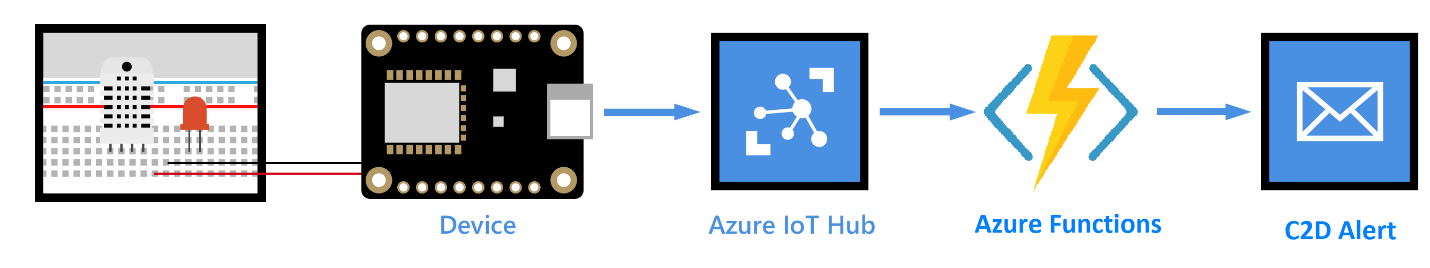
**Sending C2D Messages as an Alert After Receiving Telemetry**

**Steps of the Experiment**

Throughout all the steps, all the other adjustable information we encounter that are not told to be changed, should stay the same as default.

1. Create an IoT hub which is the cloud platform that our device will connect to
2. Register a device for NodeMCU 1.0 (ESP-12E Module) in your IoT hub
3. Setup the circuit
4. Configure NodeMCU 1.0 (ESP-12E Module) with Arduino
5. Download the sample application which will send telemetry to our IoT Hub
6. Create service bus namespace and add a queue to it
7. Add an endpoint and a routing rule to your IoT hub
8. Create an Azure function app to send C2D messages
9. Run the sample application and monitor all activity from VS Code

**Hardware Used for this Experiment**

Breadboard, 10 k ohm resistor, LED, M/M Jumper wires

NodeMCU 1.0 (ESP-12E Module)

DHT11 temperature and humidity sensor

Micro USB to Type A USB cable

PC running Windows

Wireless Network

**Software Used for This Experiment**

Microsoft Azure

Arduino IDE

Visual Studio Code

Azure IoT Toolkit as an extension for VS Code

If you have completed the experiment called “Visualize Real-Time Sensor Data from Azure IoT Hub using Power BI” you can skip the first 5 steps.

**STEP 1: Create an IoT hub which is the cloud platform that our device will connect to**

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. Select Create a resource > Internet of Things > IoT Hub.
3. In the IoT hub pane, enter the following information for your IoT hub:

* Subscription: Choose the subscription that you want to use to create this IoT hub.
* Resource group: Create a resource group to host the IoT hub or use an existing one. For more information, see [Use resource groups to manage your Azure resources](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-portal).
* Region: Select the closest location to you.
* Name: Create a name for your IoT hub. If the name you enter is available, a green check mark appears.

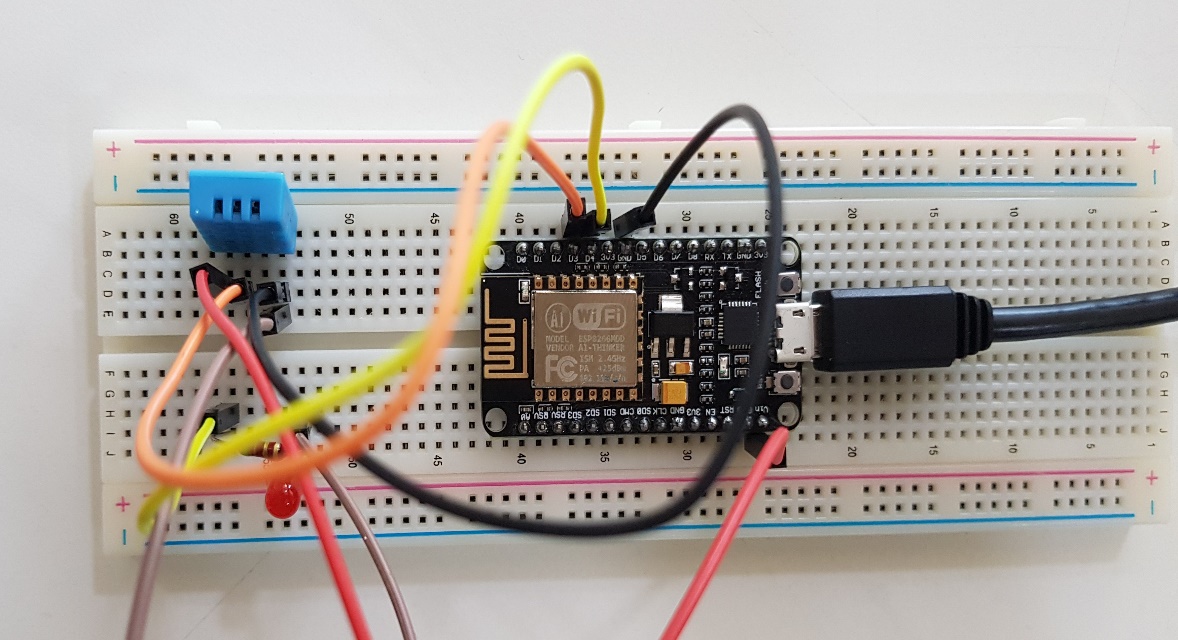
1. Select Next: Size and scale to continue creating your IoT hub.
2. Choose your Pricing and scale tier. Select the F1 - Free tier if it's still available on your subscription.
3. Select Review + create.
4. Review your IoT hub information, then click Create. Your IoT hub might take a few minutes to create. You can monitor the progress in the Notifications pane.

