Lab 7 Redes

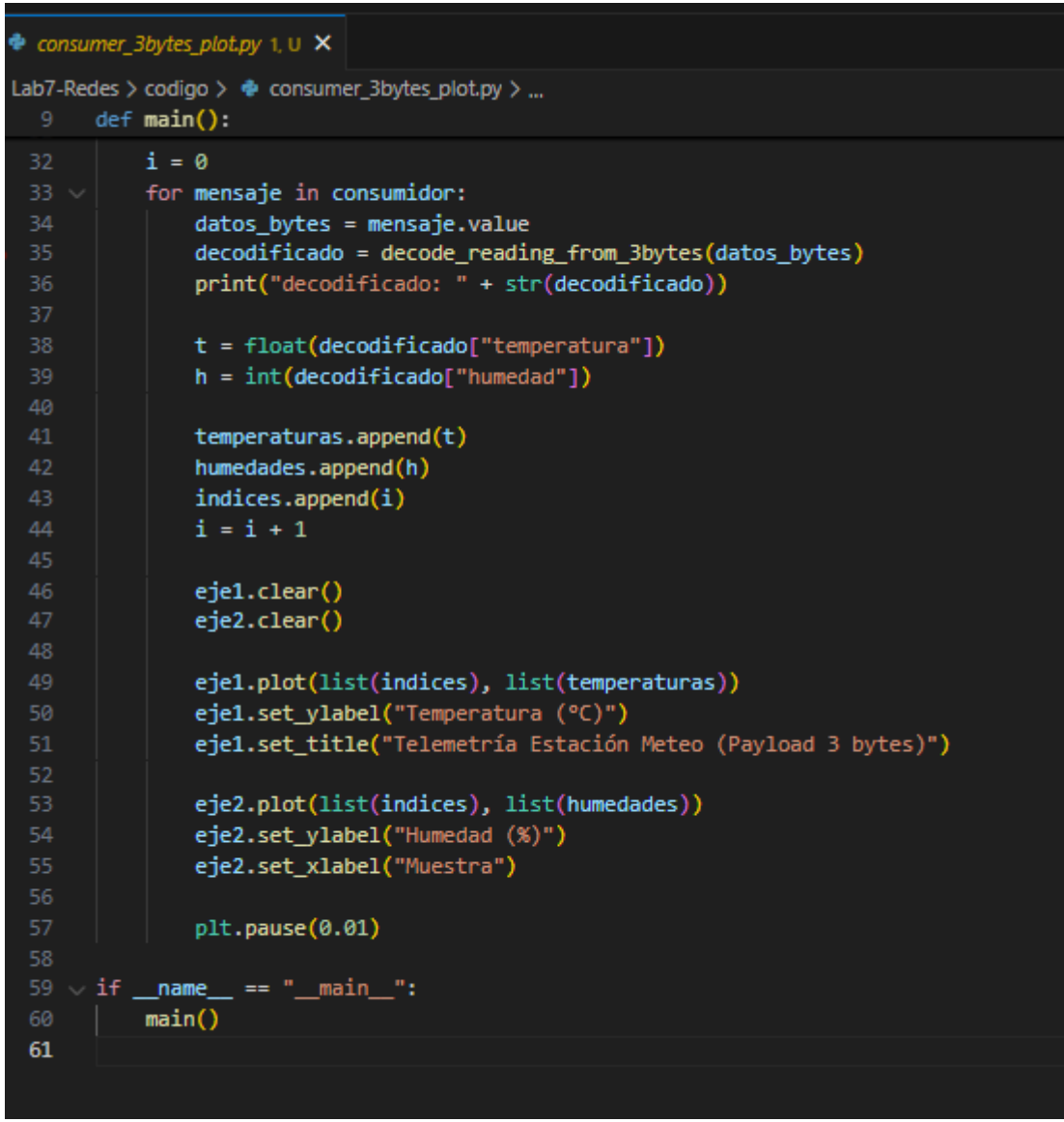
Brandon Reyes Morales 22992

Carlos Valladares 221164

Se crea el ambiente y las rutas del laboratorio

consumer_3bytes_plot.py

Recibe los 3 bytes, los decodifica con decode_reading_from_3bytes y grafica T/H igual que la versión JSON. Confirma que el flujo funciona igual con payload compacto.



consumer_json_plot.py

Se suscribe al topic, lee los JSON y actualiza dos gráficas (T °C y H %) en tiempo real con matplotlib. Verifica recepción y visualización continua.

