

1.	Prerequisite	3
2.	Install KSQL DB – 60 Minutes(D)	4
4.	Installing Confluent Kafka (Local) – 60 Minutes	10
5.	Workflow using KSQL - CLI – 90 Minutes(D)	26
6.	Errors	46
	I. LEADER_NOT_AVAILABLE	46
	java.util.concurrent.ExecutionException:	46
7.	Annexure Code:	50
	II. DumplogSegment	50
	III. Data Generator – JSON	52
	IV. Resources	62

First Round Done – Need some find tuning

Date : 23th Oct 2022.

<https://docs.confluent.io/current/ksql/docs/tutorials/examples.html#ksql-examples>

1. Prerequisite

JDK : 1.11

Confluent : 7.3.3

Kafka : 3.11

Registry & KSQL DB

2. Install KSQL DB – 60 Minutes(D)

Prerequisite: Kafka Node installation.

And kafka registry, required for Avro integration.

Update kafka server.properties with the following entries.

```
#vi /opt/kafka/config/server.properties
```

```
transaction.state.log.replication.factor=1
```

```
transaction.state.log.min.isr=1
```

```
offsets.topic.replication.factor=1
```

Restart the kafka broker.

Get standalone ksqlDB

Since ksqlDB runs natively on Apache Kafka®, you'll need to have a Kafka cluster that ksqlDB is configured to use. Use the steps to the right to install the latest release of ksqlDB.

```
# Download the archive and its signature
```

```
curl http://ksqldb-packages.s3.amazonaws.com/archive/0.23/confluent-ksqldb-0.23.1.tar.gz --output  
confluent-ksqldb-0.23.1.tar.gz
```

5 Kafka – Dev Ops

```
# Extract the tarball to the directory of your choice
```

```
#tar -xf confluent-ksqldb-0.23.1.tar.gz -C /opt/  
#mv confluent-ksq* ksqldb
```

Configure ksqldb server

Ensure your ksqldb server has network connectivity to Kafka.

Edit the highlighted line in `/opt/ksqldb/etc/ksqldb/ksql-server.properties` to match your Kafka hostname and port.

```
#----- Kafka -----
```

```
# The set of Kafka brokers to bootstrap Kafka cluster information from:  
bootstrap.servers=kafka0:9092
```

```
# Enable snappy compression for the Kafka producers  
compression.type=snappy
```

To enable Schema Registry Add the following line at the end of the configuration file.

```
#----- Schema Registry -----
```

Uncomment and complete the following to enable KSQL's integration to the Confluent Schema Registry:

```
ksql.schema.registry.url=http://kafka:8081
```

Start ksqlDB's server

ksqlDB is packaged with a startup script for development use. We'll use that here.

When you're ready to run it as a service, you'll want to manage ksqlDB with something like `systemd`.

```
#!/opt/ksqldb/bin/ksql-server-start /opt/ksqldb/etc/ksqldb/ksql-server.properties
```

if any issue in start up because of jar.

Download and store in the following folder.

```
#cd /opt/ksqldb/share/java/ksqldb
```

```
#wget https://repo1.maven.org/maven2/io/netty/netty-all/4.1.30.Final/netty-all-4.1.30.Final.jar
```

```
[2022-02-15 16:17:02,735] INFO ksqlDB API server listening on http://0.0.0.0:8088 (io.confluent.ksql.rest.server.KsqlRestApplication:405)
```

```
=====
```

```
=                                     =
```

```
=      _   -       _    _     _   )      =
```

```
=      | / / _/_/_\ \ | | _ \  _ \      =
```

```
=      | < \ \ C | | | | | | |_) |      =
```

```
=      |_/\_/_/_/,|_|_|_|_|_|_/        =
```

```
=              |_|                        =
```

```
=          The Database purpose-built     =
```

```
=          for stream processing apps     =
```

```
=====
```

Copyright 2017-2021 Confluent Inc.

Server 0.23.1 listening on http://0.0.0.0:8088

To access the KSQL CLI, run:

ksql http://0.0.0.0:8088

```
[2022-02-15 16:17:02,813] INFO Server up and running (io.confluent.ksql.rest.server.KsqlServerMain:92)
[2022-02-15 16:17:07,390] INFO Successfully submitted metrics to Confluent via secure endpoint (io.confluent.support.metrics.submitters.ConfluentSubmitter:146)
```

Start ksqlDB's interactive CLI

ksqlDB runs as a server which clients connect to in order to issue queries.

Run this command to connect to the ksqlDB server and enter an interactive command-line interface (CLI) session.

```
#/opt/ksqldb/bin/ksql http://0.0.0.0:8088
```

```
[root@kafka0 ksqldb]# /opt/ksqldb/bin/ksql http://0.0.0.0:8088
```

[illegible]

Copyright 2017-2021 Confluent Inc.

CLI v0.23.1, Server v0.23.1 located at <http://0.0.0.0:8088>

Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>

```
#show topics;
```



```
ksql> show topics;
```

Kafka Topic	Partitions	Partition Replicas
default_ksql_processing_log	1	1
test	1	1
topic1	2	1

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

4. Installing Confluent Kafka (Local) – 60 Minutes

Demonstrates both the basic and most powerful capabilities of Confluent Platform, including using Control Center for topic management and event stream processing using KSQL. In this quick start you create Apache Kafka® topics, use Kafka Connect to generate mock data to those topics, and create KSQL streaming queries on those topics. You then go to Control Center to monitor and analyze the streaming queries.

You need to install java before installing zookeeper and Kafka.

