Cloudera Streaming Analytics 1.4.1

Using SQL Stream Builder

Date published: 2019-12-17
Date modified: 2021-07-20

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.

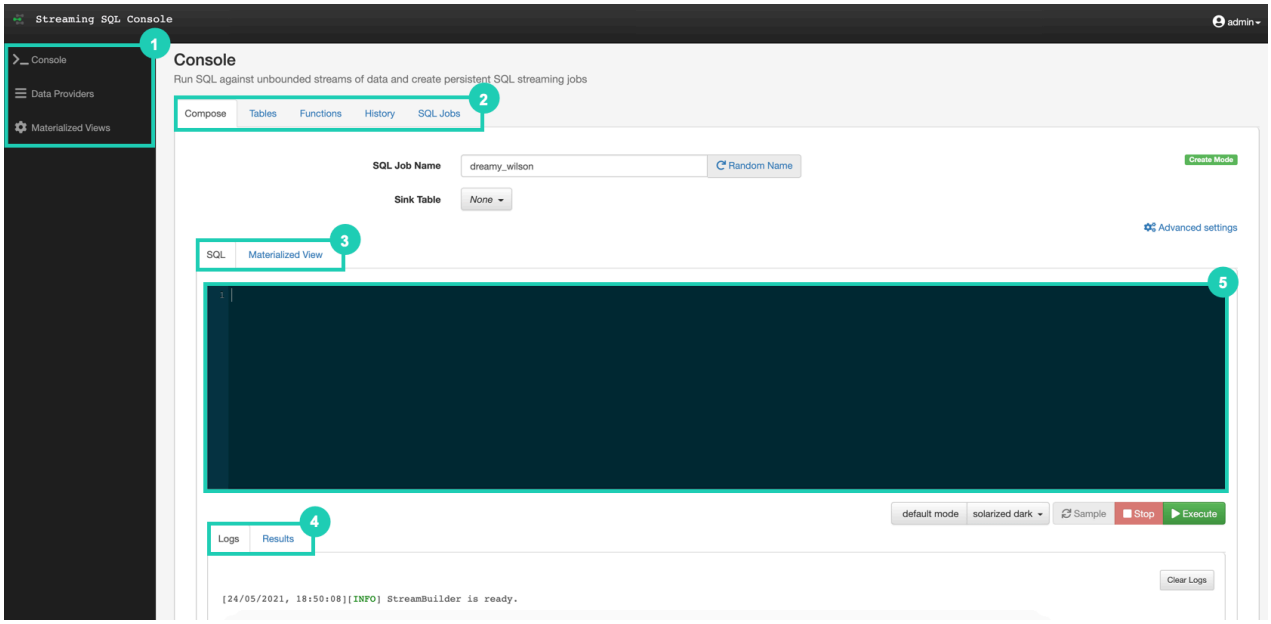
You can access the Streaming SQL Console through Cloudera Manager:

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

The Streaming SQL Console opens 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 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 Providers - Adding and managing data providers for Tables.
   - Materialized Views - Setting and managing API keys.
2. The main tabs consist of the following:
   - Compose - By default, the Compose tab is selected on the Console. You can create your SQL queries on the Compose tab.
   - Tables - You can view and register 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 - By 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.

Registering Data Providers in SSB

Data Providers are a set of data endpoints to be used as sources, sinks and catalogs. Data Providers allow you to connect to an already installed component on your cluster, then use that provider for adding tables in SQL Stream Builder.

You can access the Data Providers page through the Streaming SQL Console:

