Last Updated: 20 March March 2023

JDK : Jdk : 1.11
Apaceh kafka : kafka_2.13-3.2.1.tar

https://developer.confluent.io/learn-kafka/kafka-streams/get-started/

Note: Use Full path with .sh extension for Kafka broker else the commands are for confluent kafka. Replace the hostname accordingly.

Required confluent kafka for CLI

1) Data Generation

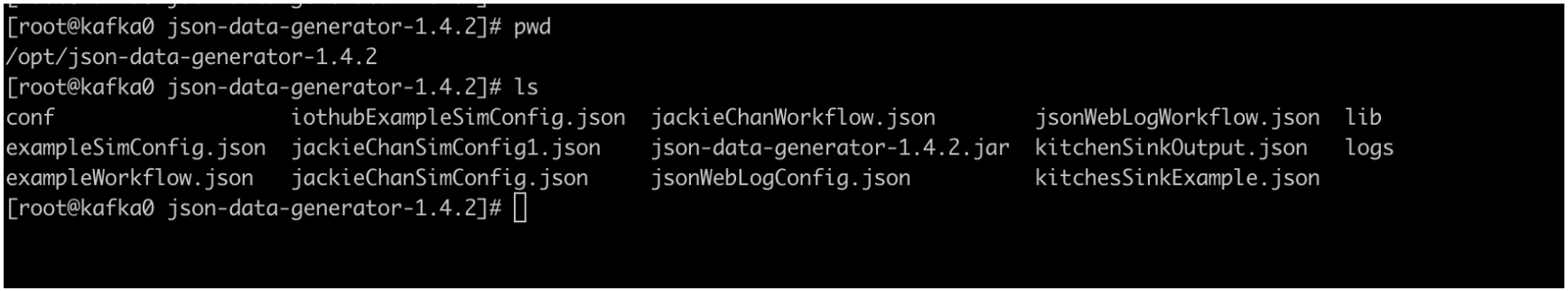
This tool generate data in json format.
If you have not downloaded the tool already, go ahead and download the most recent release of the json-data-generator.
https://github.com/acesinc/json-data-generator/releases

1. Unpack the file you downloaded to a directory.

(#tar -xvf json-data-generator-1.4.2-bin.tar -C /opt)

2. Copy all configs into the main directory directory. Execute the following commands.

```
# cd /opt/json-data-generator-1.4.2
# cp conf/* .
```



As shown above all json configs file should be in the main folder.

3. Update the json config to refer our broker. (#vi jackieChanSimConfig.json)

```
{
    "workflows": [{
            "workflowName": "jackieChan",
            "workflowFilename": "jackieChanWorkflow.json"
       }],
    "producers": [{
        "type": "kafka",
        "broker.server": "kafka0",
        "broker.port": 9092,
        "topic": "jackieChanCommand",
        "flatten": false,
        "sync": false

    }]
}
~
```

4. Then run the generator like below:

java -jar json-data-generator-1.4.2.jar jackieChanSimConfig.json

You will see logging in your console showing the events as they are being generated. The jackieChanSimConfig.json generates events like these:

{"timestamp":"2015-05-20T22:21:18.036Z","style":"WUSHU","action":"BLOCK","weapon":"CHAIR","target":"BODY","strength":4.7912}

{"timestamp":"2015-05-20T22:21:19.247Z","style":"DRUNKEN_BOXING","action":"PUNCH","weapon":"BROAD_SWORD","target":"ARMS","strength":3.0248}
{"timestamp":"2015-05-20T22:21:20.947Z","style":"DRUNKEN_BOXING","action":"BLOCK","weapon":"ROPE","target":"HEAD","strength":6.7571}
{"timestamp":"2015-05-20T22:21:22.715Z","style":"WUSHU","action":"KICK","weapon":"BROAD_SWORD","target":"ARMS","strength":9.2062}
{"timestamp":"2015-05-20T22:21:23.852Z","style":"KUNG_FU","action":"PUNCH","weapon":"BROAD_SWORD","target":"HEAD","strength":4.6202}
{"timestamp":"2015-05-20T22:21:25.195Z","style":"KUNG_FU","action":"JUMP","weapon":"ROPE","target":"ARMS","strength":7.5303}
{"timestamp":"2015-05-20T22:21:26.492Z","style":"DRUNKEN_BOXING","action":"PUNCH","weapon":"STAFF","target":"HEAD","strength":1.1247}
{"timestamp":"2015-05-20T22:21:28.042Z","style":"WUSHU","action":"BLOCK","weapon":"STAFF","target":"ARMS","strength":5.5976}
{"timestamp":"2015-05-20T22:21:29.422Z","style":"KUNG_FU","action":"BLOCK","weapon":"ROPE","target":"ARMS","strength":2.152}

{"timestamp":"2015-05-20T22:21:30.782Z","style":"DRUNKEN_BOXING","action":"BLOCK","weapon":"STAFF","target":"ARMS","strength":6.2686}
{"timestamp":"2015-05-20T22:21:32.128Z","style":"KUNG_FU","action":"KICK","weapon":"BROAD_SWORD","target":"BODY","strength":2.3534}

-------- Data Generation Ends Here ----------

2) Using kafka Stream – 90 Minutes

In this tutorial, we will produce messages continuously using a data tool and inserted these messages into a topic, ***fight*****.**

Each message represents a martial art style as shown below.

{"timestamp":"2015-05-20T22:21:18.036Z","style":"WUSHU","action":"BLOCK","weapon":"CHAIR","target":"BODY","strength":4.7912}

Messages will be filtered by its style "KUNG_FU" and only message with style = "KUNG_FU" gets inserted into a separate topic, ***kungfu_ja***

Client Application will be implemented using Producer/Consumer(P&C) API and Stream API.

a) Producer/Consumer API

In the P&C API code, you create a producer to push the message to the topic and consumer, that subscribes to the single topic `fight`. Then you `poll()` your records, and the `ConsumerRecords` collection is returned. You loop over the records and pull out values, filtering out the ones that are "KUNG_FU". Then you take the " KUNG_FU" records, create a new `ProducerRecord` for each one, and write those out to the `KUNG_FU_ja` topic.

Sample Data Message.

{"timestamp":"2023-03-20T20:14:08.287Z","style":"KUNG_FU","action":"PUNCH","weapon":"CHAIR","target":"HEAD","strength":8.5128}

First Define the message bean – **MartialArt.java**

```
package com.tos.kafka.stream;

import java.sql.Timestamp;

public class MartialArt {

    private Timestamp timestamp;
    private String style;
    private String action;
    private String weapon;
    private String target;
    private Double strength;

    public MartialArt() {
        super();
```

```
        // TODO Auto-generated constructor stub
    }
    public MartialArt(Timestamp timestamp, String style, String
action, String weapon, String target, Double strength) {
        super();
        this.timestamp = timestamp;
        this.style = style;
        this.action = action;
        this.weapon = weapon;
        this.target = target;
        this.strength = strength;
    }
    public Timestamp getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }
    public String getStyle() {
        return style;
    }
    public void setStyle(String style) {
        this.style = style;
    }
    public String getAction() {
```

```
        return action;
    }
    public void setAction(String action) {
        this.action = action;
    }
    public String getWeapon() {
        return weapon;
    }
    public void setWeapon(String weapon) {
        this.weapon = weapon;
    }
    public String getTarget() {
        return target;
    }
    public void setTarget(String target) {
        this.target = target;
    }
    public Double getStrength() {
        return strength;
    }
    public void setStrength(Double strength) {
        this.strength = strength;
    }
```

```
}
```

Since our message is a json, let us create a Serializer and Deserializer.

Serializer Class.

```
package com.tos.kafka.stream;

import java.nio.charset.StandardCharsets;
import org.apache.kafka.common.serialization.Serializer;
import com.google.gson.Gson;

public class MartialArtSerializer implements Serializer<MartialArt>
{
    private Gson gson = new Gson();
    @Override
    public byte[] serialize(String topic, MartialArt data) {
        // TODO Auto-generated method stub

        if (data == null)
            return null;
        return gson.toJson(data).getBytes(StandardCharsets.UTF_8);
```

```
    }

}
```

Deserializer Class

```
package com.tos.kafka.stream;

import java.nio.charset.StandardCharsets;
import org.apache.kafka.common.serialization.Deserializer;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;



public class MartialArtDeserializer implements
Deserializer<MartialArt> {
    private Gson gson = new GsonBuilder().create();
    @Override
    public MartialArt deserialize(String topic, byte[] data) {
        // TODO Auto-generated method stub
        if (data == null)
            return null;
```

```
        return gson.fromJson(new String(data,
StandardCharsets.UTF_8), MartialArt.class);

    }

}
```

Define the main class. - `UnderstandingStream.java`

```
package com.tos.kafka.stream;

public class UnderstandingStream {


}
```

Import the following packages.

```
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
```

```
import java.util.concurrent.CountDownLatch;

import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
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.Serdes;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
```

Define the connection parameter

```
private static final String TOPIC = "fight";
private static final String OP_JA_TOPIC = "kungfu_ja";
private static final String OP_KSA_TOPIC = "kungfu_sa";
private static final String BOOOTSTRAP_SERVERS = "kafka0:9092";
```

Let us add a method that will consume messages and filter the message by it style then push it into a new topic using Producer/ Consumer API.

```
static void processUsingAPI() {

        Properties properties = new Properties();

properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
BOOOTSTRAP_SERVERS);
        properties.setProperty(ProducerConfig.CLIENT_ID_CONFIG,
"MartialArtProducer");
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "ME16");
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");

properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());

properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG
, MartialArtSerializer.class.getName());
```

```
properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG
, StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONF
IG, MartialArtDeserializer.class.getName());
        KafkaProducer<String,MartialArt> producer =  new
KafkaProducer<>(properties);

       // Create the consumer using props.
        final Consumer<String, MartialArt> consumer = new
KafkaConsumer<>(properties);

        // Subscribe to the topic.
        consumer.subscribe(Collections.singletonList(TOPIC));
            while (true) {
            final ConsumerRecords<String, MartialArt>
consumerRecords = consumer.poll(Duration.ofMillis(1000));
            if(consumerRecords!=null)
                consumerRecords.forEach(record -> {
                System.out.printf("Consumer Record:(%d, %s, %d,
%d)\n", record.key(), record.value().getWeapon(),
                        record.partition(), record.offset());
                MartialArt ma = record.value();
```

```
                if (ma.getStyle() !=null &&
ma.getStyle().equals("KUNG_FU")) {
                    ProducerRecord<String, MartialArt>
producerRecord =
                            new ProducerRecord<>(OP_JA_TOPIC,
record.key(), ma);
                    producer.send(producerRecord, (metadata,
exception)-> {
                        System.out.println(" After Insert
(Style:Action:offset): " + ma.getStyle() + " : " +  ma.getAction()
+ " : "+  metadata.offset());
                    }) ;
                }
            });
        consumer.commitAsync();

    }
   }
```

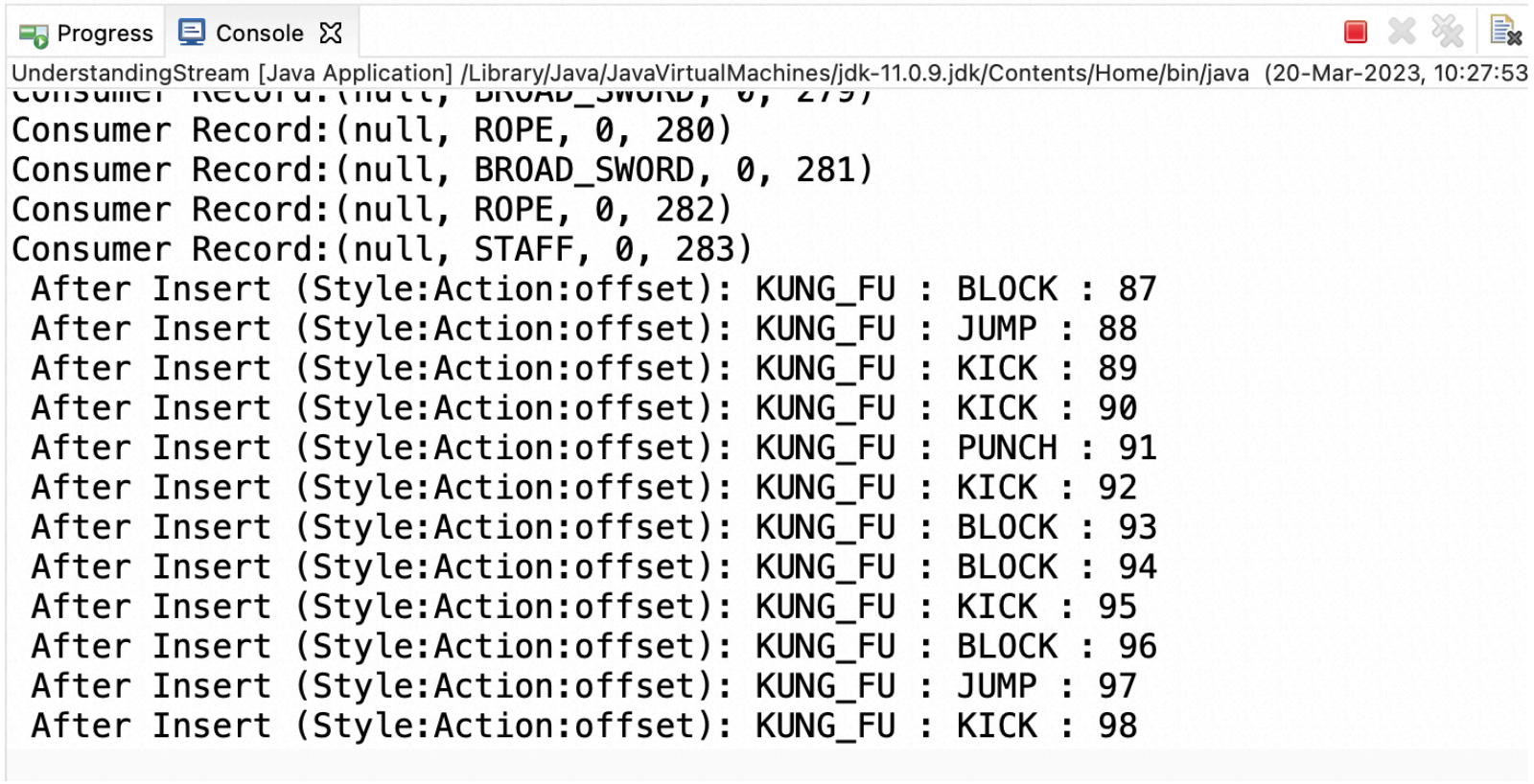
Finally add the main method and invoke the above method.

```
public static void main(String[] args) {
```

```
        processUsingAPI();

}
```

Generate Data. Refer the prerequisite section.

