**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

**Інститут комп’ютерних технологій, автоматики та метрології**

**Кафедра "Комп'ютеризовані системи автоматики"**



**ЗВІТ**

про виконання **лабораторної роботи №3 “**Опрацювання даних з IoT сенсорів за допомогою Serverless технологій хмарних провайдерів**”**

з навчальної дисципліни**: «Хмарні технології»**

**Виконав:**

студент групи ІР-31

Борецький Б. І.

**Прийняв:**

к.т.н., асистент

Рибак В. І.

Львів — 2023

**Завдання**

Обовязкові

1. (3б) Написати сценарій для емуляції роботи ІоТ сервісів.

* Використовувати можна як і спеціалізовані хмарні сервіси (https://apps.azureiotcentral.com/home) або власний скрипт, який буде завантажений і працюватиме у хмарі.
* Емулюватись повинна робота 3ьох різного типу датчиків, кожен з яких має свою частоту виконання (яка повинна налаштовуватись через конфігурацію). Емулятор повинен вміти працювати в діапазоні - 1 виклик у 20-100 ms
* Конфігуруватись також має адреса надсилання даних
* Кожен з 3ьох типів датчика повинен слати якийсь свій унікальний параметр (значення температури, вологості, освітленості, рівня шуму, рівня диму і т.п.) а також інформацію про свою локацію і дату, коли відбулась передача даних (наприклад датчик вологості може передавати дані про рівень вологості, а також дату замірів і свої координати).

1. (2б) Дані надсилати у чергу з використанням REST запитів. Чергу використати підходящу до обробки ІоТ сигналів відповідно до рекомандацій хмарного провайдера.
2. (3б) Дані з черги обробляти за допомогою serverless функцій чи сервісів відповідно до рекомендацій хмарного провадйера. Тригером повинно бути додавання нових даних до черги. Спроектувати дизайн для обробки різного типу даних з різних датчиків.
3. (2б) Спроектувати таблицю (або таблиці, в залежності від обраного дизайну) в базі даних для відповідних значень даних із сенсорів. Доступитись до БД за допомогою функцій з попереднього пункту і зберігати там інформацію. Дописати REST API запит на отримання історії даних з датчиків, використовуючи це саме serverless рішення.

Бонусні

1. (2б) Реалізувати Dead letter Queue handling у черзі, запропонувати дизайн.

**Виконання**

Написав сценарій для емуляції роботи IoT сервісів

main.py

from flask import Flask, request, render\_template, redirect

import threading

import time

import logging

from utils import generate\_sensor\_data

from queue\_utils import send\_sensor\_data, sqs\_queue\_url

app = Flask(\_\_name\_\_)

sensor\_configurations = {}

sensor\_threads = {}

def emulate\_sensor(sensor\_type, frequency, data\_address):

    while True:

        if sensor\_type in sensor\_configurations:

            data = generate\_sensor\_data(sensor\_type)

            app.logger.info(f"Sending {data}")

            send\_sensor\_data(data, data\_address)

            time.sleep(frequency / 1000)

            if sensor\_configurations[sensor\_type]['stop\_flag']:

                del sensor\_configurations[sensor\_type]

                break

        else:

            time.sleep(1)

@app.post('/')

def post\_index():

    sensor\_type = request.form['sensor\_type']

    frequency = int(request.form['frequency'])

    data\_address = request.form['data\_address']

    if sensor\_type in sensor\_configurations:

        sensor\_configurations[sensor\_type]['stop\_flag'] = True

    thread = threading.Thread(target=emulate\_sensor, args=(sensor\_type, frequency, data\_address))

    sensor\_threads[sensor\_type] = thread

    thread.start()

    sensor\_configurations[sensor\_type] = {

        'frequency': frequency,

        'data\_address': data\_address,

        'stop\_flag': False

    }

    return redirect('/')

@app.get('/')

def get\_index():

    return render\_template('index.html', configurations=sensor\_configurations,

                           sqs\_queue\_url=sqs\_queue\_url)

@app.post('/stop')

def stop\_sensor():

    sensor\_type = request.form['sensor\_type']

    if sensor\_type in sensor\_threads:

        config = sensor\_configurations[sensor\_type]

        config['stop\_flag'] = True

        del sensor\_threads[sensor\_type]

    return redirect('/')

if \_\_name\_\_ == '\_\_main\_\_':

    app.logger.setLevel(logging.INFO)

    app.run(host="0.0.0.0")

import json

import boto3

from utils import generate\_sensor\_data

ssm = boto3.client('ssm', region\_name='us-east-1')

sqs = boto3.client('sqs', region\_name='us-east-1')

sqs\_queue\_url = ssm.get\_parameter(

    Name='/iot/devices-queue-url',

    WithDecryption=False

)['Parameter']['Value']

def send\_sensor\_data(data, data\_address):

    response = sqs.send\_message(

        QueueUrl=data\_address,

        MessageBody=json.dumps(data)

