

UNIVERSIDAD DEL VALLE DE GUATEMALA - REDES

Internet of Things Estación Meteorológica

Cristian Fernando Laynez Bachez - 201281

Mario David de León Muralles - diesinuebeserodiesinuebe

Java

Este será el lenguaje de programación a utilizar para llevar a cabo dicha práctica.

Originalmente Apache Kafka fue desarrollado en Java, luego se implementó en 10 lenguajes más.



Instalación y Configuración del servidor con Apache Kafka

ESTA PARTE YA ESTA HECHA

Dicho servidor se encuentra en

lab9.alumchat.xyz

En la dirección IP:

157.245.244.105

Se utiliza el puerto estándar de Kafka:

9092

La Simulación de un Sensor

Los Sensores

Las estaciones meteorológicas poseen una gran variedad de sensores y datos que miden constantemente.

Sensor de temperatura (Termómetro)

Rango: [0, 100.00]°C. Float de dos decimales.

Sensor de Humedad Relativa (Higrómetro)

Rango: [0, 100]%. Entero.

Rango: [0, 100]%. Entero.

{N, NW, W, SW, S, SE, E, NE}.

Resultados de los Sensores

Termómetro

```
{sensorsOutput.json} 3
1  [
2  {
3    "direccion_viento": "SE",
4    "temperatura": 51.00490035306831,
5    "humedad": 32
6  },
7  {
8    "direccion_viento": "NW",
9    "temperatura": 56.47904670268041,
10   "humedad": 36
11 },
12 {
13   "direccion_viento": "NE",
14   "temperatura": 63.50139710444756,
15   "humedad": 48
16 },
17 {
18   "direccion_viento": "S",
19   "temperatura": 49.79242851813468,
20   "humedad": 72
21 },
22 {
23   "direccion_viento": "W",
24   "temperatura": 45.56549121991968,
25   "humedad": 45
26 },
27 {
28   "direccion_viento": "NE",
29   "temperatura": 52.235641424251114,
30   "humedad": 53
31 },
32 {
33   "direccion_viento": "NW",
34   "temperatura": 50.49054157871676,
35   "humedad": 84
36 },
37 {
38   "direccion_viento": "SE",
39   "temperatura": 39.02528547705488,
40   "humedad": 50
41 },
42 {
43   "direccion_viento": "SE",
44   "temperatura": 51.36584338685913,
45   "humedad": 47
46 },
47 {
48   "direccion_viento": "N"
```

Higrómetro

```
{sensorsOutput.json} 3
54  "temperatura": 48.377573556837994,
55  "humedad": 56
56 },
57 {
58   "direccion_viento": "E",
59   "temperatura": 43.56256653000107,
60   "humedad": 35
61 },
62 {
63   "direccion_viento": "NW",
64   "temperatura": 37.13647331137909,
65   "humedad": 66
66 },
67 {
68   "direccion_viento": "W",
69   "temperatura": 37.68323870509222,
70   "humedad": 60
71 },
72 {
73   "direccion_viento": "SE",
74   "temperatura": 52.02395976287511,
75   "humedad": 60
76 },
77 {
78   "direccion_viento": "W",
79   "temperatura": 33.3917115842775,
80   "humedad": 24
81 },
82 {
83   "direccion_viento": "NW",
84   "temperatura": 48.7846209292526,
85   "humedad": 38
86 },
87 {
88   "direccion_viento": "E",
89   "temperatura": 48.5072920146555,
90   "humedad": 59
91 },
92 {
93   "direccion_viento": "N",
94   "temperatura": 29.506885977841804,
95   "humedad": 30
96 },
97 {
98   "direccion_viento": "N",
99   "temperatura": 50.48827586477165,
100  "humedad": 30
```

Dirección de viento

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

JSON y SOAP pertenecen a la capa de Aplicación del Modelo OSI

Ambos son formatos de intercambio de datos usados para estructurar y transmitir información

La capa de Aplicación se encarga de la interacción entre las aplicaciones y los servicios de red, lo que incluye formatos y protocolos para el intercambio de datos

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

JSON

Eficiente y Fácil de Usar: Ligero, fácil de leer y escribir; compatible con muchos lenguajes de programación.