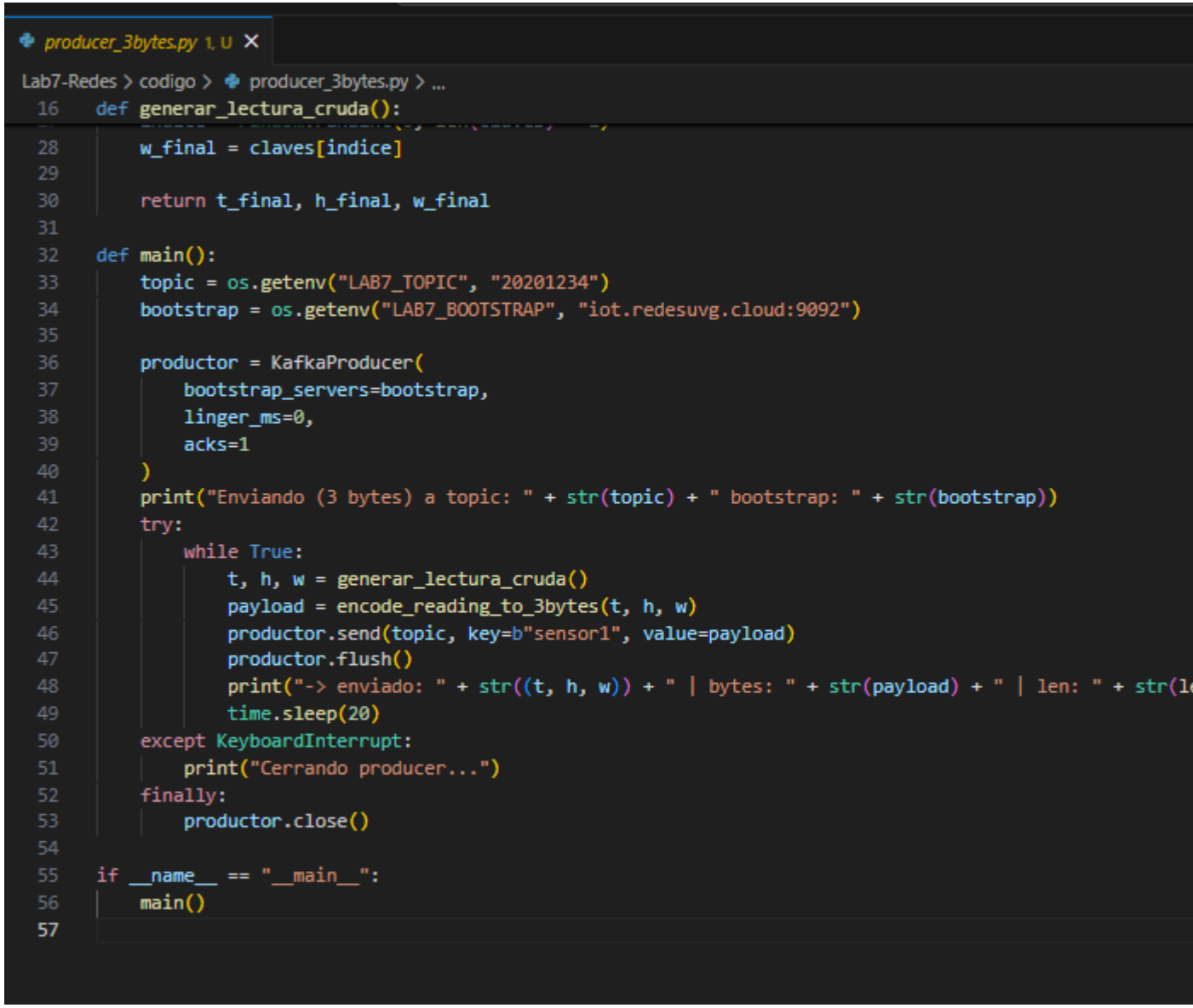
consumer_json_plot.py 1, U

Lab7-Redes > codigo > consumer_json_plot.py > main

```python
# consumer_json_plot.py
# NOTA: Código explícito y detallado; sin atajos ni compactaciones.
import os
import json
from collections import deque
from kafka import KafkaConsumer
import matplotlib.pyplot as plt

def main():
    topic = os.getenv("LAB7_TOPIC", "20201234")
    bootstrap = os.getenv("LAB7_BOOTSTRAP", "iot.redesuvg.cloud:9092")

    consumidor = KafkaConsumer(
        topic,
        group_id="grupo-lab7",
        bootstrap_servers=bootstrap,
        auto_offset_reset="latest",
        enable_auto_commit=True,
        value_deserializer=lambda b: json.loads(b.decode("utf-8"))
    )

    print("Escuchando topic: " + str(topic) + " bootstrap: " + str(bootstrap))

    # Buffers de datos con tamaño máximo definido para no crecer indefinidamente
    temperaturas = deque(maxlen=200)
    humedades = deque(maxlen=200)
    indices = deque(maxlen=200)

    # Configuración de gráfico interactivo
    plt.ion()
    figura = plt.figure()
    eje1 = figura.add_subplot(211)
    eje2 = figura.add_subplot(212)

    i = 0
    for mensaje in consumidor:
        payload = mensaje.value

        # Obtener valores con validación simple
        if "temperatura" in payload:
            valor_t = float(payload["temperatura"])
        else:
            valor_t = 0.0

        if "humedad" in payload:
            valor_h = int(payload["humedad"])
        else:
            valor_h = 0

        temperaturas.append(valor_t)
        humedades.append(valor_h)
        indices.append(i)
        i = i + 1

        # Redibujar ambas series, sin usar estilos abreviados
        eje1.clear()
        eje2.clear()

        eje1.plot(list(indices), list(temperaturas))
        eje1.set_ylabel("Temperatura (°C)")
        eje1.set_title("Telemetría Estación Meteo (JSON)")
```

producer_3bytes.py

Simula la lectura y, en lugar de JSON, empaqueta y envía exactamente 3 bytes por mensaje usando encode_reading_to_3bytes. Demuestra ahorro de ancho de banda.



producer_json.py

Genera lecturas simuladas (temperatura, humedad, viento) y las envía al broker Kafka en formato JSON cada ~20 s. Sirve para probar el flujo Pub/Sub “real” con datos legibles.

