Cloudera Streaming Analytics 1.3.0

Using SQL Stream Builder

Date published: 2019-12-17
Date modified: 2021-03-25

CLOUDERA

https://docs.cloudera.com/

Legal Notice

Contents

Using the Streaming SQL Console

The Streaming SQL Console is the user interface for the SQL Stream Builder. You can manage your queries, tables, functions and monitor the history of the SQL jobs using the SQL Stream Console.

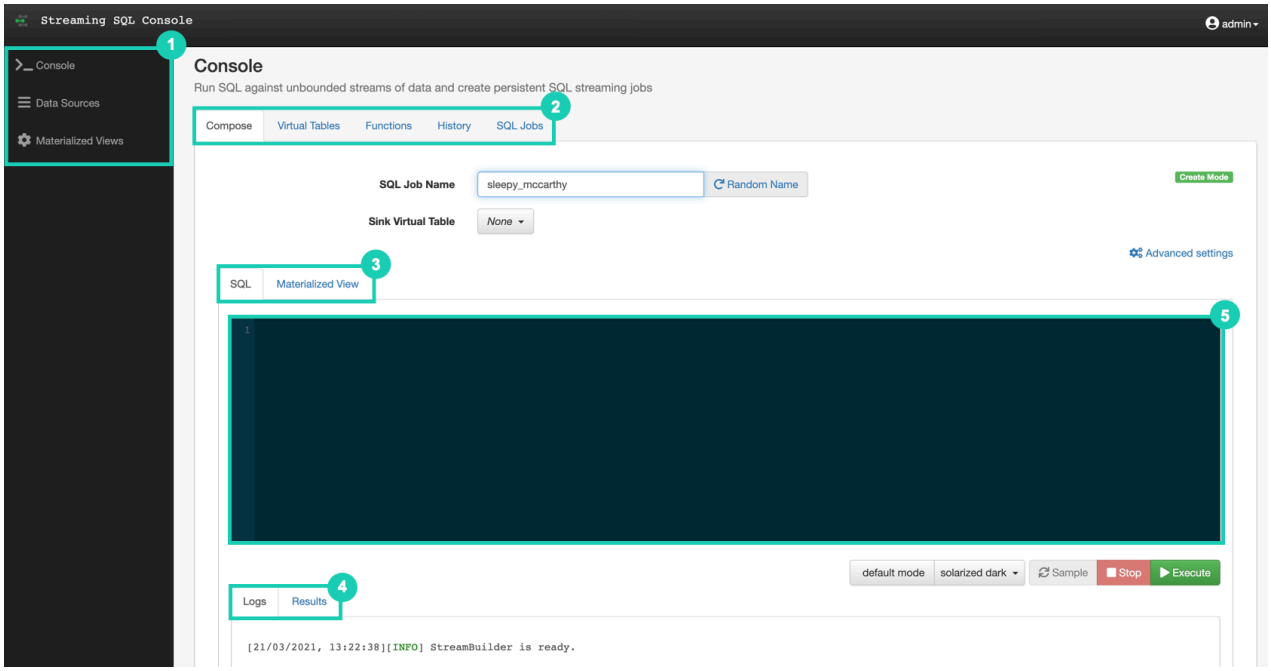
You can access the Streaming SQL Console through Cloudera Manager:

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.

The Streaming SQL Console opens up in a new window.

Important: On an unsecured cluster, you will be prompted to provide your username and password when logging in for the first time to the Streaming SQL Console. If you do not have an SSB account yet, click on Create an account.

The following illustration details the main menu and the tabs of the user interface.



1. The main menu consist of the following:
   - Console - The homepage of the Console where you can find the SQL window, other tabs and the Log window.
   - Data Sources - Setting up data sources and sinks for the Virtual Tables.
   - Materialized Views - Setting up and managing Materialized Views and API keys.
2. The main tabs consist of the following:
   - Compose - At default, the Compose tab is selected on the Console. You can create your SQL queries on the Compose tab.
   - Virtual Tables - You can select your Virtual Tables on this tab.
   - Functions - You can add the Javascript Functions to your SQL query.
   - History - You can review the history of submitted SQL queries.
   - SQL Jobs - You can review the history of submitted SQL jobs.

3. The alternate windows consist of the following:
   - SQL - At default, the SQL Window is displayed on the Console. You can add your SQL Queries to the window to compose queries.
   - Materialized View - Displays settings to create Materialized Views.
4. The audit tabs consist of the following:
   - Logs - Displays the status of the SQL Console.
   - Results - Displays the results of the executed SQL queries.
5. SQL Window - SQL statement editor of the Console.

Data Sources in SSB

Data sources are a catalog of data endpoints to be used as sources and sinks. Data providers allow you to register the provider using your security credentials, then use that provider for a source or sink in SQL Stream Builder. After adding a data source, you are able to use the registered component as virtual tables.