Installing Java

```
#tar -xvf jdk-8u45-linux-x64.tar.gz -C /opt
```

Set in the path variable and JAVA_HOME

[ex:

```
export JAVA_HOME=/opt/jdk
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

]

Include in the profile as follow

```
[root@tos opt]# more ~/.bashrc
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

export JAVA_HOME=/opt/jdk1.8.0_45

export PATH=$PATH:$JAVA_HOME/bin
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
[root@tos opt]#
```

Installing a Kafka Broker

The following example installs Confluence Kafka in /apps.

Installing and Configuring Confluent CLI

Inflate the confluent kafka compress file as shown below:

```
#tar -xvf confluent-5.5.1-2.12.tar -C /apps
```

Rename the folder.

```
#mv /apps/confluent* /apps/confluent
```

Set the environment variable for the Confluent Platform directory (<path-to-confluent>).

```
export CONFLUENT_HOME=/apps/confluent
```

```
(base) [root@tos confluent]# pwd
/apps/confluent
(base) [root@tos confluent]# ls
bin confluent etc legal lib logs README share src
(base) [root@tos confluent]#
```

Set your PATH variable:

vi ~/.bashrc

```
export PATH=/apps/confluent/bin:${PATH};
```

```
if [ -f "/apps/anaconda3/etc/profile.d/conda.sh" ]; then
    . "/apps/anaconda3/etc/profile.d/conda.sh"
else
    export PATH="/apps/anaconda3/bin:${PATH}"
fi
fi
unset __conda_setup
# <<< conda initialize <<<

export JAVA_HOME=/apps/jdk
export PATH=${JAVA_HOME}/bin:${PATH}:${SCALA_HOME}/bin

export PATH=/apps/confluent/bin:${PATH};
-- INSERT --
```

After decompressing the file. You should have the following directories:

```
(base) [root@tos confluent]#  
(base) [root@tos confluent]# pwd  
/apps/confluent  
(base) [root@tos confluent]# ls -ltr  
total 16  
drwxr-xr-x.  3 life life   21 Jun  5 10:11 lib  
drwxr-xr-x.  7 life life  106 Jun  5 10:42 share  
drwxr-xr-x. 23 life life 4096 Jun  5 10:42 etc  
drwxr-xr-x.  3 life life 4096 Jun  5 10:42 bin  
drwxr-xr-x.  2 life life  178 Jun  5 11:17 src  
-rw-r--r--.  1 life life  871 Jun  5 11:17 README  
drwxr-xr-x.  2 root root 4096 Jul  7 02:01 logs  
(base) [root@tos confluent]#
```

Install the [Kafka Connect Datagen](#) source connector using the Confluent Hub client. This connector generates mock data for demonstration purposes and is not suitable for production. [Confluent Hub](#) is an online library of pre-packaged and ready-to-install extensions or add-ons for Confluent Platform and Kafka.

```
#confluent-hub install --no-prompt confluentinc/kafka-connect-datagen:latest
```

```
(base) [root@tos ~]# cd /apps
(base) [root@tos apps]# confluent-hub install --no-prompt confluentinc/kafka-connect-datagen:latest
Running in a "--no-prompt" mode
Implicit acceptance of the license below:
Apache License 2.0
https://www.apache.org/licenses/LICENSE-2.0
Downloading component Kafka Connect Datagen 0.1.3, provided by Confluent, Inc. from Confluent Hub and installing into /apps/confluent/share/confluent-hub-components
Adding installation directory to plugin path in the following files:
/apps/confluent/etc/kafka/connect-distributed.properties
/apps/confluent/etc/kafka/connect-standalone.properties
/apps/confluent/etc/schema-registry/connect-avro-distributed.properties
/apps/confluent/etc/schema-registry/connect-avro-standalone.properties
/tmp/confluent.8A2Ii704/connect/connect.properties

Completed
(base) [root@tos apps]#
```

Start Confluent Platform using the Confluent CLI `confluent local start` command. This command starts all of the Confluent Platform components; including Kafka, ZooKeeper, Schema Registry, HTTP REST Proxy for Kafka, Kafka Connect, KSQL, and Control Center.

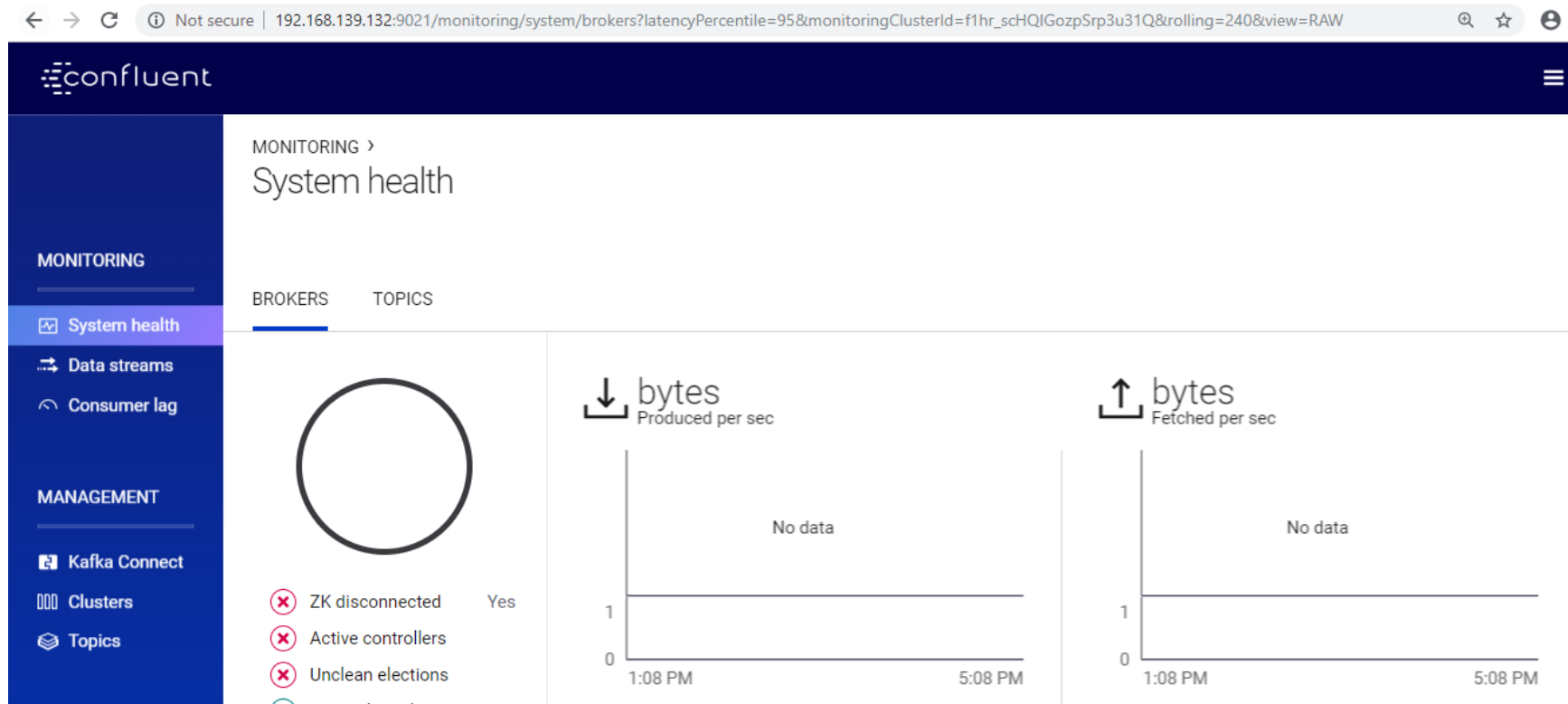
```
#export CONFLUENT_CURRENT=/opt/data/ckafka
```

