AGGREGATING DATA ACROSS DEVICES FOR IOT CENTRAL

Using Node.js to query a database, aggregate data and send data back to IoT Central

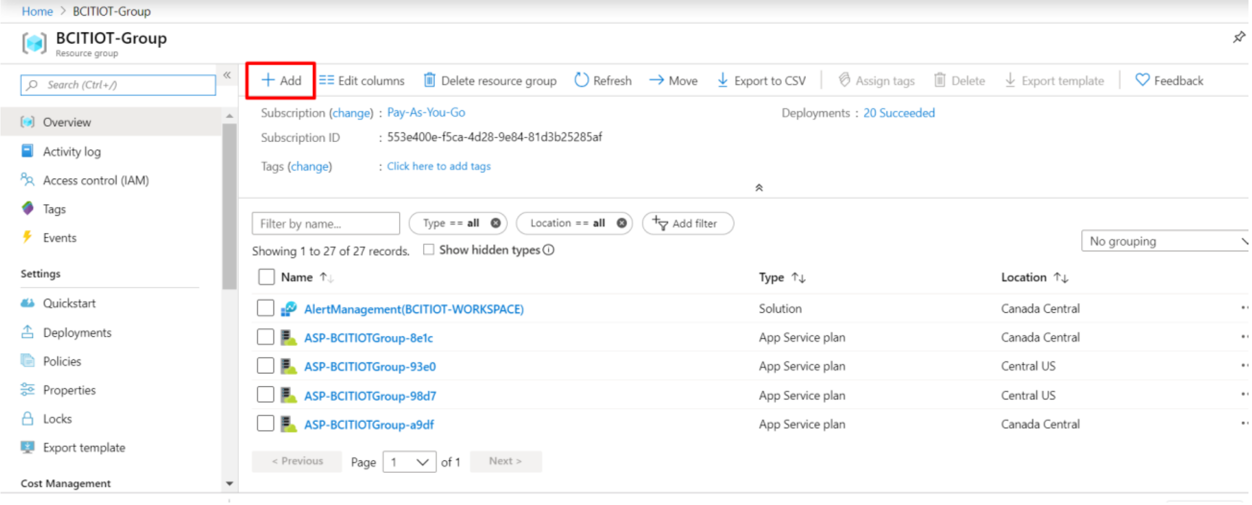
Abstract

Azure's IoT Central platform currently has a 1-widget-to-1-device limitation. When thinking of use cases, sometimes you will need to aggregate device information for display on a widget. This document shows you how to do just that.

Justin Cervantes
Cervantes.jfa@gmail.com

Exporting all data sent to IoT Central into a Cosmos DB (SQL-like database)

Get an Event Hub



Subscription (change) : Pay-As-You-Go

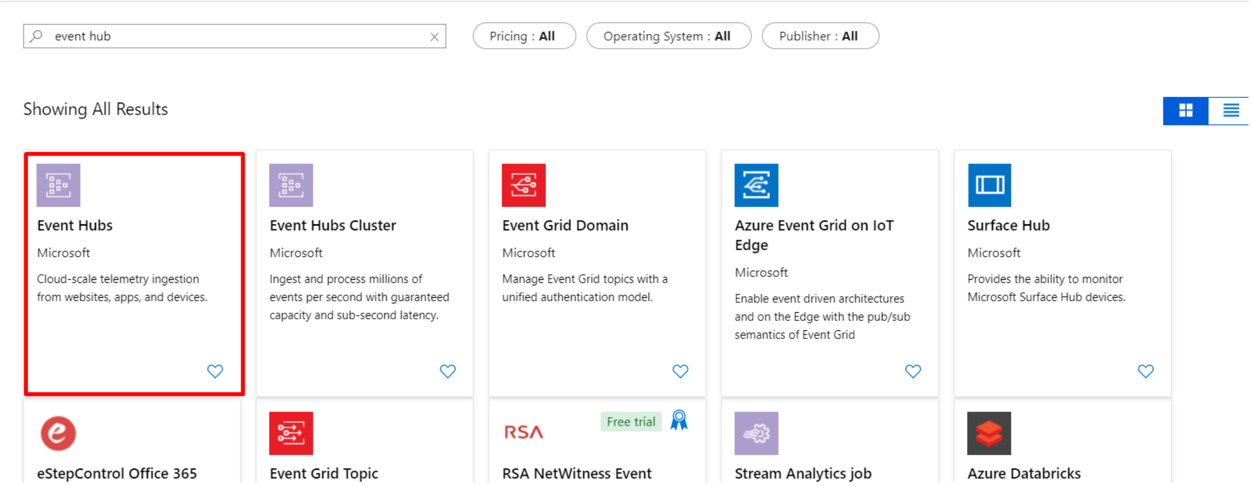
Subscription ID : 553e400e-f5ca-4d28-9e84-81d3b25285af

Tags (change) : Click here to add tags

Deployments : 20 Succeeded

Showing 1 to 27 of 27 records. □ Show hidden types

Name	Type	Location
AlertManagement(BCITIOT-WORKSPACE)	Solution	Canada Central
ASP-BCITIOTGroup-8e1c	App Service plan	Canada Central
ASP-BCITIOTGroup-93e0	App Service plan	Central US
ASP-BCITIOTGroup-98d7	App Service plan	Central US
ASP-BCITIOTGroup-a9df	App Service plan	Canada Central



Event Hubs
Microsoft
Cloud-scale telemetry ingestion from websites, apps, and devices.

Event Hubs Cluster
Microsoft
Ingest and process millions of events per second with guaranteed capacity and sub-second latency.

Event Grid Domain
Microsoft
Manage Event Grid topics with a unified authentication model.