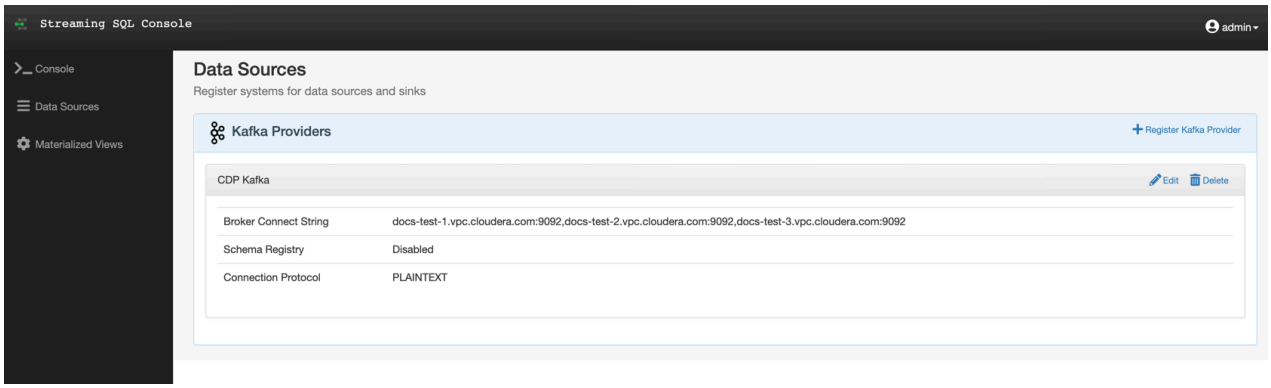
You can access Data Sources through the Streaming SQL Console:

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.

   The Streaming SQL Console opens up in a new window.
4. Click on Data Sources on the main menu.

   You are redirected to the Data Sources page.

The data source you register will be listed on the Data Sources page. You can register a data source by clicking on the Register Kafka Provider button. You can also Edit and Delete the already registered data sources.



Kafka data source integration

When integrating Kafka with SQL Stream Builder, you are able to use the Kafka specific syntaxes to customize your SQL queries based on your deployment and use case.

Timestamps

Timestamps are supported in Kafka as part of the messageformat. This is exposed to SQL Stream Builder through the virtual column eventTimestamp. The time in the message is the time when the message was created. SQL Stream Builder is also aware of the current_timestamp which is the current timestamp at query execution start time.

You can construct the queries in the following way:

```
SELECT sensor_name
```

```
FROM sensors
WHERE eventTimestamp > current_timestamp;
```

You can also use the timestamp in windowing queries in the following way:

```
SELECT SUM(CAST(amount AS numeric)) AS payment_volume,
CAST(TUMBLE_END(eventTimestamp, interval '1' hour) AS varchar) AS ts
FROM payments
GROUP BY TUMBLE(eventTimestamp, interval '1' hour);
```

For more information, see the SQL Syntax Guide.

Assigning keys for output

If using a Kafka as a sink, keys can be assigned using the special alias _eventKey. This alias specifies the column as the partition key using the Kafka driver.

```
SELECT sensor_name AS _eventKey --sensor_name becomes the key in the output
kafka topic
FROM sensors
WHERE eventTimestamp > current_timestamp;
```

Performance & Scalability

You can achieve high performance and scalability with SQL Stream Builder, but the proper configuration and design of the source Kafka topic is critical. SQL Stream Builder can read a maximum of one thread per Kafka partition. You can achieve the highest performance configuration when setting the SQL Stream Builder threads higher or equal than the number of Kafka partitions. If the number of partitions is less than the SQL Stream Builder threads, then SQL Stream Builder has idle threads and messages show up in the logs indicating as such. For more information about Kafka partitioning, see the Kafka partitions documentation.

Kafka message header access

You can write custom headers to Kafka which are used for metadata, sometimes routing information, filtering, and so on. SQL Stream Builder has access to the header information using the input transforms functionality. For more information about Input Transforms, see the Input Transforms section.

The following attributes are supported in the headers:

```
message.topic
message.key
message.value
message.headers
message.offset
message.partition
```

For example, an input transformation could be expressed as the following:

```
var out = JSON.parse(record);
out['topic'] = message.topic;
out['partition'] = message.partition;
JSON.stringify(out);
```

For which you define a schema as follows:

```
{
  "name": "myschema",
  "type": "record",
  "namespace": "com.cloudera.test",
  "fields": [
```

```
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "topic",
      "type": "string"
    },
    {
      "name": "partition",
      "type": "string"
    }
  ]
}
```

The attribute message.headers is an array that can be iterated over:

```
var out = JSON.parse(record);
var header = JSON.parse(message.headers);
var interested_keys = ['DC'];               // should match schema definitio
n

out['topic'] = message.topic;
out['partition'] = message.partition;

Object.keys(header).forEach(function(key) {
    if (interested_keys.indexOf(key) > -1){  // if match found for schema, s
et value
        out[key] = header[key];
    }
});

JSON.stringify(out);
```

For which you define a schema as follows:

```
{
  "name": "myschema",
  "type": "record",
  "namespace": "com.cloudera.test",
  "fields": [
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "topic",
      "type": "string"
    },
    {
      "name": "partition",
      "type": "string"
    },
    {
      "name": "DC",
      "type": "string"
    }
  ]
}
```

Note: Dynamic schema creation is not supported.

