# Kafka Serialization and Deserialization: JSON & Avro Examples

This is a corrected and updated guide for a Java client that demonstrates how to produce messages using **JSON** and **Avro** serialization. This version uses the latest compatible components to ensure a seamless experience.

### 1. Maven Project Configuration (pom.xml)

This pom.xml includes all the necessary dependencies for a project using Kafka clients, JSON serialization (via Jackson), and Avro serialization with the Confluent Schema Registry client. The versions have been updated to the latest stable releases.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>kafka-serialization-examples</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <kafka.version>4.0.0</kafka.version>
        <jackson.version>2.16.1</jackson.version>
        <confluent.version>8.0.0</confluent.version>
        <avro.version>1.11.3</avro.version>
    </properties>

    <repositories>
        <repository>
            <id>confluent</id>
            <url>https://packages.confluent.io/maven/</url>
        </repository>
    </repositories>
```

```
<dependencies>
    <!-- Kafka Clients -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>${kafka.version}</version>
    </dependency>

    <!-- JSON Serialization with Jackson -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>${jackson.version}</version>
    </dependency>

    <!-- Avro Serialization with Confluent -->
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro</artifactId>
        <version>${avro.version}</version>
    </dependency>

    <!-- SLF4J simple logger -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.36</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <!-- Maven Compiler Plugin -->
```

```
        <plugin>
            <groupId>org.apache.maven.plugins</artifactId>
            <version>3.11.0</version>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
            </configuration>
        </plugin>
        <!-- Maven Assembly Plugin to create a fat JAR -->
        <plugin>
            <groupId>org.apache.maven.plugins</artifactId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>3.6.0</version>
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                    <manifest>
                        <mainClass>com.example.SerializationExamples</mainClass>
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
  </build>
</project>
```

**2. Java Serialization Examples (SerializationExamples.java)**

This class defines a User model and demonstrates a producer that sends messages as JSON strings and another that uses Avro's GenericRecord format, which is registered with a Schema Registry. The BOOTSTRAP_SERVERS and SCHEMA_REGISTRY_URL have been updated to use localhost for a more general example.

```
package com.example;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import io.confluent.kafka.serializers.KafkaAvroSerializer;

import java.io.IOException;
import java.util.Properties;

/**
 * A class demonstrating Kafka producers with JSON and Avro serialization.
 */
public class SerializationExamples {

    private static final String JSON_TOPIC = "json-topic";
    private static final String AVRO_TOPIC = "avro-topic";
    private static final String BOOTSTRAP_SERVERS = "localhost:9092";
    private static final String SCHEMA_REGISTRY_URL = "http://localhost:8081";

    public static void main(String[] args) throws InterruptedException {
        System.out.println("--- Running JSON Producer ---");
        runJsonProducer();
        Thread.sleep(2000);
```

```
        System.out.println("\n--- Running Avro Producer ---");
        runAvroProducer();
        Thread.sleep(2000);
    }

    /**
     * An inner class to represent the data model for our examples.
     */
    public static class User {
        private String name;
        private int age;

        public User() {}

        public User(String name, int age) {
            this.name = name;
            this.age = age;
        }

        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public int getAge() { return age; }
        public void setAge(int age) { this.age = age; }
    }

    /**
     * Creates and runs a producer that serializes a User object to a JSON string.
     */
    private static void runJsonProducer() {
        Properties props = createBaseProducerProperties();
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);

        ObjectMapper objectMapper = new ObjectMapper();
        User user = new User("Alice", 30);

        try {
            String jsonString = objectMapper.writeValueAsString(user);
            ProducerRecord<String, String> record = new ProducerRecord<>(JSON_TOPIC,
"user-key-1", jsonString);
```

```
            producer.send(record, (metadata, exception) -> {
                if (exception == null) {
                    System.out.println("JSON message sent successfully: " + jsonString);
                } else {
                    exception.printStackTrace();
                }
            });
        } catch (IOException e) {
            e.printStackTrace();
        }

        producer.flush();
        producer.close();
    }

    /**
     * Creates and runs a producer that serializes a User object using Avro.
     * This requires the Schema Registry to be running.
     */
    private static void runAvroProducer() {
        Properties props = createBaseProducerProperties();
        // The value serializer for Avro messages
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
KafkaAvroSerializer.class.getName());
        // The URL of the Schema Registry
        props.put("schema.registry.url", SCHEMA_REGISTRY_URL);

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

        // Define the Avro schema inline
        String userSchemaString =
"{\"type\":\"record\",\"name\":\"User\",\"fields\":[{\"name\":\"name\",\"type\":\"string\"},{
\"name\":\"age\",\"type\":\"int\"}]}";
        Schema schema = new Schema.Parser().parse(userSchemaString);

        // Create a GenericRecord from the schema
        GenericRecord userRecord = new GenericData.Record(schema);
        userRecord.put("name", "Bob");
        userRecord.put("age", 45);
```