Execute it - JAVA API.(`UnderstandingStream.java`)

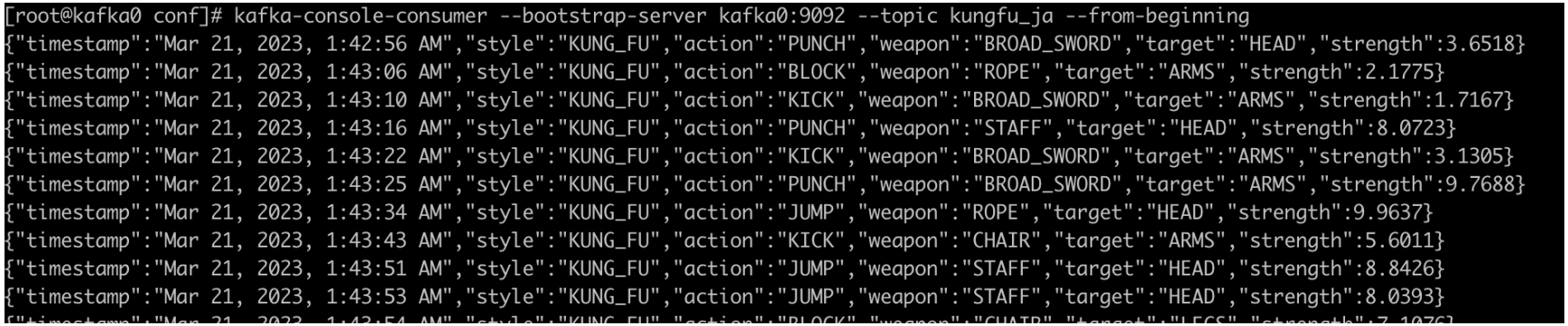


Note:

1) Result prepend with “Consumer Record :” -> Messages consume from the source topic before filter e.x first entry “ROPE” shows the style of Martial Art.
2) Line that begins with “After Insert ” signify the filtered message after being inserted into the output topic.

You can verify using the consumer console too. All style should be “Kung_FU”.

#kafka-console-consumer --bootstrap-server kafka0:9092 --topic kungfu_ja --from-beginning



B) Using KafkaStream API.

Let us implement the same using Stream API.

Stream API.

You instantiate a `StreamsBuilder`, then you create a stream based off of a topic and give it a SerDes. Then you filter the records and write back out to the `Kungfu_sa` topic.

With Kafka Streams, you state *what* you want to do, rather than *how* to do it. This is far more declarative than the vanilla Kafka example.

You need to define a Deserializer for Stream API.

```
package com.tos.kafka.stream;

import org.apache.kafka.common.serialization.Deserializer;
import org.apache.kafka.common.serialization.Serde;
import org.apache.kafka.common.serialization.Serializer;

public class MASerdes implements Serde<MartialArt>{

    @Override
    public Serializer<MartialArt> serializer() {
        // TODO Auto-generated method stub
```

```
        return new MartialArtSerializer();
    }

    @Override
    public Deserializer<MartialArt> deserializer() {
        // TODO Auto-generated method stub
        return new MartialArtDeserializer();
    }

}
```

Then Add the following method in `UnderstandingStream.java`.

```
static void processingUsingStreamAPI(){

        final Properties streamsConfiguration = new Properties();

    streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG,
"MES");
        streamsConfiguration.put(StreamsConfig.CLIENT_ID_CONFIG,
"MES");

    streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG
, BOOOTSTRAP_SERVERS);
```

```
    streamsConfiguration.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_
CONFIG, Serdes.String().getClass());

    streamsConfiguration.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLAS
S_CONFIG, MASerdes.class);

    streamsConfiguration.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RES
ET_CONFIG, "earliest");
        final StreamsBuilder builder = new StreamsBuilder();

        KStream<String, MartialArt> maStream =
builder.stream(TOPIC);
        KStream<String, MartialArt> ma = maStream
                .filter((key, ma1) ->
ma1.getStyle().equals("KUNG_FU"))
                .peek((k, v) -> System.out.println("Key:  " + k + "
, Martial Style : " + v.getStyle()));

        ma.to(OP_KSA_TOPIC);

        final KafkaStreams streams = new
KafkaStreams(builder.build(), streamsConfiguration);

        final CountDownLatch latch = new CountDownLatch(1);
```

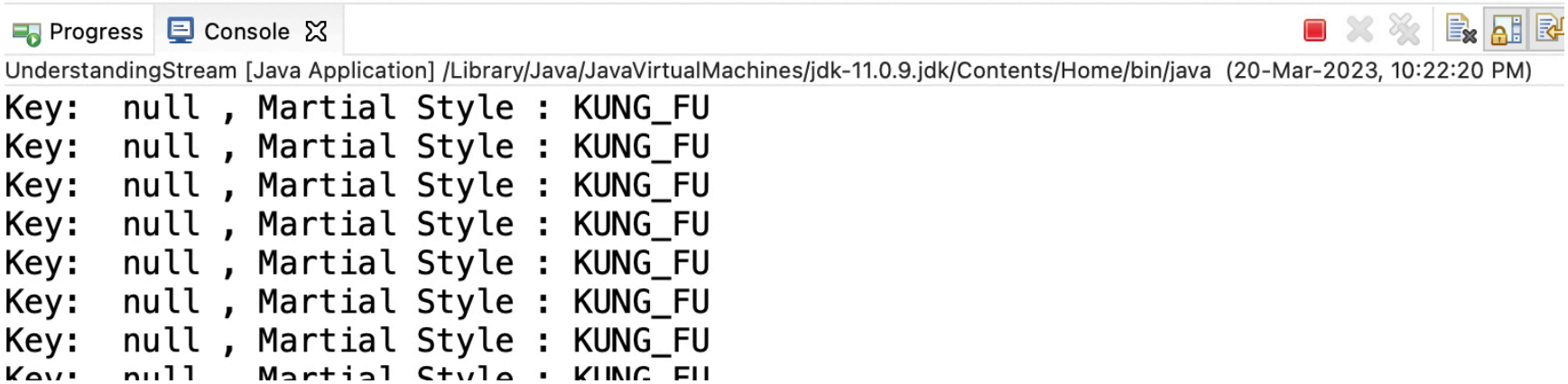
```
        // Attach shutdown handler to catch Control-C.
        Runtime.getRuntime().addShutdownHook(new Thread("streams-
shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
        System.out.println("Done");
    }
```

Subsequently, invoke the above from the main method, comment the earlier invocation.

```
public static void main(String[] args) {
```

```
        //processUsingAPI();
        processingUsingStreamAPI();

    }
```

Execute the program.



It should only shows the filtered message which is of the type “KUNG_FU”.

You can verify it using the consumer console as shown below.

#kafka-console-consumer --bootstrap-server kafka0:9092 --topic kungfu_sa --from-beginning

```
[root@kafka0 conf]# kafka-console-consumer --bootstrap-server kafka0:9092 --topic kungfu_sa --from-beginning
{"timestamp":"Mar 21, 2023, 1:42:56 AM","style":"KUNG_FU","action":"PUNCH","weapon":"BROAD_SWORD","target":"HEAD","strength":3.6518}
{"timestamp":"Mar 21, 2023, 1:43:06 AM","style":"KUNG_FU","action":"BLOCK","weapon":"ROPE","target":"ARMS","strength":2.1775}
{"timestamp":"Mar 21, 2023, 1:43:10 AM","style":"KUNG_FU","action":"KICK","weapon":"BROAD_SWORD","target":"ARMS","strength":1.7167}
{"timestamp":"Mar 21, 2023, 1:43:16 AM","style":"KUNG_FU","action":"PUNCH","weapon":"STAFF","target":"HEAD","strength":8.0723}
{"timestamp":"Mar 21, 2023, 1:43:22 AM","style":"KUNG_FU","action":"KICK","weapon":"BROAD_SWORD","target":"ARMS","strength":3.1305}
{"timestamp":"Mar 21, 2023, 1:43:25 AM","style":"KUNG_FU","action":"PUNCH","weapon":"BROAD_SWORD","target":"ARMS","strength":9.7688}
{"timestamp":"Mar 21, 2023, 1:43:34 AM","style":"KUNG_FU","action":"JUMP","weapon":"ROPE","target":"HEAD","strength":9.9637}
{"timestamp":"Mar 21, 2023, 1:43:43 AM","style":"KUNG_FU","action":"KICK","weapon":"CHAIR","target":"ARMS","strength":5.6011}
{"timestamp":"Mar 21, 2023, 1:43:51 AM","style":"KUNG_FU","action":"JUMP","weapon":"STAFF","target":"HEAD","strength":8.8426}
```

Verify the logic, With Kafka Streams, you state what you want to do, rather than how to do it. This is far more declarative than the vanilla Kafka example.

Summary of Prod/Consumer API.

```
// Subscribe to the topic.
consumer.subscribe(Collections.singletonList(TOPIC));
    while (true) {
    final ConsumerRecords<String, MartialArt> consumerRecords = consumer.poll(Duration.ofMillis(1000));
    if(consumerRecords!=null)
        consumerRecords.forEach(record -> {
        System.out.printf("Consumer Record:(%d, %s, %d, %d)\n", record.key(), record.value().getWeapon(),
                record.partition(), record.offset());
        MartialArt ma = record.value();
         if (ma.getStyle() !=null && ma.getStyle().equals("KUNG_FU")) {
             ProducerRecord<String, MartialArt> producerRecord =
                     new ProducerRecord<>(OP_JA_TOPIC, record.key(), ma);
             producer.send(producerRecord, (metadata, exception)-> {
                 System.out.println(" After Insert (Style:Action:offset): " + ma.getStyle() + " : " +  ma.getAc
             }) ;
         }
    });
```

Summary of Stream API

```
KStream<String, MartialArt> maStream = builder.stream(TOPIC);
KStream<String, MartialArt> ma = maStream
        .filter((key, ma1) -> ma1.getStyle().equals("KUNG_FU"))
        .peek((k, v) -> System.out.println("Key:  " + k + " , Martial Style : " + v.getStyle()));

ma.to(OP_KSA_TOPIC);
```

Stream API is more brevity compare to Proc/cons API.

----------Lab Ends Here ----------

3) Basic Stateless Stream Transformation – 60 Minutes

Scenario:
Fetch fight records from input topic: **fight**
Split the stream into two : One with KUNGFU and Others style.(**split)**
Transform the Kungfu style by doubling the strength and others by 1.5 times.(m**ap**)
And merge the above two streams (**merge)**
Apply filter, allow only the strength greater than 6.(**filter)**
Store the transform message into a topic - **mighty-ma**.

Include a free form diagram(DFD)

Create a java class. `BasicStream.java` and add the following packages.

```
package com.tos.kafka.stream;

import java.util.Map;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.KeyValue;
```

```
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.Branched;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KeyValueMapper;
import org.apache.kafka.streams.kstream.Named;

public class BasicStream {

}
```

Define the following variables.

```
private static final String TOPIC = "fight";
private static final String OP_KSA_TOPIC = "mighty-ma";
private static final String BOOOTSTRAP_SERVERS = "kafka0:9092";
```

Add a method in the above class. All logic of transformation will be written in the following method only.

```
static void processingKungfuStream() {

}
```

Define the required properties for configuring the stream object. Here we are using the existing Serializer and Deserializer of Martial Art. Verify it.

```
    final Properties streamsConfiguration = new Properties();

    streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG,
"MES");
        streamsConfiguration.put(StreamsConfig.CLIENT_ID_CONFIG,
"MES1");

    streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG
, BOOOTSTRAP_SERVERS);

    streamsConfiguration.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_
CONFIG, Serdes.String().getClass());
```

```
    streamsConfiguration.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLAS
S_CONFIG, MASerdes.class);

    streamsConfiguration.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RES
ET_CONFIG, "earliest");
```

Next create an instance of stream using the above configuration. This stream object will connect to the input topic , **fight**.

```
final StreamsBuilder builder = new StreamsBuilder();
KStream<String, MartialArt> maStream = builder.stream(TOPIC);
```

Split the above stream into two branches,

1- Kungfu style martial art
2- Other style martial art

```
/*
        * Split the stream into two : One with KUNGFU and Others
style.
        * Transform the Kungfu style by doubling the strength and
others by  1.5 times.
        * And merge the above two streams
        * Apply filter, allow only the strength greater than 6.
```

```
         * Store the transform message into a topic - mighty-ma.
         *
         */
Map<String, KStream<String, MartialArt>> bMA =
                maStream.split(Named.as("Branch-"))
                        .branch((key, value) ->
value.getStyle().equals("KUNG_FU"),  /* first predicate  */
                                Branched.as("K"))
                        .defaultBranch(Branched.as("C"));
        KStream<String, MartialArt> kungfuStream = bMA.get("Branch-
K");
        KStream<String, MartialArt> oStream = bMA.get("Branch-C");
```

Now at this point, the stream is branches into two, kungfuStream , oStream respectively.

Let us transform the Kungfu stream by increasing the strength by 2 times.

```
/*
         *  Transform the Kungfu stream by multiplying the strength
2 times
         */
        kungfuStream.map(new KeyValueMapper<String,MartialArt,
KeyValue<String,MartialArt>>() {
            @Override
```

```
            public KeyValue<String, MartialArt> apply(String key,
MartialArt ma) {
                    ma.setStrength(ma.getStrength()*2);
                     return new KeyValue<String, MartialArt>(key,ma);
            }

        });
```

Next increase the strength of other style by 1.5 times. Here we will be using map transformation.

```
/*
         *  Transform the Other style fight stream by multiplying
the strength 1.5 times
         */
        oStream.map(new KeyValueMapper<String,MartialArt,
KeyValue<String,MartialArt>>() {
            @Override
            public KeyValue<String, MartialArt> apply(String key,
MartialArt ma) {
                    ma.setStrength(ma.getStrength()*1.5);
                     return new KeyValue<String, MartialArt>(key,ma);
            }

        });
```

Look how we use KeyValueMapper interface to transform the stream.