Adding Kafka as Data Sources

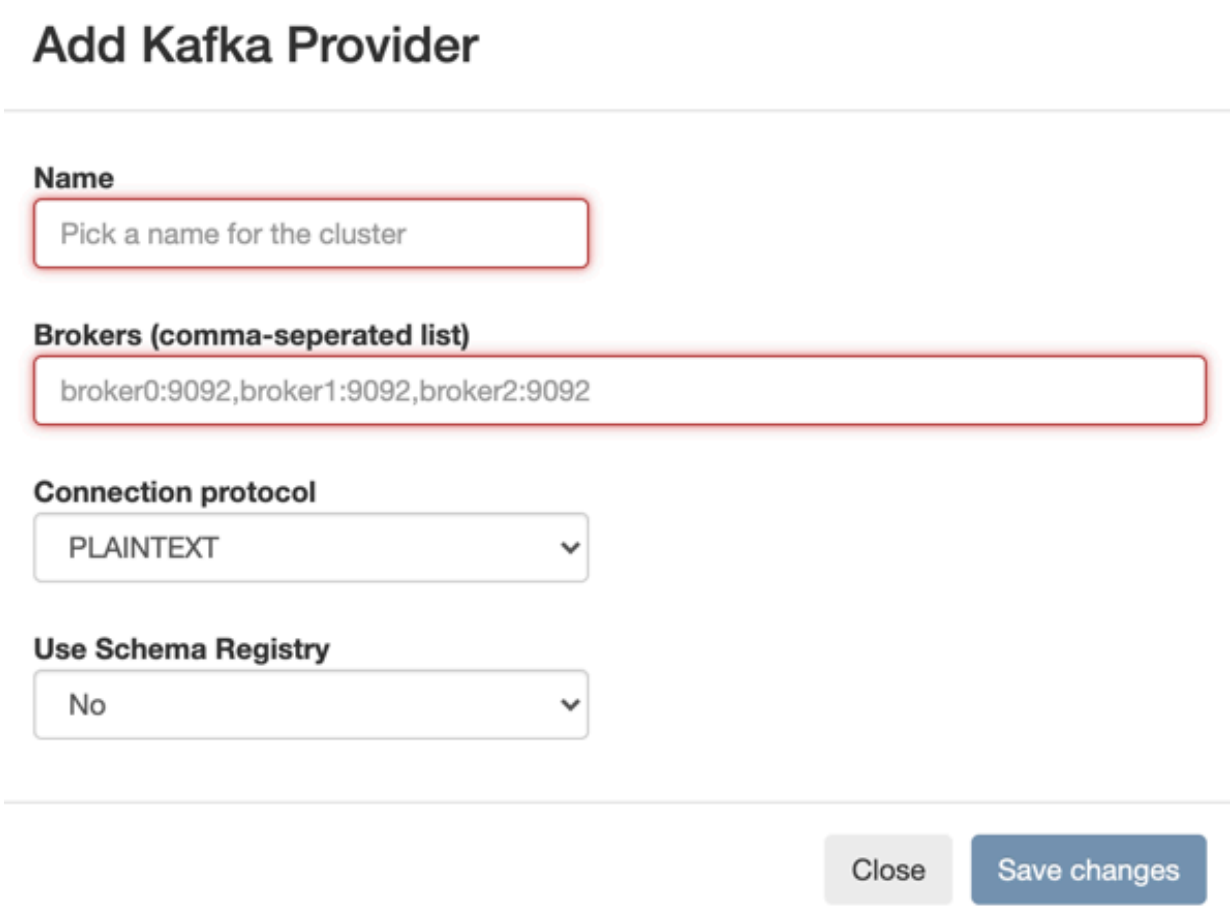
After installing Kafka as a service on your cluster, you can register the Kafka as a data source to use it as a Virtual Table in SQL Stream Builder (SSB).

Before you begin

- Make sure that you have Kafka service on your cluster.
- Make sure that you have the right permissions set in Ranger.
- In case you want to use Schema Registry, make sure that you have Schema service on your cluster.
- In case you want to use Schema Registry, make sure to add a schema before registering Kafka as a data source.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.
   The Streaming SQL Console opens up in a new window.
4. Click on Data Sources from the main menu.
5. Click on Register Kafka Provider.
   The Add Kafka Provider window appears.



6. Add a Name to your Kafka source.

7. Add the broker host name(s) to Brokers.

   You need to copy the Kafka broker name(s) from Cloudera Manager.

   a) Go to your cluster in Cloudera Manager.
   b) Click on Kafka from the list of Services.
   c) Click on Instances.
   d) Copy the Hostname of the Kafka broker(s) you want to use.
   e) Go back to the Add Kafka Provider page.
   f) Paste the broker hostname to the Brokers field.

      Note: You can add more than one broker hostname by separating them by commas.