Azure Event Grid on IoT Edge
Microsoft
Enable event driven architectures and on the Edge with the pub/sub semantics of Event Grid

Surface Hub
Microsoft
Provides the ability to monitor Microsoft Surface Hub devices.

eStepControl Office 365

Event Grid Topic

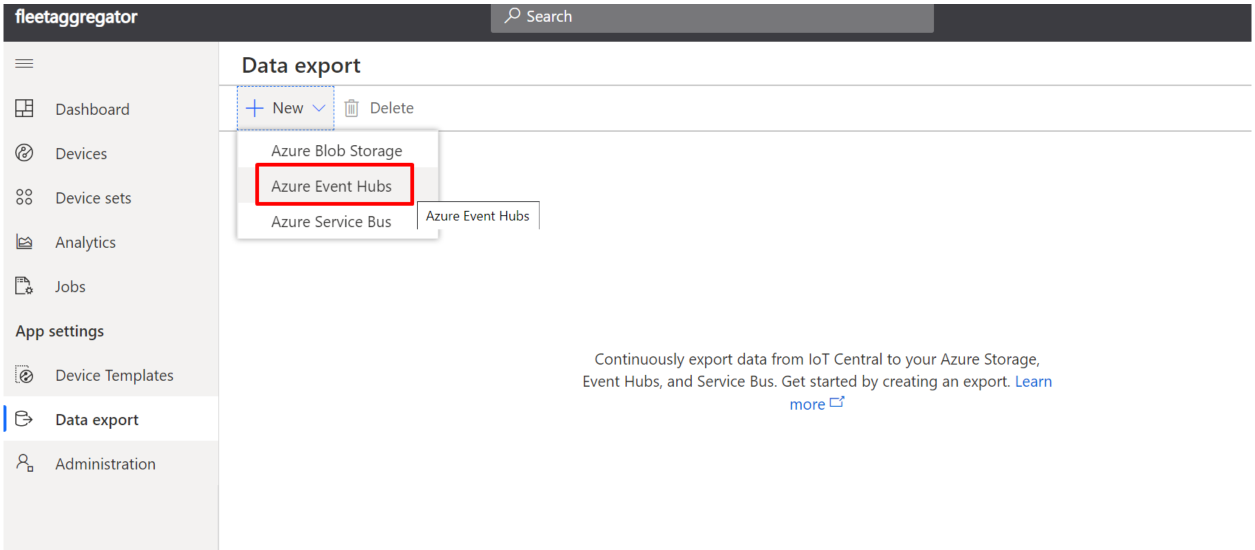
RSA NetWitness Event

Stream Analytics job

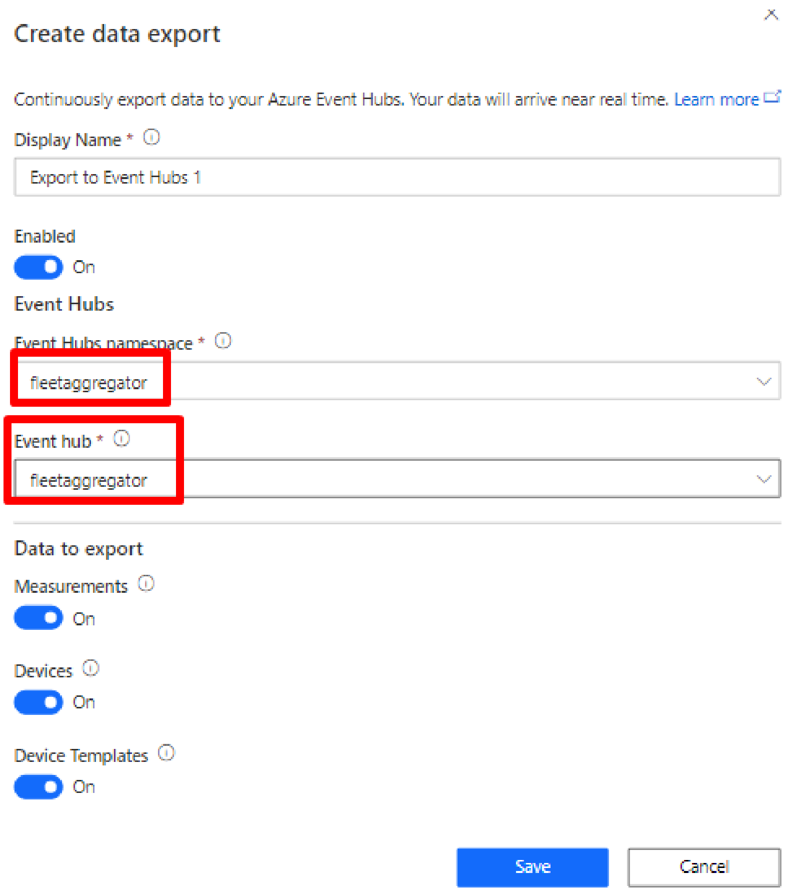
Azure Databricks

Follow the default prompts to build the event hub namespace and event hub.

Start exporting your data to the Event Hubs you just made



Continuously export data from IoT Central to your Azure Storage, Event Hubs, and Service Bus. Get started by creating an export. Learn more



Create data export

Continuously export data to your Azure Event Hubs. Your data will arrive near real time. Learn more

