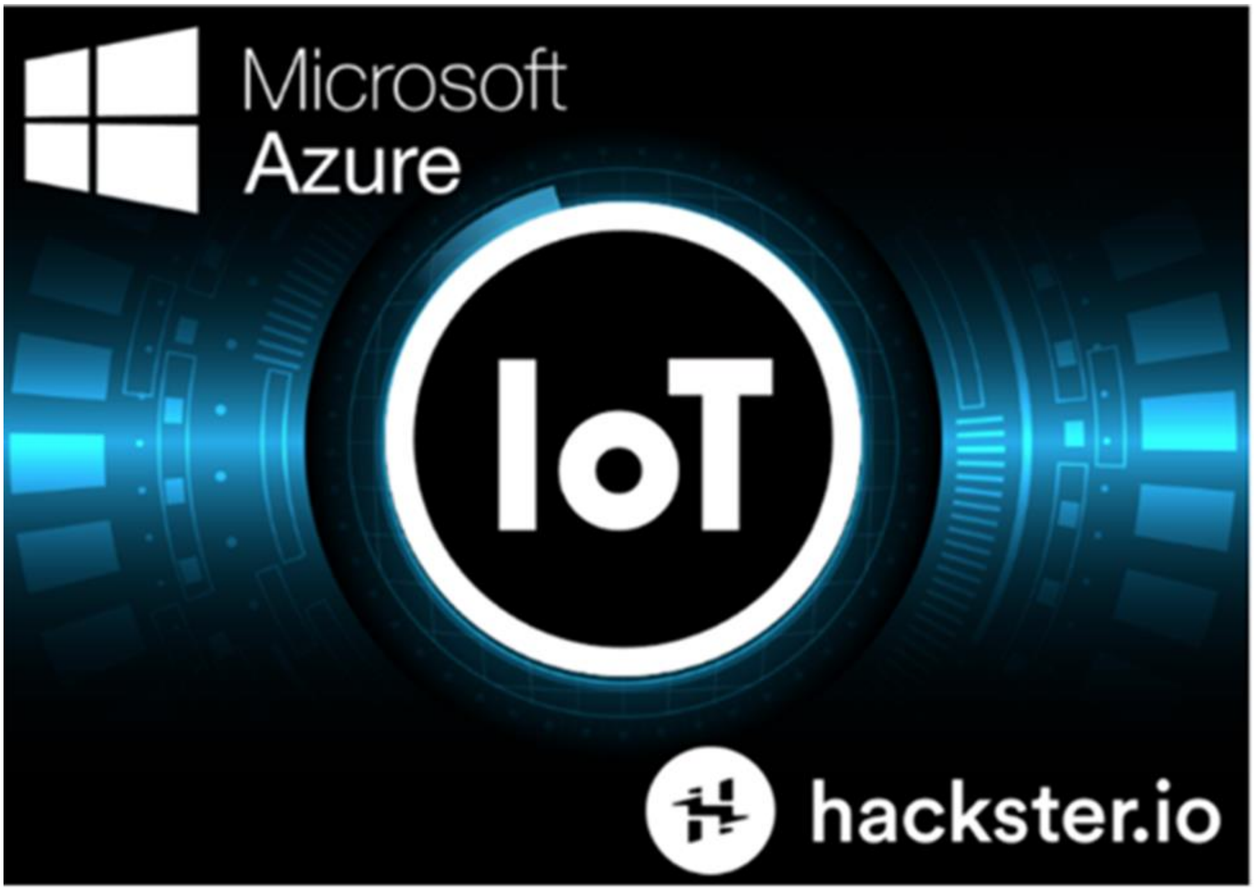


Azure IoT for the Enterprise Maker

Lab Manual

Final Version

Abstract: This workshop is part of the joint Microsoft and Hackster LIVE event series. This workshop introduces the "enterprise maker" to the Azure IoT portfolio and provides a hands on experience with the Azure IoT Hub, Azure Functions, Azure Web Apps and Power BI. At the end of this workshop, the student will have created a simple, end-to-end IoT platofrm

Author: Kent Stroker (stkent@microsoft.com)

Version: MIHL-2017-2.0

Date: 18 June 2017

Microsoft

CONTENTS

1 Lab One - Azure IoT Hub

This first lab is to create an Azure IoT Hub and two devices to send data to the hub.

WARNING: All the illustrations in this document are to only illustrate the screen you should be working with – DO NOT FOLLOW illustrations exactly! Your entry to various fields WILL BE DIFFERENT. For the most part, accept the defaults in all screens during this lab, if it not explicitly called out to change a field, do not.

1.1 Azure Portal Login

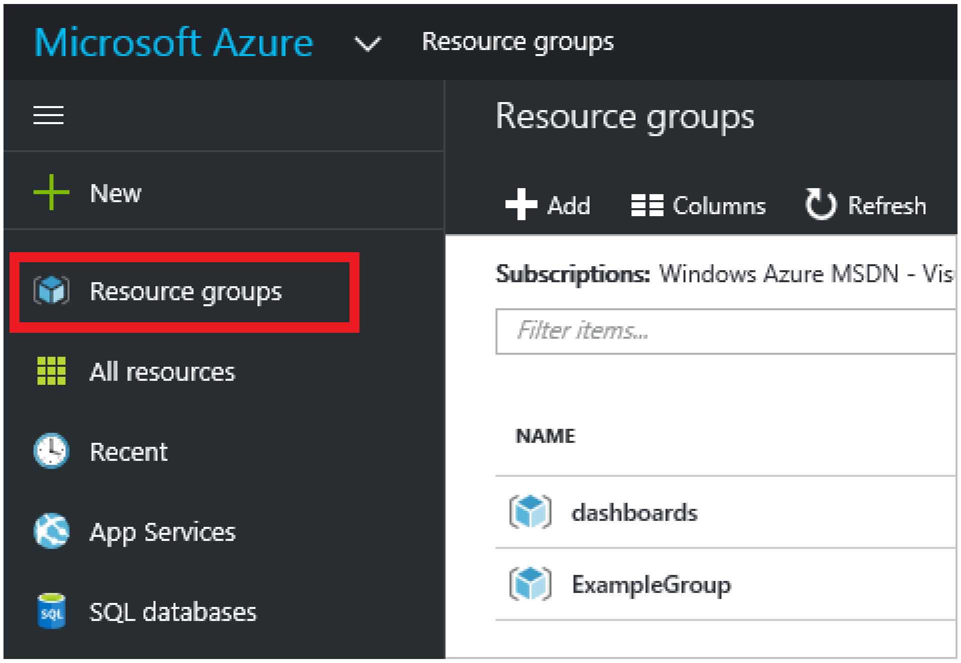
Start by logging into your Azure account.

1. Open a browser and navigate to **azure.microsoft.com**
2. Login to your account **PORTAL**

1.2 Create an Empty Resource Group

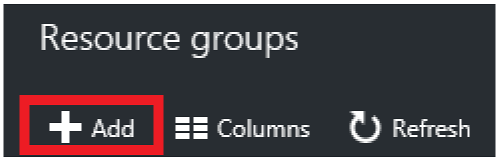
A resource group is a container that holds related resources for an Azure solution.

1. Click on **Resource groups**

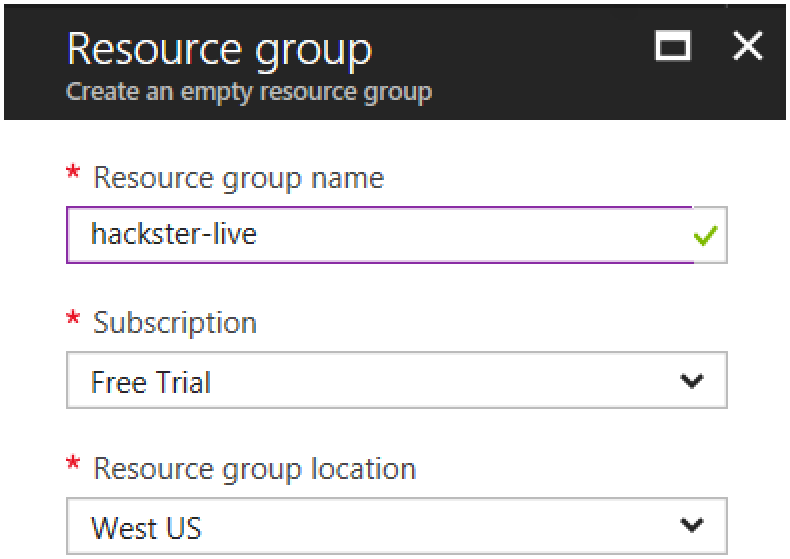


Subscriptions: Windows Azure MSDN - Vis

2. To create a new resource group, select **+ Add**.



3. **Resource group name**: Please the name **hackster-live** for this workshop
4. **Subscription**: (Free Trial) May vary depending on your account, Free is typical
5. **Resource group location**: Please only **West US** or **East US** for this workshop