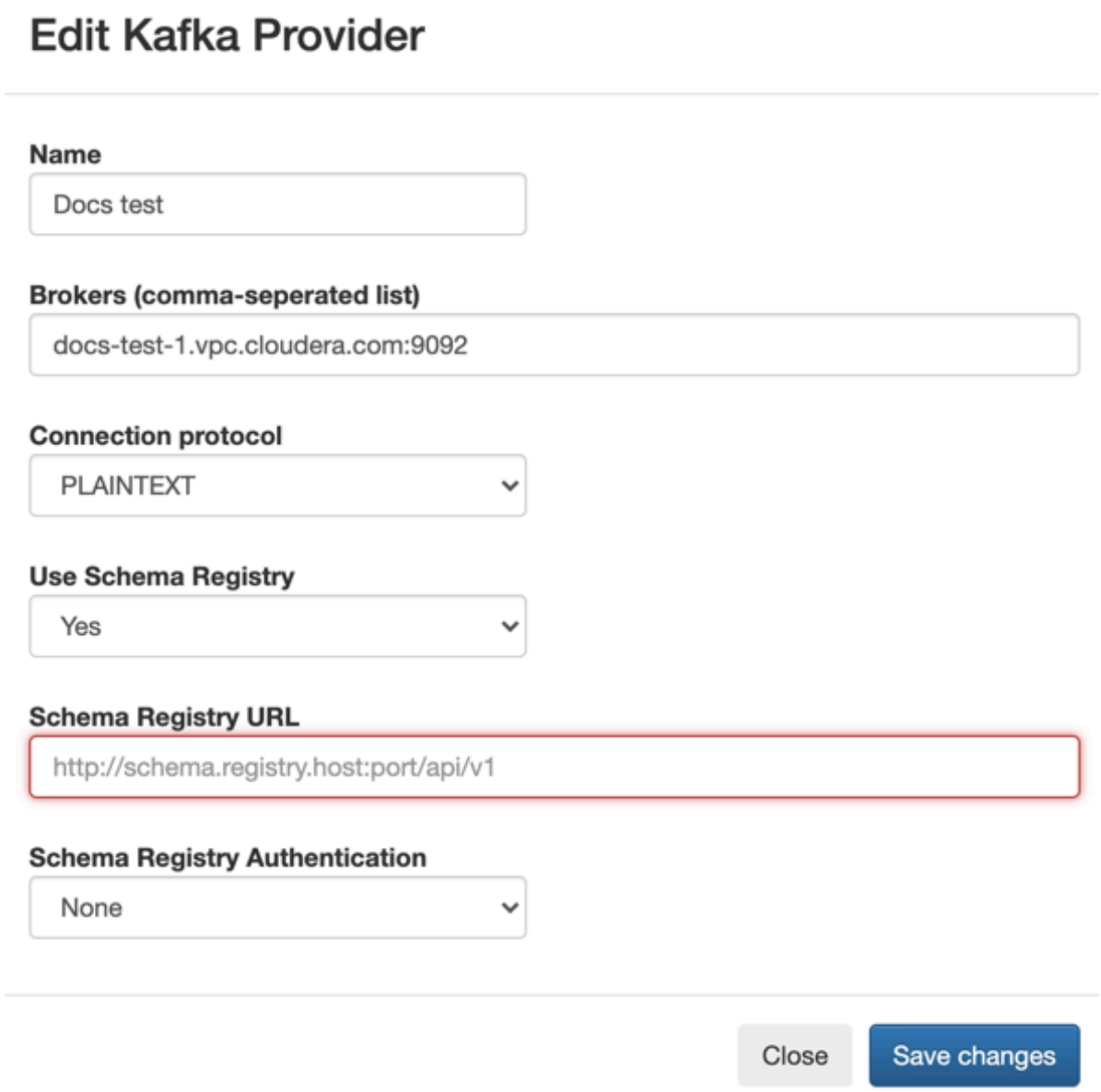
   g) Add the default Kafka port after the hostname(s).

      Example:

      ```
      docs-test-1.vpc.cloudera.com:9092,
      docs-test-2.vpc.cloudera.com:9092
      ```

8. Select the Connection protocol.

9. Select if you want to use Schema Registry.
   a) If you select No, click on Save Changes.
      Kafka is registered as a data source, and listed under Kafka Providers on the Data Sources page.
   b) If you select Yes, the configurations of Schema Registry appear.



   c) Add the Schema Registry URL.
      1. Go to your cluster in Cloudera Manager.
      2. Select Schema Registry from the list of services.
      3. Click on Instances.
      4. Copy the Hostname of Schema Registry.
      5. Add the default prt of Schema Registry after the hostname.

         Example:

         ```
         http://<schemaregistry-url>:7788/api/v1
         ```

   d) Select the Authentication method for Schema Registry.

1. If you select None, click on Save Changes.

   The registered Kafka data source is connected to Schema Registry.
2. If you select Basic, you need to provide the authentication username and password. After providing the information, click Save Changes.
3. If you select SSL, you need to provide the Trustore location and password. After providing the information, click Save Changes.

   Note: You do not need to provide the Trustore location and password in case it is configured for Schema Registry in Cloudera Manager.

Results

You have registered Kafka as a data source. You are able to select the registered Kafka as a virtual table source or sink.

Managing registered data sources

You can edit and delete the already registered data sources if you need to change their configurations or if you no longer need them.

Editing registered data sources

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.

   The Streaming SQL Console opens up in a new window.
4. Click on Data Sources from the main menu.
5. Search for the Kafka data source you want to modify.
6. Click on Edit.

   The Edit Kafka Provider window appears.
7. Change the Kafka settings as required.
8. Click Save Changes.

Deleting registered data sources

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.

   The Streaming SQL Console opens up in a new window.
4. Click on Data Sources from the main menu.
5. Search for the Kafka data source you want to modify.
6. Click on Delete.
7. Click on Confirm to delete the data source.

Using Virtual Tables

Virtual Tables are the interface for inputs (sources) and outputs (sinks) in SQL Stream Builder. Sources and sinks are configured separately, and may require a data source to be configured before they can be selectable. Data is continuously processed via the SQL Stream Builder SQL job from the source and the results of the query are continuously processed to the sink.

Virtual Table Sources

A Virtual Table Source is a logical definition of the data source that includes the location and connection parameters, a schema, an optional Input Transform, and any required, context specific configuration parameters.

The FROM clause run in SQL query specifies the Virtual Table Source. If you are performing a join, then each table specified in the SQL statement must reference a Virtual Table Source. This can be the same source multiple times, or a self-join.

```
SELECT
lat,lon
FROM
airplanes -- the name of the virtual table source
WHERE
icao <> 0;
```

Kafka

In this release of SSB, only Kafka is available as a virtual table source. To use Kafka as a source, you need to create a Kafka topic from which SSB reads the incoming data from.

Schema

Schema is defined for a given source when the source is created. When adding a schema you can either paste it to the Schema Definition field or click the Detect schema button to identify the schema used on the generated data. You can use JSON or AVRO as formats. The AVRO format can only be used when using Schema Registry.

