Table of Contents

NOTE : PLEASE UPDATE THE CONFLUENT PATH TO YOUR PATH AS AND WHERE NEEDED

NOTE: win-scripts folder needs to be copied in every new project that is imported. Make the changes as and when mentioned for a particular project.

ALSO MAKE SURE TO STOP THE SERVER, BROKER, CONSUMERS BEFORE MOVING ON TO NEXT PROJECT

1. Callback Function:

1.1. Whether the data was correctly produced, where it was produced, retrieving its offset and partition value, etc. we need to implement a callback function. This function is implemented for asynchronously handling the request completion. That's why its return type will be void. This function will be implemented in the block where the producer sends data to Kafka.

1.2. The callback function invoked by the producer is onCompletion(). Basically, this method requires two arguments:
- 1.2.1. Metadata of the Record: Metadata of the record e.g. information regarding the partition and its offsets.
- 1.2.2. Exception: Any exception thrown during the produce request

1.3. Create a new topic as callback with partition 3 and replication factor 3

1.4. Modify the producer.send() as follows to use the callback function

```
producer.send(new ProducerRecord<>("callback", i, "Msg" + i), new Callback() {
   public void onCompletion(RecordMetadata recordMetadata, Exception e) {
      // executes every time a record is successfully sent or an exception is thrown
      if (e == null) {
         // the record was successfully sent
         System.out.println ("Received new metadata. \n" +
               "Topic:" + recordMetadata.topic() + "\n" +
               "Partition: " + recordMetadata.partition() + "\n" +
               "Offset: " + recordMetadata.offset() + "\n" +
               "Timestamp: " + recordMetadata.timestamp());
      } else {
         System.out.println ("Error while producing", e);
      }
   }
});
```

1.5. You can start the consumer if in case you want to see the messages consumed

2. ISR and acks

2.1. **Make sure all your 3 brokers are up and running**

2.2. Create a topic as follows and configure in sync replicas while creating topic

```
$KAFKA_HOME/bin/kafka-topics --create --bootstrap-server localhost:9092,localhost:9003 --topic test --partitions 3 --
replication-factor 3 --config min.insync.replicas=3
```

2.3. Once this is done add below property in Kafka Producer.

```
props.put(ProducerConfig.ACKS_CONFIG, "all");// 0, 1,  all
props.put(ProducerConfig.RETRIES_CONFIG, 2); // Integer.MAX_VALUE
```

2.4. Update producer record to send messages to test as follows:

```
producer.send(new ProducerRecord<>("test", i, "Msg" + i), new Callback() {
   public void onCompletion(RecordMetadata recordMetadata, Exception e) {
      // executes every time a record is successfully sent or an exception is thrown
      if (e == null) {
         // the record was successfully sent
```

```
                System.out.println ("Received new metadata. \n" +
                        "Topic:" + recordMetadata.topic() + "\n" +
                        "Partition: " + recordMetadata.partition() + "\n" +
                        "Offset: " + recordMetadata.offset() + "\n" +
                        "Timestamp: " + recordMetadata.timestamp());
            } else {
                System.out.println ("Error while producing"+ e);
            }
        }
    });
```

2.5. Now send messages it should work as expected.
2.6. You can start the consumer if in case you want to see the messages consumed
2.7. Stop any 1 broker and retry sending message from broker, you should get an exception
2.8. Now comment the retries property and try sending message again. The default value for retries is MAX of Integer. Hence if you restart the broker before producer closes you should see the messages sent.