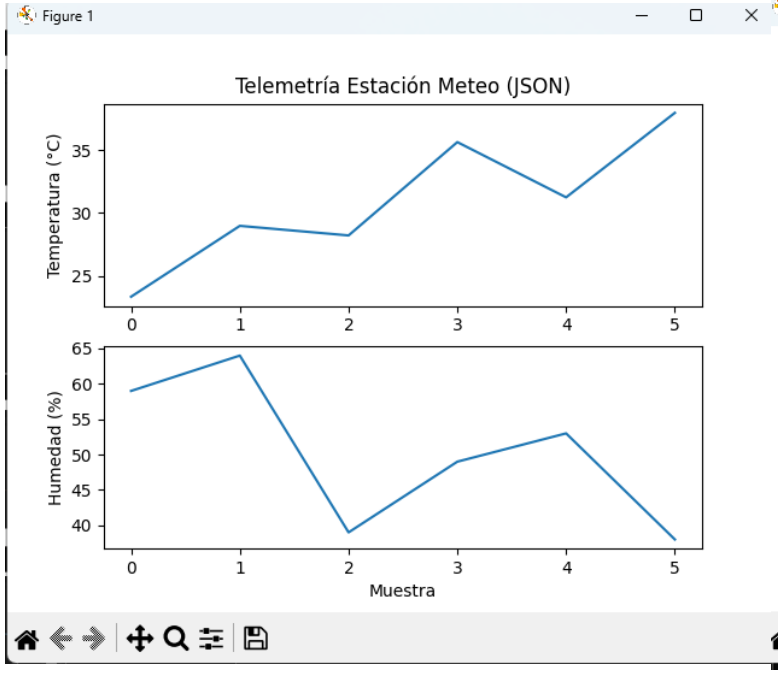
producer_json.py 1, U

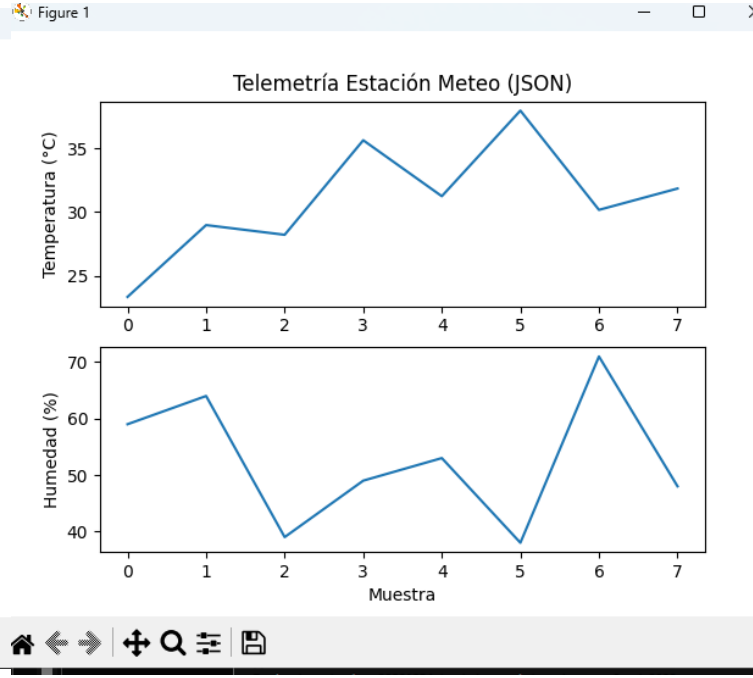
Lab7-Redes > codigo > producer_json.py > ...

```python
def main():
    # Leer topic y bootstrap de variables o usar valores por defecto
    topic = os.getenv("LAB7_TOPIC", "20201234")
    bootstrap = os.getenv("LAB7_BOOTSTRAP", "iot.redesuvg.cloud:9092")

    # Crear productor con serializadores explícitos
    productor = KafkaProducer(
        bootstrap_servers=bootstrap,
        value_serializer=lambda d: json.dumps(d).encode("utf-8"),
        key_serializer=lambda k: str(k).encode("utf-8"),
        linger_ms=0,
        acks=1
    )

    print("Enviando a topic: " + str(topic) + " bootstrap: " + str(bootstrap))
    try:
        while True:
            lectura = generar_lectura()
            # Enviar con key fija "sensor1" (string)
            productor.send(topic, key="sensor1", value=lectura)
            productor.flush()
            print("-> enviado: " + str(lectura))
            time.sleep(20)  # ~15-30 s sugerido
    except KeyboardInterrupt:
        print("Cerrando producer...")
    finally:
        productor.close()

if __name__ == "__main__":
    main()
```

Se verificó de extremo a extremo el flujo con JSON: el producer se conectó al broker iot.redesuvg.cloud:9092 y publicó periódicamente en el topic de mi carné lecturas válidas de la estación (temperatura, humedad y dirección del viento), mientras el consumer se suscribió al mismo topic, recibió los mensajes sin errores y actualizó en vivo dos gráficas en matplotlib (temperatura °C y humedad % vs. número de muestra).

