1. Prerequisite

Scenario 1 : Using VM

Refer any tutorial in the web to configure Centos VM using VM Player or Workstation in your laptop.

Start the VM using VM player and Logon to the server using telnet or directly in the VM console. Enter the root credentials to logon.

Scenario 2: Using Docker

All the necessary software should be in the /Software folder. If its not there, ensure to copy it using winscp.exe from the windows desktop to /Software folder. You can create /Software folder using mkdir /Software.

The following instruction will create a network and bind to the container, ckafka0. Replace the -v parameter with any of the folder in your Host machine.

#docker network create --driver bridge spark-net

#docker run --name ckafka0 --hostname ckafka0 -p 9094:9092 -p 8086:8081 -p 2184:2181 -p 9031:9021 -p 8098:8088 -i -t --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt centos:7 /usr/sbin/init

2. 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

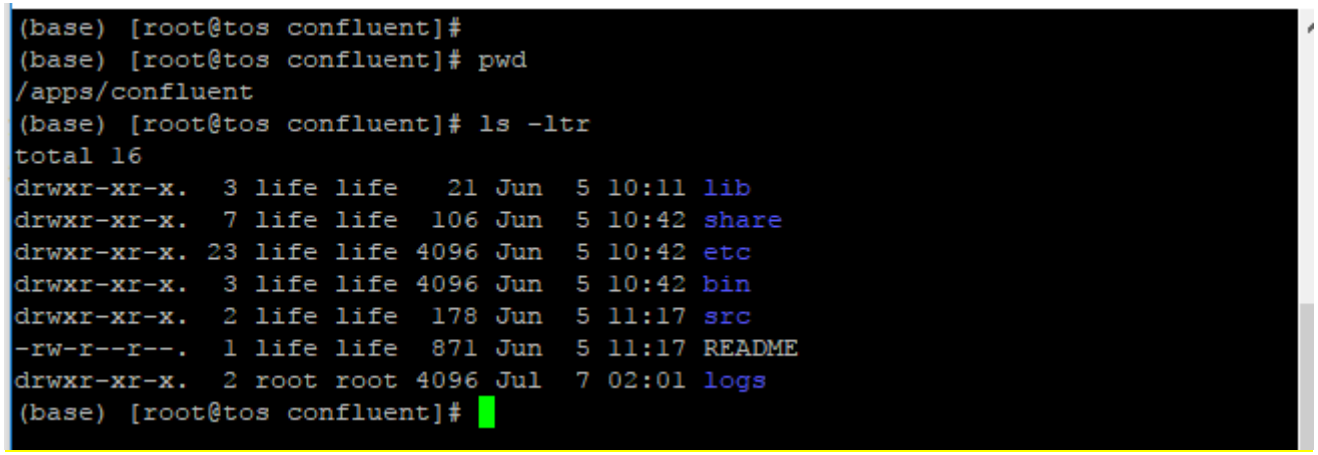
```
export PATH=/apps/confluent/bin:${PATH};
```

```
    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:



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-con
nect-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. f
rom Confluent Hub and installing into /apps/confluent/share/confluent-hub-compon
ents
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.8A2Ii7O4/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.

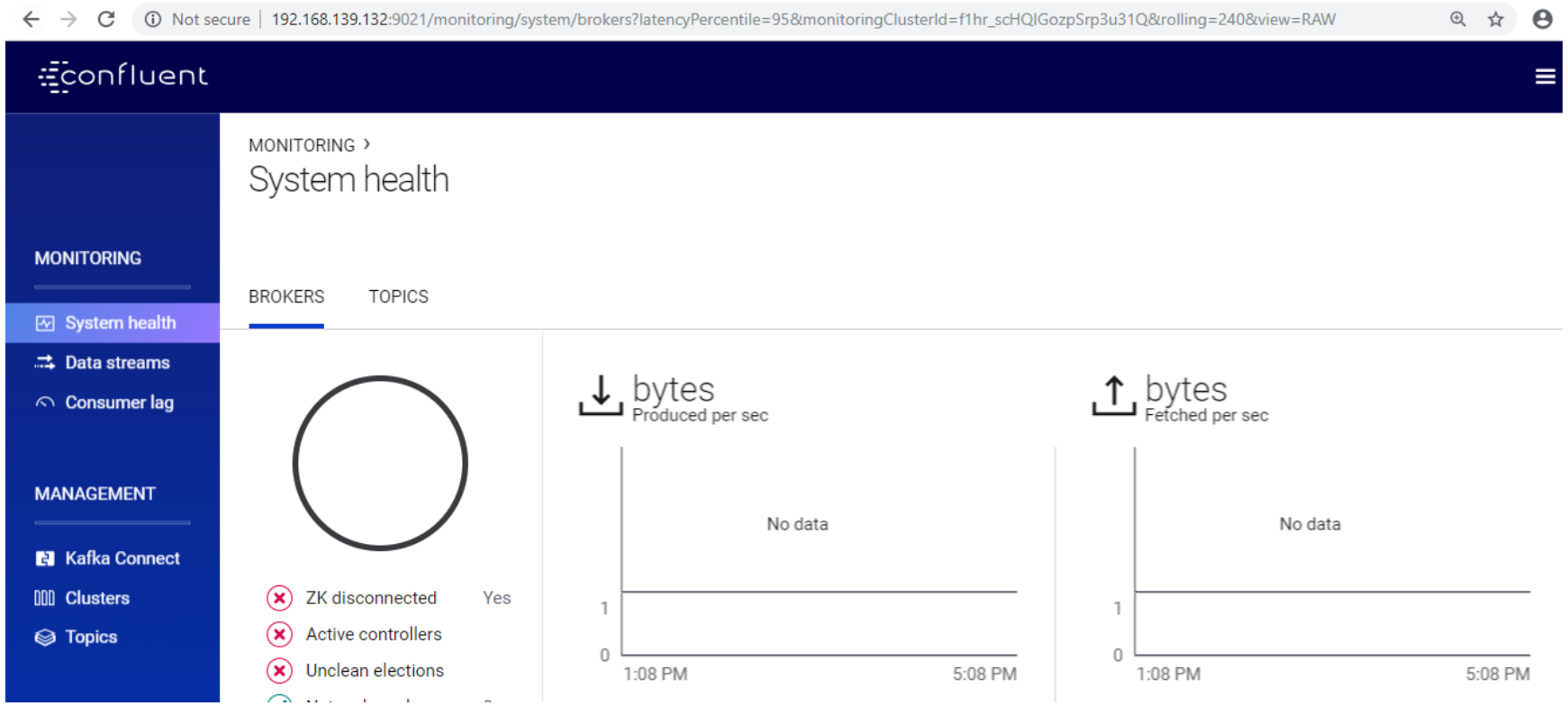
#export CONFLUENT_CURRENT=/opt/data/ckafka

#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/.


