



Laboratorio 8: Estación Meteorológica

Redes

Angel Herrarte - 22873

Jose Gramajo - 22907

16 de noviembre del 2025

Repository

<https://github.com/aherrarte2019037/streaming-kafka>

Sección 3.1: Simulación de Sensores

Implementación

sensor_simulator.py con tres sensores:

- Temperatura: 0-110°C (float, 2 decimales), distribución gaussiana
- Humedad: 0-100% (entero), distribución gaussiana
- Dirección viento: 8 opciones (N, NO, O, SO, S, SE, E, NE), distribución uniforme

Formato JSON: {"temperatura": 56.32, "humedad": 51, "direccion_viento": "SO"}

Captura de pantalla

- Ejecución de python sensor_simulator.py

```
● (venv) angelherrarte@MacBook-Pro streaming-kafka % python3 sensor_simulator.py
== Simulador de Sensores Meteorológicos ==

Generando 10 muestras de datos:

Muestra 1: {"temperatura": 36.47, "humedad": 61, "direccion_viento": "NE"}
Muestra 2: {"temperatura": 31.05, "humedad": 62, "direccion_viento": "O"}
Muestra 3: {"temperatura": 27.32, "humedad": 44, "direccion_viento": "SE"}
Muestra 4: {"temperatura": 23.66, "humedad": 66, "direccion_viento": "E"}
Muestra 5: {"temperatura": 26.29, "humedad": 36, "direccion_viento": "S"}
Muestra 6: {"temperatura": 10.16, "humedad": 26, "direccion_viento": "SE"}
Muestra 7: {"temperatura": 17.66, "humedad": 56, "direccion_viento": "S"}
Muestra 8: {"temperatura": 11.62, "humedad": 37, "direccion_viento": "O"}
Muestra 9: {"temperatura": 22.09, "humedad": 72, "direccion_viento": "SO"}
Muestra 10: {"temperatura": 28.02, "humedad": 38, "direccion_viento": "S"} ■
○ (venv) angelherrarte@MacBook-Pro streaming-kafka %
```

Respuestas

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

- JSON/SOAP pertenecen a la Capa 6 del Modelo OSI. Se encarga de la representación de datos, traducción, codificación y compresión. Ambos son formatos de representación de datos que definen cómo estructurar la información de manera que pueda ser interpretada

por diferentes sistemas. También pueden considerarse parte de la Capa 7 cuando se usan como parte de protocolos de aplicación como HTTP/REST o SOAP sobre HTTP.

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

- Interoperabilidad: Permiten comunicación entre sistemas con diferentes lenguajes y plataformas
- Legibilidad: JSON es fácil de leer y comprender para humanos
- Estructura: Organizan datos de forma jerárquica y estructurada
- Estándar: Formatos ampliamente aceptados y soportados

Sección 3.2: Kafka Producer

Implementación

producer.py con:

- Conexión a iot.redesuvg.cloud:9092
- Topic: 22873
- Envío periódico cada 15-30 segundos
- Key: 'sensor1', Value: JSON stringify
- Manejo de señales para cierre graceful

Capturas de pantalla

- Ejecución mostrando múltiples envíos con timestamps y metadata

```
● (venv) angelherrarte@MacBook-Pro streaming-kafka % python3 producer.py
=====
Estación Meteorológica - Kafka Producer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Modo: JSON
=====

Presiona Ctrl+C para detener el producer.

[JSON] Enviando datos: {"temperatura": 30.96, "humedad": 44, "direccion_viento": "S"}
[2025-11-15 19:51:25] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 1
    Datos: {"temperatura": 14.62, "humedad": 64, "direccion_viento": "O"}

Esperando 26.0 segundos antes del siguiente envío...
```

```

[JSON] Enviando datos: {"temperatura": 25.7, "humedad": 70, "direccion_viento": "SE"}
[2025-11-15 19:51:51] Mensaje enviado exitosamente:
Topic: 22873
Partición: 1
Offset: 2
Datos: {"temperatura": 7.11, "humedad": 64, "direccion_viento": "O"}

Esperando 23.5 segundos antes del siguiente envío...

[JSON] Enviando datos: {"temperatura": 17.48, "humedad": 63, "direccion_viento": "NO"}
[2025-11-15 19:52:15] Mensaje enviado exitosamente:
Topic: 22873
Partición: 1
Offset: 3
Datos: {"temperatura": 32.38, "humedad": 79, "direccion_viento": "E"}

Esperando 25.8 segundos antes del siguiente envío...

[JSON] Enviando datos: {"temperatura": 16.3, "humedad": 71, "direccion_viento": "SE"}
[2025-11-15 19:52:41] Mensaje enviado exitosamente:
Topic: 22873
Partición: 1

