# How to integrate RunMyProcess to Azure IoT Hub:

# **Retrieving messages sent by a simulated device with REST calls**

#### Rodrigo Riveros – Senior Consultant – Fujitsu Canada

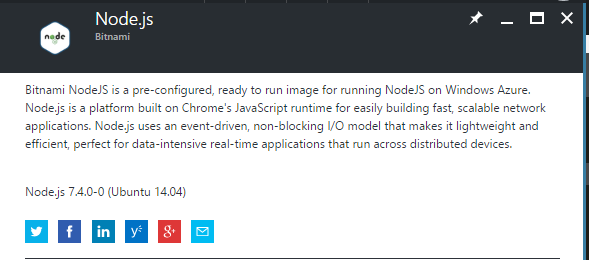
### Pre-requisites:

* An Azure Account

[https://portal.azure.com]

* A NodeJS server (If you’re going to simulate the IoT Device)

My suggestion is to use an Azure NodeJS Virtual Machine from bitnami. It came pre-configured to run NodeJS:



* Access to RMP IDE

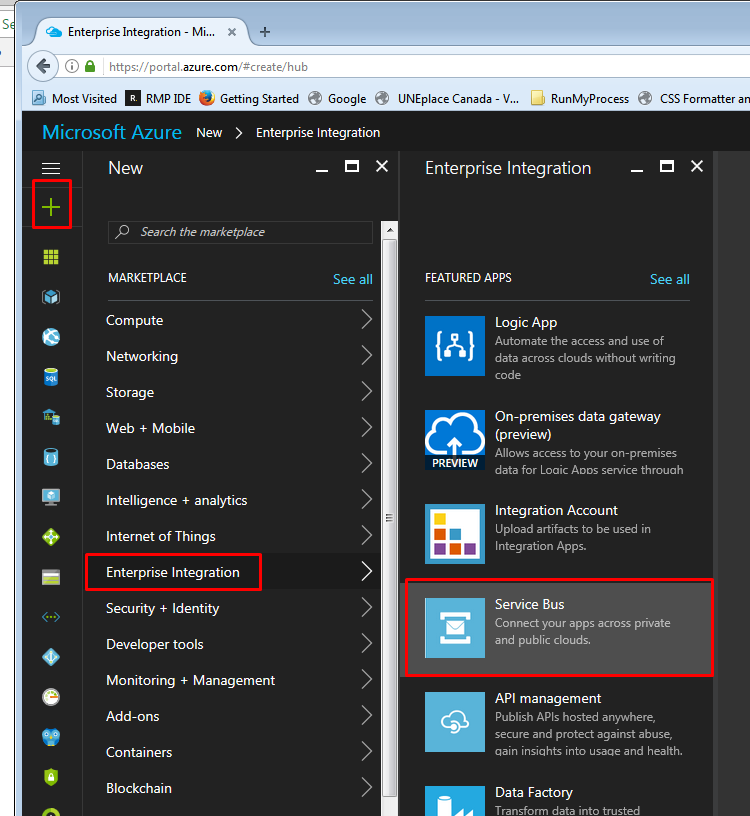
[https://portal.runmyprocess.com/ide/]

* Postman (To test the message to send before implementing it in RMP) [https://www.getpostman.com/]

## Create Service Bus

Reference: <https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues>

1. Go to the azure portal
2. Click on new, Enterprise Integration, Service Bus.



1. Fill the form with the required data.
2. Wait until the deployment ends.

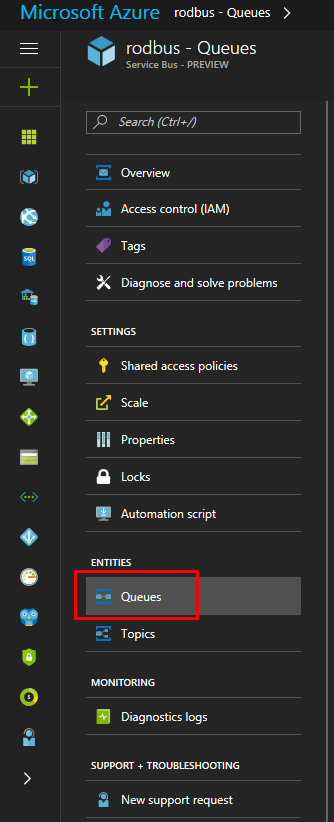
## Create Queue and route the message to it.

Reference:

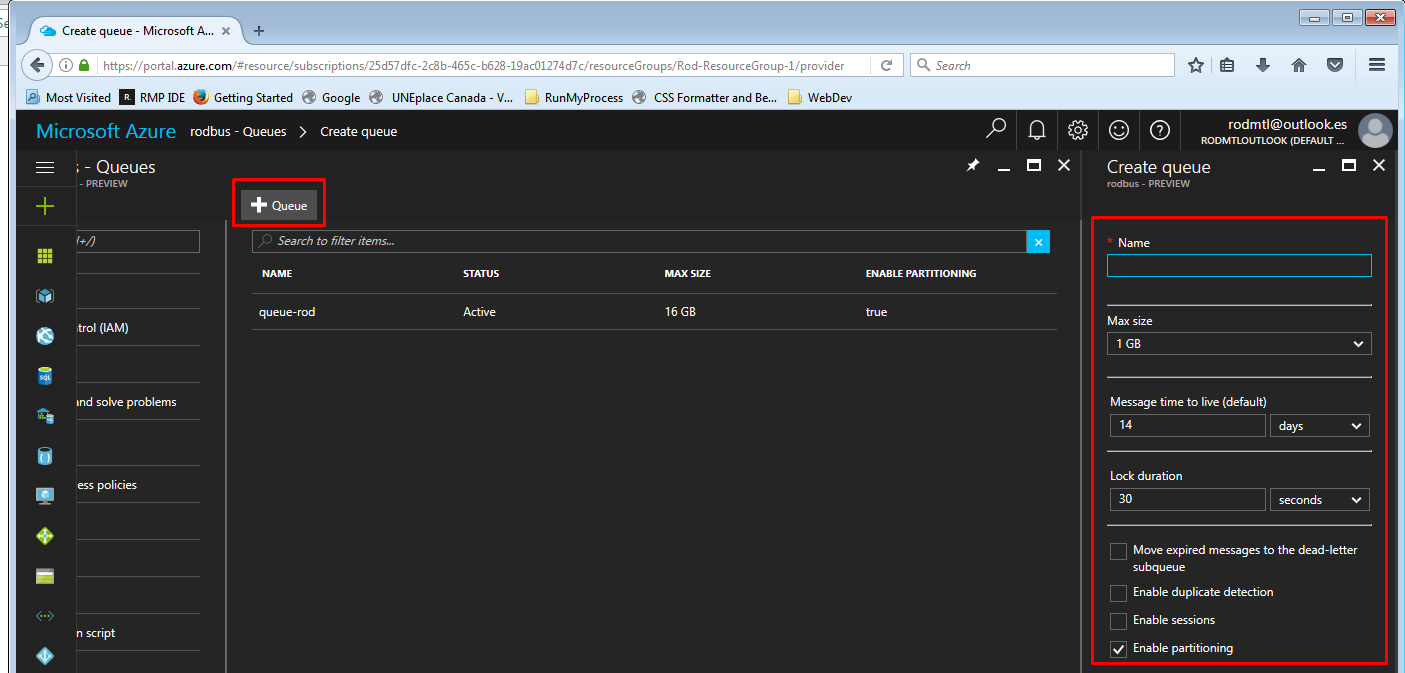
<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues>

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-process-d2c>

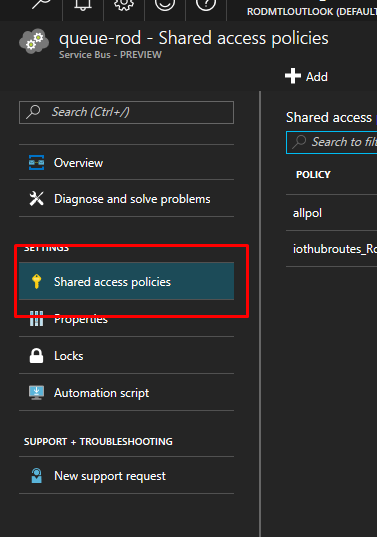
1. Open the service bus you just have created in the previous step, click on Queues:



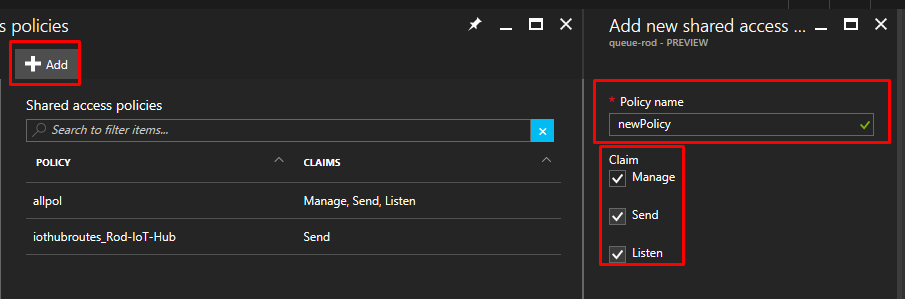
1. Click on the plus button to add a new queue and fill the form.



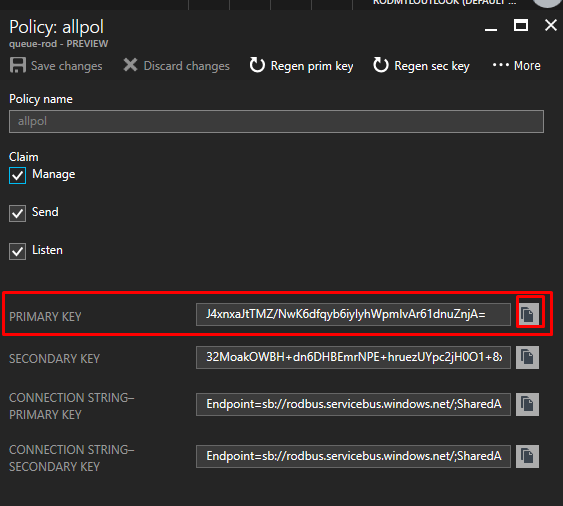
1. Once created, click on the queue & go to shared access policies:



1. Click in the plus button to create a new shared access policy for the queue. For the demo select all the permissions available.