Not secure | 192.168.139.132:9021/monitoring/system/brokers?latencyPercentile=95&monitoringClusterId=f1hr_scHQIGozpSrp3u31Q&rolling=240&view=RAW
confluent
MONITORING
System health
Data streams
Consumer lag
MANAGEMENT
Kafka Connect
Clusters
Topics
MONITORING ›
System health
BROKERS
TOPICS
ZK disconnected Yes
Active controllers
Unclean elections
bytes
Produced per sec
No data
1
0
1:08 PM
5:08 PM
bytes
Fetched per sec
No data
1
0
1:08 PM
5:08 PM

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

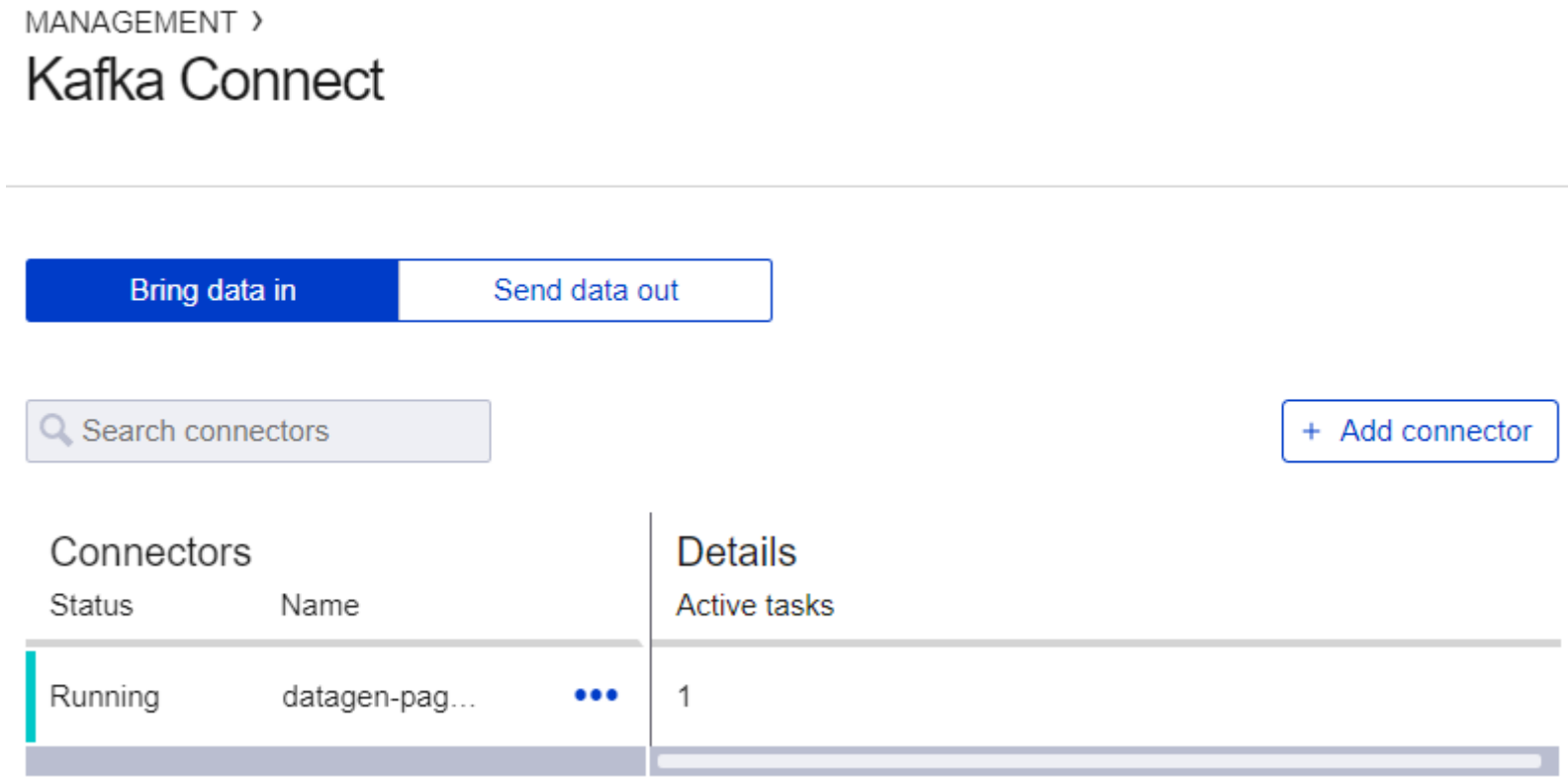
Find the DatagenConnector tile and click **Connect**.

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**.



Run another instance of the Kafka Connect Datagen connector to produce Kafka data to the `users` topic in AVRO format.

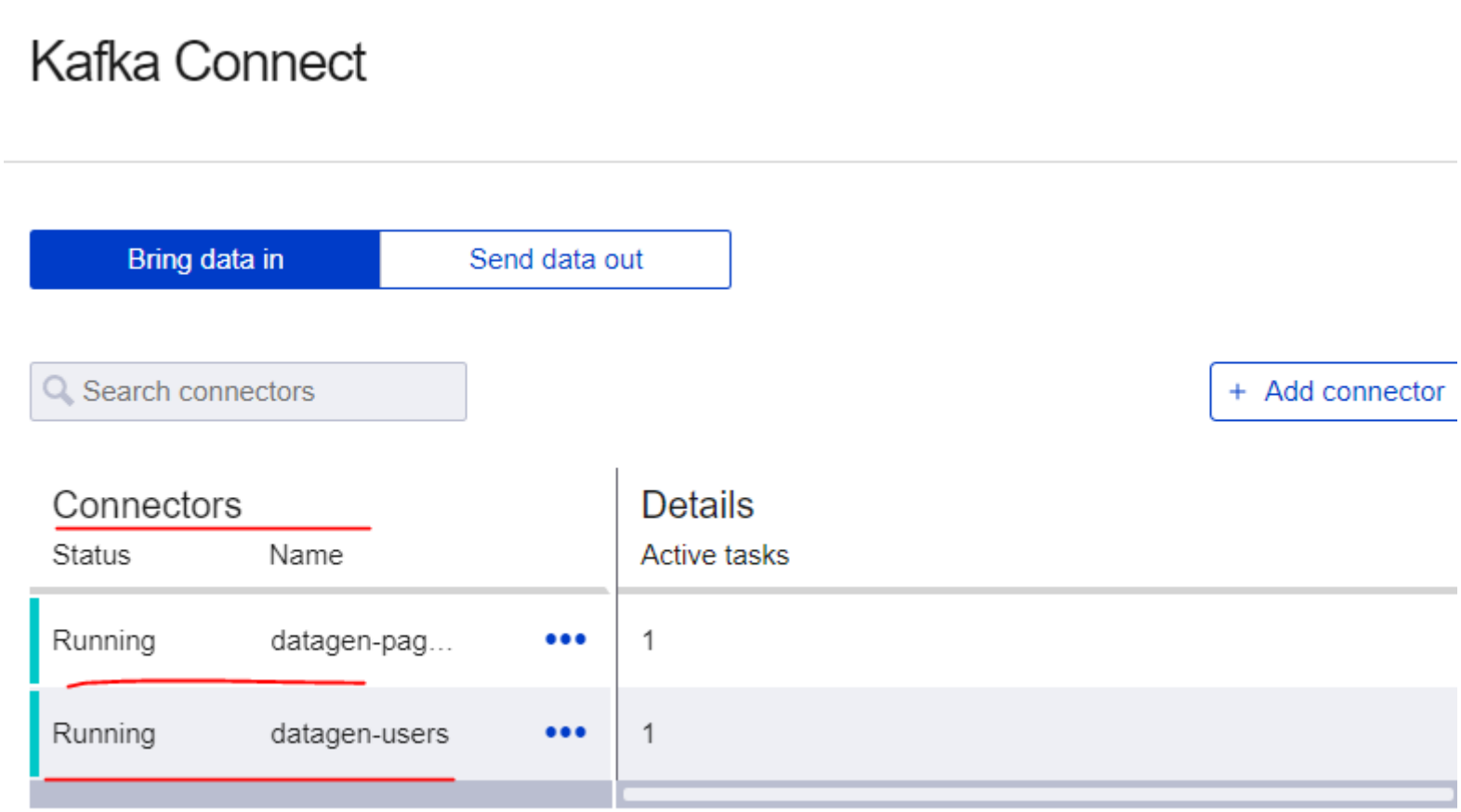
Click **Add connector**.

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**.

