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

Basic Stateless Stream Transformation – 60 Minutes

Scenario:
Fetch fight records from input topic: f**ight**
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 the required classes as shown below

Stream API.

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);

    }

}
```

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();
    }

}
```

Generate some messages.

Open a terminal and run the data generator tool.

#java -jar json-data-generator-1.4.0.jar jackieChanSimConfig.json

```
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

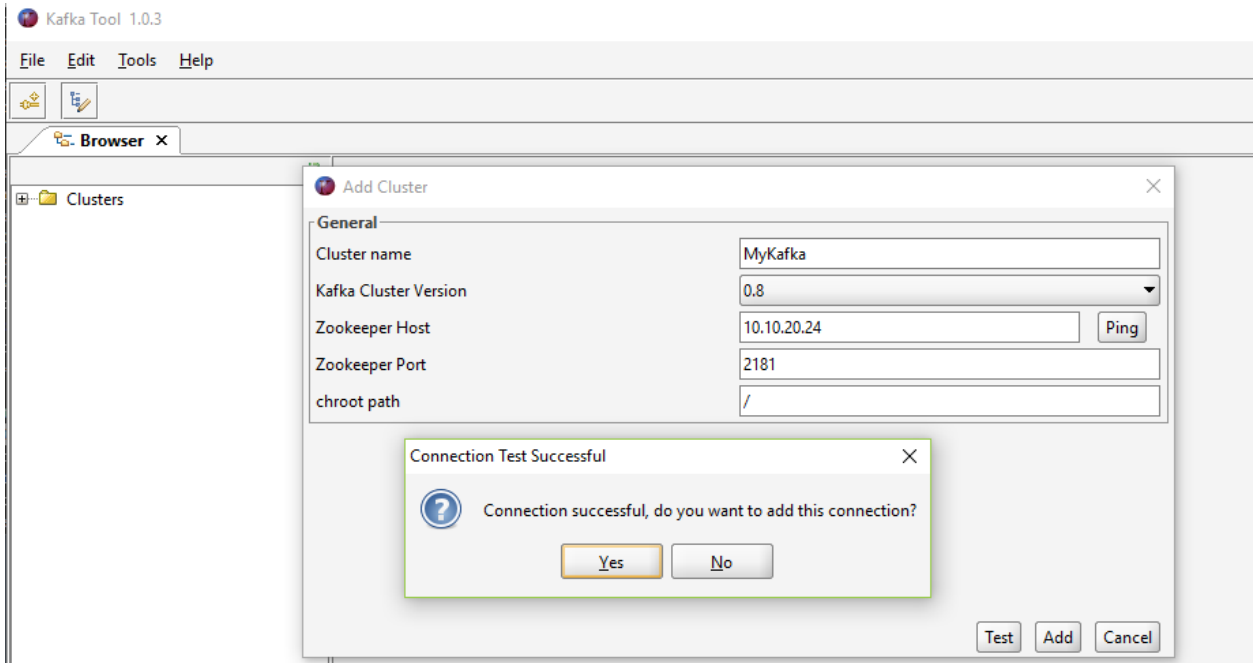
Verify the messages in the Output topic.

#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 ----------

Kafkatools


Kafka Tool 1.0.3
File Edit Tools Help
Browser
Clusters
Add Cluster
General
Cluster name
MyKafka
Kafka Cluster Version
0.8
Zookeeper Host
10.10.20.24
Ping
Zookeeper Port
2181
chroot path
/
Connection Test Successful
Connection successful, do you want to add this connection?
Yes
No
Test
Add
Cancel

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>
```

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.

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:

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

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

3. 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

   }]
```

```
}
```

4. 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:

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 /apps)

7. Copy your custom configs into the conf directory
8. 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.

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/