* Resource group name
hackster-live ✓

* Subscription
Free Trial

* Resource group location
West US

6. Check **Pin to dashboard**
7. Click **Create** button

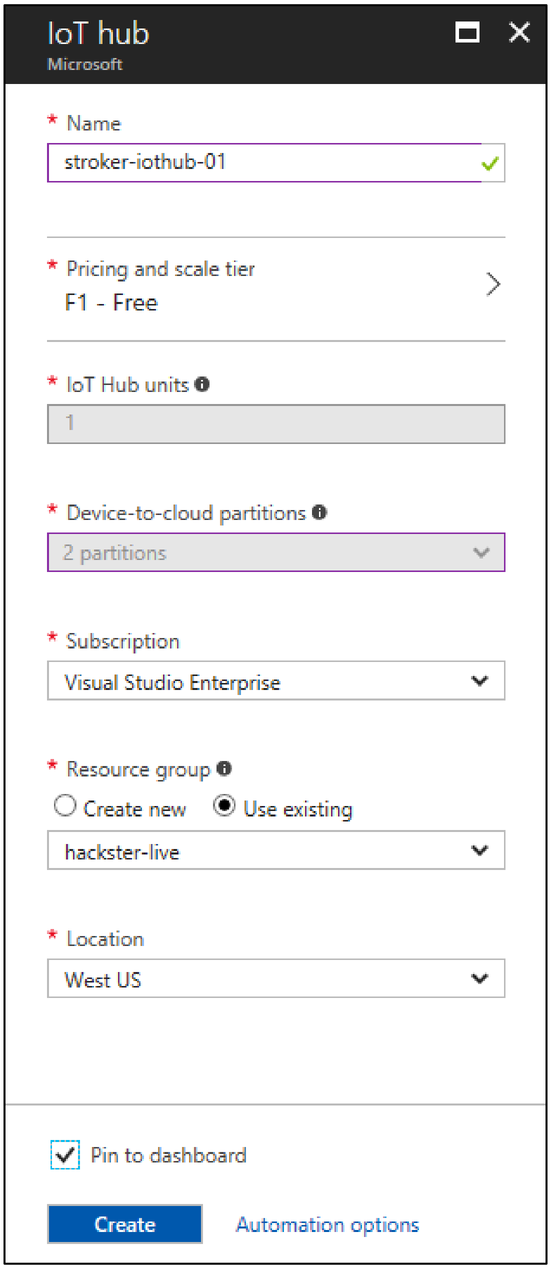
The resource group hackster-live is typically created in under a minute and then the resource group overview screen opens.

1.3 Azure IoT Hub Service

This step describes how to find the IoT Hub service in the Azure portal, and how to create and manage IoT hubs.

1.3.1 Create an Azure IoT Hub

1. **New > Internet of Things** > **IoT Hub**
2. **Name**: A unique name is required (e.g., **{lastName}-iothub-01**)
3. **Pricing and scale tier**: (Free) The default is the S1 – Standard, change to F1 - Free



IoT hub
Microsoft

* Name
stroker-iothub-01

* Pricing and scale tier
F1 - Free

* IoT Hub units
1

* Device-to-cloud partitions
2 partitions

* Subscription
Visual Studio Enterprise

* Resource group
○ Create new ● Use existing
hackster-live

* Location
West US

☑ Pin to dashboard

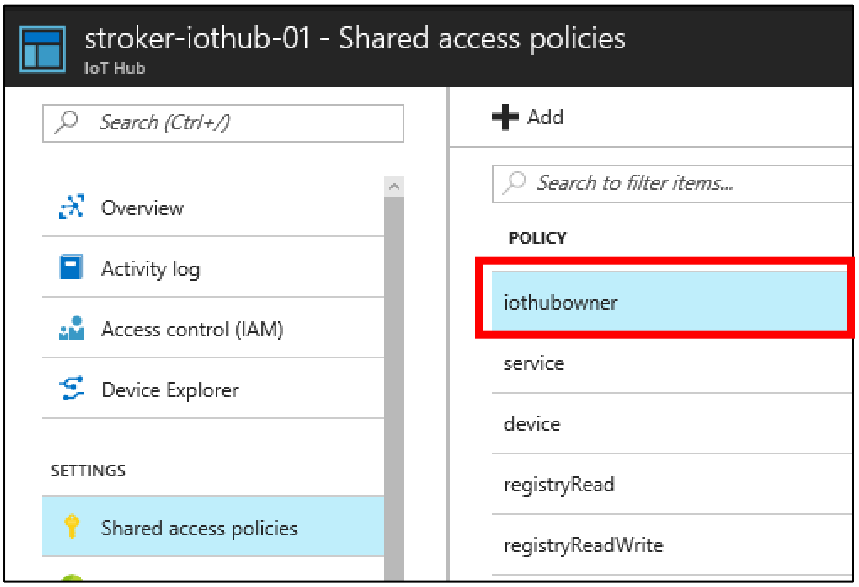
4. **IoT hub units**: (default, 1) Accept the default
5. **Device to cloud partitions and resource group:** (default, 2 or 4) Accept the default
6. **Choose subscriptions:** (Free Trial)
7. **Resource group:** Select **Use existing > hackster-live**
8. **Choose the location:** All services today need to be in the same region as you choose for the resource group for today's workshop
9. **Pin to dashboard**: Check this option
10. Click **Create** button.

The IoT Hub service is deployed, this typically takes 2-3 minutes. The overview screen for the IoT Hib service appears automatically after the deployed is completed and successful.

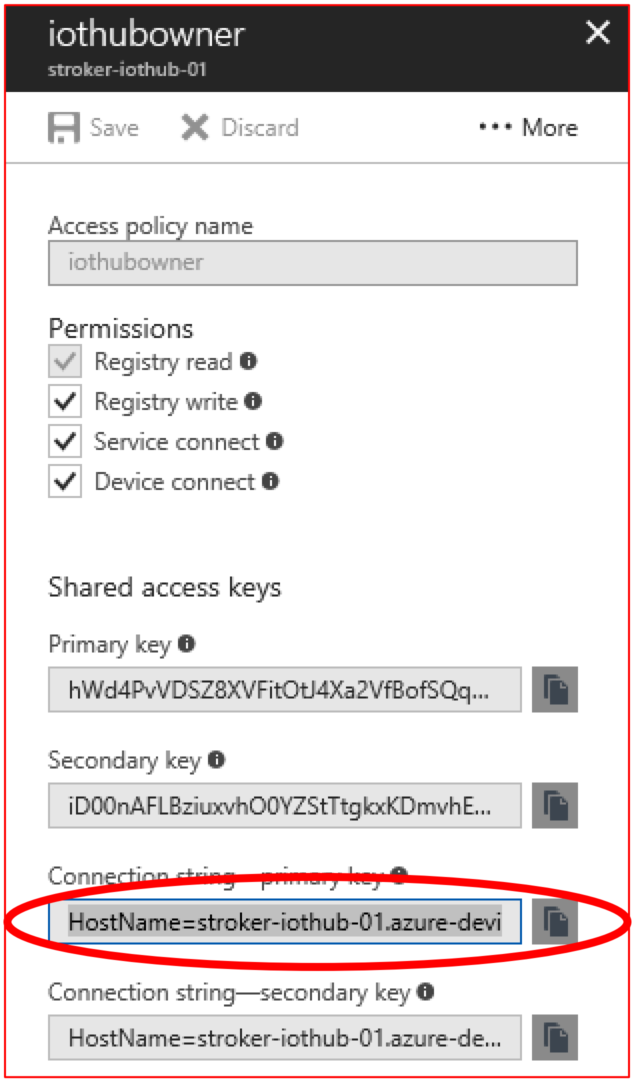
1.3.2 IoT Hub Credentials

There are several credentials you will need to paste into other files during this lab.

1. The scrollable panel to the left contains all the options for configuring the IoT Hub service
2. Click on **SETTING > Shared access policies**
3. Select **iothubowner**



4. A new blade opens on the right of the screen, click on the icon to copy the **Connection string – primary key** credentials



Access policy name
iothubowner

Permissions
☑ Registry read
☑ Registry write
☑ Service connect
☑ Device connect

Shared access keys

Primary key
hWd4PvVDSZ8XVFitOtJ4Xa2VfBofSQq...

Secondary key
iD00nAFLBziuxvhO0YZStTtgkxKDmvhE...