Display Name *: Export to Event Hubs 1

Enabled: On

Event Hubs

Event Hubs namespace *: fleetaggregator

Event hub *: fleetaggregator

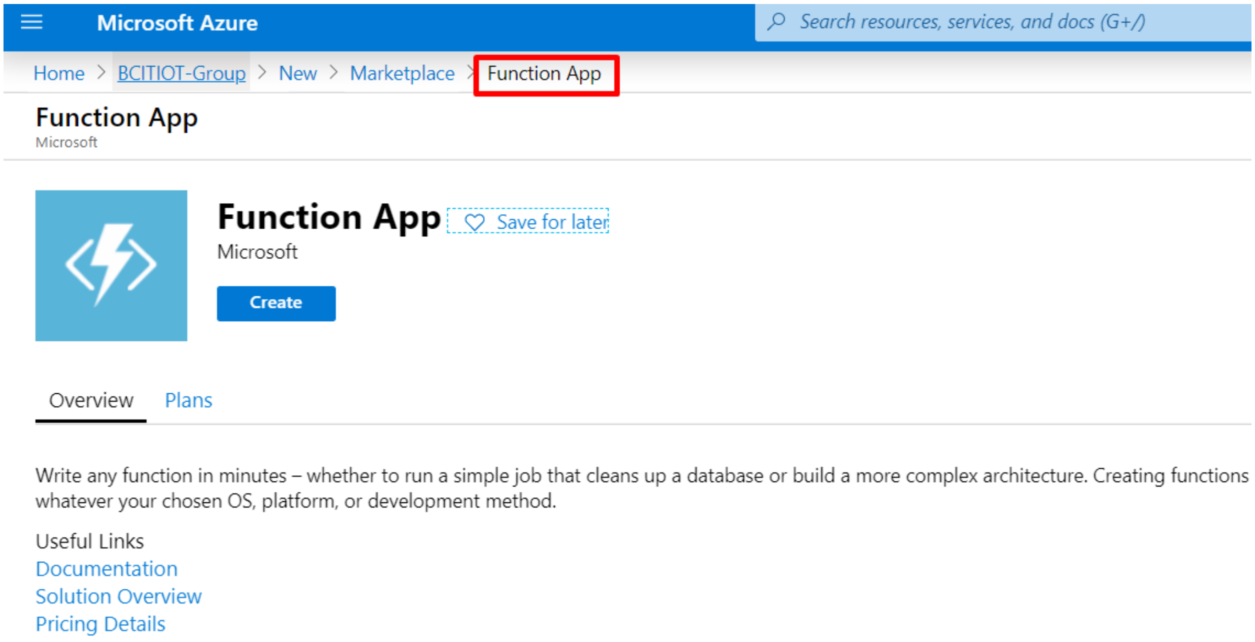
Data to export

Measurements: On

Devices: On

Device Templates: On

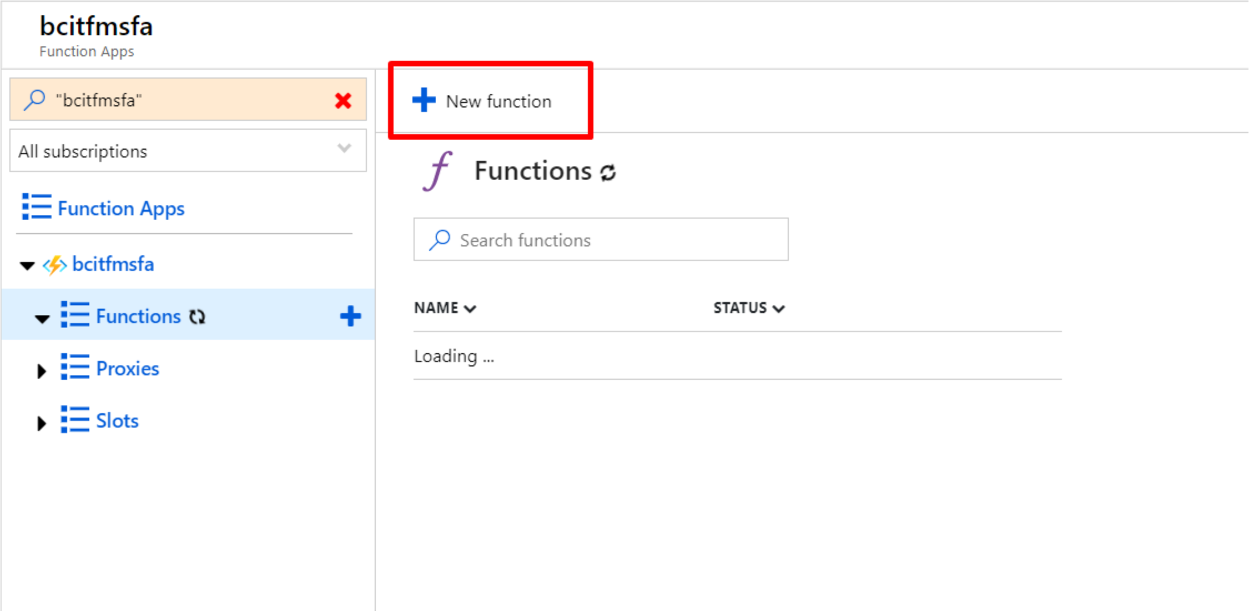
Get a FunctionApp service


Microsoft Azure
Search resources, services, and docs (G+/)
Home > BCITIOT-Group > New > Marketplace > Function App
Function App
Microsoft
Function App
Save for later
Microsoft
Create
Overview
Plans

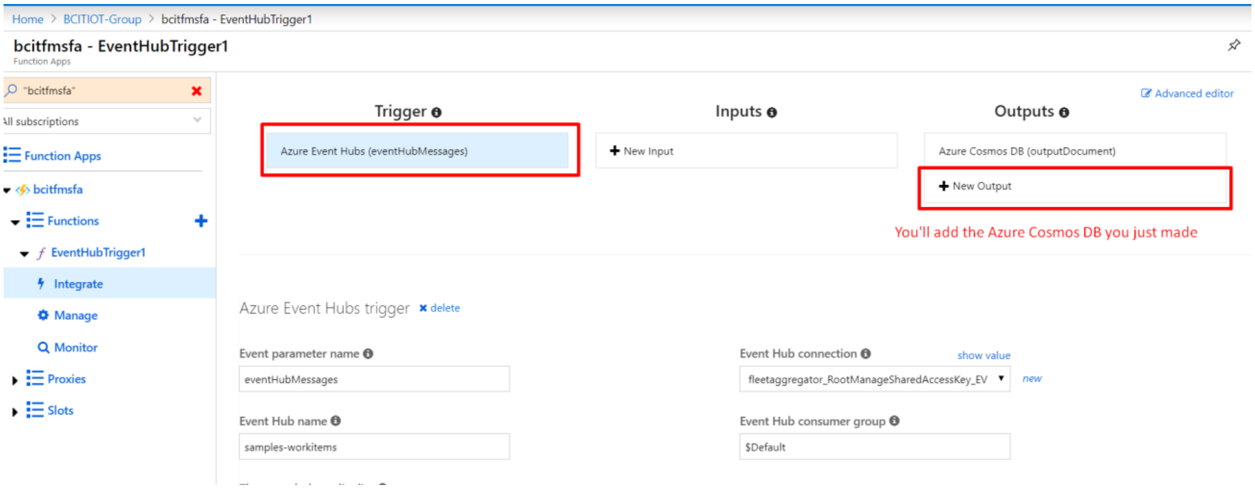
Write any function in minutes – whether to run a simple job that cleans up a database or build a more complex architecture. Creating functions whatever your chosen OS, platform, or development method.