Merge both the stream.

```
// Merge the above two streams.
        KStream<String, MartialArt> tMA =
kungfuStream.merge(oStream);
```

Then let us apply **filter** function to extract message that strength > 8

```
// Extract Fight with strength greater than the strength 8 only.
        KStream<String, MartialArt> mf = tMA.filter((key, ma1) ->
ma1.getStrength() > 8)
                .peek((k, v) -> System.out.println("Strength:  " +
v.getStrength()
                        + " , Martial Style : " + v.getStyle()));
```

```
mf.to(OP_KSA_TOPIC);
```

And then save to the above mention topic.(mighty-ma)

Finally invoke the above define topology and start the stream.

```
final KafkaStreams streams = new KafkaStreams(builder.build(),
streamsConfiguration);

        final CountDownLatch latch = new CountDownLatch(1);

        // Attach shutdown handler to catch Control-C.
        Runtime.getRuntime().addShutdownHook(new Thread("streams-
shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
```

```
            System.exit(1);
        }
        System.exit(0);
        System.out.println("Done");
```

Here, we complete the implementation of the topology. Let us invoke this method from the **main** method.

```
public static void main(String[] args) {
      processingKungfuStream();
}
```

Generate some messages.

Open a terminal and run the data generator tool.

```
2023-03-22 09:34:34,600 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:34.599Z","style"
:"KUNG_FU","action":"KICK","weapon":"CHAIR","target":"LEGS","strength":8.4433} ]
2023-03-22 09:34:34,931 TRACE n.a.d.j.g.EventGenerator [Thread-0] Generator( jackieChan ) generated 1.0 events/sec
2023-03-22 09:34:35,874 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:35.873Z","style"
:"WUSHU","action":"PUNCH","weapon":"ROPE","target":"BODY","strength":4.673} ]
2023-03-22 09:34:36,194 TRACE n.a.d.j.g.EventGenerator [Thread-0] Generator( jackieChan ) generated 1.0 events/sec
2023-03-22 09:34:37,393 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:37.392Z","style"
:"DRUNKEN_BOXING","action":"KICK","weapon":"CHAIR","target":"ARMS","strength":6.9987} ]
2023-03-22 09:34:37,761 TRACE n.a.d.j.g.EventGenerator [Thread-0] Generator( jackieChan ) generated 1.0 events/sec
2023-03-22 09:34:38,640 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:38.640Z","style"
:"WUSHU","action":"JUMP","weapon":"CHAIR","target":"LEGS","strength":6.9492} ]
2023-03-22 09:34:38,946 TRACE n.a.d.j.g.EventGenerator [Thread-0] Generator( jackieChan ) generated 1.0 events/sec
2023-03-22 09:34:40,142 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:40.141Z","style"
:"KUNG_FU","action":"KICK","weapon":"ROPE","target":"ARMS","strength":1.8166} ]
2023-03-22 09:34:40,538 TRACE n.a.d.j.g.EventGenerator [Thread-0] Generator( jackieChan ) generated 1.0 events/sec
2023-03-22 09:34:41,790 DEBUG n.a.d.j.g.l.KafkaLogger [Thread-0] Sending event to Kafka: [ {"timestamp":"2023-03-22T09:34:41.790Z","style"
:"DRUNKEN_BOXING","action":"JUMP","weapon":"ROPE","target":"BODY","strength":8.2626} ]
```

Execute the main program.

```
19 public class BasicStream {
20
21     private static final String TOPIC = "fight";
22     private static final String OP_KSA_TOPIC = "mighty-ma";
23     private static final String BOOOTSTRAP_SERVERS = "kafka0:9092";
24
25     public static void main(String[] args) {
26       processingKungfuStream();
27     }
28
29     static void processingKungfuStream() {
30
31         final Properties streamsConfiguration = new Properties();
32         streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "MES");
```

Progress | Console

```
<terminated> BasicStream [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.9.jdk/Contents/Home/bin/java (22-Mar-2023, 9:32:36 AM – 9:34:40 AM)
Strength:  12.50325 , Martial Style : DRUNKEN_BOXING
Strength:  8.020050000000001 , Martial Style : WUSHU
Strength:  11.6355 , Martial Style : DRUNKEN_BOXING
Strength:  12.25635 , Martial Style : WUSHU
Strength:  18.7788 , Martial Style : KUNG_FU
Strength:  9.549150000000001 , Martial Style : DRUNKEN_BOXING
Strength:  13.78815 , Martial Style : DRUNKEN_BOXING
Strength:  9.487 , Martial Style : KUNG_FU
```

As shown above, All strength should be greater than 8.0

Verify the messages in the Output topic.

#kafka-console-consumer --bootstrap-server kafka0:9092 --topic mighty-ma

```
[root@kafka0 json-data-generator-1.4.2]# kafka-console-consumer --bootstrap-server kafka0:9092 --topic mighty-ma --from-beginning
{"timestamp":"Mar 22, 2023, 3:01:04 PM","style":"KUNG_FU","action":"JUMP","weapon":"ROPE","target":"LEGS","strength":9.6088}
{"timestamp":"Mar 22, 2023, 3:01:07 PM","style":"DRUNKEN_BOXING","action":"JUMP","weapon":"CHAIR","target":"HEAD","strength":8.9199}
{"timestamp":"Mar 22, 2023, 3:01:09 PM","style":"KUNG_FU","action":"BLOCK","weapon":"CHAIR","target":"BODY","strength":16.2402}
{"timestamp":"Mar 22, 2023, 3:01:12 PM","style":"DRUNKEN_BOXING","action":"KICK","weapon":"STAFF","target":"BODY","strength":14.1279}
{"timestamp":"Mar 22, 2023, 3:01:15 PM","style":"KUNG_FU","action":"BLOCK","weapon":"BROAD_SWORD","target":"LEGS","strength":8.8498}
{"timestamp":"Mar 22, 2023, 3:01:18 PM","style":"DRUNKEN_BOXING","action":"JUMP","weapon":"STAFF","target":"HEAD","strength":9.85709999999
9999}
{"timestamp":"Mar 22, 2023, 3:01:21 PM","style":"DRUNKEN_BOXING","action":"PUNCH","weapon":"ROPE","target":"ARMS","strength":13.65345}
{"timestamp":"Mar 22, 2023, 3:01:24 PM","style":"KUNG_FU","action":"KICK","weapon":"ROPE","target":"LEGS","strength":9.9026}
{"timestamp":"Mar 22, 2023, 3:01:27 PM","style":"KUNG_FU","action":"JUMP","weapon":"ROPE","target":"LEGS","strength":16.4106}
{"timestamp":"Mar 22, 2023, 3:01:30 PM","style":"DRUNKEN_BOXING","action":"KICK","weapon":"STAFF","target":"LEGS","strength":9.3108}
{"timestamp":"Mar 22, 2023, 3:01:32 PM","style":"KUNG_FU","action":"PUNCH","weapon":"CHAIR","target":"HEAD","strength":17.95}
{"timestamp":"Mar 22, 2023, 3:01:35 PM","style":"DRUNKEN_BOXING","action":"KICK","weapon":"CHAIR","target":"HEAD","strength":9.6147}
```

---------- Lab Ends Here ----------

4)Stream Using – FMV Stateless – 45 Minutes (Demo Only)

In this lab, following actions will be performed.

- Kstream creation from a Topic
- Perform transformation using flatMapValues function
- Perform aggregation method
- Conversion from stream to KTable.

Logic:

Stream from the input topic, streams-plaintext-input is created and perform some transformation to it and the output is written to the topic, streams-wordcount-output.

Transformation logic:

Parse the text inserted in the input topic and count the occurrence of each word. It will be calculate the count for the complete execution.

Create a class file with the following specification.
`BasicStream`
Package: com.ostech.ktable
Class: WordCountLambdaExample.java

Update the following with the IP address of your broker.
final String bootstrapServers = "kafka0:8082";

// Update with the following code in the class file.
Import the following packages and classes:

```
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import java.util.regex.Pattern;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Produced;
```

Define the Input and output topic.

```
public class WordCountLambdaExample {
    final static String inputTopic =  "streams-plaintext-input";
     final static String outputTopic =  "streams-wordcount-output";

    public static void main(String args[]) {
```

```
    }}
```

Define the config properties: The state of the aggregration will be stored in directory specified by `StreamsConfig.STATE_DIR_CONFIG.`

```
private static Properties config() {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-
temperature");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"kafka0:8082");
        props.put(StreamsConfig.STATE_DIR_CONFIG, "//tmp//WC");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");
        props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG,
0);
        return props;

    }
```

Transformation logic define below:

a) Read the stream from input topic.
b) Convert each word of a sentences into array of words like (“kafka”,”again”).
c) Group the list by word i.e “kafka”
d) Finally count the occurrence of the word using count aggregration.

```
public static void countWordByUsingFMValues() {

        Properties props = config();
        StreamsBuilder builder = new StreamsBuilder();

        // Update with the following code in createWordCountStream
method body.
        final KStream<String, String> textLines =
builder.stream(inputTopic);

        final Pattern pattern = Pattern.compile("\\W+",
Pattern.UNICODE_CHARACTER_CLASS);

        final KTable<String, Long> wordCounts = textLines
                .flatMapValues(value ->
Arrays.asList(pattern.split(value.toLowerCase())))
                .groupBy((keyIgnored, word) -> word).count();
```

```
        // Write the `KTable<String, Long>` to the output topic.
        wordCounts.toStream().to(outputTopic,
Produced.with(Serdes.String(), Serdes.Long()));
        // Update till Here

        final KafkaStreams streams = new
KafkaStreams(builder.build(), props);
        final CountDownLatch latch = new CountDownLatch(1);
        // attach shutdown handler to catch control-c
        Runtime.getRuntime().addShutdownHook(new Thread("streams-
temperature-shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
```

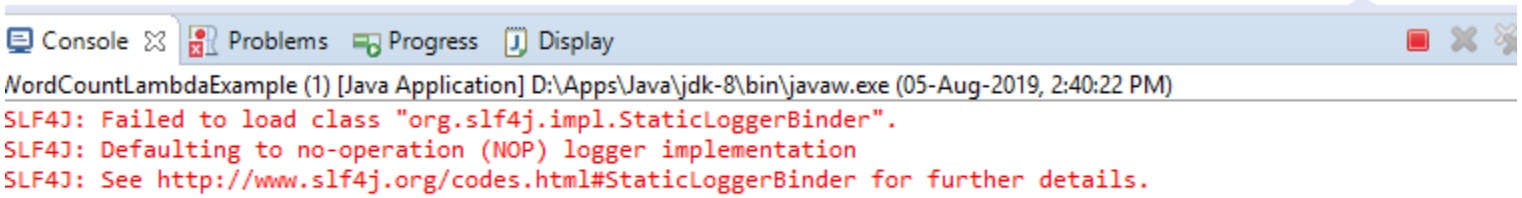
```
        System.exit(0);

    }
```

Executing the Application:

Create the input and output topics used by this example. Execute the Kafka topic command from the bin folder of kafka installation folder.

```
# sh kafka-topics.sh --create --topic streams-plaintext-input \
            --bootstrap-server kafka0:9092 --partitions 1 --replication-factor 1
# sh kafka-topics.sh --create --topic streams-wordcount-output \
            --bootstrap-server kafka0:9092  --partitions 1 --replication-factor 1
```

Start this example application either in your IDE or in the command line.



Start the console producer. You can then enter input data by writing some line of text, followed by ENTER:

* #
* # hello kafka streams<ENTER>

* # all streams lead to kafka<ENTER>
* # join kafka summit<ENTER>
* #
* # Every line you enter will become the value of a single Kafka message.

sh kafka-console-producer.sh --broker-list kafka0:9092 --topic streams-plaintext-input

```
(base) [root@tos ~]# kafka-console-producer --broker-list tos.hp.com:9092 --topi
c streams-plaintext-input1
>Hello
>finally its seems to be workinhg
>Hey
>is it working now
>Great It working
>
```

Inspect the resulting data in the output topic. * You should see output data similar to below. Please note that the exact output * sequence will depend on how fast you type the above sentences. If you type them
* slowly, you are likely to get each count update, e.g., kafka 1, kafka 2, kafka 3.
* If you type them quickly, you are likely to get fewer count updates, e.g., just kafka 3.
* This is because the commit interval is set to 10 seconds. Anything typed within
* that interval will be compacted in memory.

```
# sh kafka-console-consumer.sh  --topic streams-wordcount-output --from-beginning \
                    --bootstrap-server kafka0:9092 \
                   --property print.key=true \
```

--property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer

```
(base) [root@tos logs]# kafka-console-consumer --topic streams-wordcount-output1
 --from-beginning \
>                            --bootstrap-server tos.hp.com:9092 \
>                           --property print.key=true \
>                            --property value.deserializer=org.apache.kafka.co
mmon.serialization.LongDeserializer
hello   1
finally 1
its     1
seems   1
to      1
be      1
workinhg        1
hey     1
is      1
it      1
working 1
now     1
great   1
it      2
working 2
```

Once you're done with your experiments, you can stop this example via {@code Ctrl-C}.

---------- Lab Ends Here ----------

5) DSL - Transform a stream of events – 90 Minutes

In this lab, the following features will be implemented:

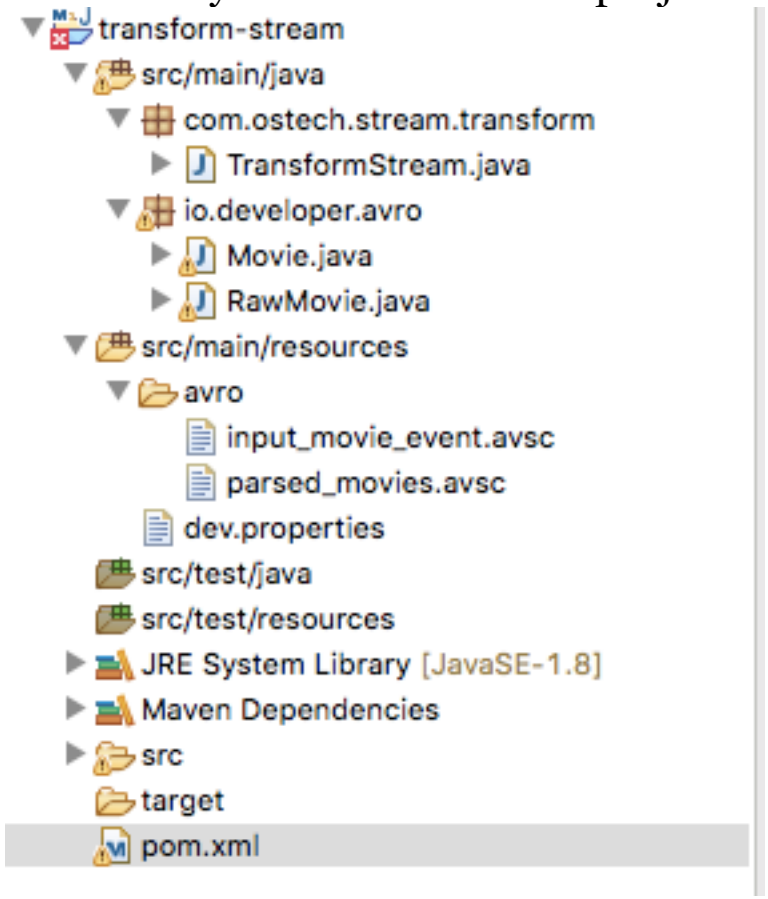
- Use Schema Registry with Avro
  - Define a Schema
- Map Transformation and KeyValue Pair
- Create Topic with AdminClient API

Scenario:
Consider a topic with events that represent **movies**.
Each event has an attribute **title** that combines its title and its release year into a string e.x **Die Hard::1988**.
In this tutorial, we'll write a program that creates a new topic with the title and release date turned into their own attributes.
Define avro schema for input message as well as parsed message
Generate Class file out of the above avro schema using maven-avro plugin

Example message:
Input : {"id":294,"title":"Die Hard::1988","genre":"action"}
Output: {"id":294,"title":"Die Hard","release_year":1988,"genre":"action"}

Create a Maven Project or you can use any existing maven project:
com.ostech:transform-stream

At the end you should have the project structure as shown below:



Create a development configuration file at **src/main/resources/dev.properties**:

```
application.id=transforming-app
bootstrap.servers=localhost:9092
schema.registry.url=http://localhost:8081

input.topic.name=raw-movies
input.topic.partitions=1
input.topic.replication.factor=1

output.topic.name=movies
output.topic.partitions=1
output.topic.replication.factor=1
```

Above we define, the kafka broker address along with the schema registry URL. In addition to it, the input topic (raw-movies) and output topic (movies) is being specified.