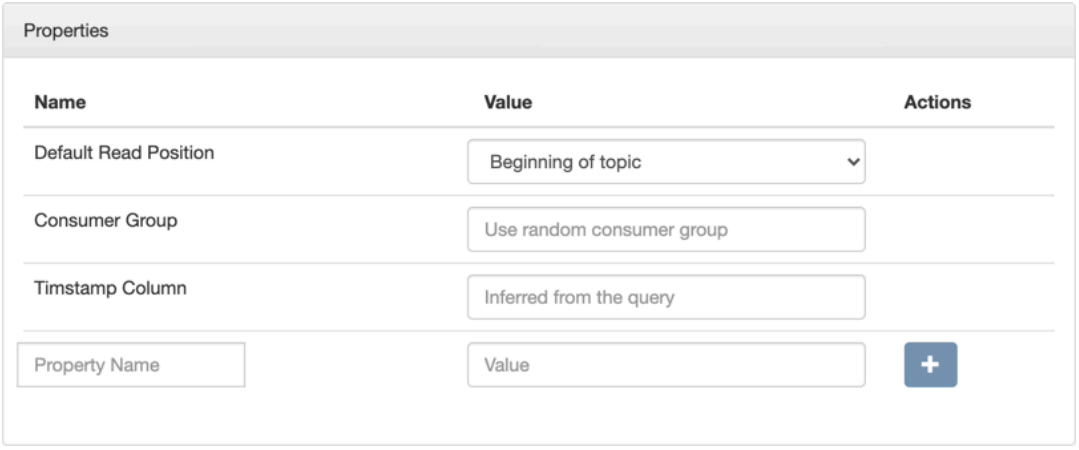
Input Transform

You can apply input transformations on your data when adding a virtual table source to clean or arrange the data coming from the source using javascript functions.

For more information about Input Transform, see the Creating input transformations document.

Properties

You can specify certain properties to define your source in detail. You can add customized properties additionally to the default ones. To create properties, you need to give a name to the property and provide a value for it, then click Actions.



Virtual Table Sinks

When you execute a query, the results go to the Virtual Table Sink that you selected in the SQL window. This allows you to create aggregations, filters, joins, and so on, and then route the results to a sink. The schema for the results is the schema that you created when you ran the query.

Selecting a sink for your SQL job is optional. You can output data to the screen when composing a SQL statement, or when configuring a Materialized View. This way data is not sent to a sink directly. To create a chain of SQL Stream Builder jobs, you can select a source topic from a different SQL job for a sink to process data in pieces.

Sampling the results to your browser allows you to inspect the queried data and iterate on your query. You can sample 100 rows in the Results tab by clicking on the Sample button in the Console. In case you do not add any sink to the SQL job, the results automatically appear in the Results tab.

Based on the type of Virtual Table Sink, you need to configure a data source before creating the sink. You can specify the Virtual Table Sink either before running a SQL job or when running a SQL job. The results from the SQL Stream Builder SQL query are sent to the Virtual Table Sink continuously.

Webhook sink

The webhook sink is useful for sending data to HTTP endpoints through POST or PUT.

Kafka sink

Kafka sinks are an output virtual table to send the data resulting from the query to. The data of a Kafka sink is in JSON format.

Schema

For sinks, the schema is defined by the table structure in the SQL statement, both for column names and datatypes. SSB supports aliases for column names like SELECT bar AS foo FROM baz as well as CAST(value AS type) for type conversions. SSB supports meta commands like show tables and describe <table> that make it easier to visualize the schema used on the data.

Creating a Kafka source

You can use the registered Kafka source to create a Kafka Virtual Table Source. You need to add a Kafka topic and choose to use Schema Registry to have a Kafka source for your SQL Stream job.

