IST769 Lab K

Search Model: Kafka and KSQLdb

In this lab, we will explore the streaming model using Apache Kafka and Confluent's KSQLDb which allows you to use SQL for stream processing. We will also integrate Kafka into Drill.

Learning Outcomes

At the end of this lab you should be able to:

- Query Kafka topics with KSQL, Drill and Python.
- Use KSQL to manipulate a data stream.
- Write data streams to other databases or files.

Pre-Requisites

Before you begin:

- Open a terminal window in the lab environment
- Set the current working directory to **advanced-databases**
- Start the following services required by the lab:
  jupyter drill zookeeper broker ksqldb-server ksqldb-cli schema-registry connect

Tools Used In this Lab

The following tools will be used in this lab:

1. To access Jupyter Lab from your Windows host:
   http://localhost:8888
   The password is **SU2orange!**
2. To access the Drill Use
   http://localhost:8047
3. To access KSQL Db Client:
   `docker-compose exec ksqldb-cli ksql http://ksqldb-server:8088`
4. Start the ATM producer example from Jupyter, located at:
   /work/examples/Kafka-Producer.ipynb

Lab Problem Set

QUESTIONS:

For each question, include a copy of the code required to complete the question along with a screenshot of the code and a screenshot of the output.

1. Write a drill query to display only the atm transactions that ended in an Error status. Show all columns and sort output so the newest errors are first.
   NOTE: It is strongly suggested you use `backticks` and table aliases when working in drill

Query

```
1  select * from kafka.`atm`
2      where status = 'error'
3      order by `TimeStamp` desc
```

Open *Un... Save
1 hkushta
Plain Text Tab Width: 8 Ln 1, Col 8

Id	Location	User	TimeStamp	Amount	Status	kafkaTopic	kafkaPartitionId	kafkaMsgOffset	kafkaMsgTimestamp
87e7d8b5-75c0-4c0b-86e8-948bf796678d	tully	zeke	1713299271000	120	error	atm	0	609	1713299271005
b6ac21c2-788a-4b3e-85b1-9332e6aca161	cicero	walt	1713299261000	200	error	atm	0	604	1713299261979
c0c26c2c-ec91-44f9-8c06-835342f19800	syracuse	devin	1713299209000	100	error	atm	0	567	1713299209796
74970086-cbdc-406d-863f-036b97c30e82	dewitt	abby	1713299167000	180	error	atm	0	565	1713299167648
ce150a28-96d8-4e9f-afa5-cd74259c95d1	cicero	tosh	1713299137000	100	error	atm	0	564	1713299137530
5250ce33-276b-4aec-84a0-f3c36888abe5	dewitt	bob	1713299124000	80	error	atm	0	507	1713299124481

2. Write a drill query to display the total amount withdrawn by user and do not include error transactions in the totals.

Query type: SQL Physical Logical

Query

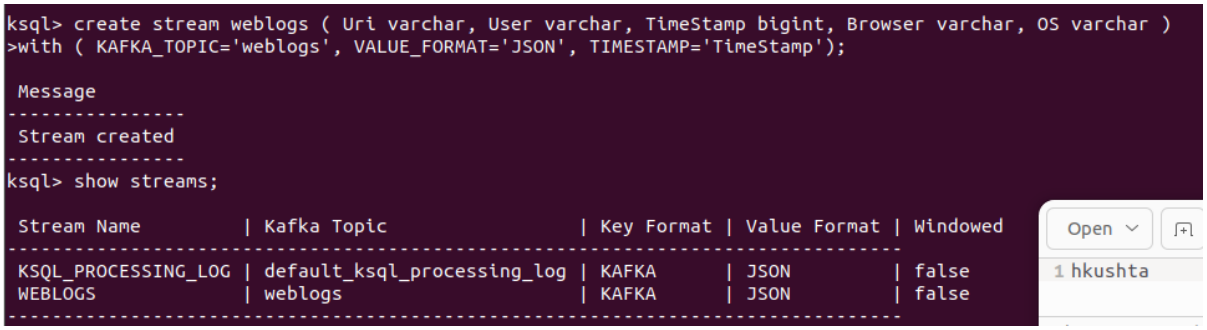
```
1  select a.`User`, sum(Amount) as TotalWithdrawn
2      from kafka.`atm` a
3      where status <> 'error'
4      group by a.`User`
```