    )

    return response

# Local testing

if \_\_name\_\_ == "\_\_main\_\_":

    # data = generate\_sensor\_data("temperature")

    # print(data)

    data = {"sensor\_type": "air\_pollution\_sensor"}

    print(send\_sensor\_data(data, sqs\_queue\_url))

utils.py

import random

import time

def generate\_random\_coordinates(min\_lat=-90, max\_lat=90, min\_lon=-18, max\_lon=180, decimal\_places=3):

    latitude = random.uniform(min\_lat, max\_lat)

    longitude = random.uniform(min\_lon, max\_lon)

    return f"{latitude:.{decimal\_places}f},{longitude:.{decimal\_places}f}"

def generate\_sensor\_data(sensor\_type: str):

    location = generate\_random\_coordinates()

    if sensor\_type == "temperature":

        return {

            "sensor\_type": sensor\_type,

            "temp\_celsius": round(random.uniform(10, 40), 2),

            "location": location,

            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")

        }

    elif sensor\_type == "humidity":

        return {

            "sensor\_type": sensor\_type,

            "value": round(random.uniform(30, 70), 2),

            "location": location,

            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")

        }

    elif sensor\_type == "light":

        return {

            "sensor\_type": sensor\_type,

            "value\_lux": round(random.uniform(0, 1000), 2),

            "location": location,

            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")

        }

    else:

        return {}

Дані з черги обробляв за допомогою Serverless Lambda функції.

import json

import pymysql

import boto3

ssm = boto3.client('ssm')

db\_host = ssm.get\_parameter(Name='/iot/db\_host', WithDecryption=True)['Parameter']['Value']

db\_user = ssm.get\_parameter(Name='/iot/db\_user', WithDecryption=True)['Parameter']['Value']

db\_password = ssm.get\_parameter(Name='/iot/db\_password', WithDecryption=True)['Parameter']['Value']

db\_name = ssm.get\_parameter(Name='/iot/db\_name', WithDecryption=True)['Parameter']['Value']

def lambda\_handler(event, context):

    with pymysql.connect(

        host=db\_host,

        user=db\_user,

        password=db\_password,

        database=db\_name,

        connect\_timeout=10,

    ) as conn, conn.cursor() as cursor:

        for record in event['Records']:

            message\_body = json.loads(record['body'])

            sensor\_type = message\_body['sensor\_type']

            if sensor\_type == 'temperature':

                table\_name = 'temperature\_sensor\_data'

                field\_name = 'temp\_celsius'

            elif sensor\_type == 'humidity':

                table\_name = 'humidity\_sensor\_data'

                field\_name = 'value'

            elif sensor\_type == 'light':

                table\_name = 'light\_sensor\_data'

                field\_name = 'value\_lux'

            else:

                raise ValueError(f'No sensor\_type={sensor\_type} logic implemented')

            try:

                insert\_query = f"INSERT INTO {table\_name} (sensor\_type, {field\_name}, location, timestamp) VALUES (%s, %s, %s, %s)"

                cursor.execute(

                    insert\_query,

                    (sensor\_type, message\_body[field\_name], message\_body['location'], message\_body['timestamp'])

                )

                conn.commit()

            except Exception as e:

                print(f"Error inserting data into {table\_name}: {e}")

Схема бази даних

CREATE SCHEMA sensor\_warehouse;

USE sensor\_warehouse;

-- Create the temperature sensor data table

CREATE TABLE temperature\_sensor\_data (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    sensor\_type VARCHAR(255),

    temp\_celsius DECIMAL(5, 2),

    location VARCHAR(255),

    timestamp TIMESTAMP

);

-- Create the humidity sensor data table

CREATE TABLE humidity\_sensor\_data (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    sensor\_type VARCHAR(255),

    value DECIMAL(5, 2),

    location VARCHAR(255),

    timestamp TIMESTAMP

);

-- Create the light sensor data table

CREATE TABLE light\_sensor\_data (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    sensor\_type VARCHAR(255),

    value\_lux DECIMAL(5, 2),

    location VARCHAR(255),

    timestamp TIMESTAMP

);

**Висновок**

Виконавши цю лабораторну роботу, я навчився працювати з сервісами SQS, SNS, Lambda.