```
#confluent local services start
```

```
(base) [root@tos bin]# confluent start
This CLI is intended for development only, not for production
https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /tmp/confluent.8A2Ii7O4
Starting zookeeper
zookeeper is [UP]
Starting kafka
kafka is [UP]
Starting schema-registry
schema-registry is [UP]
Starting kafka-rest
kafka-rest is [UP]
Starting connect
connect is [UP]
Starting ksql-server
ksql-server is [UP]
Starting control-center
control-center is [UP]
(base) [root@tos bin]#
```

Navigate to the Control Center web interface at <http://localhost:9021/>.



Install a Kafka Connector and Generate Sample Data

In this step, you use Kafka Connect to run a demo source connector called kafka-connect-datagen that creates sample data for the Kafka topics pageviews and users.

Run one instance of the [Kafka Connect Datagen](#) connector to produce Kafka data to the `pageviews` topic in AVRO format.

Management → **Add connector**. Or Connectors → Add Connector

Find the DatagenConnector tile and click **Connect**.

Name the connector `datagen-pageviews`. After naming the connector, new fields appear. Scroll down and specify the following configuration values:

- Tasks max : 1
- In the **Key converter class** field, type `org.apache.kafka.connect.storage.StringConverter`.
- In the **kafka.topic** field, type `pageviews`.
- In the **max.interval** field, type `100`.
- In the **iterations** field, type `1000000000`.
- In the **quickstart** field, type `pageviews`.
-

1. Click **Continue**.

2. Review the connector configuration and click **Launch**.

MANAGEMENT ›

Kafka Connect

Bring data in

Send data out

Search connectors

+ Add connector

Connectors			Details
Status	Name		Active tasks
Running	datagen-pag...	...	1

Run another instance of the [Kafka Connect Datagen](#) connector to produce Kafka data to the `users` topic in AVRO format.

Click **Add connector**.

Find the DatagenConnector tile and click **Connect**.

Name the connector `datagen-users`. After naming the connector, new fields appear. Scroll down and specify the following configuration values:

- Max Task : 1
- In the **Key converter class** field, type `org.apache.kafka.connect.storage.StringConverter`.
- In the **kafka.topic** field, type `users`.
- In the **max.interval** field, type `1000`.
- In the **iterations** field, type `1000000000`.
- In the **quickstart** field, type `users`.
 - Click **Continue**.
 - Review the connector configuration and click **Launch**.

At the end of this.

Kafka Connect

[Bring data in](#)[Send data out](#)[+ Add connector](#)

Connectors

Status

Name

Details

Active tasks

Running

datagen-pag...

...

1

Running

datagen-users

...

1

Verify the messages in the both the topics:

Using the control centers:

Topics -> pageviews -> Messages:

The screenshot displays the Apache Kafka Control Center interface for the 'pageviews' topic. The left sidebar contains navigation links: Cluster overview, Brokers, Topics (selected), Connect, ksqldb, Consumers, Replicators, Cluster settings, and Health+ (with a 'New' badge). The main panel shows the 'Messages' tab for the 'pageviews' topic. It includes tabs for Overview, Messages, Schema, and Configuration. The 'Messages' tab displays the following information:

- Producers:** Bytes in/sec: 2.68K
- Consumers:** Bytes out/sec: 522
- Message fields:**
 - topic
 - partition
 - offset
 - timestamp

Two message records are shown, each with a 'Newest' button:

- Record 1: `{"viewtime":70861,"userid":"User_8","pageid":"Page_12"}`
Partition: 0 Offset: 7086 Timestamp: 1641706995703
- Record 2: `{"viewtime":70851,"userid":"User_5","pageid":"Page_35"}`
Partition: 0 Offset: 7085 Timestamp: 1641706995632

Topics -> Users -> Messages:

Cluster overview

Brokers

Topics

Connect

ksqlDB

Consumers

Replicators

Cluster settings

Health+ New

Overview Messages Schema Configuration

Producers

Bytes in/sec 295

Consumers

Bytes out/sec 5

Message fields

- topic
- partition
- offset
- timestamp
- timestampType
- ✓ headers
- key

Filter by keyword

Jump to offset

offset

+ Produce a new message to this topic

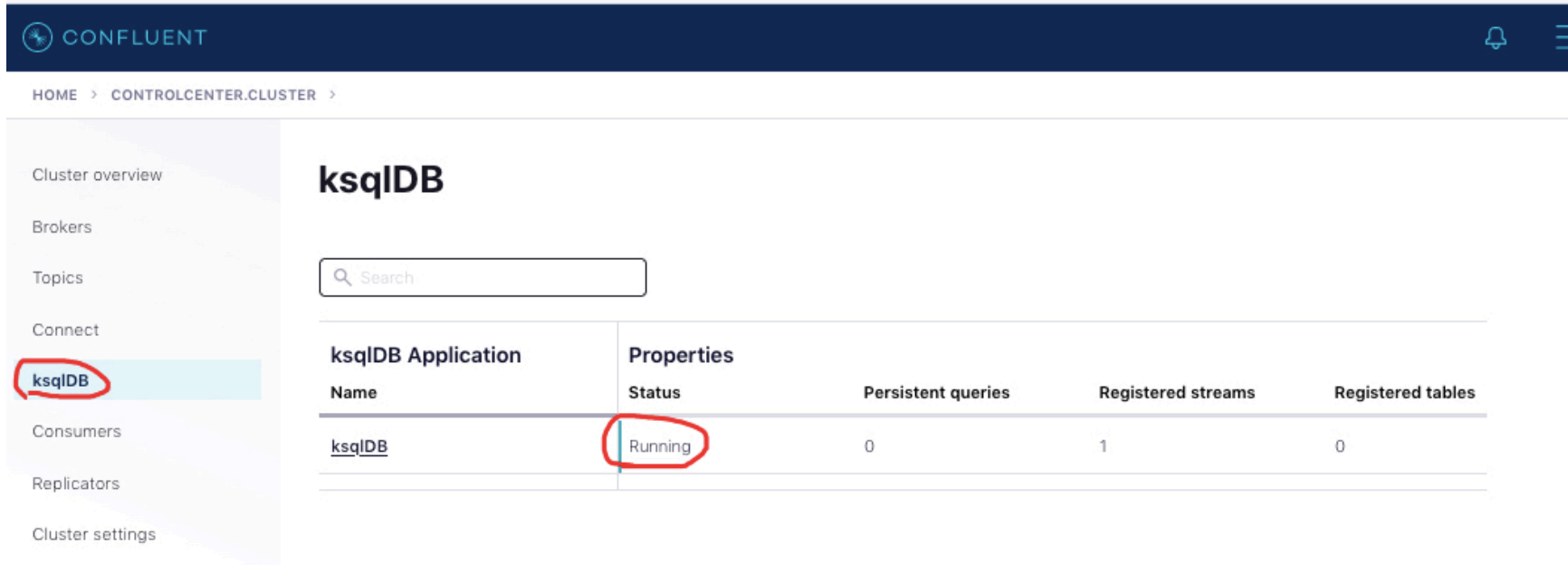
Newest

✓ {"registertime":1505601918721,"userid":"User_8","regionid":"Region_8","gender":"OTHER"}
Partition: 0 Offset: 508 Timestamp: 1641707060914

✓ {"registertime":1512168635241,"userid":"User_2","regionid":"Region_6","gender":"FEMALE"}
Partition: 0 Offset: 507 Timestamp: 1641707060763

✓ {"registertime":1487807651768,"userid":"User_7","regionid":"Region_2","gender":"MALE"}
Partition: 0 Offset: 506 Timestamp: 1641707059920

Ensure that ksql DB services is up.



The screenshot shows the Confluent Control Center interface. On the left, a sidebar menu lists various cluster management options, with 'ksqlDB' highlighted and circled in red. The main panel displays the 'ksqlDB' application details. A search bar is present above a table. The table has columns for 'ksqlDB Application Name', 'Properties Status', 'Persistent queries', 'Registered streams', and 'Registered tables'. The 'ksqlDB' application is listed with a status of 'Running', which is also circled in red. The other metrics are 0 persistent queries, 1 registered stream, and 0 registered tables.

ksqlDB Application Name	Properties Status	Persistent queries	Registered streams	Registered tables
ksqlDB	Running	0	1	0

If there is any issue, verify the status and configuration as shown below:

```
#confluent local services status
```

```
[root@ckafka0 ckafka]# confluent local services status
The local commands are intended for a single-node development environment only,
NOT for production usage. https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /opt/data/ckafka/confluent.652875
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
[root@ckafka0 ckafka]#
```

If unable to connect in 8088 port. Verify that the KSQL listeners IP and port are specify correctly in the configuration files.

/apps/confluent/etc/ksqldb/ksql-server.properties

listeners=http://localhost:8088 or

listeners=http://0.0.0.0:8088

Restart after any modification.

confluent local services ksql-server status

confluent local services ksql-server stop

confluent local services ksql-server start

confluent local services ksql-server status

After that verify the listening port.

lsof -i:8088

