Smart Cities Hackathon
Tier 2 Challenge 5

Challenge 5: Configure a virtual Sensor to generate random readings

Summary

In this challenge you will create a Node.js script to generate random Hex readings for your chosen Virtual sensor which you built in challenge 1. The readings will then be uploaded to https://core.aql.com using the Core HTTP API https://api.core.aql.com/

This challenge can be completed by creating a Docker Image first or if you have node installed locally you can run the script locally. You will learn how to install docker, create a new image and create a Node.JS file for submitting readings.

Challenge 5 Step 1

Step 1

Go to the official Docker website: https://www.docker.com/products/docker-desktop
Download Docker Desktop for your operating system (Windows, macOS, or Linux).

Follow the installation instructions provided for your operating system. Make sure to install docker app repository - this contains the programme engine required to run commands.

Open a terminal or command prompt and start up the docker engine.
You can ask your AI Assistant how to do this for your operating system, as well as how to configure your system to start docker on boot.

Verify the installation & start-up of the docker engine by running;

```
sudo docker run hello-world
```

Step 2

Choose a location on your computer for the project files for example c:\code

Create a new folder for your challenge, for example with `mkdir challenge5`.

Step 3

Create a **dockerfile**

```
# Use the official Node.js image as the base image
FROM node:latest

# Set the working directory inside the container
WORKDIR /app

# Copy the local files into the container
COPY . .

# Install dependencies and start the application
RUN npm install

# Start the application
CMD ["node", "challenge5.js"]
```

Step 4

Create a file called **compose.yml**

```
services:
 node:
   build: .
   container_name: hackathon
   restart: always
```

Step 5

To run the scripts over HTTP we require a package for this so we next need to create a file called **package.json**

```
{
 "name": "hackathon",
 "version": "1.0.0",
 "description": "",
 "main": "challenge5.js",
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
   "axios": "^1.4.0"
```

```
  }
}
```

Step 6

We next need to create a new **challenge5.js** file , the first part of the script defines the constants and HTTP details. You will need the deviceid from your virtual device created in challenge 1, you will also need the bearer token you created in Challenge 1

Make sure to replace the bearer token & device ID with the values from Challenge 1;

```
const axios = require('axios');

// Constants
const CORE_URL = 'https://api.core.aql.com/v1/';
const TOKEN = 'YOUR_BEARER_TOKEN';
const DEVICE_ID = 'YOUR_DEVICE_ID';
```

Step 7

We next need to generate the function to create the random readings and then convert these to hex for our decoder to process.

```
function toHex(value) {
    console.log(value)
    // Convert value to hexadecimal string
    if (typeof value === 'number') {
        return value.toString(16);
    } else if (typeof value === 'string') {
        if (/\d+/.test(value)) {
            return Number(value).toString(16);
        } else {
            return Buffer.from(value).toString('hex');
        }
    }
    return value;
}

function generateReading() {
    // Generate a random reading
    const temperature = Math.floor(Math.random() * 38);  // Random
temperature between -20 and 40
    const humidity = Math.floor(Math.random() * 101);  // Random humidity
between 0 and 100
    const co2 = Math.floor(Math.random() * 701 + 300);  // Random CO2 level
between 300 and 1000 ppm
```

```
    const pm25 = Math.floor(Math.random() * 51);  // Random PM2.5 level
between 0 and 50 µg/m³
    const pm10 = Math.floor(Math.random() * 101);  // Random PM10 level
between 0 and 100 µg/m³

    return {
        temperature: toHex(temperature),
        humidity: toHex(humidity),
        co2: toHex(co2),
        pm25: toHex(pm25),
        pm10: toHex(pm10)
    };
}
```

Step 8

We next need to add the function to submit the generated readings to the api.core.aql.com end point

```
async function submitReading(payload) {
    const url = `${CORE_URL}devices/external-data-webhook/other/${DEVICE_ID}`;


    const headers = { 'Authorization': `Bearer ${TOKEN}`, 'Content-Type':
'application/json' };

    try {
        const response = await axios.post(url, payload, { headers });
        console.log('Reading submitted successfully:', payload);
    } catch (error) {
        console.error('Error submitting reading:', error);
    }
}
```

Step 9

We next need to create a function which is called when the script starts and then submits a reading every 5 mins. Note if you change the timer to be too aggressive with its submissions you will receive a too many requests response.

```
async function generateAndSubmitReadings() {
    /**
     * Generate and submit readings every 5 minutes.
     */
```

```
    // Immediately submit the first reading
    const firstReading = generateReading();
    await submitReading(firstReading);

    // Schedule subsequent readings every 5 minutes
    setInterval(async () => {
        const reading = generateReading();
        await submitReading(reading);
    }, 300000);  // 300000 milliseconds = 5 minutes
}

// Test the function
generateAndSubmitReadings();
```

Next run the container by calling docker compose up --build

You should see a response like

Attaching to hackathon-1
hackathon-1 | Reading submitted successfully: {
hackathon-1 | temperature: '32322e37',
hackathon-1 | humidity: 'f',
hackathon-1 | co2: '36f',
hackathon-1 | pm25: '30',
hackathon-1 | pm10: '7'
hackathon-1 | }
hackathon-1 | Reading submitted successfully: {
hackathon-1 | temperature: '32302e37',
hackathon-1 | humidity: '28',
hackathon-1 | co2: '2f2',
hackathon-1 | pm25: '1f',
hackathon-1 | pm10: '28'
hackathon-1 | }

Step 10

Log into core.aql.com and confirm you can see the readings being displayed.
If they've successfully been uploaded, exit the docker build.
You have now completed this challenge