Useful Links
Documentation
Solution Overview
Pricing Details

Build a new function…



Integrate the input trigger and the output binding (where to store the data)



Trigger: Azure Event Hubs (eventHubMessages)

Outputs: Azure Cosmos DB (outputDocument)

You'll add the Azure Cosmos DB you just made

Azure Event Hubs trigger

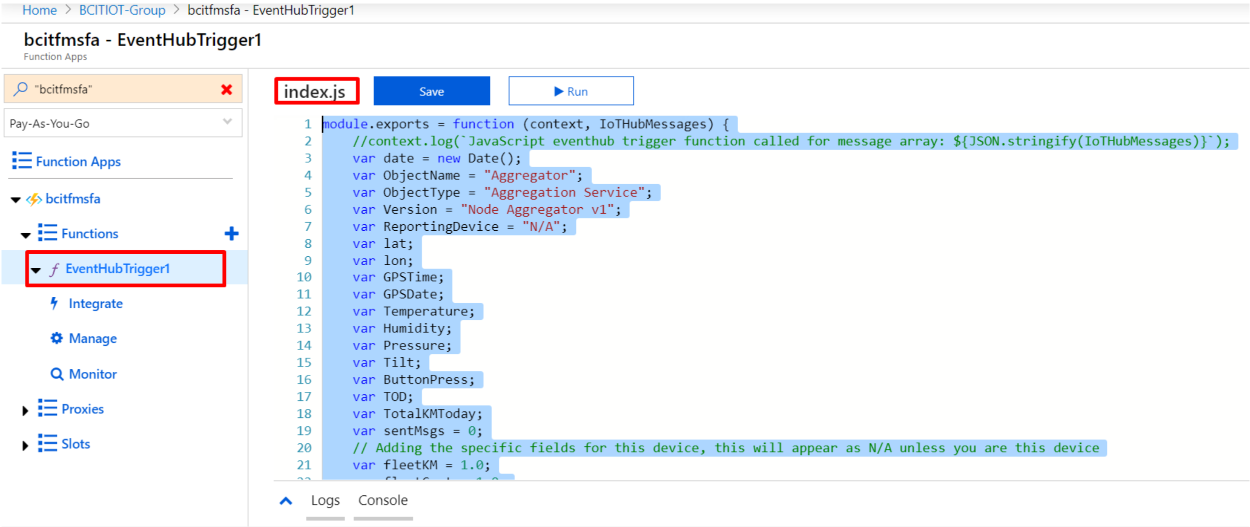
Event parameter name: eventHubMessages

Event Hub connection: fleetaggregator_RootManageSharedAccessKey_EV

Event Hub name: samples-workitems

Event Hub consumer group: $Default

Update the function you just made to match your input data and what you want to store in the database…

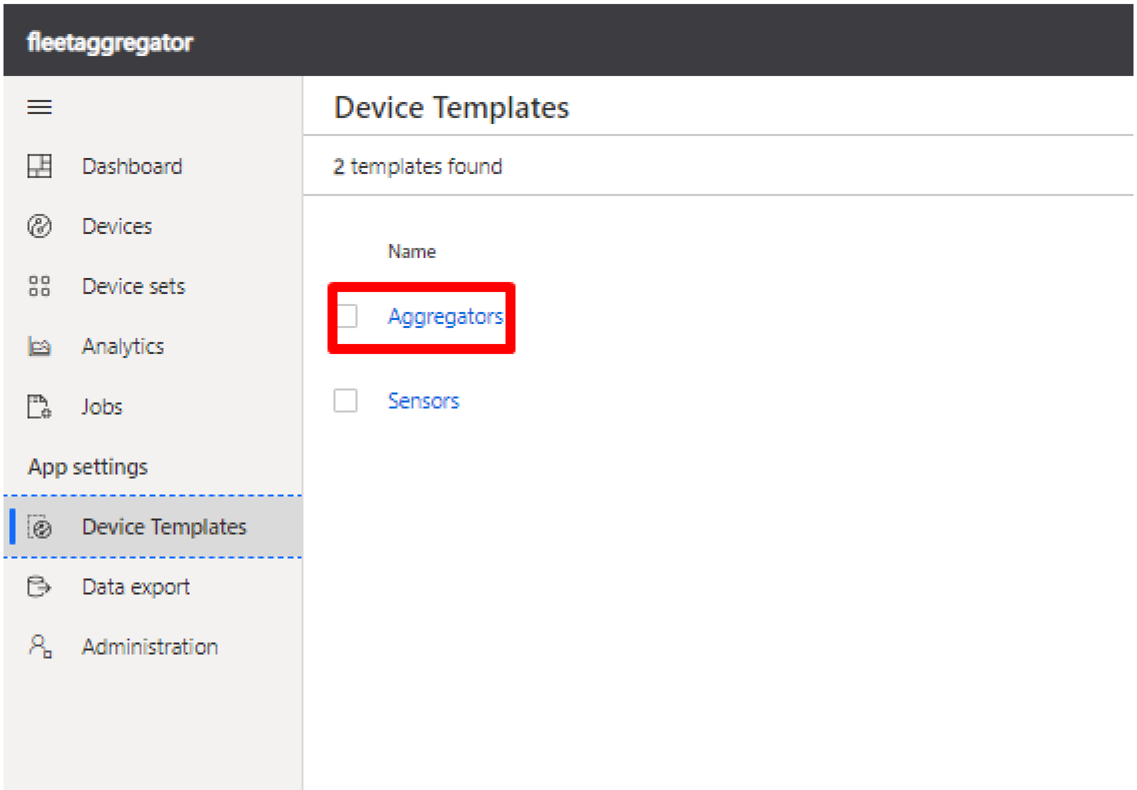


```
module.exports = function (context, IoTHubMessages) {
    //context.log(`JavaScript eventhub trigger function called for message array: ${JSON.stringify(IoTHubMessages)}`);
    var date = new Date();
    var ObjectName = "Aggregator";
    var ObjectType = "Aggregation Service";
    var Version = "Node Aggregator v1";
    var ReportingDevice = "N/A";
    var lat;
    var lon;
    var GPSTime;
    var GPSDate;
    var Temperature;
    var Humidity;
    var Pressure;
    var Tilt;
    var ButtonPress;
    var TOD;
    var TotalKMToday;
    var sentMsgs = 0;
    // Adding the specific fields for this device, this will appear as N/A unless you are this device
    var fleetKM = 1.0;
```