Las consolas evidencian la conectividad (mensajes “Enviando a topic…” y “Escuchando topic…”) y múltiples envíos/recepciones consecutivas; las figuras muestran una serie temporal coherente con variaciones realistas, lo que confirma que la simulación, la publicación en Kafka y la visualización funcionan correctamente y están listas para repetir el flujo con el payload compactado de 3 bytes






```
PROBLEMS 1   OUTPUT   PORTS
TERMINAL

(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python producer_json.py
C:\Program Files\Python312\python.exe: can't open file 'C:\\Fichero\\Universidad\\8 Semestre\\Redes\
\Lab7-Redes\\Lab7-Redes\\producer_json.py': [Errno 2] No such file or directory
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python producer_json.py
C:\Program Files\Python312\python.exe: can't open file 'C:\\Fichero\\Universidad\\8 Semestre\\Redes\
\Lab7-Redes\\Lab7-Redes\\producer_json.py': [Errno 2] No such file or directory
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python codigo/producer_jso
n.py
>>
Enviando a topic: 20201234 bootstrap: iot.redesuvg.cloud:9092
-> enviado: {'temperatura': 23.36, 'humedad': 59, 'direccion_viento': 'SO'}
```

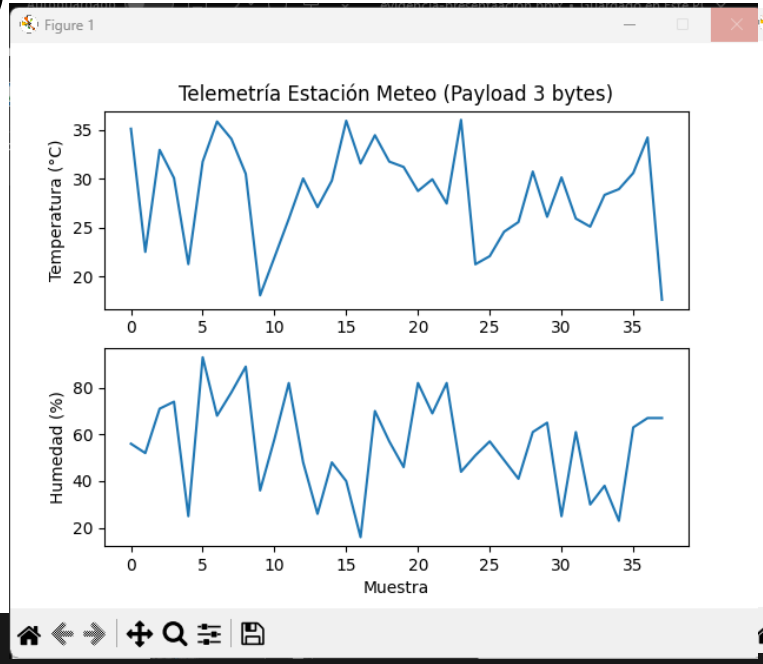
```
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> ls
----                 -------------         ------ ----
d-----        16/11/2025     16:17                codigo
-a----        16/11/2025     15:59          32856 evidencia-presentaacion.pptx
-a----        16/11/2025     16:24             70 notas.txt
-a----        16/11/2025     15:57             12 README.md


(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python consumer_json_plot.
py
C:\Program Files\Python312\python.exe: can't open file 'C:\\Fichero\\Universidad\\8 Semestre\\Redes\
\Lab7-Redes\\Lab7-Redes\\consumer_json_plot.py': [Errno 2] No such file or directory
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python codigo/consumer_jso
n_plot.py
Escuchando topic: 20201234 bootstrap: iot.redesuvg.cloud:9092
```