Now that you have created an IoT hub, locate the important information that you use to connect devices and applications to your IoT hub.

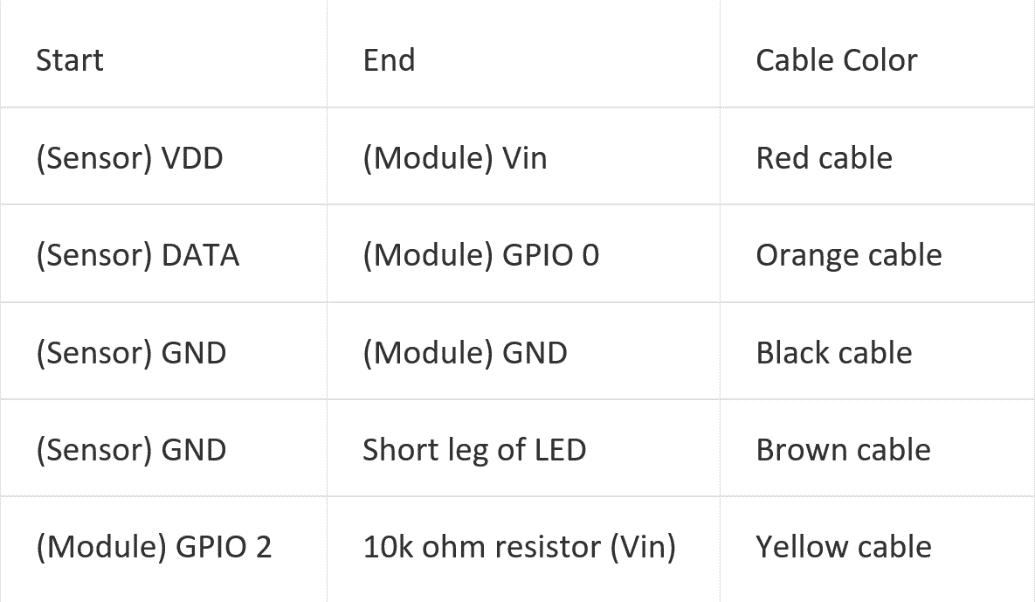
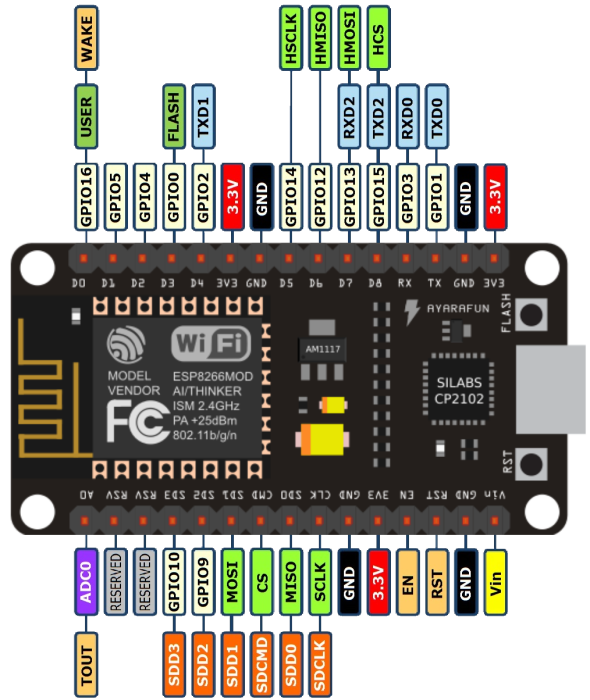
**STEP 2: Register a device for NodeMCU 1.0 (ESP-12E Module) in your IoT hub**

1. In your IoT hub navigation menu, open IoT devices, then click Add to register a device in your IoT hub.
2. Enter a Device ID for the new device. Device IDs are case sensitive.
3. Click Save.
4. After the device is created, open the device from the list in the IoT devices pane.
5. Copy the Connection string---primary key and save it to Notepad to use later.