Connection string—primary key
HostName=stroker-iothub-01.azure-devi

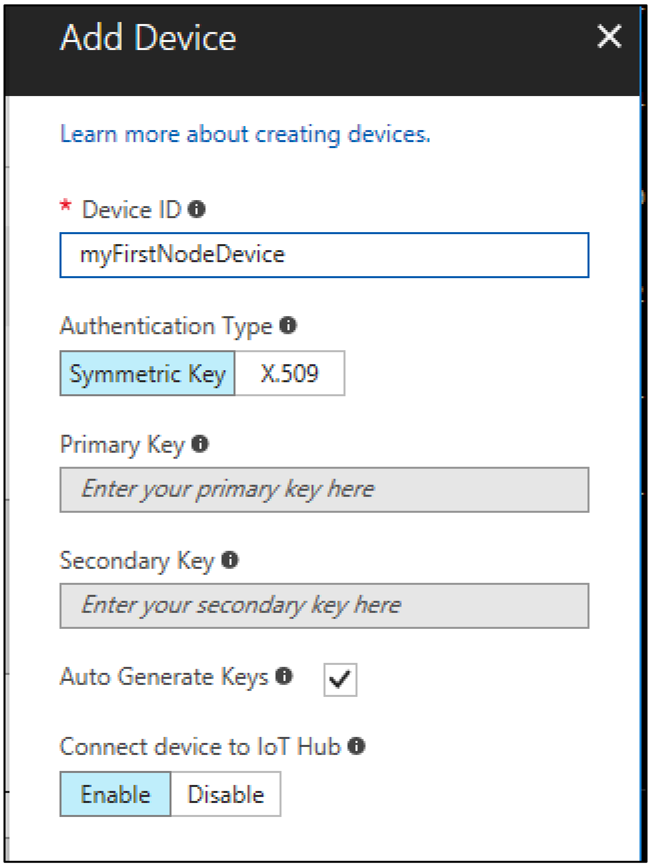
Connection string—secondary key
HostName=stroker-iothub-01.azure-de...

5. Open the **Credentials.txt** file, this file is used to hold all of the various keys you need for this workshop.
6. Paste the connection string into the **Credentials.txt** file, leave this file open.

1.3.3 Create Device Entry for myFirstNodeDevice

Use the Device Explorer option to view, create, update and delete devices on the IoT Hub.

1. Click on **Device Explorer**
2. Click on the **+Add** to add a new device; a blade opens on the right side of the screen
3. **Device ID**: (myFirstNodeDevice) Enter this name exactly as **myFirstNodeDevice**, and yes, the upper- and lower-case are important
4. All other fields are left to the default settings



Add Device

Learn more about creating devices.

* Device ID
myFirstNodeDevice

Authentication Type
[✓ Symmetric Key] X.509

Primary Key
Enter your primary key here

Secondary Key
Enter your secondary key here

Auto Generate Keys ☑

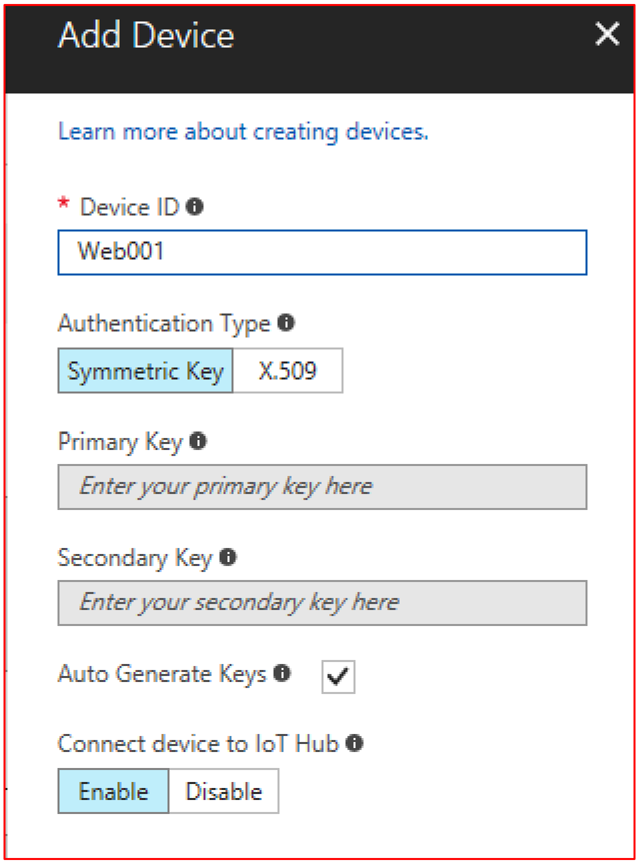
Connect device to IoT Hub
[✓ Enable] Disable

5. Click **SAVE** button

1.3.4 Create Device Entry for web01

Use the Device Explorer option to view, create, update and delete devices on the IoT Hub.

1. Click on the **+Add** to add new devices and credentials
2. **Device ID**: (web01) Enter this name exactly as **web01**
3. All other fields are left to the default settings
4. Click **SAVE** button



Add Device

Learn more about creating devices.

* Device ID: Web001

Authentication Type: [✓ Symmetric Key] X.509

Primary Key: *Enter your primary key here*

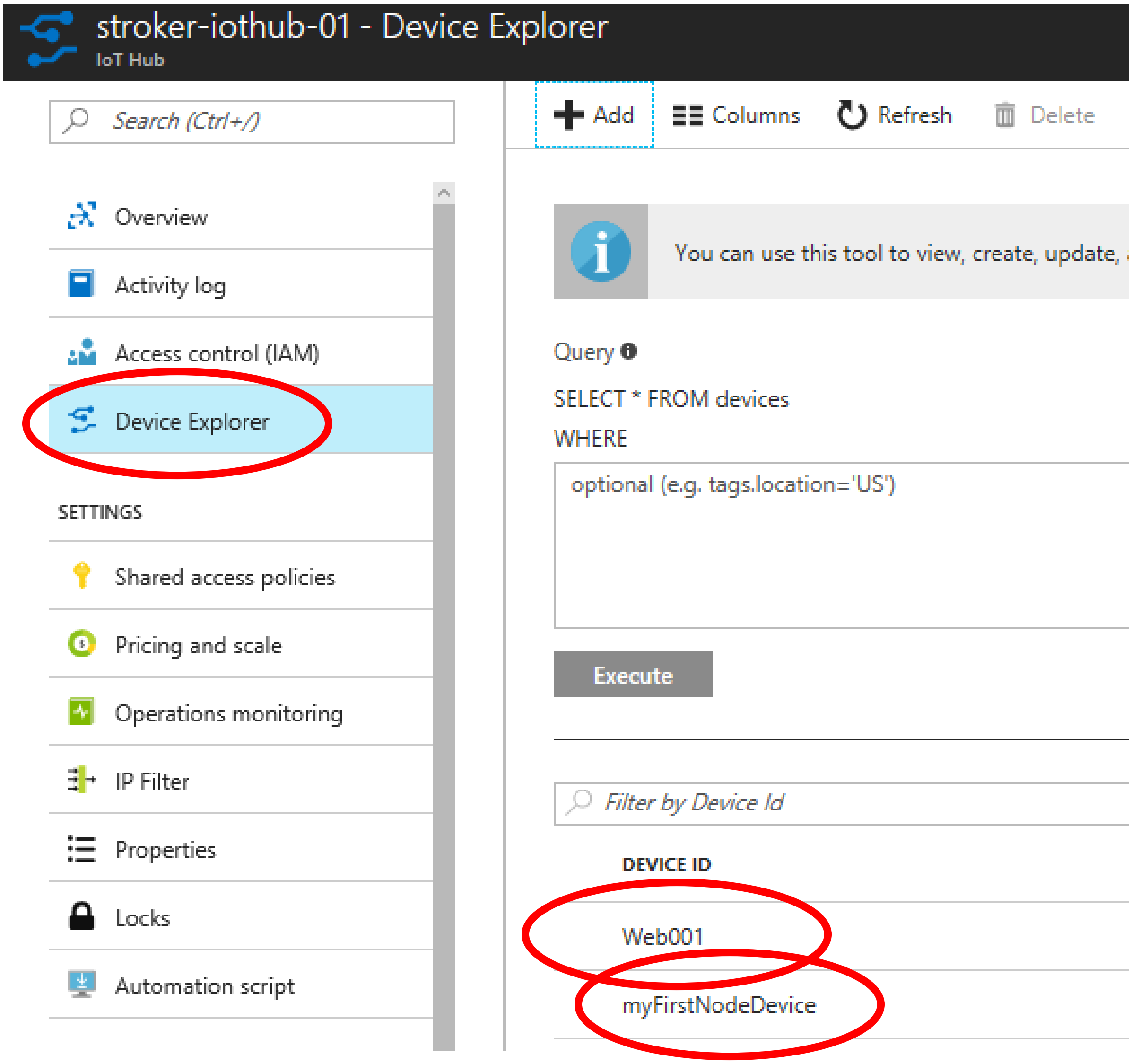
Secondary Key: *Enter your secondary key here*

Auto Generate Keys ☑

Connect device to IoT Hub: [✓ Enable] Disable

1.3.5 Gather the Devices Credentials

Click on each device entry and copy and paste the **Connection string – primary key** credentials to the **Credential.txt** file you still have open.



