Real-Time Big Data Processing with Azure

Lab 1 - Getting Started with Event Hubs

Overview

In this lab, you will create an Azure Event Hub and use it to collect event 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 Azure Event Hub

In this exercise, you will create an Azure event hub to which applications can submit event details.

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 a Namespace

Before you can create event hubs, you must create an event hub namespace.

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Internet of Things** menu, click **Event Hubs**.
3. In the **Create namespace** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name (and make a note of it!)*
    - **Pricing tier**: Basic
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Create a new resource group with a unique name*

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

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

Add an Event Hub

Now that you have a namespace, you are ready to add an event hub.

1. In the Azure portal, browse to the namespace you created previously.
2. In the blade for your namespace, click **Add Event Hub**.
3. In the **Create Event Hub** blade, enter the following settings (other options may be shown as unavailable) and click **Create**:
    - **Name**: *Enter a unique name (and make a note of it!)*
    - **Partition Count**: 2
4. In the Azure portal, wait for the notification that the event hub has been created.

Create a Shared Access Policy

Access to event hubs is controlled using shared access policies.

1. In the Azure portal, in the blade for your namespace, select the event hub you just created.
2. In the blade for your event hub, click **Shared access policies**.
3. On the **Shared access policies** blade for your event hub, click **Add**. Then add a shared access policy with the following settings:
    - **Policy name**: DeviceAccess
    - **Claim**: *Select **Send** and **Listen***
4. Wait for the shared access policy to be created, then select it, and note that primary and secondary connection strings have been created for it. Copy the primary connection string to the clipboard - you will use it to connect to your event hub from a simulated client device in the next exercise.

Exercise 2: Create a Client to Submit Events

In this exercise, you will create a simple client applications that submits event details.

Create a Node.JS Application to Submit Events

Now that you have created an event hub, you can submit events to it from devices and applications. In this exercise, you will create a Node.JS application to simulate random device readings.

1. Open a Node.JS console and navigate to the **eventclient** 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 Event Hubs package:

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

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

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

var connStr = '<EVENT_HUB_CONNECTION_STRING>';

var client = EventHubClient.fromConnectionString(connStr)
client.createSender()
    .then(function (tx) {
        setInterval(function(){
          dev = 'dev' + String(Math.floor((Math.random() * 10) + 1));
          val = String(Math.random());
          console.log(dev + ": " + val);
          tx.send({ device: dev, reading: val});
        }, 1000);
    });
```

6. Save the script and close the file.

Submit Events to the Event Hub

Now that you have a client application, you can run it to submit events to your event hub.

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

   ```
   node eventclient.js
   ```

2. Observe the script running as it submits simulated events. Then leave it running and continue to the next exercise.

Exercise 3: Read Data from an Event Hub

In this exercise, you will create a simple client applications that reads data from an event hub.

Create a Client Application to Read Messages from the Event Hub

Now you will create a second Node.JS client application to read events from your event hub.

1. Open a second Node.JS console, and navigate to the **eventreader** 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-event-hubs
   ```

4. Use a text editor to edit the **eventreader.js** file in the **eventreader** 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 EventHubClient = require('azure-event-hubs').Client;

   var connStr = '<EVENT_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);
```

6. Save the script and close the file.

Read Events from the Event Hub

You can now run your second client application to read event data from the event hub

1. Ensure that your **eventclient.js** script is still running.
2. In the Node.JS console window for the folder containing your **eventreader.js** script, enter the following command to run the script:

   ```
   node eventtreader.js
   ```

3. Observe the script running as it reads the device readings that are submitted to the event hub by the **eventclient.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.