**STEP 3: Setup the circuit**

****

1. Install the NodeMCU 1.0 (ESP-12E Module) onto the breadboard and make the following connections as shown below in the table.

****

1. Connect long leg of LED in series with 10k ohm resistor.
2. Use the Micro USB to Type A USB cable to connect NodeMCU 1.0 (ESP-12E Module) to your computer.

**STEP 4: Configure NodeMCU 1.0 (ESP-12E Module) with Arduino**

1. Open the Arduino IDE, go to Files and click on Preferences
2. Copy the below code in Additional Boards Manager URLs  
   <http://arduino.esp8266.com/stable/package_esp8266com_index.json>
3. Click OK to close the Preferences Tab
4. Go to Tools and Board, and then select Boards Manager
5. Navigate to esp8266 by ESP8266 Community and install the software for Arduino  
   Go to Board and select NodeMCU 1.0 (ESP-12E Module)
6. Select the correct COM port
7. Make sure that these settings are correct: (Upload Speed: 115200, CPU Frequency: 80MHz, Flash Size: 4M (3M SPIFFS), Upload Using: Serial)

**STEP 5: Download the sample application which will send telemetry to our IoT Hub**

1. Download the application from GitHub:

<https://github.com/Azure-Samples/iot-hub-feather-huzzah-client-app.git>

1. In order to assure the workability of the application, download the necessary libraries. Select Sketch > Include Library > Manage Libraries. Type in and download the followings.

* Azure IoT Hub
* Azure IoT Utility
* Azure IoT Protocol MQTT
* Arduino Json

Then, download the DHT Sensor Library from GitHub and simply add it to ‘’libraries’’ folder in ‘’Arduino’’ folder.

<https://github.com/adafruit/DHT-sensor-library>

1. Change a few lines in ‘’configurations.h‘’ to ensure it runs.

* **from** ‘’DEVICE\_ID: Feather HUZZAH ESP8266 Wi-Fi ’’ **to** ‘’#define DEVICE\_ID "NodeMCU 1.0 (ESP-12E Module)"’’
* **from** ‘’#define DHT\_TYPE DHT22’’ **to** ‘’#define DHT\_TYPE DHT11’’
* **from** ‘’#define LED\_PIN 0’’ **to** ‘’#define LED\_PIN 2’’
* **from** ‘’#define DHT\_PIN 2’’ **to** ‘’#define DHT\_PIN 0’’
* **from** ‘’ #define TEMPERATURE\_ALERT 30” **to** ‘’ #define TEMPERATURE\_ALERT 25”

**STEP 6: Create service bus namespace and add a queue to it**

1. On the [Azure portal](https://portal.azure.com/), click Create a resource > Enterprise Integration > Service Bus.
2. Provide the following information:

* Name: The name of the service bus.
* Pricing tier: Click Basic > Select. The Basic tier is sufficient for this tutorial.
* Resource group: Use the same resource group that your IoT hub uses.
* Location: Use the same location that your IoT hub uses.

1. Click Create.

To add a service bus queue to your service bus namespace, follow these steps:

1. Open the service bus namespace, and then click + Queue.
2. Enter a name for the queue and then click Create.
3. Open the service bus queue, and then click Shared access policies > + Add.
4. Enter a name for the policy, check Manage, and then click Create.

**STEP 7: Add an endpoint and a routing rule to your IoT hub**

1. Open your IoT hub, click Endpoints > + Add.
2. Enter the following information:

* Name: The name of the endpoint.
* Endpoint type: Select Service Bus Queue.
* Service Bus namespace: Select the namespace you created.
* Service Bus queue: Select the queue you created.

1. Click OK.

### To add a routing rule to your IoT Hub, follow these steps:

1. In your IoT hub, click Routes > + Add.
2. Enter the following information:
3. Name: The name of the routing rule.
4. Data source: Select DeviceMessages.
5. Endpoint: Select the endpoint you created.
6. Query string: Enter “ **temperatureAlert = “true”** “
7. Click Save.