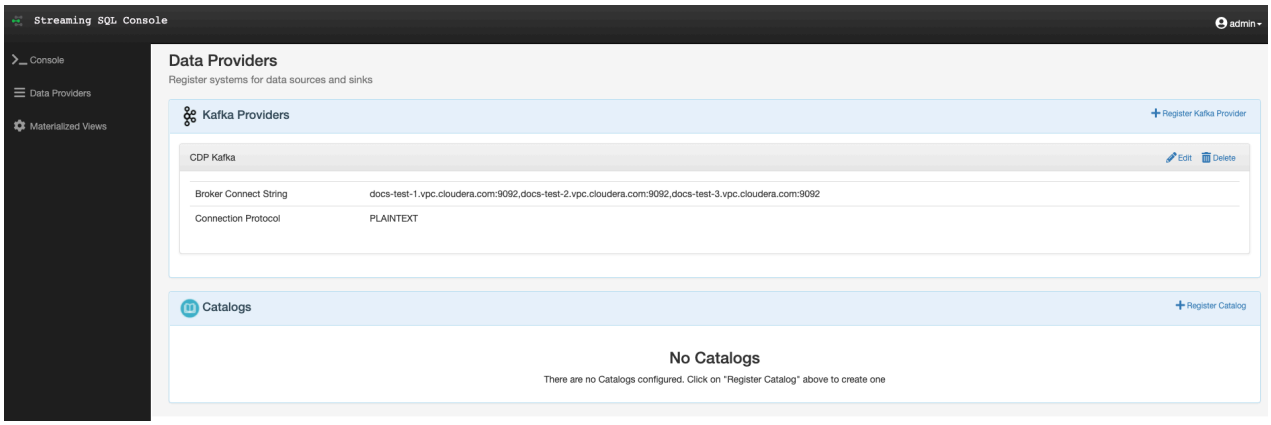
1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.

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

   You are redirected to the Data Providers page.

You can register Kafka as a data provider, or Kudu, Hive and Schema Registry as a catalog. When registering the components, SSB can access the already existing topics from Kafka, tables from Kudu and Hive, and the schema in Schema Registry. This also means that when you update a data provider, for example add new topics, tables and schemas, SSB automatically detects the changes.

You can also manage your data providers after registering them. You can Edit the providers if there is any change in the connection. You can also Delete them when you no longer need the specific provider.



Adding Kafka as Data Provider

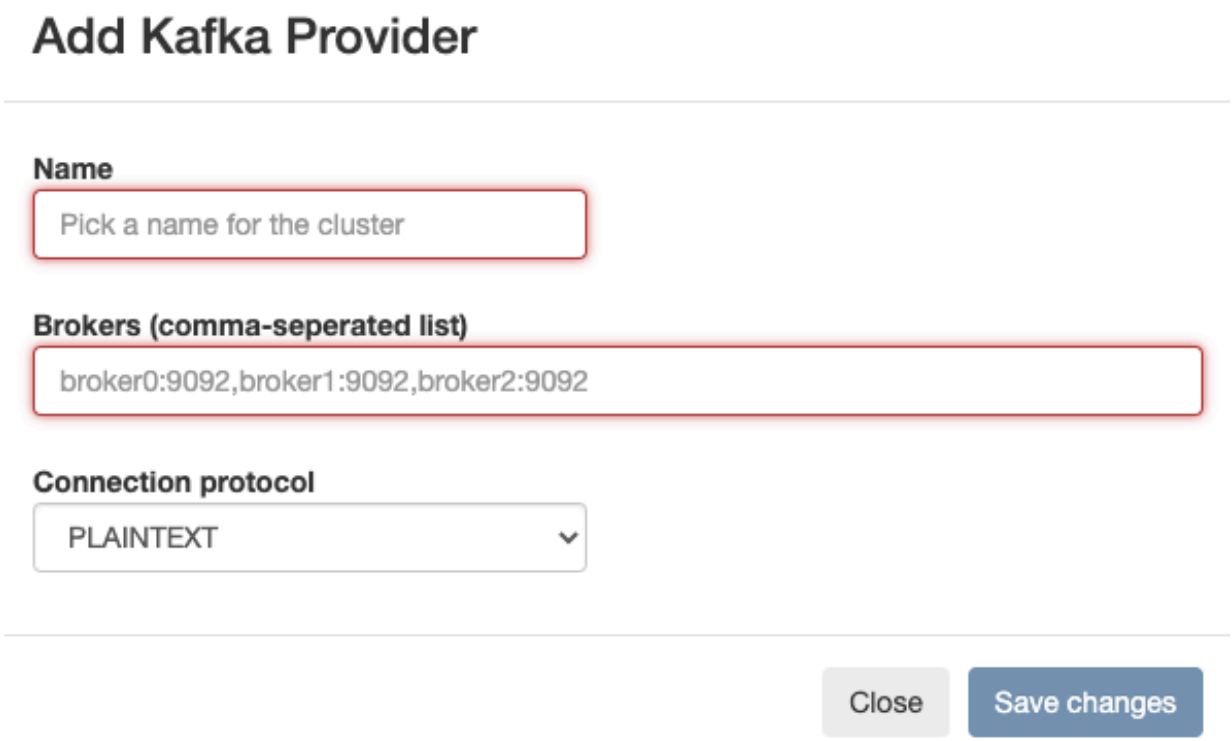
After installing Kafka as a service on your cluster, you can register Kafka as a Data Provider to use it as a Table in SQL Stream Builder (SSB).

