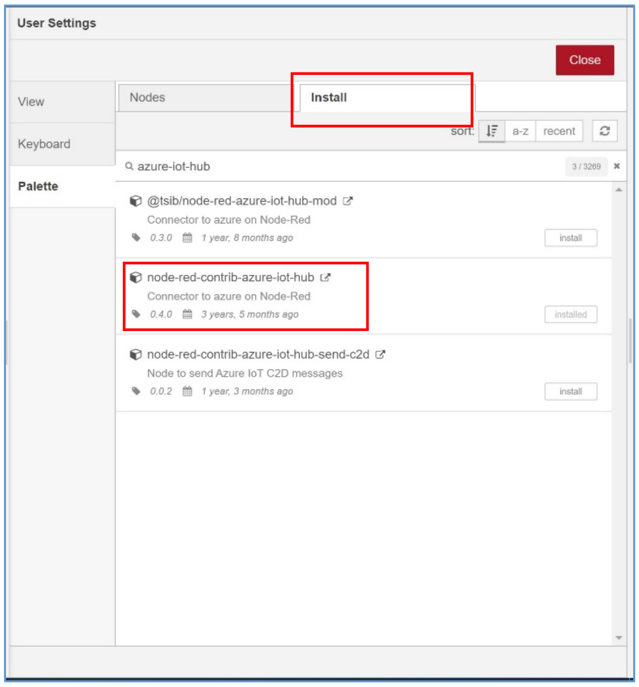

User Settings
Close
View
Keyboard
Palette
Nodes
Install
sort:
a-z
recent
azure-iot-hub
3 / 3269
@tsib/node-red-azure-iot-hub-mod
Connector to azure on Node-Red
0.3.0
1 year, 8 months ago
install
node-red-contrib-azure-iot-hub
Connector to azure on Node-Red
0.4.0
3 years, 5 months ago
installed
node-red-contrib-azure-iot-hub-send-c2d
Node to send Azure IoT C2D messages
0.0.2
1 year, 3 months ago
install

Figure 6: Node-RED Add Azure IoT Hub module

When the module installation is complete you will see it in the **nodes** tab:

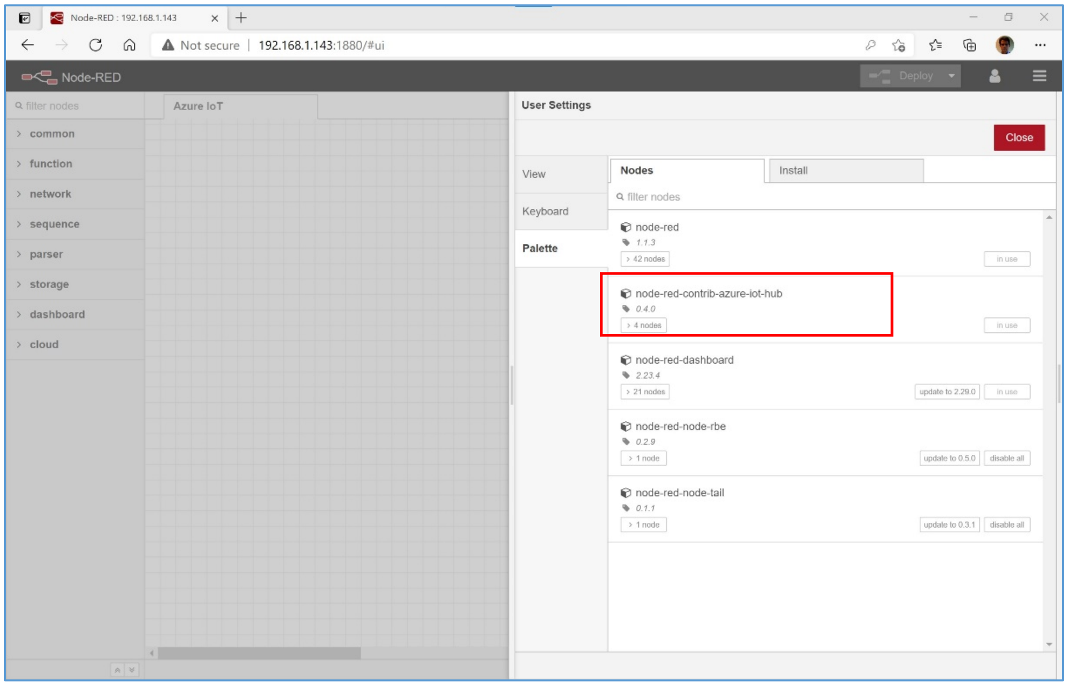
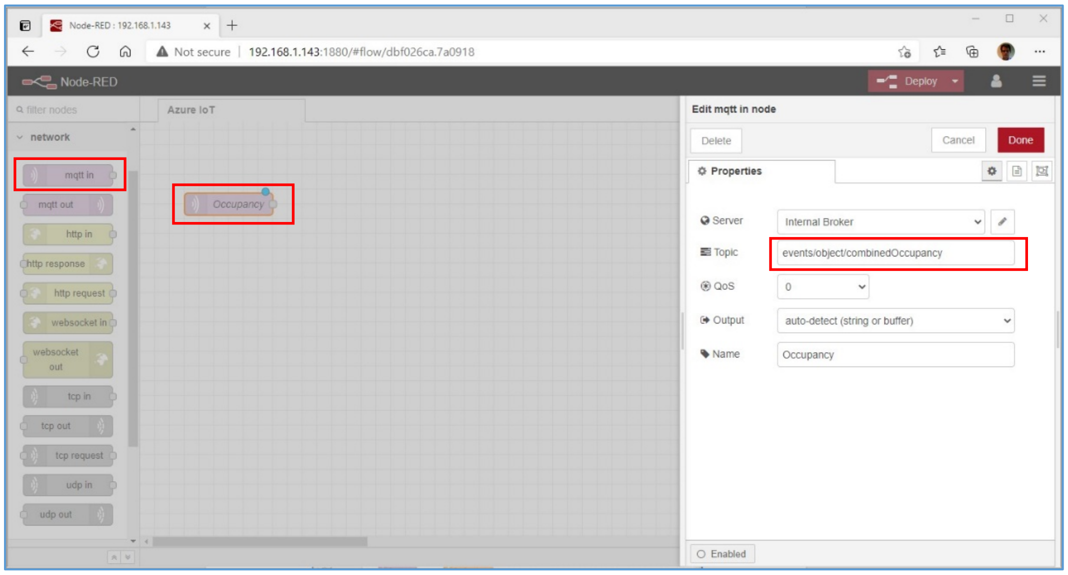


Figure 7: Node-RED Manage palette

4.2.2 Creating the flow

Now we need to create a 'flow' and deploy it so the data can be sent from the hub to Azure. This involves creating 'nodes' and connecting them. First, we create a node to get the occupancy data from the O3 Edge. We need to know the name to use to request the occupancy data, which you can find in the published list of MQTT topics here: O3 Sensor Hub 2.0 MQTT API Reference Guide.

For occupancy, the MQTT topic we need to use is **events/object/combinedOccupancy**, to create the occupancy node drag an **mqtt in** node onto the canvas and enter data as show below:



Server: Internal Broker
Topic: events/object/combinedOccupancy
QoS: 0
Output: auto-detect (string or buffer)
Name: Occupancy

Figure 8: Node-RED Add Occupancy node

Repeat this process for all the other sensors using the following MQTT topics:

Sensor Name	MQTT Topic	Description
Occupancy	events/object/combinedOccupancy	Indicates occupancy based on the PIR motion sensor, sound level sensor and IR sensor
Temperature	events/object/occupantTemperature	Returns temperature based on IR temperature sensor and built-in thermistors in the unit
Humidity	events/object/occupantHumidity	Returns calculated humidity at the occupant height
Sound Level	events/object/soundLevel	Gives sound level in dB (does not record or translate speech)
Light Level	events/object/lightLevel	Returns light level as seen by the hub
Motion	events/object/motion	Uses PIR sensor to return the presence of motion

The finished input section will look like this:

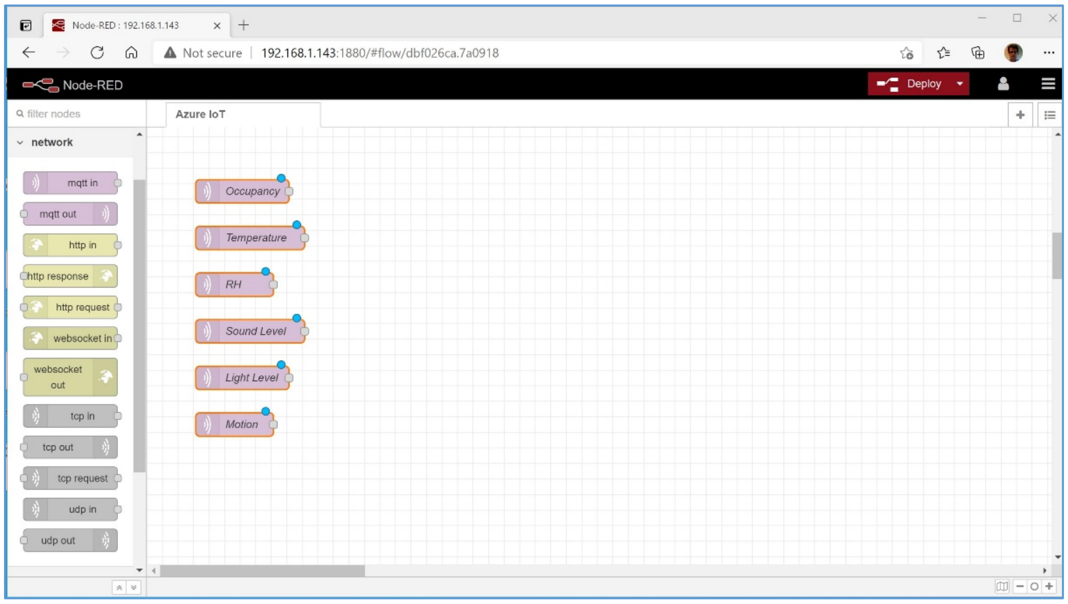
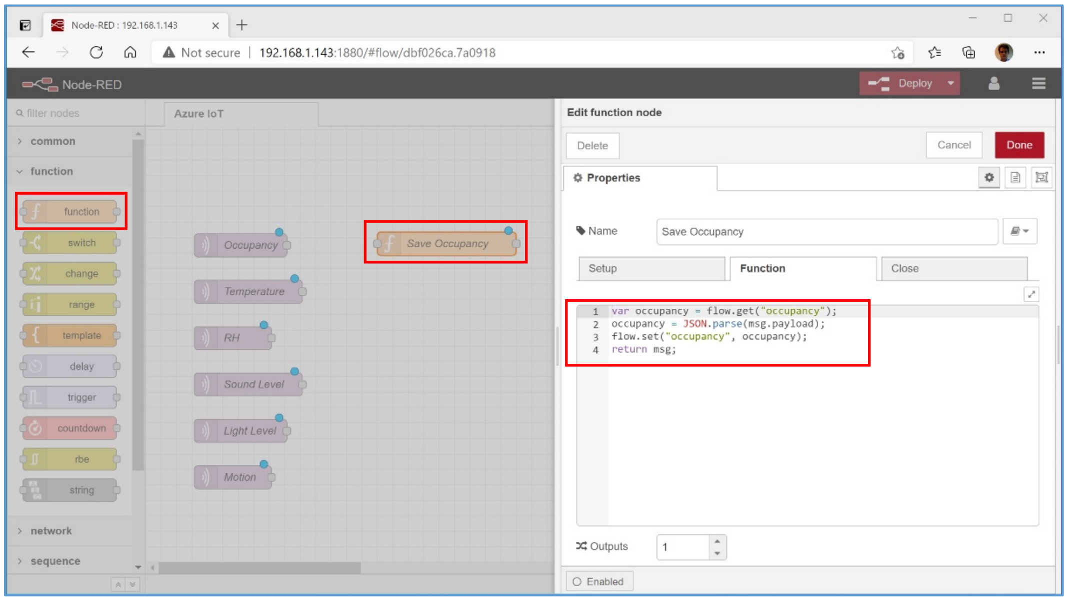


