



# Laboratorio 9

## Estación meteorológica

Jun Woo Lee  
Andrés de la Roca

# Simulación de un sensor



main.py &gt; delivery\_report

```
1 import json
2 import time
3 import random
4 from confluent_kafka import Producer
5
6 # Configuration
7 bootstrap_servers = 'lab9.alumchat.xyz:9092'
8 topic = 'datos-meteorologicos'
9
10 # Configuración de generadores de datos
11 temperatura_media = 50.0
12 humedad_media = 50
13 varianza_temperatura = 10.0
14 varianza_humedad = 10
15
16 # Configuración de productor
17 conf = {'bootstrap.servers': bootstrap_servers}
18 producer = Producer(conf)
19
20 def generar_datos():
21     temperatura = round(random.uniform(temperatura_media - varianza_temperatura, temperatura_media + varianza_temperatura), 2)
22     humedad = random.randint(humedad_media - varianza_humedad, humedad_media + varianza_humedad)
23     direccion_viento = random.choice(["N", "NW", "W", "SW", "S", "SE", "E", "NE"])
24
25     return {
26         "temperatura": temperatura,
27         "humedad": humedad,
28         "direccion_viento": direccion_viento
29     }
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

python3.10 + -

Andre@Chrysler MINGW64 ~/OneDrive/Universidad/Cuarto año/Segundo Semestre/Redes/Laboratorios/Lab 9 (main)

\$ python main.py

{ "temperatura": 41.67, "humedad": 55, "direccion\_viento": "N" }

{ "temperatura": 45.53, "humedad": 56, "direccion\_viento": "NW" }

{ "temperatura": 41.96, "humedad": 55, "direccion\_viento": "SE" }


{ "temperatura": 59.19, "humedad": 59, "direccion\_viento": "E" }



## ¿A qué capa pertenece JSON/SOAP según el Modelo OSI y por qué?

JSON y SOAP son formatos de intercambio de datos utilizados en la capa de aplicación del modelo OSI. Por un lado, JSON es un formato de texto ligero que se utiliza para intercambiar datos estructurados entre un servidor y un cliente. Se utiliza en la capa de aplicación porque define la estructura de los datos y cómo se deben interpretar, lo que lo hace más relevante para la lógica de la aplicación. Por otro lado, SOAP es un protocolo de comunicación que define reglas para la estructura de los mensajes intercambiados entre aplicaciones. Al igual que JSON, SOAP opera en la capa de aplicación porque está diseñado para proporcionar servicios a nivel de aplicación, como servicios web.

Por lo que tanto JSON como SOAP operan en la capa de aplicación ya que ambos tienen relación con la presentación y estructura de los datos en un nivel que es específico de la aplicación.



## ¿Qué beneficios tiene utilizar un formato como JSON/SOAP?

Utilizar formatos como JSON o SOAP para representar datos de una estación meteorológica ofrece varios beneficios. Primero, ambos formatos proporcionan una estructura organizada y fácilmente interpretable, lo que facilita la modularidad del código y la expansión del sistema. JSON, al ser liviano y basado en texto, ofrece una alta legibilidad y es ampliamente compatible con diferentes lenguajes de programación, facilitando la interoperabilidad. Por otro lado, SOAP, al ser un protocolo estándar para la comunicación entre aplicaciones, garantiza una mayor consistencia y permite definir claramente operaciones y tipos de datos. Además, ambas opciones son escalables y permiten una fácil integración con sistemas externos, contribuyendo a la robustez y flexibilidad de la implementación de la estación meteorológica y otras posibles aplicaciones.

Envío de datos al server Edge

The background features a series of dark gray, three-dimensional rectangular planes that recede into the distance, creating a sense of depth. A light green parallelogram is positioned on one of the upper planes, and a blue parallelogram is on a lower plane, both appearing to be part of the geometric structure.

