

Final Project

Azure DataBox Emulator

TURI, François

Deep Azure@McKesson

Dr. Zoran B. Djordjević

Introduction

- Microsoft is introducing a DataBox designed to upload hundreds of terrabytes from client to Azure by shipping basically a rugged computer

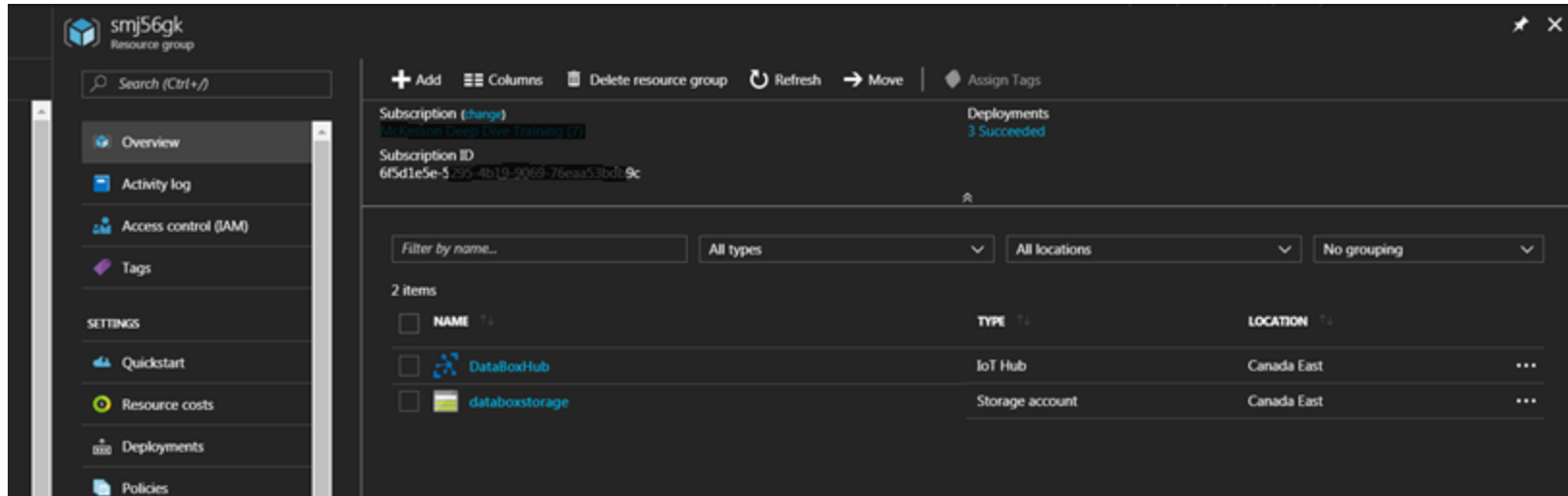
Announcing the Azure Data Box preview



- Well, we are not going to play with that toy. We will create a simulator using IOT technology, geolocalisation and an emulator that will carry a single snowman photo.

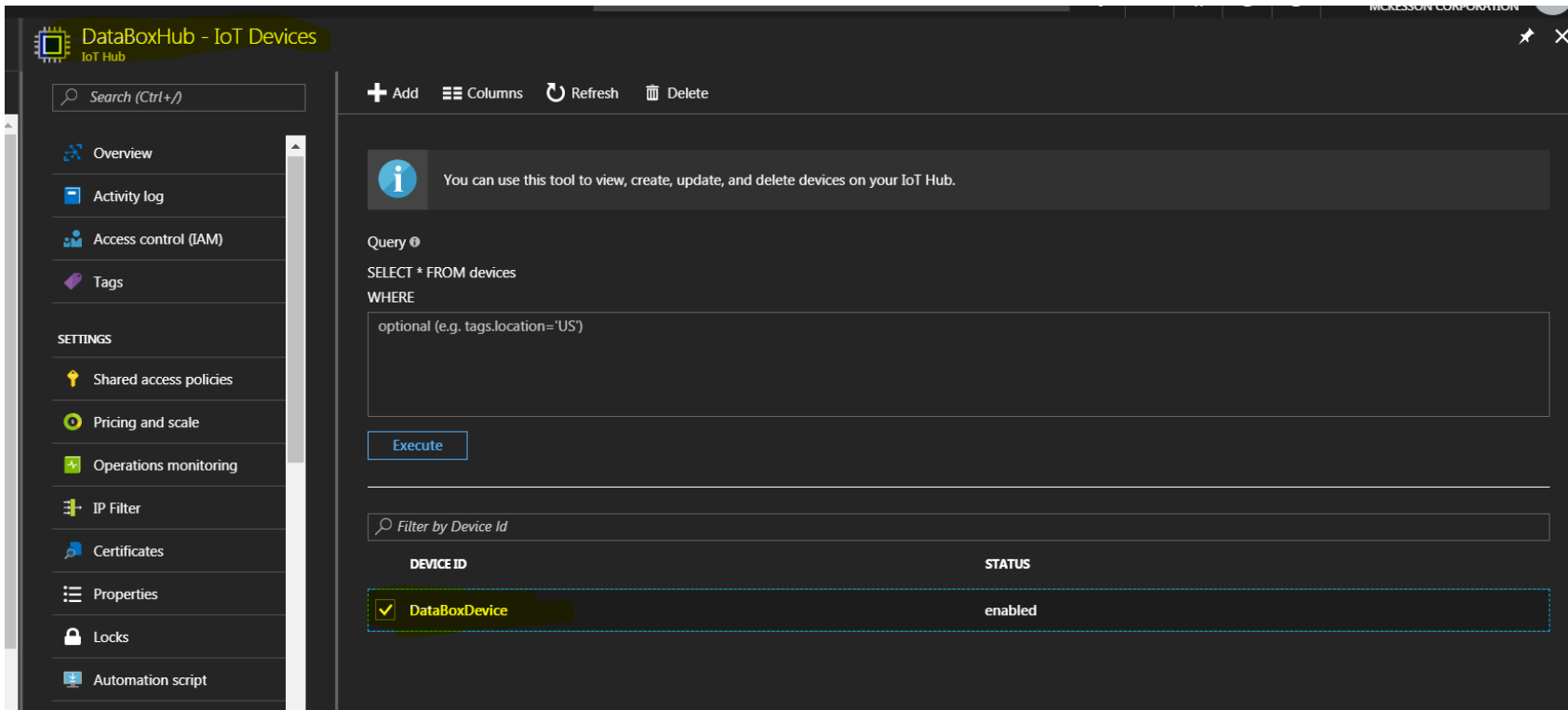
Azure Ressource-Group

- We are using two ressources:



IoT DataBoxHub

- We publish a IoT Device inside the IoT DataBox Hub
- For the project we will use either the IOT hub AccessKey or The IOT device, depending of the library used



The screenshot displays the IoT DataBoxHub interface. On the left, a sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, and a SETTINGS section with Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Certificates, Properties, Locks, and Automation script. The main panel features a search bar, action buttons (Add, Columns, Refresh, Delete), and an information message. Below this is a query editor with the text: "Query ⓘ", "SELECT * FROM devices", "WHERE", and "optional (e.g. tags.location='US')". An "Execute" button is present. At the bottom, a table lists devices with columns "DEVICE ID" and "STATUS". One device, "DataBoxDevice", is listed with a checked checkbox and a status of "enabled".