Create schema for the events
Create a directory for schemas that represent events in our stream:
src/main/resources/avro

Then create the following Avro schema file at `src/main/resources/avro/input_movie_event.avsc` for the raw movies:

```
{
  "namespace": "com.ostech.avro",
  "type": "record",
  "name": "RawMovie",
  "fields": [
    {"name": "id", "type": "long"},
    {"name": "title", "type": "string"},
    {"name": "genre", "type": "string"}
  ]
}
```

While you're at it, create another Avro schema file at `src/main/resources/avro/parsed_movies.avsc` for the transformed movies:

```
{
  "namespace": "com.ostech.avro",
  "type": "record",
  "name": "Movie",
```

```
  "fields": [
    {"name": "id", "type": "long"},
    {"name": "title", "type": "string"},
    {"name": "release_year", "type": "int"},
    {"name": "genre", "type": "string"}
  ]
}
```

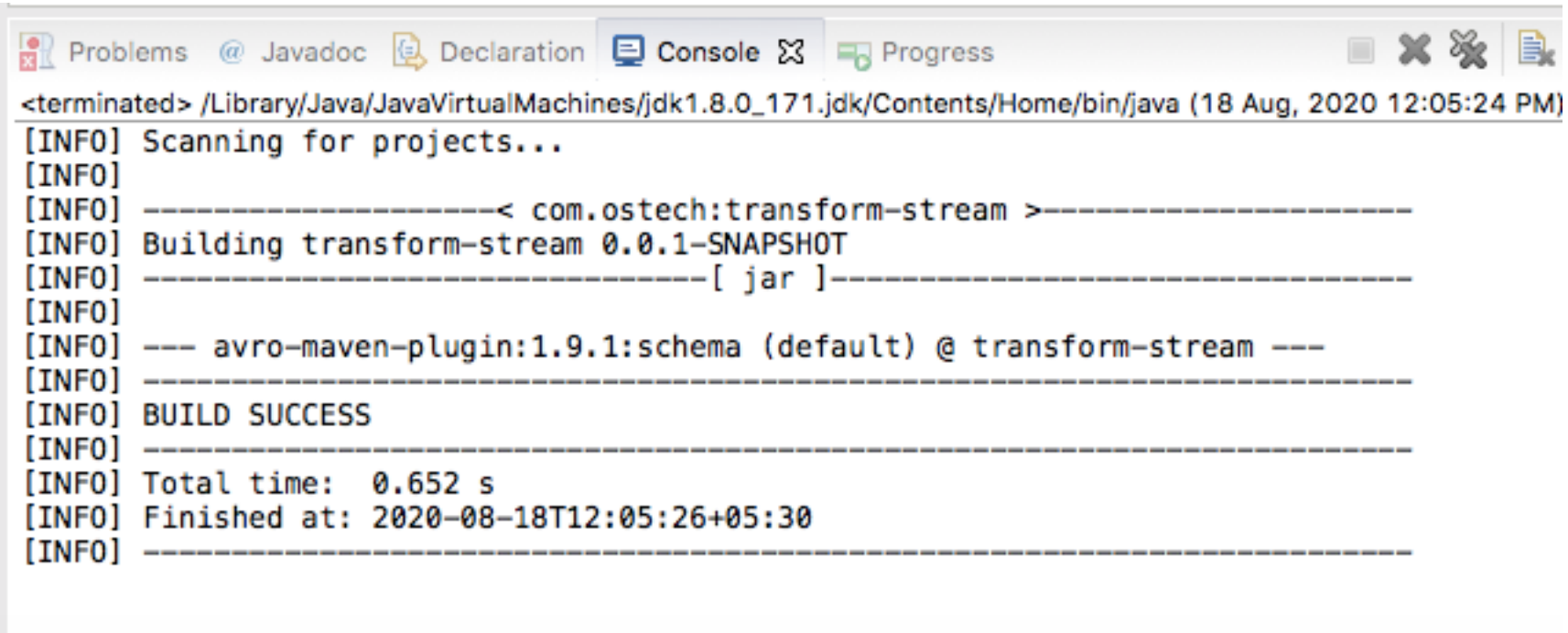
Because we will use this Avro schema in our Java code, we'll need to compile it and generate sources code. The Maven Avro plugin is a part of the build, so it takes the Avro files, generate Java code based on the Avro schema, and compile it and all other Java sources. Run this command to get it all done:

Include the above Avro Schema in the Avro-maven-plugin source directory to generate the source code.

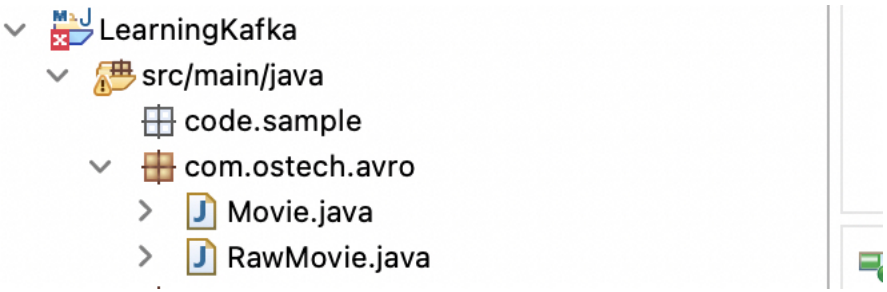
```
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>${avro.version}</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>
                            <sourceDirectory>${project.basedir}/src/main/resources/avro</sourceDirectory>
                            <includes>
                                <include>*.avsc</include>
                            </includes>
                            <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

Verify the Source Directory and its output Directory, it should be as specified above.

Next, perform the following to Generate classes file from the above schema files
Pom.xml -> Run As ->Maven → Generated-Sources



Ensure that generated classes in the package (com.ostech.avro) don't have any compile error.



Create the Kafka Streams topology

Then create the following file `TransformStream.java` inside **package** com.ostech.stream.transform

```
package com.ostech.stream.transform;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

import org.apache.kafka.clients.admin.AdminClient;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
```

```
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.Produced;

import com.ostech.avro.Moviep;
import com.ostech.avro.RawMovie;

import io.confluent.kafka.serializers.AbstractKafkaAvroSerDeConfig;
import io.confluent.kafka.streams.serdes.avro.SpecificAvroSerde;

public class TransformStream {
    public Properties buildStreamsProperties(Properties envProps) {
        Properties props = envProps;

        props.put(StreamsConfig.APPLICATION_ID_CONFIG,
envProps.getProperty("application.id"));
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
envProps.getProperty("bootstrap.servers"));
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
SpecificAvroSerde.class);

props.put(AbstractKafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CONFIG,
envProps.getProperty("schema.registry.url"));
```

```
        props.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 10 *
1000);
        return props;
    }

    public Topology buildTopology(Properties envProps) {
        final StreamsBuilder builder = new StreamsBuilder();
        final String inputTopic =
envProps.getProperty("input.topic.name");
       // final String inputTopic = "raw-movies";
        KStream<String, RawMovie> rawMovies =
builder.stream(inputTopic);
        KStream<Long, Moviep> movies = rawMovies.map((key,
rawMovie) ->
                    new KeyValue<Long, Moviep>(rawMovie.getId(),
convertRawMovie(rawMovie)));
                   //.peek((k,v) -> System.out.println("Key:  " + k + "
, Movie : " + v.getTitle()));


        movies.to("movies", Produced.with(Serdes.Long(),
movieAvroSerde(envProps)));
        return builder.build();
    }
```

```
    public static Moviep convertRawMovie(RawMovie rawMovie) {
        String titleParts[] =
rawMovie.getTitle().toString().split("::");
        String title = titleParts[0];
        int releaseYear = Integer.parseInt(titleParts[1]);
        return new Moviep(rawMovie.getId(), title, releaseYear,
rawMovie.getGenre());
    }
    /*
     * Avro Serde for Pushing to a Topic.
     */
    private SpecificAvroSerde<Moviep> movieAvroSerde(Properties
envProps) {
        SpecificAvroSerde<Moviep> movieAvroSerde = new
SpecificAvroSerde<>();

        final HashMap<String, String> serdeConfig = new
HashMap<>();

serdeConfig.put(AbstractKafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CO
NFIG,
                envProps.getProperty("schema.registry.url"));

        movieAvroSerde.configure(serdeConfig, false);
        return movieAvroSerde;
```

```
    }

    public void createTopics(Properties envProps) {
        Map<String, Object> config = new HashMap<>();
        config.put("bootstrap.servers",
envProps.getProperty("bootstrap.servers"));
        AdminClient client = AdminClient.create(config);

        List<NewTopic> topics = new ArrayList<>();

        topics.add(new NewTopic(
                envProps.getProperty("input.topic.name"),

Integer.parseInt(envProps.getProperty("input.topic.partitions")),

Short.parseShort(envProps.getProperty("input.topic.replication.fact
or"))));

        topics.add(new NewTopic(
                envProps.getProperty("output.topic.name"),

Integer.parseInt(envProps.getProperty("output.topic.partitions")),

Short.parseShort(envProps.getProperty("output.topic.replication.fac
tor"))));
```

```
        client.createTopics(topics);
        client.close();
    }

    public Properties loadEnvProperties(String fileName) throws
IOException {
        Properties envProps = new Properties();
        FileInputStream input = new FileInputStream(fileName);
        envProps.load(input);
        input.close();

        return envProps;
    }

    public static void main(String[] args) throws Exception {
       /* if (args.length < 1) {
            throw new IllegalArgumentException("This program takes
one argument: the path to an environment configuration file.");
        }
        */

        String path =
TransformStream.class.getClassLoader().getResource("dev.properties"
).getPath();
```

```
        TransformStream ts = new TransformStream();
        Properties envProps = ts.loadEnvProperties(path);
        envProps.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());

envProps.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
SpecificAvroSerde.class);

        Properties streamProps =
ts.buildStreamsProperties(envProps);
        Topology topology = ts.buildTopology(streamProps);

        ts.createTopics(envProps);

        final KafkaStreams streams = new KafkaStreams(topology,
streamProps);
        final CountDownLatch latch = new CountDownLatch(1);

        // Attach shutdown handler to catch Control-C.
        Runtime.getRuntime().addShutdownHook(new Thread("streams-
shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
```

```
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

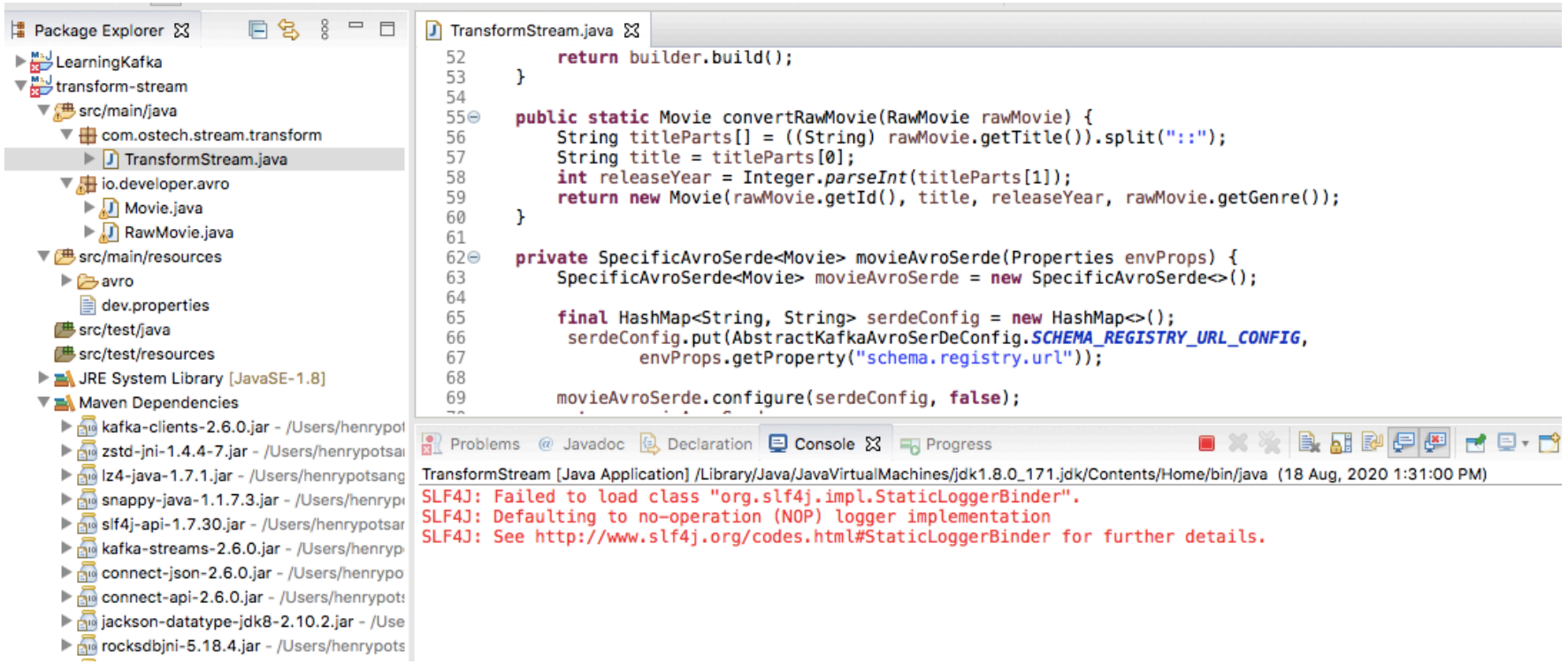
`Description :`

The first thing the method does is create an instance of `StreamsBuilder`, which is the helper object that lets us build our topology. Next we call the `stream()` method, which creates a `KStream` object (called `rawMovies` in this case) out of an underlying Kafka topic. Note the type of that stream is `Long, RawMovie`, because the topic contains the raw movie objects we want to transform. RawMovie's `title` field contains the title and the release year together, which we want to make into separate fields in a new object.

