Real-Time Big Data Processing with Azure

Lab 2 - Getting Started with IoT Hubs

Overview

In this lab, you will create an Azure IoT Hub and use it to collect data from a client application.

What You'll Need

To complete the labs, you will need the following:

- A web browser
- A Microsoft account
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- The lab files for this course

Note: To set up the required environment for the lab, follow the instructions in the Setup document for this course. Specifically, you must have signed up for an Azure subscription and installed Node.JS on your computer.

Exercise 1: Provision an IoT Hub

In this exercise, you will create an IoT hub.

Note: The Microsoft Azure portal is continually improved in response to customer feedback. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of the portal exactly.

Create an IoT Hub

In this procedure, you will use the Azure portal to create an IoT hub.

1. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Internet of Things** menu, click **IoT Hub**.
2. In the **IoT Hub** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name (and make a note of it!)*
    - **Pricing and scale tier**: Free
    - **IoT Hub Units**: 1
    - **Device-to-cloud partitions**: 2 partitions
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the existing resource group you created previously*

- **Location:** *Select any available region*
- **Pin to dashboard**: *Not selected*

3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the service bus namespace to be deployed (this can take a few minutes.)

Exercise 2: Register a Client Device

Unlike event hubs, all devices that use an IoT hub must be individually registered, and use their own endpoint and shared access key to access the hub. In this exercise, you will register a client device.

Get the Hostname and Connection String for the Hub

To register a client device, you must run a script that uses a connection with sufficient permissions to access the hub registry. In this case, you will use the built-in **iothubowner** shared access policy to accomplish this.

1. In the Azure portal, browse to the IoT Hub you created previously.
2. In the blade for your IoT Hub, click **Shared access policies**.
3. In the **Shared access policies** blade, click the **iothubowner** policy, and then copy the **connection string-primary key** for your IoT Hub to the clipboard – you will need this later.

   Note: For more information about access control for IoT hubs, see Access control in the "Azure IoT Hub developer guide."

Create a Device Identity

Each device that sends data to the IoT hub must be registered with a unique identity.

1. Open a Node.JS console and navigate to the **createdeviceid** folder in the folder where you extracted the lab files.
2. Enter the following command, and press RETURN to accept all the default options. This creates a package.json file for your application:

   ```
   npm init
   ```

3. Enter the following command to install the Azure IoT Hub package:

   ```
   npm install azure-iothub
   ```

4. Use a text editor to edit the **createdeviceid.js** file in the **createdeviceid** folder.
5. Modify the script to set the **connStr** variable to reflect the shared access policy connection string for your IoT Hub, as shown here:

   ```
   'use strict';

   var iothub = require('azure-iothub');

   var connStr = '<IOT-HUB-CONNECTION-STRING>';

   var registry = iothub.Registry.fromConnectionString(connStr);

   var device = new iothub.Device(null);
   device.deviceId = 'MyDevice';
   registry.create(device, function(err, deviceInfo, res) {
     if (err) {
       registry.get(device.deviceId, printDeviceInfo);
     }
     if (deviceInfo) {
   ```

```
    printDeviceInfo(err, deviceInfo, res)
  }
});

function printDeviceInfo(err, deviceInfo, res) {
  if (deviceInfo) {
    console.log('Device id: ' + deviceInfo.deviceId);
  }
}
```

6. Save the script and close the file.
7. In the Node.JS console window, enter the following command to run the script:

   ```
   node createdeviceid.js
   ```

8. Verify that the script registers a device with the ID *MyDevice*.
9. In the Azure portal, on the blade for your IoT Hub, click the **Overview** tab and then at the top of the blade, click **Devices** and verify that **MyDevice** is listed.
10. Click **MyDevice** and view the device-specific keys and connection strings that have been generated. Then copy the **connection string-primary key** for *MyDevice* to the clipboard. You will use this in the next exercise.

Exercise 3: Submit Data from a Client Device Application

Now that you have registered a client device, you can create an application that the device can use to submit data to the IoT Hub.

Create a Client Device Application

Now that you have registered a device, it can submit data to the IoT hub.

Note: The code for *MyDevice* is similar to the code you used in the previous lab to submit events to an event hub – it generates readings from ten random devices. In a real solution, each device would use its own device-specific ID and connection string to enable you to manage them individually.