3. Synchronous producer

3.1. Default Kafka producer send API is asynchronous and nonblocking. When you call the send API, it merely adds the ProducerRecord into the buffer and returns immediately. Asynchronous and non-blocking send is efficient. However, it also leaves the possibility to lose some messages without even knowing that some of your messages never reached the Kafka broker. If your use case demands, you can easily turn your send method into a synchronous blocking call and wait for the acknowledgment or an exception. Let's create an example to understand the mechanics.

```
RecordMetadata metadata;
for (int i = 1; i <= numEvents; i++) {
   metadata = producer.send(new ProducerRecord<>(topicName, i, "Simple Message-" + i)).get();
   logger.info("Message " + i + " persisted with offset " + metadata.offset()
           + " and timestamp on " + new Timestamp(metadata.timestamp()));
}
```

4. Message Time Stamp

4.1. message.timestamp.type
Define whether the timestamp in the message is message create time or log append time. The value should be either CreateTime or LogAppendTime

Type:	string
Default:	CreateTime
Valid Values:	[CreateTime, LogAppendTime]
Server Default Property:	log.message.timestamp.type
Importance:	medium

4.2. Use --config to set the timestamp type while creating the topic

5. Transactional Producer

5.1. Create topic-1-create.cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
        %KAFKA_HOME%\bin\windows\kafka-topics .bat --create --bootstrap-server localhost:9092 --topic hello-producer-1
--partitions 5 --replication-factor 3 --config min.insync.replicas=2
```

5.2. Create topic-2-create.cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
        %KAFKA_HOME%\bin\windows\kafka-topics.bat --create --bootstrap-server localhost:9092 --topic hello-producer-2 -
-partitions 5 --replication-factor 3 --config min.insync.replicas=2
```

5.3. Create consumer-start.cmd with below lines within win-scripts folder

set KAFKA_HOME=<path to confluent>/confluent-7.5.0
%KAFKA_HOME%\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --from-beginning --whitelist "hello-producer-1|hello-producer-2"

5.4. Start zookeeper and the 3 brokers

5.5. Execute topic-1-create and topic-2-create

5.6. Create Producer for creating transactional producer

```
System.out.println(x:"Creating Kafka Producer...");
Properties props = new Properties();
props.put(ProducerConfig.CLIENT_ID_CONFIG, AppConfigs.applicationID);
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfigs.bootstrapServers);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, IntegerSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, AppConfigs.transaction_id);
KafkaProducer<Integer, String> producer = new KafkaProducer<>(props);
producer.initTransactions();
System.out.println(x:"Start sending messages...");
producer.beginTransaction();
try {
    for (int i = 1; i <= AppConfigs.numEvents; i++) {
        producer.send(new ProducerRecord<>(AppConfigs.topicName1, i, "Simple Message-T1 " + i));
        producer.send(new ProducerRecord<>(AppConfigs.topicName2, i, "Simple Message-T1 " + i));
    }
    producer.commitTransaction();
}catch (Exception e){
    producer.abortTransaction();
    producer.close();
    throw new RuntimeException();
}
System.out.println(x:"Start Second transaction messages...");
producer.beginTransaction();
try {
    for (int i = 1; i <= AppConfigs.numEvents; i++) {
        producer.send(new ProducerRecord<>(AppConfigs.topicName1, i, "Simple Message-T2" + i));
        producer.send(new ProducerRecord<>(AppConfigs.topicName2, i, "Simple Message-T2" + i));
    }
    producer.abortTransaction();
}catch (Exception e){
    producer.abortTransaction();
    throw new RuntimeException();
}finally {
    producer.close();
}
```

5.7. Run the producer.

5.8. Then execute the consumer-start.cmd to check for success messages from transaction 1 and no messages from transaction 2

6. Idempotent producer

6.1. Each producer assigned a unique Producer Id (PID). Producer always includes its PID every time it sends messages to a broker. In addition, each message gets a monotonically increasing sequence number. A separate sequence is maintained for each topic partition that a producer sends messages to. On the broker side, on a per partition basis, it keeps track of the largest PID-Sequence Number combination it has successfully written.

properties.setProperty(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG, "true");

