0) Checklist (on the QuickStart VM)

Yes — there **is** a way to consume Kafka messages in Spark **without using the Spark–Kafka jar**, but with a workaround. The key idea is: since Spark can natively read from **sockets**, we can **pipe Kafka messages to a TCP port** and let Spark read from that port.

This method works for testing or small projects, but it **does not give the reliability or offset management** of the official Kafka-Spark integration.

Here's a complete setup using your existing Kafka 0.8 tgz on Cloudera QuickStart VM:

1️⃣ Start Zookeeper & Kafka

Open one terminal:

```bash
# Go to Kafka folder
cd ~/kafka_2.10-0.8.2.2

# Start Zookeeper
bin/zookeeper-server-start.sh config/zookeeper.properties
```

Open a **second terminal**:

```bash
# Start Kafka broker
bin/kafka-server-start.sh config/server.properties
```

2️⃣ Create a test topic

Open a **third terminal**:

```bash
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --
topic test-topic
```

Check it:

```bash
bin/kafka-topics.sh --list --zookeeper localhost:2181
# Output should be: test-topic
```

3️⃣ Pipe Kafka messages to TCP port

We will forward messages to port **`9999`** using **`kafka-console-consumer.sh`** and **`nc`**:

```bash
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test-topic --from-beginning |
nc -lk 9999
```

> Notes:
>
> - **`-l`** → listen mode
> - **`-k`** → keep listening after client disconnect (may not be supported on all **`nc`**)

Leave this terminal running.

4️⃣ Spark Streaming consumer (PySpark)

Create **`spark_socket_consumer.py`**:

```python
from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext

conf = SparkConf().setAppName("SocketKafkaForwardConsumer").setMaster("local[2]")
sc = SparkContext(conf=conf)
ssc = StreamingContext(sc, 2)  # 2-second batch interval

lines = ssc.socketTextStream("localhost", 9999)

def process(rdd):
    count = rdd.count()
    if count > 0:
        print(f"Received {count} records in this batch")
        for i, record in enumerate(rdd.take(10), start=1):
            print(f"[{i}] {record}")
    else:
        print("No records in this batch")

lines.foreachRDD(process)

ssc.start()
ssc.awaitTermination()
```

Run it:

```bash
$SPARK_HOME/bin/spark-submit --master local[2] spark_socket_consumer.py
```

5️⃣ Produce messages to Kafka

In another terminal:

```bash
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test-topic
```

Type messages and press Enter. You should see them appear in the PySpark consumer terminal.

✅ Advantages / Limitations

Advantages:

- No extra jar needed.
- Works with your existing Kafka tgz.
- Quick testing on Cloudera VM.

Limitations:

- No offset tracking → if Spark crashes, you lose messages.
- Not suitable for production.
- You cannot scale reliably to multiple Spark workers without extra scripting.

If you want, I can also give a **more robust version** using a **small Python TCP forwarder** that handles client reconnects and keeps Kafka messages flowing to Spark — safer than `nc`.

Do you want me to show that?