Here's an example which expects parameters like Humidity, Temperature, Location, TotalKMToday, etc. and stores them into the database specified :

```
module.exports = function (context, IoTHubMessages) {
    //context.log(`JavaScript eventhub trigger function called for message array: 
${JSON.stringify(IoTHubMessages)}`);
    var date = new Date();
    var ObjectName = "Aggregator";
    var ObjectType = "Aggregation Service";
    var Version = "Node Aggregator v1";
    var ReportingDevice = "N/A";
    var lat;
    var lon;
    var GPSTime;
    var GPSDate;
    var Temperature;
    var Humidity;
    var Pressure;
    var Tilt;
    var ButtonPress;
    var TOD;
    var TotalKMToday;
    var sentMsgs = 0;
    // Adding the specific fields for this device, this will appear as N/A unless
you are this device
    var fleetKM = 1.0;
    var fleetCost = 1.0;

   IoTHubMessages.forEach(message => {
       context.log("Printing the raw message received...");
       context.log(JSON.stringify(message, null, 1));


        //context.log(`JavaScript eventhub trigger function called for message
array: ${JSON.stringify(IoTHubMessages)}`);
        ObjectName = message.ObjectName;
        ObjectType = message.ObjectType;
        Version = message.Version;
        ReportingDevice = message.ReportingDevice;
        lat = message.location["lat"];
        lon = message.location["lon"];
        GPSTime = message.GPSTime;
        GPSDate = message.GPSTime;
        Temperature = message.Temperature;
        Humidity = message.Humidity;
        Pressure = message.Pressure;
        Tilt = message.Tilt;
        ButtonPress = message.ButtonPress;
        TOD = new Date().toLocaleString();
        TotalKMToday = message.TotalKMToday;
        sentMsgs = message.sentMsgs;
        // Adding the specific fields for this device, this will appear as N/A
unless you are this device
        fleetKM = message.fleetKM;
        fleetCost = message.fleetCost;

    });

    var output = {

        "ObjectName": ObjectName,
        "ObjectType": ObjectType,
        "Version": Version,
        "ReportingDevice": ReportingDevice,
        "lat": lat,
        "lon": lon,
        "GPSTime": GPSTime,
        "GPSDate": GPSDate,
        "Temperature": Temperature,
        "Humidity": Humidity,
        "Pressure": Pressure,
        "Tilt": Tilt,
        "ButtonPress": ButtonPress,
        "TOD": TOD,
        "TotalKMToday": TotalKMToday,
        "sentMsgs": sentMsgs,
        "fleetKM" : fleetKM,
        "fleetCost" : fleetCost
    };

    //this will print your output object in a nicely formatted way:
       context.log(TOD);
       context.log(JSON.stringify(output, null, 1));

    context.bindings.outputDocument = JSON.stringify(output);

    context.done();
};
```

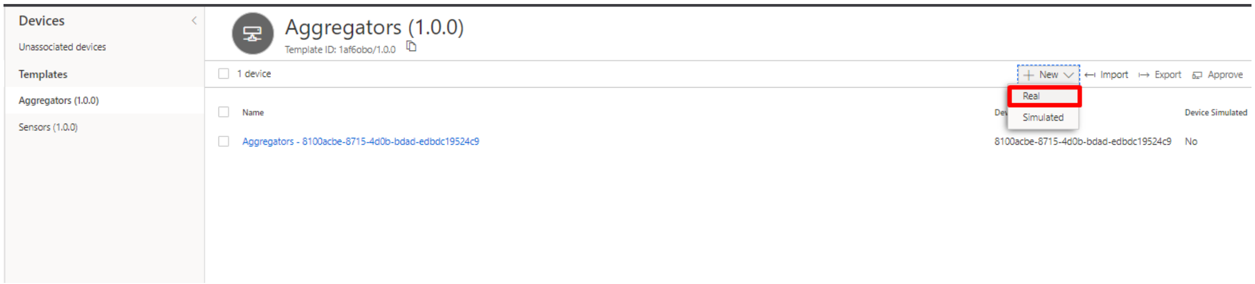
Provision a device to be the aggregator in IoT Central