```
[root@ckafka0 ckafka]# lsof -i:8088
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
java	1092	root	628u	IPv4	140454	0t0	TCP	localhost:37410->localhost:radan-http (ESTABLISHED)
java	1092	root	634u	IPv4	140457	0t0	TCP	localhost:37414->localhost:radan-http (ESTABLISHED)
java	1092	root	637u	IPv4	145818	0t0	TCP	localhost:37430->localhost:radan-http (ESTABLISHED)
java	1092	root	638u	IPv4	144459	0t0	TCP	localhost:37432->localhost:radan-http (ESTABLISHED)
java	2968	root	502u	IPv4	143524	0t0	TCP	localhost:radan-http (LISTEN)
java	2968	root	506u	IPv4	143555	0t0	TCP	localhost:radan-http->localhost:37430 (ESTABLISHED)
java	2968	root	507u	IPv4	143556	0t0	TCP	localhost:radan-http->localhost:37432 (ESTABLISHED)
java	2968	root	511u	IPv4	143551	0t0	TCP	localhost:radan-http->localhost:37410 (ESTABLISHED)
java	2968	root	512u	IPv4	143552	0t0	TCP	localhost:radan-http->localhost:37414 (ESTABLISHED)

It means, the KSQL server is running.

-----Lab Installation completes End here. -----

5. Workflow using KSQL - CLI – 90 Minutes(D)

Following features will be demonstrated.

- Create Topics and Produce Data
- Create and produce data to the Kafka topics pageviews and users.
- Inspect Kafka Topics by Using SHOW and PRINT Statements
- Create a Stream and Table
- Write Queries

This tutorial demonstrates a simple workflow using KSQL to write streaming queries against messages in Kafka.

To get started, you must start a Kafka cluster, including ZooKeeper and a Kafka broker. KSQL will then query messages from this Kafka cluster. KSQL is installed in the Confluent Platform by default.

Create Topics and Produce Data

Create and produce data to the Kafka topics `pageviews` and `users`. These steps use the KSQL datagen that is included Confluent Platform.

1. Create the `pageviews` topic and produce data using the data generator. The following example continuously generates data with a value in DELIMITED format.

ksql-datagen bootstrap-server=kafkao:9092 quickstart=pageviews format=json topic=pageviews maxInterval=500

```
(base) [root@tos ~]#
(base) [root@tos ~]#
(base) [root@tos ~]# ksql-datagen quickstart=pageviews format=delimited topic=pa
geviews maxInterval=500
[2019-07-31 21:35:34,823] INFO AvroDataConfig values:
    schemas.cache.config = 1
    enhanced.avro.schema.support = false
    connect.meta.data = true
(io.confluent.connect.avro.AvroDataConfig:179)
1 --> ([ 1564589135082 | 'User_3' | 'Page_97' ]) ts:1564589135333
11 --> ([ 1564589135590 | 'User_7' | 'Page_66' ]) ts:1564589135591
21 --> ([ 1564589135857 | 'User_1' | 'Page_34' ]) ts:1564589135861
31 --> ([ 1564589135959 | 'User_6' | 'Page_37' ]) ts:1564589135959
41 --> ([ 1564589136036 | 'User_6' | 'Page_66' ]) ts:1564589136036
51 --> ([ 1564589136428 | 'User_2' | 'Page_98' ]) ts:1564589136428
61 --> ([ 1564589136761 | 'User_9' | 'Page_26' ]) ts:1564589136761
```

2. Produce Kafka data to the `users` topic using the data generator. The following example continuously generates data with a value in JSON format.

```
$ ksql-datagen bootstrap-server=kafka0:9092 quickstart=users format=json topic=users
maxInterval=100
```

Tip

You can also produce Kafka data using the `kafka-console-producer` CLI provided with Confluent Platform.

```
(base) [root@tos ~]#
(base) [root@tos ~]# ksql-datagen quickstart=pageviews format=delimited topic=pa
geviews maxInterval=500
[2019-07-31 21:35:34,823] INFO AvroDataConfig values:
    schemas.cache.config = 1
    enhanced.avro.schema.support = false
    connect.meta.data = true
(io.confluent.connect.avro.AvroDataConfig:179)
1 --> ([ 1564589135082 | 'User_3' | 'Page_97' ]) ts:1564589135333
11 --> ([ 1564589135590 | 'User_7' | 'Page_66' ]) ts:1564589135591
21 --> ([ 1564589135857 | 'User_1' | 'Page_34' ]) ts:1564589135861
31 --> ([ 1564589135959 | 'User_6' | 'Page_37' ]) ts:1564589135959
41 --> ([ 1564589136036 | 'User_6' | 'Page_66' ]) ts:1564589136036
51 --> ([ 1564589136428 | 'User_2' | 'Page_98' ]) ts:1564589136428
61 --> ([ 1564589136761 | 'User_9' | 'Page_26' ]) ts:1564589136761
```

Launch the KSQL CLI

To launch the CLI, run the following command. It will route the CLI logs to the `./ksql_logs` directory, relative to your current directory. By default, the CLI will look for a KSQL Server running at `http://localhost:8088`.

```
$ LOG_DIR=./ksql_logs ksql
```

Important

By default KSQL attempts to store its logs in a directory called `logs` that is relative to the location of the `ksql` executable. For example, if `ksql` is installed at `/usr/local/bin/ksql`, then it would attempt to store its logs in `/usr/local/logs`. If you are running `ksql` from the default Confluent Platform location, `<path-to-confluent>/bin`, you must override this default behavior by using the `LOG_DIR` variable.

After KSQL is started, your terminal should resemble this.

```
(base) [root@tos apps]# LOG_DIR=./ksql_logs ksql
```

[illegible]

Copyright 2017-2018 Confluent Inc.

```
CLI v5.2.2, Server v5.2.2 located at http://localhost:8088
```

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>

Inspect Kafka Topics By Using SHOW and PRINT Statements

KSQL enables inspecting Kafka topics and messages in real time.

- Use the SHOW TOPICS statement to list the available topics in the Kafka cluster.
- Use the PRINT statement to see a topic's messages as they arrive.