```
        // Send the Avro message
        ProducerRecord<String, GenericRecord> record = new
ProducerRecord<>(AVRO_TOPIC, "user-key-2", userRecord);
        producer.send(record, (metadata, exception) -> {
            if (exception == null) {
                System.out.println("Avro message sent successfully: " + userRecord);
            } else {
                System.err.println("Error sending Avro message: " +
exception.getMessage());
            }
        });

        producer.flush();
        producer.close();
    }

    /**
     * Common producer properties.
     */
    private static Properties createBaseProducerProperties() {
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
BOOTSTRAP_SERVERS);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
        // For the JSON producer, the value serializer is a StringSerializer
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
        return props;
    }
}
```

**3. Step-by-Step Execution Guide**

Follow these steps to set up the project, run the necessary services, and execute the examples.

**Step 1: Project Setup and Compilation**

1. Create a project directory structure: mkdir -p kafka-serialization-examples/src/main/java/com/example.
2. Copy the pom.xml and SerializationExamples.java content into the correct locations.
3. Navigate to the project root and build the fat JAR: mvn clean package.

**Step 2: Install Confluent Platform**

This is an essential step for the Avro example. The example assumes you have a user named kafka and your Kafka is installed in /usr/local/kafka.

```
# Define versions for Confluent and Kafka
CONFLUENT_VERSION_DL="8.0.0"
CONFLUENT_VERSION="8.0"
KAFKA_VERSION="4.0.0"

# Download the Confluent Platform tarball
wget
"https://packages.confluent.io/archive/${CONFLUENT_VERSION}/confluent-community
-${CONFLUENT_VERSION_DL}.tar.gz" -O /tmp/confluent.tar.gz

# Extract to a new directory
sudo mkdir -p /usr/local/confluent
sudo tar -xzf /tmp/confluent.tar.gz -C /usr/local/confluent --strip-components 1

# Change ownership to the kafka user
sudo chown -R kafka:kafka /usr/local/confluent
```

*Note: The community download filename for Confluent Platform 8.0.0 is slightly different, removing the Kafka version suffix.*

**Step 3: Start Confluent Schema Registry**

With the Confluent Platform installed, you can now start the Schema Registry.

Update

sudo vim /usr/local/confluent/etc/schema-registry/schema-registry.properties

#listeners=http://0.0.0.0:8081

listeners=http://kafka-199:8081

# Use this setting to specify the bootstrap servers for your Kafka cluster and it

# will be used both for selecting the leader schema registry instance and for storing the data for

# registered schemas.

#kafkastore.bootstrap.servers=PLAINTEXT://localhost:9092

kafkastore.bootstrap.servers=PLAINTEXT://kafka-199:9092,kafka-200:9092,kafka-201:9092

# The name of the topic to store schemas in

kafkastore.topic=_schemas

# If true, API requests that fail will include extra debugging information, including stack traces

debug=false

metadata.encoder.secret=REPLACE_ME_WITH_HIGH_ENTROPY_STRING

resource.extension.class=io.confluent.dekregistry.DekRegistryResourceExtension

sudo -u kafka /usr/local/confluent/bin/schema-registry-start /usr/local/confluent/etc/schema-registry/schema-registry.properties

**Step 4: Create the Topics**

Create the two topics that the producers will write to.

```
# JSON Topic
sudo -u kafka /usr/local/kafka/bin/kafka-topics.sh --create \
  --topic json-topic \
```

```
  --partitions 3 \
  --replication-factor 3 \
  --bootstrap-server localhost:9092

# Avro Topic
sudo -u kafka /usr/local/kafka/bin/kafka-topics.sh --create \
  --topic avro-topic \
  --partitions 3 \
  --replication-factor 3 \
  --bootstrap-server localhost:9092
```

Step 5: Run the Java Producer Application

Execute the JAR file to run both producer examples sequentially.

```
java -jar
target/kafka-serialization-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Step 6: Verify Messages with a Consumer

Open two separate terminals to verify the messages produced to each topic.

Consumer for JSON Messages:
Since the JSON message is just a string, you can use the default consumer.

```
sudo -u kafka /usr/local/kafka/bin/kafka-console-consumer.sh \
  --topic json-topic \
  --bootstrap-server kafka-199:9092 \
  --from-beginning \
  --property print.key=true
```

You should see: user-key-1 {"name":"Alice","age":30}

Consumer for Avro Messages:
To read Avro messages, you must use the Confluent-provided consumer script and specify the Schema Registry URL. The kafka-avro-console-consumer is the correct tool to use for this purpose.

```
sudo -u kafka /usr/local/confluent/bin/kafka-avro-console-consumer \
  --topic avro-topic \
  --bootstrap-server kafka-199:9092 \
  --from-beginning \
  --property print.key=true \
  --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
  --property schema.registry.url=http://kafka-199:8081
```

You should see: user-key-2 {"name": "Bob", "age": 45}