Device Templates

2 templates found

Name
□ Aggregators
□ Sensors

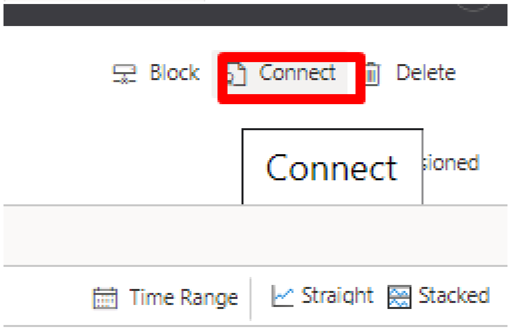


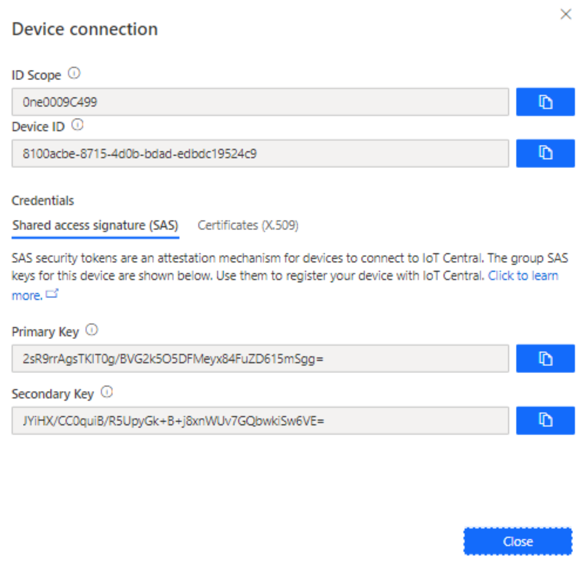
Aggregators (1.0.0)

Template ID: 1af6obo/1.0.0

□ 1 device

□ Name	Dev	Device Simulated
□ Aggregators - 8100acbe-8715-4d0b-bdad-edbdc19524c9	8100acbe-8715-4d0b-bdad-edbdc19524c9	No





Device connection

ID Scope

0ne0009C499

Device ID

8100acbe-8715-4d0b-bdad-edbdc19524c9

Credentials

Shared access signature (SAS) Certificates (X.509)

SAS security tokens are an attestation mechanism for devices to connect to IoT Central. The group SAS keys for this device are shown below. Use them to register your device with IoT Central. Click to learn more.

Primary Key

2sR9rrAgsTKIT0g/BVG2k5O5DFMeyx84FuZD615mSgg=

Secondary Key

JYiHX/CC0quiB/R5UpyGk+B+j8xnWUv7GQbwkiSw6VE=

Provision the device using the same method from the Azure “Configuring Device and Setting Up IoT Central” document. As a reminder, you’ll need to run a command like: *dps-keygen -di:8100acbe-8715-4d0b-bdad-edbdc19524c9 -dk:2sR9rrAgsTKIT0g/BVG2k5O5DFMeyx84FuZD615mSgg= -si:0ne0009C499* in order to get your connection string. This connection string will be used to push data from the node service you’ll create to fake being a device transmitting messages to IoT Central.

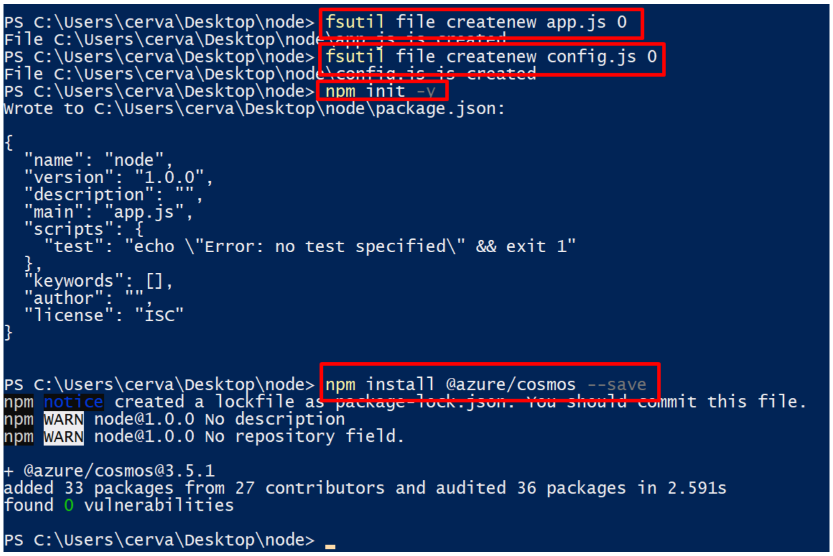
Make sure you create a widget which is made to receive data from the device!

Pull all the data you want via a Node.js service

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-nodejs-get-started#SetupNode

*Please refer to the sample code provided to follow along.

Set up a node project with dependencies to work with Azure

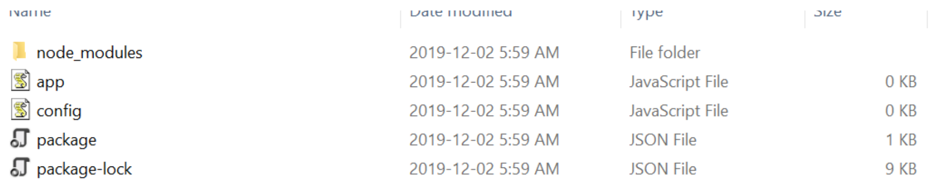


```
PS C:\Users\cerva\Desktop\node> fsutil file createnew app.js 0
File C:\Users\cerva\Desktop\node\app.js is created
PS C:\Users\cerva\Desktop\node> fsutil file createnew config.js 0
File C:\Users\cerva\Desktop\node\config.js is created
PS C:\Users\cerva\Desktop\node> npm init -y
Wrote to C:\Users\cerva\Desktop\node\package.json:

{
  "name": "node",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

PS C:\Users\cerva\Desktop\node> npm install @azure/cosmos --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN node@1.0.0 No description
npm WARN node@1.0.0 No repository field.

+ @azure/cosmos@3.5.1
added 33 packages from 27 contributors and audited 36 packages in 2.591s
found 0 vulnerabilities

PS C:\Users\cerva\Desktop\node> _
```