```

3.3 Kafka Consumer y visualización de datos

Implementación general

Para el rol de *consumer* se utilizó el script `consumer.py`, configurado para conectarse al broker Kafka provisto por el curso (`iot.redesuvg.cloud:9092`), suscribiéndose al tópico `22873` con el *group id* `weather-station-consumer`. El programa se ejecutó en dos modos:

1. **Modo JSON** (`--format json`):

- El consumer recibe directamente el payload como JSON.
- Cada mensaje se procesa para extraer `temperatura`, `humedad` y `direccion_viento`.
- Los datos se almacenan en listas históricas y se grafican en tiempo real utilizando `matplotlib`, en tres subgráficas:
 - Temperatura vs. tiempo.
 - Humedad relativa vs. tiempo.

- Dirección del viento vs. tiempo (codificada como categorías N, NO, O, SO, S, SE, E, NE).

2. **Modo codificado** (`--format encoded` con `USE_ENCODING = True` en el producer):

- El consumer recibe un payload binario de 3 bytes.
- Antes de graficar, se aplica la función `decode()` (del archivo `encoder.py`) para reconstruir el JSON original con los tres campos.
- Una vez decodificado, el flujo de procesamiento y visualización es idéntico al modo JSON.

En ambos casos, la visualización se actualiza cada vez que arriba un nuevo mensaje, generando un comportamiento de “streaming” donde se observa la evolución temporal de las variables ambientales.

Pruebas en modo JSON

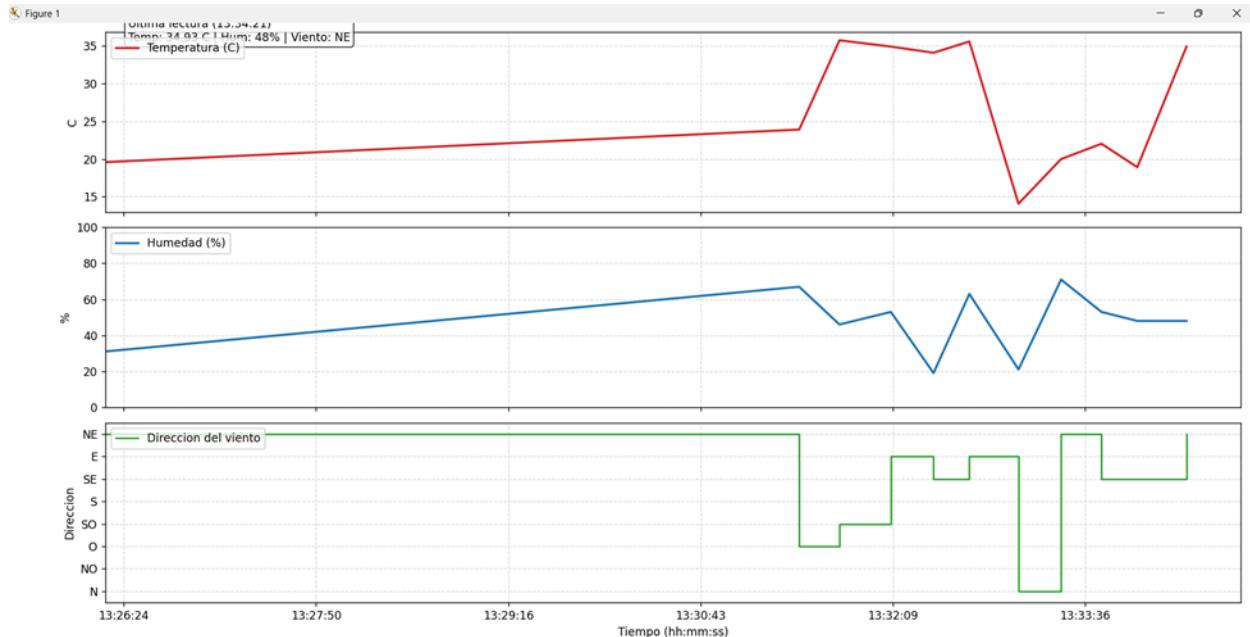
Para validar el funcionamiento básico se ejecutaron en paralelo:

```
python consumer.py --format json --auto-offset-reset earliest
python producer.py    # en modo JSON
```

