



UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Ingeniería en Ciencia de la Computación y Tecnologías de la Información

CC3067 Redes

Diego Ruiz, 18761 y Jose Jorge Pérez, 18364.

Laboratorio 9

IoT: Estación Meteorológica

1 Objetivos

- Implementar y simular una posible solución comúnmente vista en proyectos de IoT. - Conocer y utilizar herramientas de software para aplicaciones de IoT y Edge Computing. - Resolver las necesidades y problemas requeridos por un proyecto en presencia de restricciones fuera de nuestro control.

2 Preámbulo

[Redacted]

3 Desarrollo

El laboratorio será desarrollado en parejas. Toda la evidencia de las fases debe de capturarla y entregarla en láminas de una presentación (PowerPoint, Keynote, Google Slides...). La presentación debe incluir explicación de lo que se hizo y capturas de pantalla de los resultados, así como las respuestas a las preguntas que se hagan en el transcurso de la actividad.

Para el laboratorio pueden utilizar cualquier lenguaje de programación, siempre y cuando tenga una librería o API de Kafka, naturalmente. Estaremos implementando y “emulando” un nodo de sensores en una estación meteorológica. Dichos nodos (las “cosas”) enviarán su telemetría periódicamente a un servidor Kafka en el borde (Edge), del cual luego se consumen datos para desplegar y graficar.

El paso cero sería instalar y configurar un servidor con Apache Kafka, el cual tomará el rol de nuestro Edge Server. Esta parte ya se les provee, por lo que no tienen que instalarlo ustedes; nos enfocaremos en el resto del “stack” IoT de esta implementación.

El servidor se encuentra en lab9.alumchat.xyz, correspondiente a la IP: 157.245.244.105. Se utiliza el puerto estándar de Kafka (9092).

3.1 Simulación de un Sensor

Las estaciones meteorológicas poseen una gran variedad de sensores y datos que miden constantemente. Nuestro nodo tendrá tres tipos de sensores, los cuales poseen los siguientes rangos, tipos y resoluciones de datos:

- Sensor de temperatura (Termómetro)
 - Rango: [0, 100.00]°C. Float de dos decimales.
- Sensor de Humedad Relativa (Higrómetro)
 - Rango: [0, 100]%. Entero.
- Sensor de dirección del viento.
 - {N, NW, W, SW, S, SE, E, NE}

En implementaciones reales dichos sensores son componentes electrónicos, usualmente externos o

embebidos, que generan datos en base a principios físicos y sus lecturas. En este laboratorio estaremos simulando los datos mediante generadores de números pseudoaleatorios (random). Para hacerlo más realista y que el despliegue se pueda apreciar mejor, los datos de Temperatura y Humedad Relativa deberán ser generados siguiendo una Distribución Uniforme (Gaussiana) en el intervalo de [0,100]. En otras palabras deberán muestrear de tal distribución para generar datos que están “centrados” en 50 (grados y %, respectivamente). Mas específico aun: la media deberá ser 50 y la varianza un valor razonable que ustedes consideren.

No hay restricción en la forma en que se generen las mediciones de Dirección de Viento; con una uniforme basta, o la que deseen. Se sugiere fuertemente condensar los datos a una representación tipo SOAP o JSON, dependiendo de su preferencia, para darle mayor modularidad y menos fragilidad al código. Por ejemplo, en el caso de JSON se podría ver algo así:

{"temperatura":56.32, "humedad":50, "direccion_viento":"SW"}

Se utilizó un script de Python para generar estos datos y guardarlos en un JSON.

```
def generate_sensor_data():
    temperature = np.clip(np.random.normal(50, 10), 0, 100)
    relative_humidity = int(np.clip(np.random.normal(50, 15), 0, 100))
    wind_direction = random.choice(['N', 'NW', 'W', 'SW', 'S', 'SE', 'E', 'NE'])

    return {
        "temperatura": round(temperature, 2), "humedad": relative_humidity, "direccion_viento": wind_direction
    }

def generate_and_save_json_data(num_records, filename):
    data = [generate_sensor_data() for _ in range(num_records)]

    with open(filename, 'w') as file:
        json_data = '[' + ',\n'.join(json.dumps(record) for record in data) + ']'
        file.write(json_data)

generate_and_save_json_data(50, 'sensor_data.json')
```

```
{ } sensor_data.json > ...
1 [{"temperatura": 40.38, "humedad": 37, "direccion_viento": "S"},
2 {"temperatura": 19.95, "humedad": 54, "direccion_viento": "NE"},
3 {"temperatura": 39.51, "humedad": 59, "direccion_viento": "E"},
4 {"temperatura": 39.48, "humedad": 50, "direccion_viento": "NW"},
5 {"temperatura": 46.27, "humedad": 73, "direccion_viento": "SW"},
6 {"temperatura": 55.26, "humedad": 50, "direccion_viento": "E"},
7 {"temperatura": 45.9, "humedad": 54, "direccion_viento": "S"},
8 {"temperatura": 64.31, "humedad": 42, "direccion_viento": "E"},
9 {"temperatura": 30.61, "humedad": 58, "direccion_viento": "SW"},
10 {"temperatura": 62.0, "humedad": 77, "direccion_viento": "SW"},
11 {"temperatura": 56.11, "humedad": 43, "direccion_viento": "NE"},
12 {"temperatura": 45.24, "humedad": 58, "direccion_viento": "SW"},
13 {"temperatura": 38.31, "humedad": 64, "direccion_viento": "W"},
14 {"temperatura": 34.94, "humedad": 59, "direccion_viento": "N"},
15 {"temperatura": 46.73, "humedad": 61, "direccion_viento": "SE"},
16 {"temperatura": 40.52, "humedad": 29, "direccion_viento": "NE"},
17 {"temperatura": 52.72, "humedad": 63, "direccion_viento": "S"},
18 {"temperatura": 42.38, "humedad": 69, "direccion_viento": "NW"},
19 {"temperatura": 32.61, "humedad": 47, "direccion_viento": "SE"},
20 {"temperatura": 46.96, "humedad": 58, "direccion_viento": "NE"},
21 {"temperatura": 53.46, "humedad": 44, "direccion_viento": "NE"},
22 {"temperatura": 63.39, "humedad": 29, "direccion_viento": "W"},
23 {"temperatura": 44.62, "humedad": 48, "direccion_viento": "N"},
24 {"temperatura": 49.7, "humedad": 57, "direccion_viento": "W"},
25 {"temperatura": 65.41, "humedad": 32, "direccion_viento": "SE"},
26 {"temperatura": 57.01, "humedad": 35, "direccion_viento": "W"},
27 {"temperatura": 65.45, "humedad": 74, "direccion_viento": "S"}]
```

➤ *Responda: ¿A qué capa pertenece JSON/SOAP según el Modelo OSI y porque?*

JSON/SOAP pertenecen a la capa 7 según el modelo OSI. Esta capa es la más cercana al usuario. Además, es la capa que se encarga de la interacción de software con la capa de red, y JSON es un formato que se utiliza para el intercambio de datos que se utilizan en los protocolos de comunicación entre aplicaciones, como HTTP o HTTPS, que operan en la Capa de Aplicación.

➤ *Responda: ¿Qué beneficios tiene utilizar un formato como JSON/SOAP?*

El beneficio de utilizar un formato como JSON o SOAP viene de que son lo suficientemente comprensibles para la lectura humana pero también son formatos que las aplicaciones pueden reconocer y navegar sin necesidad de modificaciones como puede ser un documento de texto (txt) o una hoja de cálculo (xls/xlsx/csv).

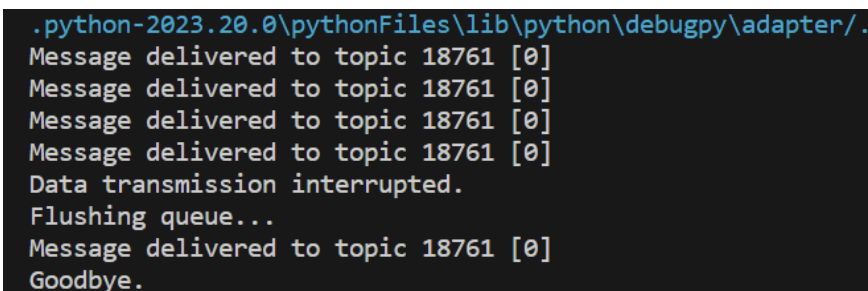
3.2 Envío de Datos al Server Edge

Una vez que puedan generar datos es momento de enviarlos a nuestro Kafka Broker. Enviaremos datos entre cada 15 y 30 segundos. Es válido accionar la generación de datos manualmente mientras se prueba y desarrollan las siguientes partes, pero su “Dispositivo IoT” (el rol que cumple el Producer) deberá ser capaz de quedarse corriendo y mandando hasta ser interrumpido. Nuevamente, el servidor se encuentra en lab9.alumchat.xyz, correspondiente a la IP: 157.245.244.105. Se utiliza el puerto estándar de Kafka (9092).