Before you begin

- Make sure that you have Kafka service on your cluster.
- Make sure that you have the right permissions set in Ranger.

Procedure

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



6. Add a Name to your Kafka provider.
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 Kafka from the list of services.
   c) Click 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.

   The connection protocol must be the same as it is configured for the Kafka cluster in Cloudera Manager.

   You can choose from the following protocols:

   a) Select Plaintext, and click Save Changes.
   b) Select SSL, and click Save Changes.
   c) Select SASL/SSL, and choose an SASL Mechanism.

      1. Select Kerberos, and provide the Kafka Truststore location. Click Save Changes.
      2. Select Plain, and provide the SASL username and password. Click Save Changes.

Results

You have registered Kafka as a data provider to be able to add Kafka as a table in your SQL query. The already existing Kafka topics can be selected when adding Kafka as a table.

Integration with Kafka

The Kafka and SQL Stream Builder integration enables you to use the Kafka-specific syntax to customize your SQL queries based on your deployment and use case.

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 equal to or higher than the number of Kafka partitions.

If the number of partitions is less than the number of 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 record metadata access

There are cases when it is required to access additional metadata from the Kafka record to implement the correct processing logic. SQL Stream Builder has access to this 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:

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

For example, an input transformation can 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 manually, or use the Detect Schema feature:

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

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

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

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

out['topic'] = record.topic;
out['partition'] = record.partition;
Object.keys(header).forEach(function(key) {
    if (interested_keys.indexOf(key) > -1){  // if match found for schema,
set 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"
    }
  ]
}
```

Related Information

Creating Input Transforms

Adding Schema Registry as Catalog

After installing Schema Registry as a service on your cluster, you can add it as a Catalog in SQL Stream Builder (SSB) to use it in Flink DDL.

Before you begin

- Make sure that you have Schema Registry service on your cluster.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Click Register Catalog.
   The Add Catalog window appears.
6. Add a Name to your catalog.
7. Select Schema Registry from the Catalog Type drop-down.
8. Select the Kafka cluster you registered as Data Provider.
9. Add the Schema Registry URL.
   a) Go to your cluster in Cloudera Manager.
   b) Select Schema Registry from the list of services.
   c) Click on Instances.
   d) Copy the Hostname of Schema Registry.
   e) Add the default port of Schema Registry after the hostname.
   Example:

   ```
   http://docs-test-1.vpc.cloudera.com:7788/api/v1
   ```

10. Select if you wan to enable TLS for Schema Registry.
    a) If yes, provide the Schema Registry Truststore location and password.
11. Click on Add Filter.
    a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
12. Click on Validate.
13. If the validation is successful, click Add Tables.

Results

Schema Registry is added as a Catalog and ready to be used in Flink DDL. The already existing schemas in Schema Registry are automatically imported to SSB.

Adding Kudu as Catalog

After installing Kudu as a service on your cluster, you can add it as a Catalog in SQL Stream Builder (SSB) to use it in Flink DDL.

Before you begin

- Make sure that you have Kudu service on your cluster.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Click Register Catalog.
   The Add Catalog window appears.
6. Add a Name to your catalog.
7. Select Kudu from the Catalog Type drop-down.
8. Add the Host URL of Kudu Masters.
   a) Go to your cluster in Cloudera Manager.
   b) Select Kudu from the list of services.
   c) Click on Instances.
   d) Copy the Hostname of the Master Default Group.
   e) Add the default port of Kudu after the hostname.
      Example:

      ```
      http://docs-test-1.vpc.cloudera.com:7051
      ```
9. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
10. Click on Validate.
11. If the validation is successful, click Add Tables.

Results

Kudu is added as a Catalog and ready to be used in Flink DDL. The already existing tables in Kudu are automatically imported to SSB.

Adding Hive as Catalog

After installing Hive as a service on your cluster, you can add it as a Catalog in SQL Stream Builder (SSB) to use it in Flink DDL.

Before you begin

- Make sure that you have Hive service on your cluster.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Click Register Catalog.
   The Add Catalog window appears.
6. Add a Name to your catalog.
7. Select Hive from the Catalog Type drop-down.

8. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
9. Click on Validate.
10. If the validation is successful, click Add Tables.

Results

Hive is added as a Catalog and ready to be used in Flink DDL. The already existing tables in Hive are automatically imported to SSB.

Adding Custom Catalogs

You can use custom catalogs in cases when you need to add a catalog that is not predefined in the Streaming SQL Console, and when more advanced settings need to be specified for the catalogs. This case you need to provide the configuration directly to Flink with the custom catalog option.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Click Register Catalog.

   The Add Catalog window appears.
6. Add a Name to your catalog.
7. Select Custom from the Catalog Type drop-down.
8. Provide a Property Name and Value for the Catalog.
9. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
10. Click on Validate.
11. If the validation is successful, click Add Tables.

Results

The custom catalog is added and ready to be used in Flink DDL.

Managing registered Data Providers

You can edit or delete the registered Data Providers if you need to change their configurations or if you no longer need them.

Editing registered Data Providers

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

   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Search for the Kafka provider or catalog you want to modify.