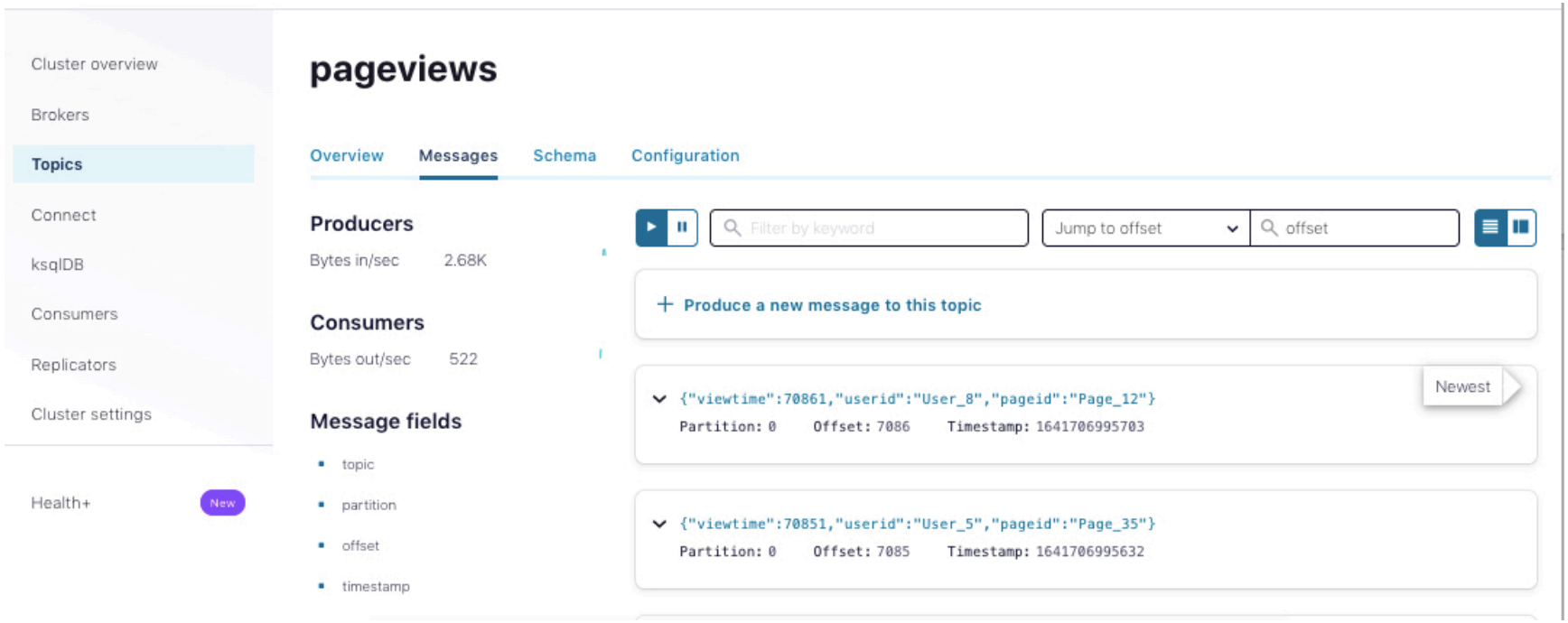
At the end of this.


Kafka Connect
Bring data in
Send data out
Search connectors
+ Add connector
Connectors
Status
Name
Details
Active tasks
Running
datagen-pag...
1
Running
datagen-users
1

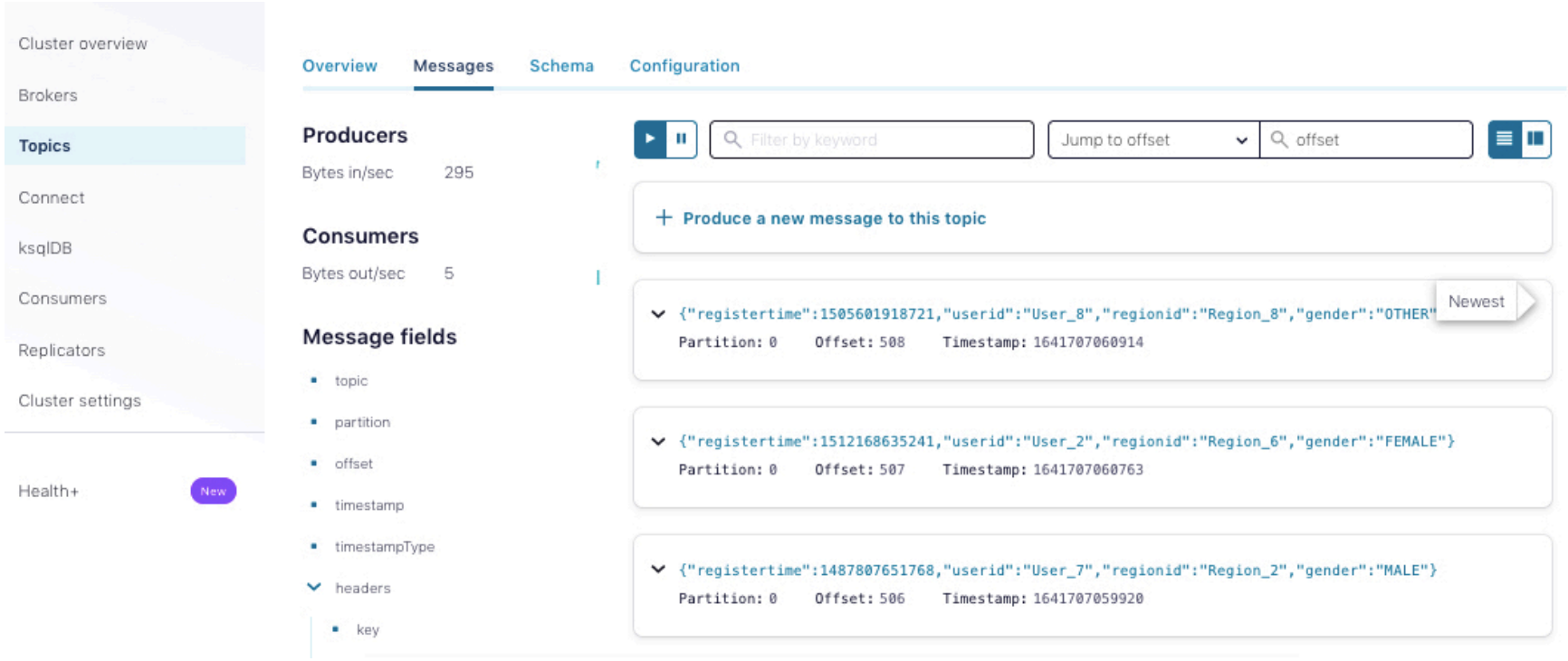
Verify the messages in the both the topics:

Using the control centers:

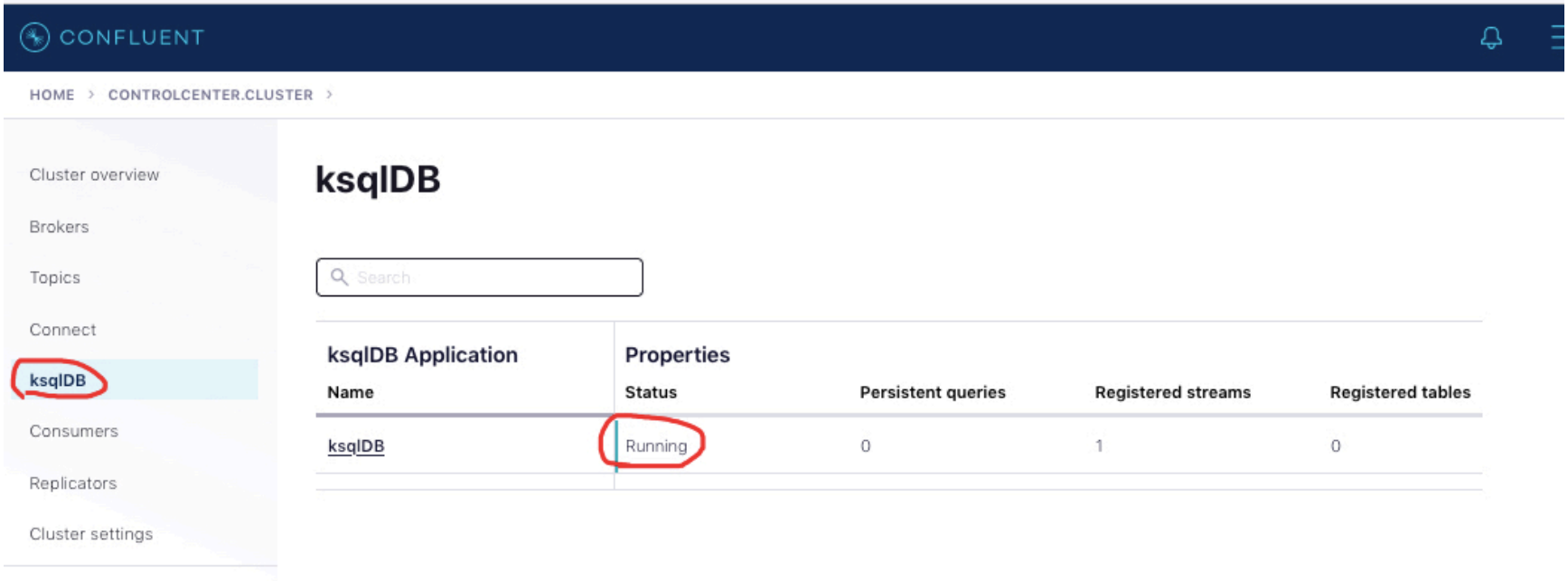
Topics -> pageviews -> Messages:



Topics -> Users -> Messages:



Ensure that ksql DB services is up.



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. ----------

3. Workflow using KSQL - CLI – 90 Minutes

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.

Prerequisites:

- Confluent Platform is installed and running. This installation includes a Kafka broker, KSQL, Control Center, ZooKeeper, Schema Registry, REST Proxy, and Connect.