Para enviar datos deberán crear un **Kafka Producer**, el cual enviará datos al susodicho **bootstrap server**. Cada pareja deberá enviar datos a un **topic** único, para no cruzar datos con otras parejas. Para ello, cada pareja puede usar un número de carné de alguno de sus integrantes como topic.

Se debe consultar la documentación del lenguaje de su elección para encontrar la sintaxis específica para ello. Un esqueleto del programa en pseudocódigo puede ser:

```
//importar modulos de kafka
//importar elementos para random, y demas
public static void main(String[]: args){
    //inicializar random seeds, etc
    //crear topic, instanciar demas objetos necesarios
    KafkaProducer producer = new KafkaProducer(server='<host_or_ip>:9092');
    while(corriendo){
        JSONObject data = generarData().toJSON();//{'temperatura':50.1, ...}
        producer.send(topic='12345',key='sensor1',..., value=data.toString()); }
}
```



A screenshot of a terminal window with a dark background. The text is white and shows the output of a Python script. It starts with the directory path '.python-2023.20.0\pythonFiles\lib\python\debugpy\adapter/'. followed by several lines of 'Message delivered to topic 18761 [0]'. Then, it says 'Data transmission interrupted.' followed by 'Flushing queue...'. Then, another 'Message delivered to topic 18761 [0]'. and finally 'Goodbye.'.


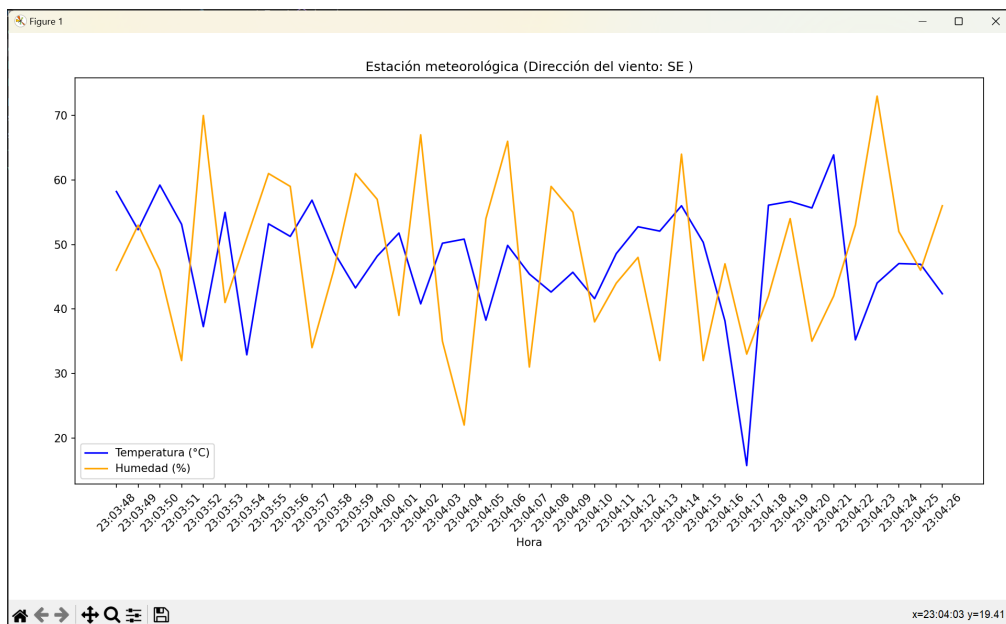
3.3 Consumir y Desplegar Datos Meteorológicos

Una vez se genere data y se esté publicando en Topics de nuestro Kafka Broker es momento de consumirlos y darles utilidad. Para ello usaremos un **Kafka Consumer**, el cual se suscribira a un Topic y escuchara por nuevos mensajes entrantes. El Consumer juega el rol de los elementos que están detrás del Edge, posiblemente en el core o en algún otro componente Edge, o bien alguna herramienta externa. Un pseudocódigo ejemplo del Consumer es el siguiente:

```
from kafka import KafkaConsumer
from kafka import ...
#import random y demas utilidades, modulos, etc.
consumer = KafkaConsumer('12345',group_id='foo2', ..., bootstrap_server='...')

for mensaje in consumer:
    print(mensaje)
    payload = procesarMensaje(mensaje)
    all_temp.append(payload['temperatura'])
    all_hume.append(payload['humedad'])
    all_wind.append(payload['direccion_viento'])
    #graficar, plotear, analizar, etc.
    plotAllData(all_temp, all_hume, all_wind)
```