We get that transforming work done with the next line, which is a call to the `map()` method. `map()` takes each input record and creates a new stream with transformed records in it. Its parameter is a single Java Lambda that takes the input key and value and returns an instance of the `KeyValue` class with the new record in it. This does two things. First, it rekeys the incoming stream, using the `movieId` as the key. We don't absolutely need to do that to accomplish the transformation, but it's easy enough to do at the same time, and it sets a useful key on the output stream, which is generally a good idea. Second, it calls the `convertRawMovie()` method to turn the `RawMovie` value into a `Movie`. This is the essence of the transformation.
The `convertRawMovie()` method contains the sort of unpleasant string parsing that is a part of many stream processing pipelines, which we are happily able to encapsulate in a single, easily testable method. Any further stages we might build in the pipeline after this point are blissfully unaware that we ever had a string to parse in the first place.
Moreover, it's worth noting that we're calling `map()` and not `mapValues()`:

Start kafka and the registry server.

Execute the Kafka Streams program



Verify that Topics (raw-movies and movies) are created.

Execute the following command in a terminal

```
# kafka-topics --bootstrap-server kafka0:9092 --list
```

```
[root@kafka0 /]# kafka-topics --bootstrap-server kafka0:9092 --list
__consumer_offsets
_schemas
click
clicks
events
events-country
impressions
movies
my-first-topic
raw-movies
```

Produce events to the input topic

In a new terminal, run:

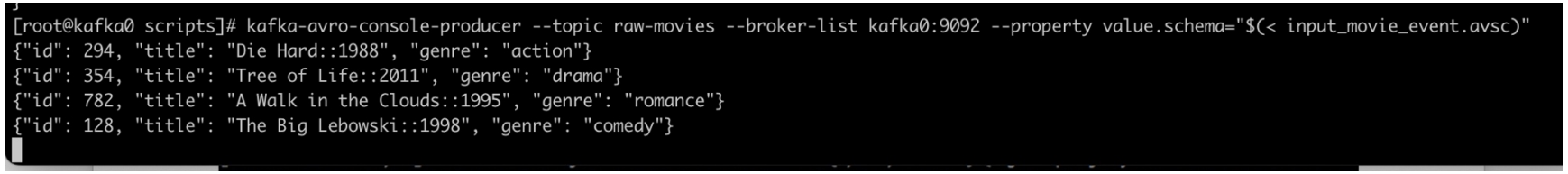
Copy the avsc file created earliert in a specific folder and Execute the following from that folder.

```
#kafka-avro-console-producer --topic raw-movies --broker-list kafka0:9092 --property value.schema="$(< input_movie_event.avsc)"
```

When the console producer starts, it will log some messages and hang, waiting for your input. Type the following, one line at a time and press enter to send it. Each line represents

an event. To send all of the events below, paste the following into the prompt and press enter:

```
{"id": 294, "title": "Die Hard::1988", "genre": "action"}
{"id": 354, "title": "Tree of Life::2011", "genre": "drama"}
{"id": 782, "title": "A Walk in the Clouds::1995", "genre": "romance"}
{"id": 128, "title": "The Big Lebowski::1998", "genre": "comedy"}
```



At registry log, it should log an url invocation as shown below(focus the last line of the log).

Observe the transformed movies in the output topic.

Leave your original terminal running. To consume the events produced by your Streams application you'll need another terminal open.

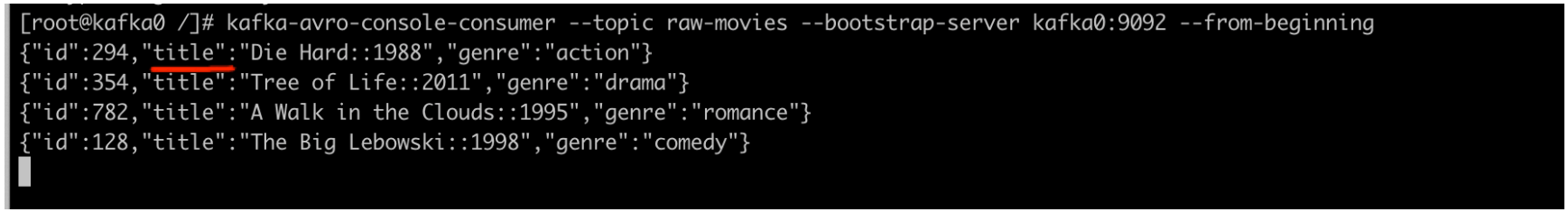
First, to consume the events of drama films(Output topic after transformation), run the following:
This should yield the following messages:

```
#kafka-avro-console-consumer --topic movies --bootstrap-server kafka0:9092 --from-beginning
```

```
[root@kafka0 /]# kafka-avro-console-consumer --topic movies --bootstrap-server kafka0:9092 --from-beginning
{"id":294,"title":"Die Hard","release_year":1988,"genre":"action"}
{"id":354,"title":"Tree of Life","release_year":2011,"genre":"drama"}
{"id":782,"title":"A Walk in the Clouds","release_year":1995,"genre":"romance"}
{"id":128,"title":"The Big Lebowski","release_year":1998,"genre":"comedy"}
```

Open another terminal, you can consume the raw-movies(before transformation) as shown below:

kafka-avro-console-consumer --topic raw-movies --bootstrap-server kafka0:9092 --from-beginning



As shown above, observe that title has movie name along with year of release in a single attribute in the raw-movies topic and movie topic has sperated fields respectively.

Example “"title":"Die Hard::1988"” -> "title":"Die Hard","release_year":1988

Task:

Add: peek method to get a print after transformation:

```
peek((k,v) -> System.out.println("Key:  " + k + " , Movie : " +
v.getTitle()));
```

```
44        KStream<String, RawMovie> rawMovies = builder.stream(inputTopic);
45        KStream<Long, Movie> movies = rawMovies.map((key, rawMovie) ->
46                new KeyValue<Long, Movie>(rawMovie.getId(), convertRawMovie(rawMovie)))
47                .peek((k,v) -> System.out.println("Key:  " + k + " , Movie : " + v.getTitle()));
48
49      //  movies.to("movies", Produced.with(Serdes.Long(), movieAvroSerde(envProps)));
50
```

```
Console
TransformStream (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java  (03-Mar-2022, 10:35:23 PM)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Key:  128 , Movie : The Big Lebowski
```

Execute the program or restart it.

Your output should be as shown above, when you enter the following in the producer console.

{"id": 128, "title": "The Big Lebowski::1998", "genre": "comedy"}

```
[root@kafka0 code]# kafka-avro-console-producer --topic raw-movies --broker-list kafka0:9092 --property value.schema="$(<
input_movie_event.avsc)"
{"id": 294, "title": "Die Hard::1988", "genre": "action"}
{"id": 354, "title": "Tree of Life::2011", "genre": "drama"}
{"id": 782, "title": "A Walk in the Clouds::1995", "genre": "romance"}
{"id": 128, "title": "The Big Lebowski::1998", "genre": "comedy"}
{"id": 128, "title": "The Big Lebowski::1998", "genre": "comedy"}
```

---------- Lab Ends Here ----------

6) DSL - Stateful transformations – reduce & count – 60 Minutes

Demonstrate the following APIs:

- Ktable
- Filter
- selectKey
- Aggregration: GroupBy & reduce

Demonstrates how to use `reduce` to sum numbers(rolling addition of the number).

#kafka-topics --create --topic numbers-topic --bootstrap-server kafka0:9092 --partitions 1 --replication-factor 1

#kafka-topics --create --topic sum-of-odd-numbers-topic --bootstrap-server kafka0:9092 --partitions 1 --replication-factor 1

We will create two java classes in this lab. You can use any existing maven project with the pom updated as specified in the annexure.

Create a java class, **SumLambdaExample** and replace the code which is provided below. It Perform the transformation.

// Code Begins Here.

```
package com.tos.stream.dsl;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;

public class SumLambdaExample {
	static final String SUM_OF_ODD_NUMBERS_TOPIC = "sum-of-odd-
numbers-topic";
	static final String NUMBERS_TOPIC = "numbers-topic";

	public static void main(final String[] args) {
```

```
            final String bootstrapServers = args.length > 0 ?
args[0] : "kafka0:9092";
            final Properties streamsConfiguration = new
Properties();

streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "sum-
lambda-example");

streamsConfiguration.put(StreamsConfig.CLIENT_ID_CONFIG, "sum-
lambda-example-client");

streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);

streamsConfiguration.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONF
IG, Serdes.Integer().getClass().getName());

streamsConfiguration.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CO
NFIG, Serdes.Integer().getClass().getName());

streamsConfiguration.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");

streamsConfiguration.put(StreamsConfig.STATE_DIR_CONFIG,
"\\tmp\\kafka-streams");
```

```
            // Records should be flushed every 10 seconds. This is
less than the default
            // in order to keep this example interactive.

streamsConfiguration.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG,
10 * 1000);

            final Topology topology = getTopology();
            final KafkaStreams streams = new KafkaStreams(topology,
streamsConfiguration);

            streams.cleanUp();
            streams.start();

            // Add shutdown hook to respond to SIGTERM and
gracefully close Kafka Streams
            Runtime.getRuntime().addShutdownHook(new
Thread(streams::close));
        }

        static Topology getTopology() {
            final StreamsBuilder builder = new StreamsBuilder();
            // We assume the input topic contains records where the
values are Integers.
```

```
            // We don't really care about the keys of the input
records;  for simplicity, we assume them
            // to be Integers, too, because we will re-key the
stream later on, and the new key will be
            // of type Integer.
            final KStream<Integer, Integer> input =
builder.stream(NUMBERS_TOPIC);

            final KTable<Integer, Integer> sumOfOddNumbers = input
              // We are only interested in odd numbers.
              .filter((k, v) -> v % 2 != 0)
              // We want to compute the total sum across ALL
numbers, so we must re-key all records to the
              // same key.  This re-keying is required because in
Kafka Streams a data record is always a
              // key-value pair, and KStream aggregations such as
`reduce` operate on a per-key basis.
              // The actual new key (here: `1`) we pick here
doesn't matter as long it is the same across
              // all records.
              .selectKey((k, v) -> 1)
              // no need to specify explicit serdes because the
resulting key and value types match our default serde settings
              .groupByKey()
              // Add the numbers to compute the sum.
```

```
                .reduce((v1, v2) -> v1 + v2);

sumOfOddNumbers.toStream().to(SUM_OF_ODD_NUMBERS_TOPIC);

            return builder.build();
        }

    }
```

// Code Ends Here.

Create another class file: SumLambdaExampleDriver.
It will Produce message for the input topic and consume the transform message from the Output topic.

// Code Begins Here.

```
package com.tos.stream.dsl;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
import java.util.stream.IntStream;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
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.IntegerDeserializer;
import org.apache.kafka.common.serialization.IntegerSerializer;

public class SumLambdaExampleDriver {

	public static void main(final String[] args) {
```

```
        final String bootstrapServers = "kafka0:9092";
        produceInput(bootstrapServers);
        consumeOutput(bootstrapServers);
    }

    private static void consumeOutput(final String
bootstrapServers) {
        final Properties properties = new Properties();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
IntegerDeserializer.class);

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
IntegerDeserializer.class);
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "sum-lambda-
example-consumer");
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");
        final KafkaConsumer<Integer, Integer> consumer = new
KafkaConsumer<>(properties);

consumer.subscribe(Collections.singleton(SumLambdaExample.SUM_OF_OD
D_NUMBERS_TOPIC));
```

```
        while (true) {
          final ConsumerRecords<Integer, Integer> records =
                  consumer.poll(Duration.ofMillis(Long.MAX_VALUE));

          for (final ConsumerRecord<Integer, Integer> record :
records) {
               System.out.println("Current sum of odd numbers is:"
+ record.value());
          }
        }

      }

      private static void produceInput(final String
bootstrapServers) {
        final Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);

        final KafkaProducer<Integer, Integer> producer = new
KafkaProducer<>(props);
```

```
        IntStream.range(0, 100)
                .mapToObj(val -> new
ProducerRecord<>(SumLambdaExample.NUMBERS_TOPIC, val, val))
                .forEach(producer::send);

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


}
```

// Code Ends Here.

Execute the following sequentially.

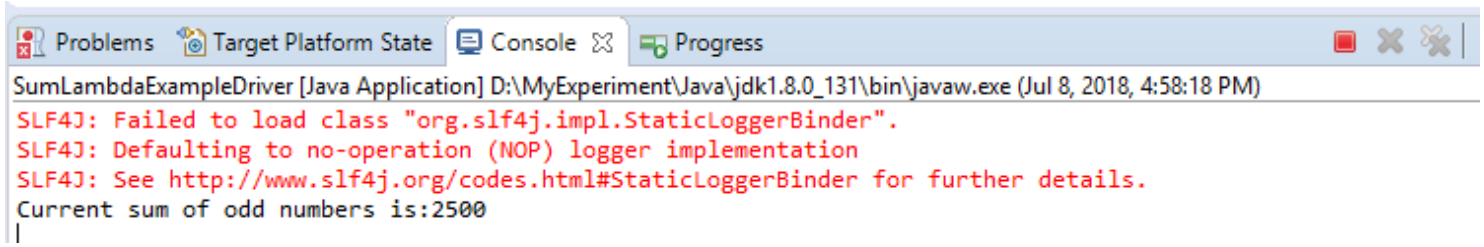
SumLambdaExample → It subscribes to the input topic (`numbers-topic`), perform transformation on the number inserted into the topic and insert the output to the destination topic(`sum-of-odd-numbers-topic`).