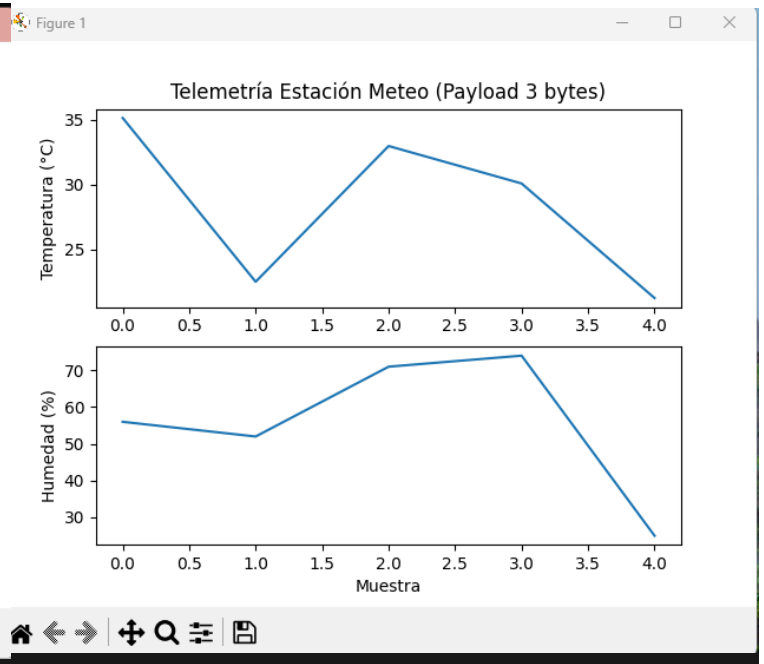
```
powershell
python Lab7-R...
python Lab7-R...
```

Flujo con 3 bytes (24 bits)

Con el payload de 3 bytes empaquetamos cada lectura (14 bits temp×100, 3 bits viento, 7 bits humedad) y el producer confirma len=3 en cada envío. El consumer decodifica correctamente (muestra decodificado: {...}) y las gráficas de temperatura y humedad mantienen el mismo patrón que con JSON. Resultado: misma información útil con mucho menos ancho de banda,

validando el flujo encode → Kafka → decode → visualización.





PROBLEMS 1 OUTPUT PORTS

TERMINAL

```
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python codigo/producer_3bytes.py
Enviando (3 bytes) a topic: 20201234 bootstrap: iot.redesuvg.cloud:9092
-> enviado: (26.178766872887902, 74, 'NE') | bytes: b'(\xeb\xca' | len: 3
-> enviado: (24.43277482861218, 53, 'SE') | bytes: b'&.\xb5' | len: 3
-> enviado: (19.253577586524948, 61, 'S') | bytes: b'\x1e\x16=' | len: 3
-> enviado: (35.11738359171415, 56, 'O') | bytes: b'6\xe18' | len: 3
-> enviado: (22.518247299131943, 52, 'O') | bytes: b'#14' | len: 3
-> enviado: (32.963487671263564, 71, 'NE') | bytes: b'3\x83\xc7' | len: 3
-> enviado: (30.077536123033607, 74, 'SO') | bytes: b'/\x01\xca' | len: 3
-> enviado: (21.27182273744688, 25, 'S') | bytes: b'!>\x19' | len: 3
```

```
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes>
(.venv) PS C:\Fichero\Universidad\8 Semestre\Redes\Lab7-Redes\Lab7-Redes> python codigo/consumer_3bytes_plot.py
Escuchando topic: 20201234 bootstrap: iot.redesuvg.cloud:9092
decodificado: {'temperatura': 35.12, 'humedad': 56, 'direccion_viento': 'O'}
decodificado: {'temperatura': 22.52, 'humedad': 52, 'direccion_viento': 'O'}
decodificado: {'temperatura': 32.96, 'humedad': 71, 'direccion_viento': 'NE'}
decodificado: {'temperatura': 30.08, 'humedad': 74, 'direccion_viento': 'SO'}
decodificado: {'temperatura': 21.27, 'humedad': 25, 'direccion_viento': 'S'}
```

1) ¿Qué ventajas y desventajas tiene el acercamiento Pub/Sub de Kafka?