```

17
18 def enviar_datos():
19     while True:
20         datos = generar_datos()
21         mensaje = json.dumps(datos)
22         producer.produce(topic, key="sensor", value=str(mensaje), callback=delivery_reporter)
23         print(mensaje)
24         time.sleep(5) # Enviar datos cada 5 segundos
25
26 if __name__ == '__main__':
27     enviar_datos()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS

python3.10 + -

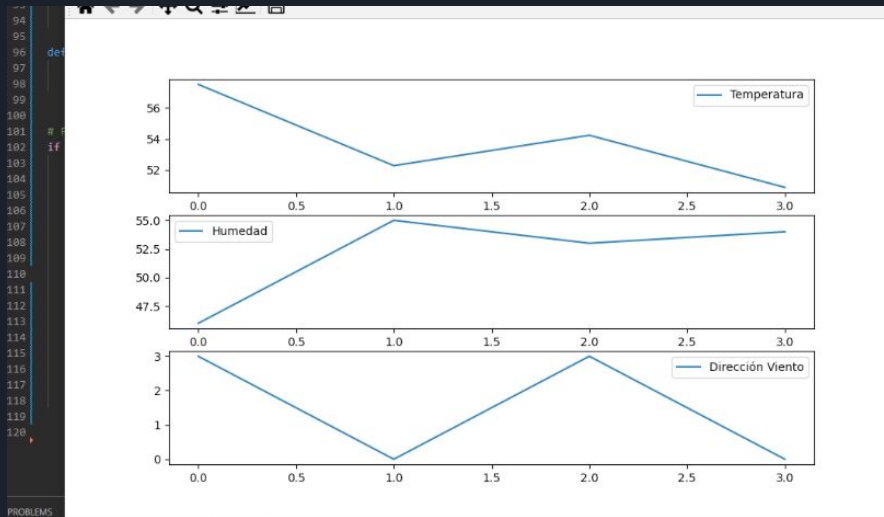
File "C:\Users\Andre\OneDrive\Universidad\Cuarto año\Segundo Semestre\Redes\Laboratorios\Lab 9\main.py", line 24, in enviar\_datos  
 time.sleep(5) # Enviar datos cada 5 segundos  
KeyboardInterrupt

Andre@Chrysler MINGW64 ~/OneDrive/Universidad/Cuarto año/Segundo Semestre/Redes/Laboratorios/Lab 9 (main.py)  
python main.py  
{"temperatura": 44.09, "humedad": 45, "direccion\_viento": "S"}  
{"temperatura": 50.07, "humedad": 49, "direccion\_viento": "N"}  
{"temperatura": 51.09, "humedad": 60, "direccion\_viento": "SW"}  
{"temperatura": 45.7, "humedad": 60, "direccion\_viento": "S"}  
{"temperatura": 50.31, "humedad": 58, "direccion\_viento": "SE"}

Consumir y desplegar datos  
meteorológicos








```
PS C:\Users\Jun\OneDrive\School\4th Year\2nd Semester\Redes\Laboratorio-9-Redes_UMG> cd 'c:\Users\Jun\OneDrive\School\4th Year\2nd Semester\Redes\Laboratorio-9-Redes_UMG'; & 'c:\Users\Jun\AppData\Local\Programs\Python\Python318\python.exe' 'c:\Users\Jun\vscode\extensions\ms-python.python-2023.28.8\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '58882' '-c' 'c:\Users\Jun\OneDrive\School\4th Year\2nd Semester\Redes\Laboratorio-9-Redes_UMG\main.py'
ConsumerRecord(topic='datos-meteorologicos-20332', partition=0, offset=66, timestamp=1699772363601, timestamp_type=0, key=b'sensor', value=b'{"temperatura": 57.52, "humedad": 46, "direccion_viento": "SE"}', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=63, serialized_header_size=1)
Mensaje entregado a datos-meteorologicos-20332 [0]
Mensaje entregado a datos-meteorologicos-20332 [0]
ConsumerRecord(topic='datos-meteorologicos-20332', partition=0, offset=67, timestamp=1699772394566, timestamp_type=0, key=b'sensor', value=b'{"temperatura": 52.28, "humedad": 55, "direccion_viento": "N"}', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=62, serialized_header_size=1)
ConsumerRecord(topic='datos-meteorologicos-20332', partition=0, offset=68, timestamp=1699772424639, timestamp_type=0, key=b'sensor', value=b'{"temperatura": 54.24, "humedad": 53, "direccion_viento": "SE"}', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=63, serialized_header_size=1)
Mensaje entregado a datos-meteorologicos-20332 [0]
ConsumerRecord(topic='datos-meteorologicos-20332', partition=0, offset=69, timestamp=1699772454728, timestamp_type=0, key=b'sensor', value=b'{"temperatura": 50.88, "humedad": 54, "direccion_viento": "N"}', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=62, serialized_header_size=1)
Mensaje entregado a datos-meteorologicos-20332 [0]
```