- If you installed Confluent Platform via TAR or ZIP, navigate into the installation directory. The paths and commands used throughout this tutorial assume that you are in this installation directory.
- Consider installing the Confluent CLI to start a local installation of Confluent Platform.

- Java: Minimum version 1.8. Install Oracle Java JRE or JDK >= 1.8 on your local machine

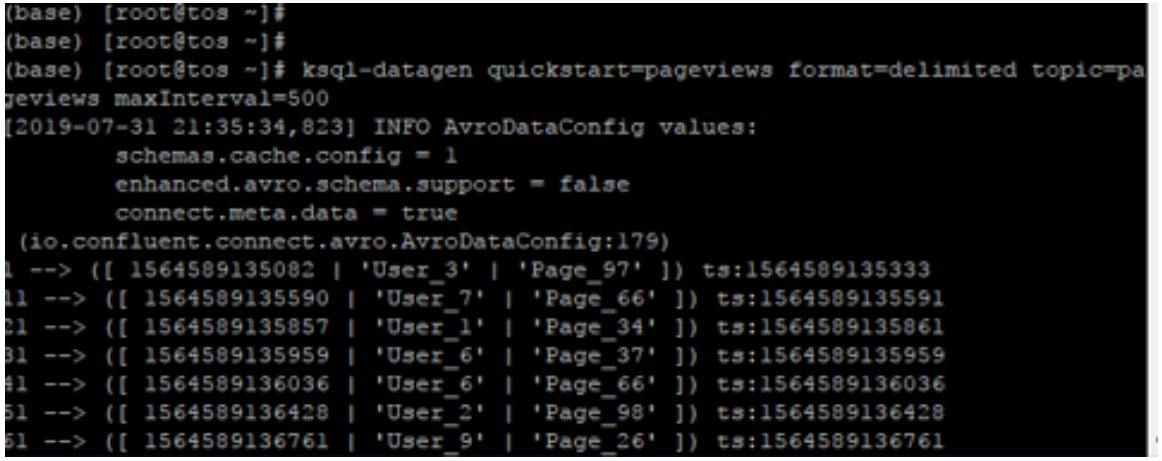
Create Topics and Produce Data

https://ksqldb.io/quickstart.html?_ga=2.53841192.1438767497.1642131382-2002989446.1641377120&_gac=1.255954681.1642171371.CjwKCAiA24SPBhB0EiwAjBgkhg1qFCOJ-Ohq2cWlGrT9c3232dWfPKKpOG6zXpZrNXjqUelgasqp5BoCTE0QAvD_BwE

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 quickstart=pageviews format=json topic=pageviews maxInterval=500
```