1.4 LAPTOP JAVASCRIPT SIMULATED DEVICE

It is easy to make any computer with Node.js (JavaScript) into an "IoT device" and that is exactly what you will do in this portion of the lab. The code is already written for you and all that is required is that you edit the connection strings and then run the programs.

There are two (2) Node applications:

- **ReadDeviceToCloudMessages.js**, which displays the telemetry sent by your simulated devices from the IoT Hub; this reads the IoT Hub in real-time
- **SimulatedDevice.js**, which connects to your IoT hub with the device identity created earlier, and sends a telemetry message every ten (10) seconds using the MQTT protocol

1.4.1 Read Messages from Cloud (READER)

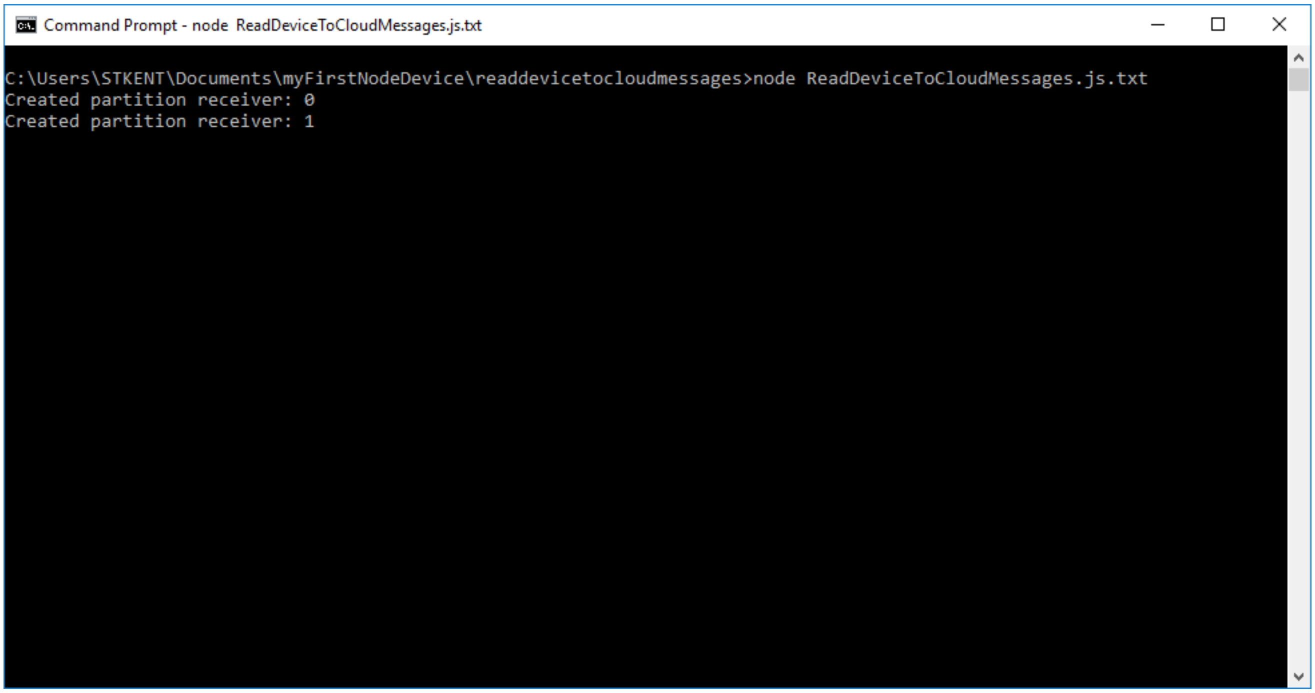
In this step, you configure a Node.js console app that reads device-to-cloud messages from IoT Hub. An IoT hub exposes an Event Hub-compatible endpoint to enable you to read device-to-cloud messages. To keep things simple, this tutorial creates a basic reader that is not suitable for a high throughput deployment.

1.4.1.1 Configure ReadToDeviceCloud.js

1. Open the **ReadDeviceToCloudMessages.js** file for editing.
    a. Find this in your lab download desktop folder
    b. Desktop > {lab folder} > lab1 > reddevicetocloudmessages
2. Open the file for editing using WordPad (hint: right-click the filename)
3. Edit the `var connectionString`
    a. the IoT Hub connect string you pasted into the **Credentials.txt** file earlier
    b. CAUTION – Make sure that the string is enclosed by single-quotes and the line ends with a semi-colon making sure no extra line returns have been entered into the string
4. Save and close the **ReadDeviceToCloudMessages.js** file.

1.4.1.2 Launch the ReadtoDeviceCloud application

1. Launch the Node JS application by clicking on the **startReader.bat** file
2. Nothing will display after the initialization messages; nothing is yet sending messages to the IoTHub, so nothing to see yet.
3. Leave this open for remainder of the lab, if accidentally closed, just restart.



```
C:\Users\STKENT\Documents\myFirstNodeDevice\readdevicetocloudmessages>node ReadDeviceToCloudMessages.js.txt
Created partition receiver: 0
Created partition receiver: 1
```

1.4.2 Create a Simulated Device (DEVICE)

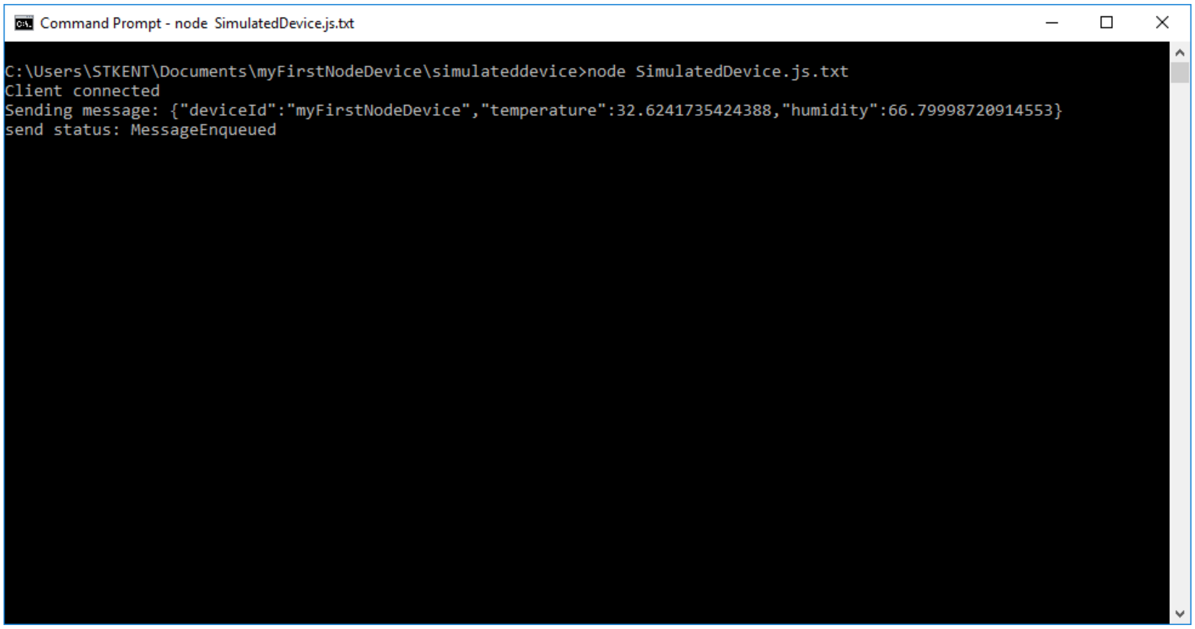
This application simulates a device that sends device-to-cloud (D2C) messages to the IoT Hub.

1.4.2.1 Configure SimulatedDevice.js

1. Open the **SimulatedDevice.js** file for editing.
    a. Find this in your IoT-Live-2017 desktop folder
    b. Desktop > {lab folder} > lab1 > simulateddevice
2. Open the file for editing using WordPad (hint: right-click the filename)
3. Edit the `var connectionString`
    a. the IoT Hub connect string for myFirstNodeDevice you pasted into the **Credentials.txt** file earlier
    b. CAUTION – Make sure that the string is enclosed by single-quotes and the line ends with a semi-colon making sure no extra line returns have been entered into the string
4. Save and close the **SimulatedDevice.js** file.

1.4.2.2 Launch the Simulated Device application

1. Launch the Node JS application by clicking on the **startDevice.bat** file
2. Every ten (10) seconds, a line of data is transmitted
3. Leave this open for remainder of the lab, if accidentally closed, just restart.



```
C:\Users\STKENT\Documents\myFirstNodeDevice\simulateddevice>node SimulatedDevice.js.txt
Client connected
Sending message: {"deviceId":"myFirstNodeDevice","temperature":32.6241735424388,"humidity":66.79998720914553}
send status: MessageEnqueued
```

1.4.3 Confirm Data Received at IoT Hub

Look at the command prompt window that holds the ReadDeviceToCloudMessage. You should now see the message confirmation appear each time the Simulated Device sends a message.