Durante esta sesión se recibieron los mensajes con offsets **21–32**. Algunos ejemplos de mensajes consumidos fueron:

- Offset 21:
`{"temperatura": 19.59, "humedad": 31, "direccion_viento": "NE"}`
- Offset 23:
`{"temperatura": 35.75, "humedad": 46, "direccion_viento": "SO"}`
- Offset 32:
`{"temperatura": 34.09, "humedad": 57, "direccion_viento": "NO"}`

La consola del consumer mostró estos JSON ya decodificados, mientras que la consola del producer reportó los mismos valores al momento de enviarlos, confirmando que el broker preserva el contenido y el orden por partición. (**Capturas B y C**).



```
PS C:\Users\joses\PycharmProjects\streaming-kafka> python producer.py
=====
Estación Meteorológica - Kafka Producer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Modo: JSON
=====

Presiona Ctrl+C para detener el producer.

[JSON] Enviando datos: {"temperatura": 23.91, "humedad": 67, "direccion_viento": "0"}
[2025-11-16 13:31:27] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 22
    Datos: {"temperatura": 26.97, "humedad": 40, "direccion_viento": "E"}

Esperando 17.4 segundos antes del siguiente envío...

[JSON] Enviando datos: {"temperatura": 35.75, "humedad": 46, "direccion_viento": "SO"}
[2025-11-16 13:31:45] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 23
    Datos: {"temperatura": 30.99, "humedad": 68, "direccion_viento": "E"}

Esperando 22.0 segundos antes del siguiente envío...

[JSON] Enviando datos: {"temperatura": 34.92, "humedad": 53, "direccion_viento": "E"}
[2025-11-16 13:32:08] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 24
    Datos: {"temperatura": 33.67, "humedad": 54, "direccion_viento": "O"}
```

```

PS C:\Users\joses\PycharmProjects\streaming-kafka> python consumer.py --format json --auto-offset-reset earliest
=====
Estacion Meteorologica - Kafka Consumer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Grupo: weather-station-consumer
Formato payload: json
Visualizacion: habilitada
=====
Esperando mensajes...

[2025-11-16 13:26:15] key=sensor1 partition=1 offset=21
    Datos: {"temperatura": 19.59, "humedad": 31, "direccion_viento": "NE"}
[2025-11-16 13:31:27] key=sensor1 partition=1 offset=22
    Datos: {"temperatura": 23.91, "humedad": 67, "direccion_viento": "O"}
[2025-11-16 13:31:45] key=sensor1 partition=1 offset=23
    Datos: {"temperatura": 35.75, "humedad": 46, "direccion_viento": "SO"}
[2025-11-16 13:32:08] key=sensor1 partition=1 offset=24
    Datos: {"temperatura": 34.92, "humedad": 53, "direccion_viento": "E"}
[2025-11-16 13:32:27] key=sensor1 partition=1 offset=25
    Datos: {"temperatura": 34.1, "humedad": 19, "direccion_viento": "SE"}
[2025-11-16 13:32:43] key=sensor1 partition=1 offset=26
    Datos: {"temperatura": 35.59, "humedad": 63, "direccion_viento": "E"}
[2025-11-16 13:33:06] key=sensor1 partition=1 offset=27
    Datos: {"temperatura": 14.08, "humedad": 21, "direccion_viento": "N"}
[2025-11-16 13:33:25] key=sensor1 partition=1 offset=28
    Datos: {"temperatura": 20.0, "humedad": 71, "direccion_viento": "NE"}
[2025-11-16 13:33:43] key=sensor1 partition=1 offset=29
    Datos: {"temperatura": 22.03, "humedad": 53, "direccion_viento": "SE"}
[2025-11-16 13:33:59] key=sensor1 partition=1 offset=30
    Datos: {"temperatura": 18.92, "humedad": 48, "direccion_viento": "SE"}
[2025-11-16 13:34:21] key=sensor1 partition=1 offset=31
    Datos: {"temperatura": 34.93, "humedad": 48, "direccion_viento": "NE"}
[2025-11-16 13:34:42] key=sensor1 partition=1 offset=32
    Datos: {"temperatura": 34.09, "humedad": 57, "direccion_viento": "NO"}