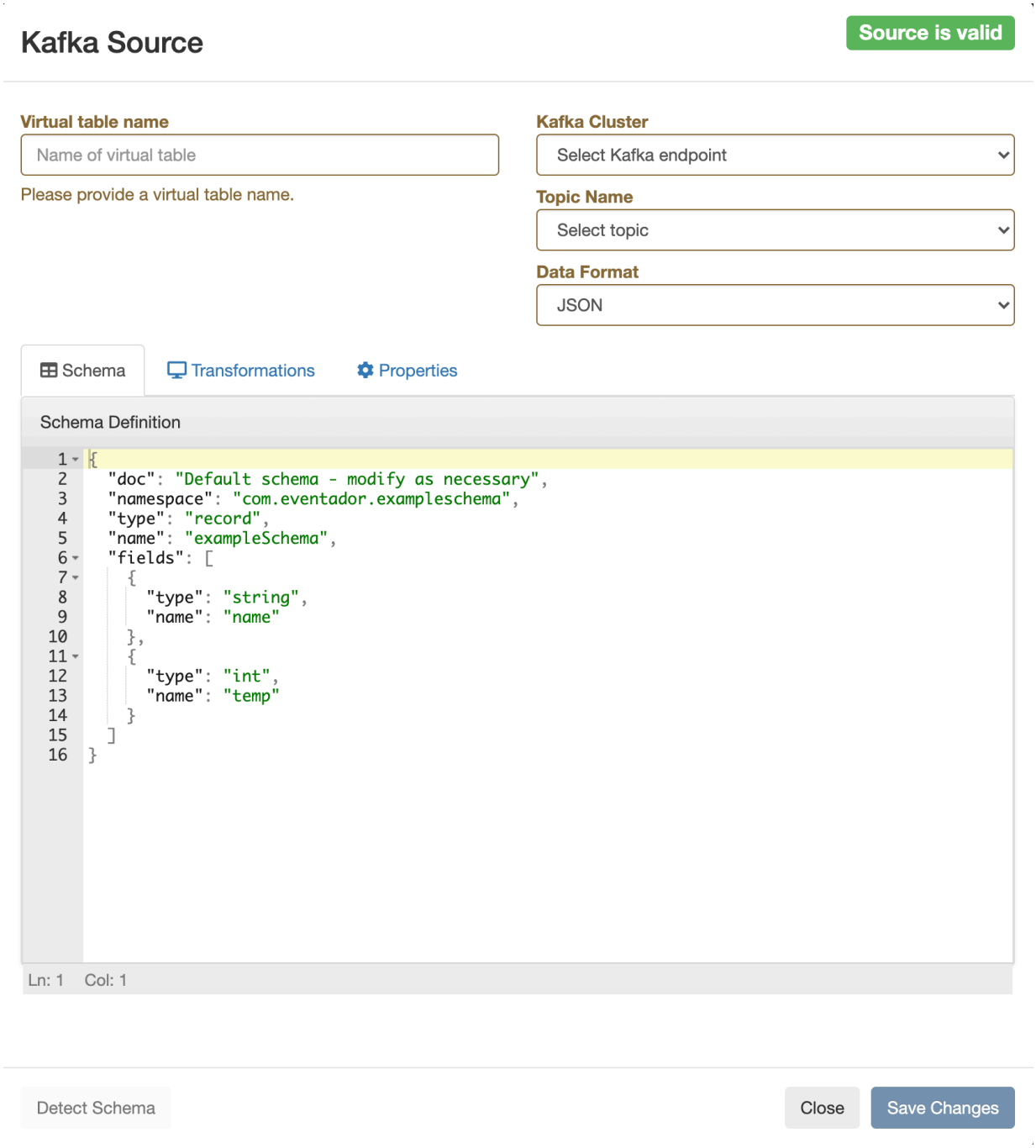
Before you begin

- Make sure that you have created a Kafka topic.
- Make sure that you have the right permissions set in Ranger.
- Make sure there is generated data in the Kafka topic.
- If you are using Schema Registry, you have added a schema.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.
   The Streaming SQL Console opens up in a new window.
4. Select Console from the left hand menu.
5. Select the Virtual Tables tab.
6. Select the Virtual Table Source sub-tab.

7. Select Add Source > Apache Kafka .
   The Kafka Source window appears.



8. Provide a name to the Virtual Table.

   Note: You will use this name in the FORM clause when running the SQL statement.

9. Select a registered Kafka data source.
10. Select a Kafka topic from the list.

    Note: The automatically created topics for the websocket output is also listed here. Select the topic you want to use for the SQL job.

11. Select the Data Format.
    - You can select JSON as data format.
    - You can select Avro as data format.

    Note: You can only select the Avro format when Schema Registry is used.

12. Determine the schema for the Virtual Table Source.
    a) Add a customized schema to the Schema Definition field.
    b) Click Detect Schema to read a sample of the messages and automatically infer the schema.

    Note: If there are no messages in the topic, then no schema will be inferred.

    c) Click Detect Schema if you are using Schema Registry.
13. Customize your Virtual Table Source.
    a) Configure an Input Transform, add the code using the Transformations tab.
    b) Configure any Kafka properties required using the Properties tab.
14. Select Save Changes.

Results

The Kafka Virtual Table Source is ready for queries to be run. You can check its configuration by a DESC <ta blename> command in the Compose tab, and selecting Execute. You can use the Kafka sink for your SQL job by declaring in with the FORM clause in the SQL statement.

Creating a Kafka sink

You can use the registered Kafka source to create a Kafka Virtual Table Sink. You only need to specify the Kafka topic where the data should be generated when executing your SQL Stream job.

Before you begin

- Make sure that you have created a Kafka topic.