**STEP 8: Create an Azure function app to send C2D messages**

1. Select the New button found on the upper left-hand corner of the Azure portal, then select Compute > Function App.
2. Provide the following information:

* App Name: Name that identifies your new function app.
* Subscription: The subscription under which this new function is created.
* Resource Group: Name for the new resource group in which to create your function app.
* OS: Serverless hosting is currently only available when running on Windows. For Linux hosting, see [Create your first function running on Linux using the Azure CLI](https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function-azure-cli-linux).
* Hosting Plan: Hosting plan that defines how resources are allocated to your function app. In the default Consumption Plan, resources are added dynamically as required by your functions.
* Location: Choose a [region](https://azure.microsoft.com/regions/) near you or near other services your functions access.
* Storage Account: Name of the new storage account used by your function app.

1. Click Create.
2. Select the Notification icon in the upper-right corner of the portal and watch for the Deployment succeeded message.
3. Select Go to resource to view your new function app.
4. Click on your created function app to reach the “Functions” tab then click “+”
5. Click “IoT Hub(Service Bus Queue)” and fill in the following information:

* Language: Choose C#.
* Service Bus Connection: Choose the created service bus connection and its policy.
* Access Rights: Choose Manage.
* Queue Name: Name to identify the queue.

1. Much like in a C# project, dependencies in an Azure Function are controlled through a file referred to as a Project.json file. The following is a sample of the Project.json file used in this sample to reference Microsoft.Azure.Devices and its dependencies to allow for communication with IoT Hub.

### {

### "frameworks": {

### "net46":{

### "dependencies": {

### "Microsoft.Azure.Amqp": "1.1.5",

### "Microsoft.AspNet.WebApi.Client": "5.2.3",

### "Microsoft.AspNet.WebApi.Core": "5.2.3",

### "Microsoft.Azure.Devices": "1.0.9"

### }

### }

### }

### }

1. Once you create a Project.json file, you will need to deploy it to your Azure Function. Click “View Files” near on Function tab, drag and drop your Project.json file on the page to add it to the folder. If your Project.json file was properly deployed, your Azure Function will begin loading the desired nugget packages. To test this deployment, scroll down to the Logs pane for your Function. You should see references to the nugget packages loading.
2. Under your Function App’s “Overview”, go to “Application Settings”. Here, there should be a table which contains application settings. Add another variable by providing the following information:

* App Setting Name: Type in “AZURE\_IOTHUB” or anything else but make sure the variable name matches your application code.
* Value: Paste your IoT Hub connection string.
* Slot Setting: Leave it unchecked.

1. Open “run.csx” folder and paste this application code which sends an alert message to our device when the room temperature goes over 25 degrees. Don’t forget to modify the “deviceID” string variable. Run and test it to assure the workability of the application.

### #r "Newtonsoft.Json"

### using System;

### using Microsoft.Azure.Devices;

### using Newtonsoft.Json;

### using System.Text;

### using System.Threading.Tasks;

### static ServiceClient serviceClient;

### public static void Run(string myQueueItem, TraceWriter log)

### {

### log.Info($"C# ServiceBus queue trigger function processed message: {myQueueItem}");

### string messageString = "IT IS OVER 25 DEGREES!!!";

### string deviceID = "myDeviceID97";

### var connectionString = Environment.GetEnvironmentVariable("AZURE\_IOTHUB");

### serviceClient = ServiceClient.CreateFromConnectionString(connectionString);

### var commandMessage = new Message(Encoding.ASCII.GetBytes(messageString));

### serviceClient.SendAsync(deviceID, commandMessage);

### }

**STEP 9: Run the sample application and monitor all activity from VS Code**

1. Upload the Arduino application to NodeMCU 1.0 (ESP-12E Module)
2. After the upload completes successfully, follow these steps to enter your credentials:

Click Tools > Serial Monitor.

In the serial monitor window, notice the two drop-down lists in the lower-right corner. Select No line ending for the left drop-down list. Select 115200 baud for the right drop-down list.

In the input box located at the top of the serial monitor window, enter the following information if you are asked to provide them, and then click Send.

* + Wi-Fi SSID
  + Wi-Fi password
  + Device connection string

1. While the sample application is running, get back to VS Code. After downloading the Azure IoT Toolkit as an extension for VS Code, sign in to your Azure. Select your IoT Hub and the devices which are registered will appear on the bottom left corner tab named “AZURE IOT HUB DEVICES”.
2. Right-click on the registered device and click both “Start monitoring D2C Message” and “Start monitoring C2D Message”. To see the messages being sent, open the Output pane.

**You’ve successfully sent alert messages with a condition from your Azure IoT hub to your device and monitored all activity via VS Code.**