Se sugiere explorar sobre los distintos parámetros del KafkaConsumer y el KafkaServer. Mediante se vaya recibiendo data, se debe ir graficando la telemetría pasada y entrante en vivo, actualizando el gráfico cada vez que entre un nuevo dato (que será aprox cada 15-30 segundos)



The screenshot shows three RStudio script windows. The first window, titled 'wind_record', contains a table with 10 rows of data. The second window, titled 'hum_record', contains a single column of 10 values. The third window, titled 'temp_record', contains a single column of 10 values. The data is as follows:

Direction	Humidity	Temperature
SE	52	35.77
N	52	64.63
E	63	48.37
NE	50	46.8
E	26	44.75
SE	62	47.43
E	57	52.04
NE	51	36.2
S	73	51.48
N	14	60.91
S	62	48.69

➤ *Responda: ¿Qué ventajas y desventajas considera que tiene este acercamiento basado en Pub/Sub de Kafka?*

Dentro de las ventajas que tiene este acercamiento es que permite la creación de aplicaciones con reacción en tiempo real, que es escalable horizontalmente y que ofrece garantía en cuanto a la entrega de mensajes. Por otro lado, sí presenta otras desventajas como lo puede llegar a ser que con mensajes grandes deja de tener su utilidad y de ser tan funcional, tiene pocas opciones de monitoreo y en algunas ocasiones un importante rebalanceo de datos lo cual afecta el rendimiento.

➤ *Responda: ¿Para qué aplicaciones tiene sentido usar Kafka? ¿Para cuáles no?*

Algunas aplicaciones con las que tiene sentido usar Kafka son en las que se tiene que publicar y suscribirse a datos constantemente cambiantes, que tengan que procesar estos datos o almacenarlos como se demostró en este laboratorio. Esto significa que sirve en sistemas de mensajería, event sourcing o registros de transacciones. Por otro lado, no es apta cuando los mensajes deben sí o sí ser procesados en orden y se requiera de un consumidor y una partición, aplicaciones con un servicio de cola típico o con mecanismos FIFO.

3.4 IoT en Entornos con Restricciones

En algunos entornos más complicados o remotos, como es el caso de nuestra estación, surgen diversos retos y restricciones que determinan fuertemente el diseño de nuestro protocolo/estructura de mensaje/etc. En redes IoT como LoRa y Sigfox una de esas limitaciones es el tamaño de la Carga Útil de un paquete enviado (payload).

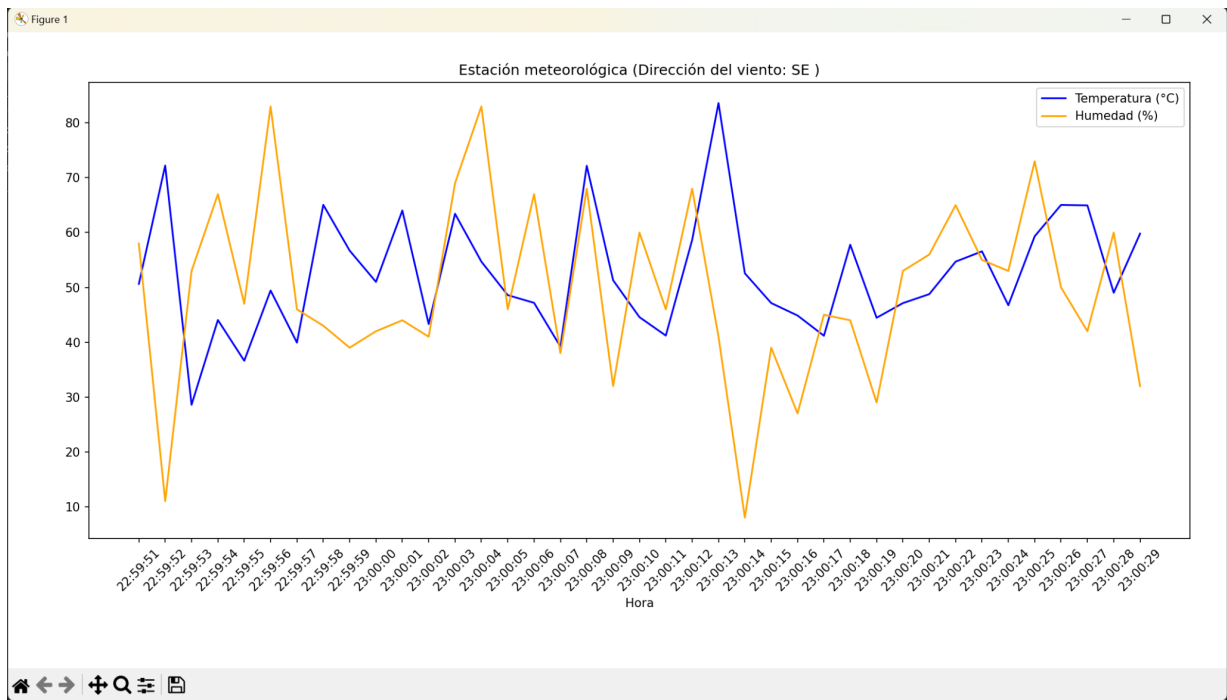
Modifique su código para adaptarlo a la siguiente restricción: el tamaño máximo de mensaje (payload) a enviar es de 3 bytes (24 bits). Recordemos que un Char pesa un byte, por lo que nuestro mensaje debe caber dentro de 3 caracteres ("aF!", "~Z", etc...). Debemos entonces **codificar** y **decodificar** nuestros mensajes antes de enviarlos y luego de recibirlos.

