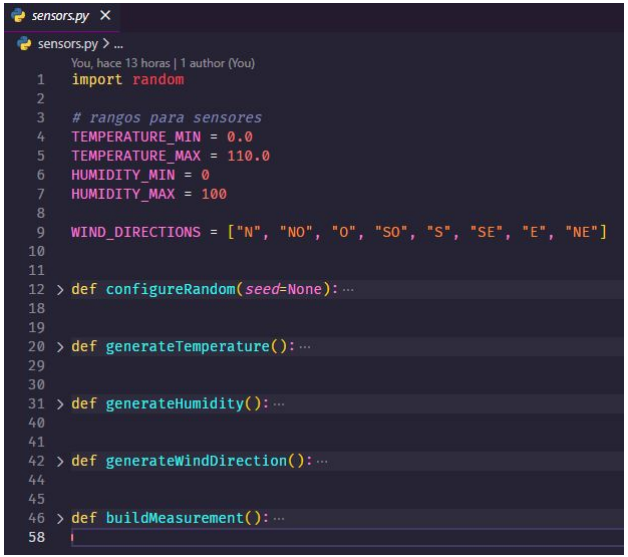
Estación Meteorológica con Apache Kafka

Implementación: Enlace a repositorio

Integrante: Josué Say

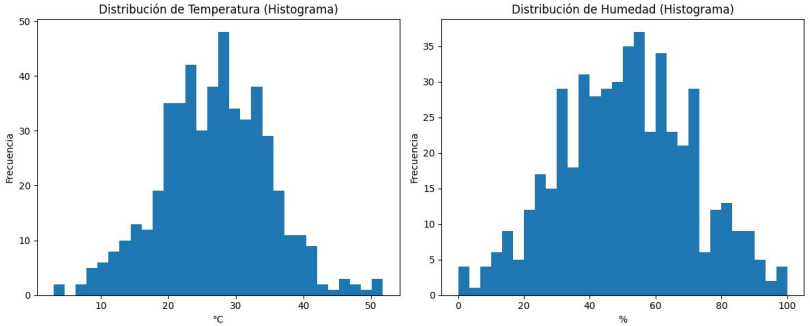
Simulación de Sensores

- Se generó la distribución normal para temperatura y humedad.
- El rango fue para los valores del sensor.
- La dirección de viento fue aleatoria entre 8 valores.



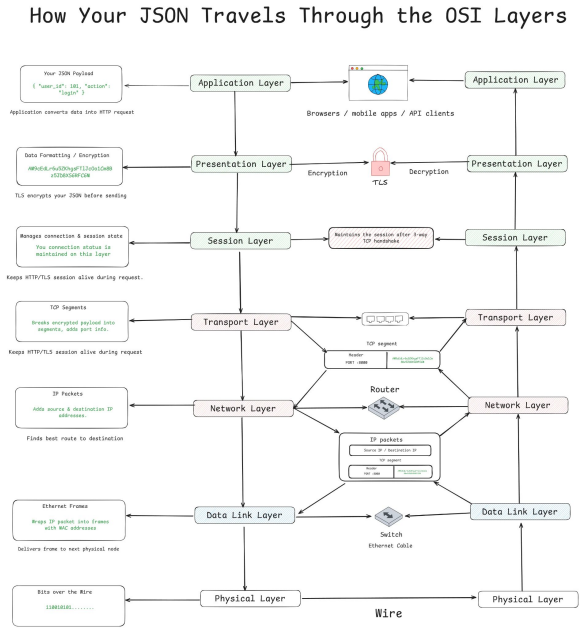
Evidencia de Distribución Normal

Se graficó la temperatura usando histogramas para ver la distribución de los datos generados para implementación encoding para 3 bytes.



¿A qué capa OSI pertenece JSON/SOAP?

Pertenecen a la capa de aplicación, porque son formatos de datos por aplicaciones para estructurar la información.