Figure 9: All sensor nodes

Next, we need to create a node that processes the data retrieved from the O3 Edge. To control how and when the data is sent to Azure, we will save the data temporarily in a local variable which we will retrieve later. The code we will use to do this is this:

```
var occupancy = flow.get("occupancy");
occupancy = JSON.parse(msg.payload);
flow.set("occupancy", occupancy);
return msg;
```

To do this, create a **Save Occupancy** node by dragging a **function** node onto the canvas and enter the code above as show below:



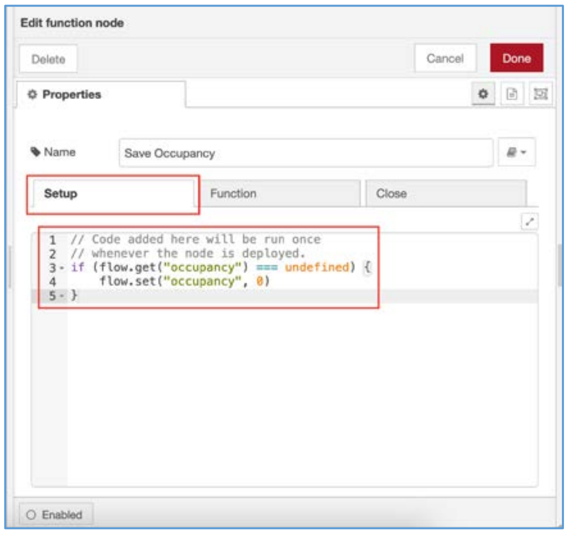
Name: Save Occupancy

```
var occupancy = flow.get("occupancy");
occupancy = JSON.parse(msg.payload);
flow.set("occupancy", occupancy);
return msg;
```

Outputs: 1

Figure 10: Add Save Occupancy node

Select the Setup tab and enter the following code to initialize the occupancy variable.



Name: Save Occupancy

```
// Code added here will be run once
// whenever the node is deployed.
if (flow.get("occupancy") === undefined) {
    flow.set("occupancy", 0)
}
```

Figure 11. Initialize local variable

Once the node is created, to tell it where the occupancy data is, you need to wire it to the occupancy node. Drag a line from the **Occupancy** node to the **Save Occupancy** node:

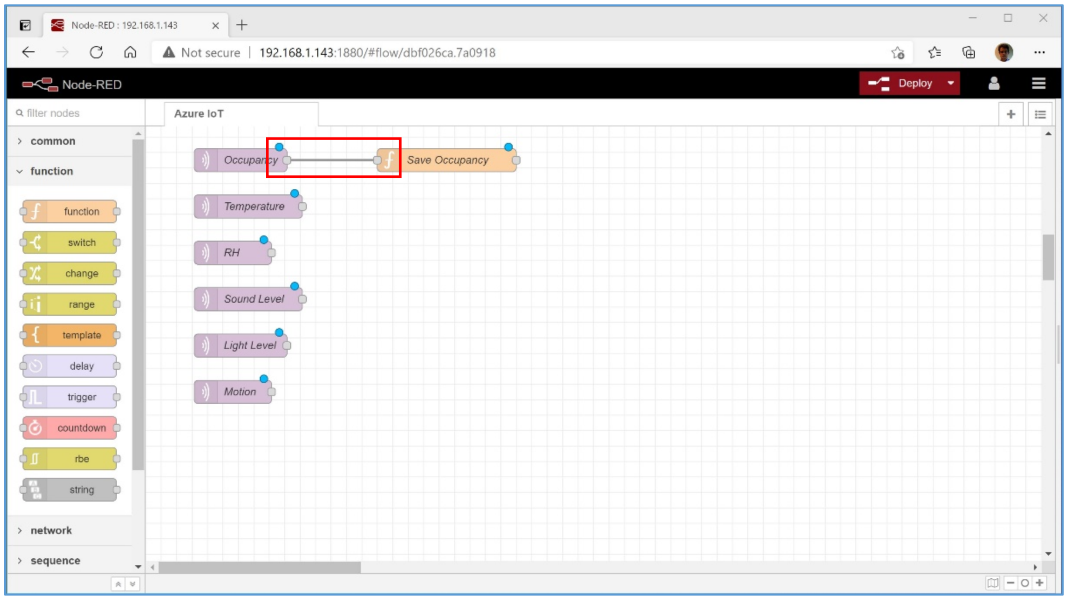


Figure 12: Save Occupancy node connected

Repeat this process for all the other nodes, adjusting the **Save** code appropriately. The finished product:

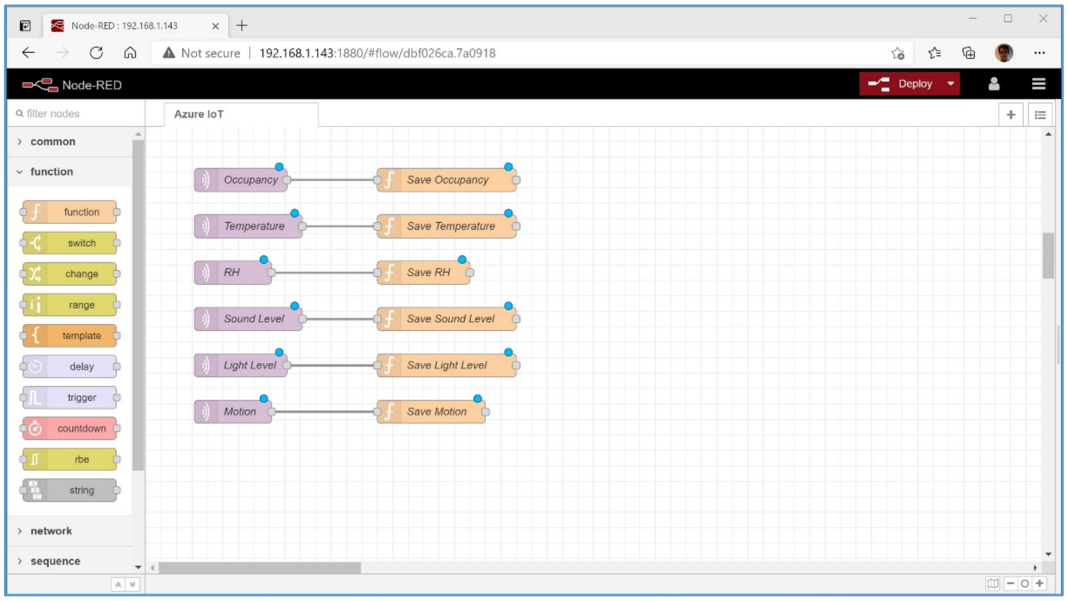


Figure 13: Save nodes connected

Next, we need to prepare the date to send to Azure. For this, we create a **Prepare Sensors Message** function node and connect it to any one of the **Save** nodes:

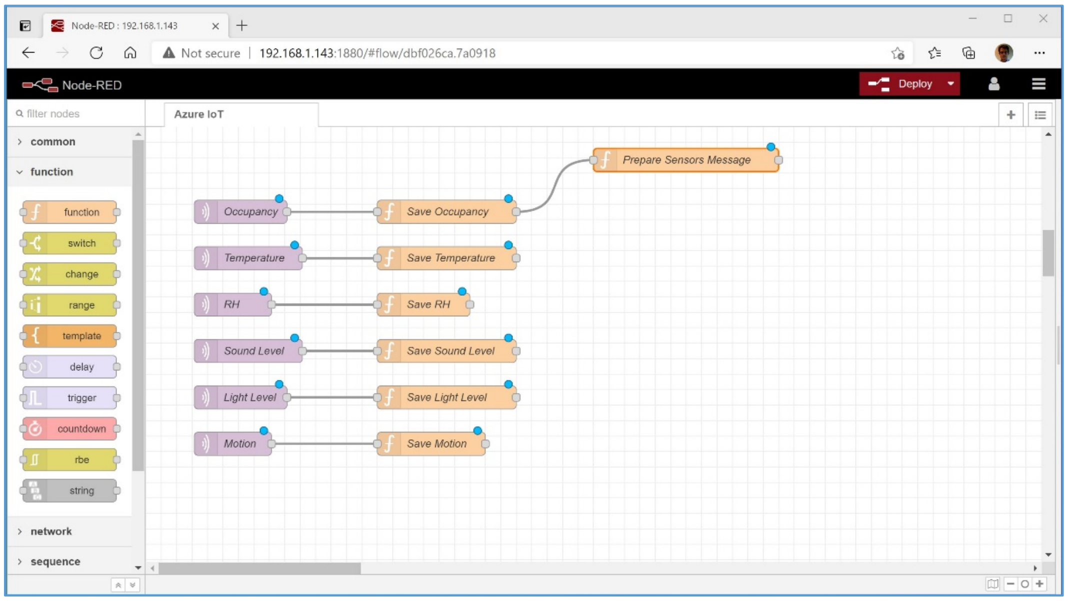


Figure 14: Prepare sensors node

The code we put inside the **Prepare Sensors message** is as follows:

```
//cycle through all sensors and send a friendly id name
//as well as the Present_Value of that sensor
//
var id = ““; // sensor name
var pv = ““; // present value
var s = “15s”; // indicator that message is sent at regular value of 15 seconds
var time = new Date().toISOString();

for (i = 1; i < 7; i++) {
    if (i==1) {
        id = “Occupant_temperature”;
        pv = flow.get(“temperature”).Present_Value;
    }
    if (i==2) {
        id = “Humidity”;
        pv = flow.get(“rh”).Present_Value;
    }
    if (i==3) {
        id = “Sound_level”;
        pv = flow.get(“sound”).Present_Value;
    }
    if (i==4) {
        id = “Light_level”;
        pv = flow.get(“lightlevel”).Present_Value;
    }
    if (i==5) {
        id = “Occupancy”;
        pv = flow.get(“occupancy”).Present_Value;
    }
    if (i==6) {
        id = “Motion”;
        pv = flow.get(“motion”).Present_Value;
    }
    msg.payload = {
        'deviceId':”Delta03”,
        'key':”OpzBY78kK4dn2J2K548MMgWn0SIJOHDeTosqmaASTw0=“,
        'protocol':”amqp”,
        'data':{“id”:id,”v”:pv,”t”:time, “s”:s}
    } // end msg.payload
    node.send(msg);
} // end for
return;
```

To get the JSON structure we want to send to Azure, we need to create the variables **id** (the thing being measured), **pv** (the present value of that thing)**, time,** and **s** (a variable which will distinguish whether we are sending on a regular interval or change of value). In the code, we initialize the variables at the top and then get the saved data in a loop, putting it into the variables. We then create the message payload using the structure needed for the Azure message.

