



# Estación Meteorológica con Apache Kafka

Implementación: [Enlace a repositorio](#)

Integrante: Josué Say



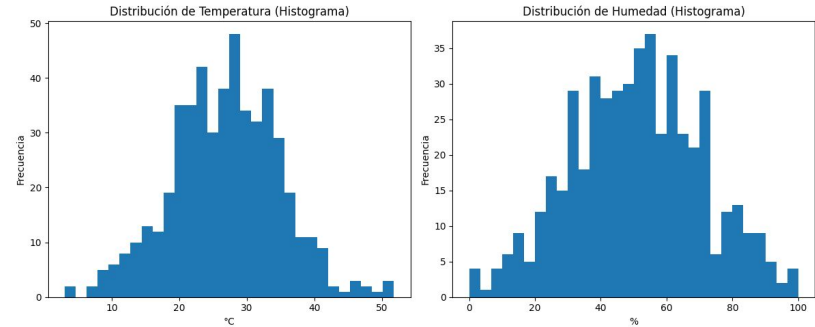
# Simulación de Sensores

- 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.

```
sensors.py X
sensors.py > ...
You, hace 13 horas | 1 author (You)
1  import random
2
3  # rangos para sensores
4  TEMPERATURE_MIN = 0.0
5  TEMPERATURE_MAX = 110.0
6  HUMIDITY_MIN = 0
7  HUMIDITY_MAX = 100
8
9  WIND_DIRECTIONS = ["N", "NO", "O", "SO", "S", "SE", "E", "NE"]
10
11
12 > def configureRandom(seed=None): ...
18
19
20 > def generateTemperature(): ...
29
30
31 > def generateHumidity(): ...
40
41
42 > def generateWindDirection(): ...
44
45
46 > def buildMeasurement(): ...
58 |
```

# Evidencia de Distribución Normal

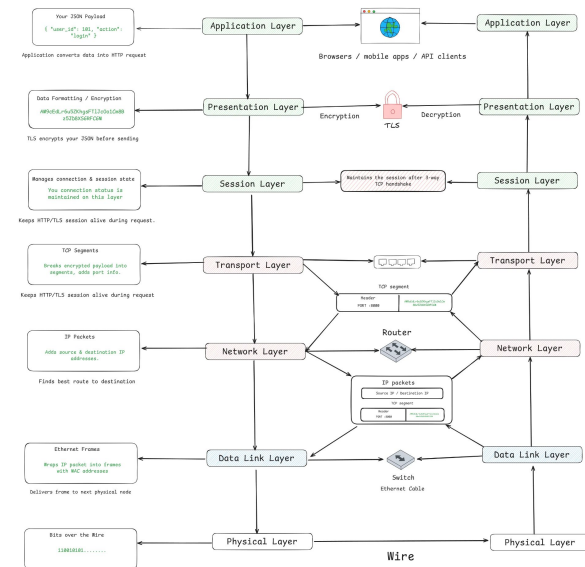
Se graficó la temperatura usando histogramas para ver la distribución de los datos generados para implementación encoding para 3 bytes.



# ¿A qué capa OSI pertenece JSON/SOAP?

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

How Your JSON Travels Through the OSI Layers





## 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.

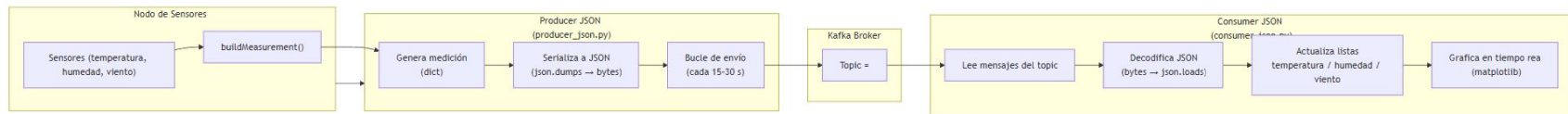
# Funcionamiento Producer - Consumer (json)

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)
1 import json
2 import sys
3
4 from kafka import KafkaConsumer
5 import matplotlib.pyplot as plt
6
7
8 > def createConsumer(topic_name, bootstrap_servers, group_id):...
18
19
20 > def parseMeasurement(raw_value):...
25
26
27 > def updateTimeSeries(all_temp, all_hume, all_wind, measurement):...
31
32
33 > def updatePlots(all_temp, all_hume):...
50
51
52 > def consumeLoop(topic_name, bootstrap_servers, group_id):...
69
70
71 def main():
72     if len(sys.argv) < 3:
73         print("uso: python consumer_json.py <carnet> <group_id>")
74         sys.exit(1)
75
76     topic_name = sys.argv[1]
77     group_id = sys.argv[2]
78     bootstrap_servers = "iot.redesuvg.cloud:9092"
79
80     consumeLoop(topic_name, bootstrap_servers, group_id)
81
82
83 if __name__ == "__main__":
84     main()
85