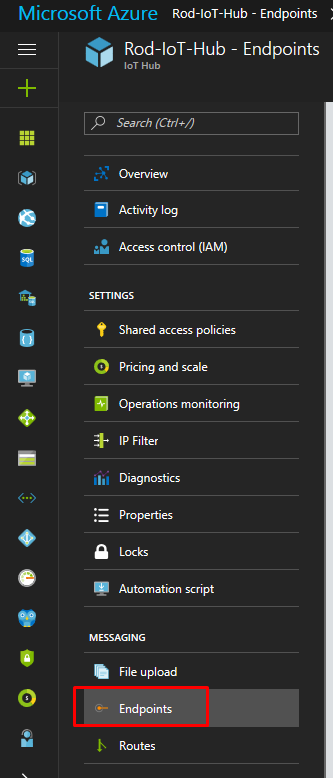
1. Once created, please Take note of the primary key, you will need it.



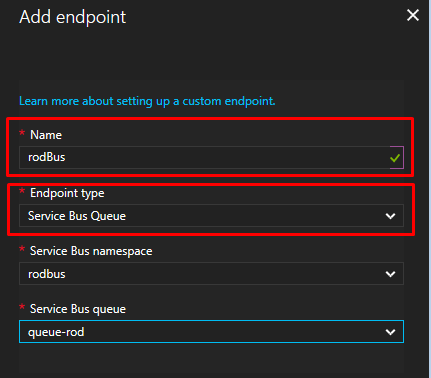
## Create Endpoint in the IoT Hub

Reference: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-process-d2c>

1. Open your IoT hub and click **Endpoints**.



1. Click on the add button.
2. Fill the add endpoint form (the service bus namespace & queue will be select boxes with the bus data you’ve created in the previous step)



1. Save the new endpoint

## Create Route in the IoT hub

Reference:

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-process-d2c>

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-query-language#get-started-with-device-to-cloud-message-routes-query-expressions>

1. Open your IoT hub and click **Routes**:

2. Click on the add button and fill the information.

The data source will be DeviceMessages & the endpoint will be the name of the endpoint you just have created.

As query string we’re going to filter the wind speed greater than 12:

|  |
| --- |
| as\_number(windSpeed)>12 |

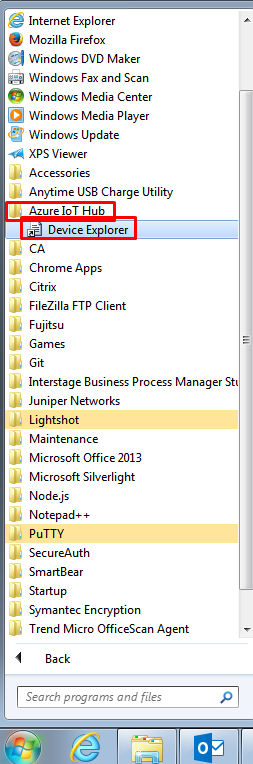
3. Save the route.

## Create a device identity in the IoT hub

Reference: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-node-node-direct-methods#create-a-device-identity>

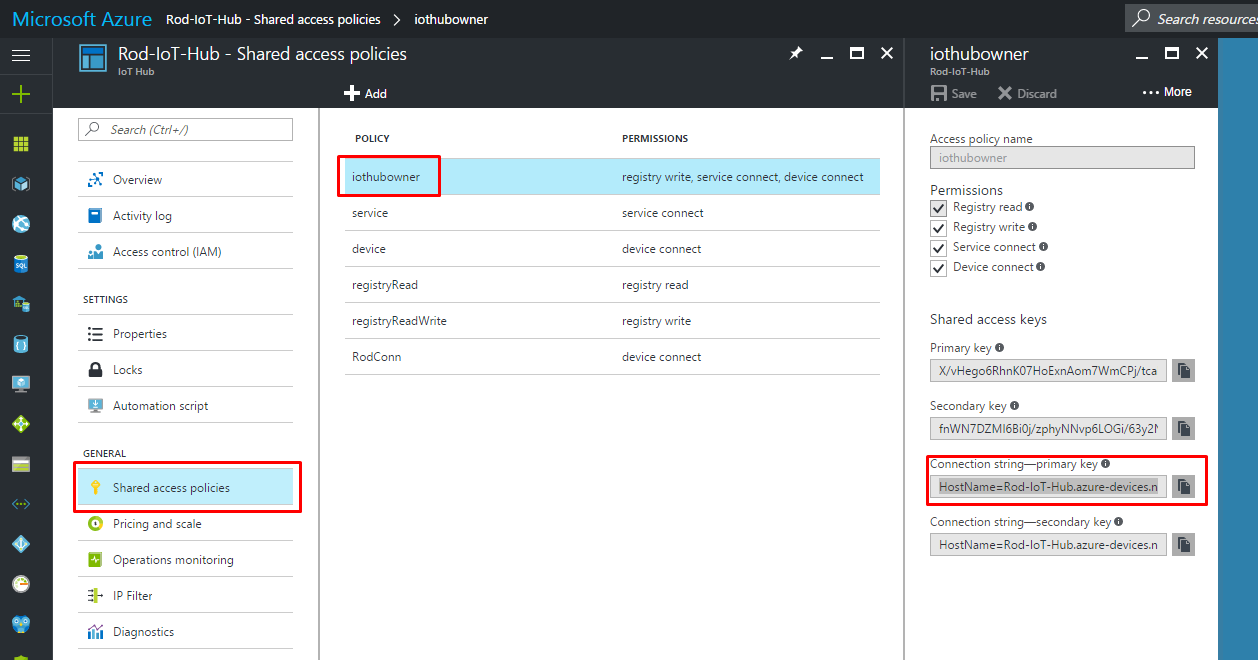
There is several ways to do this using the IoT hub SDK, I found that the simpler way for the purpose of this example, is doing it from the device explorer tool:

* Open the Device Explorer:

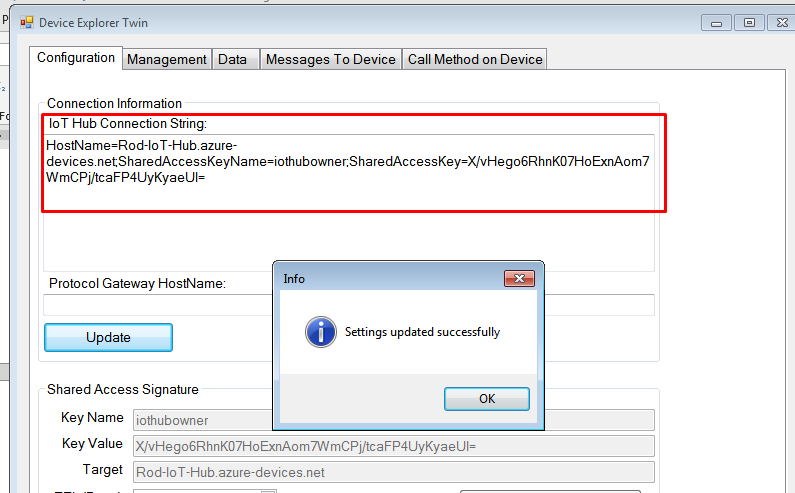


* Setup your IoT hub connection string:

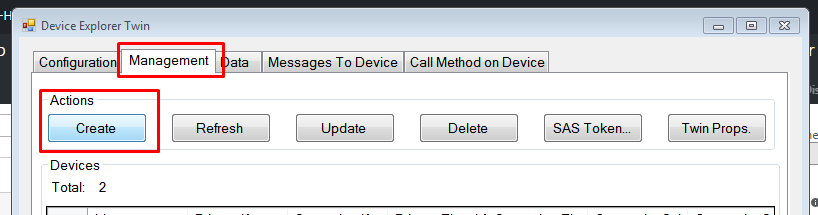
You can find the IoT hub connection string in the azure portal, then go to your hub, click on the Shared Access policies, click on the iothubowner policy, the connection string will be in the right blade:



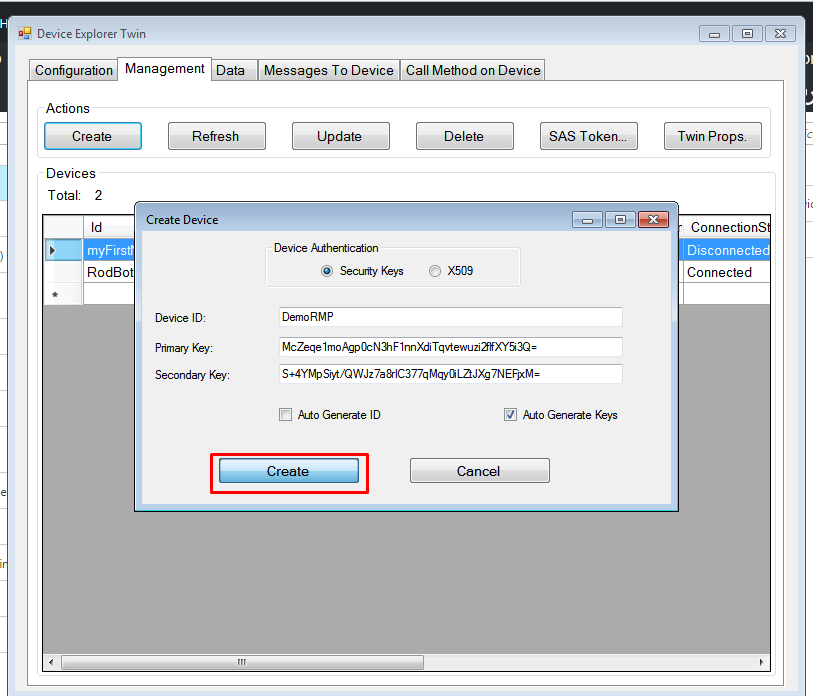
Paste the connection string in the device explorer and click on update:



* Click in the Management Tab & Click on the create Button



* Fill the Device Name and click Create:



## Create a Simulated App.

Reference: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-node-node-getstarted>

Send wind speed as a property. [Could be any telemetric data!]

* Open a SSH session to your NodeJS Server.