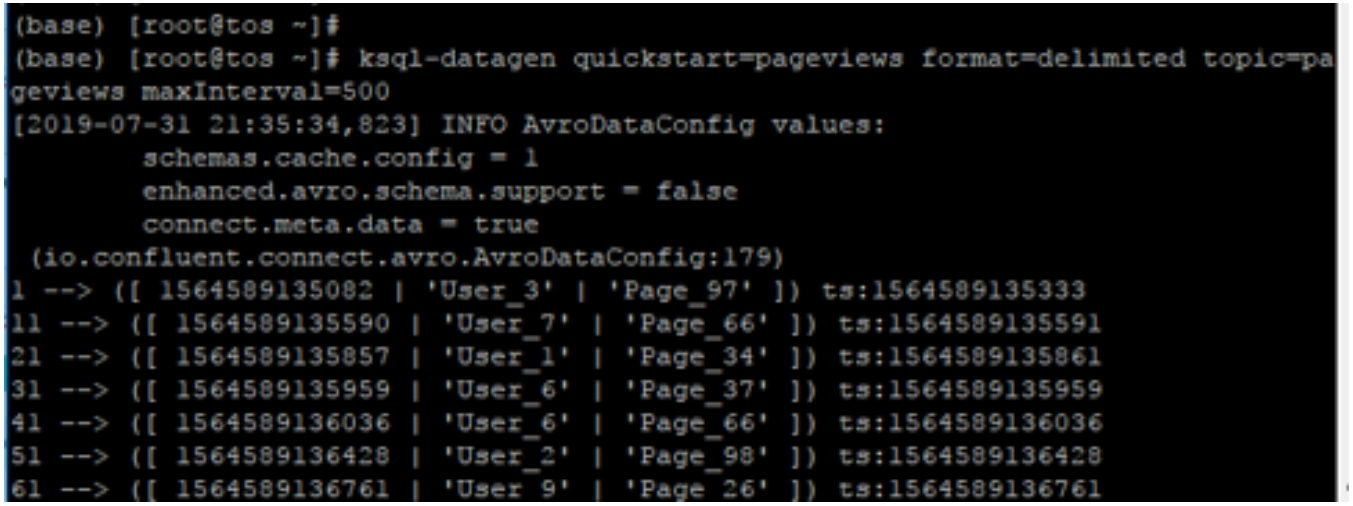


2. Produce Kafka data to the `users` topic using the data generator. The following example continuously generates data with a value in JSON format.

```
$ ksql-datagen 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.



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`.

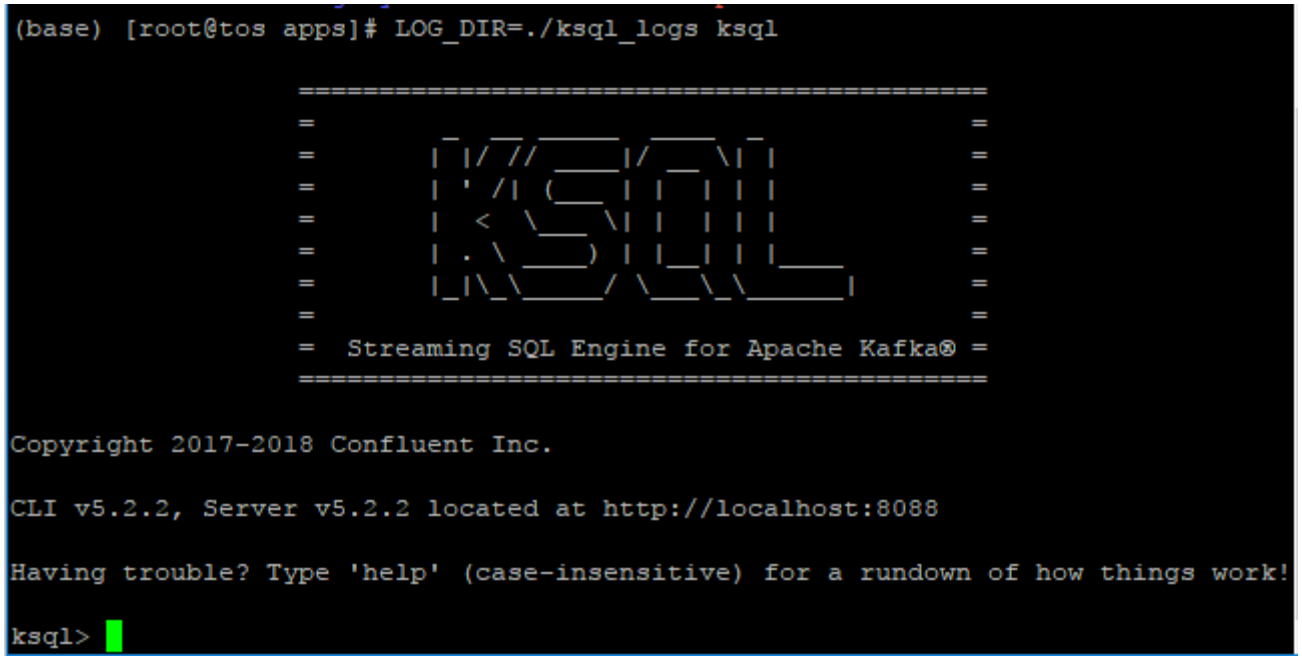
```
$ LOG_DIR=./ksql_logs ksql
```

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.