Transformation:

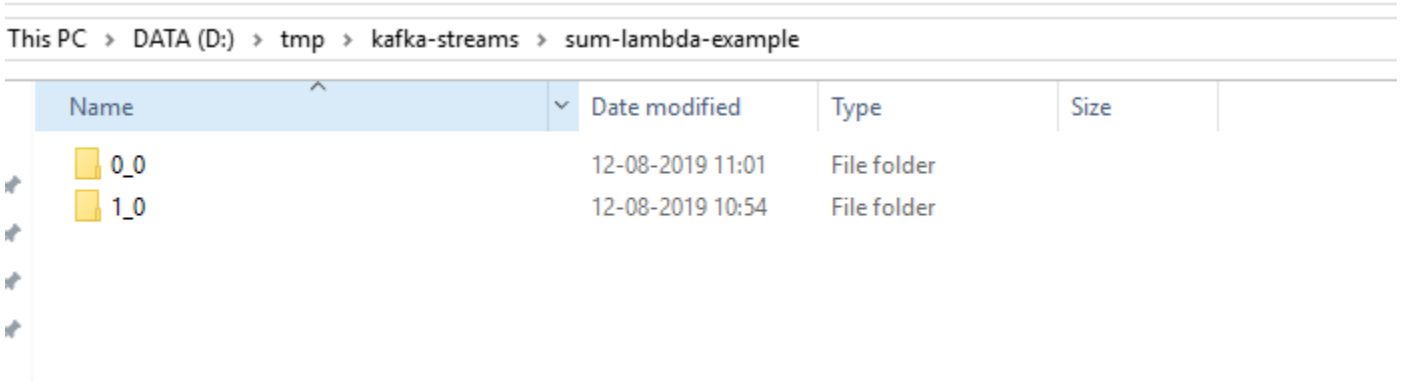
We are only interested in odd numbers. – Apply Using ***filter()***
We want to compute the total sum across ALL odd numbers, so we must re-key all records to the same key. (***selectKey((k, v) -> 1)***This re-keying is required because in Kafka Streams a data record is always a key-value pair, and KStream aggregations such as `reduce` operate on a per-key basis.
The actual new key (here: `1`) we pick here doesn't matter as long it is the same across all records.
(groupByKey())no need to specify explicit serdes because the resulting key and value types match our default serde settings

reduce((v1, v2) -> v1 + v2): Add the numbers to compute the sum.

SumLambdaExampleDriver → It generates messages in the range of 0 to 100 number and sent messages to the i/p topic and consume from the o/p topic(Sum of all odd numbers) and display it to the console.



If you get error while running subsequently. Clear the following folder manually.



This is where messages are store for state full calculation of message. i.e sum of odd numbers

Inspect the resulting data in the output topics,

```
#kafka-console-consumer --topic sum-of-odd-numbers-topic --from-beginning \
      --bootstrap-server kafka0:9092 \
```

--property
value.deserializer=org.apache.kafka.common.serialization.IntegerDeserializer

```
[root@kafka0 bin]# kafka-console-consumer --topic sum-of-odd-numbers-topic --from-beginning \
>       --bootstrap-server kafka0:9092 \
>       --property value.deserializer=org.apache.kafka.common.serialization.IntegerDeserializer
2500
```

Next Let us implement another word count using count method.

Create a java class – **WordCountApplication**.java in the following package.

```
package com.tos.kafka.stream
```

Import the following packages and classes.

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Produced;
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Arrays;
import java.util.Locale;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
```

It configures the parameter required to connect to the kafka broker.

```
public static final String INPUT_TOPIC = "streams-plaintext-input";
public static final String OUTPUT_TOPIC = "streams-wordcount-
output";

    static Properties getStreamsConfig(final String[] args) throws
IOException {
        final Properties props = new Properties();
        if (args != null && args.length > 0) {
            try (final FileInputStream fis = new
FileInputStream(args[0])) {
                props.load(fis);
            }
            if (args.length > 1) {
```

```
                System.out.println("Warning: Some command line
arguments were ignored. This demo only accepts an optional
configuration file.");
            }
        }
        props.putIfAbsent(StreamsConfig.APPLICATION_ID_CONFIG,
"streams-wordcount");
        props.putIfAbsent(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"kafka0:9092");

props.putIfAbsent(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG,
0);

props.putIfAbsent(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
              Serdes.String().getClass().getName());

props.putIfAbsent(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
              Serdes.String().getClass().getName());

        // setting offset reset to earliest so that we can re-run
the demo code with the same pre-loaded data
        // Note: To re-run the demo, you need to use the offset
reset tool:
```

```
        //
https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+App
lication+Reset+Tool
        props.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");
        return props;
    }
```

Next code creates an instance of stream connecting to input topic and perform the transformation.

```
  static void createWordCountStream(final StreamsBuilder builder) {
        final KStream<String, String> source =
builder.stream(INPUT_TOPIC);

        final KTable<String, Long> counts = source.peek( (k,v) ->
System.out.println("Value" + v))
            .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split("\\W+"))
)
            .groupBy((key, value) -> value)
            .count();
```

```
        // need to override value serde to Long type
        counts.toStream().to(OUTPUT_TOPIC,
Produced.with(Serdes.String(), Serdes.Long()));
    }
```

It implements the WordCount algorithm, which computes a word occurrence histogram from the input text. However, unlike other WordCount examples you might have seen before that operate on bounded data, the WordCount demo application behaves slightly differently because it is designed to operate on an **infinite, unbounded stream** of data. Similar to the bounded variant, it is a stateful algorithm that tracks and updates the counts of words. However, since it must assume potentially unbounded input data, it will periodically output its current state and results while continuing to process more data because it cannot know when it has processed "all" the input data.

Update the following code in the main method().

```
final Properties props = getStreamsConfig(args);

        final StreamsBuilder builder = new StreamsBuilder();
        createWordCountStream(builder);
        final KafkaStreams streams = new
KafkaStreams(builder.build(), props);
        final CountDownLatch latch = new CountDownLatch(1);

        // attach shutdown handler to catch control-c
```

```
        Runtime.getRuntime().addShutdownHook(new Thread("streams-
wordcount-shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (final Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
```

At the end, you should have the following code structure:

```
public class WordCountDemo {

    public static final String INPUT_TOPIC = "streams-plaintext-input";
    public static final String OUTPUT_TOPIC = "streams-wordcount-output";

    static Properties getStreamsConfig(final String[] args) throws IOException {
        final Properties props = new Properties();
        if (args != null && args.length > 0) {
            try (final FileInputStream fis = new FileInputStream(args[0])) {
                props.load(fis);
            }
            if (args.length > 1) {
                System.out.println("Warning: Some command line arguments were ignored. This demo only accep
            }
        }
        props.putIfAbsent(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount");
        props.putIfAbsent(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:8082");
        props.putIfAbsent(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
        props.putIfAbsent(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
                Serdes.String().getClass().getName());
        props.putIfAbsent(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
                Serdes.String().getClass().getName());

        // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded da
        // Note: To re-run the demo, you need to use the offset reset tool:
        // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
        props.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        return props;
```

```
static void createWordCountStream(final StreamsBuilder builder) {
    final KStream<String, String> source = builder.stream(INPUT_TOPIC);

    final KTable<String, Long> counts = source.peek( (k,v) -> System.out.println("Value" + v))
        .flatMapValues(value -> Arrays.asList(value.toLowerCase(Locale.getDefault()).split("\\W+")))
        .groupBy((key, value) -> value)
        .count();

    // need to override value serde to Long type
    counts.toStream().to(OUTPUT_TOPIC, Produced.with(Serdes.String(), Serdes.Long()));
}

public static void main(final String[] args) throws IOException {
    final Properties props = getStreamsConfig(args);

    final StreamsBuilder builder = new StreamsBuilder();
    createWordCountStream(builder);
    final KafkaStreams streams = new KafkaStreams(builder.build(), props);
    final CountDownLatch latch = new CountDownLatch(1);

    // attach shutdown handler to catch control-c
    Runtime.getRuntime().addShutdownHook(new Thread("streams-wordcount-shutdown-hook") {
        @Override
        public void run() {
            streams.close();
            latch.countDown();
        }
```

```
        try {
            streams.start();
            latch.await();
        } catch (final Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

Start the Kafka server.

Prepare input topic and start Kafka producer

Next, we create the input topic named streams-plaintext-input and the output topic named streams-wordcount-output:

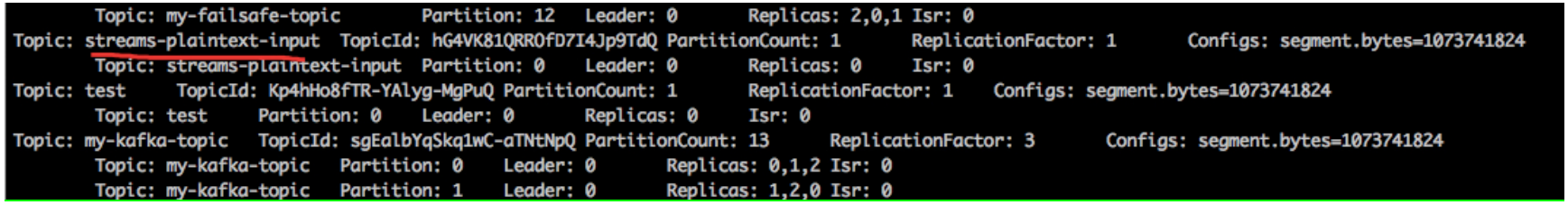
```
# /opt/kafka/bin/kafka-topics.sh --create \
  --bootstrap-server kafka0:9092 \
  --replication-factor 1 \
  --partitions 1 \
  --topic streams-plaintext-input
```

Note: we create the output topic with compaction enabled because the output stream is a changelog stream.

```
# /opt/kafka/bin/kafka-topics.sh --create \
   --bootstrap-server kafka0:9092 \
   --replication-factor 1 \
   --partitions 1 \
   --topic streams-wordcount-output \
   --config cleanup.policy=compact
```

The topics created above can be described with the same kafka-topics tool:

```
#/opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka0:9092 --describe
```



Start the Wordcount Application.

The demo application will read from the input topic **streams-plaintext-input**, perform the computations of the WordCount algorithm on each of the read messages, and continuously write its current results to the output topic **streams-wordcount-output**. Hence there won't be any STDOUT output except log entries as the results are written back into in Kafka.

Now we can start the console producer in a separate terminal to write some input data to this topic:

```
#/opt/kafka/bin/kafka-console-producer.sh --bootstrap-server kafka0:9092 --topic streams-plaintext-input
```

and inspect the output of the WordCount demo application by reading from its output topic with the console consumer in a separate terminal:

```
# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 \
  --topic streams-wordcount-output \
  --from-beginning \
  --formatter kafka.tools.DefaultMessageFormatter \
  --property print.key=true \
  --property print.value=true \
  --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
  --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```

Process some data

Now let's write some message with the console producer into the input topic **streams-plaintext-input** by entering a single line of text and then hit <RETURN>. This will send a new message to the input topic, where the message key is null and the message value is the string encoded text line that you just entered (in practice, input data for applications will typically be streaming continuously into Kafka, rather than being manually entered):

```
all streams lead to kafka
```

Console will be as shown below:

```
[root@kafka0 scripts]#
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-producer.sh --bootstrap-server kafka0:9092 --topic streams-plaintext-input
>all streams lead to kafka
>
```

This message will be processed by the Wordcount application and the following output data will be written to the **streams-wordcount-output** topic and printed by the console consumer:

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 \
>     --topic streams-wordcount-output \
>     --from-beginning \
>     --formatter kafka.tools.DefaultMessageFormatter \
>     --property print.key=true \
>     --property print.value=true \
>     --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
>     --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
all     1
streams 1
lead    1
to      1
kafka   1
```

Here, the first column is the Kafka message key in java.lang.String format and represents a word that is being counted, and the second column is the message value in java.lang.Longformat, representing the word's latest count.

Now let's continue writing one more message with the console producer into the input topic **streams-plaintext-input**. Enter the text line "hello kafka streams" and hit <RETURN>. Your terminal should look as follows:

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-producer.sh --bootstrap-server kafka0:9092 --topic streams-plaintext-input
>all streams lead to kafka
>hello kafka streams
>
```

In your other terminal in which the console consumer is running, you will observe that the WordCount application wrote new output data:

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 \
>     --topic streams-wordcount-output \
>     --from-beginning \
>     --formatter kafka.tools.DefaultMessageFormatter \
>     --property print.key=true \
>     --property print.value=true \
>     --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
>     --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
all     1
streams 1
lead    1
to      1
kafka   1
hello   1
kafka   2
streams 2
```

Here the last printed lines kafka 2 and streams 2 indicate updates to the keys kafka and streams whose counts have been incremented from 1 to 2. Whenever you write further input messages to the input topic, you will observe new messages being added to the streams-wordcount-output topic, representing the most recent word counts as computed by the WordCount application. Let's enter one final input text line "join kafka training" and hit <RETURN> in the console producer to the input topic streams-plaintext-input before we wrap up this quickstart:

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic streams-plaintext-input
>all streams lead to kafka
>hello kafka streams
>join kafka training
>
```

The streams-wordcount-output topic will subsequently show the corresponding updated word counts (see last three lines):

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
>     --topic streams-wordcount-output \
>     --from-beginning \
>     --formatter kafka.tools.DefaultMessageFormatter \
>     --property print.key=true \
>     --property print.value=true \
>     --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
>     --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer

all     1
streams 1
lead    1
to      1
kafka   1
hello   1
kafka   2
streams 2
join    1
kafka   3
training        1
```

As one can see, outputs of the Wordcount application is actually a continuous stream of updates, where each output record (i.e. each line in the original output above) is an updated count of a single word, aka record key such as "kafka". For multiple records with the same key, each later record is an update of the previous one.

You can now stop the console consumer, the console producer, the Wordcount application, the Kafka broker and the ZooKeeper server in order via **Ctrl-C**
----------Lab Ends Here ----------
https://kafka.apache.org/31/documentation/streams/quickstart

7) Stream – DSL and Windows – 60 Minutes

The following features will be demonstrated:

- Window - Tumbling
- Reduce and Filter
- Window Surpress.

Demonstrates, using the high-level KStream DSL, how to implement an IoT demo application which ingests temperature value processing the maximum value in the latest TEMPERATURE_WINDOW_SIZE seconds (which * is 5 seconds) sending a new message if it exceeds the TEMPERATURE_THRESHOLD (which is 5)

In this example, the input stream reads from a topic named "iot-temperature", where the values of messages represent temperature values; using a TEMPERATURE_WINDOW_SIZE seconds "tumbling" window, the maximum value is processed and sent to a topic named "iot-temperature-max" if it exceeds the TEMPERATURE_THRESHOLD.