Finally, we need to add another function node that will connect to Azure IoT Hub:

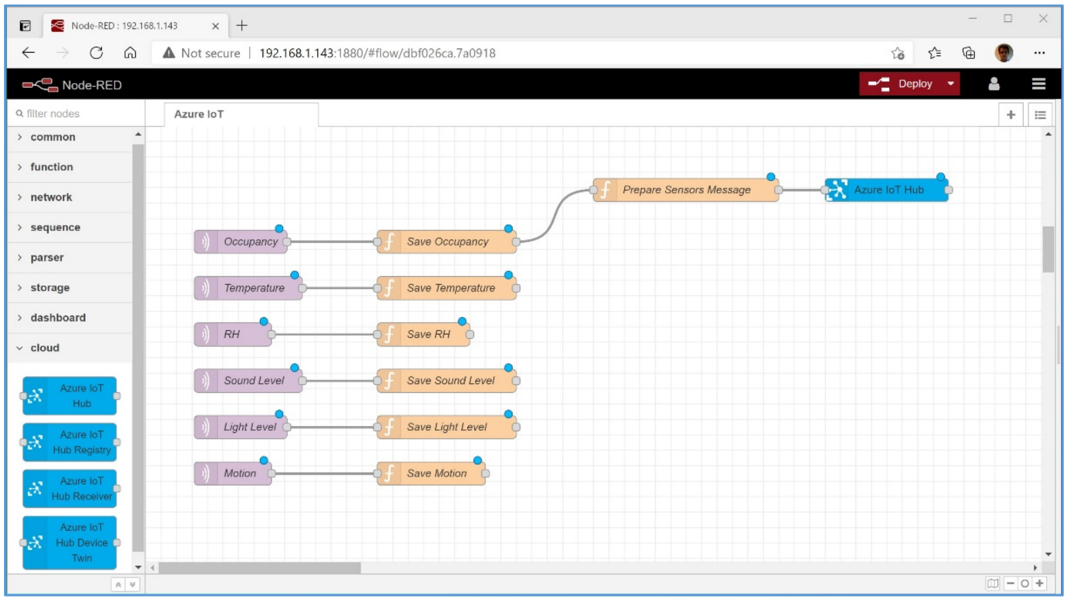
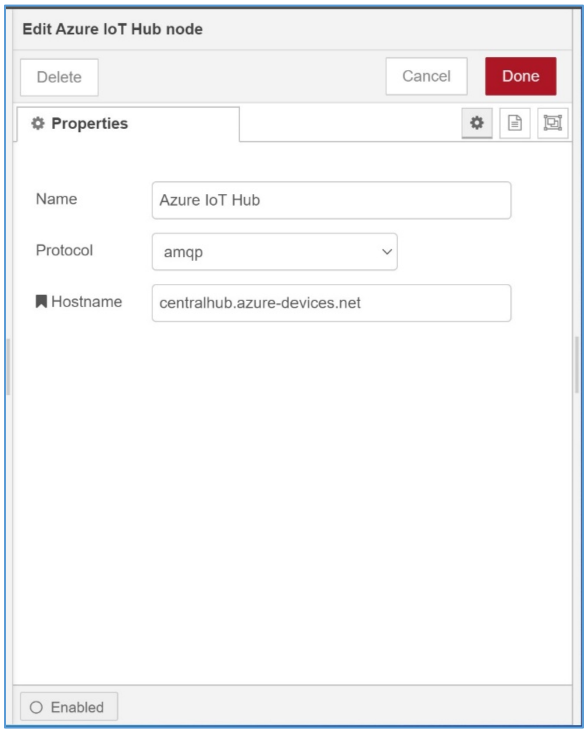


Figure 15: Add Azure IoT Hub node

Inside the **Azure IoT Hub** node we add the URL for the IoT Hub we created in Section 3.2 above:



Edit Azure IoT Hub node

Delete Cancel Done

Properties

Name: Azure IoT Hub

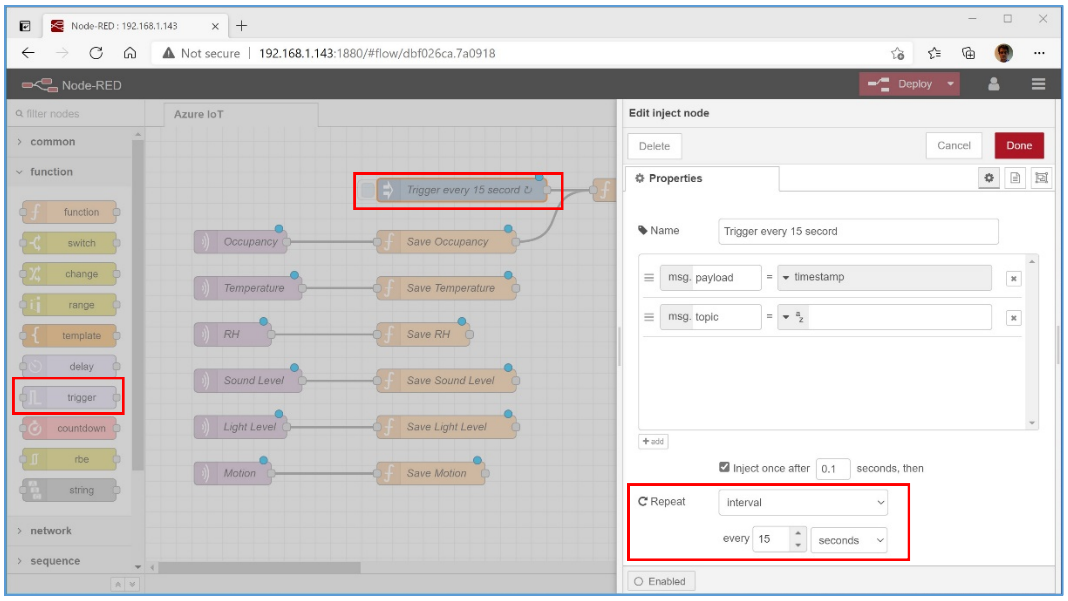
Protocol: amqp

Hostname: centralhub.azure-devices.net

Enabled

Figure 16: Add Azure IoT Hub node

This completes the end-to-end flow. Now we need to add a node to specify the interval to send data. To do this, we add a **Trigger** node with the configuration shown below, and wire it to the **Prepare Sensors Message** node:



Name: Trigger every 15 secord

msg. payload = timestamp

msg. topic =

Inject once after 0.1 seconds, then

Repeat: interval

every 15 seconds

Figure 17: Add Trigger node

Sensor data changes on Change of Value (COV) and is saved when the COV increment is exceeded. This flow will retrieve the saved data, process, and send data to Azure IoT hub every 15 seconds. We have set the time interval small for testing purposes. In production, we would probably change this to about every five minutes as there is no need to record sensor data such as room temperature or humidity every 15 seconds. However, we will want to know the instant occupancy status changes, not just on a regular interval. To do this, we add a separate **Prepare Occupancy Message** node that is executed on change of value (COV):

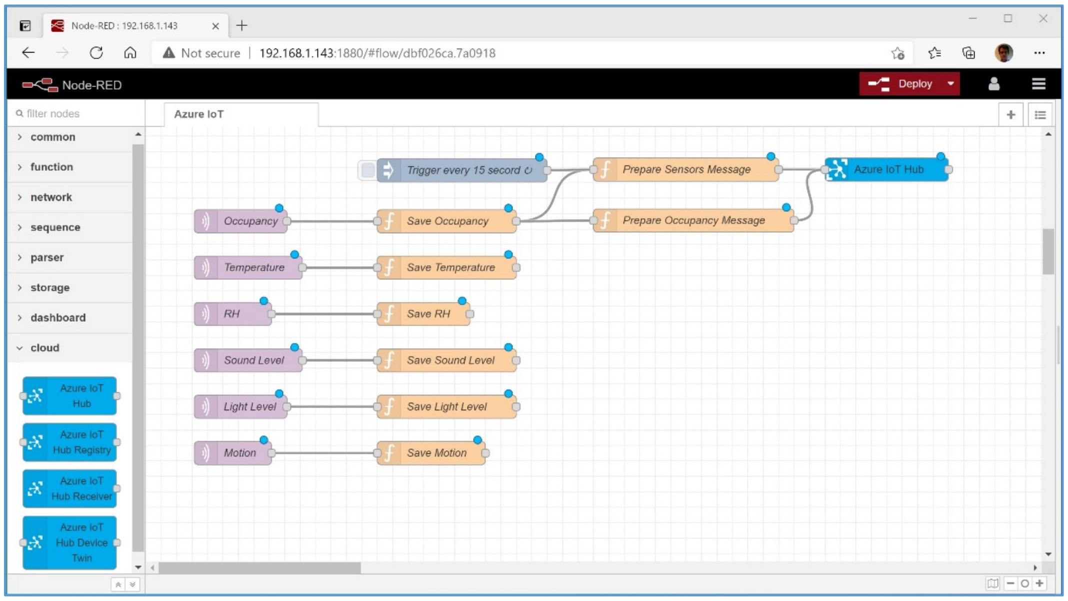


Figure 18: Add Prepare Occupancy node

The new node, including the code inside it, is similar to the **Prepare Sensors Message**, with the exception that the node is not wired to the trigger node, and we have changed the explanatory variable **s** from **15s** to **COV**:

```
var occupancy = flow.get("occupancy");
var s = "COV"; // indicator that message is sent because value changed
msg.payload = {
    'deviceId':"DeltaO3",
    'key':"OpzBY78kK4dn2J2K548MMgWn0SIJOHDeTosqmaASTw0=",
    'protocol':"amqp",
    'data':{"id":"Occupancy","v":occupancy.Present_Value,"t":new Date().toISOString(), "s":s}
}
return msg
```

Having created all the nodes, click the red **Deploy** button at the top right to activate this flow. Note the solid green boxes under the MQTT incoming messages and the solid blue box under the Azure IoT Hub box, this indicates that the nodes are successfully connected. If this box is not solid check the syntax and spelling used in these nodes. The completed flow looks like this:

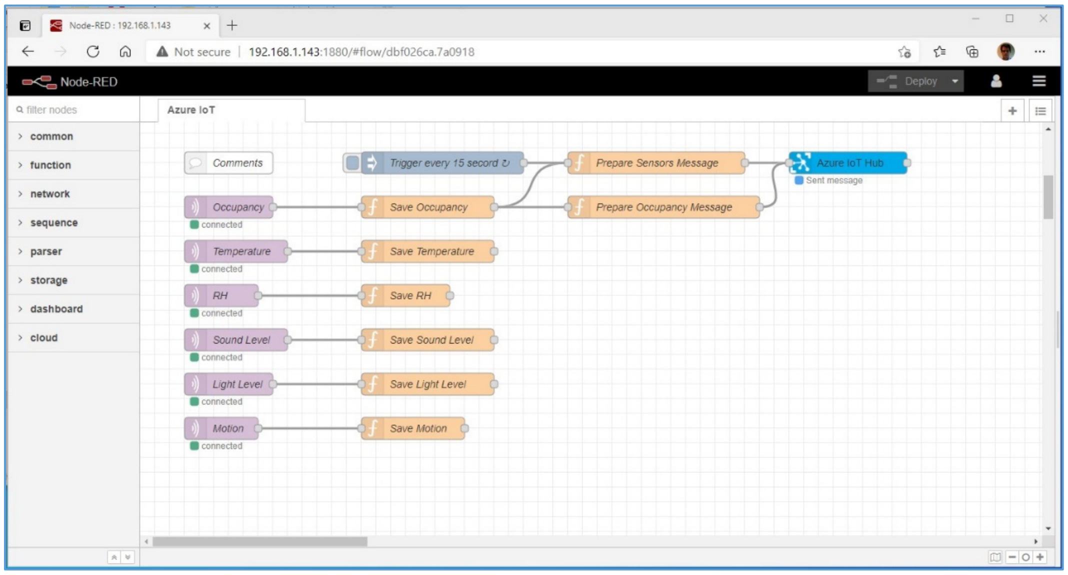
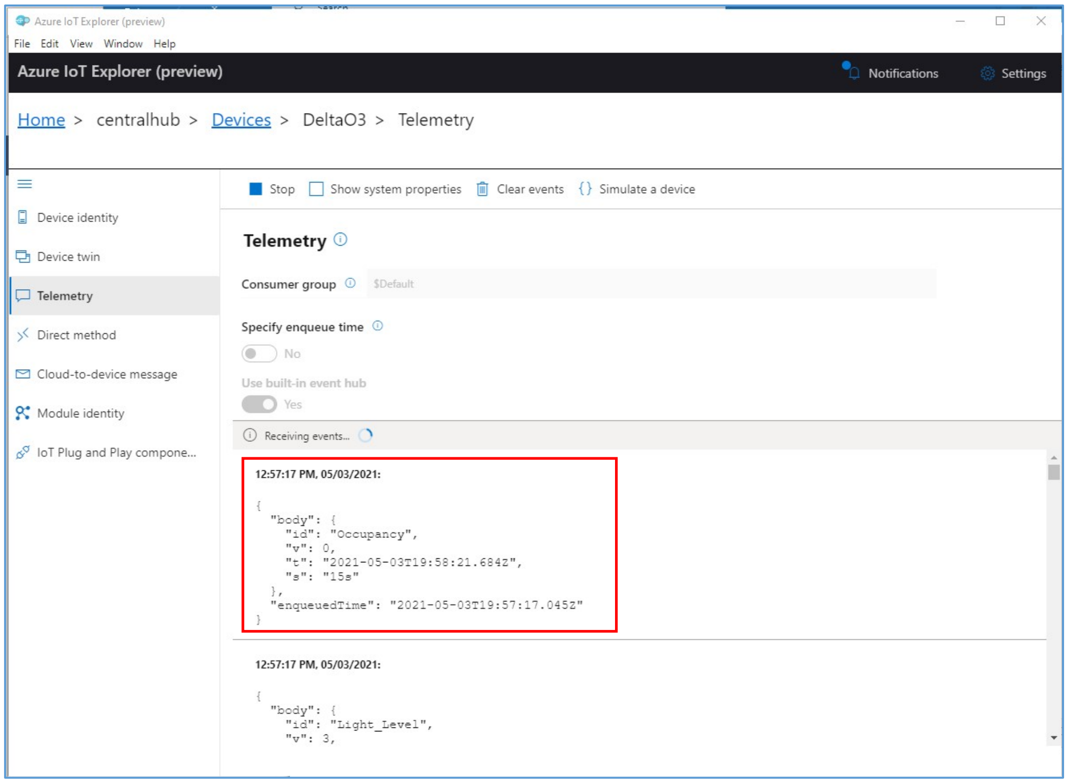


Figure 19: Completed flow

4.3 *Viewing data received by IoT Hub*

See Install and use Azure IoT explorer for step-by-step instructions for using the Azure IoT explorer tool to monitor incoming data. Upon launching Azure IoT Explorer, enter **the IoT Hub primary connection string** noted in Section 3.2 above.

If the O3 Edge and IoT Hub are configured as described in this article, after navigating to **centralhub → Devices → DeltaO3 → Telemetry** and clicking **Start**, you should see the O3 Edge data streaming in every 15 seconds:



```
12:57:17 PM, 05/03/2021:

{
  "body": {
    "id": "Occupancy",
    "v": 0,
    "t": "2021-05-03T19:58:21.684Z",
    "s": "15s"
  },
  "enqueuedTime": "2021-05-03T19:57:17.045Z"
}
```

```
12:57:17 PM, 05/03/2021:

{
  "body": {
    "id": "Light_Level",
    "v": 3,
```

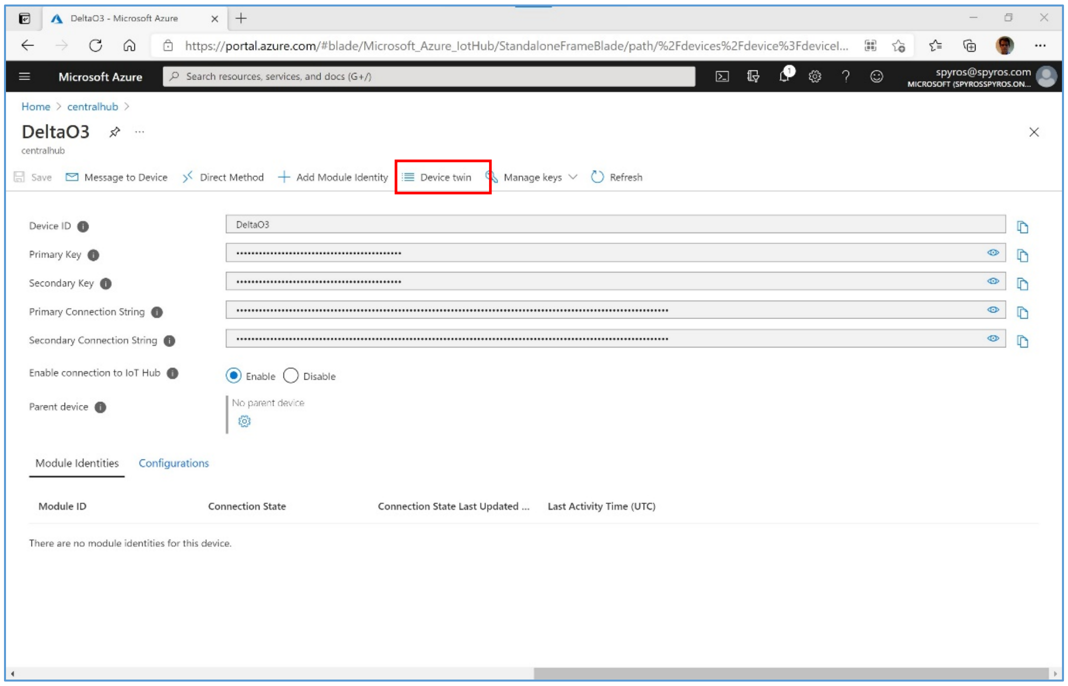
Figure 20: Azure IoT Explorer showing data from O3 Edge

5 Routing data from IoT Hub to Event Hub

Typically, you would have many devices send data to the same IoT Hub, so we need a way to filter the incoming data from just the O3.

5.1 *Creating a filter for the data*

First, we need to create an attribute on the incoming data by which to filter it. To do this, we add a property to the Azure device twin for the device as configured in the IoT Hub. In the Azure portal, select the IoT Hub **centralhub** and click on **IoT devices** and select the **DeltaO3** device. On the **DeltaO3** device page, click on **Device twin**:


DeltaO3 - Microsoft Azure
https://portal.azure.com/#blade/Microsoft_Azure_IotHub/StandaloneFrameBlade/path/%2Fdevices%2Fdevice%3Fdevicel...
Microsoft Azure
Search resources, services, and docs (G+/)
spyros@spyros.com
Home > centralhub >
DeltaO3
centralhub
Save
Message to Device
Direct Method
Add Module Identity
Device twin
Manage keys
Refresh

Device ID	DeltaO3
Primary Key	••
Secondary Key	••
Primary Connection String	••
Secondary Connection String	••
Enable connection to IoT Hub	[✓ Enable] Disable
Parent device	No parent device

Module Identities | Configurations

Module ID	Connection State	Connection State Last Updated ...	Last Activity Time (UTC)

There are no module identities for this device.

Figure 21: Azure IoT Hub DeltaO3 Device

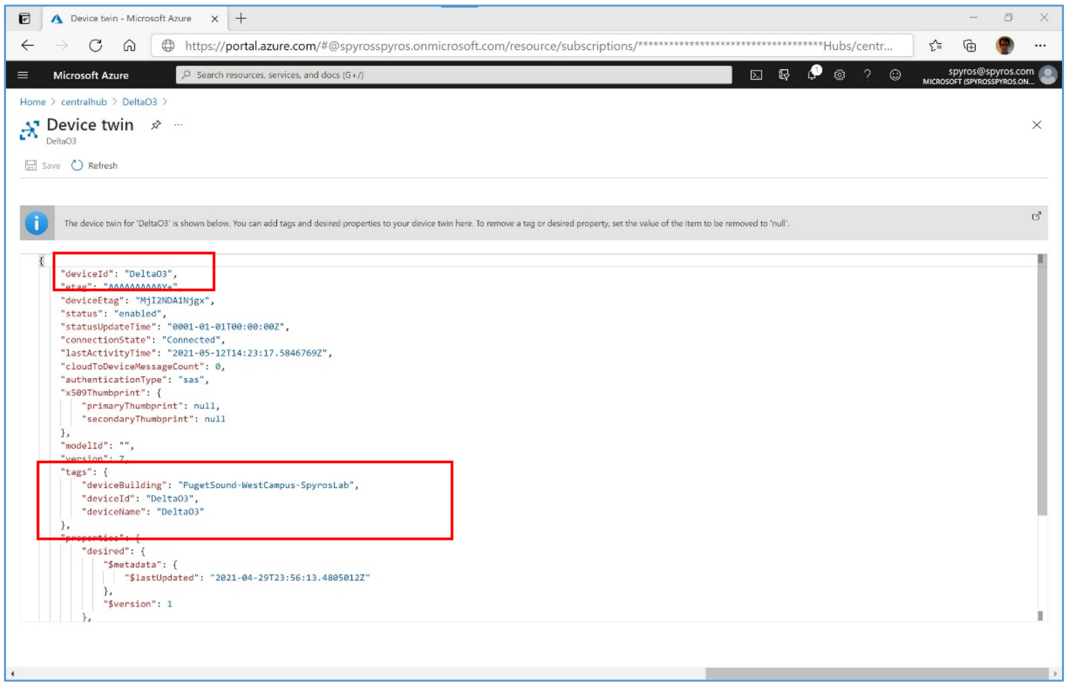
On the next screen, note the value of the deviceID. This was automatically created for the Twin when the IoT device was created in IoT Hub:

```
      "deviceId": "DeltaO3"
```