DEVICE ID	STATUS
<input checked="" type="checkbox"/> DataBoxDevice	enabled

IoT Storage

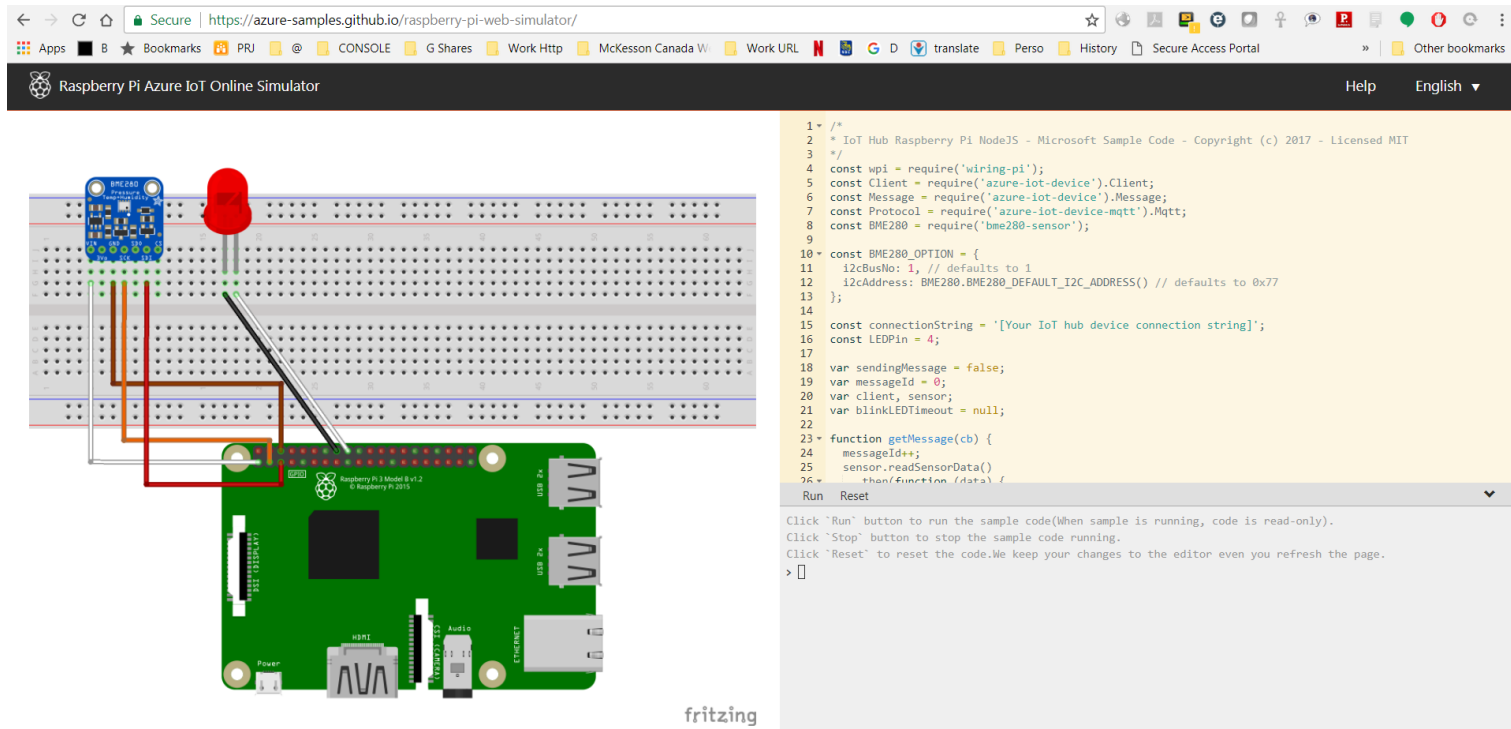
- We also need a IoT Storage account, with a Blob Service, with a Blob Container (databoxpayload)
- The Blobfile databoxfile is created automatically by the code

The screenshot displays the Azure portal interface for a Blob service. The breadcrumb navigation at the top indicates the path: `smj56gk > databoxstorage > Blob service > databoxpayload`. The left-hand 'Essentials' pane shows a search bar for containers and a list containing the `databoxpayload` container. The main content area is titled `databoxpayload` and shows the container's location and a search bar for blobs. Below this, a table lists the blobs within the container.

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
<code>databoxfile</code>	2018-02-11, 11:20:50 a.m.	Cool (Inferred)	Append blob	41.53 KiB	Available

The Raspberry-Pi emulator

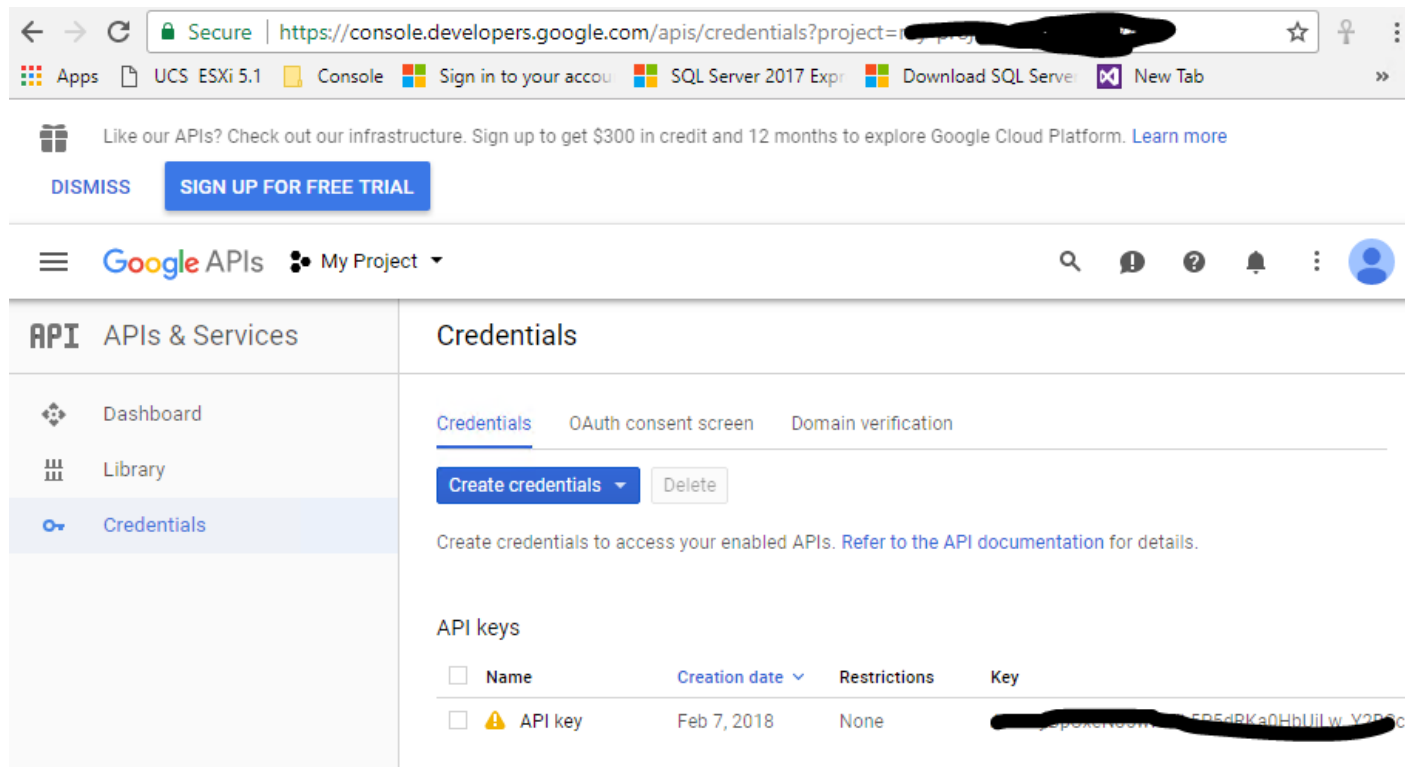
- There is a very nice looking raspberry emulator provided by Azure as git repository:
We will install, and adapt this emulator to create a `functioning` databox