```

Pruebas en modo codificado (payload de 3 bytes)

Para probar el escenario con restricciones se activó la codificación en el producer:

```

# En producer.py

USE_ENCODING = True

python consumer.py --format encoded --auto-offset-reset earliest

python producer.py      # ahora en modo codificado (3 bytes)

```

En este caso, el producer ya no imprime directamente el JSON sino el valor codificado en hexadecimal, por ejemplo:

- Offset 33: 314fb

- Offset 34: `25082e`

Sin embargo, el consumer, después de aplicar `decode()`, reconstruyó correctamente los siguientes datos:

- Offset 33:

```
{"temperatura": 27.04, "humedad": 69, "direccion_viento": "NO"}
```

- Offset 35:

```
{"temperatura": 21.2, "humedad": 43, "direccion_viento": "O"}
```

- Offset 40:

```
{"temperatura": 21.08, "humedad": 65, "direccion_viento": "SO"}
```

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

Ventajas

- **Desacoplamiento fuerte** entre productores y consumidores: mientras ambos comparten el mismo tópico y formato de mensaje, pueden evolucionar de forma independiente (lenguaje, tecnología, ritmo de envío/consumo).
- **Escalabilidad horizontal**: es posible añadir más productores o consumidores en paralelo, así como particionar los tópicos para repartir la carga entre varios brokers.
- **Tolerancia a fallos y persistencia**: los mensajes se almacenan en disco por partición, con opciones de replicación; si algún consumer cae, puede retomar desde el *offset* donde se quedó.
- **Orden por partición**: dentro de una partición los mensajes se mantienen ordenados, lo que es muy útil para análisis temporales como el de la estación meteorológica.

Desventajas

- **Complejidad operativa**: mantener un clúster de Kafka implica administrar brokers, almacenamiento, monitoreo y, en implementaciones tradicionales, también Zookeeper/Kraft.
- **Requisitos de infraestructura**: se necesita un servidor (o varios) en el borde o en la nube, lo cual puede ser costoso o poco viable en ambientes muy restringidos.

- **Latencia añadida:** aunque suele ser baja, se suma al tiempo de transmisión y procesamiento; en escenarios de *hard real-time* puede ser un problema.
- **Curva de aprendizaje:** la configuración de tópicos, particiones, grupos de consumidores, políticas de retención, etc., requiere cierto nivel de experiencia.

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

Aplicaciones donde sí conviene:

- **Telemetría IoT y monitoreo** de sensores en tiempo real (como en este laboratorio).
- **Pipelines de streaming y ETL** donde se procesan grandes volúmenes de eventos (clickstreams, logs, métricas).
- **Integración de microservicios** basada en eventos, donde Kafka actúa como *backbone* de comunicación entre servicios independientes.
- **Sistemas de analítica en tiempo (casi) real** que alimentan dashboards, alarmas y decisiones automatizadas.