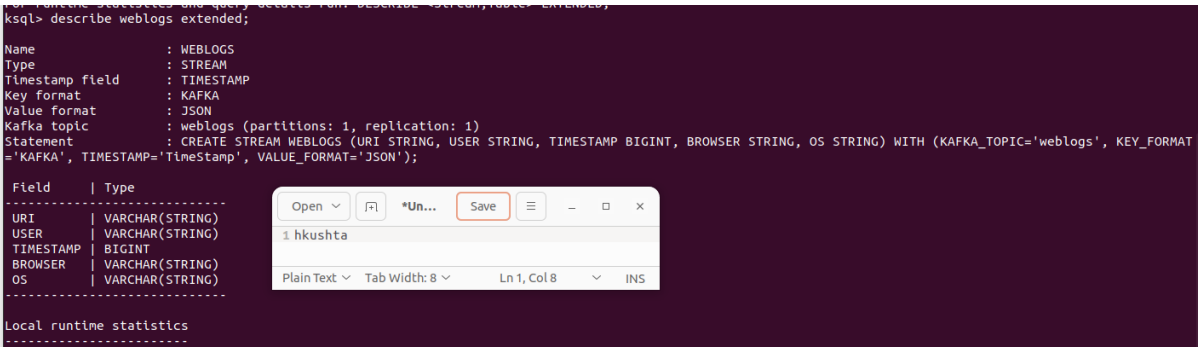
Open *Un..
1 hkushta
Plain Text Tab Width:

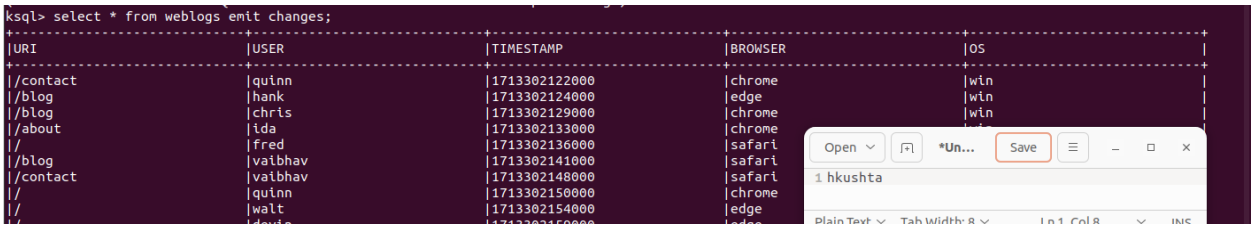
User	TotalWithdrawn
vaibhav	4680
karley	4340
walt	4660
patty	3440
gigi	2380
fred	3860

Open
1 hkushta
Plain Text Tab

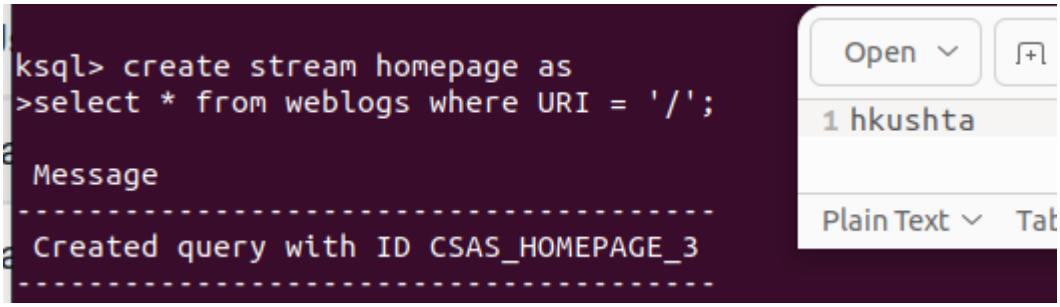
3. Write KSQL to create a stream named **weblogs** from the JSON keys in the weblogs Kafka topic. Make sure to set the TIMESTAMP property to the timestamp from the stream.