Google Maps

- We also need a google api Keys using this procedure
<https://developers.google.com/maps/documentation/javascript/get-api-key>

And visible after here:

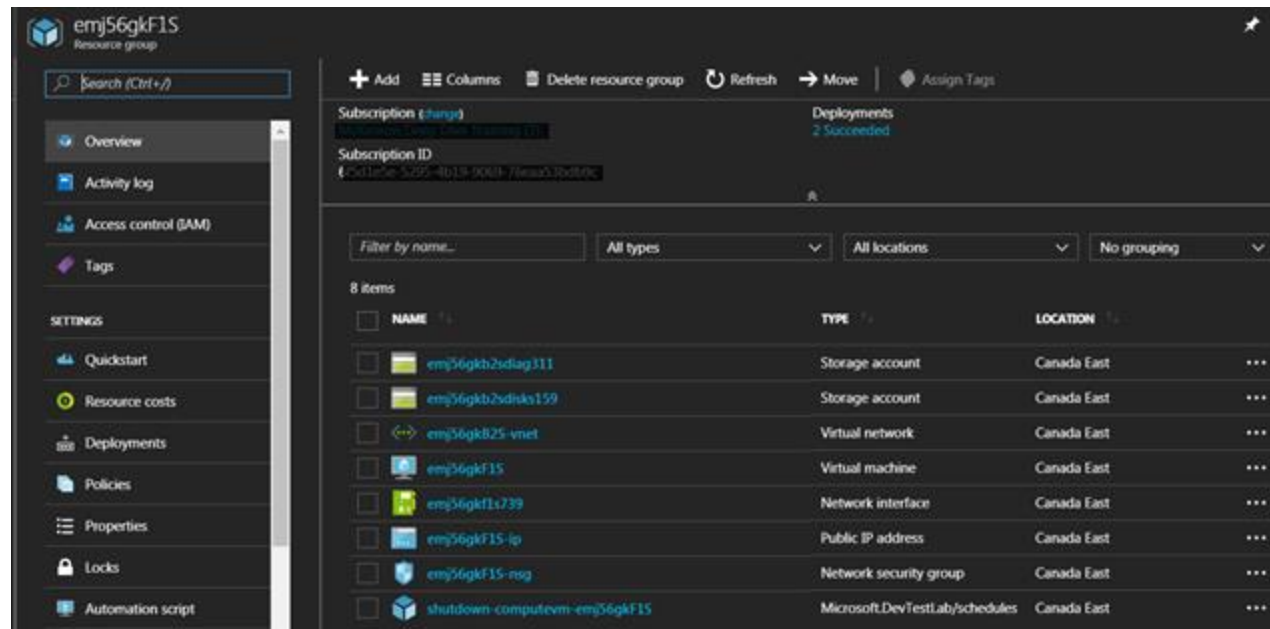


The screenshot shows the Google Cloud Platform console interface. The top navigation bar includes the Google APIs logo and a 'My Project' dropdown. The left sidebar shows the 'APIs & Services' menu with options for Dashboard, Library, and Credentials. The main content area is titled 'Credentials' and includes tabs for 'Credentials', 'OAuth consent screen', and 'Domain verification'. A 'Create credentials' button is visible. Below this, a table lists the API keys. The table has columns for 'Name', 'Creation date', 'Restrictions', and 'Key'. One key is listed with the name 'API key', a creation date of 'Feb 7, 2018', and no restrictions. The key itself is partially redacted with a black bar.

Name	Creation date	Restrictions	Key
API key	Feb 7, 2018	None	AIzaSyA...Y2PSc

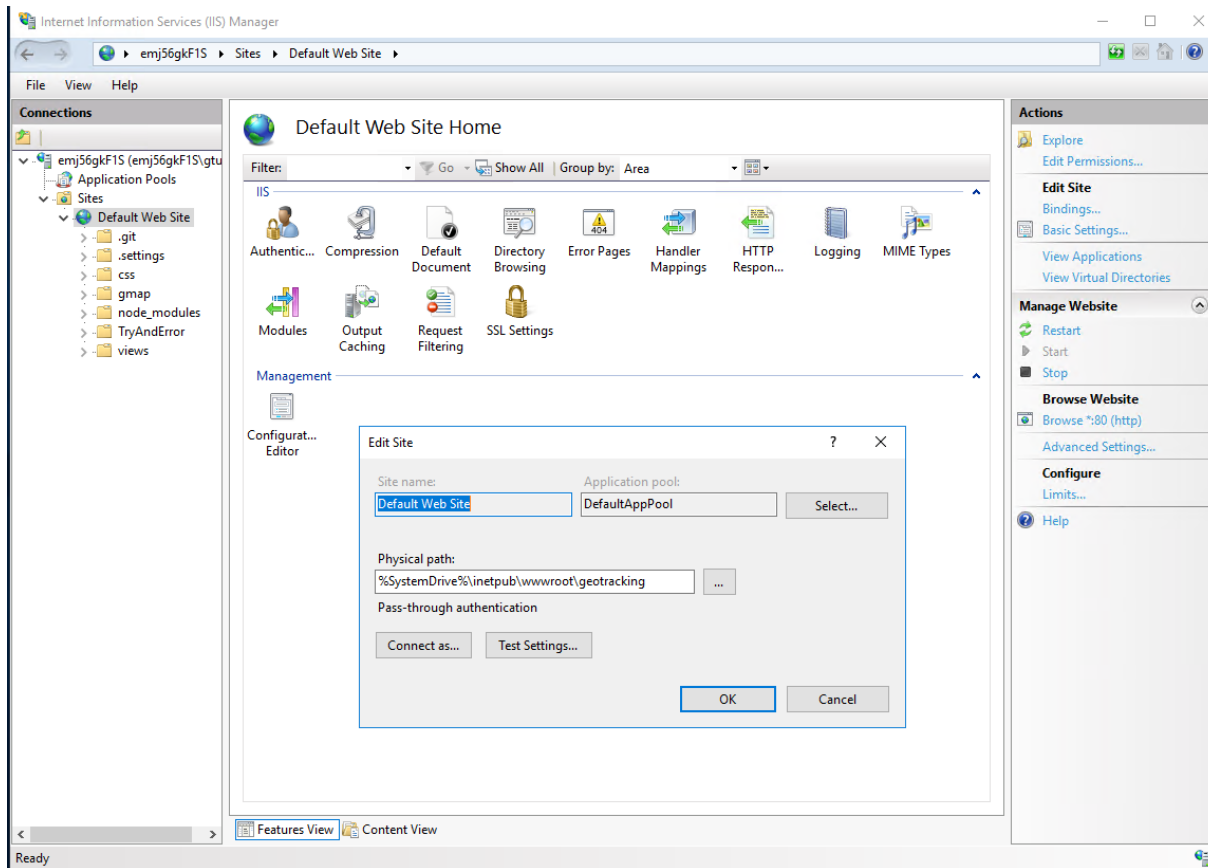
VM, Node.js host

- At last we need a machine to host the three Node.js server and also the HTTP server
- As we use Eclipse to develop the code, we decided to use a Windows10 VM on Azure cloud
- It probably doesnt matter where the Node.JS is residing. I do believe that a redhat linux server will do the job with an Apache http server.



