**Tutorial Link:** <https://www.youtube.com/watch?v=EiDLKECLcZw&t=524s>

**Dependencies used:**

**node-rdkafka to interact with kafka in nodejs.**

**Avsc to serialize and deserialize objects (not needed for plain texts)**

1. We setup the **docker-compose.yml** for the Kafka integration and initiate the file using **docker-compose up** command. This will setup the Kafka containers (dependency zookeeper and Kafka) up and running.
2. We then need to create a topic. For that we run this command in a separate terminal (same target location where the project is being built)

**docker exec -it kafka1 /opt/bitnami/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test\_kafka\_topic\_1**

**name of Kafka container as defined in docker-compose.yml**

**target Kafka topic location (list of topics)**

**this is a replacement for zookeeper keyword (deprecated)**

**new topic name**

1. Kafka is now setup and running on docker. Now, **npm init** to create the **package.json** and create **two folders** with **index.js** inside **both folders**. The folders should be **consumer** and **producer**. So, **in each folder there is a dedicated index.js inside**.
2. **Change the scripts in package.json** so that both can run separately.

"scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start:producer": "node ./producer/index.js",

    "start:consumer": "node ./consumer/index.js"

  },

We run the command npm run start:producer to run the **index.js** inside **producer** and same for **consumer**.

1. Install the dependencies described in the top.
   1. **npm i node-rdkafka**
   2. **npm i avsc**
2. Start coding with producer -> index.js (coding information is written in the code itself)
3. Then move to consumer
4. Remember: **When the consumer is stopped but the producer is still running, as soon as the consumer started, all the events will be read by consumer that were missed.**
5. To write objects (instead of just strings), **the objects should be serialized and deserialized** unlike strings. Objects or other data structures should be written into a buffer before adding into the data stream.

This is done using **avsc** library. This is quick and **faster than JSON.**

1. Add a new file called **eventType.js** where the type of object, etc. is defined using avsc.

(**Refer** [**https://www.npmjs.com/package/avsc**](https://www.npmjs.com/package/avsc) **to get a clear idea on defining event types**)

1. **Then, use the event type defined in eventType.js to consume and produce events in producer and consumer.**