4. Write a KSQL statement create a persistent stream/table called **homepage** which only displays visitors to the root of the website (/). It should display all columns from the weblogs stream.



```
ksql> select * from homepage emit changes;
+-------------------------+-------------------------+-------------------------+-------------------------+-------------------------+
|URI                      |USER                     |TIMESTAMP                |BROWSER                  |OS                       |
+-------------------------+-------------------------+-------------------------+-------------------------+-------------------------+
|/                        |abby                     |1713306668000            |chrome                   |osx                      |
|/                        |abby                     |1713306681000            |chrome                   |osx                      |
|/                        |rose                     |1713306706000            |chrome                   |osx                      |
|/                        |gigi                     |1713306711000            |chrome                   |win                      |
```

5. Write a KSQL statement to count operating systems users (os) in 60 second windows. After 60 seconds, the counter should reset, and counts should begin again.

```
ksql> select os, count(*) from weblogs window tumbling (size 60 seconds) group by os emit changes;
+-------------------------------------------------------------------------------------------------+-----+
|OS                                                                                               |KSQL |
+-------------------------------------------------------------------------------------------------+-----+
|osx                                                                                              |1    |
|osx                                                                                              |2    |
|osx                                                                                              |3    |
|win                                                                                              |1    |
|win                                                                                              |2    |
|osx                                                                                              |4    |
|osx                                                                                              |5    |
|osx                                                                                              |6    |
|win                                                                                              |3    |
```

6. Write a KSQL persistent stream/table called **user_activity** which will display a count of user activity on the website within 1-minute sessions.

```
ksql> create table user_activity as select user, count(*) as usercount from weblogs
window tumbling (size 60 seconds ) group by user emit changes;

 Message
-----------------------------------------
 Created query with ID CTAS_USER_ACTIVITY_7
-----------------------------------------
ksql> show tables;

 Table Name    | Kafka Topic   | Key Format | Value Format | Windowed
-------------------------------------------------------------------------
 REQHOUR       | REQHOUR       | KAFKA      | JSON         | true
 USER_ACTIVITY | USER_ACTIVITY | KAFKA      | JSON         | true
-------------------------------------------------------------------------
```

```
ksql> show topics;

 Kafka Topic                 | Partitions | Partition Replicas
-----------------------------------------------------------------
 HOMEPAGE                    | 1          | 1
 REQHOUR                     | 1          | 1
 USER_ACTIVITY               | 1          | 1
 atm                         | 1          | 1
 default_ksql_processing_log | 1          | 1
 weblogs                     | 1          | 1
-----------------------------------------------------------------
```

```
ksql> select * from user_activity emit changes;
+------------------+------------------+------------------+------------------+
|USER              |WINDOWSTART       |WINDOWEND         |USERCOUNT         |
+------------------+------------------+------------------+------------------+
|ida               |1713308700000     |1713308760000     |1                 |
|abby              |1713308700000     |1713308760000     |1                 |
|hank              |1713308700000     |1713308760000     |1                 |
|xavier            |1713308700000     |1713308760000     |1                 |
|otto              |1713308700000     |1713308760000     |1                 |
|ida               |1713308700000     |1713308760000     |2                 |
|bob               |1713308700000     |1713308760000     |1                 |
```

Open *Un
1 hkushta

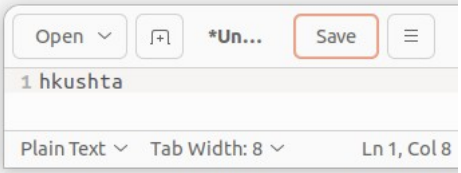
7. Write a KSQL statement to display users who have more than 1 pages of activity in a 1-minute window.

```
ksql> select * from user_activity where usercount > 1 emit changes;
+------------------+------------------+------------------+------------------+
|USER              |WINDOWSTART       |WINDOWEND         |USERCOUNT         |
+------------------+------------------+------------------+------------------+
|ida               |1713308700000     |1713308760000     |2                 |
|otto              |1713308700000     |1713308760000     |2                 |
|bob               |1713308700000     |1713308760000     |2                 |
|gigi              |1713308700000     |1713308760000     |2                 |
|karley            |1713308700000     |1713308760000     |2                 |
|otto              |1713308700000     |1713308760000     |3                 |
|gigi              |1713308700000     |1713308760000     |3                 |
|abby              |1713308700000     |1713308760000     |2                 |
```