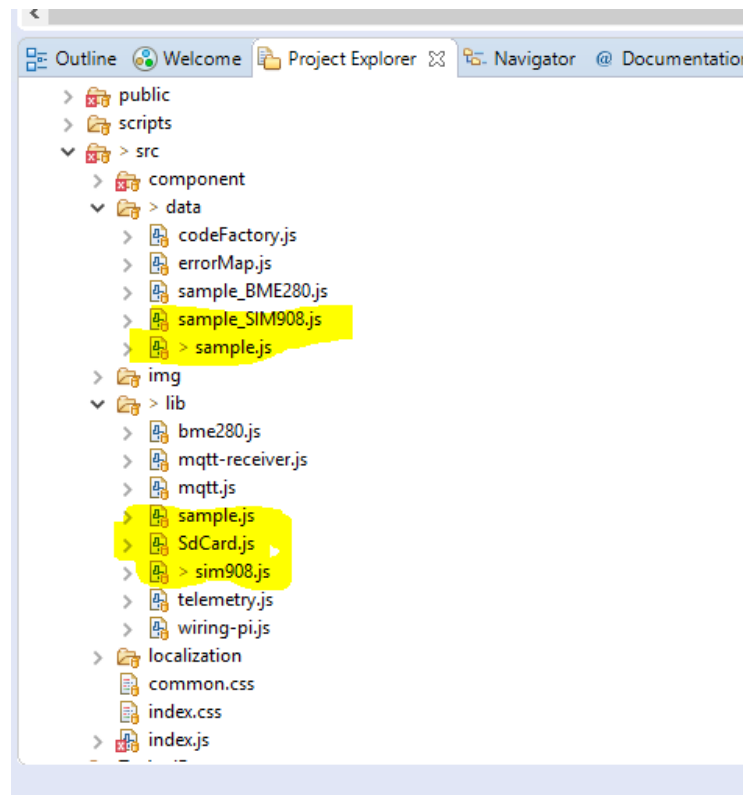
HTTP server

- We need an HTTP server to host the static pages of the tracker node.js application and the icons for the console node.js application
- For simplicity, we used IIS. The Node.js and IIS use the same root



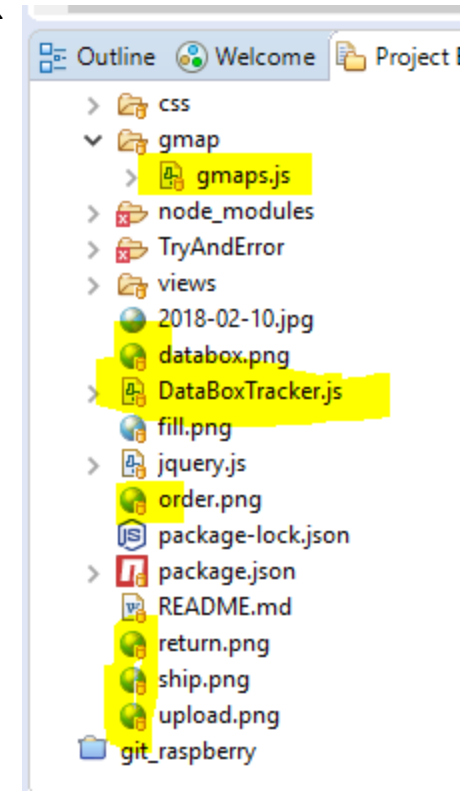
Raspberry-Pi Emulator Technology

- The emulator is in fact a Native-react project **hosted** on top of a node.js
- The right part of the window is actually the heart of this product and is basically a pseudo language (using **pseudo** node.js syntax). This part will be largely changed to fit our need



