If all services are running properly then point your browser to localhost:9021 to open the Confluent server UI.

If the UI is not running or displaying warnings or error wait a few minutes until all services have successfully

In case the wiki connector is not ingesting any new data, simply restart it:

curl -X POST localhost:8083/connectors/wiki/tasks/0/restart

# KSQL Self-learning Examples

## Example 1

We will create a stream of events that represent ratings of movies. We’ll then run a streaming query that maintains tumbling windows counting the total number of ratings that each movie has received.

The first thing we’ll need to start modeling this scenario is a stream that represents ratings of movies. One important attribute of these events is their timestamp since we’ll be modeling the number of ratings that each movie receives over time.

CREATE STREAM ratings (title VARCHAR, release\_year INT, rating DOUBLE, timestamp VARCHAR) WITH (kafka\_topic='ratings',

timestamp='timestamp',

timestamp\_format='yyyy-MM-dd HH:mm:ss',

partitions=1,

value\_format='avro');

The above statement creates a stream named **ratings** and automatically creates a Kafka topic under the same name.

Let us produce some events manually:

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Die Hard', 1998, 8.2, '2019-07-09 01:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Die Hard', 1998, 4.5, '2019-07-09 05:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Die Hard', 1998, 5.1, '2019-07-09 07:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Tree of Life', 2011, 4.9, '2019-07-09 09:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Tree of Life', 2011, 5.6, '2019-07-09 08:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('A Walk in the Clouds', 1995, 3.6, '2019-07-09 12:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('A Walk in the Clouds', 1995, 6.0, '2019-07-09 15:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('A Walk in the Clouds', 1995, 4.6, '2019-07-09 22:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('The Big Lebowski', 1998, 9.9, '2019-07-09 05:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('The Big Lebowski', 1998, 4.2, '2019-07-09 02:00:00');

INSERT INTO ratings (title, release\_year, rating, timestamp) VALUES ('Super Mario Bros.', 1993, 3.5, '2019-07-09 18:00:00');

Let’s check out what’s in the underlying ratings topic:

PRINT ratings FROM BEGINNING LIMIT 11;

Now that we have a stream with some events in it, let’s start to leverage them. Before we run a KSQL Select query against this stream, we need to make sure that the query will read from the beginning of the stream. This is because by default KSQL queries read data that has arrived after the query has started. Click on **Add query properties** in the KSQL UI and make sure **auto.offset.reset** is set to **earliest**

Now we can run a simple aggregation query:

SELECT title,

COUNT(\*) AS rating\_count,

TIMESTAMPTOSTRING(WINDOWSTART, 'yyy-MM-dd HH:mm:ss', 'UTC') AS window\_start,

TIMESTAMPTOSTRING(WINDOWEND, 'yyy-MM-dd HH:mm:ss', 'UTC') AS window\_end

FROM ratings

WINDOW TUMBLING (SIZE 6 HOURS)

GROUP BY title

EMIT CHANGES

LIMIT 11;

Check the output of this query and make sure you understand it. What happens if you remove the LIMIT clause?

That’s a fine snapshot, but we want to make this rolling count of ratings continuous. The following creates a new table that is continuously populated by its query:

CREATE TABLE rating\_count

WITH (kafka\_topic='rating\_count') AS

SELECT title,

COUNT(\*) AS rating\_count,

WINDOWSTART AS window\_start,

WINDOWEND AS window\_end

FROM ratings

WINDOW TUMBLING (SIZE 6 HOURS)

GROUP BY title;

This will create a persistent query that rating\_count that will persist its data to its underlying topic. This table can then also be queried in a SELECT statement in the usual way.

## Example 2

Please follow the example outlined here - <https://kafka-tutorials.confluent.io/create-stateful-aggregation-count/ksql.html>

As we already have a Confluent installation, you can skip the first steps and start directly from the CREATE statement in step 3.

## Example 3

<https://kafka-tutorials.confluent.io/transform-a-stream-of-events/ksql.html>

## Example 4

<https://kafka-tutorials.confluent.io/split-a-stream-of-events-into-substreams/ksql.html>

# KSQL Wikimedia Exercise

To create the **wikimedia\_recentchanges** stream run the following KSQL query:

CREATE STREAM WIKIMEDIA\_RECENTCHANGES (meta MAP<VARCHAR, VARCHAR>, id BIGINT, type VARCHAR, title VARCHAR, comment VARCHAR, parsedcomment VARCHAR, timestamp BIGINT, user VARCHAR, bot BOOLEAN, server\_url VARCHAR, server\_name VARCHAR, server\_script\_path VARCHAR, wiki VARCHAR) WITH (kafka\_topic='wikimedia\_recentchanges', value\_format='JSON', TIMESTAMP='timestamp');

This stream represents the changes made to Wikipedia.

Link to the stream: <https://stream.wikimedia.org/v2/stream/recentchange>

The stream schema is documented here

<https://github.com/wikimedia/mediawiki-event-schemas/blob/master/jsonschema/mediawiki/recentchange/current.yaml>

<https://www.mediawiki.org/wiki/Manual:RCFeed>

And each field in the schema is described here

<https://www.mediawiki.org/wiki/Manual:Recentchanges_table>

The purpose of this exercise is to run simple KSQL statements to understand a little bit more about the aggregate activity behind the changes in Wikipedia.

Your task is to research and find a way to answer the following questions:

* How many changes are made by bots? (Note: the field ‘bot’ is a boolean. You need to cast it to STRING in a query)
* How many changes in total are made by bots every minute?
* The Type field indicates the type of the change. What is the distribution of Type over Windows of say 60 seconds?

* The field Wiki indicates the language version of the changed articles. Which 3 versions have been changing most often in your whole stream?
* What are the top 5 users with the most changes in your stream data? What about every 60 seconds?

# Miscellaneous

* Query the KSQL Server directly via Rest API - <https://docs.confluent.io/4.1.0/ksql/docs/api.html>

Example:

curl -X "POST" "http://localhost:8088/query" \

-H "Content-Type: application/json; charset=utf-8" \

-d $'{"ksql": "SELECT \* FROM RATING\_COUNT EMIT CHANGES;", "streamsProperties": {"auto.offset.reset": "earliest"}}'

* Consume messages from a Kafka topic from the console:

./usr/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic <topic\_name> --from-beginning

# Documentation

KSQL Basics - <https://kafka-tutorials.confluent.io/>

KSQL Basics - <https://docs.confluent.io/current/ksqldb/tutorials/basics-local.html>

Advanced KSQL - <https://www.confluent.io/blog/stream-processing-twitter-data-with-ksqldb/>