Now we can add a tags section with device location if you want to use Device Twin Data Enrichment functionality. In the portal add the following:

```
    "tags": {
        "deviceBuilding": "PugetSound-WestCampus-SpyrosLab",
        "deviceName": "DeltaO3"
    },
```

So that it looks like this:


Device twin - Microsoft Azure
https://portal.azure.com/#@spyrosspyros.onmicrosoft.com/resource/subscriptions/**********************************Hubs/centr...
Microsoft Azure
Search resources, services, and docs (G+/)
spyros@spyros.com
MICROSOFT (SPYROSSPYROS.ON...
Home > centralhub > DeltaO3 >
Device twin
DeltaO3
Save
Refresh

The device twin for 'DeltaO3' is shown below. You can add tags and desired properties to your device twin here. To remove a tag or desired property, set the value of the item to be removed to 'null'.

```
{
  "deviceId": "Delta03",
  "etag": "AAAAAAAAAAY=",
  "deviceEtag": "MjI2NDA1Njgx",
  "status": "enabled",
  "statusUpdateTime": "0001-01-01T00:00:00Z",
  "connectionState": "Connected",
  "lastActivityTime": "2021-05-12T14:23:17.5846769Z",
  "cloudToDeviceMessageCount": 0,
  "authenticationType": "sas",
  "x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null
  },
  "modelId": "",
  "version": 7,
  "tags": {
    "deviceBuilding": "PugetSound-WestCampus-SpyrosLab",
    "deviceId": "Delta03",
    "deviceName": "Delta03"
  },
  "properties": {
    "desired": {
      "$metadata": {
        "$lastUpdated": "2021-04-29T23:56:13.4805012Z"
      },
      "$version": 1
    },
```

Figure 22: Azure IoT Hub Device Twin properties

5.2 Configuring routing and data enrichment

Next, we configure the IoT Hub message routing for data with the device twin tag of **DeltaO3** to the Event Hub we created earlier. In the Azure portal, select the IoT Hub **centralhub** and click on **Message routing**
in the left menu.

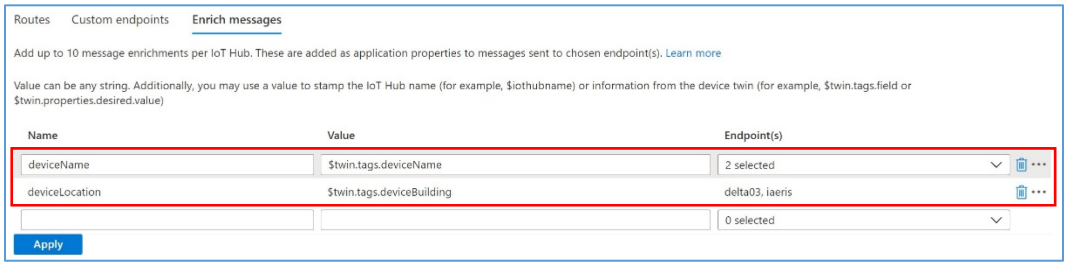
In the **Enrich messages** tab, add two message enrichment entries with the following parameters:

Parameter	**Value**
Name	Enter **deviceBuilding**.
Value	Enter **$twin.tags.deviceBuilding.**
Endpoint	Select **deltao3** in the dropdown, **Event Hubs** section**.**

And

Parameter	**Value**
Name	Enter **deviceName**.
Value	Enter **$twin.tags.deviceName.**
Endpoint	Select **deltao3** in the dropdown, **Event Hubs** section**.**

In the portal it should look like this:



Routes | Custom endpoints | **Enrich messages**

Add up to 10 message enrichments per IoT Hub. These are added as application properties to messages sent to chosen endpoint(s). Learn more

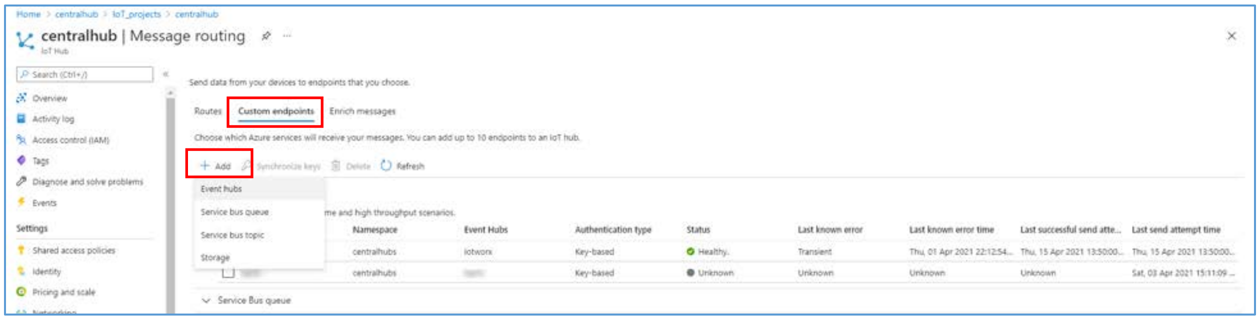
Value can be any string. Additionally, you may use a value to stamp the IoT Hub name (for example, $iothubname) or information from the device twin (for example, $twin.tags.field or $twin.properties.desired.value)

Name	Value	Endpoint(s)
deviceName	$twin.tags.deviceName	2 selected
deviceLocation	$twin.tags.deviceBuilding	delta03, iaeris
		0 selected

Apply

Figure 239: Azure IoT Hub Data Enrichment

Next, to add the route we want, we need to create a Custom Endpoint first. Select **Custom endpoints** tab, click **+ Add**, and select **Event hubs.**



Home > centralhub > IoT_projects > centralhub

centralhub | Message routing

IoT Hub

Send data from your devices to endpoints that you choose.

Routes | **Custom endpoints** | Enrich messages

Choose which Azure services will receive your messages. You can add up to 10 endpoints to an IoT hub.

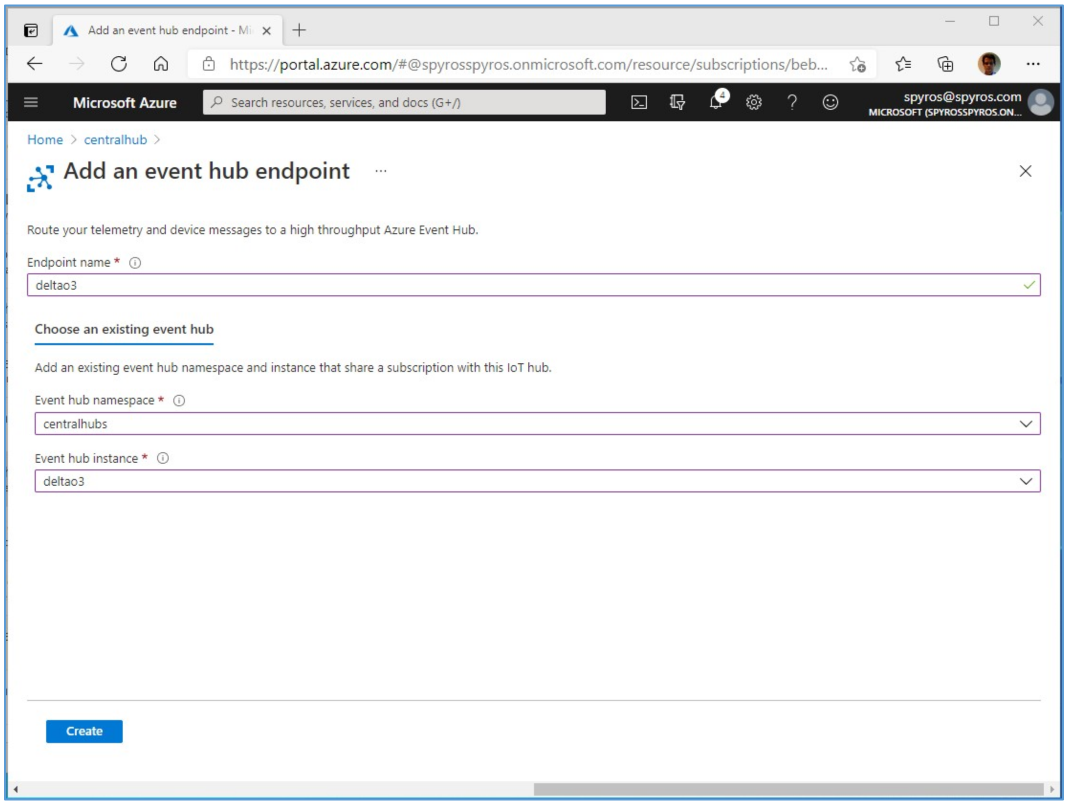
me and high throughput scenarios.

	Namespace	Event Hubs	Authentication type	Status	Last known error	Last known error time	Last successful send atte...	Last send attempt time
	centralhubs	iotworx	Key-based	Healthy.	Transient	Thu, 01 Apr 2021 22:12:54...	Thu, 15 Apr 2021 13:50:00...	Thu, 15 Apr 2021 13:50:00...
[redacted]	centralhubs	[redacted]	Key-based	Unknown	Unknown	Unknown	Unknown	Sat, 03 Apr 2021 15:11:09 ...

Service Bus queue

Figure 20: Azure IoT Hub Custom Endpoints

On the next page, enter **deltao3** for the **Endpoint name**, select **centralhubs** for the **Event hub namespace**, select **deltao3** for **the Event hub instance,** and click **Create:**


Add an event hub endpoint - Mi
https://portal.azure.com/#@spyrosspyros.onmicrosoft.com/resource/subscriptions/beb...
Microsoft Azure
Search resources, services, and docs (G+/)
spyros@spyros.com
MICROSOFT (SPYROSSPYROS.ON...
Home > centralhub >

Add an event hub endpoint

Route your telemetry and device messages to a high throughput Azure Event Hub.

Endpoint name * : deltao3

Choose an existing event hub

Add an existing event hub namespace and instance that share a subscription with this IoT hub.