1. Create a folder for simulatedSensor app.
2. In your NodeJS server execute the following commands:

|  |
| --- |
| npm init  npm install azure-iot-device azure-iot-device-mqtt --save |

1. Create the file **SimulatedDevice.js**

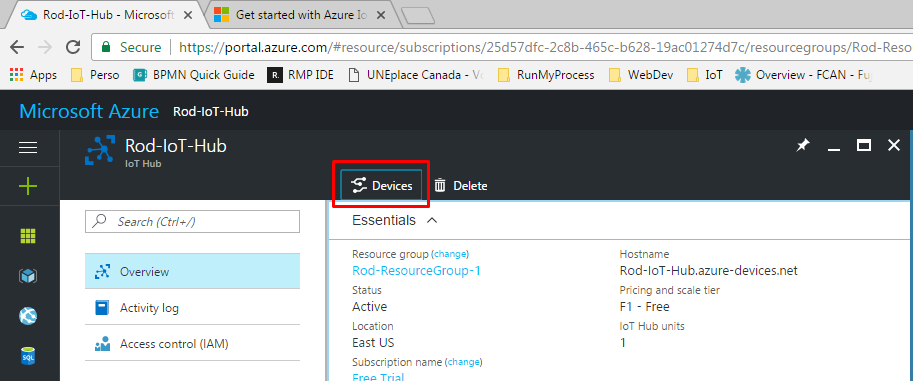
|  |
| --- |
| Note that we send the wind speed as a property in the message. This way allows the IoT hub route to identify the message and evaluate the query string. |

|  |
| --- |
| 'use strict';  var clientFromConnectionString = require('azure-iot-device-mqtt').clientFromConnectionString;  var Message = require('azure-iot-device').Message;  var connectionString = '{Device\_connection\_string}';  var client = clientFromConnectionString(connectionString);  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 windSpeed = 10 + (Math.random() \* 4);  var data = JSON.stringify({ deviceId: 'RodSensor', windSpeed: windSpeed});  var message = new Message(data);  message.properties.add("windSpeed",windSpeed);  console.log(message.properties);  console.log("Sending message: " + message.getData());  client.sendEvent(message, printResultFor('send'));  }, 10000);  }  };  client.open(connectCallback); |

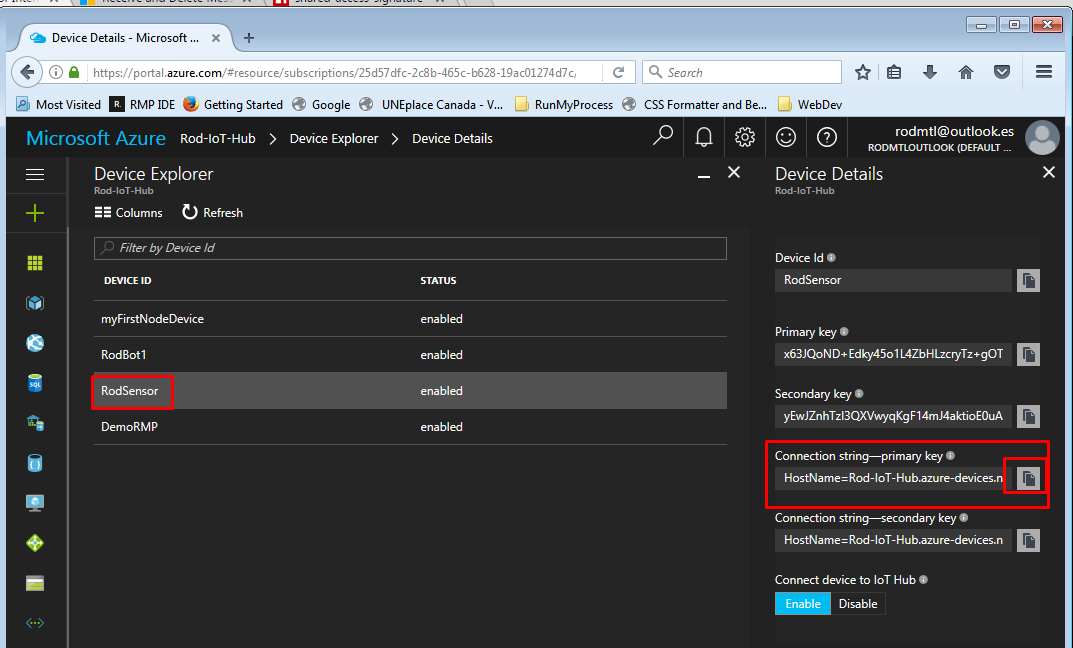
1. Replace the values in red with the Device Connection String

##### Where is the device Connection String?

- Go to the Azure iot hub, click on the devices button:



- Select the device you’ve created and in the right blade use the copy button to get the connection string



1. Save the .js file

## Create REST API call in POSTMAN

### Use NodeJS app to create a SAS token

Reference: <https://www.npmjs.com/package/shared-access-signature>

We need to create a valid SAS token to access the service bus queue, to achieve that we’re going to create use a NodeJS app

* Open a SSH session to your NodeJS Server.

1. In the NodeJS Server Execute:

|  |
| --- |
| npm install shared access signature |

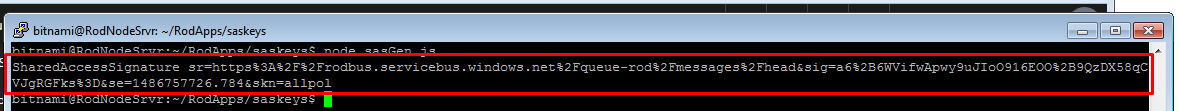
1. Create a folder saskeys
2. Create a sasGen.js file and copy the code:

|  |
| --- |
| var sas = require('shared-access-signature');  var url = 'https://{YourServiceBusName}.servicebus.windows.net/{YourQueueName}/messages/head';  var sharedAccessKeyName = '{YourSharedPolicyName}';  var sharedAccessKey = '{YourSharedPolicyKey}';  var currentDate = new Date();  var expiry = currentDate.getTime() / 1000 + 360000; // We require expiry time in seconds since epoch.  var sas = require('shared-access-signature');  var signature = sas.generateServiceBusSignature(url, sharedAccessKeyName, sharedAccessKey, expiry);  console.log(signature); |