6.2. Note that there are a couple of limitations when using this feature.

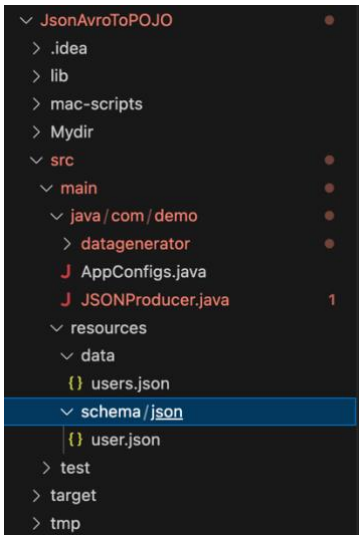
6.2.1. Limitation 1: Acks=All
For one, you can only use acks=all. If you set acks to 0 or 1 then you'll see the error

7. JSON Serialization and Schema Registry

7.1. To understand the schema definition

https://json-schema.org/learn/getting-started-step-by-step

7.2. Project structure looks as follows:



7.3. Either open the project shared or follow below steps to create a new project. If not creating a new project then directly jump to step 7.10

NOTE: DO NOT FORGET TO RUN STEP 7.7 SINCE THERE WILL BE ERRORS IN PROJECT FOR User CLASS

7.4. Create a new Java maven quickstart project JsonAvroProducer
Add below dependencies for json producer in pom.xml file:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.5.1</version>
  </dependency>

  <dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
  </dependency>

  <!--Jackson Databind-->
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.14.2</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/io.confluent/kafka-json-serializer -->
  <dependency>
    <groupId>io.confluent</groupId>
    <artifactId>kafka-json-serializer</artifactId>
    <version>7.4.0</version>
  </dependency>
```

7.5. Add below plugins

```
    <build>
  <plugins>
    <!-- Json Schema to POJO plugin-->
    <plugin>
      <groupId>org.jsonschema2pojo</groupId>
      <artifactId>jsonschema2pojo-maven-plugin</artifactId>
      <version>1.2.1</version>
      <executions>
```

```
        <execution>
          <goals>
            <goal>generate</goal>
          </goals>
          <configuration>
            <sourceDirectory>${project.basedir}/src/main/resources/schema/json/</sourceDirectory>
            <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
            <includeAdditionalProperties>false</includeAdditionalProperties>
            <includeHashcodeAndEquals>false</includeHashcodeAndEquals>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>
 <repositories>
  <repository>
    <id>confluent</id>
    <url>https://packages.confluent.io/maven/</url>
  </repository>
 </repositories>
</project>
```

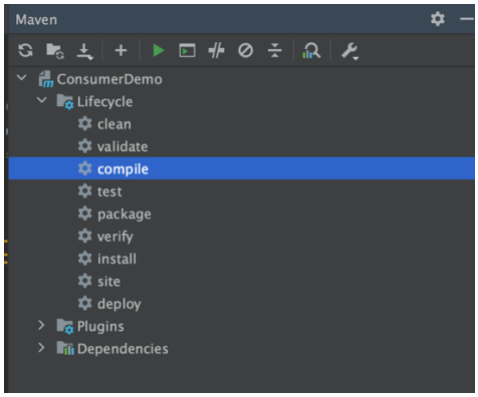
7.6. Create a schema folder under resources and a json folder within schema folder. Carete a file user.json within the json folder as follows:

NOTE :

1. the value for javaType key is the java package name and class name User. Make sure to give your package name. Class user will be created automatically with the plugin configured above
2. The value for sourceDirectory in the plugin above is based on this step. Please make sure to create th folder structure properly

```
{
 "title": "User",
 "javaType": "com.demo.types.User",
 "type": "object",
 "properties": {
  "userId": {
   "description": "The unique identifier for a user",
   "type": "integer"
  },
  "userName": {
   "description": "Name of the user",
   "type": "string"
  },
  "age": {
   "description": "The age of the user",
   "type": "number",
   "exclusiveMinimum": 0
  },
  "notification": {
   "description": "Notifications way for the user",
   "type": "array",
   "items": {
    "type": "string"
   },
   "minItems": 1,
   "uniqueItems": true
  }
 },
 "required": [ "userId", "userName", "age" ]
}
```

7.7. Expand the Maven tab and run the lifecyle compile phase . It will generate respective POJOS from the schemas. Check avro and types folder for respective generated POJOS



7.8. Create data folder within resources folder and a file user.json as follows:

```
[
 {
  "userId": 1,
  "userName": "Shalini",
  "age": 30,
  "notification": ["sms"]
 },
 {
  "userId": 2,
  "userName": "Siya",
  "age": 2
 },
 {
  "userId": 3,
  "userName": "Jack",
  "age": 10,
  "notification": ["email"]
 },
 {
  "userId": 4,
  "userName": "Ron",
  "age": 35
 },
 {
  "userId": 5,
  "userName": "Don",
  "age": 15
 },
 {
 "userId": 6,
 "userName": "Maya",
 "age": 45
 },
 {
  "userId": 7,
  "userName": "Sam",
  "age": 46
 },
 {
  "userId": 8,
  "userName": "Nisha",
  "age": 8
 },
 {
  "userId": 9,
  "userName": "Nitin",
  "age": 23
 },
 {
  "userId": 10,
  "userName": "Asha",
  "age": 43
 },
```

```
 {
  "userId": 11,
  "userName": "John",
  "age": 44
 },
 {
  "userId": 12,
  "userName": "Nivedita",
  "age": 3
 },
 {
  "userId": 13,
  "userName": "Bala",
  "age": 56
 },
 {
  "userId": 14,
  "userName": "Charlie",
  "age": 53
 },
 {
  "userId": 15,
  "userName": "Payal",
  "age": 21
 }
]
```

7.9. Create a package datagenerator and a class UserGenerator as follows:

```
package com.demo.datagenerator;

import com.demo.types.User;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;
import java.util.Random;

public class UserGenerator {
   private static final UserGenerator ourInstance = new UserGenerator();
   private final Random random;

   private User[] users;
   public static UserGenerator getInstance() {
      return ourInstance;
   }

   private UserGenerator() {
      String DATAFILE = "src/main/resources/data/users.json";
      ObjectMapper mapper = new ObjectMapper();
      random = new Random();
      try {
         users = mapper.readValue(new File(DATAFILE), User[].class);
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   public int getUsersLength() {
      return users.length;
   }

   public User getNextUser(int index) {
      return users[index];
   }
}
```

7.10. Go through the resources folder for data and schemas for your understanding

7.11. datagenerator folder has a class that generates user data.

7.12. Do Copy the win-scripts folder

7.13. Create topic-json-create.cmd with below lines within win-scripts folder

set KAFKA_HOME=<path to confluent>/confluent-7.5.0
%KAFKA_HOME%\bin\windows\kafka-topics .bat --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic user-topic

7.14. Create consumer-json-start.cmd with below lines within win-scripts folder

set KAFKA_HOME=<path to confluent>/confluent-7.5.0
%KAFKA_HOME%\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --from-beginning --topic user-topic

7.15. Start zookeeper and the 3 brokers

7.16. Execute topic-create commands

7.17. Update JSONProducer to producer User objects

```
Properties props = new Properties();
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, value:"localhost:9092");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        value:IntegerSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        value:KafkaJsonSerializer.class);
props.put(key:"schema.registry.url", value:"http://127.0.0.1:8081");

Producer<Integer, User> producer = new KafkaProducer<>(props);
UserGenerator generator = UserGenerator.getInstance();

for(int i=0;i< generator.getUsersLength();i++){
    User user = generator.getNextUser(i);
    ProducerRecord<Integer, User> record
            = new ProducerRecord<>(AppConfigs.jsontopicname, user.getUserId(), user);
    System.out.println(producer.send(record).get());
}
producer.close();
```

7.18. Run consumer-json-start to check if data can be consumed

NOTE: Below steps will not work with WINDOWS

8. Schema Registry

8.1. To register the JSON schema with the schema registry follow below steps

8.1.1. Add below dependency