Aplicaciones donde no es la mejor opción:

- Sistemas de **mensajería muy simples o de baja frecuencia**, donde un broker ligero como MQTT o incluso colas de mensajes básicas sería suficiente.
- Casos que requieran **transacciones complejas por mensaje** (por ejemplo, operaciones fuertemente ACID sobre datos relacionales); en estos escenarios una base de datos transaccional es más apropiada.
- Aplicaciones extremadamente críticas en tiempo real (control de hardware a microsegundos) donde la latencia adicional de Kafka puede resultar inaceptable.

Sección 3.4: Codificación para Payload de 3 Bytes

Implementación

encoder.py con codificación/decodificación en 3 bytes (24 bits):

- Temperatura: 14 bits (0-16383, escalado de 0-110°C)

- Humedad: 7 bits (0-100)
- Dirección viento: 3 bits (8 opciones mapeadas)

Fórmula: valor_24_bits = temp | (humedad << 14) | (viento << 21)

Modificación en producer.py: parámetro USE_ENCODING para activar codificación

Capturas de pantalla

- Ejecución de python encoder.py mostrando: JSON → bytes hex → JSON decodificado

```
(venv) angelherrarte@MacBook-Pro streaming-kafka % python3 encoder.py
== Prueba de Codificación/Decodificación ==

JSON original: {"temperatura": 56.32, "humedad": 51, "direccion_viento": "S0"}
Bytes codificados (3 bytes): 6ce0c4
Tamaño: 3 bytes
JSON decodificado: {"temperatura": 56.32, "humedad": 51, "direccion_viento": "S0"}

== Comparación ==
Temperatura: 56.32 -> 56.32
Humedad: 51 -> 51
Dirección viento: S0 -> S0
○ (venv) angelherrarte@MacBook-Pro streaming-kafka %
```

- Producer en modo codificado mostrando mensajes de 3 bytes

```

○ (venv) angelherrarte@MacBook-Pro streaming-kafka % python3 producer.py
=====
Estación Meteorológica - Kafka Producer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Modo: Codificado (3 bytes)
=====

Presiona Ctrl+C para detener el producer.

[ENCODED] Enviando datos codificados (3 bytes): 8a88e3
[2025-11-15 20:03:44] Mensaje enviado exitosamente:
  Topic: 22873
  Partición: 1
  Offset: 5
  Datos: {"temperatura": 27.73, "humedad": 67, "direccion_viento": "S0"}

Esperando 24.9 segundos antes del siguiente envío...

[ENCODED] Enviando datos codificados (3 bytes): 6c8c53
[2025-11-15 20:04:09] Mensaje enviado exitosamente:
  Topic: 22873
  Partición: 1
  Offset: 6
  Datos: {"temperatura": 20.65, "humedad": 39, "direccion_viento": "NE"}

Esperando 24.4 segundos antes del siguiente envío...

[ENCODED] Enviando datos codificados (3 bytes): 66133e
[2025-11-15 20:04:34] Mensaje enviado exitosamente:
  Topic: 22873
  Partición: 1
  Offset: 7
  Datos: {"temperatura": 19.06, "humedad": 15, "direccion_viento": "E"}

Esperando 21.5 segundos antes del siguiente envío...

```

El sistema se adapta a la restricción de un payload máximo de 3 bytes (24 bits). Esto obliga a mover parte de la lógica de interpretación al consumer:

- El consumer ya no recibe JSON legible, sino una secuencia de 3 bytes.
- Cada vez que llega un mensaje, la primera operación es aplicar `decode()` para reconstruir los campos originales.
- Solo después de decodificar se actualizan las series temporales y las gráficas.

Este enfoque demuestra cómo, en entornos IoT reales (LoRa, Sigfox, etc.), mucho de la “inteligencia” y del procesamiento se delega a los nodos de borde o al backend, ya que los dispositivos remotos se ven muy limitados en tamaño de mensaje, energía y capacidad de cómputo.