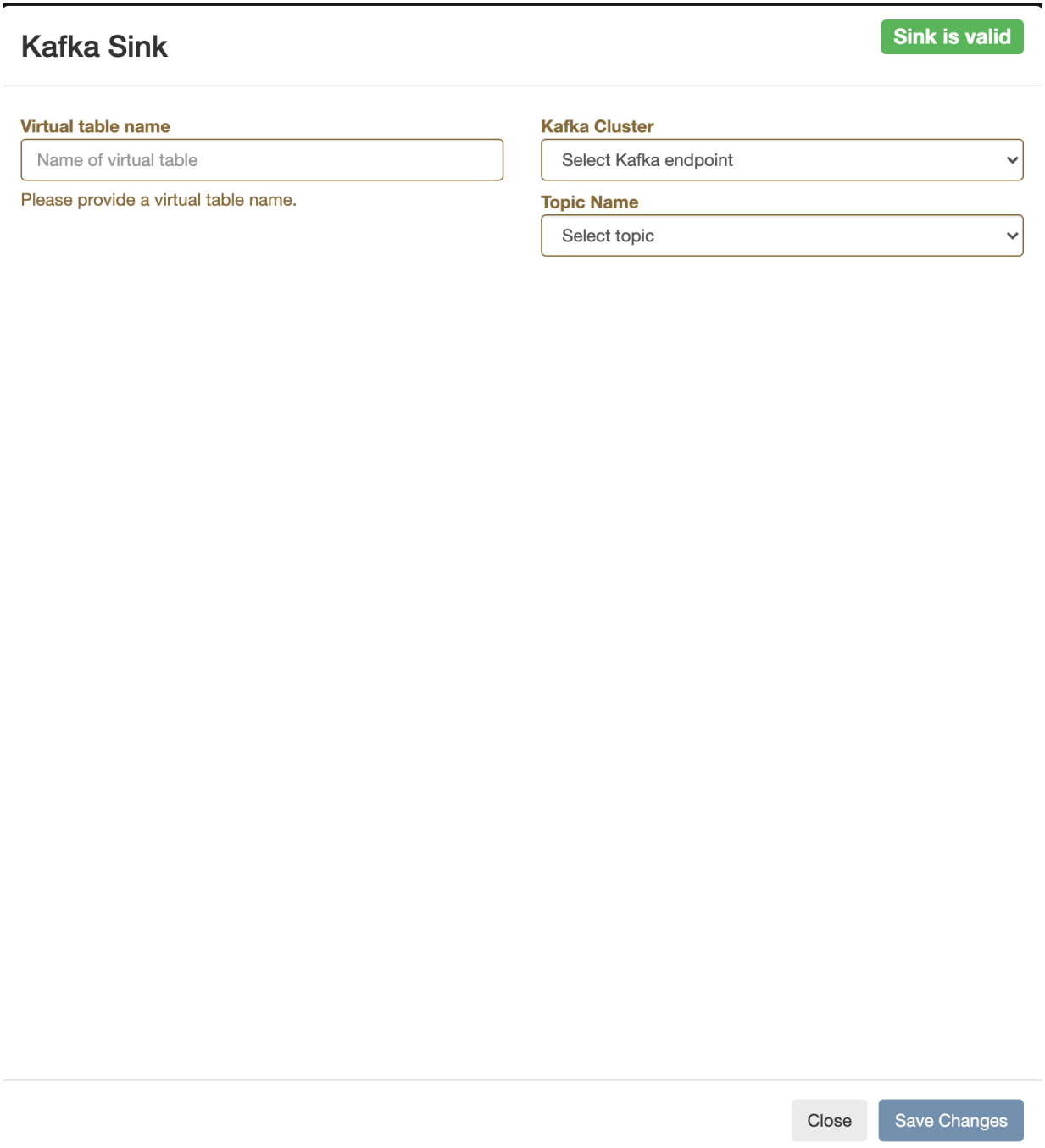
  Important: When creating the topic for the Kafka sink, make sure to not use log compaction as it can cause the SQL job to fail.

- Make sure that you have the right permissions set in Ranger.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.
   The Streaming SQL Console opens up in a new window.
4. Select Console from the main menu.
5. Select the Virtual Tables tab.
6. Select the Virtual Table Sink sub-tab.

7. Select Add Sink > Apache Kafka .
   The Kafka Sink window appears.



8. Provide a name to the Virtual Table.

   Note: You will select the provided name when adding a sink on the Compose tab.

9. Select a Kafka topic from the list.

   Note: The automatically created topics for the websocket output is also listed here. Select the topic you want to use for the SQL job.

10. Click Save Changes.

Note:

All output is JSON encoded data in {column>:<value>} format.

Results

The Kafka Virtual Table Sink is ready for queries to be run. You can add the created sink on the Compose tab by selecting its name from the list of sinks.