```
<dependency>
    <groupId>io.confluent</groupId>
    <artifactId>kafka-json-schema-serializer</artifactId>
    <version>7.5.0</version>
  </dependency>
```

8.1.2. Modify the value property as follows:
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"io.confluent.kafka.serializers.json.KafkaJsonSchemaSerializer");

8.1.3. Start schema registry with the following command:
$KAFKA_HOME/bin/schema-registry-start $KAFKA_HOME/etc/schema-registry/schema-registry.properties

8.1.4. Now run the JSONproducer and you should see the json schema for user registered as shown in the below screenshot:

```
[2024-04-03 22:01:44,366] INFO Wait to catch up until the offset at 12 (io.confluent.kafka.schemaregistry.storage.KafkaStore:313)
[2024-04-03 22:01:44,367] INFO Reached offset at 12 (io.confluent.kafka.schemaregistry.storage.KafkaStore:315)
[2024-04-03 22:01:44,367] INFO 127.0.0.1 - - [03/Apr/2024:16:31:44 +0000] "POST /subjects/user-topic-value/versions?normalize=false HTTP/1.1" 200 8 "-"
"Java/17.0.8" 108 (io.confluent.rest-utils.requests:62)
```

8.1.5. To view the schema subjects open browser and type in below url:

http://127.0.0.1:8081/schemas
You should see output similar to below

9. Schema registry Query

https://docs.confluent.io/platform/7.6/schema-registry/develop/using.html#schemaregistry-using

9.1. To returns all the subjects registered

http://127.0.0.1:8081/schemas

{
subject: "user-topic-value",
version: 1,
id: 6,
schemaType: "JSON",
schema: "{"$schema":"http://json-schema.org/draft-07/schema#","title":"User","type":"object","additionalProperties":false,"properties":{"userId":{"oneOf":[{"type":"null","title":"Not included"},{"type":"integer"}],"description":"Unique id of the user"},"userName":{"oneOf":[{"type":"null","title":"Not included"},{"type":"string"}],"description":"name of the user"},"age":{"oneOf":[{"type":"null","title":"Not included"},{"type":"integer"}],"description":"Age of the user"},"notification":{"oneOf":[{"type":"null","title":"Not included"},{"type":"array","items":{"type":"string"}}],"description":"If user chooses to recieve notifications"}}}"
}

9.2. To see the name of all the subjects:
http://127.0.0.1:8081/subjects
NOTE: Output may differ based on how many schemas you registered
[
"SNACKS_JSONSCHEMA-value",
"emp-topic-value",
"user-topic-value"
]

9.3. To get the latest version for a particular subject, gives output same as 9.1
http://127.0.0.1:8081/subjects/user-topic-value/versions/latest

9.4. To get the version number
http://127.0.0.1:8081/subjects/user-topic-value/versions

9.5. To get schema for a particular subject with id
http://localhost:8081/schemas/ids/6/schema
http://localhost:8081/schemas/ids/6

9.6. Soft delete
curl -X DELETE http://localhost:8081/subjects/emp-topic-value

9.7. Permanent delete [But before this command you need to soft delete first]
curl -X DELETE http://localhost:8081/subjects/emp-topic-value?permanent=true

9.8. To get the schema config, it provides with compatibility level which can be backward/forward/none
http://localhost:8081/config

9.9. Update the compatibility requirements globally.