Respuestas a las preguntas de la sección 3.4

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

- Obliga a **codificar de forma compacta**, usando operaciones bit a bit y escalamiento de rangos, lo cual complica el diseño y la depuración.
- **Reduce la precisión** disponible: al cuantizar temperatura y humedad en pocos bits se pierde resolución frente al valor real.
- Hace más difícil **agregar nuevos campos** (por ejemplo, presión o velocidad del viento), ya que cualquier cambio puede romper el protocolo y exigir una recodificación completa.
- Aumenta la probabilidad de **errores de interpretación** si no se documenta bien el formato o si versiones distintas del sistema usan mapeos diferentes.

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

- 14 bits permiten representar enteros en el rango $0,2^{14}-10, 2^{14} - 10,2^{14}-1 = 0,163830, 163830,16383$.
- El sensor de temperatura trabaja en $[0, 110] \text{ } ^\circ\text{C}$.
- Se aplica entonces una **cuantización lineal**:
 1. Se normaliza la temperatura real `T_real` al rango $[0,1]$.
 2. Se multiplica por 16383 y se redondea al entero más cercano: `T_int = round(T_real * 16383 / 110)`.
 3. En la decodificación se invierte el proceso: `T_real ≈ T_int * 110 / 16383`.

De esta forma se almacena solo un entero de 14 bits y se recupera posteriormente con un error de cuantización muy pequeño.

¿Qué sucedería si ahora la humedad también fuera tipo float con un decimal?

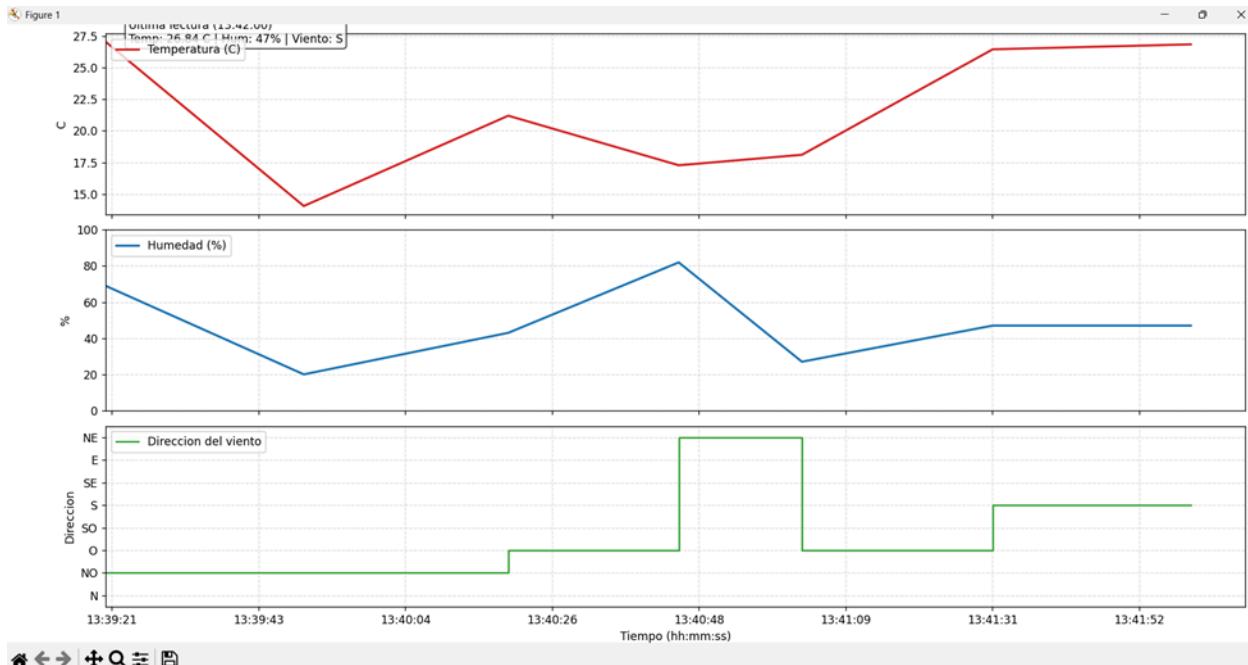
¿Qué decisiones tendríamos que tomar?