Rápido para Web y Móviles: Ideal para la transmisión de datos en aplicaciones web y móviles, especialmente en AJAX.

SOAP

Altamente Estandarizado y Seguro: Ofrece rigurosos estándares y soporte para seguridad avanzada.

Flexible y Extensible: Independiente del protocolo de transporte, extensible para necesidades específicas como fiabilidad en la entrega de mensajes.

Envío de Datos al Server Edge

Implementacion Producer

```
// mvn clean install exec:java -Dexec.mainClass=firstapp.ProducerKafka
public static void main(String args[]) {
    Properties props = new Properties();
    props.put("bootstrap.servers", bootstrapServers);
    props.put("client.id", tenancyName);
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

    Producer<String, String> producer = new KafkaProducer<>(props);

    try {
        while (true) {
            JSONObject data = Sensors.generateJsonData();
            String key = "sensor1";

            // Enviar el mensaje al topic
            producer.send(new ProducerRecord<>(TOPIC, key, data.toString()));
            Thread.sleep(Sensors.randomIntNumber(min:15000, max:30000));
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        // Cerrar el productor al salir
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
23:22:51.564 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Node 1 has finalized features epoch: 0, finalized features: [], supported features: [], API versions: (Produce(0): 0 to 9 [usable: 9], Fetch(1): 0 to 13 [usable: 12], ListOffsets(2): 0 to 7 [usable: 6], Metadata(3): 0 to 12 [usable: 11], LeaderAndIsr(4): 0 to 6 [usable: 5], StopReplica(5): 0 to 3 [usable: 3], UpdateMetadata(6): 0 to 7 [usable: 7], ControlledShutdown(7): 0 to 3 [usable: 3], OffsetCommit(8): 0 to 8 [usable: 8], OffsetFetch(9): 0 to 8 [usable: 7], FindCoordinator(10): 0 to 4 [usable: 3], JoinGroup(11): 0 to 9 [usable: 7], Heartbeat(12): 0 to 4 [usable: 4], LeaveGroup(13): 0 to 5 [usable: 4], SyncGroup(14): 0 to 5 [usable: 5], DescribeGroups(15): 0 to 5 [usable: 5], ListGroups(16): 0 to 4 [usable: 4], SaslHandshake(17): 0 to 1 [usable: 1], ApiVersions(18): 0 to 3 [usable: 3], CreateTopics(19): 0 to 7 [usable: 7], DeleteTopics(20): 0 to 6 [usable: 6], DeleteRecords(21): 0 to 2 [usable: 2], InitProducerId(22): 0 to 4 [usable: 4], OffsetForLeaderEpoch(23): 0 to 4 [usable: 4], AddPartitionsToTxn(24): 0 to 3 [usable: 3], AddOffsetsToTxn(25): 0 to 3 [usable: 3], EndTxn(26): 0 to 3 [usable: 3], WriteTxnMarkers(27): 0 to 1 [usable: 1], TxnOffsetCommit(28): 0 to 3 [usable: 3], DescribeAcls(29): 0 to 3 [usable: 2], CreateAcls(30): 0 to 3 [usable: 2], DeleteAcls(31): 0 to 3 [usable: 2], DescribeConfigs(32): 0 to 4 [usable: 4], AlterConfigs(33): 0 to 2 [usable: 2], AlterReplicaLogDirs(34): 0 to 2 [usable: 2], DescribelogDirs(35): 0 to 4 [usable: 2], SaslAuthenticate(36): 0 to 2 [usable: 2], CreatePartitions(37): 0 to 3 [usable: 3], CreateDelegationToken(38): 0 to 3 [usable: 2], RenewDelegationToken(39): 0 to 2 [usable: 2], ExpireDelegationToken(40): 0 to 2 [usable: 2], DescribeDelegationToken(41): 0 to 3 [usable: 2], DeleteGroups(42): 0 to 2 [usable: 2], ElectLeaders(43): 0 to 2 [usable: 2], IncrementalAlterConfigs(44): 0 to 1 [usable: 1], AlterPartitionReassignments(45): 0 [usable: 0], ListPartitionReassignments(46): 0 [usable: 0], OffsetDelete(47): 0 [usable: 0], DescribeClientQuotas(48): 0 to 1 [usable: 1], AlterClientQuotas(49): 0 to 1 [usable: 1], DescribeUserScramCredentials(50): 0 [usable: 0], AlterUserScramCredentials(51): 0 [usable: 0], AlterIsr(56): 0 to 2 [usable: 0], UpdateFeatures(57): 0 to 1 [usable: 0], DescribeCluster(60): 0 [usable: 0], DescribeProducers(61): 0 [usable: 0], UNKNOWN(65): 0, UNKNOWN(66): 0, UNKNOWN(67): 0).
23:22:51.577 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Sending PRODUCE request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=3) and timeout 30000 to node 1: {acks=1,timeout=30000,partitionSizes=[12345-0=147]}
23:22:51.701 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received PRODUCE response from node 1 for request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=3): ProduceResponseData(responses=[TopicProduceResponse(name='12345', partitionResponses=[PartitionProduceResponse(index=0, errorCode=0, baseOffset=18, logAppendTimeMs=-1, logStartOffset=0, recordErrors=[], errorMessage=null)]), throttleTimeMs=0)
23:23:11.094 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Sending PRODUCE request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=4) and timeout 30000 to node 1: {acks=1,timeout=30000,partitionSizes=[12345-0=147]}
23:23:11.263 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received PRODUCE response from node 1 for request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=4): ProduceResponseData(responses=[TopicProduceResponse(name='12345', partitionResponses=[PartitionProduceResponse(index=0, errorCode=0, baseOffset=19, logAppendTimeMs=-1, logStartOffset=0, recordErrors=[], errorMessage=null)]), throttleTimeMs=0)
23:23:31.387 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Sending PRODUCE request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=5) and timeout 30000 to node 1: {acks=1,timeout=30000,partitionSizes=[12345-0=147]}
23:23:31.565 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received PRODUCE response from node 1 for request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=5): ProduceResponseData(responses=[TopicProduceResponse(name='12345', partitionResponses=[PartitionProduceResponse(index=0, errorCode=0, baseOffset=20, logAppendTimeMs=-1, logStartOffset=0, recordErrors=[], errorMessage=null)]), throttleTimeMs=0)
23:23:31.593 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Sending PRODUCE request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=6) and timeout 30000 to node 1: {acks=1,timeout=30000,partitionSizes=[12345-0=146]}
23:23:51.525 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received PRODUCE response from node 1 for request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=6): ProduceResponseData(responses=[TopicProduceResponse(name='12345', partitionResponses=[PartitionProduceResponse(index=0, errorCode=0, baseOffset=21, logAppendTimeMs=-1, logStartOffset=0, recordErrors=[], errorMessage=null)]), throttleTimeMs=0)
```