```
curl -X PUT -H "Content-Type: application/vnd.schemaregistry.v1+json" \
--data '{"compatibility": "NONE"}' \
http://localhost:8081/config
```

9.10. More examples:
https://docs.confluent.io/platform/7.6/schema-registry/develop/using.html#schemaregistry-using

10. AVRO Producer

10.1. Add below dependency and avro plugin in pom.xml

```
<dependency>
    <groupId>org.apache.avro</groupId>
    <artifactId>avro</artifactId>
    <version>1.11.3</version>
  </dependency>

 <dependency>
    <groupId>io.confluent</groupId>
    <artifactId>kafka-avro-serializer</artifactId>
    <version>6.1.0</version>
  </dependency>

<plugin>
     <groupId>org.apache.avro</groupId>
     <artifactId>avro-maven-plugin</artifactId>
     <version>1.10.2</version>
     <executions>
```

```
        <execution>
          <phase>generate-sources</phase>
          <goals>
            <goal>schema</goal>
          </goals>
          <configuration>
            <sourceDirectory>${project.basedir}/src/main/resources/schema/avro/</sourceDirectory>
            <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
          </configuration>
        </execution>
      </executions>
    </plugin>
```

10.2. Within schema folder, create a folder avro and a file Employee.avsc within avro folder with following contents:

```
{"namespace": "com.demo.avro",
"type": "record", "name": "Employee",
"fields": [
  {"name": "firstName", "type": "string"},
  {"name": "lastName", "type": "string"},
  {"name": "age",  "type": "int"},
  {"name": "phoneNumber",  "type": "string"},
]
}
```

10.3. Run maven compile phase to generate respective POJO

10.4. Create a topic emp-topic

$KAFKA_HOME/bin/kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic emp-topic

10.5. Now Create an AvroProducer as follows:

```
package com.demo;

import com.demo.avro.Employee;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.*;
import org.apache.kafka.common.errors.SerializationException;

import java.io.File;
import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

public class AvroProducer {
   public static void main(String[] args) throws ExecutionException, InterruptedException {

      Properties props = new Properties();
      props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
      props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
            org.apache.kafka.common.serialization.StringSerializer.class);
      props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
            io.confluent.kafka.serializers.KafkaAvroSerializer.class);
      props.put("schema.registry.url", "http://127.0.0.1:8081");
         Employee bob = Employee.newBuilder().setAge(25)
            .setFirstName("Shalini")
            .setLastName("Chaudhary")
            .setPhoneNumber("125-555-1212")
               .setCity("Mumbai")
               .setCountry("India")
            .build();

         ProducerRecord<String, Employee> record = new ProducerRecord<>("emp-topic", null, bob);
         try {
            RecordMetadata data = (RecordMetadata) producer.send(record).get();
            System.out.println(data.partition()+" "+data.topic());
         } catch(SerializationException e) {
            // may need to do something with it
            System.out.println("error");
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
```

```
        // When you're finished producing records, you can flush the producer to ensure it has all been written to Kafka and
        // then close the producer to free its resources.
                finally {
                   System.out.println("DONE");
                   producer.flush();
                   producer.close();
                }
        //      } catch (IOException e) {
        //         e.printStackTrace();
        //      }


           }
        }
```

10.6. Now start the zookeeper, 3 brokers and create topic
10.7. Make sure to start schema registry as follows:
$KAFKA_HOME/bin/schema-registry-start $KAFKA_HOME/etc/schema-registry/schema-registry.properties
10.8. Now run the AvroProducer and you should see the subject registered with the name emp-topic-value
10.9. Schema Registry provides with REST API to query for subjects as discussed in section 9
10.10. Now try modifying the Employee.avsc and add a new property
{"name": "city", "type": "string"}
10.11. Again run maven compile and start the producer, you should get an error for schema compatibility since default config is backward. There are 3 solutions for this:
    10.11.1. Either delete the subject from the schema registry
    10.11.2. Change compatibility to none
    10.11.3. Create city as default null
        {"name": "city", "type": ["null","string"], "default": null}

11. REFERENCES

1. Difference in kafka serialization schemes
   https://simon-aubury.medium.com/kafka-with-avro-vs-kafka-with-protobuf-vs-kafka-with-json-schema-667494cbb2af
2. Schema Evolution
   https://docs.confluent.io/platform/7.6/schema-registry/fundamentals/schema-evolution.html