Note: Your commands might differ if in a Linux OS

1. Windows:
   - fsutil file createnew app.js 0
   - fsutil file createnew config.js 0
2. Linux/OS X:
   - touch app.js
   - touch config.js

At the end of those 4 commands, you're file structure should look like this:

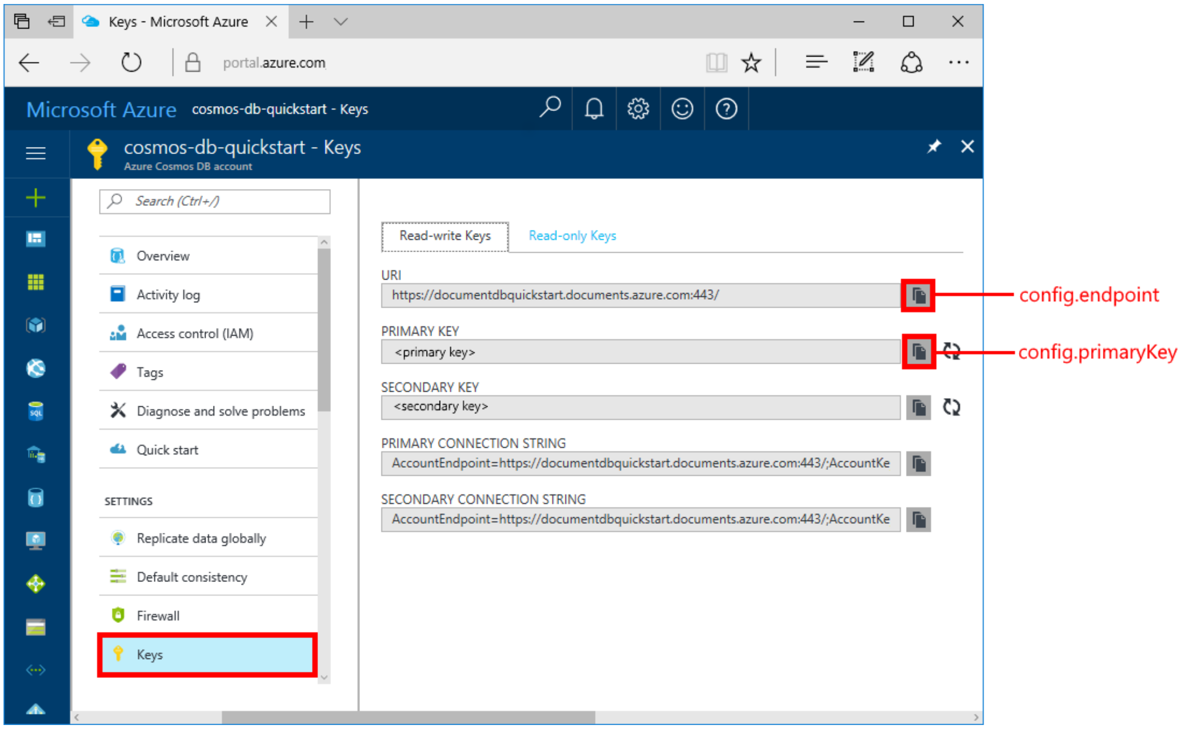


Name	Date modified	Type	Size
node_modules	2019-12-02 5:59 AM	File folder	
app	2019-12-02 5:59 AM	JavaScript File	0 KB
config	2019-12-02 5:59 AM	JavaScript File	0 KB
package	2019-12-02 5:59 AM	JSON File	1 KB
package-lock	2019-12-02 5:59 AM	JSON File	9 KB

Set your app's configurations

Now that your app exists, you need to make sure it can talk to Azure Cosmos DB. By updating a few configuration settings, as shown in the following steps, you can set your app to talk to Azure Cosmos DB:

1. Open config.js in your favorite text editor.
2. Copy and paste the code snippet below and set properties config.endpoint and config.key to your Azure Cosmos DB endpoint URI and primary key. Both these configurations can be found in the Azure portal.



URI
https://documentdbquickstart.documents.azure.com:443/

PRIMARY KEY
<primary key>

SECONDARY KEY
<secondary key>

PRIMARY CONNECTION STRING
AccountEndpoint=https://documentdbquickstart.documents.azure.com:443/;AccountKe

SECONDARY CONNECTION STRING
AccountEndpoint=https://documentdbquickstart.documents.azure.com:443/;AccountKe

Sample config.js connection strings and writing your aggregation code

```
var config = {}

config.endpoint = "https://fmscosmos.documents.azure.com:443/";
config.key =
"0QwJ7UK7BwDh9MJabChx8nei5aj8PKH0jFRTYLQAPyOx8l3bQCVDRnNL5jRgfcUKQDJ2Q5lfKq5P
taKwzhrjDg==";

config.database = {
  id: "outDatabase"
}

config.container = {
  id: "MyCollection"
}

config.items = {}

module.exports = config;
```

Sample app.js

Note: You'll need to input the details of the connection string of where you want to send the payload.