Consumir y Desplegar Datos Meteorológicos

Implementacion Consumer

The screenshot shows a Java code editor with a code editor window titled "ConsumerKafka.java 1, u" and a terminal window below it.

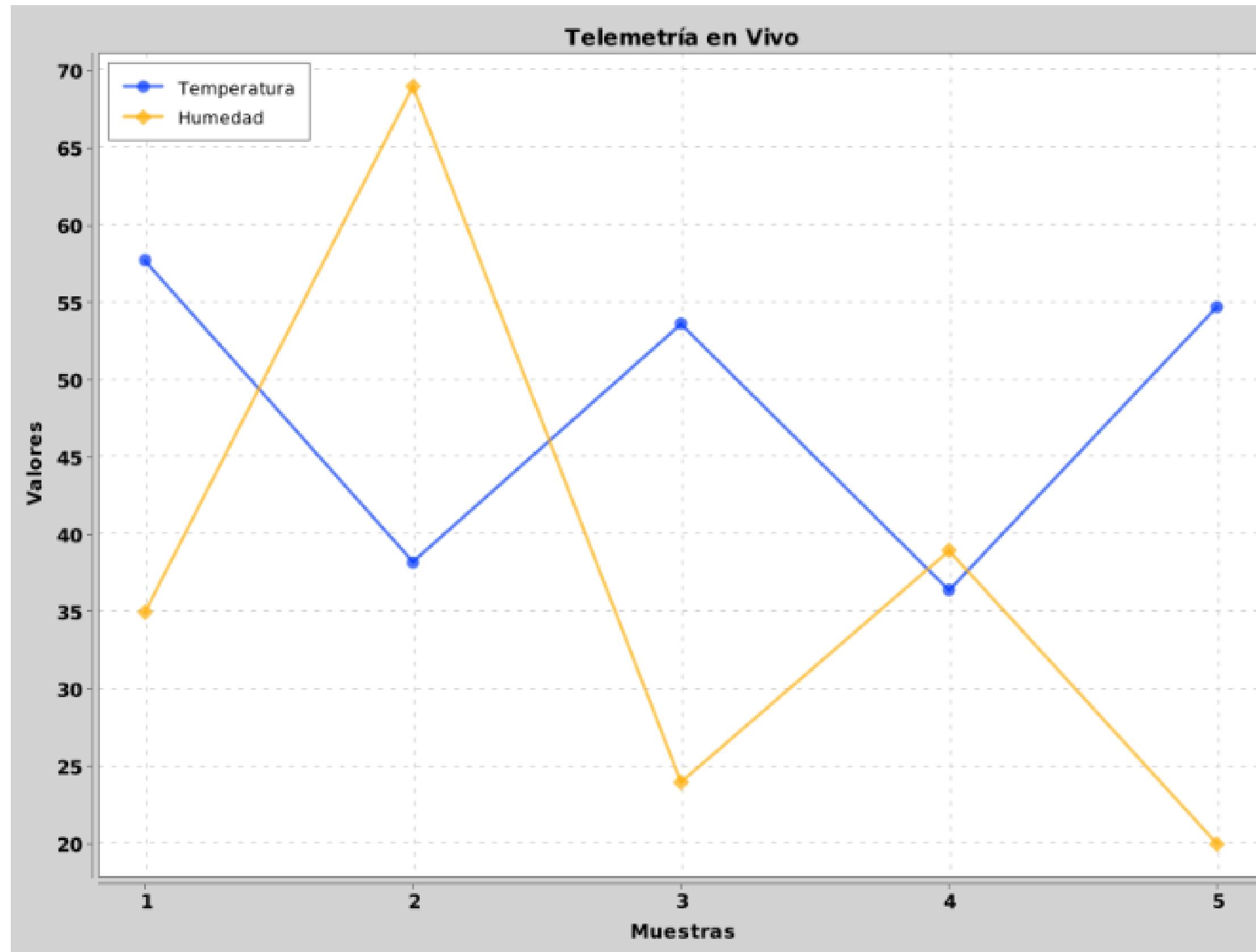
Code Editor Content:

```
src > main > java > firstapp > ConsumerKafka.java > updateChart(XYChart, List<Double>, List<Integer>, List<String>)
48     try {
49         while (true) {
50             ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
51
52             records.forEach(record -> {
53                 String message = record.value();
54                 JSONObject json = new JSONObject(message);
55
56                 listTemp.add(json.getDouble(key:"temperatura"));
57                 listHume.add(json.getInt(key:"humedad"));
58                 listWind.add(json.getString(key:"direccion_viento"));
59
60                 System.out.println("\n====> " + message + "\n");
61                 updateChart(chart, listTemp, listHume, listWind);
62             });
63         } finally {
64             consumer.close();
65         }
66     }
67
68
69     private static void updateChart(XYChart chart, List<Double> allTemp, List<Integer> allHume, List<String> allWind) {
70         chart.removeSeries(seriesName:"Temperatura");
71         chart.removeSeries(seriesName:"Humedad");
72
73         chart.addSeries(seriesName:"Temperatura", xData:null, allTemp);
74         chart.addSeries(seriesName:"Humedad", xData:null, allHume);
75     }
76
77
78     PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

Terminal Log Output:

```
ons=false, includeTopicAuthorizedOperations=false)
00:07:08.710 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received METADATA response from node 1 for request with header RequestHeader(apiKey=METADATA, apiVersion=11, clientId=sensor-data-producer, correlationId=127): MetadataResponseData(throttleTimeMs=0, brokers=[MetadataResponseBroker(nodeId=1, host='lab9.alumchat.xyz', port=9092, rack=null)], clusterId='BQmNLrmPTieD59aPg-7_7w', controllerId=1, topics=[MetadataResponseTopic(errorCode=0, name='12345', topicId=1DXXXB8g7Rl-r0fp2UmvouQ, isInternal=false, partitions=[MetadataResponsePartition(errorCode=0, partitionIndex=0, leaderId=1, leaderEpoch=0, replicaNodes=[1], isrNodes=[1], offlineReplicas=[])]), topicAuthorizedOperations=-2147483648], clusterAuthorizedOperations=-2147483648)
00:07:08.711 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.Metadata - [Producer clientId=sensor-data-producer] Updating last seen epoch for partition 12345-0 from 0 to epoch 0 from new metadata
00:07:08.712 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Updated cluster metadata updateVersion 11 to MetadataCache{clusterId='BQmNLrmPTieD59aPg-7_7w', nodes={1=lab9.alumchat.xyz:9092 (id: 1 rack: null)}, partitions=[PartitionMetadata(error=NONE, partition=12345-0, leader=Optional[1], leaderEpoch=Optional[0], replicas=1, isr=1, offlineReplicas=[])], controller=lab9.alumchat.xyz:9092 (id: 1 rack: null)}
00:07:15.415 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Sending PRODUCE request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=128) and timeout 30000 to node 1: {acks=1,timeout=30000,partitionSizes=[12345-0=148]}
00:07:15.574 [kafka-producer-network-thread | sensor-data-producer] DEBUG org.apache.kafka.clients.NetworkClient - [Producer clientId=sensor-data-producer] Received PRODUCE response from node 1 for request with header RequestHeader(apiKey=PRODUCE, apiVersion=9, clientId=sensor-data-producer, correlationId=128): ProduceResponseData(responses=[TopicProduceResponse(name='12345', partitionResponses=[PartitionProduceResponse(index=0, errorCode=0, baseOffset=134, logAppendTimeMs=-1, logStartOffset=0, recordErrors=[], errorMessage=null)])), throttleTimeMs=0