Event hub namespace * : centralhubs

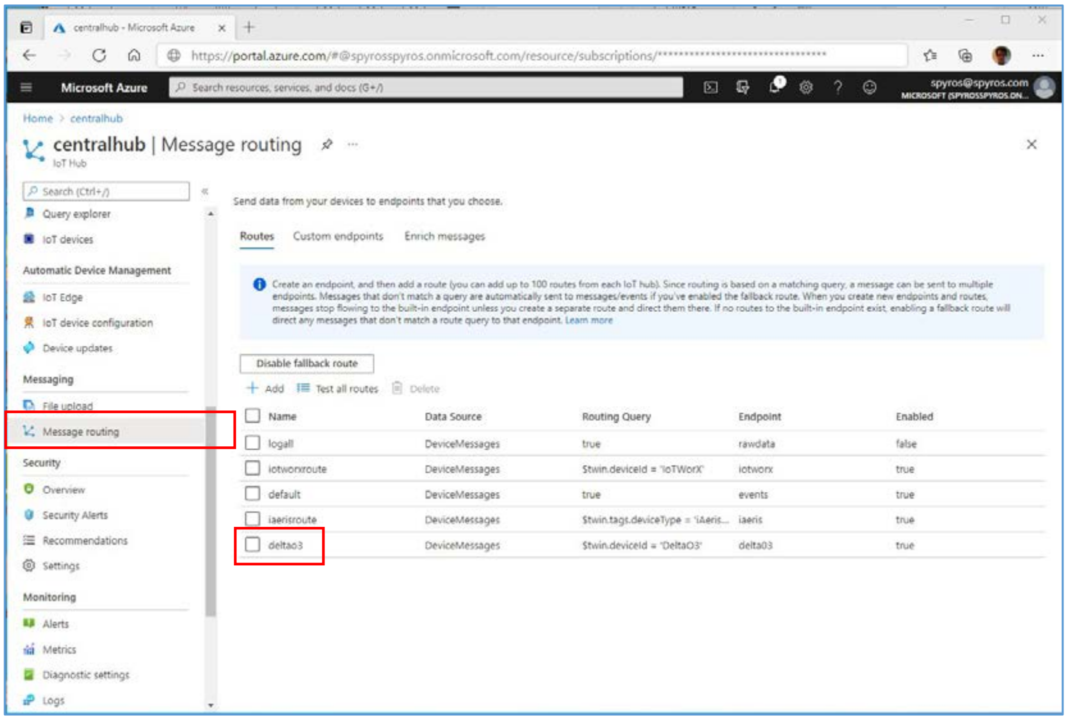
Event hub instance * : deltao3

Figure 21: Azure IoT Hub Custom Endpoints creation

Now we are ready to create new Route, select the **Routes** tab, and click **+ Add.** To create the environment used in this example, set the parameters as follows:

Parameter	Value
Name	Enter **deltao3**.
Endpoint	Click the down arrow and select **deltao3**.
Routing query	Enter **$twin.deviceId** = **'DeltaO3'**

The **Message routing** tab may look like this after the route has been added to the list of existing routes:



Send data from your devices to endpoints that you choose.

Create an endpoint, and then add a route (you can add up to 100 routes from each IoT hub). Since routing is based on a matching query, a message can be sent to multiple endpoints. Messages that don't match a query are automatically sent to messages/events if you've enabled the fallback route. When you create new endpoints and routes, messages stop flowing to the built-in endpoint unless you create a separate route and direct them there. If no routes to the built-in endpoint exist, enabling a fallback route will direct any messages that don't match a route query to that endpoint. Learn more

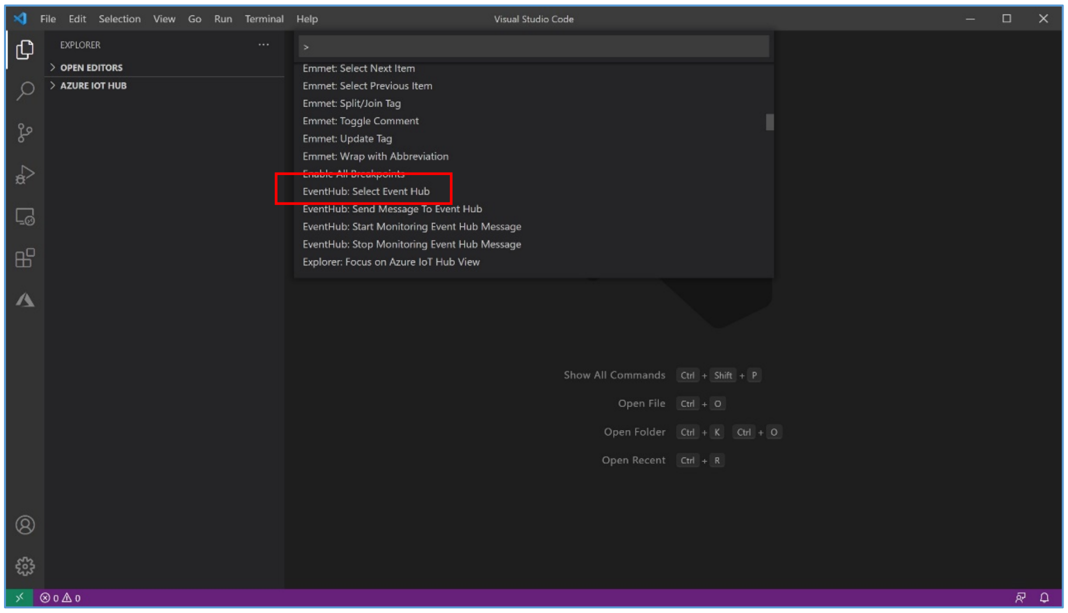
Name	Data Source	Routing Query	Endpoint	Enabled
logall	DeviceMessages	true	rawdata	false
iotworxroute	DeviceMessages	$twin.deviceId = 'IoTWorX'	iotworx	true
default	DeviceMessages	true	events	true
iaerisroute	DeviceMessages	$twin.tags.deviceType = 'iAeris...	iaeris	true
deltao3	DeviceMessages	$twin.deviceId = 'DeltaO3'	delta03	true

Figure 24: Message routes

5.3 *Viewing data received by Event Hub*

To monitor the data received from the IoT Hub by the Event Hub, we will use Microsoft Visual Studio. First download and install Visual Studio Code, then the Azure Event Hub Explorer. Open Visual Studio Code and follow these steps.

1. Select **View → Extensions → Azure Event Hub Explorer.**
2. Select **View → Command Palette → Event Hub: Select Event Hub**



```
Emmet: Select Next Item
Emmet: Select Previous Item
Emmet: Split/Join Tag
Emmet: Toggle Comment
Emmet: Update Tag
Emmet: Wrap with Abbreviation
Enable All Breakpoints
EventHub: Select Event Hub
EventHub: Send Message To Event Hub
EventHub: Start Monitoring Event Hub Message
EventHub: Stop Monitoring Event Hub Message
Explorer: Focus on Azure IoT Hub View
```

File Edit Selection View Go Run Terminal Help
Visual Studio Code
EXPLORER
OPEN EDITORS
AZURE IOT HUB
Show All Commands Ctrl + Shift + P
Open File Ctrl + O
Open Folder Ctrl + K Ctrl + O
Open Recent Ctrl + R

Figure 25: Select Event Hub

3. From the drop-down select subscription **Subscription-1**.
4. From the drop-down select resource group **iotprojects.**
5. From the drop-down select event hub namespace **centralhubs.**
6. From the drop-down select event hub **deltao3.**
7. From the top menu select **View → Command Palette → Event Hub: Start monitoring.**

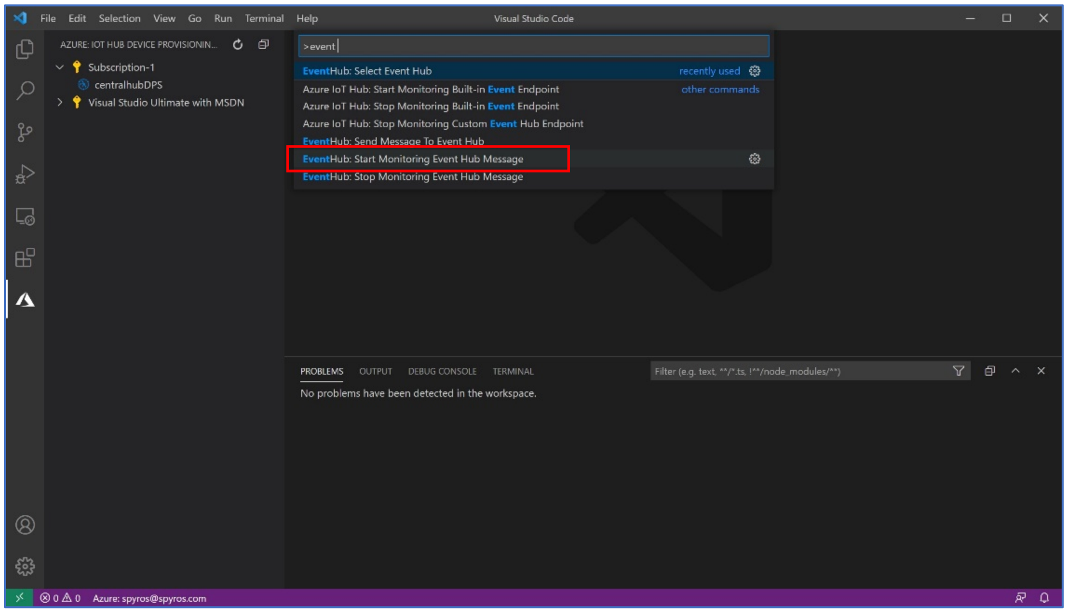
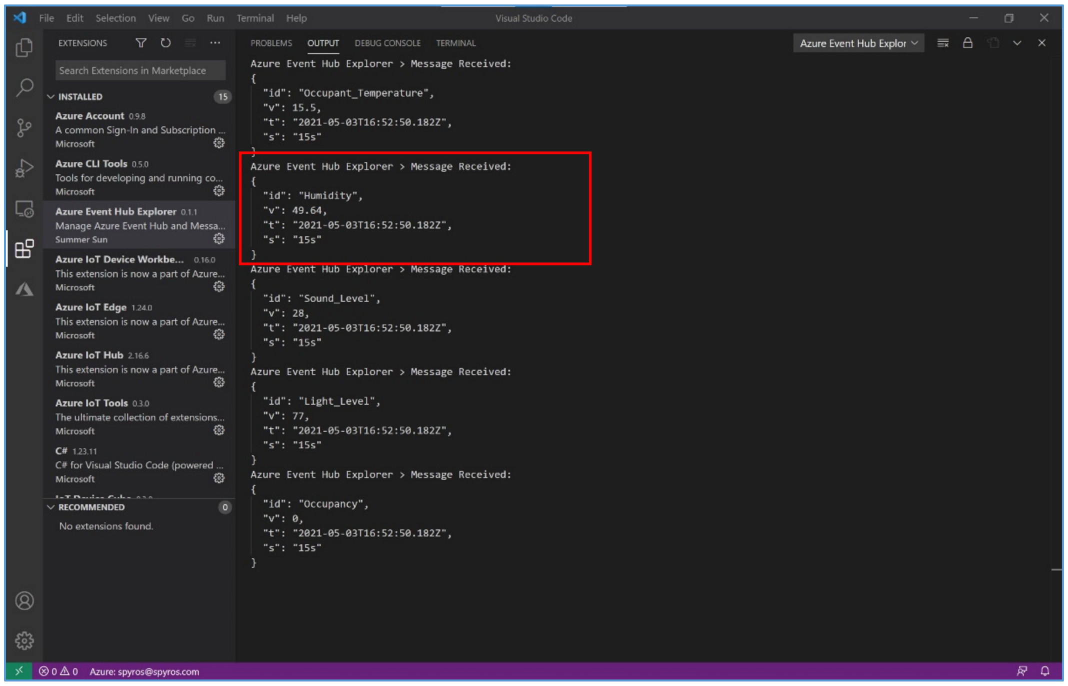