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';
```

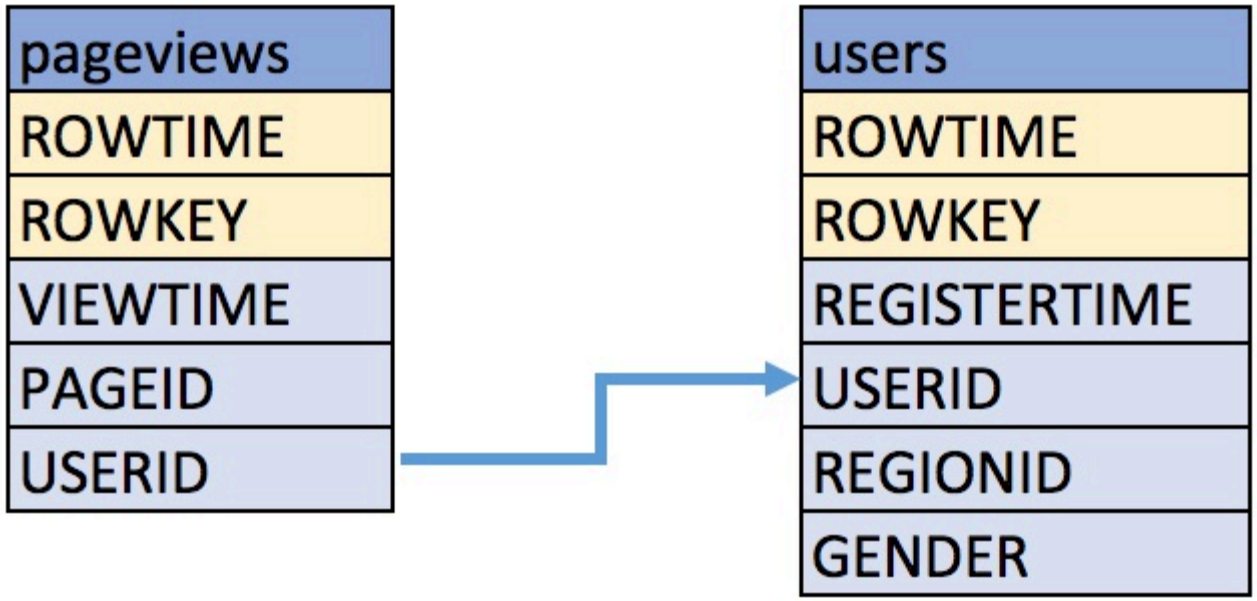
Your output should resemble:

```
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

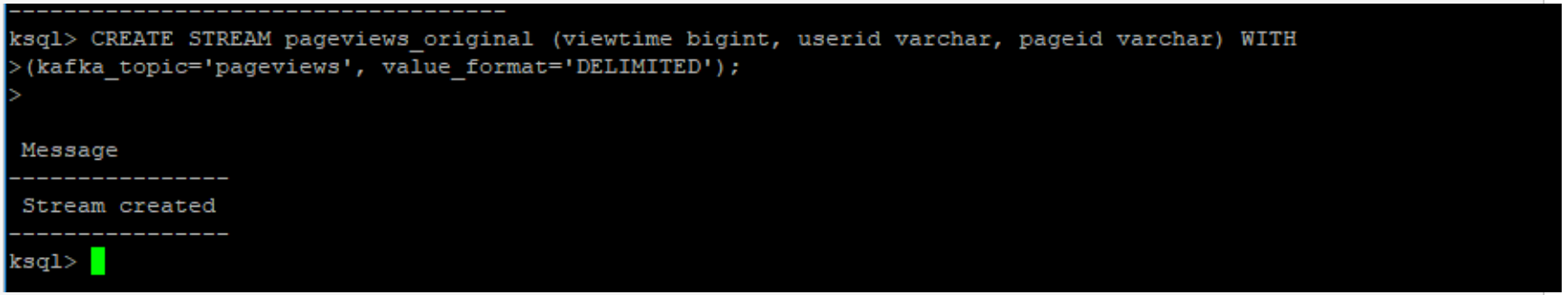
These examples query messages from Kafka topics called `pageviews` and `users` using the following schemas:



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) W
ITH
(kafka_topic='pageviews', value_format='JSON');
```