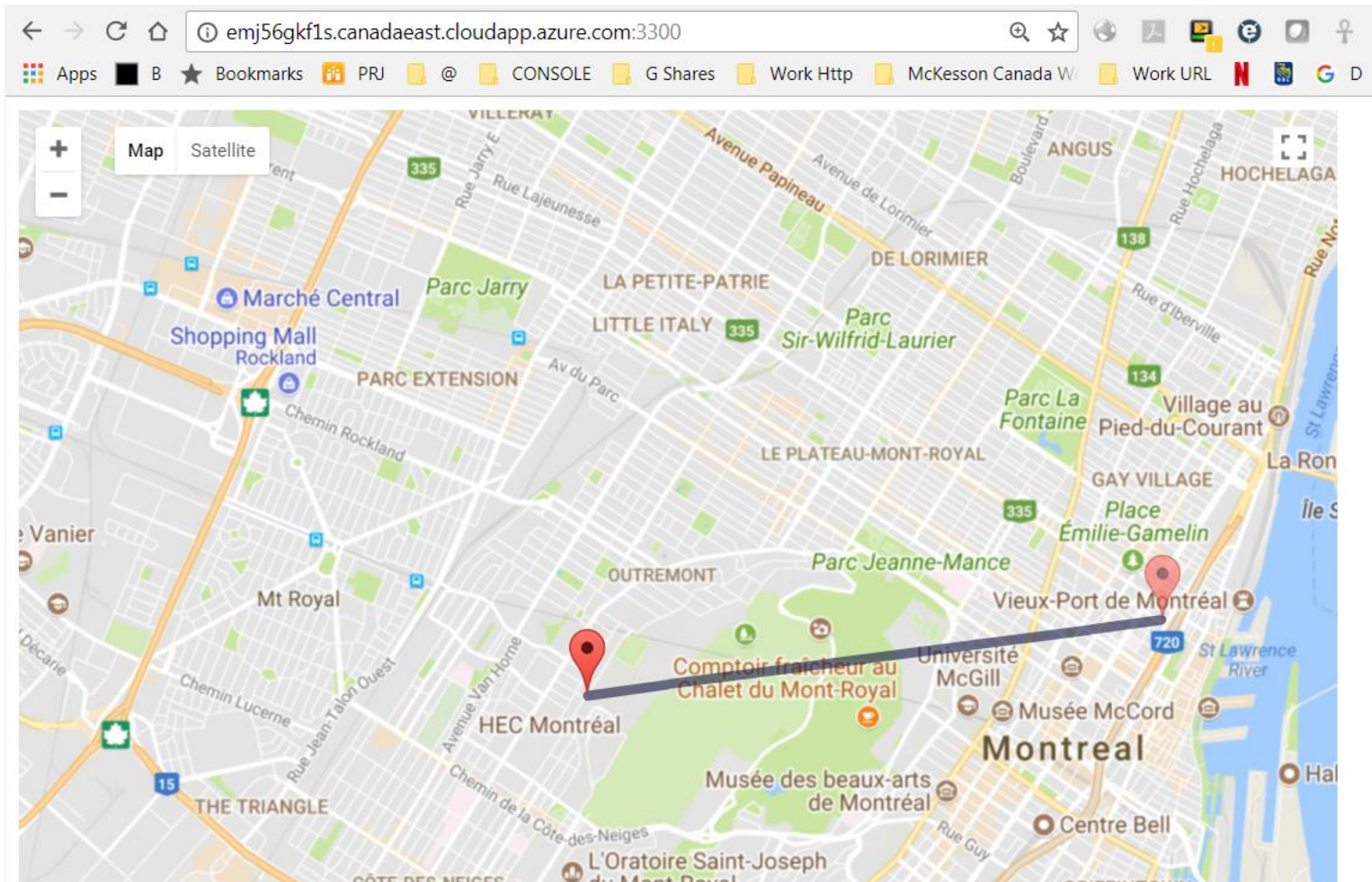
The Tracker

- We will use the Gmaps.js library wrapper to generate a google map with marker. Nice library 😊
- The rest of the project is a basic Node.js Express (http) & Mqtt & Azure storage
- The Azure storage is due to the fact that the raspberry written in Native-React does not support to use node.js azure library. So we have to hop on that node.js code to forward the file from the simulated Node.js disk
- We also map the IIS http server on the same directory for static images (the fill, upload icons..)



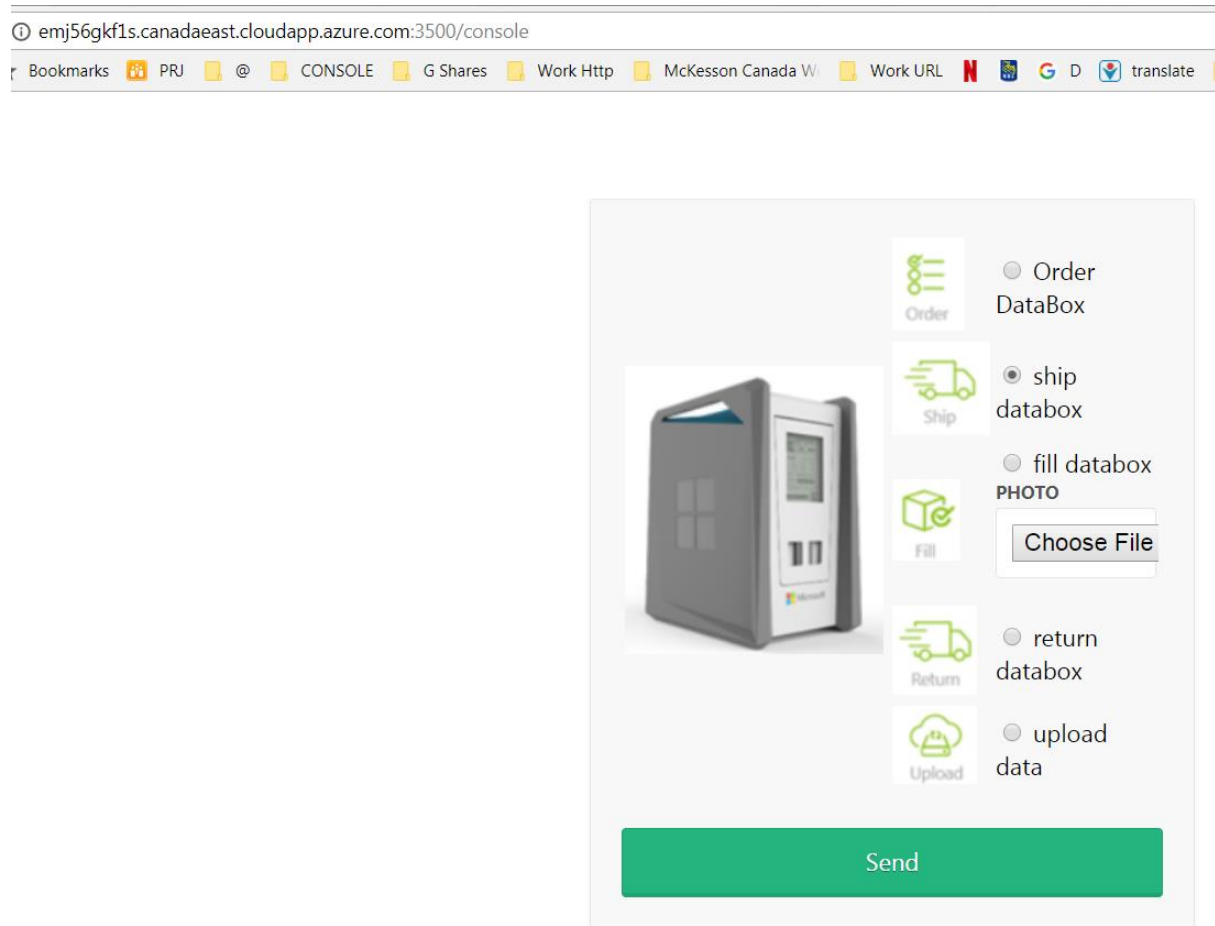
Tracker in action

- The light mark show the origin of the databox. Map can be zoomed in



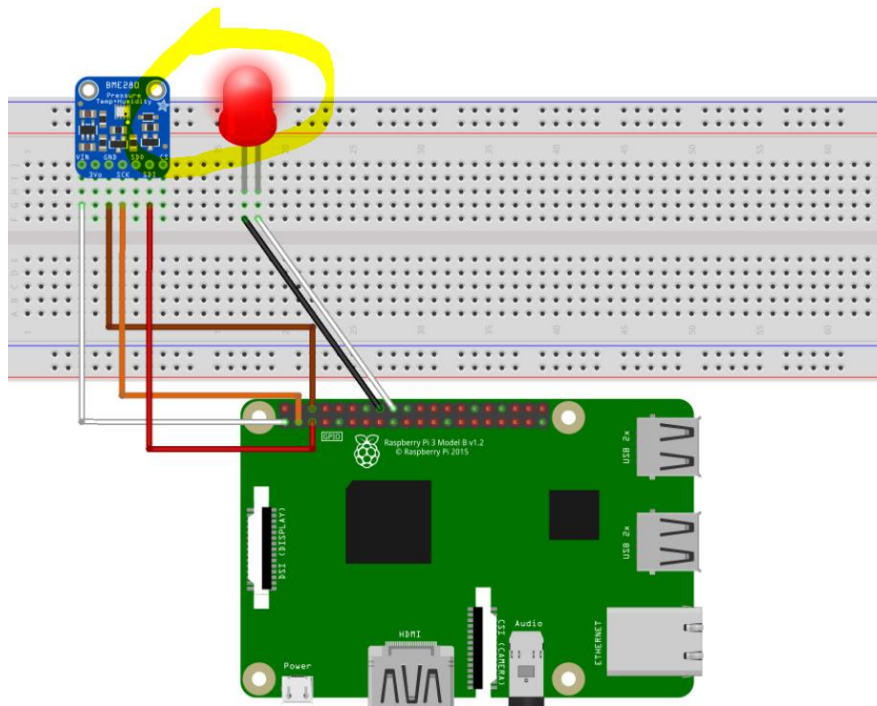
DataBox Console

- We also implemented a DataBox console (simple html forms allowing file to be uploaded). Node.js doesnt allow it to be simple...