>event
EventHub: Select Event Hub
recently used
other commands
Azure IoT Hub: Start Monitoring Built-in Event Endpoint
Azure IoT Hub: Stop Monitoring Built-in Event Endpoint
Azure IoT Hub: Stop Monitoring Custom Event Hub Endpoint
EventHub: Send Message To Event Hub
EventHub: Start Monitoring Event Hub Message
EventHub: Stop Monitoring Event Hub Message
AZURE: IOT HUB DEVICE PROVISIONIN...
Subscription-1
centralhubDPS
Visual Studio Ultimate with MSDN
No problems have been detected in the workspace.
Azure: spyros@spyros.com

Figure 26: Start monitoring Event Hub

At this point, data should start appearing:



```
Azure Event Hub Explorer > Message Received:
{
  "id": "Occupant_Temperature",
  "v": 15.5,
  "t": "2021-05-03T16:52:50.182Z",
  "s": "15s"
}
Azure Event Hub Explorer > Message Received:
{
  "id": "Humidity",
  "v": 49.64,
  "t": "2021-05-03T16:52:50.182Z",
  "s": "15s"
}
Azure Event Hub Explorer > Message Received:
{
  "id": "Sound_Level",
  "v": 28,
  "t": "2021-05-03T16:52:50.182Z",
  "s": "15s"
}
Azure Event Hub Explorer > Message Received:
{
  "id": "Light_Level",
  "v": 77,
  "t": "2021-05-03T16:52:50.182Z",
  "s": "15s"
}
Azure Event Hub Explorer > Message Received:
{
  "id": "Occupancy",
  "v": 0,
  "t": "2021-05-03T16:52:50.182Z",
  "s": "15s"
}
```

Figure 27: Data arriving in Event Hub

In this screen capture we see the data collected by the O3 as it is received at the Event Hub.

6 Push data from Event Hub to Azure Table Storage

There are multiple ways to store the data streamed to IoT Hub. In a previous whitepaper, Monitoring Building Air Quality, we describe the steps to do this with an Azure Stream Analytics job writing to SQL Server. In the section below, we show how to do this by routing the data from the IoT Hub to an Event Hub and then writing it with an Azure Function to an Azure Table. This is more cost-efficient than using Azure Stream Analytics and SQL Server, though this way is more complex to set up and requires some coding skills. Depending upon the way you want to use the data, it may be adequate to store it in an Azure Table, but some analytical tools may require it being stored in SQL Server.

6.1 Creating the Function App

In the Azure portal select **+ Create a resource** and select the **Function App** category. To create the environment used in this example, on the **Basics** page set the parameters as follows:

Setting	**Value**
Subscription	Enter your Azure IoT subscription name. In our example, this is **Subscription-1**.
Resource Group	Enter **IoT_projects**.
Function App name	Enter Simple**DataEnrichment**.
Publish	Select **Code**.
Runtime stack	Select **Node.js.**
Version	Select **14 LTS**.
Region	Select the region where you have created the IoT Hub. In our example, this is **East US**.

Select **Next : Hosting**. On the **Hosting** page, accept the defaults then select **Next : Monitoring**. On the **Monitoring** page, turn off **Application Insights**. Finally, select **Review + Create**, then **Create** to deploy the function app.

6.2 Creating the Function

Next, we create a function. When the deployment is complete, select **Go To Resource**. From the left menu select **Functions**, then select **+ Add** from the top menu. In the **Add Function** window, set the parameters as follows:

Setting	Value
Develop environment	Select **Develop in portal.**
Template	Select **Azure Event Hub trigger.**
New Function	Enter **EventHub2Table**
Event Hub connection	Click New, then select **centralhubs** for **Event Hub connection, deltao3** for Event Hub connection, and click **OK**
Event Hub name	Enter **deltao3**
Event Hub consumer group	Enter **EventHub2Table.**

Click **Add** to create the function. Once created, click on **EventHub2Table** in the list on the right to open the function page. Click **Integration**, to bring up the wire frame:

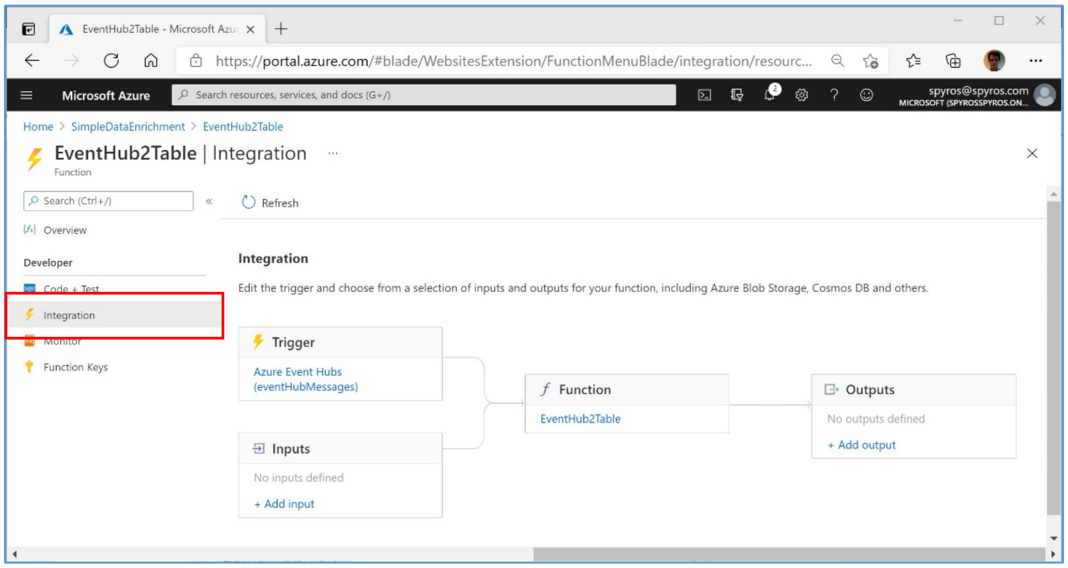

EventHub2Table - Microsoft Azure
https://portal.azure.com/#blade/WebsitesExtension/FunctionMenuBlade/integration/resourc...
Microsoft Azure
Search resources, services, and docs (G+/)
spyros@spyros.com
MICROSOFT (SPYROSSPYROS.ON...
Home > SimpleDataEnrichment > EventHub2Table
EventHub2Table | Integration
Function
Search (Ctrl+/)
Overview
Developer
Code + Test
Integration
Monitor
Function Keys
Refresh
Integration
Edit the trigger and choose from a selection of inputs and outputs for your function, including Azure Blob Storage, Cosmos DB and others.
Trigger
Azure Event Hubs (eventHubMessages)
Inputs
No inputs defined
+ Add input
Function
EventHub2Table
Outputs
No outputs defined
+ Add output

Figure 28: Integration wire frame

Next, click **Code + Test** in the left menu and select function.json in the drop-down at the top. The JSON should contain the information from the **Create Function** wizard:

```
{
  "bindings": [
    {
      "type": "eventHubTrigger",
      "name": "eventHubMessages",
      "direction": "in",
      "eventHubName": "delta03",
      "connection": "centralhubs_RootManageSharedAccessKey_EVENTHUB",
      "cardinality": "many",
      "consumerGroup": "eventhub2table"
    }
  ]
}
```

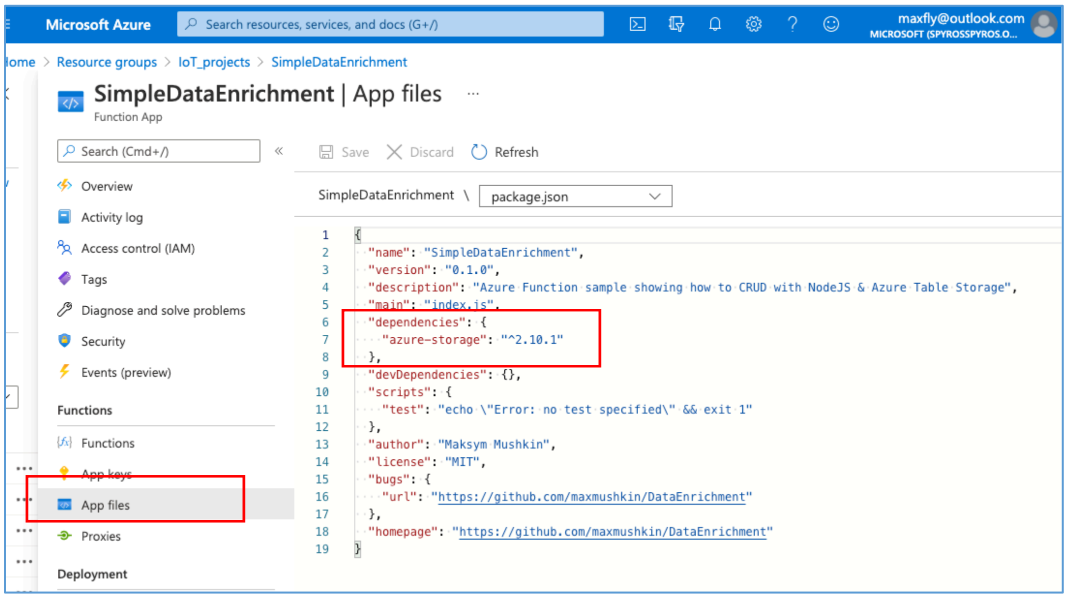
Next, select **index.js** in the drop-down at the top, and replace the code in the window with the following and click **Save** (formatting below modified to fit to page):

```
const azure = require('azure-storage');

// App Settings should have variables AZURE_STORAGE_ACCOUNT and AZURE_STORAGE_ACCESS_KEY, or
// AZURE_STORAGE_CONNECTION_STRING
const tableService = azure.createTableService();
var tableName = process.env["OutputTableName"] || "OutputTable";
var tableUpdateInterval = process.env["TableUpdateInterval"] || 5;
module.exports = async function (context, eventHubMessages) {
    // Create Table if not exists
    tableService.createTableIfNotExists(tableName, function (error, result, response) {
        if (error) {
            context.log.warn(error);
        }
    });
    var roundCoef = 60 * tableUpdateInterval; // number of seconds in {tableUpdateInterval} minutes
    var updateTasks = {};
    eventHubMessages.forEach((message, index) => {
        // Extract partition key(device location and name) from the IotHub Enriched properties taken from
        // IoTHub device twin
        var deviceLocation = context.bindingData.propertiesArray[index].deviceLocation;
        var deviceName = context.bindingData.propertiesArray[index].deviceName;
        var partitionKey = deviceLocation + '-' + deviceName;
        // Convert datetime to unix timestamp and round it
        var unixTime = Math.round(new Date(message.t).getTime() / 1000);
        var rowKey = (Math.floor(unixTime / roundCoef) * roundCoef).toString();
        // Check if Update Task already exists for this row, otherwise create
        if(updateTasks[partitionKey+rowKey] == null)
            {
                updateTasks[partitionKey+rowKey] =
                {
                    PartitionKey: partitionKey,
                    RowKey: rowKey
                };
            }
        // Add new property to Update Task
        updateTasks[partitionKey+rowKey][message.id] = message.v;
    });

(Continued on next page)
// Proceed with all Insert/Update operations
    for(var key in updateTasks){
        var updateTask = updateTasks[key];
        context.log(updateTask);
        // Create new row or update exisitng
        tableService.insertOrMergeEntity(tableName, updateTask, function(error, result, response){
            if(error){
                context.log.warn(error);
            }
        });
    }
};
```