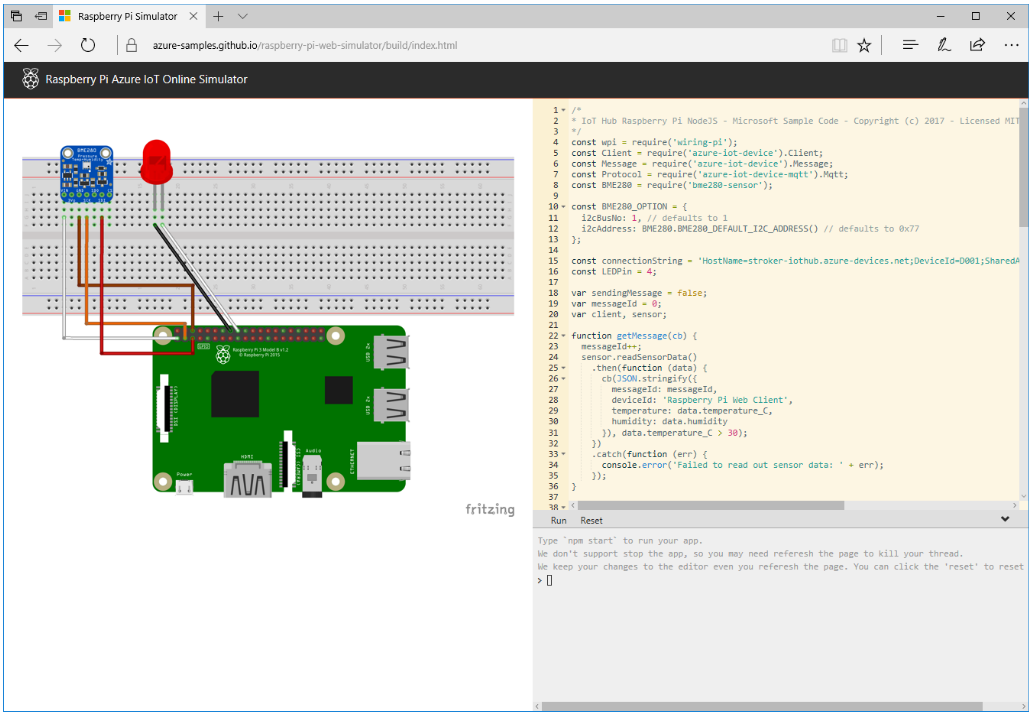
1.5 Raspberry Pi Web Simulator Device (web01)

This is awesome and a totally cool tool. A virtual Raspberry Pi running Azure IoT device code online as a web application. An open source project written in JavaScript and available from GitHub:

> https://azure-samples.github.io/raspberry-pi-web-simulator/
>
> NOTE: Use the Chrome browser, this is a preview version and it is not entirely happy with Edge right now, so please use Chrome. I know, I know…

1.5.1 Setup and Run web01

1. Open a Chrome browser session and navigate to: https://azure-samples.github.io/raspberry-pi-web-simulator
2. Click on the **Demo page** link to launch the simulator


Raspberry Pi Simulator
azure-samples.github.io/raspberry-pi-web-simulator/build/index.html
Raspberry Pi Azure IoT Online Simulator
fritzing
Run
Reset

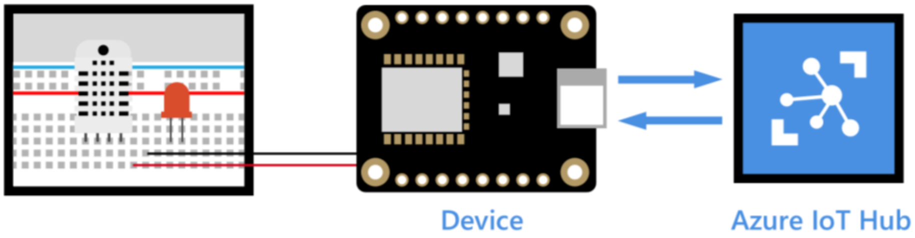
```
1  /*
2  * IoT Hub Raspberry Pi NodeJS - Microsoft Sample Code - Copyright (c) 2017 - Licensed MIT
3  */
4  const wpi = require('wiring-pi');
5  const Client = require('azure-iot-device').Client;
6  const Message = require('azure-iot-device').Message;
7  const Protocol = require('azure-iot-device-mqtt').Mqtt;
8  const BME280 = require('bme280-sensor');
9
10 const BME280_OPTION = {
11   i2cBusNo: 1, // defaults to 1
12   i2cAddress: BME280.BME280_DEFAULT_I2C_ADDRESS() // defaults to 0x77
13 };
14
15 const connectionString = 'HostName=stroker-iothub.azure-devices.net;DeviceId=D001;SharedA
16 const LEDPin = 4;
17
18 var sendingMessage = false;
19 var messageId = 0;
20 var client, sensor;
21
22 function getMessage(cb) {
23   messageId++;
24   sensor.readSensorData()
25     .then(function (data) {
26       cb(JSON.stringify({
27         messageId: messageId,
28         deviceId: 'Raspberry Pi Web Client',
29         temperature: data.temperature_C,
30         humidity: data.humidity
31       }), data.temperature_C > 30);
32     })
33     .catch(function (err) {
34       console.error('Failed to read out sensor data: ' + err);
35     });
36 }
37
```

```
Type `npm start` to run your app.
We don't support stop the app, so you may need referesh the page to kill your thread.
We keep your changes to the editor even you referesh the page. You can click the 'reset' to reset
> ▯
```

3. Be default, the code in this simulator sends a data payload every second; this will overrun the IoT Hub Free level, make the following edit
   a. Edit Line 119 changing the 2000 to 10000
4. Replace the connection string on Line 15 with the web01 device connection string, and type `npm start` in the console to run; or click the Run button
5. You should now see the second device messages in the ReadDevicetoCloudMessage console screen
6. Leave the simulator running for remainder of the lab

1.6 IoT Hub Explorer

A command line interface (CLI) tool to manage device identities in your IoT hub registry, send and receive messages and files from your devices, and monitor your IoT hub operations. Also lets you simulate a device connected to your IoT hub.



Full source and documentation is at GitHub URL https://github.com/azure/iothub-explorer.

To install the latest version of the **iothub-explorer** tool, run the following command in your command-line environment:

1. Open a command prompt
2. `npm install -g iothub-explorer`

You can use the following command to get additional help about all the **iothub-explorer** commands:

```
iothub-explorer --version

iothub-explorer help
```

1.6.1 Login

Supply your IoT hub connection string once using the **login** command. This means you do not need to supply the connection string for subsequent commands for the duration of the session (defaults to one hour):

```
iothub-explorer login "HostName=<my-hub>.azure-
devices.net;SharedAccessKeyName=<my-policy>;SharedAccessKey=<my-policy-key>"
```

NOTE: Use the Iot Hub credentials, not the device endpoint credentials.

1.6.2 Device-to-Cloud (D2C) Example

To retrieve information about an already-registered device from the device identity registry in your IoT hub, including the device connection string, use the following command:

```
iothub-explorer get web01 --connection-string

iothub-explorer get myFirstNodeDevice --connection-string

iothub-explorer list
```

NOTE: Use the device endpoint credentials.

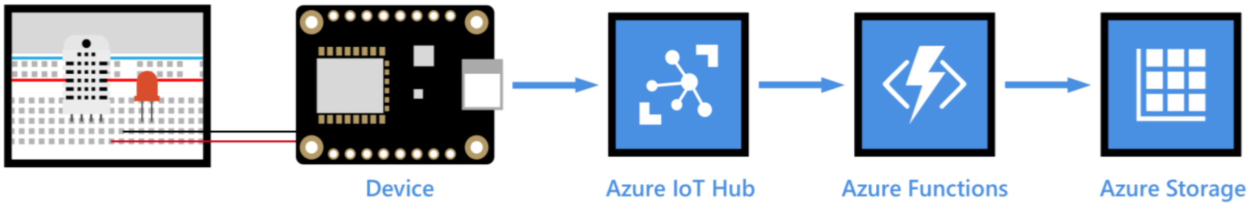
1.6.3 Cloud-to-Device (C2D) Example

Use the following commands to send a cloud-to-device command and then wait for the device to respond with an acknowledgment:

1. Open the Chrome browser with the running Raspberry PI simulator
2. `iothub-explorer send web01 “Hello World”`
3. You may now close the Raspberry Pi web simulator application browser and stop this device from running, it is no longer needed for the remainder of this lab. Leaving it open is fine and causes no problems if you choose to leave it open.