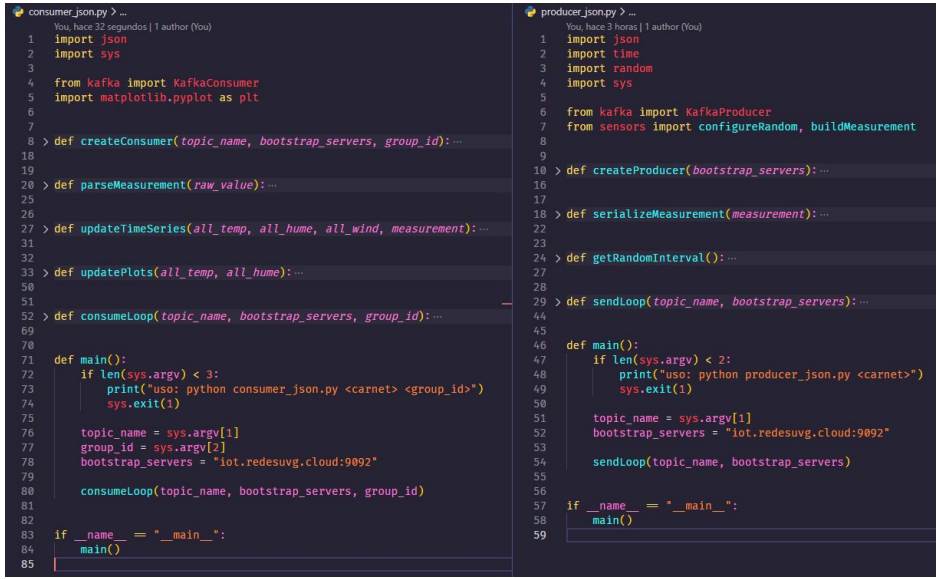


Beneficios de usar JSON/SOAP

- Permite desacoplar el producer y consumer, al no conocerse entre sí, sólo intercambian mensajes a través de Kafka y permite que el consumidor pueda interpretar el mensaje sin importar cómo esté implementado el productor.
- Mensajes cortos como json para data de temperatura y humedad, el consumer puede traducir este mensaje y en el caso del lab actualizar la gráfica sin negociación.
- En temas como IoT donde varios servicios avanzan o se actualizan se puede agregar otro campo y para algunos consumidores ignorarlos y seguir funcionando y además de favorecer interoperabilidad en arquitecturas distribuidas.

Funcionamiento Producer - Consumer (json)

Este fue el demo preparado para la uso de sensores y transporte de data sin usar explícitamente “encoding” para el transporte restringido a los bytes pedidos.



consumer_json.py > ...

You, hace 32 segundos | 1 author (You)

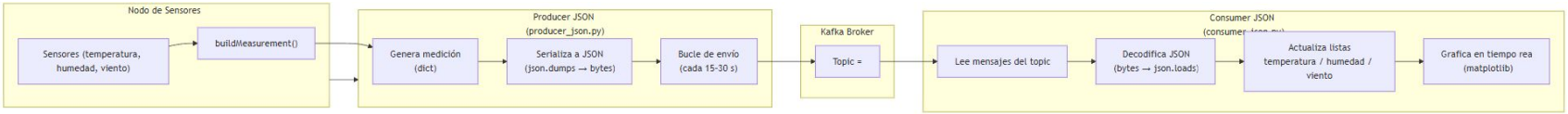
```python
import json
import sys

from kafka import KafkaConsumer
import matplotlib.pyplot as plt


def createConsumer(topic_name, bootstrap_servers, group_id):...


def parseMeasurement(raw_value):...


def updateTimeSeries(all_temp, all_hume, all_wind, measurement):...


def updatePlots(all_temp, all_hume):...


def consumeLoop(topic_name, bootstrap_servers, group_id):...


def main():
    if len(sys.argv) < 3:
        print("uso: python consumer_json.py <carnet> <group_id>")
        sys.exit(1)

    topic_name = sys.argv[1]
    group_id = sys.argv[2]
    bootstrap_servers = "iot.redesuvg.cloud:9092"

    consumeLoop(topic_name, bootstrap_servers, group_id)


if __name__ == "__main__":
    main()

```

producer_json.py > ...

You, hace 3 horas | 1 author (You)

```python
import json
import time
import random
import sys

from kafka import KafkaProducer
from sensors import configureRandom, buildMeasurement


def createProducer(bootstrap_servers):...


def serializeMeasurement(measurement):...


def getRandomInterval():...


def sendLoop(topic_name, bootstrap_servers):...


def main():
    if len(sys.argv) < 2:
        print("uso: python producer_json.py <carnet>")
        sys.exit(1)

    topic_name = sys.argv[1]
    bootstrap_servers = "iot.redesuvg.cloud:9092"

    sendLoop(topic_name, bootstrap_servers)


if __name__ == "__main__":
    main()

```

Pipeline IoT (json)



Restricción de 3 bytes

Tenemos 24 bits por lo tanto, cada dato del sensor debe comprimirse para los datos que nos llega:

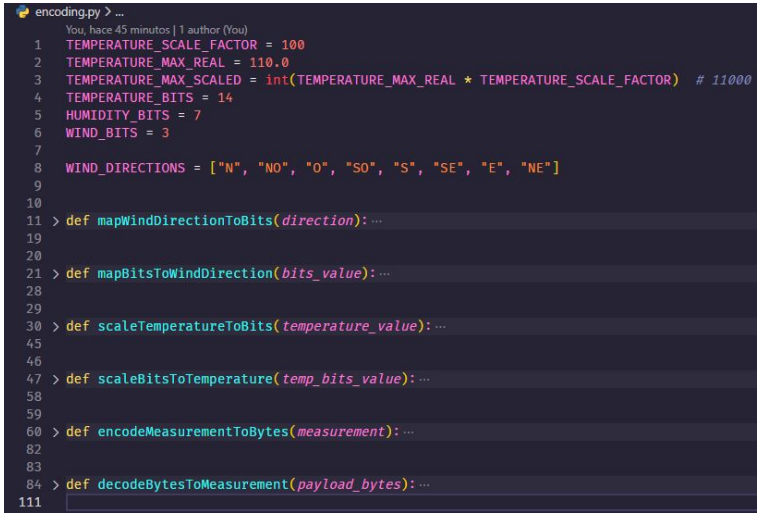
- 14 bits para temperatura, esto con el fin de conservar dos decimales sin usar explícitamente floats para rangos “válidos/coherentes” 0 - 110°C.
- 7 bits para humedad, al ir de 0 - 100, con la cantidad de bits definida llegamos a 127 valores y justamente cabe.
- 3 bits para la dirección del viento, como hay 8 posibilidades se asigna cada categoría a cada valor (0-7).

[Temperatura (14 bits) | Humedad (7 bits) | Viento (3 bits)]

Encoding

Se empacan los bits para la cantidad definida:

- Se mapea la dirección de viento a 0–7.
- Se escala la temperatura a entero.
- Se combinan bits y se generan 3 bytes.



¿Cómo hacer que temperatura quepa en 14 bits?

Se escala la temperatura, al multiplicar por 100 se eliminan decimales (2) y se almacena como entero. Con estos valores llegan a un valor de 16383 y justo para representar los valores en temperatura de 0-100ºC o su escala de 0-11000.

Complejidades del payload pequeño

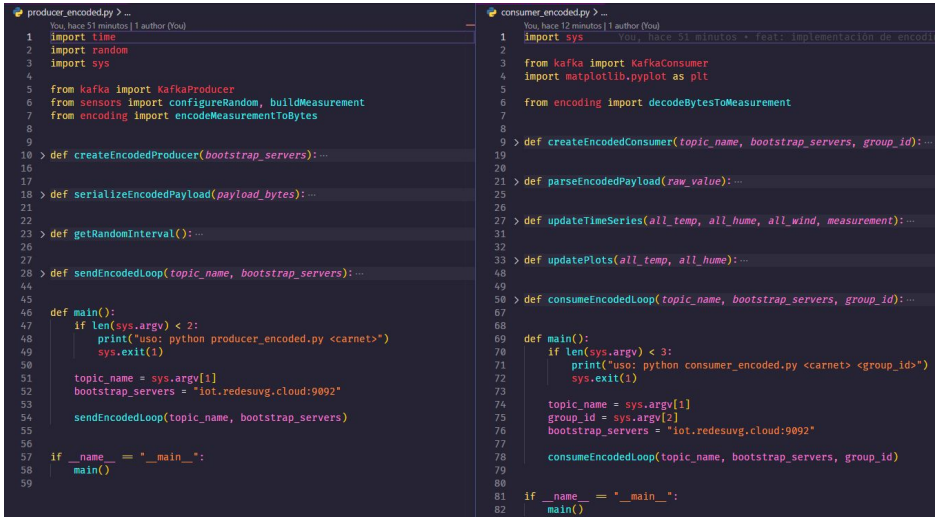
- No se puede enviar floats directamente (más trabajo consumidor).
- Se deben eliminar decimales, escalar y comprimir.
- Mayor riesgo de pérdida de precisión (en casos donde amerite).

¿Qué pasa si la humedad fuera float?

En el diseño ya no podrían utilizarse los 7 bits y por ende tendríamos que escalarla y esto implicaría aumentar los bits y por ende reducir la precisión o simplemente se eliminaría algunos decimales para ajustarlo al rango.

Funcionamiento Producer - Consumer (json)

El productor envía los bytes y medición y el consumer los decodifica y gráfica.