Finally, we need to edit the file **package.json**, which you can access from the **App files** section in the portal, and add a reference to Azure storage there. In the portal:


Microsoft Azure
Search resources, services, and docs (G+/)
maxfly@outlook.com
MICROSOFT (SPYROSSPYROS.O...
Home > Resource groups > IoT_projects > SimpleDataEnrichment
SimpleDataEnrichment | App files
Function App
Search (Cmd+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Security
Events (preview)
Functions
Functions
App keys
App files
Proxies
Deployment
Save
Discard
Refresh
SimpleDataEnrichment \ package.json

```
{
  "name": "SimpleDataEnrichment",
  "version": "0.1.0",
  "description": "Azure Function sample showing how to CRUD with NodeJS & Azure Table Storage",
  "main": "index.js",
  "dependencies": {
    "azure-storage": "^2.10.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Maksym Mushkin",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/maxmushkin/DataEnrichment"
  },
  "homepage": "https://github.com/maxmushkin/DataEnrichment"
}
```

Figure 29: Azure Function App files

The section that needs to be added there is this:

```
"dependencies": {
    "azure-storage": "^2.10.1"
  }
```

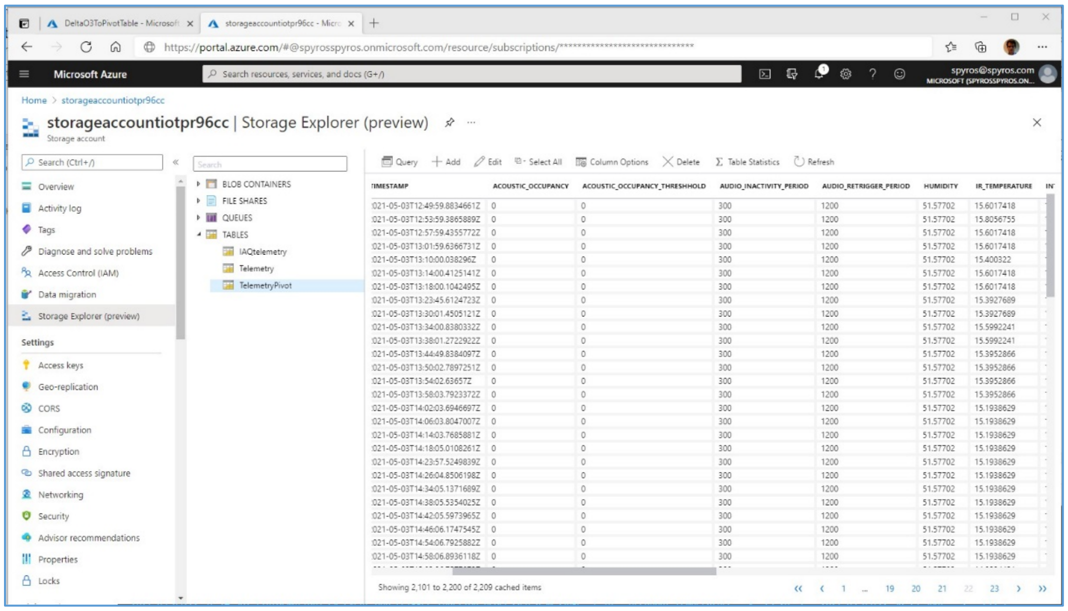
It tells application runtime to load the azure-storage **npm** module since it will be used in the source code to access Azure Table Storage. Without this section the code will not import the Azure Storage module and will raise an exception for the lines bellow:

```
const azure = require('azure-storage');

// App Settings should have variables AZURE_STORAGE_ACCOUNT and
AZURE_STORAGE_ACCESS_KEY, or AZURE_STORAGE_CONNECTION_STRING
const tableService = azure.createTableService();
```

6.3 Viewing data received by Azure Table Storage

To verify that the function is working correctly, from the Azure portal select **storageaccountiotpr96cc**, then from the left menu select **Storage Explorer (preview) → TABLES → TelemetryPivot**. This should show data in the **TelemetryPivot** table specified in **Function.json:**



storageaccountiotpr96cc | Storage Explorer (preview)

IMESTAMP	ACOUSTIC_OCCUPANCY	ACOUSTIC_OCCUPANCY_THRESHHOLD	AUDIO_INACTIVITY_PERIOD	AUDIO_RETRIGGER_PERIOD	HUMIDITY	IR_TEMPERATURE
021-05-03T12:49:59.8834661Z	0	0	300	1200	51.57702	15.6017418
021-05-03T12:53:59.3865889Z	0	0	300	1200	51.57702	15.8056755
021-05-03T12:57:59.4355772Z	0	0	300	1200	51.57702	15.6017418
021-05-03T13:01:59.6366731Z	0	0	300	1200	51.57702	15.6017418
021-05-03T13:10:00.038296Z	0	0	300	1200	51.57702	15.400322
021-05-03T13:14:00.4125141Z	0	0	300	1200	51.57702	15.6017418
021-05-03T13:18:00.1042495Z	0	0	300	1200	51.57702	15.6017418
021-05-03T13:23:45.6124723Z	0	0	300	1200	51.57702	15.3927689
021-05-03T13:30:01.4505121Z	0	0	300	1200	51.57702	15.3927689
021-05-03T13:34:00.8380332Z	0	0	300	1200	51.57702	15.5992241
021-05-03T13:38:01.2722922Z	0	0	300	1200	51.57702	15.5992241
021-05-03T13:44:49.8384097Z	0	0	300	1200	51.57702	15.3952866
021-05-03T13:50:02.7897251Z	0	0	300	1200	51.57702	15.3952866
021-05-03T13:54:02.63657Z	0	0	300	1200	51.57702	15.3952866
021-05-03T13:58:03.7923372Z	0	0	300	1200	51.57702	15.3952866
021-05-03T14:02:03.6946697Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:06:03.8047007Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:14:03.7685881Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:18:05.0108261Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:23:57.5249839Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:26:04.8506198Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:34:05.1371689Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:38:05.5354025Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:42:05.5973965Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:46:06.1747545Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:54:06.7925882Z	0	0	300	1200	51.57702	15.1938629
021-05-03T14:58:06.8936118Z	0	0	300	1200	51.57702	15.1938629

Showing 2,101 to 2,200 of 2,209 cached items

Figure 30: Storage Explorer showing data in TelemetryPivot table

7 Next steps

If you have successfully completed the above steps, you have a working end-to-end example of pushing data from the O3 Edge to Azure. Now you can build various monitoring dashboards with tools like Power BI, Time Series Insights, Node-RED , and others. Here is an example of a Power BI report we created using the data from the above flow:

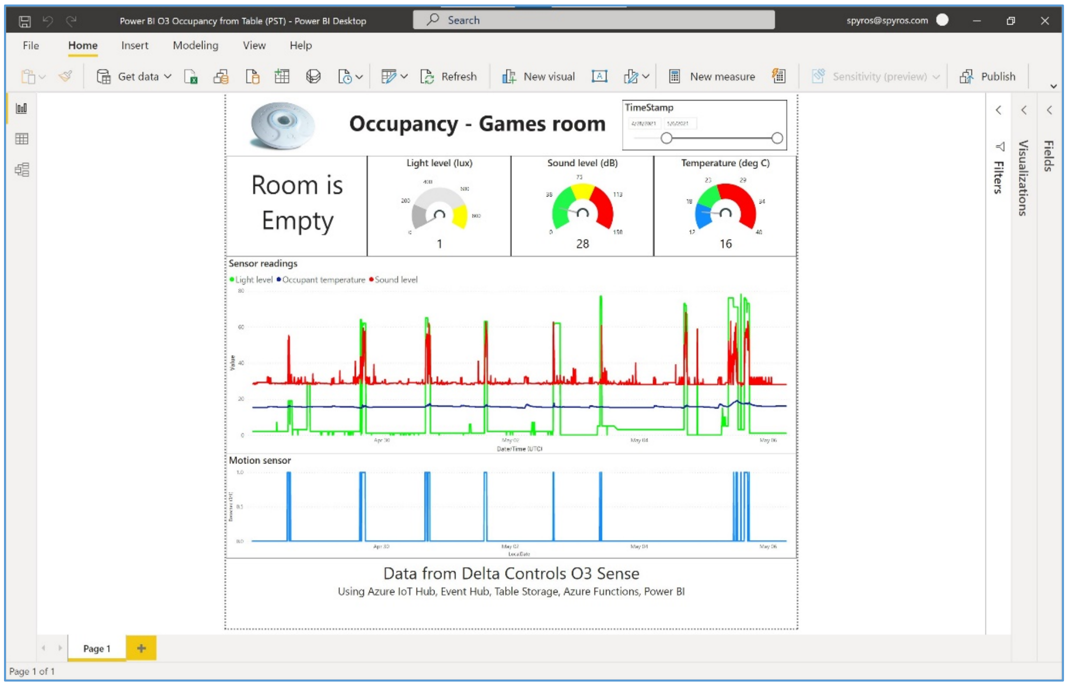


Figure 31: O3 Edge data in Power BI desktop

The O3 Edge is in a room we call the **Games** room, containing some exercise equipment and an Xbox console. This report spans only a few days, but there are still a few interesting observations we can make.

First, from the light trend line, we see that the light in the room is turned on only for about an hour a day. This happens to be when my son uses the room to exercise, and the sound level trend line shows that he turns on music while he is doing so. On the last day shown in the trend, he and some friends were playing an online game on the Xbox, so the light, sound, and motion trends remain on for longer.

A second observation that we can make is from the light trend. Note that the intensity of the light is higher on the three days at the end than the preceding days. This is because the weather on the preceding days was gloomy and overcast, and there was little light coming in through the windows, whereas on the three days at the end there was bright sunlight. Not an earth-shattering observation, but still interesting to see how that is reflected in the O3 Edge data!