Data Engineering Project

How to run ?

1. bin/zookeeper-server-start.sh config/**zookeeper.properties** (Go to kafka folder and run this in a terminal #1)
2. bin/kafka-server-start.sh config/**server.properties** (Go to kafka folder and run this in a NEW terminal #2)
3. Run the producer (Terminal -> sbt run -> 2)
4. Run the consumer (Terminal -> sbt run -> 1)
5. Or simply run them by clicking on the "Run" button that is on the left side of the scala object

Spark Installation - Configuration :

brew install spark
`spark-submit -version`
add library dependencies in the build.sbt
sbt update
close and reopen the project
To locate the installation path and other details of ***Spark***, enter the following command:
brew info apache-spark

Problem : WARN org.apache.spark.util.Utils - Your hostname, Meliss-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; using 192.168.1.18 instead (on interface en0)
[sbt-bg-threads-1] WARN org.apache.spark.util.Utils - Set SPARK_LOCAL_IP if you need to bind to another address

https://vinta.ws/code/setup-spark-on-macos.html

In a terminal run in order ->
cd $SPARK_HOME
export SPARK_LOCAL_IP=127.0.0.1

Spark Streaming :

1. Create Spark Session
2. Read the input DF from Kafka (.readStream -> reads each line from the source, .format() -> the format you're reading, returns a dataframe but you cannot do actions on this like .count()!!!)
3. Print the data schema of this data frame .printSchema()
4. *Create a data schema*
5. *Parse each line of the "value" of the Kafka message*
6. Transform/Process the data (word count, filter etc.)
7. Write the output DF into the stream (.format(console/json/avro/csv…), .outputMode(append/ complete/…))

.start() -> starts a background Spark job and this is a non-blocking method that runs in the background and returns a Streaming Query Object. So we must wait for the background job to be complete. For this, we use .awaitTermination(). A stream normally runs forever so this method allows the stream to end when we want to kill the streaming application for maintenance reasons or the streaming app. faces some error/exceptions. In both cases we want our application to end gracefully so we add this configuration to our Spark
Session : .config("spark.streaming.stopGracefullyOnShutdown", "true")

https://sparkbyexamples.com/spark/spark-streaming-with-kafka/

Serialization - Deserialization

/*
//By nature, json objects are unordered. This creates a problem in the Spark stream when reading the columns.

```
Json.toJson(Map("droneId" -> droneId.toString, "longitude" -> longitude.toString, "latitude" ->
latitude.toString,
  "timestamp" -> timestamp.toString, "citizens" -> citscore, "words" -> randWords.mkString(",")))
*/


//The distribution of alerts depending on the day of the week
/*
println("The distribution of alerts depending on the day of the week")
val alertsOnDay = convertedDF.select(expr("date_format(convertedTimestamp, 'EEEE') as day"))
  .select("day")
  //.groupBy("day")
  //.count()
  //.orderBy(col("count").desc)
alertsOnDay.show()*/
```