You can use any of the Project you have created earlier.

Create a java class TemperatureDemo.java in package: `com.ostech.windows`.

Copy the following logic in the class file.

```java
package com.ostech.windows;

import java.time.Duration;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serde;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.Printed;
import org.apache.kafka.streams.kstream.Produced;
import org.apache.kafka.streams.kstream.TimeWindows;
import org.apache.kafka.streams.kstream.Windowed;
import org.apache.kafka.streams.kstream.WindowedSerdes;

public class TemperatureDemoWithoutSurpress {

    // threshold used for filtering max temperature values
    private static final int TEMPERATURE_THRESHOLD = 5;
    // window size within which the filtering is applied
```

```
    private static final int TEMPERATURE_WINDOW_SIZE = 1;

    /*
     *  Case 1: Without Cache and window trigger for each event
with
     *  the intermediate event
     *
     *  Case 2: Enable surpress to trigger the window output after
windows get closed only.
     *
     */
    public static void main(String[] args) {

        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-
temperature");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"kafka0:9092");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
       // props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.Long().getClass());
```

```
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");
        props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG,
0);

        StreamsBuilder builder = new StreamsBuilder();

         KStream<String,String> source = builder.stream("iot-
temperature");

        // Define the Tumbling Windows:
        TimeWindows tumblingWindow =

    TimeWindows.ofSizeWithNoGrace(Duration.ofSeconds(TEMPERATURE_WI
NDOW_SIZE));

        final KStream<Windowed<String>, String> max = source
                .peek((key,value)-> System.out.println("Incoming
record - key " +key +" value " + value))
                 // temperature values are sent without a key
(null), so in order
                 // to group and reduce them, a key is needed
("temp" has been chosen)
                .selectKey((key, value) -> {
                   return "temp";
```

```
                 })
                .filter((k,v)-> !v.trim().equals(""))
                .groupByKey()
                 .windowedBy(tumblingWindow)
                 .reduce((value1, value2) -> {
                    if ((value1 != null) && (!value1.equals(""))
                         && (value2 !=null ) && (!value2.equals("")
)) {
                         if (Integer.parseInt((String)value1) >
Integer.parseInt((String)value2)) {
                             return value1;
                         } else {
                             return value2;
                         }
                    }else
                        return "0";
                 })
                 .toStream()
                 .filter((key, value) ->
Integer.parseInt((String)(value)) > TEMPERATURE_THRESHOLD);

        final Serde<Windowed<String>> windowedSerde  =
WindowedSerdes.timeWindowedSerdeFrom(String.class,
TEMPERATURE_WINDOW_SIZE);
```

```
        // print the stream to console.
         Printed p= Printed.toSysOut().withLabel("MyStream");
         max.print(p);

        // need to override key serde to Windowed type
         max.to("iot-temperature-max", Produced.with(windowedSerde,
Serdes.String()));

         final KafkaStreams streams = new
KafkaStreams(builder.build(), props);
         final CountDownLatch latch = new CountDownLatch(1);

         // attach shutdown handler to catch control-c
         Runtime.getRuntime().addShutdownHook(new Thread("streams-
temperature-shutdown-hook") {
             @Override
             public void run() {
                 streams.close();
                 latch.countDown();
             }
         });

         try {
             streams.start();
```

```
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

Execution:

Before running this example, you must create input topic for temperature values
Using the following command:
#kafka-topics --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic iot-temperature

And at same time create the output topic for filtered values:

#kafka-topics --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic iot-temperature-max

After that, a consumer console can be started in order to read max temperature for a window from the "iot-temperature-max" topic:

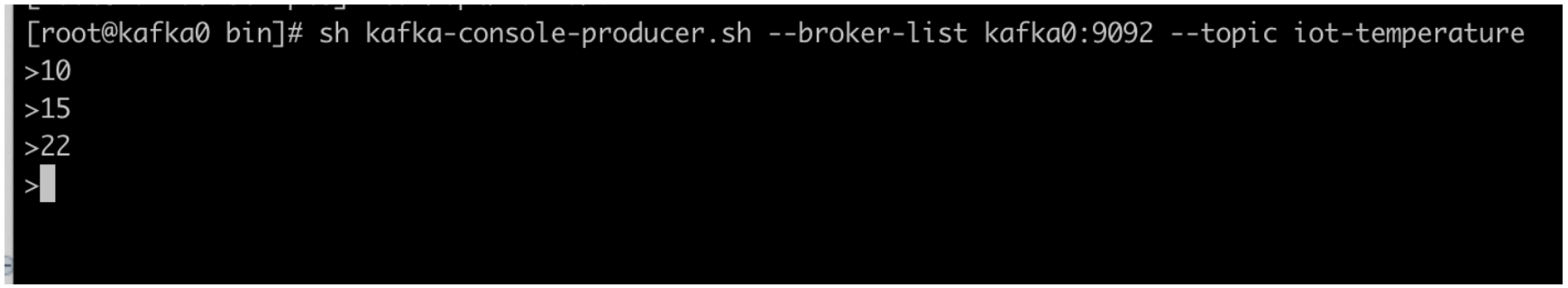
#kafka-console-consumer --bootstrap-server kafka0:9092 --topic iot-temperature-max --from-beginning

In a seperate terminal, a producer console, can be used for sending temperature values (which needs to be integers) to "**iot-temperature**" topic typing them on the console as shown below :

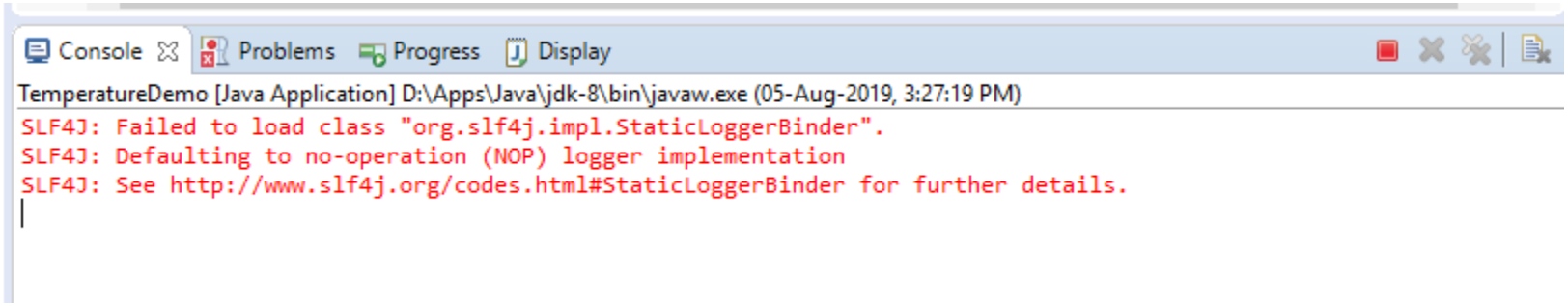
#kafka-console-producer --broker-list kafka0:9092 --topic iot-temperature
> 10
> 15
> 22

The Max temperature is calculated for each window size of 90 seconds, which is specified by the following variable.

```
private static final int TEMPERATURE_WINDOW_SIZE = 90;
```



Execute the java program.



Observe the max temperature in the Consumer console.

You can try with some more inputs and wait for 90 seconds to view the result.
e.x -> 35, 32, 36, 34
If you observed keenly, the window triggers an output for each input. The intermediate result gets publish to the output topic.
Whenever you enter a temperature on the producer console, imediately a result will be shown on the consumer console, although the window is not closed.
i.e type 10 → On consumer Console : 10. (max of 10)
type 15 → On consumer console : 15. (max of 10,15) etc.

If you want to publish the final result of a window only then enable the following and observe the result.
Enable **surpress** to trigger the window output after windows get closed only.

Add the following import

```
import org.apache.kafka.streams.state.Stores;
import org.apache.kafka.streams.state.WindowStore;
import org.apache.kafka.streams.kstream.Materialized;
import static org.apache.kafka.streams.kstream.Suppressed.*;
import static
org.apache.kafka.streams.kstream.Suppressed.BufferConfig.*;
```

Add the following declaration in the main method.

```
props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);

// Define a materialize store
        /*
         * In reduce, the result data type cant be changed.
         */
```

```
        Materialized<String, String, WindowStore<Bytes, byte[]>> tm
=
                Materialized.<String,
String>as(Stores.persistentWindowStore("mws",
                        Duration.ofMinutes(TEMPERATURE_WINDOW_SIZE),
                        Duration.ofMinutes(TEMPERATURE_WINDOW_SIZE),
false))
                .withKeySerde(Serdes.String())
                .withValueSerde(Serdes.String());
```

Then configure materialize store along with Serde in the following method to enable the **surpress** parameter.

```
, tm).suppress(untilWindowCloses(unbounded()))
```

```
final KStream<Windowed<String>, String> max = source
        .peek((key,value)-> System.out.println("Incoming record - key " +key +" value " + value))
        // temperature values are sent without a key (null), so in order
        // to group and reduce them, a key is needed ("temp" has been chosen)
        .selectKey((key, value) -> {
            return "temp";

        })
        .groupByKey()
        .windowedBy(tumblingWindow)
        .reduce((value1, value2) -> {
            if (Integer.parseInt((String)value1) > Integer.parseInt((String)value2)) {
                return value1;
            } else {
                return value2;
            }
        }, tm)
        .suppress(untilWindowCloses(unbounded()))
        .toStream()
        .filter((key, value) -> Integer.parseInt((String)(value)) > TEMPERATURE_THRESHOLD);
```

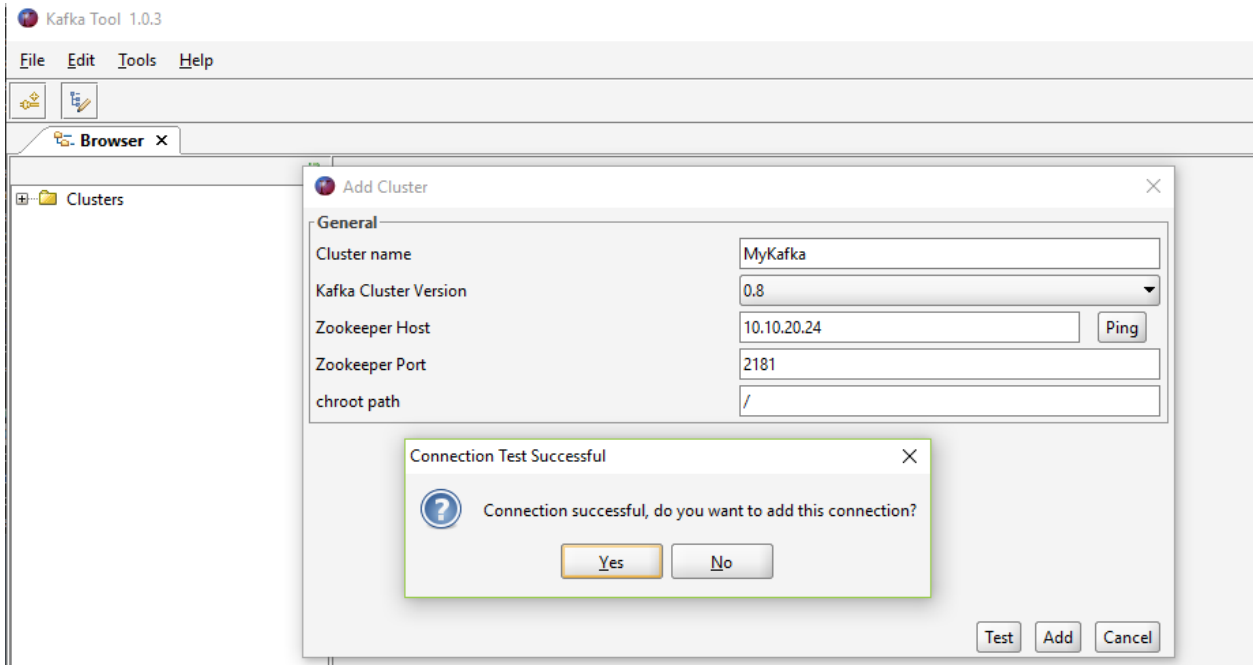
Stop and Execute the program.

On the producer console:

Enter 32,30,31 and 28 observe that the windows result will be emitted at end of the window and when trigger by another event after that window interval.

---------- Lab Ends Here ----------

8)Kafkatools



9)Pom.xml

```
<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.hp.tos</groupId>
    <artifactId>LearningKafka</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>All About Kafka</description>
    <properties>
        <algebird.version>0.13.4</algebird.version>
        <avro.version>1.8.2</avro.version>
        <avro.version>1.8.2</avro.version>
        <confluent.version>5.3.0</confluent.version>
        <kafka.version>3.0.0</kafka.version>
        <kafka.scala.version>2.11</kafka.scala.version>
    </properties>
    <repositories>
        <repository>
            <id>confluent</id>
            <url>https://packages.confluent.io/maven/</url>
        </repository>
    </repositories>
```

```
<pluginRepositories>
    <pluginRepository>
        <id>confluent</id>
        <url>https://packages.confluent.io/maven/</url>
    </pluginRepository>
</pluginRepositories>
<dependencies>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-streams-avro-serde</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-schema-registry-client</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
```

```
        <artifactId>kafka-clients</artifactId>
        <version>${kafka.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>${kafka.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams-test-utils</artifactId>
        <version>${kafka.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro</artifactId>
        <version>${avro.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro-maven-plugin</artifactId>
        <version>${avro.version}</version>
    </dependency>
```

```
        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-compiler</artifactId>
            <version>${avro.version}</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.2.6</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.6</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>${avro.version}</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>

    <sourceDirectory>${project.basedir}/src/main/resources/avro</so
urceDirectory>
                            <includes>
                                <include>*.avsc</include>
                            </includes>
```

```
    <outputDirectory>${project.basedir}/src/main/java</outputDirect
ory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

10) Errors

1. LEADER_NOT_AVAILABLE

{test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)

```
[2018-05-15 23:46:40,132] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 14 : {test=LEADER_NOT_AVAILABLE} (org.apac
he.kafka.clients.NetworkClient)
[2018-05-15 23:46:40,266] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 15 : {test=LEADER_NOT_AVAILABLE} (org.apac
he.kafka.clients.NetworkClient)
^C[2018-05-15 23:46:40,394] WARN [Producer clientId=console-producer] Error whil
e fetching metadata with correlation id 16 : {test=LEADER_NOT_AVAILABLE} (org.ap
ache.kafka.clients.NetworkClient)
[root@tos opt]#  {test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkCl
ient)
bash: syntax error near unexpected token `org.apache.kafka.clients.NetworkClient
```

Solutions: /opt/kafka/config/server.properties

Update the following information.

```
# it uses the value for "listeners" if configured.  Otherwise, it will use the v
alue
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://localhost:9092
```

java.util.concurrent.ExecutionException:

org.apache.kafka.common.errors.TimeoutException: Expiring 1 record(s) for my-kafka-topic-6: 30037 ms has passed since batch creation plus linger time

at org.apache.kafka.clients.producer.internals.FutureRecordMetadata.valueOrError(FutureRecordMetadata.java:94)

at org.apache.kafka.clients.producer.internals.FutureRecordMetadata.get(FutureRecordMetadata.java:64)
at org.apache.kafka.clients.producer.internals.FutureRecordMetadata.get(FutureRecordMetadata.java:29)
at com.tos.kafka.MyKafkaProducer.runProducer(MyKafkaProducer.java:97)
at com.tos.kafka.MyKafkaProducer.main(MyKafkaProducer.java:18)
Caused by: org.apache.kafka.common.errors.TimeoutException: Expiring 1 record(s) for my-kafka-topic-6: 30037 ms has passed since batch creation plus linger time.