- Mantenerla como float implicaría necesitar más bits para representar valores como 45.3%, 72.8%, etc. Con solo 7 bits ya no sería suficiente.
- Las opciones serían:
 - **Reducir precisión**, por ejemplo, redondear al entero más cercano y seguir usando 7 bits.

- **Redistribuir los bits** (quitar algunos a la temperatura y dárselos a la humedad), lo que empeora la resolución de temperatura.
- **Separar parte decimal** (por ejemplo, multiplicar por 10 y almacenar un entero 0–1000), lo que exigiría más de 10 bits y rompería el límite de 24 bits.
- En la práctica, se suele optar por **sacrificar precisión** en la variable menos crítica o aumentar el tamaño del mensaje si el protocolo lo permite.

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

- **Compresión de mensajes** (`compression.type` = gzip, snappy, lz4, zstd) para reducir el tamaño efectivo del payload en la red.
- Ajustar `acks`, `linger.ms` y `batch.size` para **agrupar mensajes** y disminuir overhead por encabezados y conexiones.
- Utilizar **formatos binarios eficientes con schema registry** (Avro, Protobuf) cuando la red lo permita, maximizando compatibilidad y compresión.
- Aplicar **políticas de compactación de tópicos** (log compaction) para conservar solo el último estado de cada sensor, reduciendo el volumen total almacenado y transmitido a largo plazo.



```
PS C:\Users\joses\PycharmProjects\streaming-kafka> python consumer.py --format encoded
--auto-offset-reset earliest
=====
Estacion Meteorologica - Kafka Consumer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Grupo: weather-station-consumer
Formato payload: encoded
Visualizacion: habilitada
=====
Esperando mensajes...
[2025-11-16 13:39:20] key=sensor1 partition=1 offset=33
    Datos: {"temperatura": 27.04, "humedad": 69, "direccion_viento": "NO"}
[2025-11-16 13:39:49] key=sensor1 partition=1 offset=34
    Datos: {"temperatura": 14.06, "humedad": 20, "direccion_viento": "NO"}
[2025-11-16 13:40:19] key=sensor1 partition=1 offset=35
    Datos: {"temperatura": 21.2, "humedad": 43, "direccion_viento": "O"}
    Datos: {"temperatura": 26.84, "humedad": 47, "direccion_viento": "S"}
[2025-11-16 13:42:16] key=sensor1 partition=1 offset=40
    Datos: {"temperatura": 21.08, "humedad": 65, "direccion_viento": "SO"}
```

```
PS C:\Users\joses\PycharmProjects\streaming-kafka> python producer.py
=====
Estación Meteorológica - Kafka Producer
=====
Servidor: iot.redesuvg.cloud:9092
Topic: 22873
Modo: Codificado (3 bytes)
=====

Presiona Ctrl+C para detener el producer.

[ENCODED] Enviando datos codificados (3 bytes): 314fbb
[2025-11-16 13:39:20] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 33
    Datos: {"temperatura": 28.88, "humedad": 51, "direccion_viento": "N"}

Esperando 29.0 segundos antes del siguiente envío...

[ENCODED] Enviando datos codificados (3 bytes): 25082e
[2025-11-16 13:39:49] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 34
    Datos: {"temperatura": 30.25, "humedad": 39, "direccion_viento": "NE"}

Esperando 29.3 segundos antes del siguiente envío...

[ENCODED] Enviando datos codificados (3 bytes): 4acc56
[2025-11-16 13:40:20] Mensaje enviado exitosamente:
    Topic: 22873
    Partición: 1
    Offset: 35
    Datos: {"temperatura": 25.76, "humedad": 33, "direccion_viento": "NO"}

Esperando 24.9 segundos antes del siguiente envío...
```