Debe evidenciar en su Presentación que se logró lo mismo de los pasos anteriores (envío, consumo, despliegue) pero ahora con la restricción de payload.

Producer:

```
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
Message delivered to topic 18761 [0]
```

Consumer:



Como podrán ver, ahora “cada bit cuenta”. A continuación algunos tips muy importantes:

- Implementar una función Encode (JSON to Bytes) y una Decode (Bytes to JSON) para el propósito.
- La temperatura en punto flotante es un gran problema. Solo el tipo de dato float ocupa 4 bytes (!), por lo que hay que usar nuestro ingenio para lograr que quepa.
- La humedad es un dato entero, entre 0-100, por lo que cabría en 7 bits mínimo.
- La dirección del viento es un valor categórico con 8 posibles opciones. Cabe en 3 bits mínimo.
- De los 24 bits nos quedan ahora 14... aca viene la pista importante:
 - ¿Cuánto es 2^{14} ?
 - Observe detenidamente el rango de valores posibles de temperatura, especialmente el máximo valor posible.
 - Vuelva a observar ese valor máximo detenidamente y compárelo con los 14 bits que nos quedan. ¿Cómo podemos hacer que entre ahí?

➤ *Responda: ¿Qué complejidades introduce el tener un payload restringido (pequeño)?*

Un payload pequeño nos da problemas como la fragmentación que se da cuando un mensaje es de mayor tamaño que el payload y los datos prácticamente se cortan. También hace que se tengan que mandar más paquetes para la misma información. Hace que aumente la latencia y la concurrencia de mensajes en un momento dado. Para implementar esto, se tiene que ser demasiado eficiente con lo que mandamos.

➤ *Responda: ¿Cómo podemos hacer que el valor de temperatura quepa en 14 bits?*

La temperatura se da en un rango de 0 a 100 con sus decimales. Tomando esto en cuenta, debemos evaluar que tanta exactitud necesitamos de nuestros datos y podemos ya sea limitar esta cantidad o usar el escalamiento de punto fijo, que se basa en decidir cuántos bits se usarán para la parte entera y cuántos para la parte fraccionaria, y luego escalar el valor de la temperatura acorde a esta decisión.

➤ *Responda: ¿Qué sucedería si ahora la humedad también es tipo float con un decimal? ¿Qué*

decisiones tendríamos que tomar en ese caso?

Mientras más valores tengamos que tengan que ser de tipo float, menos abundantes se sentirán los 14 bits ya que también debemos pensar en que el mensaje debe ser entendible y utilizar el formato establecido. Tomando esto en cuenta, es muy probable que los valores ya no sean posibles de enviar con ese número de bits y se de la situación de fragmentación que haga que perdamos información.

➤ *Responda: ¿Qué parámetros o herramientas de Kafka podrían ayudarnos si las restricciones fueran aún más fuertes?*

Kafka consta de un espacio predeterminado, el cual inicialmente es de 1MB, este espacio se puede modificar para aceptar más o menos información, de acuerdo a nuestras necesidades. Con kafka se tiende a la utilización de herramientas de manejo de datos y almacenamiento, entre las que se encuentran los mensajes referidos y niveles de almacenamiento

4 Entregar en Canvas

- La presentación con la evidencia de las fases, explicaciones, capturas de pantalla y respuesta a las preguntas, en formato PDF.
- Todo el código implementado en el Laboratorio.