Solution:

Update the following in all the server properties: /opt/kafka/config/server.properties

```
#    listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://tos.master.com:9093

# Hostname and port the broker will advertise to producers and consumers. If not
 set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the v
alue
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://tos.master.com:9093

# Maps listener names to security protocols, the default is for them to be the s
ame. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_
PLAINTEXT,SASL_SSL:SASL_SSL
```

Its should be updated with your hostname and restart the broker

Changes in the following file, if the hostname is to be changed.

//kafka/ Server.properties and control center

/apps/confluent/etc/confluent-control-center/control-center-dev.properties

/apps/confluent/etc/ksql/ksql-server.properties
/tmp/confluent.8A2Ii7O4/connect/connect.properties

Update localhost to resolve to the ip in /etc/hosts.

In case the hostname doesn't started, updated with ip address and restart the broker.

11) Annexure Code:

2. DumplogSegment

/opt/kafka/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files \
/tmp/kafka-logs/my-kafka-connect-0/oooooooooooooooooo.log | head -n 4

```
[root@tos test-topic-0]# more 00000000000000000000.log
[root@tos test-topic-0]# cd ../
[root@tos kafka-logs]# cd my-kafka-connect-0/
[root@tos my-kafka-connect-0]# ls
00000000000000000000.index      00000000000000000011.snapshot
00000000000000000000.log        leader-epoch-checkpoint
00000000000000000000.timeindex
[root@tos my-kafka-connect-0]# more *log
\▒▒▒▒▒▒▒▒afka Connector.--More--(53%)

[root@tos my-kafka-connect-0]# pwd
/tmp/kafka-logs/my-kafka-connect-0
[root@tos my-kafka-connect-0]# /opt/kafka/bin/kafka-run-class.sh kafka.tools.Dum
pLogSegments --deep-iteration --print-data-log --files \
> /tmp/kafka-logs/my-kafka-connect-0/00000000000000000000.log | head -n 4
Dumping /tmp/kafka-logs/my-kafka-connect-0/00000000000000000000.log
Starting offset: 0
offset: 0 position: 0 CreateTime: 1530552634675 isvalid: true keysize: -1 values
ize: 31 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence:
-1 isTransactional: false headerKeys: [] payload: This Message is from Test File
.
offset: 1 position: 0 CreateTime: 1530552634677 isvalid: true keysize: -1 values
ize: 43 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence:
-1 isTransactional: false headerKeys: [] payload: It will be consumed by the Kaf
ka Connector.
```

3. Data Generator – JSON

Example:

5. If you have not already, go ahead and download the most recent release of the json-data-generator.
6. Unpack the file you downloaded to a directory.

   (tar -xvf json-data-generator-1.4.2-bin.tar -C /opt)

7. Update custom configs into the conf directory

```
{
  "workflows": [{
        "workflowName": "jackieChan",
        "workflowFilename": "jackieChanWorkflow.json"
    }],
  "producers": [{
    "type": "kafka",
    "broker.server": "kafka0",
    "broker.port": 9092,
    "topic": "fight",
    "flatten": false,
    "sync": false

  }]
```

```
}
```

8. Then run the generator like so:

```
java -jar json-data-generator-1.4.2.jar jackieChanSimConfig.json
```

Example Ends Here.

Streaming Json Data Generator

Downloading the generator

You can always find the most recent release over on github where you can download the bundle file that contains the runnable application and example configurations. Head there now and download a release to get started!

Configuration

The generator runs a Simulation which you get to define. The Simulation can specify one or many Workflows that will be run as part of your Simulation. The Workflows then generates Events and these Events are then sent somewhere. You will also need to define Producers that are used to send the Events generated by your Workflows to some destination. These destinations could be a log file, or something more complicated like a Kafka Queue.

You define the configuration for the json-data-generator using two configuration files. The first is a Simulation Config. The Simulation Config defines the Workflows that should be run and different Producers that events should be sent to. The second is a Workflow

configuration (of which you can have multiple). The Workflow defines the frequency of Events and Steps that the Workflow uses to generate the Events. It is the Workflow that defines the format and content of your Events as well.

For our example, we are going to pretend that we have a programmable Jackie Chan robot. We can command Jackie Chan though a programmable interface that happens to take json as an input via a Kafka queue and you can command him to perform different fighting moves in different martial arts styles. A Jackie Chan command might look like this:

```
{
 "timestamp":"2015-05-20T22:05:44.789Z",
 "style":"DRUNKEN_BOXING",
 "action":"PUNCH",
 "weapon":"CHAIR",
 "target":"ARMS",
 "strength":8.3433
}
```

view rawexampleJackieChanCommand.json hosted with by **GitHub**

Now, we want to have some fun with our awesome Jackie Chan robot, so we are going to make him do random moves using our json-data-generator! First we need to define a Simulation Config and then a Workflow that Jackie will use.

SIMULATION CONFIG

Let's take a look at our example Simulation Config:

```
{
   "workflows": [{
        "workflowName": "jackieChan",
        "workflowFilename": "jackieChanWorkflow.json"
     }],
   "producers": [{
     "type": "kafka",
     "broker.server": "192.168.59.103",
     "broker.port": 9092,
     "topic": "jackieChanCommand",
     "flatten": false,
     "sync": false

   }]
}
```

view rawjackieChanSimConfig.json hosted with by **GitHub**

As you can see, there are two main parts to the Simulation Config. The Workflows name and list the workflow configurations you want to use. The Producers are where the Generator will send the events to. At the time of writing this, we have three supported Producers:

- A Logger that sends events to log files
- A Kafka Producer that will send events to your specified Kafka Broker
- A Tranquility Producer that will send events to a Druid cluster.

You can find the full configuration options for each on the github page. We used a Kafka producer because that is how you command our Jackie Chan robot.

WORKFLOW CONFIG

The Simulation Config above specifies that it will use a Workflow called jackieChanWorkflow.json. This is where the meat of your configuration would live. Let's take a look at the example Workflow config and see how we are going to control Jackie Chan:

```
{
  "eventFrequency": 400,
  "varyEventFrequency": true,
  "repeatWorkflow": true,
  "timeBetweenRepeat": 1500,
  "varyRepeatFrequency": true,
  "steps": [{
      "config": [{
          "timestamp": "now()",
          "style": "random('KUNG_FU','WUSHU','DRUNKEN_BOXING')",
          "action": "random('KICK','PUNCH','BLOCK','JUMP')",
          "weapon": "random('BROAD_SWORD','STAFF','CHAIR','ROPE')",
          "target": "random('HEAD','BODY','LEGS','ARMS')",
          "strength": "double(1.0,10.0)"
        }
      ],
```

```
            "duration": 0
        }]
    }
```

view rawjackieChanWorkflow.json hosted with by GitHub

The Workflow defines many things that are all defined on the github page, but here is a summary:

- At the top are the properties that define how often events should be generated and if / when this workflow should be repeated. So this is like saying we want Jackie Chan to do a martial arts move every 400 milliseconds (he's FAST!), then take a break for 1.5 seconds, and do another one.
- Next, are the Steps that this Workflow defines. Each Step has a config and a duration. The duration specifies how long to run this step. The config is where it gets interesting!

WORKFLOW STEP CONFIG

The Step Config is your specific definition of a json event. This can be any kind of json object you want. In our example, we want to generate a Jackie Chan command message that will be sent to his control unit via Kafka. So we define the command message in our config, and since we want this to be fun, we are going to randomly generate what kind of style, move, weapon, and target he will use.

You'll notice that the values for each of the object properties look a bit funny. These are special Functions that we have created that allow us to generate values for each of the properties. For instance, the "random('KICK','PUNCH','BLOCK','JUMP')" function will

randomly choose one of the values and output it as the value of the "action" property in the command message. The "now()" function will output the current date in an ISO8601 date formatted string. The "double(1.0,10.0)" will generate a random double between 1 and 10 to determine the strength of the action that Jackie Chan will perform. If we wanted to, we could make Jackie Chan perform combo moves by defining a number of Steps that will be executed in order.

There are many more Functions available in the generator with everything from random string generation, counters, random number generation, dates, and even support for randomly generating arrays of data. We also support the ability to reference other randomly generated values. For more info, please check out the full documentation on the github page.

Once we have defined the Workflow, we can run it using the json-data-generator. To do this, do the following:

9. If you have not already, go ahead and download the most recent release of the json-data-generator.
10. Unpack the file you downloaded to a directory.

    (tar -xvf json-data-generator-1.4.2-bin.tar -C /apps)

11. Copy your custom configs into the conf directory
12. Then run the generator like so:
    1. java -jar json-data-generator-1.4.0.jar jackieChanSimConfig.json

You will see logging in your console showing the events as they are being generated. The jackieChanSimConfig.json generates events like these:

{"timestamp":"2015-05-20T22:21:18.036Z","style":"WUSHU","action":"BLOCK","weapon":"CHAIR","target":"BODY","strength":4.7912}
{"timestamp":"2015-05-20T22:21:19.247Z","style":"DRUNKEN_BOXING","action":"PUNCH","weapon":"BROAD_SWORD","target":"ARMS","strength":3.0248}
{"timestamp":"2015-05-20T22:21:20.947Z","style":"DRUNKEN_BOXING","action":"BLOCK","weapon":"ROPE","target":"HEAD","strength":6.7571}
{"timestamp":"2015-05-20T22:21:22.715Z","style":"WUSHU","action":"KICK","weapon":"BROAD_SWORD","target":"ARMS","strength":9.2062}
{"timestamp":"2015-05-20T22:21:23.852Z","style":"KUNG_FU","action":"PUNCH","weapon":"BROAD_SWORD","target":"HEAD","strength":4.6202}
{"timestamp":"2015-05-20T22:21:25.195Z","style":"KUNG_FU","action":"JUMP","weapon":"ROPE","target":"ARMS","strength":7.5303}
{"timestamp":"2015-05-20T22:21:26.492Z","style":"DRUNKEN_BOXING","action":"PUNCH","weapon":"STAFF","target":"HEAD","strength":1.1247}

{"timestamp":"2015-05-20T22:21:28.042Z","style":"WUSHU","action":"BLOCK","weapon":"STAFF","target":"ARMS","strength":5.5976}
{"timestamp":"2015-05-20T22:21:29.422Z","style":"KUNG_FU","action":"BLOCK","weapon":"ROPE","target":"ARMS","strength":2.152}
{"timestamp":"2015-05-20T22:21:30.782Z","style":"DRUNKEN_BOXING","action":"BLOCK","weapon":"STAFF","target":"ARMS","strength":6.2686}
{"timestamp":"2015-05-20T22:21:32.128Z","style":"KUNG_FU","action":"KICK","weapon":"BROAD_SWORD","target":"BODY","strength":2.3534}

If you specified to repeat your Workflow, then the generator will continue to output events and send them to your Producer simulating a real world client, or in our case, continue to make Jackie Chan show off his awesome skills. If you also had a Chuck Norris robot, you could add another Workflow config to your Simulation and have the two robots fight it out! Just another example of how you can use the generator to simulate real world situations.

12) Pom.xml (Standalone)

```
<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.hp.tos</groupId>
    <artifactId>LearningKafka</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>All About Kafka</description>
    <properties>
        <algebird.version>0.13.4</algebird.version>
        <avro.version>1.8.2</avro.version>
        <avro.version>1.8.2</avro.version>
        <confluent.version>5.3.0</confluent.version>
        <kafka.version>3.0.0</kafka.version>
        <kafka.scala.version>2.11</kafka.scala.version>
    </properties>
    <repositories>
        <repository>
            <id>confluent</id>
            <url>https://packages.confluent.io/maven/</url>
        </repository>
    </repositories>
```

```
<pluginRepositories>
    <pluginRepository>
        <id>confluent</id>
        <url>https://packages.confluent.io/maven/</url>
    </pluginRepository>
</pluginRepositories>
<dependencies>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-streams-avro-serde</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-schema-registry-client</artifactId>
        <version>${confluent.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
```

```
        <version>${kafka.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>${kafka.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams-test-utils</artifactId>
        <version>${kafka.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro</artifactId>
        <version>${avro.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro-maven-plugin</artifactId>
        <version>${avro.version}</version>
    </dependency>
    <dependency>
```

```
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-compiler</artifactId>
            <version>${avro.version}</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.2.6</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.6</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
```

```
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>${avro.version}</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>

    <sourceDirectory>${project.basedir}/src/main/resources/avro</so
urceDirectory>
                            <includes>
                                <include>*.avsc</include>
                            </includes>

    <outputDirectory>${project.basedir}/src/main/java</outputDirect
ory>
```

```
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

https://developer.ibm.com/hadoop/2017/04/10/kafka-security-mechanism-saslplain/
https://sharebigdata.wordpress.com/2018/01/21/implementing-sasl-plain/
https://developer.ibm.com/code/howtos/kafka-authn-authz
https://github.com/confluentinc/kafka-streams-examples/tree/4.1.x/
https://github.com/spring-cloud/spring-cloud-stream-samples/blob/master/kafka-streams-samples/kafka-streams-table-join/src/main/java/kafka/streams/table/join/KafkaStreamsTableJoin.java
https://docs.confluent.io/current/ksql/docs/tutorials/examples.html#ksql-examples
https://developer.confluent.io/learn-kafka/kafka-streams/get-started/