Below is a sample using the SQL API query…

```
//@ts-check
const CosmosClient = require('@azure/cosmos').CosmosClient
const config = require('./config')
const url = require('url')
const endpoint = config.endpoint
const key = config.key
const databaseId = config.database.id
const containerId = config.container.id
const partitionKey = { kind: 'Hash', paths: ['/Country'] }
const client = new CosmosClient({ endpoint, key })


/**
 * Application specific variables
 */
var nucleoboard2km;
var nucleoboard3km;
var date = new Date();

/**
 * Variables to make sure we transmit the same format as the other devices
 */

var ObjectName = "Aggregator";
var ObjectType = "Aggregation Service";
var Version = "Node Aggregator v1";
var ReportingDevice = "N/A";
var location = {
  "lat": "",
  "lon": ""
 };
var GPSTime = "N/A";
var GPSDate = "N/A";
var Temperature = "N/A";
var Humidity = "N/A";
var Pressure = "N/A";
var Tilt = "N/A";
var ButtonPress = "N/A";
var TOD = date.toLocaleString;
var TotalKMToday = "N/A";
var sentMsgs = 0;
// Adding the specific fields for this device, this will appear as N/A unless
you are this device
var fleetKM = 1.0;
var fleetCost = 1.0;

/**
 * Sending transmission code to move data to IoT Hub, this will get picked up
by the FunctionApp service
 * The method which actually handles the sending is in the aggregateKms
function
 */
var connectionString = 'HostName=iotc-000c0e58-
161c-4800-b262-e83ffb100c36.azure-
devices.net;DeviceId=8100acbe-8715-4d0b-bdad-
edbdc19524c9;SharedAccessKey=2sR9rrAgsTKIT0g/BVG
2k5O5DFMeyx84FuZD615mSgg=';

var Mqtt = require('azure-iot-device-mqtt').Mqtt;
var DeviceClient = require('azure-iot-device').Client
var Message = require('azure-iot-device').Message;
var clientSend = DeviceClient.fromConnectionString(connectionString, Mqtt);

/**
 * Create the database if it does not exist
 */
async function createDatabase() {
  const { database } = await client.databases.createIfNotExists({
    id: databaseId
  })
  //console.log(`Created database:\n${database.id}\n`)
}

/**
 * Read the database definition
 */
async function readDatabase() {
  const { resource: databaseDefinition } = await client
```

```
    .database(databaseId)
    .read()
  //console.log(`Reading database:\n${databaseDefinition.id}\n`)
}

/**
 * Create the container if it does not exist
 */
async function createContainer() {
  const { container } = await client
    .database(databaseId)
    .containers.createIfNotExists(
      { id: containerId, partitionKey },
      { offerThroughput: 400 }
    )
  //console.log(`Created container:\n${config.container.id}\n`)
}

/**
 * Read the container definition
 */
async function readContainer() {
  const { resource: containerDefinition } = await client
    .database(databaseId)
    .container(containerId)
    .read()
  //console.log(`Reading container:\n${containerDefinition.id}\n`)
}


/**
 * Cleanup the database and collection on completion
 */
async function cleanup() {
  await client.database(databaseId).delete()
}

/**
 * Exit the app with a prompt
 * @param {string} message - The message to display
 */
function exit(message) {
  console.log(message)
  console.log('Press any key to exit')
  process.stdin.setRawMode(true)
  process.stdin.resume()
  process.stdin.on('data', process.exit.bind(process, 0))
}
```

Sample Query Functions

```
/**
 * Calls for the latest transmission received from device NUCLEOBOARD2
 */
```

```
async function executeQuery1() {
    console.log();

    const { resources: results } = await client
    .database(databaseId)
    .container(containerId)
    .items.query('SELECT TOP 1 r.ObjectName, r.TotalKMToday, r._ts FROM root
r WHERE r.ObjectName = "NUCLEOBOARD2"  ORDER BY r._ts DESC')
    .fetchAll();

    console.log(results);
    nucleoboard2km = results[0]["TotalKMToday"];

}


/**
 * Calls for the latest transmission received from device NUCLEOBOARD3
 */
async function executeQuery2() {

    const { resources: results } = await client
        .database(databaseId)
        .container(containerId)
        .items.query('SELECT TOP 1 r.ObjectName, r.TotalKMToday, r._ts FROM
root r WHERE r.ObjectName = "NUCLEOBOARD3"  ORDER BY r._ts DESC')
        .fetchAll();


    console.log(results);
    nucleoboard3km = (results[0]["TotalKMToday"]);
    console.log();

}


/**
 * This function takes the totalKMs for each device, and adds them together
 * and stores them in variable totalKM. Then, the message is sent.
 */
function aggregateKms () {

    // This manages message sending
    setInterval(function(){
    // Simulate telemetry.
        sentMsgs++;
        executeQuery1();
        executeQuery2();
        fleetKM = parseFloat(nucleoboard2km) +
parseFloat(nucleoboard3km) * 1000; // This is
where we aggregate!
        fleetCost = fleetKM * .54;
        var message = new Message(JSON.stringify({
            ObjectName : ObjectName,
            ObjectType : ObjectType,
            Version : Version,
            ReportingDevice : ReportingDevice,
            location : location,
            GPSTime : GPSTime,
            GPSDate : GPSDate,
            Temperature : Temperature,
            Humidity : Humidity,
            Pressure : Pressure,
            Tilt : Tilt,
            ButtonPress : ButtonPress,
            TOD : new Date().toLocaleString(),
            TotalKMToday : TotalKMToday,
            sentMsgs : sentMsgs,
            fleetKM : fleetKM,
            fleetCost : fleetCost
        }));

        console.log('Sending message: ' + message.getData());

        // Send the message.
        clientSend.sendEvent(message, function (err) {
            if (err) {
                console.error('send error: ' + err.toString());
            } else {
                console.log("Message sent at: " + new
Date().toLocaleString());
            }
            });
    }, 3000); // Sends every 3000ms
}



/**
 * This drives the program and queries the devices, but will enter a
setInterval
 * loop at aggregateKms which runs the transmission every 15000ms
 */
createDatabase()
  .then(() => readDatabase())
  .then(() => createContainer())
  .then(() => readContainer())
  .then(() => aggregateKms())
  .then(() => {
    exit(`Completed successfully - transmission reports will display as they
are sent...`)
  })
  .catch(error => {
    exit(`Completed with error ${JSON.stringify(error)}`)
})
```

Run the service

You can now run the service by using your CLI, just run “node app.js” and the service should begin transmitting as a fake device which is aggregating data!