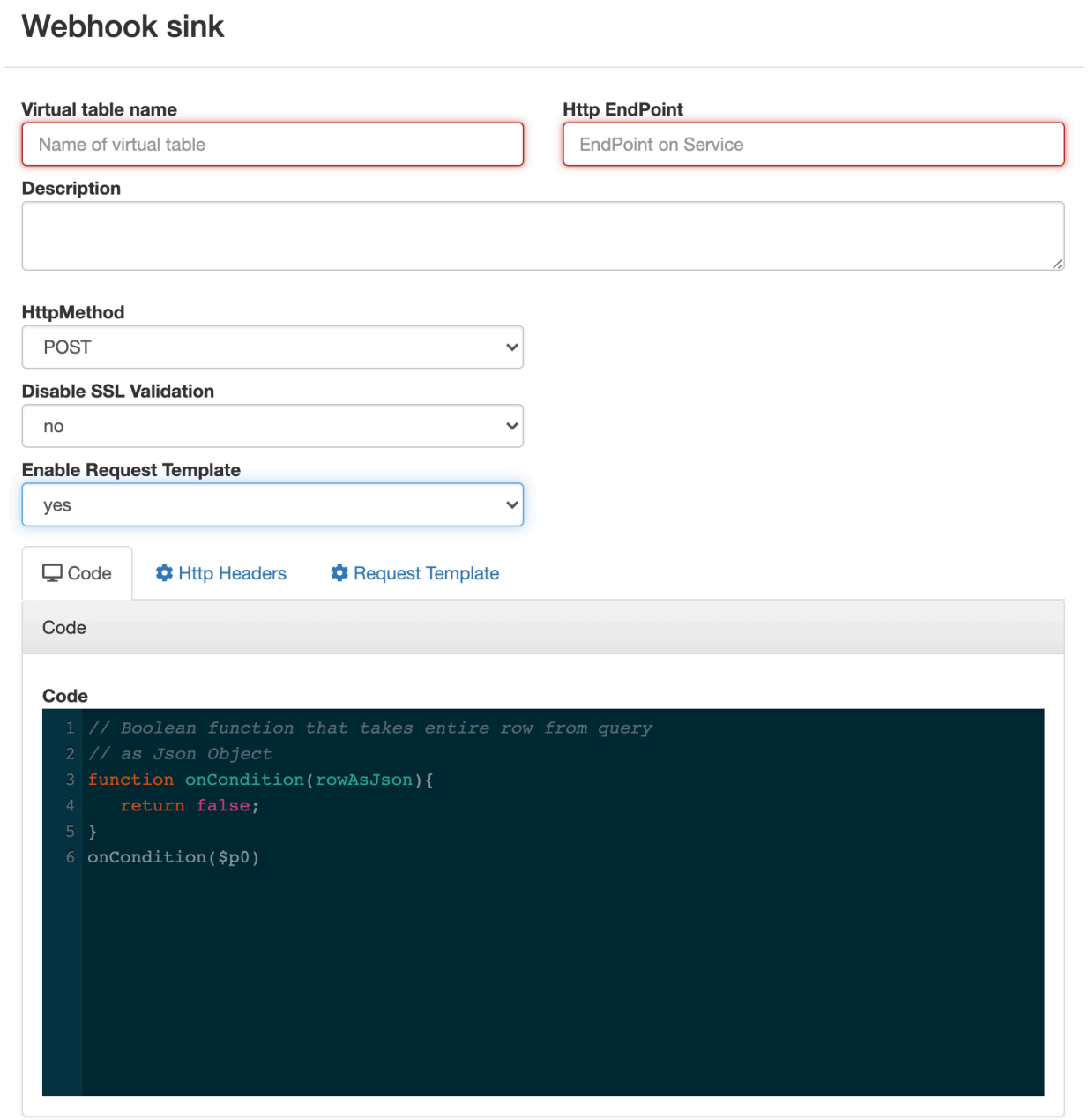
Creating a Webhook sink

You can configure the webhook sink to perform an HTTP action per message (default) or code created that controls the frequency (for instance, every N messages). When developing webhook sinks, it is recommended to check your webhook before pointing at your true destination.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click on SQL Stream Builder from the list of Services.
3. Click on SQLStreamBuilder Console.
   The Streaming SQL Console opens up in a new window.
4. Select Console from the main menu.
5. Select the Virtual Tables tab.
6. Select the Virtual Table Sink sub-tab.

7. Select Add Sink > Webhook .
   The Webhook Sink window appears.



8. Provide a name to the Virtual Table.
9. Enter an HTTP EndPoint. The endpoint must start with http:// or https://.

   Note: You can use hookbin as an initial testing of the webhook sink. Paste the hookbin endpoint into the text field, and inspect the output on the hookbin site. Once you have the right output result, then point it at your final endpoint.
10. Add a Description about the webhook sink.
11. Select POST or PUT in the HTTP Method select box.
12. Choose to Disable SSL Validation, if needed.

13. Enable Request Template, if needed.
    a) If you selected Yes, then the template defined in the Request Template tab is used for output.

       This is useful if the service you are posting requires a particular data output format. The data format must be valid JSON, and use "${columnname}" to represent fields. For example, a template for use with Pagerduty looks like this:

```
{
   "incident":{
      "type":"incident",
      "title":"${icao} is too high!",
        "body":{
   "type":"incident_body",
         "details":"Airplane with id ${icao} has reached an altitude of
 ${altitude} meters."
 }
    }
}
```

14. In the Code editor, you can specify a code block that controls how the webhook outputs the data.

    For a webhook that is called for each message the following code would be used:

```
// Boolean function that takes entire row from query as Json Object
function onCondition(rowAsJson)
{return true;   // return false here for no-op, or plug in custom
        logic}
onCondition($p0)
```

Note: The rowAsJson is the result of the SQL Stream query being run in {"name":"value"} format.

15. Add HTTP headers via the HTTP Headers tab, if needed.

    Headers are name:value header elements. For instance, Content-Type:application/json, etc.

16. Click Save Changes.

Results

The Webhook Virtual Table Sink is ready for queries to be run. You can add the created sink on the Compose tab by selecting its name from the list of sinks.

Managing time in SSB

Time attributes define how streams are processed in time. There are two high-level options for providing time attributes to your SQL Stream Builder queries.

Source-provided timestamps

Source-provided timestamps are injected directly into the data stream by the source connector. For example, when using a Kafka data source, the timestamp extracted from the Kafka message header is embedded in the data stream by default, and exposed through an eventTimestamp column. In the following query, the built-in eventTimestamp column is used to window the query based on the timestamp recorded in the source Kafka topic.

```
SELECT flight_number, flight_origin, flight_destination,
  TUMBLE_END(eventTimestamp, INTERVAL '5' MINUTE) AS window_end_timestamp
FROM airplane_flights
GROUP BY TUMBLE(eventTimestamp, INTERVAL '5' MINUTE)
```

The following query uses the source-provided timestamp from two virtual tables to perform a streaming join on multiple Kafka topics:

```
SELECT a.web_order_id, a.product_name, a.order_date
 b.next_shipment_time
FROM online_orders a, shipment_events b
WHERE a.shipping_type = 'Priority'
 AND a.eventTimestamp BETWEEN b.eventTimestamp - INTERVAL '1' HOUR AND b.ev
entTimestamp
```

User-provided timestamps

The user can also specify timestamps contained in the data stream itself.

Requirements:

- The timestamp needs to be contained in a column of type long.
- The timestamp needs to be in epoch format (for example, milliseconds since Jan 1, 1970).

If your schema includes a field called timestamp_ms, it is possible to construct a query as the following example:

```
SELECT *
FROM airplane_flights a
WHERE flight_number IS NOT NULL
GROUP BY HOP(a.timestamp_ms, INTERVAL '15' SECOND, INTERVAL '5' SECOND), fl
ight_number
```

Note: If your timestamp column is called timestamp, it is required to enclose the column name in quotation marks as it is a reserved word.

If an invalid timestamp is encountered in the stream (for example, NaN), then the timestamp of the message defaults to 0, causing the message to be excluded from the current window.

If your data does not include a timestamp in a suitable format, it is possible to compute a new timestamp column from another existing column (using input transformations).