In the KSQL CLI, run the following statement:

SHOW TOPICS;

Your output should resemble:

Kafka Topic	Registered	Partitions	Partition Replicas	Consumers	ConsumerGroups
_____	_____	_____	_____	_____	_____
_confluent-metrics	false	12	1	0	0
_schemas	false	1	1	0	0
pageviews	false	1	1	0	0
users	false	1	1	0	0
_____	_____	_____	_____	_____	_____

Inspect the **users** topic by using the PRINT statement:

PRINT 'users';

Your output should resemble:

Format:JSON

```
{ "ROWTIME":1540254230041,"ROWKEY":"User_1","registertime":1516754966866,"userid":"User_1","regionid":"Region_9","gender":"MALE"}
{ "ROWTIME":1540254230081,"ROWKEY":"User_3","registertime":1491558386780,"userid":"User_3","regionid":"Region_2","gender":"MALE"}
{ "ROWTIME":1540254230091,"ROWKEY":"User_7","registertime":1514374073235,"userid":"User_7","regionid":"Region_2","gender":"OTHER"}
^C{ "ROWTIME":1540254232442,"ROWKEY":"User_4","registertime":1510034151376,"userid":"User_4","regionid":"Region_8","gender":"FEMALE"}
```

Topic printing ceased

Press CTRL+C to stop printing messages.

Inspect the `pageviews` topic by using the PRINT statement:

PRINT 'pageviews';

Your output should resemble:

Format:STRING

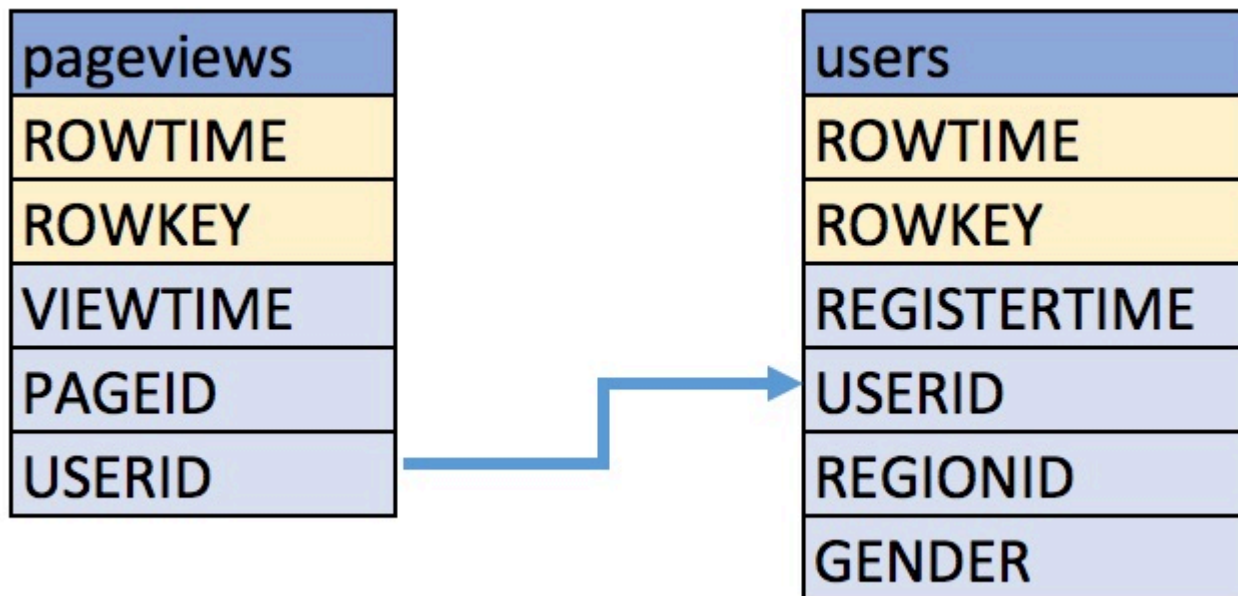
```
10/23/18 12:24:03 AM UTC , 9461 , 1540254243183,User_9,Page_20
10/23/18 12:24:03 AM UTC , 9471 , 1540254243617,User_7,Page_47
10/23/18 12:24:03 AM UTC , 9481 , 1540254243888,User_4,Page_27
```

```
^C10/23/18 12:24:05 AM UTC , 9521 , 1540254245161,User_9,Page_62
Topic printing ceased
ksql>
```

Press CTRL+C to stop printing messages.

Create a Stream and Table

These examples query messages from Kafka topics called `pageviews` and `users` using the following schemas:



1. Create a stream, named `pageviews_original`, from the `pageviews` Kafka topic, specifying the `value_format` of `DELIMITED`.

CREATE STREAM pageviews_original (viewtime bigint, userid varchar, pageid varchar) WITH (kafka_topic='pageviews', value_format='JSON');

Your output should resemble:

```
-----  
ksql> CREATE STREAM pageviews_original (viewtime bigint, userid varchar, pageid varchar) WITH  
>(kafka_topic='pageviews', value_format='DELIMITED');  
>  
  
Message  
-----  
Stream created  
-----  
ksql> █
```

Tip

You can run `DESCRIBE pageviews_original;` to see the schema for the stream. Notice that KSQL created two additional columns, named `ROWTIME`, which corresponds with the Kafka message timestamp, and `ROWKEY`, which corresponds with the Kafka message key.

```
ksql> DESCRIBE pageviews_original;

Name                               : PAGEVIEWS_ORIGINAL
Field    | Type
-----
ROWTIME   | BIGINT          (system)
ROWKEY    | VARCHAR (STRING) (system)
VIEWTIME  | BIGINT
USERID    | VARCHAR (STRING)
PAGEID    | VARCHAR (STRING)
-----
For runtime statistics and query details run: DESCRIBE EXTENDED <Stream,Table>;
ksql>
```

2. Create a table, named `users_original`, from the `users` Kafka topic, specifying the `value_format` of `JSON`.

CREATE TABLE users_original (registertime BIGINT, gender VARCHAR, regionid VARCHAR, userid VARCHAR PRIMARY KEY) WITH (kafka_topic='users', value_format='JSON');

Your output should resemble:

```
Message
-----
Table created
-----
```

Tip

You can run `DESCRIBE users_original;` to see the schema for the Table.