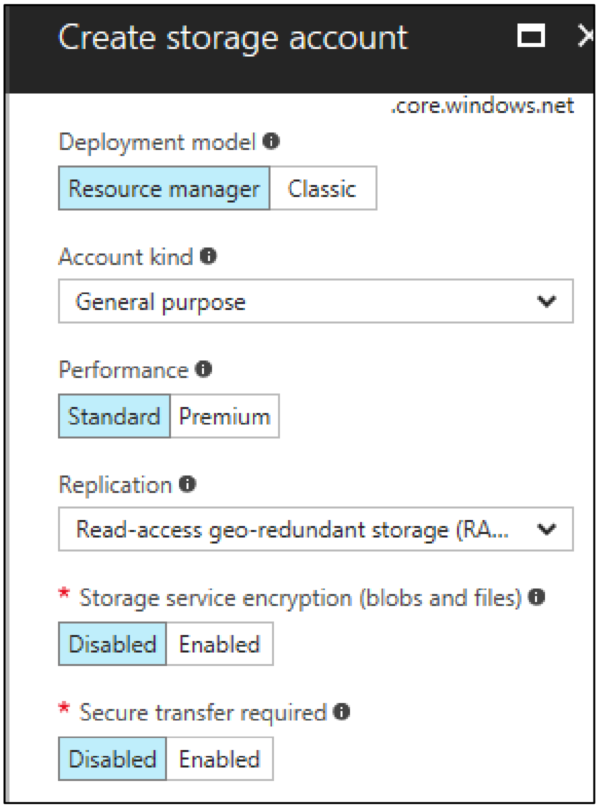
2 Lab 2 - Store Messages to Table

In this lab you create an Azure storage account and an Azure Function to detect and then store IoT Hub messages into an Azure table storage.



2.1 Create an Azure Storage Account

1. In the Azure portal, click **New > Storage > Storage account – blob, file, table, queue**
2. **Name**: (kstrokerstorageaccount01) Must be unique **{lastname}storageaccount01**
3. **Deployment model**: Resource manager (default)
4. **Account kind**: General purpose (default)
5. **Performance**: Standard (default)
6. **Replication**: Read-access (default)
7. **Storage service encryption**: Disabled (Default)
8. **Secure transfer required**: Disabled (Default)



Create storage account

.core.windows.net

Deployment model: Resource manager | Classic

Account kind: General purpose

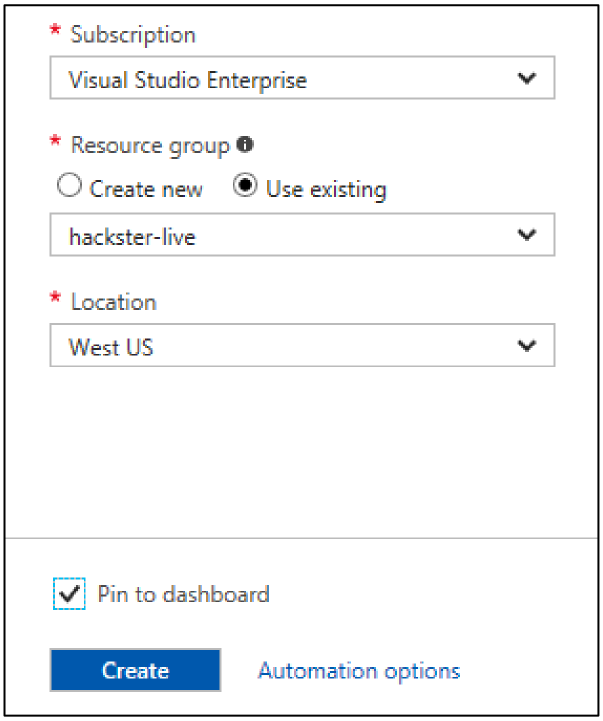
Performance: Standard | Premium

Replication: Read-access geo-redundant storage (RA...

* Storage service encryption (blobs and files): Disabled | Enabled

* Secure transfer required: Disabled | Enabled

9. **Subscription**: (Free Trial), or other as appropriate
10. **Resource group**: (hackster-live) Use existing **hackster-live** resource group
11. **Location**: **Use existing > (match your resource group)**
12. **Pin to Dashboard**: Select



* Subscription
Visual Studio Enterprise

* Resource group
○ Create new ● Use existing
hackster-live

* Location
West US

☑ Pin to dashboard

13. Click **Create** button
14. The new storage account is created and opens the storage console screen when finished

2.2 Prepare IoT Hub Endpoint

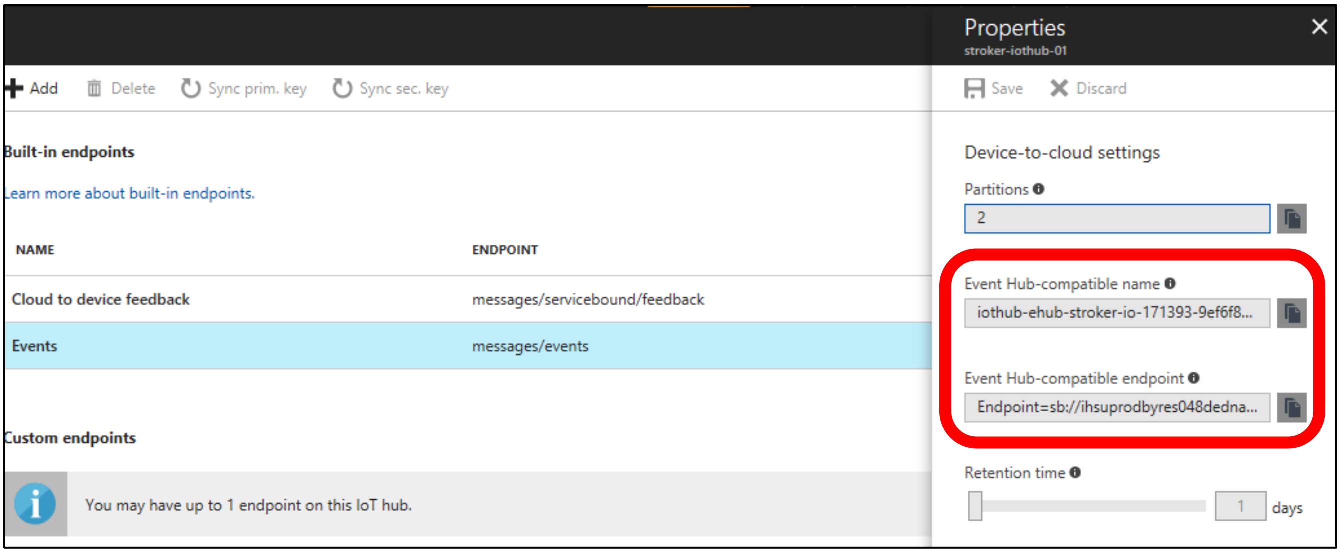
IoT Hub exposes a built-in Event Hub-compatible endpoint to enable applications to read IoT Hub messages. Meanwhile, applications use consumer groups to read data from an IoT Hub.

Before creating an Azure Function App to read data from your IoT hub, you need to:

- Get the connection string of your Event hub-compatible endpoint.
- Create a consumer group for your IoT hub.

2.2.1 Get the connection string of the IoT hub endpoint

1. Open the IoT hub control screen
2. Select **MESSAGING > End points**
3. Click **Events** under **Built-in endpoints**
4. In the **Properties** blade, make a note of the following values, paste them into your **Credentials.txt** for later reference:
    - Event Hub-compatible name



Built-in endpoints

NAME	ENDPOINT
Cloud to device feedback	messages/servicebound/feedback
Events	messages/events

Custom endpoints

You may have up to 1 endpoint on this IoT hub.

Properties
stroker-iothub-01

Device-to-cloud settings

Partitions
2

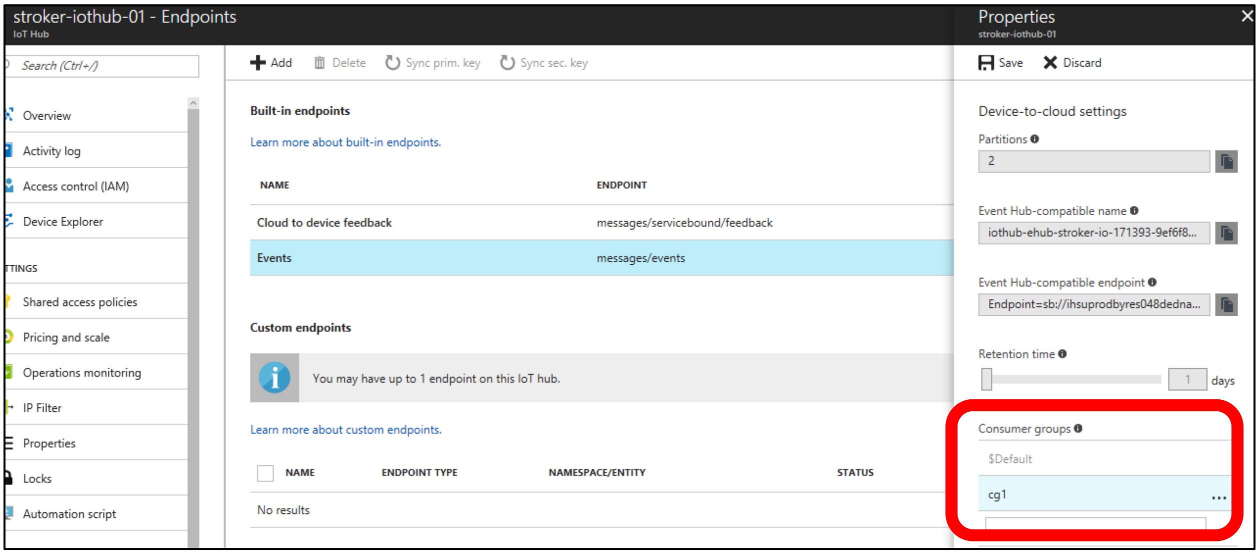
Event Hub-compatible name
iothub-ehub-stroker-io-171393-9ef6f8...