1. Replace the values in red and save the js file. (The shared policy key is the one you created with the queue)
2. Execute the js app:

|  |
| --- |
| node sasGen.js |

1. Take note of the SAS Key, you will need it to execute the REST call.

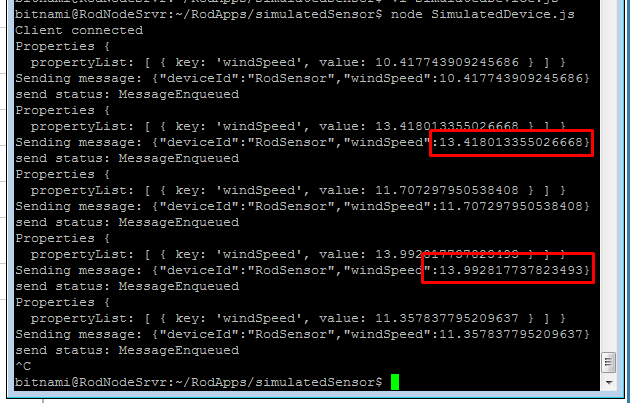


## Execute simulated App

We start to send messages from our simulated device:

1. Go back to the ssh session and excute:

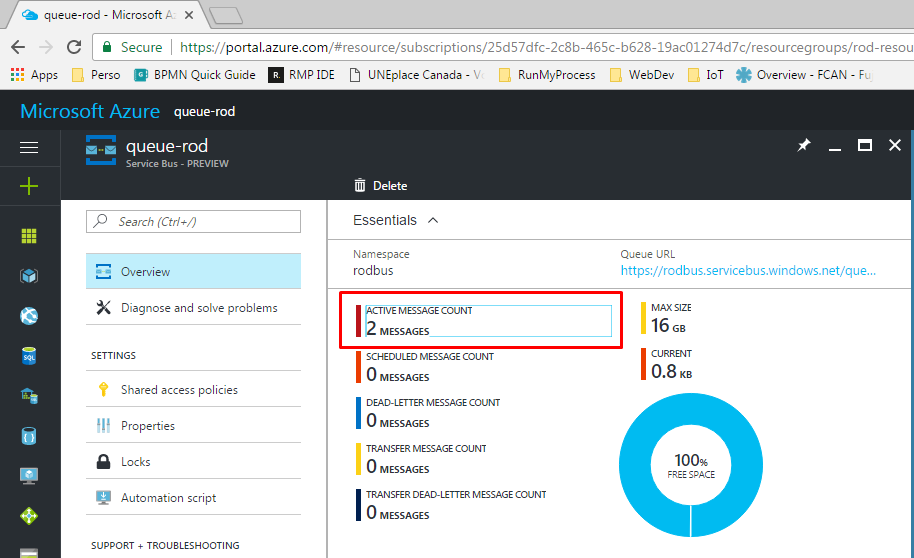
|  |
| --- |
| node SimulatedDevice.js |



Stop the client (Crtl + c)

In my case, Only 2 messages meet the rule we previously configured in the IoT hub route.

You can check if the messages are in the Queue by going to the service bus overview in azure:



## Execute REST operation to retrieve the Device to Cloud Message:

Reference: <https://docs.microsoft.com/en-us/rest/api/servicebus/receive-and-delete-message-destructive-read>

You will get the first message sent by the device. We used the **delete** method to destroy the message from the queue once we made the call. For more methods see the reference.

Method: DELETE

Request URL: https://{YourServiceBusName}.servicebus.windows.net/{YourQueueName}/messages/head?timeout=10

Authorization: No authorization

Headers:

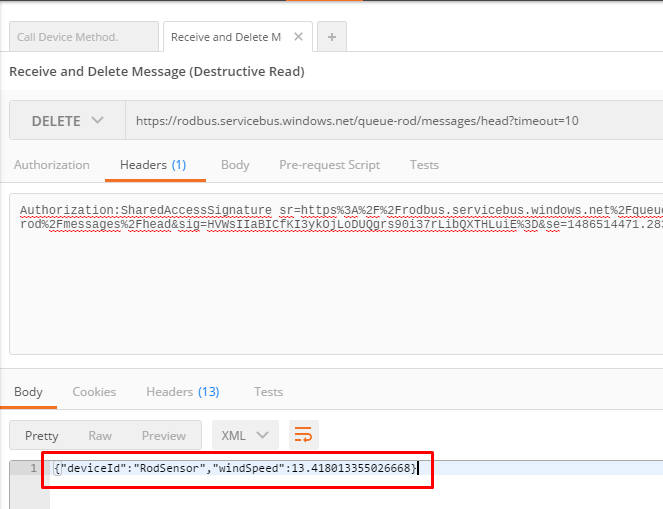
|  |  |
| --- | --- |
| Authorization | {YourSASValue} |