clientId=consumer-foo2-1, groupId=foo2] Sending READ_UNCOMMITTED IncrementalFetchRequest(toSend=(), toForget=(), implied=(12345-0)) to broker lab9.alumchat.xyz:9092 (id: 1 rack: null)
00:07:30.232 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.NetworkClient - [Consumer clientId=consumer-foo2-1, groupId=foo2] Sending FETCH request with header RequestHeader(apiKey=FETCH, apiVersion=12, clientId=consumer-foo2-1, correlationId=128) and timeout 30000 to node 1: FetchRequestData(clusterId=null, replicaId=-1, maxWaitMs=500, minBytes=1, maxBytes=52428800, isolationLevel=0, sessionId=1101332170, sessionEpoch=85, topics=[], forgottenTopicsData=[], rackId='')
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.NetworkClient - [Consumer clientId=consumer-foo2-1, groupId=foo2] Received FETCH response from node 1 for request with header RequestHeader(apiKey=FETCH, apiVersion=12, clientId=consumer-foo2-1, correlationId=128): FetchResponseData(throttleTimeMs=0, errorCode=0, sessionId=1101332170, responses[])
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.FetchSessionHandler - [Consumer clientId=consumer-foo2-1, groupId=foo2] Node 1 sent an incremental fetch response with throttleTimeMs = 0 for session 1101332170 with 0 response partition(s), 1 implied partition(s)
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.consumer.internals.Fetcher - [Consumer clientId=consumer-foo2-1, groupId=foo2] Added READ_UNCOMMITTED fetch request for partition 12345-0 at position FetchPosition{offset=135, offsetEpoch=Optional[0], currentLeader=LeaderAndEpoch{leader=Optional[lab9.alumchat.xyz:9092 (id: 1 rack: null)], epoch=0}} to node lab9.alumchat.xyz:9092 (id: 1 rack: null)