Demostracion del código: <https://clipchamp.com/watch/bMvA1pexmmm>



## ¿Qué ventajas y desventajas considera que tiene este acercamiento basado en Pub/Sub de Kafka?

El modelo Pub/Sub de Kafka ofrece una serie de ventajas, como su capacidad para manejar altos volúmenes de datos y su escalabilidad, lo que lo hace ideal para aplicaciones grandes y distribuidas. La arquitectura de Kafka proporciona alta durabilidad y un rendimiento robusto, garantizando la entrega fiable de mensajes. Además, el desacoplamiento de productores y consumidores facilita la flexibilidad y el mantenimiento del sistema. Mientras que las desventajas serían la complejidad de configuración y mantenimiento, y una curva de aprendizaje alta. Además, puede incurrir en costos significativos de infraestructura para sistemas a gran escala.




## ¿Para qué aplicaciones tiene sentido usar Kafka? ¿Para cuáles no?

Kafka es ideal para aplicaciones que necesitan procesar grandes volúmenes de datos en tiempo real, como en el procesamiento de eventos, streaming de datos, y en la integración de sistemas a gran escala. Es particularmente útil en escenarios donde la eficiencia en la comunicación entre múltiples sistemas es crucial, como en operaciones de ETL. Sin embargo, para aplicaciones con bajo volumen de datos, requerimientos mínimos de mensajería o aquellas que necesitan garantías de entrega inmediata de mensajes a muy baja latencia, Kafka podría no ser la opción más adecuada. En estos casos, soluciones más ligeras o sistemas de gestión de bases de datos tradicionales podrían ser más efectivos.

# IoT en entornos con restricciones









## ¿Qué complejidades introduce el tener un payload restringido (pequeño)?

Tener un payload restringido introduce complejidades significativas en el diseño y la implementación de sistemas de comunicación de datos. Primero, se requiere una cuidadosa planificación y codificación eficiente para asegurarse de que toda la información necesaria quepa en el espacio limitado. Esto a menudo implica sacrificar la precisión o la resolución de los datos, como limitar el rango numérico o la precisión decimal. Además, puede ser necesario implementar algoritmos de compresión o codificación especializados, lo que puede aumentar la complejidad del código y el costo computacional. También se eleva el riesgo de pérdida o distorsión de la información, especialmente si se requiere truncar o redondear los datos para que encajen en el espacio disponible.



## ¿Cómo podemos hacer que el valor de temperatura quepa en 14 bits?


Para hacer que el valor de la temperatura quepa en 14 bits, se podría escalar y transformar el rango de temperatura en un rango que se ajuste dentro de los límites de 214 posibles valores. Por ejemplo, si la temperatura varía entre -50 y 50 grados, se puede sumar 50 para eliminar los valores negativos, resultando en un rango de 0 a 100. Luego, este valor se puede escalar adecuadamente (por ejemplo, multiplicar por un factor que permita la máxima precisión dentro del rango de 14 bits) antes de convertirlo a un valor entero para la codificación.



# ¿Qué sucedería si ahora la humedad también es tipo float con un decimal? ¿Qué decisiones tendríamos que tomar en ese caso?

Si la humedad también se convierte en un valor float con un decimal, tendríamos que enfrentar desafíos adicionales. Como los floats ocupan más espacio, sería necesario aplicar un proceso de codificación similar al de la temperatura, ajustando y escalando el valor para que quepa en un número limitado de bits. Esto podría implicar reducir la precisión del decimal o asignar menos bits a la humedad, lo que disminuiría la resolución de los datos. En última instancia, podríamos tener que priorizar qué datos son más críticos para mantener su precisión y ajustar los otros en consecuencia.





## ¿Qué parámetros o herramientas de Kafka podrían ayudarnos si las restricciones fueran aún más fuertes?

Si las restricciones de payload fueran aún más estrictas, algunas herramientas y parámetros de Kafka podrían ser útiles. Por ejemplo, la compresión de mensajes en Kafka puede ayudar a reducir el tamaño de los mensajes transmitidos. Kafka soporta varios algoritmos de compresión como GZIP, Snappy o LZ4, que pueden ser particularmente efectivos para payloads más grandes o repetitivos. Además, ajustar el tamaño de lote y el intervalo de tiempo entre envíos en la configuración del productor puede optimizar la eficiencia en el envío de mensajes pequeños, permitiendo agrupar varios mensajes juntos para mejorar la eficiencia de la red y la utilización del broker.