3. Optional: Show all streams and tables.

```
ksql> SHOW STREAMS;
```

Stream Name	Kafka Topic	Format
PAGEVIEWS_ORIGINAL	pageviews	DELIMITED

```
ksql> SHOW TABLES;
```

Table Name	Kafka Topic	Format	Windowed
USERS_ORIGINAL	users	JSON	false

Write Queries

```
SET 'auto.offset.reset'='earliest';
```

These examples write queries using KSQL.

Note: By default KSQL reads the topics for streams and tables from the latest offset.

1. Use **SELECT** to create a query that returns data from a STREAM. This query includes the **LIMIT** keyword to limit the number of rows returned in the query result. Note that exact data output may vary because of the randomness of the data generation.

SELECT pageid FROM pageviews_original EMIT changes LIMIT 3;

Your output should resemble:

```
Page_24
Page_73
Page_78
LIMIT reached
Query terminated
```

2. Create a persistent query by using the **CREATE STREAM** keywords to precede the **SELECT** statement. The results from this query are written to the **PAGEVIEWS_ENRICHED** Kafka topic. The following query enriches the **pageviews_original** STREAM by doing a **LEFT JOIN** with the **users_original** TABLE on the user ID.

```
CREATE STREAM pageviews_enriched AS
SELECT users_original.userid AS userid, pageid, regionid, gender
FROM pageviews_original
```

```
JOIN users_original  
ON pageviews_original.userid = users_original.userid  
Emit changes;
```

Your output should resemble:

```
Message  
-----  
Stream created and running  
-----
```

Tip

You can run `DESCRIBE pageviews_enriched;` to describe the stream.

3. Use `SELECT` to view query results as they come in. To stop viewing the query results, press `<ctrl-c>`. This stops printing to the console but it does not terminate the actual query. The query continues to run in the underlying KSQL application.

```
SELECT * FROM pageviews_enriched Emit Changes;
```

Your output should resemble:

IUser_9	IPage_92	IRegion_2	IMALE	
IUser_2	IPage_66	IRegion_6	IMALE	
IUser_3	IPage_10	IRegion_7	IMALE	
IUser_5	IPage_30	IRegion_3	IOTHER	
IUser_2	IPage_85	IRegion_6	IMALE	
IUser_1	IPage_46	IRegion_7	IOTHER	
IUser_6	IPage_56	IRegion_3	IFEMALE	
IUser_8	IPage_13	IRegion_2	IMALE	
IUser_4	IPage_19	IRegion_4	IFEMALE	
IUser_3	IPage_44	IRegion_7	IMALE	
IUser_8	IPage_57	IRegion_2	IMALE	
IUser_8	IPage_39	IRegion_2	IMALE	
IUser_9	IPage_15	IRegion_2	IMALE	
IUser_9	IPage_71	IRegion_2	IMALE	
IUser_7	IPage_69	IRegion_8	IMALE	

4. Create a new persistent query where a condition limits the streams content, using **WHERE**. Results from this query are written to a Kafka topic called **PAGEVIEWS_FEMALE**.

```
CREATE STREAM pageviews_female AS
SELECT * FROM pageviews_enriched
WHERE gender = 'FEMALE';
```

Your output should resemble:

Message

Stream created **and** running

Tip

You can run `DESCRIBE pageviews_female;` to describe the stream.

5. Create a new persistent query where another condition is met, using `LIKE`. Results from this query are written to the `pageviews_enriched_r8_r9` Kafka topic.

```
CREATE STREAM pageviews_female_like_89
WITH (kafka_topic='pageviews_enriched_r8_r9') AS
SELECT * FROM pageviews_female
WHERE regionid LIKE '%_8' OR regionid LIKE '%_9';
```

Your output should resemble:

Message

Stream created **and** running

6. Verify the above 2 streams:

```
select * from PAGEVIEWS_FEMALE_LIKE_89 emit changes limit 6;
select * from PAGEVIEWS_FEMALE emit changes limit 3;
```

```
ksql> select * from PAGEVIEWS_FEMALE_LIKE_89 emit changes limit 6;
```

USERID	PAGEID	REGIONID	GENDER
User_9	Page_15	Region_9	FEMALE
User_9	Page_17	Region_8	FEMALE
User_9	Page_66	Region_8	FEMALE
User_9	Page_62	Region_8	FEMALE
User_9	Page_71	Region_8	FEMALE
User_6	Page_31	Region_8	FEMALE

```
Limit Reached
Query terminated
ksql> select * from PAGEVIEWS_FEMALE emit changes limit 3;
```

USERID	PAGEID	REGIONID	GENDER
User_1	Page_30	Region_8	FEMALE
User_3	Page_23	Region_6	FEMALE
User_1	Page_81	Region_8	FEMALE

```
Limit Reached
Query terminated
ksql>
```

7. Create a new persistent query that counts the pageviews for each region combination in a **tumbling window** of 30 seconds when the count is greater than one. Results from this query are written to the **PAGEVIEWS_REGIONS** Kafka topic in the Avro format. **KSQL**

will register the Avro schema with the configured Schema Registry when it writes the first message to the `PAGEVIEWS_REGIONS` topic.

```
CREATE TABLE pageviews_regions
WITH (
  KAFKA_TOPIC = 'pageviews_regions', VALUE_FORMAT='AVRO'
) AS
SELECT regionid, COUNT(*) AS numusers
FROM pageviews_enriched
WINDOW TUMBLING (size 30 second)
GROUP BY regionid
HAVING COUNT(*) > 1 emit changes;
```

Your output should resemble:

Message

Table created **and** running

Tip

You can run `DESCRIBE pageviews_regions;` to describe the table.