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.FetchSessionHandler - [Consumer clientId=consumer-foo2-1, groupId=foo2] Built incremental fetch (sessionId=1101332170, epoch=86) for node 1. Added 0 partition(s), altered 0 partition(s), removed 0 partition(s) out of 1 partition(s)
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.consumer.internals.Fetcher - [Consumer clientId=consumer-foo2-1, groupId=foo2] Sending READ_UNCOMMITTED IncrementalFetchRequest(toSend=(), toForget=(), implied=(12345-0)) to broker lab9.alumchat.xyz:9092 (id: 1 rack: null)
00:07:31.041 [firstapp.ConsumerKafka.main()] DEBUG org.apache.kafka.clients.NetworkClient - [Consumer clientId=consumer-foo2-1, groupId=foo2] Sending FETCH request with header RequestHeader(apiKey=FETCH, apiVersion=12, clientId=consumer-foo2-1, correlationId=129) and timeout 30000 to node 1: FetchRequestData(clusterId=null, replicaId=-1, maxWaitMs=500, minBytes=1, maxBytes=52428800, isolationLevel=0, sessionId=1101332170, sessionEpoch=86, topics=[], forgottenTopicsData=[], rackId='')
```

Implementacion Consumer



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

Ventajas

Alta Escalabilidad y Rendimiento: gestión de grandes volúmenes de datos y alta concurrencia, ideal para big data.

Desacoplamiento de Productores y Consumidores: Permite flexibilidad en el procesamiento y distribución de mensajes.

Desventajas:

Complejidad de Configuración y Gestión:
Requiere conocimientos técnicos para configuración y mantenimiento adecuados.

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

Adecuadas para Kafka:

Sistemas de Logs y Monitoreo en Tiempo Real: Para procesar y analizar grandes volúmenes de logs y datos en tiempo real.

Plataformas de Procesamiento de Eventos: Ideal para sistemas que requieren manejo de eventos a gran escala.

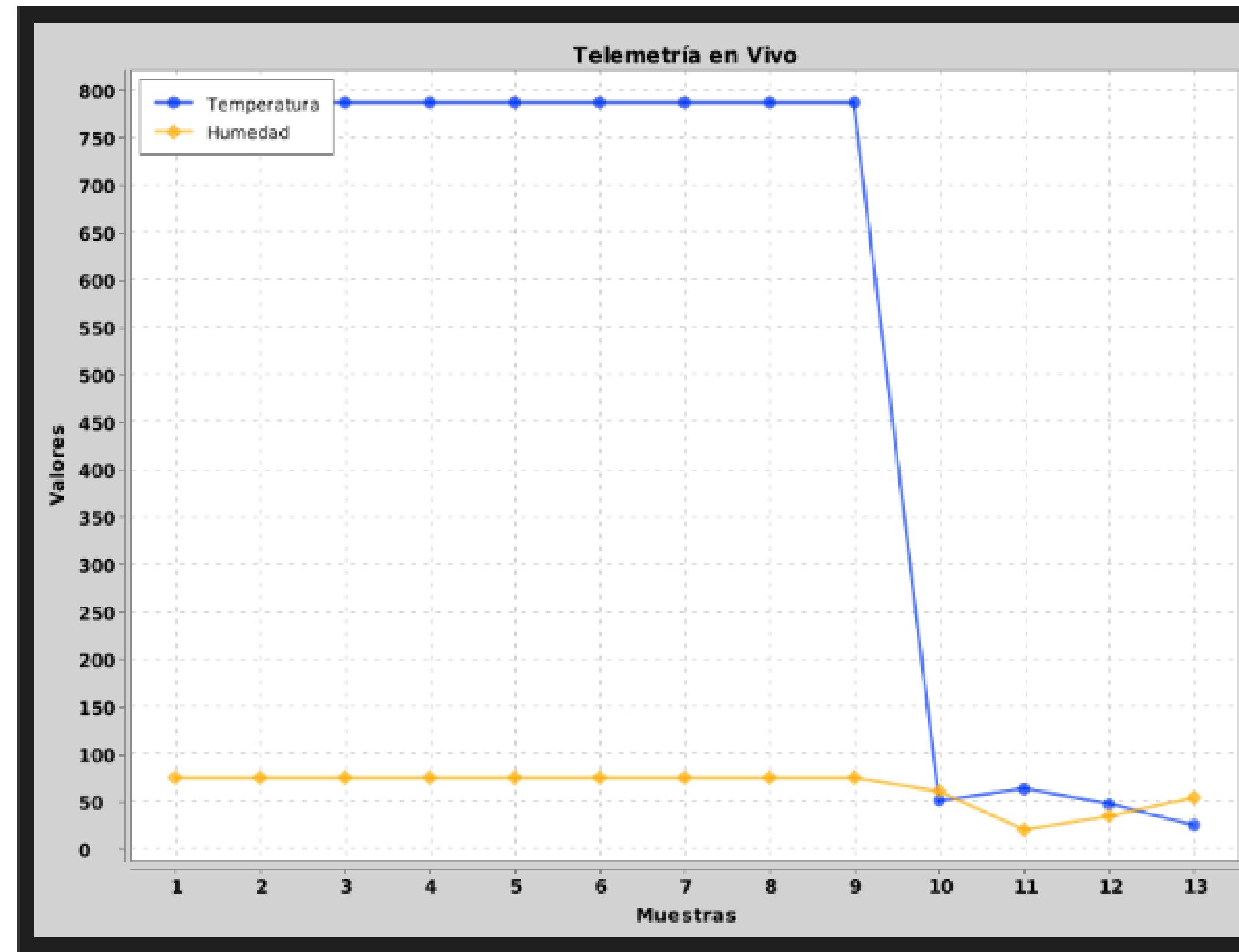
Inadecuadas para Kafka:

Transacciones con Necesidad de Respuesta Inmediata: No es óptimo para sistemas que requieren garantías fuertes de entrega de mensajes en tiempo real.

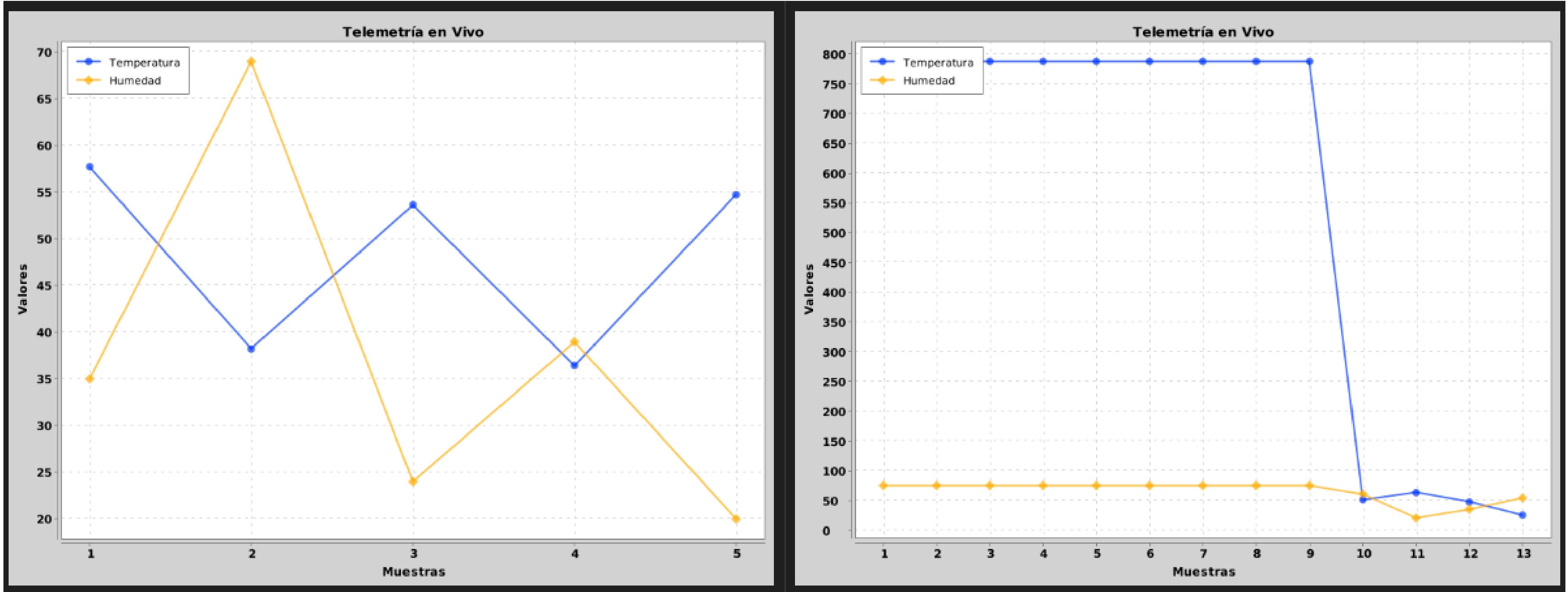


IoT en Entornos con Restricciones

Gráfica Codec



Antes Vs Despues



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

- LA LIMITACIÓN DE DATOS RESTRINGE SIGNIFICATIVAMENTE LA CANTIDAD Y EL DETALLE DE LA INFORMACIÓN TRANSMITIDA.
- SE REQUIERE UNA CODIFICACIÓN COMPLEJA PARA INCLUIR LA MAYOR CANTIDAD DE INFORMACIÓN EN UN ESPACIO LIMITADO.
- LA COMPRESIÓN NECESARIA PARA AJUSTARSE AL PAYLOAD PEQUEÑO PUEDE REDUCIR LA PRECISIÓN DE LOS DATOS.
- LOS PAYLOADS PEQUEÑOS AUMENTAN EL RIESGO DE ERRORES, NECESITANDO ESTRATEGIAS DE GESTIÓN DE ERRORES MÁS ROBUSTAS.

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

SE NECESITARÍA AJUSTAR EL RANGO
DE TODOS LOS VALORES POSIBLES
EN LA TEMPERATURA Y CODIFICARLO
DE LA MEJOR MANERA POSIBLE.

Se puede tomar en cuenta convertir los bits a
números neteros, se puede confiar el entero
en bits.

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

SE PODRÍA TOMAR EN CUENTA LAS SIGUIENTES CONSIDERACIONES:

- Se puede llevar a cabo un proceso de escala y redondeo para que los valores encajen en el rango deseado de bits.
- Se puede verificar el tipo de dato para la codificación para decidir cómo codificarlos en bits.
- Y por supuesto se puede tomar en cuenta el lado del consumidor, este sería implementado el proceso inverso.

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

SE PUEDE TOMAR EN CUENTA LAS SIGUIENTES TECNICAS Y HERRAMIENTAS

- Message Compression: Kafka puede comprimir mensajes antes de enviarlos y descomprimirlos en el lado del consumidor.
- Partición de mensajes: Si dado caso los mensajes son más grandes de lo normal entonces se puede partir por piezas pequeñas.
- Avro Serialization: Es un sistema de serialización que proporciona una representación compacta de los datos, el cual puede ser beneficioso para reducir el tamaño de los mensajes.