Your output should resemble:



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 VARCH
AR, 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

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

Your output should resemble:

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

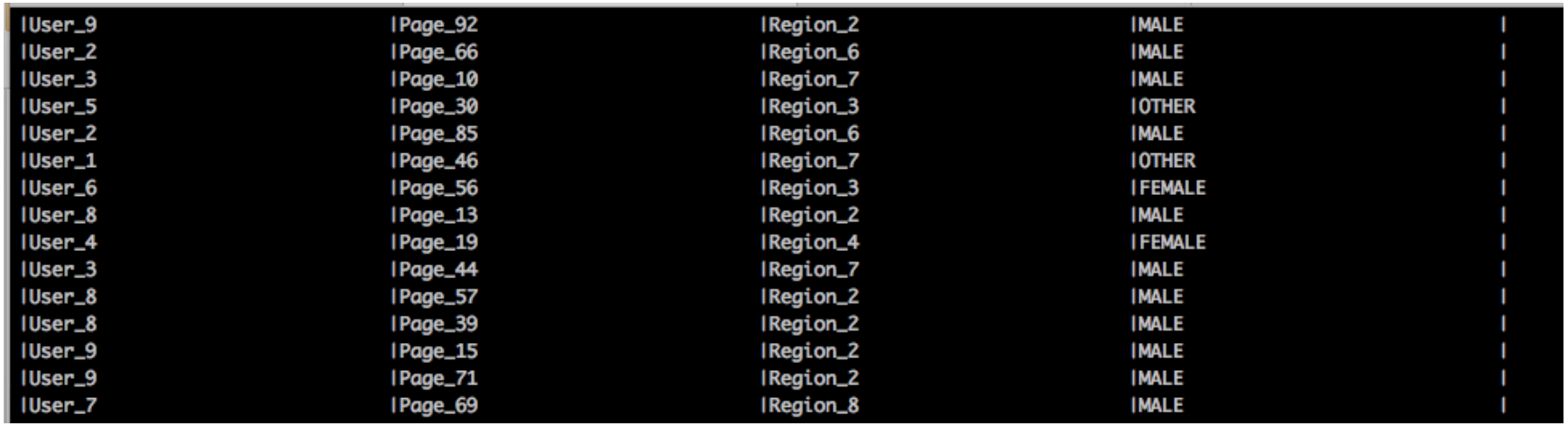
Tip

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.

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

Your output should resemble:



|User_9	|Page_92	|Region_2	|MALE
|User_2	|Page_66	|Region_6	|MALE
|User_3	|Page_10	|Region_7	|MALE
|User_5	|Page_30	|Region_3	|OTHER
|User_2	|Page_85	|Region_6	|MALE
|User_1	|Page_46	|Region_7	|OTHER
|User_6	|Page_56	|Region_3	|FEMALE
|User_8	|Page_13	|Region_2	|MALE
|User_4	|Page_19	|Region_4	|FEMALE
|User_3	|Page_44	|Region_7	|MALE
|User_8	|Page_57	|Region_2	|MALE
|User_8	|Page_39	|Region_2	|MALE
|User_9	|Page_15	|Region_2	|MALE
|User_9	|Page_71	|Region_2	|MALE
|User_7	|Page_69	|Region_8	|MALE

5. 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.

6. 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
----------------------------
```

7. 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>
```

8. 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.

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

```
SELECT regionid, numusers FROM pageviews_regions 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
```

10. Optional: Show all persistent queries.

```
11.  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_0      | 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>;
```

12. 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 (val
ue_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 ----------