1. In the Node.JS console, navigate to the **iotdevice** folder in the folder where you extracted the lab files.
2. Enter the following command, and press RETURN to accept all the default options. This creates a package.json file for your application:

   ```
   npm init
   ```

3. Enter the following command to install the Azure IoT device and AMQP protocol packages:

   ```
   npm install azure-iot-device azure-iot-device-amqp
   ```

4. Use a text editor to edit the **iotdevice.js** file in the **iotdevice** folder.
5. Modify the script to set the **connStr** variable to reflect the device connection string for the *MyDevice* device (which you copied to the clipboard in the previous exercise), as shown here:

   ```
   'use strict';

   var clientFromConnectionString = require('azure-iot-device-
   amqp').clientFromConnectionString;
   var Message = require('azure-iot-device').Message;

   var connStr = '<DEVICE_CONNECTION_STRING>';
   ```

```
var client = clientFromConnectionString(connStr);

function printResultFor(op) {
  return function printResult(err, res) {
    if (err) console.log(op + ' error: ' + err.toString());
    if (res) console.log(op + ' status: ' + res.constructor.name);
  };
}

var connectCallback = function (err) {
  if (err) {
    console.log('Could not connect: ' + err);
  } else {
    console.log('Client connected');

    // Create a message and send it to the IoT Hub every second
    setInterval(function(){
        var r = Math.random();
        var data = JSON.stringify({ device: 'MyDevice', reading: r });
        var message = new Message(data);
        console.log("Sending message: " + message.getData());
        client.sendEvent(message, printResultFor('send'));
    }, 1000);
  }
};

client.open(connectCallback);
```

6. Save the script and close the file.

Submit data to the IoT Hub

Now you can run your client application to submit data to the IoT hub.

1. In the Node.JS console window, enter the following command to run the script:

   ```
   node iotdevice.js
   ```

2. Observe the script running as it starts to submit device readings. Then leave the script running and start the next exercise.

Exercise 4: Read Data from the IoT Hub

When devices submit messages to your IoT hub, you can read the messages from its event hub endpoint.

Create an Application to Read Messages from the IoT Hub

In this procedure, you will create a client application that reads from the event hub endpoint of the IoT hub.

1. In the Azure portal, browse to the blade for the IoT Hub.
2. In the blade for your IoT Hub, click **Shared access policies**.
3. In the **Shared access policies** blade, click the **service** policy, and then copy the **connection string-primary key** for your IoT Hub to the clipboard – you will need this later.
4. Open a new Node.JS console, and navigate to the **iotreader** folder in the folder where you extracted the lab files.
5. Enter the following command, and press RETURN to accept all the default options. This creates a package.json file for your application:

```
npm init
```

6. Enter the following command to install the Azure IoT Hub package:

   ```
   npm install azure-event-hubs
   ```

7. Use a text editor to edit the **iotreader.js** file in the **iotreader** folder.
8. Modify the script to set the **connStr** variable to reflect the **service** shared access policy connection string for your IoT Hub (not the device-specific connection string you used in the previous procedure), as shown here:

   ```
   'use strict';

   var EventHubClient = require('azure-event-hubs').Client;

   var connStr = '<IOT-HUB-CONNECTION-STRING>';

   var printError = function (err) {
     console.log(err.message);
   };

   var printMessage = function (message) {
     console.log('Message received: ');
     console.log(JSON.stringify(message.body));
     console.log('');
   };

   var client = EventHubClient.fromConnectionString(connStr);
   client.open()
       .then(client.getPartitionIds.bind(client))
       .then(function (partitionIds) {
           return partitionIds.map(function (partitionId) {
               return client.createReceiver('$Default', partitionId, {
   'startAfterTime' : Date.now()}).then(function(receiver) {
                   console.log('Created partition receiver: ' + partitionId)
                   receiver.on('errorReceived', printError);
                   receiver.on('message', printMessage);
               });
           });
       })
       .catch(printError);
   ```

9. Save the script and close the file.

Read Data from the IoT Hub

Now you arew ready to run your application and read data from the IoT Hub.

1. Ensure that the **iotdevice.js** script is still running.
2. In the Node.JS console window for the **iotreader.js** application, enter the following command to run the script:

   ```
   node iotreader.js
   ```

3. Observe the script running as it reads the device readings that were submitted to the IoT hub by the **iotdevice.js** script.
4. After both scripts have been running for a while, stop them by pressing CTRL+C in each of the console windows. Then close the console windows.

Note: You will use the resources you created in this lab when performing the next lab, so do not delete them. Ensure that all Node.js scripts are stopped to minimize ongoing resource usage costs.