8. Optional: View results from the above queries using `SELECT`.

SELECT regionid, numusers FROM pageviews_regions emit changes LIMIT 5;

Your output should resemble:

```
ksql> SELECT regionid, numusers FROM pageviews_regions emit changes LIMIT 5;
```

```
+-----+-----+
|REGIONID          |NUMUSERS          |
+-----+-----+
|Region_2          |221               |
|Region_3          |6169              |
|Region_5          |10659             |
|Region_2          |11476             |
|Region_9          |2259              |
Limit Reached
Query terminated
```

9. Optional: Show all persistent queries.

SHOW QUERIES;

Your output should resemble:

Query ID	Kafka Topic	Query String
-----	-----	-----
-----	-----	-----

```

-----
-----
CSAS_PAGEVIEWS_FEMALE_1 | PAGEVIEWS_FEMALE | CREATE STREA
M pageviews_female AS SELECT * FROM pageviews_enriched WHERE gender =
'FEMALE';
CTAS_PAGEVIEWS_REGIONS_3 | PAGEVIEWS_REGIONS | CREATE TABLE
pageviews_regions WITH (VALUE_FORMAT='avro') AS SELECT gender, region
id , COUNT(*) AS numusers FROM pageviews_enriched WINDOW TUMBLING
(size 30 second) GROUP BY gender, regionid HAVING COUNT(*) > 1;
CSAS_PAGEVIEWS_FEMALE_LIKE_89_2 | PAGEVIEWS_FEMALE_LIKE_89 | CRE
ATE STREAM pageviews_female_like_89 WITH (kafka_topic='pageviews_enriche
d_r8_r9') AS SELECT * FROM pageviews_female WHERE regionid LIKE '%_8' O
R regionid LIKE '%_9';
CSAS_PAGEVIEWS_ENRICHED_o | PAGEVIEWS_ENRICHED | CREATE STR
EAM pageviews_enriched AS SELECT users_original.userid AS userid, pageid, regio
nid, gender FROM pageviews_original LEFT JOIN users_original ON pagevie
ws_original.userid = users_original.userid;
-----
-----
-----
-----

```

For detailed information on a Query run: EXPLAIN <Query ID>;

10. Optional: Examine query run-time metrics and details. Observe that information including the target Kafka topic is available, as well as throughput figures for the messages being processed.

DESCRIBE PAGEVIEWS_REGIONS EXTENDED;

Your output should resemble:

```
Name      : PAGEVIEWS_REGIONS
Type      : TABLE
Key field  : KSQL_INTERNAL_COL_0|+|KSQL_INTERNAL_COL_1
Key format : STRING
Timestamp field : Not set - using <ROWTIME>
Value format : AVRO
Kafka topic : PAGEVIEWS_REGIONS (partitions: 4, replication: 1)
```

Field | Type

```
-----
ROWTIME | BIGINT      (system)
ROWKEY  | VARCHAR(STRING) (system)
GENDER  | VARCHAR(STRING)
REGIONID | VARCHAR(STRING)
NUMUSERS | BIGINT
-----
```

Queries that write into this TABLE

```
-----
CTAS_PAGEVIEWS_REGIONS_3 : CREATE TABLE pageviews_regions WITH (value_format='avro') AS SELECT gender, regionid , COUNT(*) AS numusers FROM
```

```
pageviews_enriched    WINDOW TUMBLING (size 30 second)    GROUP BY gender,
regionid    HAVING COUNT(*) > 1;
```

For query topology **and** execution plan please run: EXPLAIN <QueryId>

Local runtime statistics

```
-----
messages-per-sec:    3.06 total-messages:    1827    last-message: 7/19/18 4:17:55 PM
UTC
failed-messages:    0 failed-messages-per-sec:    0    last-failed:    n/a
(Statistics of the local KSQL server interaction with the Kafka topic PAGEVIEWS_REGI
ONS)
ksql>
```

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

https://ksqldb.io/quickstart.html?_ga=2.53841192.1438767497.1642131382-2002989446.1641377120&_gac=1.255954681.1642171371.CjwKCAiA24SPBhBoEiwAjBgkhg1qFCOJ-Ohq2cWlGrT9c3232dWfPKKpOG6zXpZrNXjqUelgasqp5BoCTEoQAvD_BwE

Any issues related to minimum config clean the zookeeper/kafka-logs and restart the services.

6. Errors

I. 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
# Many listener names, to security protocols, the default is for them to be the
```

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
# value
# 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.

7. Annexure Code:

II. DumplogSegment

```
/opt/kafka/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files \
```

```
/tmp/kafka-logs/my-kafka-connect-o/ooooooooooooooooooooo.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      0000000000000000000011.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.DumpLogSegments --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 value
size: 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 value
size: 43 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence:
-1 isTransactional: false headerKeys: [] payload: It will be consumed by the Kaf
ka Connector.
[root@tos my-kafka-connect-0]#

```

III. Data Generator – JSON

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 through 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:

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-1.4.0-bin.tar -C /apps )
```

3. Copy your custom configs into the conf directory
4. 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}
```

[view rawjackieChanCommands.json](#) hosted with [by GitHub](#)

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.

IV. Resources

<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>