Emulator in action

- While the databox is moving, the Led is flashing
- Console is also showing what is happening



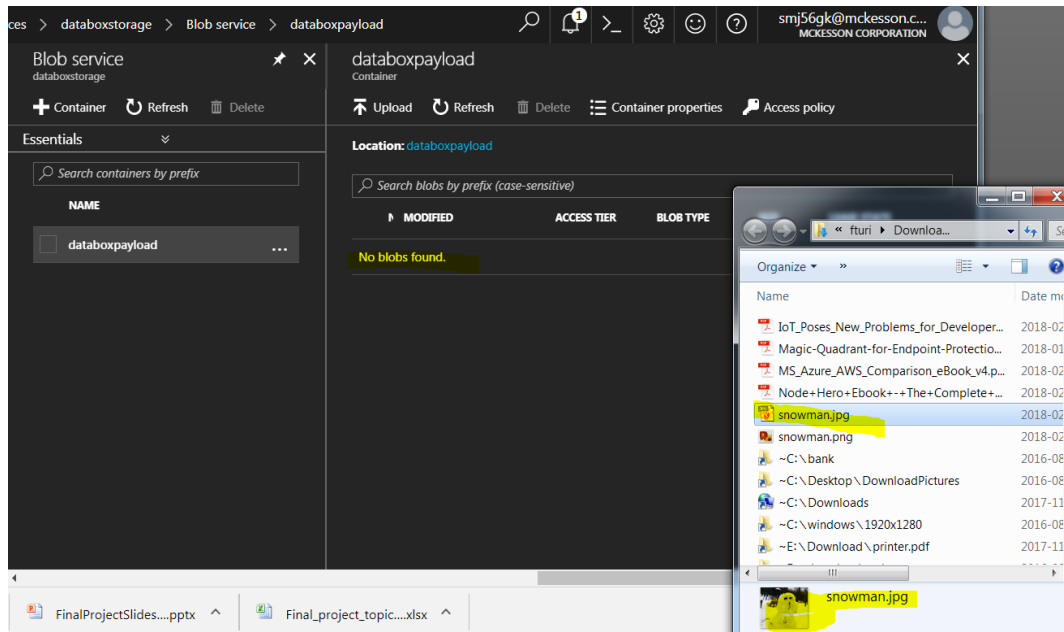
```
1  /*
2  * IoT Hub Raspberry Pi
3  */
4  const wpi = require('wpi');
5  const Client = require('wpi-client');
6  const Message = require('wpi-message');
7  const Protocol = require('wpi-protocol');
8  //const BME280 = require('bme280');
9  const SIM908 = require('sim908');
10 const SDCARD = require('sdcard');
11 const IOTHUB = require('iothub');
12
13 const SIM908_OPTION = {
14   i2cBusNo: 1, // default
15   i2cAddress: SIM908.SIM908_I2C_ADDRESS
16 };
17
18 const SDCARD_OPTION = {
19   i2cBusNo: 2, // default
20   i2cAddress: SDCARD.SDCARD_I2C_ADDRESS
21 };
22
```

Stop Reset

order is: fill
>
file is:[object Object],jspon
>
result: ok(4)
>
msg
>
order is: return
>
result: ok(5)
~ Π

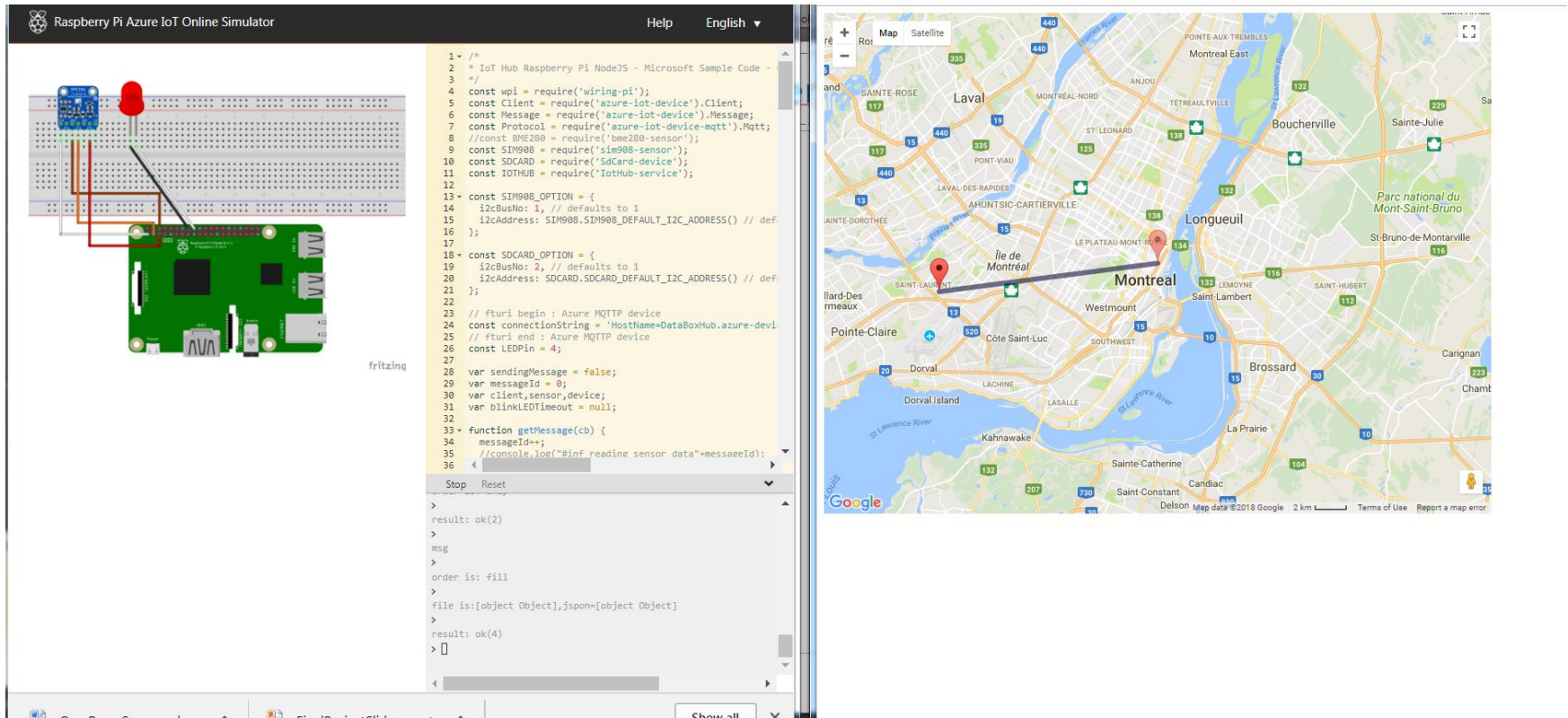
Save Private SnowMan

- Our mission is to upload the snowman to the databox, fly it through montreal on teh emulator SdCard and deposit it on Azure