Body:

|  |
| --- |
|  |

The request must return:

|  |
| --- |
| {  "deviceId": "RodSensor",  "windSpeed": 13.394490554027207  } |

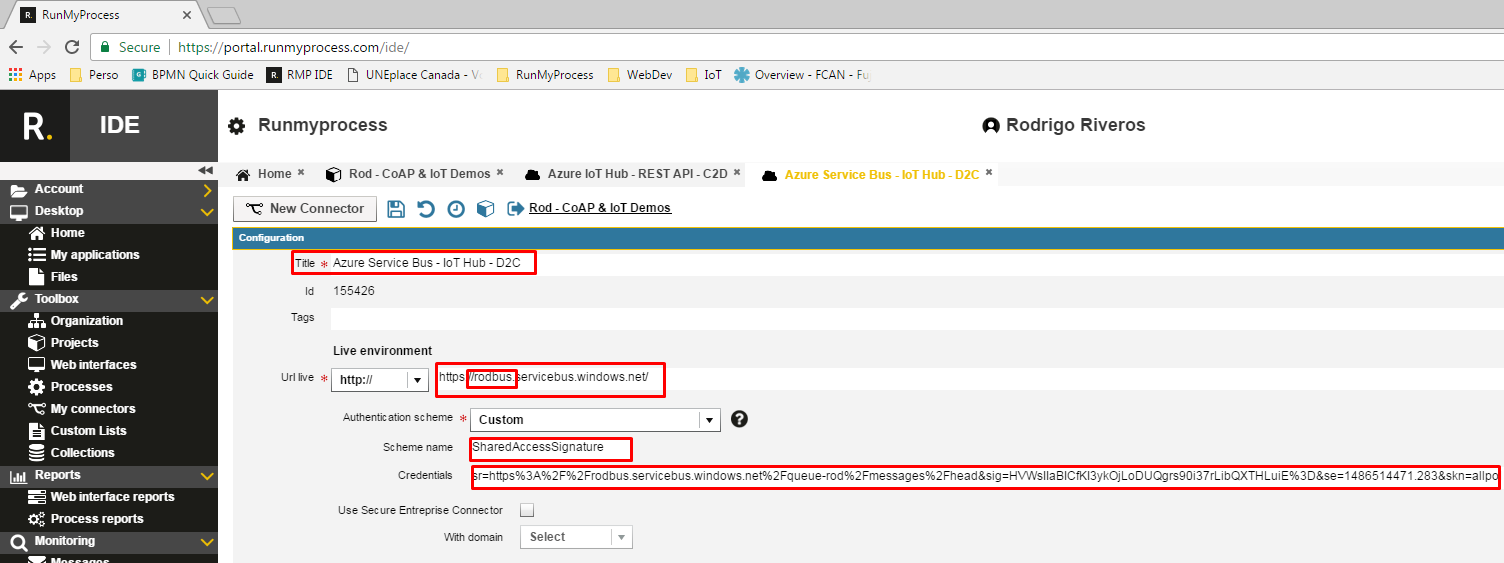


## RMP Configuration

### Create Provider

In your project create a new provider.

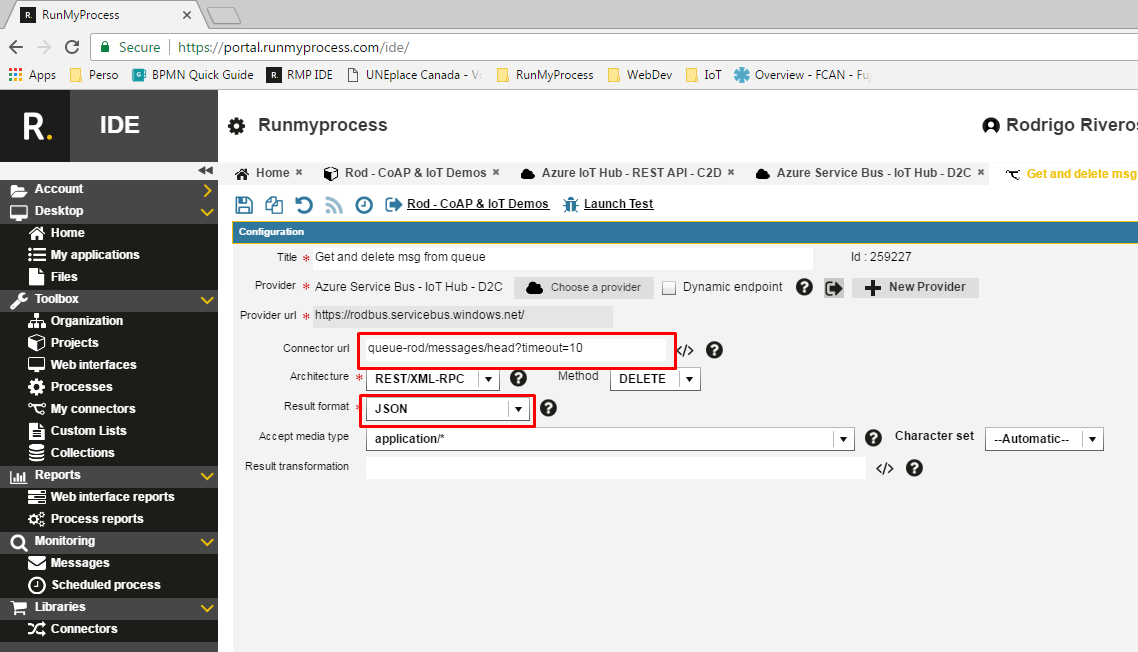
|  |  |
| --- | --- |
| Url | https://{YourBusName}.servicebus.windows.net/ |
| Authentication scheme | Custom |
| Scheme name | SharedAccessSignature |
| Credentials | {YourSASValue}  *Note: Delete the SharedAccessSignature at the start, you’ve already send it as Scheme name!!* |