Event Hub-compatible endpoint
Endpoint=sb://ihsuprodbyres048dedna...

Retention time
1 days

2.2.2 Create an IoT Hub Consumer Group

1. Using the same **Properties** blade from the lab step, scroll down
2. In the **Properties** pane, enter a name (**cg1**) under **Consumer groups**
3. Click **Save** button



Built-in endpoints

Learn more about built-in endpoints.

NAME	ENDPOINT
Cloud to device feedback	messages/servicebound/feedback
Events	messages/events

Custom endpoints

You may have up to 1 endpoint on this IoT hub.

Learn more about custom endpoints.

NAME	ENDPOINT TYPE	NAMESPACE/ENTITY	STATUS
No results			

Device-to-cloud settings

Partitions: 2

Event Hub-compatible name: iothub-ehub-stroker-io-171393-9ef6f8...

Event Hub-compatible endpoint: Endpoint=sb://ihsuprodbyres048dedna...

Retention time: 1 days

Consumer groups: $Default, cg1

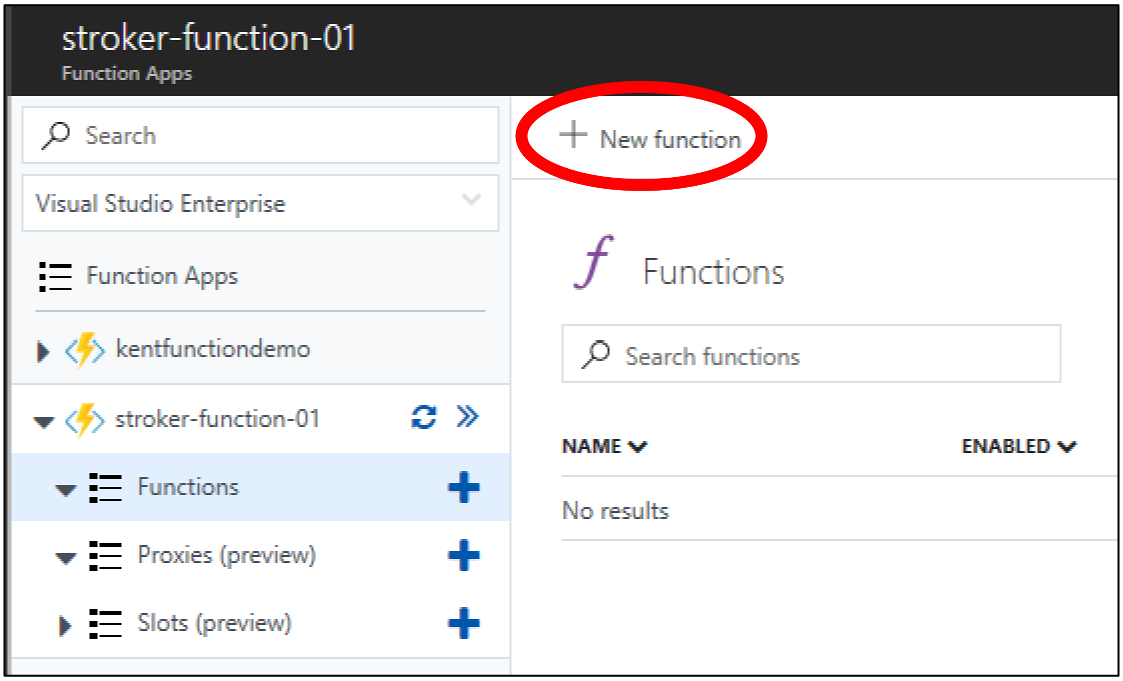
2.3 Create an Azure Function App

First create the Azure Function application, this is what “holds” the Azure Functions.

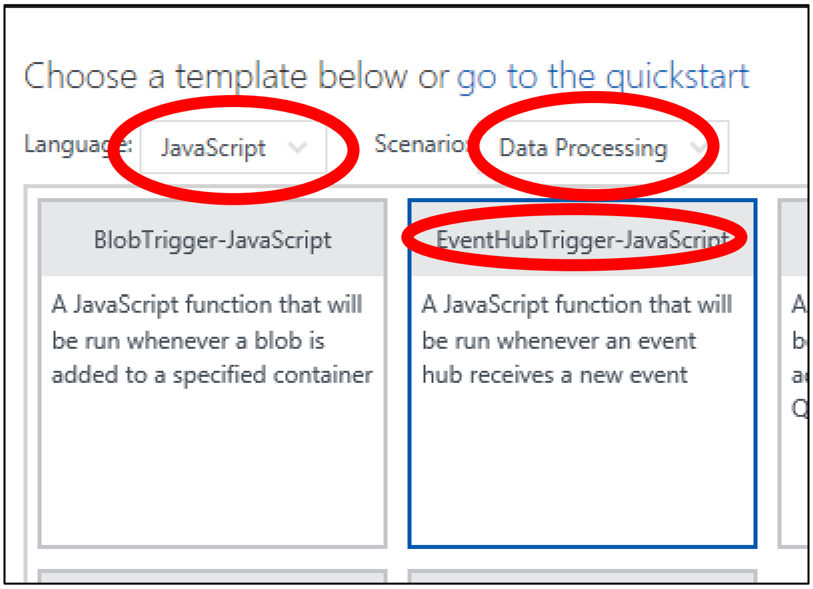
1. **New** > **Compute** > **Function App**
2. **App name**: (kstroker-funcapp-01) As an example **{lastname}-funcapp-01**
3. **Subscription**: **(**Free Trial), or what is appropriate
4. **Hosting Plan**: (default) Consumption Plan
5. **Resource group**: (hackster-live) Use existing **hackster-live** resource group
6. **Location**: CONFIRM THIS MATCHES THE RESOURCE GROUP, change if it does not!!!
7. **Storage Account**: **Select existing > kstrokerstorageaccount01**
    a. Name of Storage Account your created previously
8. **Pin to dashboard**: Check this option
9. Click **Create** button

2.4 Create an Azure Function

1. When the Function App has successfully created, it opens the control panel
2. Select **Functions**
3. Click **+ New function**



4. Select **JavaScript** for **Language**, and **Data Processing** for **Scenario**
5. Select the **EventHubTrigger-JavaScript** template


Choose a template below or go to the quickstart

Language: JavaScript

Scenario: Data Processing

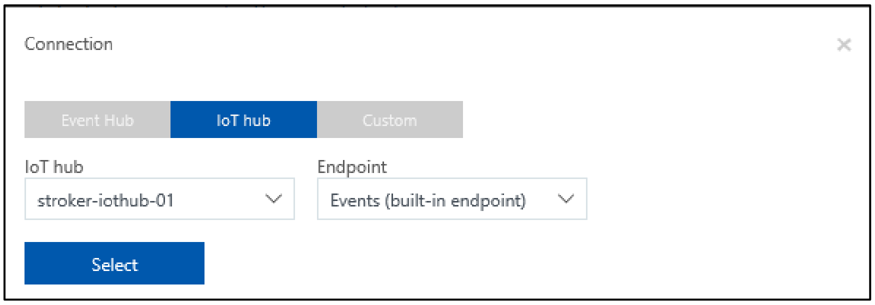
BlobTrigger-JavaScript

A JavaScript function that will be run whenever a blob is added to a specified container

EventHubTrigger-JavaScript

A JavaScript function that will be run whenever an event hub receives a new event

6. **Name your function**: EventHubTriggerJS1 (default, do not change)
7. **Event Hub name**: Paste the Event Hub-compatible name you noted down earlier
8. **Event Hub connection**: Click **new**
9. Pop-up dialog window appears:
    a. Select **IoT Hub**



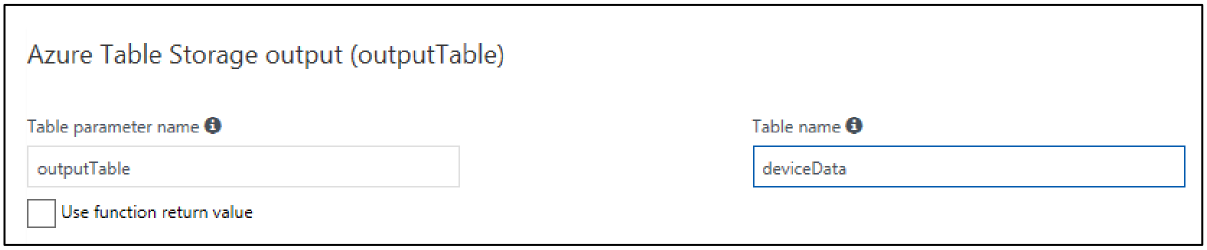
IoT hub: stroker-iothub-01

Endpoint: Events (built-in endpoint)

    b. **IoT hub**: (default) Confirm the name of your IoT Hub
    c. **Endpoints**: (default) Events
    d. Click **Select** button
10. Click **Create** button.