producer_json.py > ...
You, hace 3 horas | 1 author (You)
1 import json
2 import time
3 import random
4 import sys
5
6 from kafka import KafkaProducer
7 from sensors import configureRandom, buildMeasurement
8
9
10 > def createProducer(bootstrap_servers):...
16
17
18 > def serializeMeasurement(measurement):...
22
23
24 > def getRandomInterval():...
27
28
29 > def sendLoop(topic_name, bootstrap_servers):...
44
45
46 def main():
47     if len(sys.argv) < 2:
48         print("uso: python producer_json.py <carnet>")
49         sys.exit(1)
50
51     topic_name = sys.argv[1]
52     bootstrap_servers = "iot.redesuvg.cloud:9092"
53
54     sendLoop(topic_name, bootstrap_servers)
55
56
57 if __name__ == "__main__":
58     main()
59
```

# 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 empaquetan 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.

```
encoding.py > ...
You, hace 45 minutos | 1 author (You)
1  TEMPERATURE_SCALE_FACTOR = 100
2  TEMPERATURE_MAX_REAL = 110.0
3  TEMPERATURE_MAX_SCALED = int(TEMPERATURE_MAX_REAL * TEMPERATURE_SCALE_FACTOR) # 11000
4  TEMPERATURE_BITS = 14
5  HUMIDITY_BITS = 7
6  WIND_BITS = 3
7
8  WIND_DIRECTIONS = ["N", "NO", "O", "SO", "S", "SE", "E", "NE"]
9
10
11 > def mapWindDirectionToBits(direction): ...
19
20
21 > def mapBitsToWindDirection(bits_value): ...
28
29
30 > def scaleTemperatureToBits(temperature_value): ...
45
46
47 > def scaleBitsToTemperature(temp_bits_value): ...
58
59
60 > def encodeMeasurementToBytes(measurement): ...
82
83
84 > def decodeBytesToMeasurement(payload_bytes): ...
111
```



## ¿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 0-100°C 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)

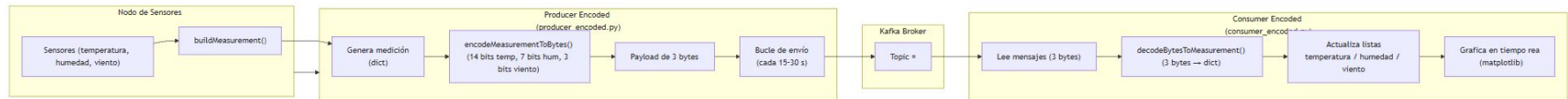
El productor envía los bytes y medición y el consumer los decodifica y gráfica.

```
producer_encoded.py > ...
1 import time
2 import random
3 import sys
4
5 from kafka import KafkaProducer
6 from sensors import configureRandom, buildMeasurement
7 from encoding import encodeMeasurementToBytes
8
9
10 > def createEncodedProducer(bootstrap_servers):...
11
12 > def serializeEncodedPayload(payload_bytes):...
13
14 > def getRandomInterval():...
15
16 > def sendEncodedLoop(topic_name, bootstrap_servers):...
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46 def main():
47     if len(sys.argv) < 2:
48         print("uso: python producer_encoded.py <carnet>")
49         sys.exit(1)
50
51     topic_name = sys.argv[1]
52     bootstrap_servers = "iot.redesuvg.cloud:9092"
53
54     sendEncodedLoop(topic_name, bootstrap_servers)
55
56
57 if __name__ == "__main__":
58     main()
59
```

```
consumer_encoded.py > ...
1 import sys
2
3 from kafka import KafkaConsumer
4 import matplotlib.pyplot as plt
5
6 from encoding import decodeBytesToMeasurement
7
8
9 > def createEncodedConsumer(topic_name, bootstrap_servers, group_id):...
10
11 > def parseEncodedPayload(raw_value):...
12
13 > def updateTimeSeries(all_temp, all_hume, all_wind, measurement):...
14
15 > def updatePlots(all_temp, all_hume):...
16
17
18 > def consumeEncodedLoop(topic_name, bootstrap_servers, group_id):...
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69 def main():
70     if len(sys.argv) < 3:
71         print("uso: python consumer_encoded.py <carnet> <group_id>")
72         sys.exit(1)
73
74     topic_name = sys.argv[1]
75     group_id = sys.argv[2]
76     bootstrap_servers = "iot.redesuvg.cloud:9092"
77
78     consumeEncodedLoop(topic_name, bootstrap_servers, group_id)
79
80
81 if __name__ == "__main__":
82     main()
```



# 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.