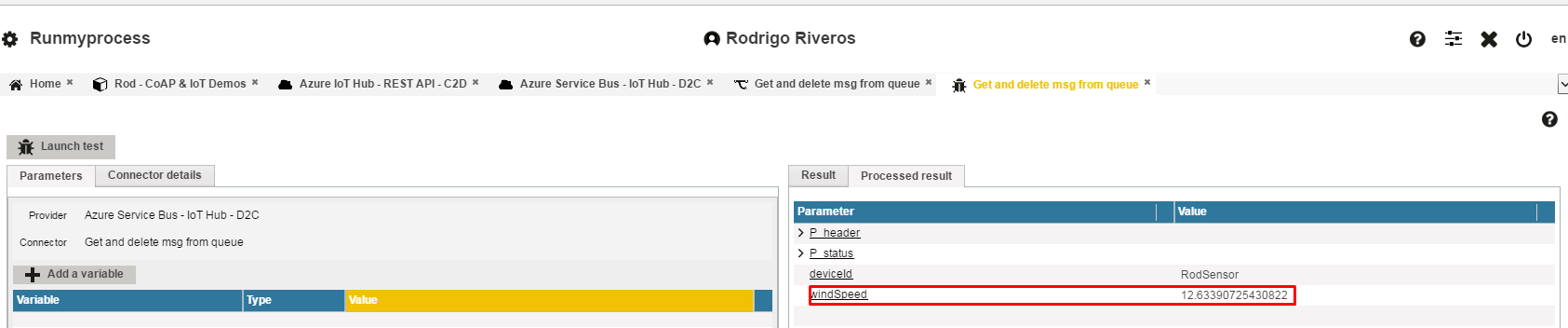
### Create Connector

Create a new connector from the provider you just created.

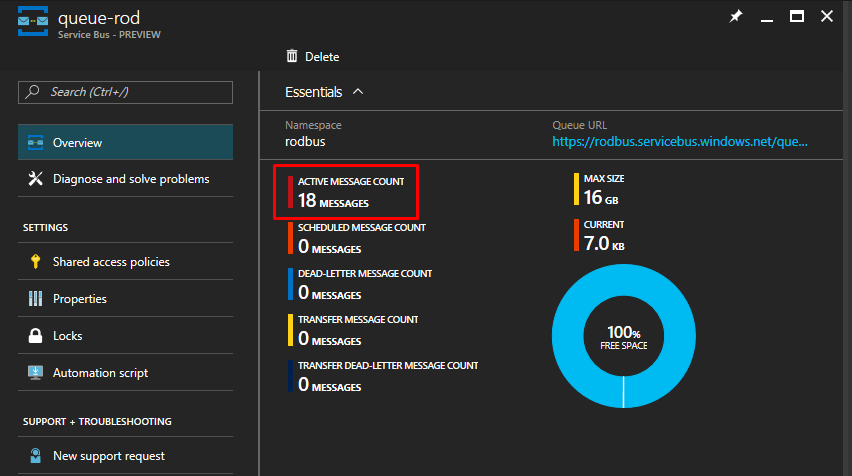
|  |  |
| --- | --- |
| Connector URL | {YourQueueName}/messages/head?timeout=10 |
| Architecture | REST/XML-RPC |
| Method | POST |
| Result Format | DELETE |
| Accept Media Type | Application/\* |
| Content |  |
| Content Type |  |
| Result transformation |  |

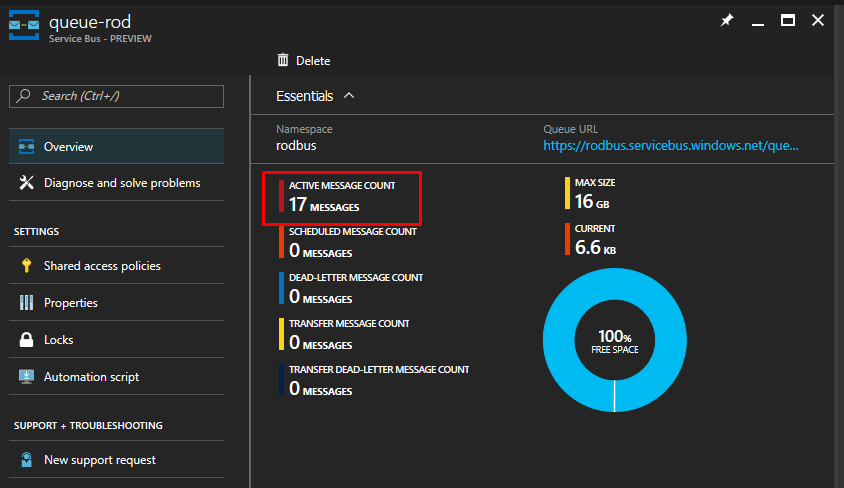


Test:



Then when the test succeed, you can go back to the azure portal open the queue and check that the number of messages diminish by each time the test is executed.





End--