El enfoque Pub/Sub de Kafka ofrece como ventajas el desacoplamiento entre productores y consumidores en tiempo y espacio, alto rendimiento y escalabilidad mediante particiones, persistencia del log que permite reprocesar (replay) y tolerancia a fallos con consumer groups; como desventajas, exige operar un clúster con mayor complejidad y curva de aprendizaje, introduce más latencia que una llamada RPC directa, puede ser excesivo cuando el tráfico es mínimo y requiere gestionar esquemas/versionado de mensajes.

2) ¿Para qué aplicaciones tiene sentido usar Kafka? ¿Para cuáles no?

Kafka tiene sentido en telemetría/IoT, recolección de logs y métricas, ingesta masiva y ETL en streaming, analítica en tiempo real y arquitecturas de microservicios impulsadas por eventos; no es ideal para peticiones sincrónicas de muy baja latencia (p. ej., login o cobros), para mensajería esporádica y diminuta donde el overhead domina, ni para transferencias punto a punto de archivos grandes.

3) ¿Cuánto es 2^14?

2^14 es 16,384.

4) Rango de temperatura y cabida en 14 bits: ¿cómo hacer que entre?

Con el rango de temperatura de 0–110.00 °C (dos decimales), basta usar escala fija: convertir T a entero con T_enc = round(T × 100), obteniendo valores 0..11,000 que caben holgadamente en 14 bits (máximo 16,383).

5) ¿Qué complejidades introduce un payload restringido (pequeño)?

Un payload restringido introduce complejidades como cuantización y pérdida de precisión por redondeo, riesgo de overflow si un campo excede su presupuesto de bits, necesidad de mapear categorías a pocos bits, problemas de compatibilidad al evolucionar el layout y depuración menos cómoda que con JSON legible.

6) ¿Cómo hacer que la temperatura quepa en 14 bits?

Para que la temperatura quepa en 14 bits se aplica punto fijo: multiplicar por 100, empaquetar T_enc en los 14 bits reservados y al decodificar dividir entre 100 para recuperar los dos decimales.

7) Si la humedad fuera float con 1 decimal, ¿qué decisiones tomar?

Si la humedad fuera flotante con un decimal (0.0–100.0), se codifica como H_enc = H × 10 (0..1000), lo que requiere 10 bits, habría que decidir entre reducir precisión de temperatura o humedad, reasignar el layout de bits (dar más a H y menos a T), o incluso enviar campos en mensajes alternos/aumentar el tamaño si la restricción lo permite.

8) ¿Qué parámetros/herramientas de Kafka ayudan si las restricciones fueran aún más fuertes?

Ante restricciones aún más fuertes se puede recurrir a compresión del producer (gzip/snappy/lz4/zstd), batching ajustando linger.ms y batch.size, serialización compacta con Avro/Protobuf o binario propio, filtrado y agregación en el borde con Kafka Streams/KSQL para reducir frecuencia/volumen, además de minimizar claves/headers y ajustar políticas de retención.

Conclusiones

- El formato JSON facilitó depuración e interoperabilidad sin cambios en la lógica.
- El payload de 3 bytes (24 bits) redujo el tamaño del mensaje manteniendo la información útil.
- Las funciones Encode/Decode demostraron recuperación correcta de T, H y viento.
- Las gráficas confirmaron patrones coherentes entre JSON y 3 bytes (misma dinámica).
- Kafka permitió desacoplar productor y consumidor, evidenciando un diseño escalable.
- Trabajo futuro: agregar timestamp/ID, checksum/CRC, persistencia y alertas por umbrales

Gracias por su atención