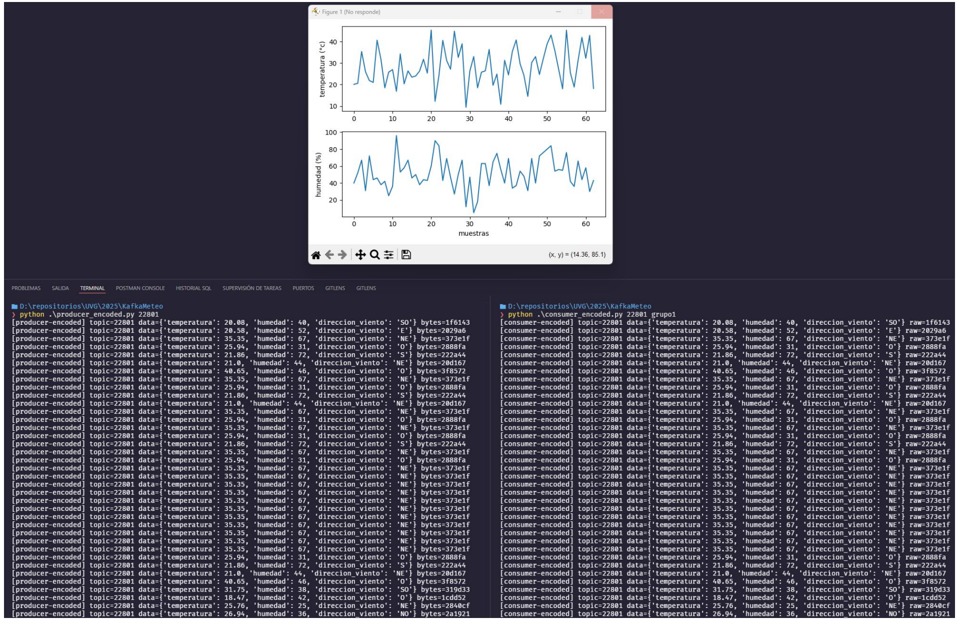
producer_encoded.py > ...

```python
import time
import random
import sys

from kafka import KafkaProducer
from sensors import configureRandom, buildMeasurement
from encoding import encodeMeasurementToBytes


def createEncodedProducer(bootstrap_servers):


def serializeEncodedPayload(payload_bytes):


def getRandomInterval():


def sendEncodedLoop(topic_name, bootstrap_servers):


def main():
    if len(sys.argv) < 2:
        print("uso: python producer_encoded.py <carnet>")
        sys.exit(1)

    topic_name = sys.argv[1]
    bootstrap_servers = "iot.redesuvg.cloud:9092"

    sendEncodedLoop(topic_name, bootstrap_servers)


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

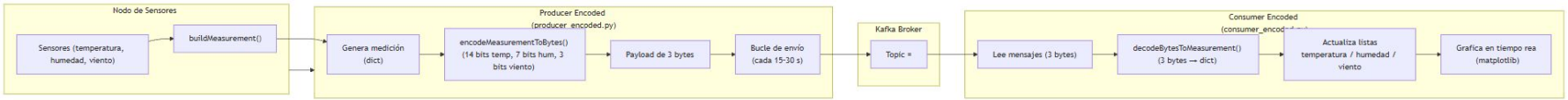
consumer_encoded.py > ...

```python
import sys

from kafka import KafkaConsumer
import matplotlib.pyplot as plt

from encoding import decodeBytesToMeasurement


def createEncodedConsumer(topic_name, bootstrap_servers, group_id):


def parseEncodedPayload(raw_value):


def updateTimeSeries(all_temp, all_hume, all_wind, measurement):


def updatePlots(all_temp, all_hume):


def consumeEncodedLoop(topic_name, bootstrap_servers, group_id):


def main():
    if len(sys.argv) < 3:
        print("uso: python consumer_encoded.py <carnet> <group_id>")
        sys.exit(1)

    topic_name = sys.argv[1]
    group_id = sys.argv[2]
    bootstrap_servers = "iot.redesuvg.cloud:9092"

    consumeEncodedLoop(topic_name, bootstrap_servers, group_id)


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

Funcionamiento Producer - Consumer (encoded)



Pipeline IoT (Encoded)



¿Para qué aplicaciones conviene Kafka?

- Para sistemas para consumo grande de datos.
- Streaming en tiempo real o sistemas distribuidos.

E intuitivamente para los casos contrarios no se conviene usar porque se pierde la utilidad de su uso como en dispositivos de bajo consumo como mensajes temporales o pequeños y cuando el IoT se está restringido.

Conclusiones

- Se comprendió cómo funcionan los flujos Publish/Subscribe y por qué son comunes en sistemas distribuidos e IoT.
- La reducción de JSON a un payload de 3 bytes mostró la diferencia entre un formato descriptivo y uno altamente optimizado para transporte.
- Se desarrollaron técnicas para convertir datos complejos en un formato compacto utilizando operaciones bitwise.