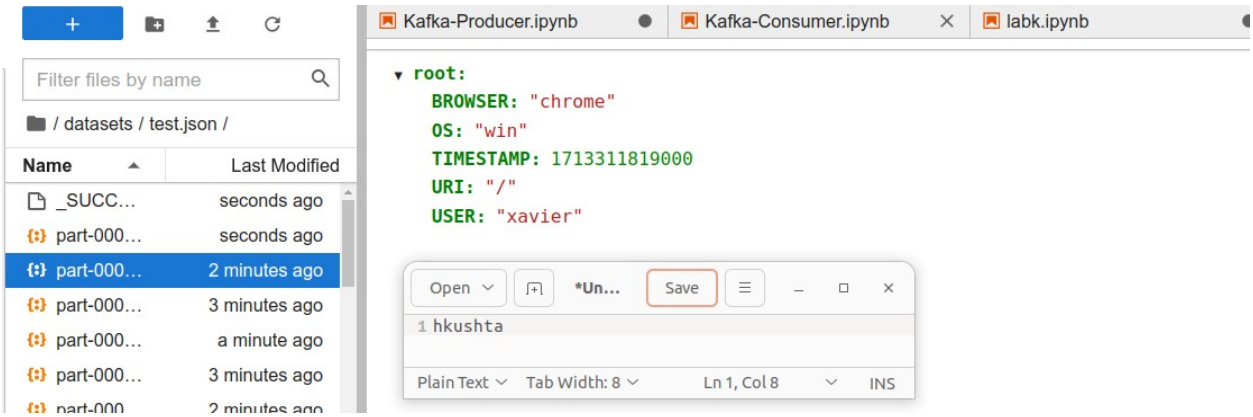
Open *U
1 hkushta
Plain Text Tab Widt

8. In Jupyter, write a program to subscribe to the homepage topic generated by the stream/table in Question 4 and display the messages to the console.
   NOTE: We could easily then write these to elasticsearch, but we will not do that in this lab.

```
consumer = Consumer({'bootstrap.servers' : 'broker:29092', 'group.id' : '*'})
consumer.subscribe(["HOMEPAGE"])
count = 0
maxcount = 10
df = None

try:
    while True:
        msg = consumer.poll(1.0)

        if msg is None:
            continue
        if msg.error():
            print(f"Consumer error: {msg.error()}")
            continue

        raw = msg.value().decode('utf-8')
        payload = json.loads(raw)

        ##
        row = spark.createDataFrame([payload])
        if df is None:
            df = row
        else:
            df = df.union(row)
            count = count + 1


        print(f"Received message: {payload}")
        print(count)
        if count == maxcount:
            print("Write to file...")
            count = 0
            df.write.mode("append").json("file:///home/jovyan/datasets/test.jso

except KeyboardInterrupt:
    consumer.close()
```




```
Received message: {'URI': '/', 'USER': 'xavier', 'TIMESTAMP': 1713311819000, 'BROWSER': 'chrome', 'OS': 'win'}
0
Received message: {'URI': '/', 'USER': 'quinn', 'TIMESTAMP': 1713311823000, 'BROWSER': 'chrome', 'OS': 'win'}
1
Received message: {'URI': '/', 'USER': 'vaibhav', 'TIMESTAMP': 1713311828000, 'BROWSER': 'safari', 'OS': 'osx'}
2
Received message: {'URI': '/', 'USER': 'yolanda', 'TIMESTAMP': 1713311829000, 'BROWSER': 'chrome', 'OS': 'osx'}
3
Received message: {'URI': '/', 'USER': 'karley', 'TIMESTAMP': 1713311831000, 'BROWSER': 'chrome', 'OS': 'win'}
4
Received message: {'URI': '/', 'USER': 'xavier', 'TIMESTAMP': 1713311832000, 'BROWSER': 'chrome', 'OS': 'win'}
5
Received message: {'URI': '/', 'USER': 'otto', 'TIMESTAMP
6
Received message: {'URI': '/', 'USER': 'hank', 'TIMESTAMP
7
Received message: {'URI': '/', 'USER': 'mike', 'TIMESTAMP
8
Received message: {'URI': '/', 'USER': 'vaibhav', 'TIMESTAMP': 1713311843000, 'BROWSER': 'safari', 'OS': 'osx'}
9
Received message: {'URI': '/', 'USER': 'nancy', 'TIMESTAMP': 1713311844000, 'BROWSER': 'chrome', 'OS': 'win'}
10
Write to file...

Received message: {'URI': '/', 'USER': 'lisa', 'TIMESTAMP': 1713311848000, 'BROWSER': 'chrome', 'OS': 'osx'}
1
Received message: {'URI': '/', 'USER': 'nancy', 'TIMESTAMP': 1713311849000, 'BROWSER': 'chrome', 'OS': 'win'}
```



IMPORTANT: When you are finished with the lab, execute:

```
PS:> docker-compose stop
```

To turn off all running services, then shut down your Azure Lab instance.