Order, Shipping & FillUp

- In the first step we order the DataBox, wait for arrival and Fill it up

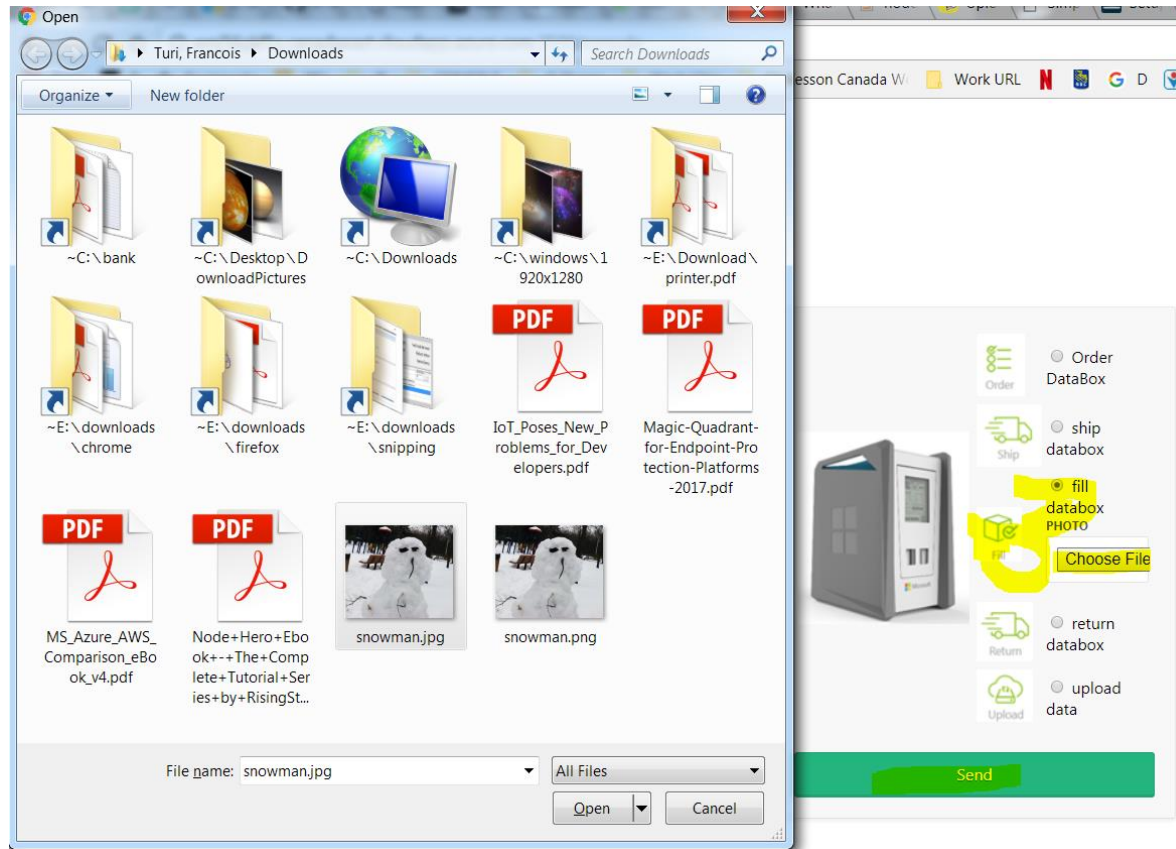


The screenshot displays the Raspberry Pi Azure IoT Online Simulator interface. On the left, a circuit diagram shows a Raspberry Pi connected to a breadboard with a red LED and a blue sensor module. The central pane contains JavaScript code for an IoT Hub Raspberry Pi NodeJS application, which includes configuration for Azure IoT Hub, MQTT protocol, and a sensor module. The right pane shows the execution output, indicating successful connection and data transmission. On the far right, a map of Montreal is shown with a red pin marking the location of the DataBox.

```
1 /*  
2  * IoT Hub Raspberry Pi NodeJS - Microsoft Sample Code -  
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  const SIM900 = require('sim900-sensor');  
10 const SD CARD = require('sdCard-device');  
11 const IOTHUB = require('IoTHub-service');  
12  
13 const SIM900_OPTION = {  
14   i2cBusNo: 1, // defaults to 1  
15   i2cAddress: SIM900.SIM900_DEFAULT_I2C_ADDRESS() // def.  
16 };  
17  
18 const SD CARD_OPTION = {  
19   i2cBusNo: 2, // defaults to 1  
20   i2cAddress: SD CARD.SD CARD_DEFAULT_I2C_ADDRESS() // def.  
21 };  
22  
23 // fturi begin : Azure MQTT device  
24 const connectionString = 'HostName=DataBoxHub.azure-devi  
25 // fturi end : Azure MQTT device  
26 const LEDPin = 4;  
27  
28 var sendingMessage = false;  
29 var messageId = 0;  
30 var client,sensor,device;  
31 var blinkLEDTIMEOUT = null;  
32  
33 * function getMessage(cb) {  
34   messageId++;  
35   //console.log("#Inf reading sensor data="+messageId);  
36  
37   >  
38   result: ok(2)  
39   >  
40   msg:  
41   order is: fill  
42   >  
43   file is:[object Object],json=[object Object]  
44   >  
45   result: ok(4)  
46   >  
47   >  
48   >
```

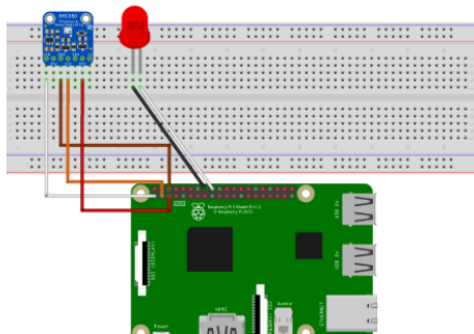

Console Fillingup

- We also embark Private Snowman on-board

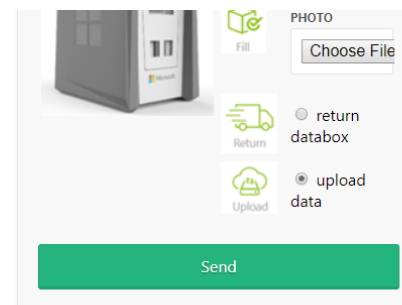


Return and Upload

- We are flying back to Viger datacenter and order an upload

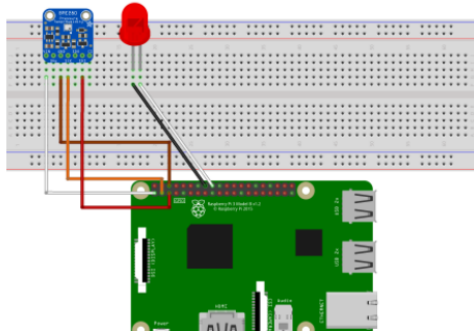


```
2  * IoT Hub Raspberry Pi NodeJS - Microsoft Sample Code - C
3  */
4  const wpi = require('wiring-pi');
5  const Client = require('azure-iot-device').Client;
6  const Message = require('azure-iot-device').Message;
7  const Protocol = require('azure-iot-device-mqtt').Mqtt;
8  //const BME280 = require('bme280-sensor');
9  const SIM908 = require('sim908-sensor');
10 const SDCARD = require('SdCard-device');
11 const IOTHUB = require('IotHub-service');
12
13 const SIM908_OPTION = {
14   i2cBusNo: 1, // defaults to 1
15   i2cAddress: SIM908.SIM908_DEFAULT_I2C_ADDRESS() // defa
16 };
17
18 const SDCARD_OPTION = {
19   i2cBusNo: 2, // defaults to 1
20   i2cAddress: SDCARD.SDCARD_DEFAULT_I2C_ADDRESS() // defa
21 };
22
23 // fturi begin : Azure MQTT device
24 const connectionString = 'HostName=DataBoxHub.azure-devic
25 // fturi end : Azure MQTT device
```