2.5 Configure an Output for the Azure Function

1. Click **Integrate** > **New Output** > **Azure Table Storage** > **Select**.
2. **Table parameter name**: (default) Leave the outputTable for the name
3. **Table name**: Change to deviceData for the name of the table



Azure Table Storage output (outputTable)

Table parameter name: outputTable

Table name: deviceData

□ Use function return value

4. **Storage account connection**:
    a. Click **new**
    b. Select your storage account name
5. Click **Save**.
6. Under **Triggers**, click **Azure Event Hub (eventHubMessages)**.
7. Under **Event Hub consumer group**, enter the name of the consumer group (cg1)
8. Click **Save** button

2.6 Update the Function JavaScript Code

1. Click **function name** (EventHubTriggerJS1)
2. Replace the code in `index.js` with the following:
    a. The file lab2-code.js in the lab2 folder contains a copy of the code
3. Click **Save** button
4. Click **Run** button

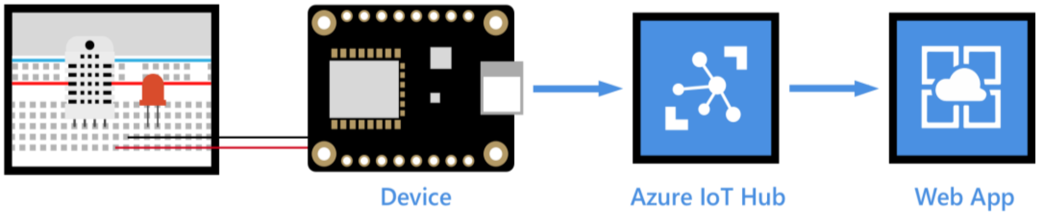
```
'use strict';

// This function is triggered each time a message is revieved in the IoTHub.
// The message payload is persisted in an Azure Storage Table

module.exports = function (context, iotHubMessage) {
 context.log('Message received: ' + JSON.stringify(iotHubMessage));
 var date = Date.now();
 var partitionKey = Math.floor(date / (24 * 60 * 60 * 1000)) + '';
 var rowKey = date + '';
 context.bindings.outputTable = {
  "partitionKey": partitionKey,
  "rowKey": rowKey,
  "message": JSON.stringify(iotHubMessage)
 };
 context.done();
};
```

2.7 Verify Messages Being Written to Table

1. Open **Microsoft Azure Storage Explorer** program
    a. you install as part of the pre-workshop laptop setup
2. **Add an Azure Account** > **Sign in**, and then sign in to the Azure account
3. **Storage Accounts** > storage account > **Tables** > **deviceData**
4. You should see messages sent from your devices to the IoT hub logged in the **deviceData** table

3 Lab 3 - Data Visualization Web App

In this lesson, you learn how to visualize real-time sensor data that your Azure IoT hub receives by running a web application that is hosted on an Azure web app.



3.1 Add another Consumer Group to the IoT Hub

Consumer groups are used by applications to pull data from Azure IoT Hub. In this lesson, you create a consumer group to be used by a coming Azure service to read data from your IoT hub.+

1. Open the IoT Hub control screen
2. **MESSAGING > Endpoints > Events**
3. In the **Properties** blade, scroll down and enter a name (**webapp**) under **Consumer groups**
4. Click **Save** button

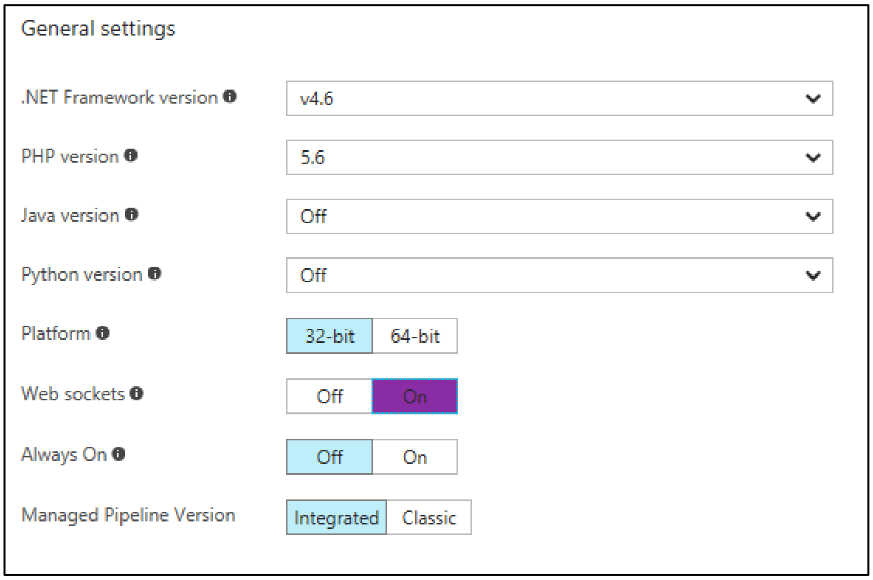
3.2 Create an Azure web app

1. **New > Web + Mobile > Web App**
   a. **App name**: (kstroker-webapp-01) As an example **{lastname}-webapp-01**
   b. **Subscription**: **(**Free Trial), or what is appropriate
   c. **Resource group**: (hackster-live) Use existing **hackster-live** resource group
   d. **App Service plan/Location**
      i. Click on **Create New**
      ii. **App Service Plan**: (kstroker-servplan-01)
      iii. Location: Make sure this matches the region your using
      iv. Pricing tier – **S1 Standard**
         1. We cannot use the F1 since this design requires the web app server to connect back into other Azure services, only Standard and Premium support this feature
         2. Click **OK** button
2. Select **Pin to dashboard**
3. Click **Create** button

The web app server is now built and open the control panel when finished.

3.3 Configure the web app to read data from your IoT hub

1. Click **SETTINGS > Application settings**
2. In **Application settings**, toggle the Web sockets option under General settings.



General settings

.NET Framework version	v4.6
PHP version	5.6
Java version	Off
Python version	Off
Platform	[✓ 32-bit] 64-bit
Web sockets	Off [✓ On]
Always On	[✓ Off] On
Managed Pipeline Version	[✓ Integrated] Classic

3. Add the following key/value pairs under **App settings**:

Key	Value
Azure.IoT.IoTHub.ConnectionString	(Your IoT Hub Connect)
Azure.IoT.IoTHub.DeviceId	myFirstNodeDevice
Azure.IoT.IoTHub.ConsumerGroup	webapp

4. Click **Save** button

3.4 UPLOAD A WEB APPLICATION TO BE HOSTED BY THE WEB APP

The code for the web application server in in the folder on the desktop and you will use Visual Studio to open the solution file, then publish from Visual studio directly to the web application server in the Azure cloud.

3.4.1 Connect Visual Studio to Azure

Visual Studio and the Azure cloud work closely together once the connection is established.

1. Launch the Visual Studio 2017 program
2. Click on **View > Cloud Explorer**
3. Add your Azure account credentials, the little person icon

Once the connection is established, you can see all the resources in your Azure account in the Cloud Explorer panel. In an earlier step, you used the standalone program Microsoft Azure Storage Explorer to view the contents of the Table Store database. Try viewing your table data from inside Visual Studio.

Close the Cloud Explorer panel when finished.

3.4.2 Open and Deploy the Source Code Solution

Open the solution file

1. Open the solution
    a. **File > Open > Project/Solution > {location of lab download} > lab3**
    b. IoTHubWebAppLab.sln
    c. There is a panel on the right; the **Solution Explorer** panel
        i. If missing, open it using **View > Storage Explorer**
    d. Right-click on the project name **IoTHubWebAppLab**
    e. Click on **Publish**
    f. Select **Microsoft Azure App Service**
    g. Click the expand icon of the web application service (kstroker-webapp-01) folder
    h. Select your web app service name (kstroker-webapp-01)
    i. Click **OK** button
    j. Click **Validate Connection** button
    k. Click **Publish** button
        i. Observe progress in the **Output** panel
    l. A web browser automatically launches after successful deployment