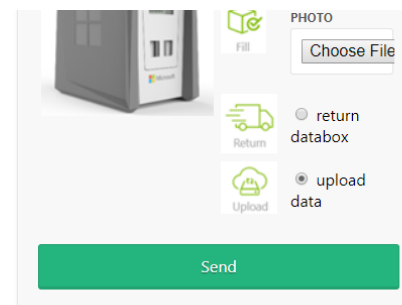
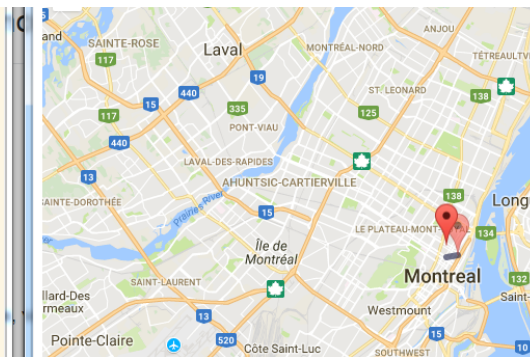


Return and Upload

- We are flying back to Viger datacenter and order an upload

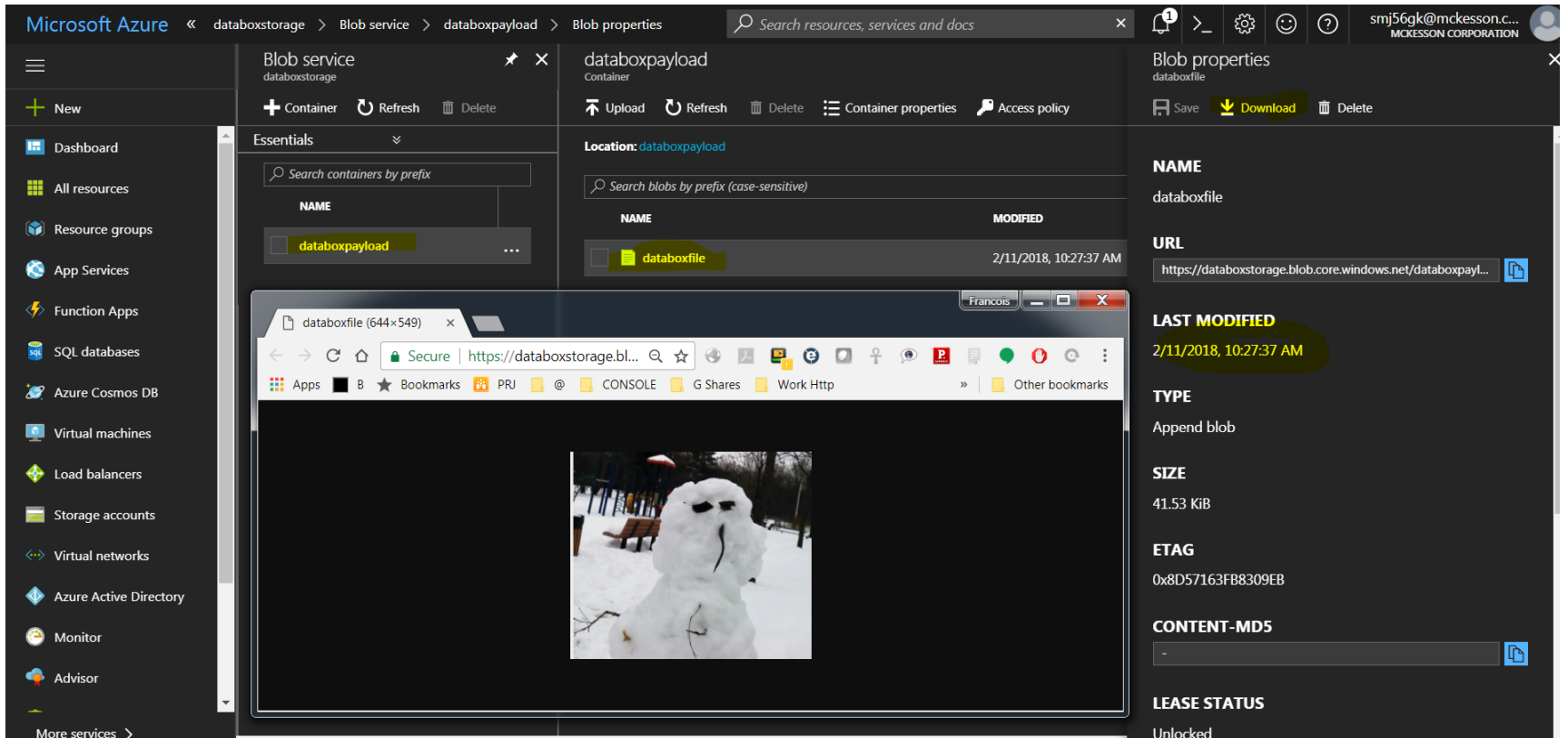


```
2  * IoT Hub Raspberry Pi NodeJS - Microsoft Sample Code - C
3  */
4  const wpi = require('wiring-pi');
5  const Client = require('azure-iot-device').Client;
6  const Message = require('azure-iot-device').Message;
7  const Protocol = require('azure-iot-device-mqtt').Mqtt;
8  //const BME280 = require('bme280-sensor');
9  const SIM908 = require('sim908-sensor');
10 const SDCARD = require('SdCard-device');
11 const IOTHUB = require('IotHub-service');
12
13 const SIM908_OPTION = {
14   i2cBusNo: 1, // defaults to 1
15   i2cAddress: SIM908.SIM908_DEFAULT_I2C_ADDRESS() // defa
16 };
17
18 const SDCARD_OPTION = {
19   i2cBusNo: 2, // defaults to 1
20   i2cAddress: SDCARD.SDCARD_DEFAULT_I2C_ADDRESS() // defa
21 };
22
23 // fturi begin : Azure MQTT device
24 const connectionString = 'HostName=DataBoxHub.azure-devic
25 // fturi end : Azure MQTT device
```



Mission Accomplished

- Mr Snowman is safe:



YouTube URLs, GitHub URL, Last Page

- Two minute (short):
<https://youtu.be/yg7ieQeYTSo>
- 15 minutes (long):
<https://youtu.be/Es7b7VV7s78>
- GitHub Repository with all artifacts:
 - Raspbery emulator (branch from Azure code):
<https://github.com/sysarchitek/DataBox.raspberry-pi-web-simulator>
 - The console:
<https://github.com/sysarchitek/DataBox.Console>
 - The Tracker and Storage hopper-uploader
<https://github.com/sysarchitek/DataBox.geotracking>
 - Main Documentation project
<https://github.com/sysarchitek/DataBox-IOT-Emulator>