6. Click Edit.

   The Edit Provider or Catalog window appears.
7. Change the settings as required.

   Note: You must validate the modified catalog before saving the changes.
8. Click Save Changes.

Deleting registered Data Providers

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

   The Streaming SQL Console opens in a new window.
4. Click Data Providers from the main menu.
5. Search for the Kafka provider or catalog you want to modify.
6. Click Delete.
7. Click Confirm to delete the provider or catalog.

Using Tables in SQL Stream jobs

The core abstraction for Streaming SQL is a Table which represents both inputs and outputs of the queries. SQL Stream Builder tables are an extension of the tables used in Flink SQL to allow a bit more flexibility to the users.

A Table is a logical definition of the data source that includes the location and connection parameters, a schema, and any required, context specific configuration parameters. Tables can be used for both reading and writing data in most cases. You can create and manage tables either manually or they can be automatically loaded from one of the catalogs as specified using the Data Providers section.

In SELECT queries the FROM clause defines the table sources which can be multiple tables at the same time in case of JOIN or more complex queries.

For example:

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

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

Supported tables in SSB

SSB supports different table types to ease development and access to all kinds of data sources. There are two main categories of tables, user defined tables and catalog tables. User defined tables need to be added manually and catalog tables are accessible automatically after registering a catalog provider.

User defined tables

The user defined tables need to be added manually using the Add Tables wizard of the Streaming SQL Console.

- Kafka Table

  Apache Kafka Tables represent data contained in a single Kafka topic in JSON or AVRO format. It can be defined using the Streaming SQL Console wizard. For more advanced use cases Kafka tables can be substituted by Flink DDL tables.
- Flink DDL Table

  Flink DDL tables represent tables created by using the standard Flink SQL CREATE TABLE/ CREATE VIEW syntax. This supports full flexibility in defining new or derived tables and views. You can either provide the syntax by directly adding it to the Flink DDL window or use one of the predefined DDL templates.
- Webhooks

  Webhooks can only be used as sink tables for SQL queries. The result of your SQL query is sent to a specified webhook.

Catalog tables

The catalog tables are automatically imported based on the catalog type you have registered on the Data Providers page. The following catalogs are supported in SSB:

- Schema Registry
- Kudu
- Hive
- Custom catalogs

Note: You cannot edit the properties of the already existing tables that are automatically imported from the catalogs. To distinguish between editable and not editable tables, in other words, user defined and catalog tables, the Edit and Delete table options are not available on the Tables page as the illustration shows below:

Creating Kafka tables

You can use the registered Kafka provider to create Kafka tables that can be used as source and sink in your SQL Stream jobs.

Before you begin

- Make sure that you have created a Kafka topic.

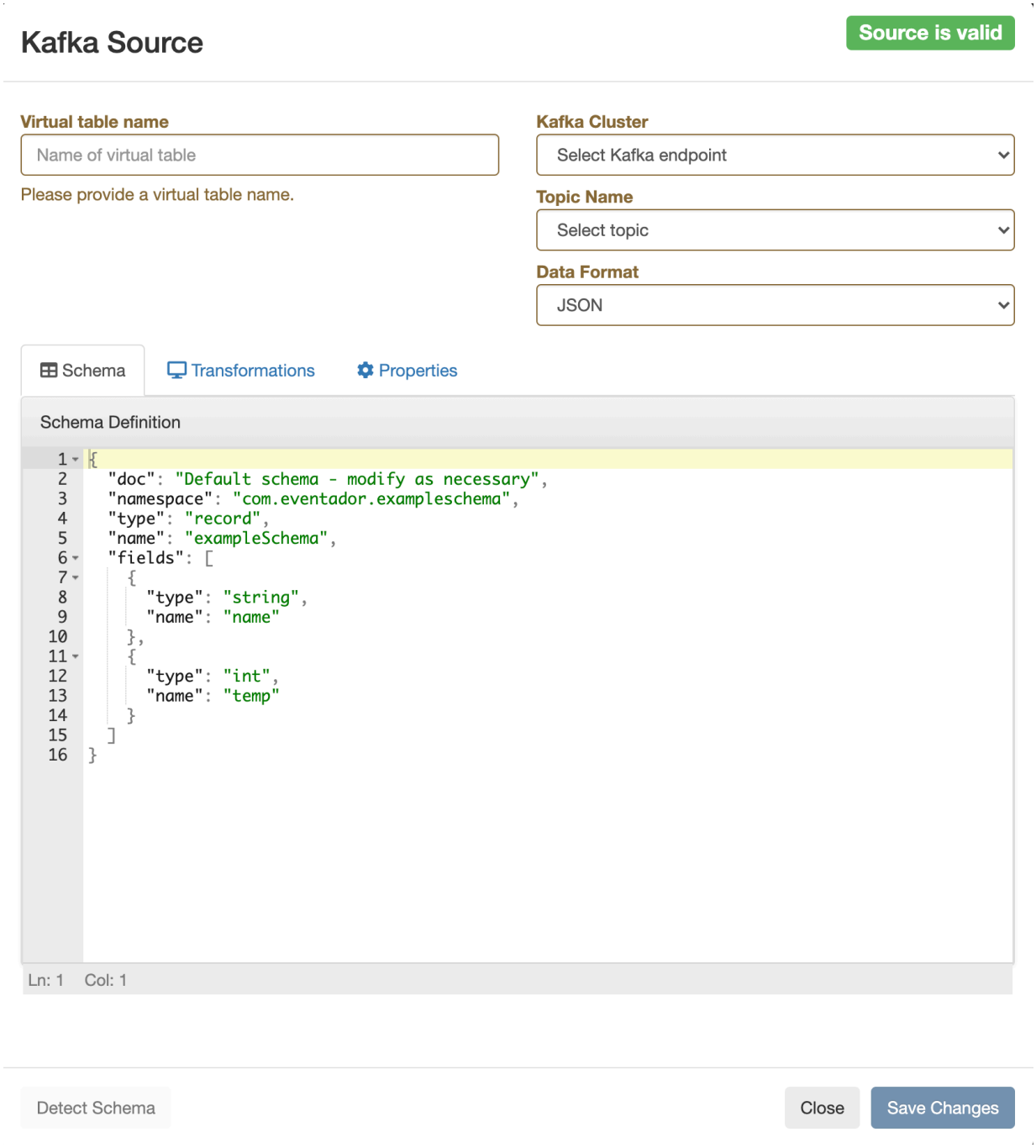
  

  Important: When creating the topic for the Kafka sink, make sure to not use log compaction as it can cause the SQL job to fail.
- Make sure there is generated data in the Kafka topic.
- Make sure that you have the right permissions set in Ranger.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Select Console from the left- side menu.
5. Go to the Tables tab.

6. Select Add table > Apache Kafka .
   The Kafka Table window appears.



7. Provide a Name for the Table.

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

8. Select a registered Kafka provider as Kafka cluster.
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. 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.

11. Determine the Schema for the Kafka table.
    a) Add a customized schema to the Schema Definition field.
    b) Click Detect Schema to read a sample of the JSON messages and automatically infer the schema.

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

12. Customize your Kafka Table with the following options:
    a) Configure the Event Time if you do not want to use the Kafka Timestamps.
        1. Unselect the checkbox of Use Kafka Timestamps.
        2. Provide the name of the Input Timestamp Column.
        3. Add a name for the Event Time Column.
        4. Add a value to the Watermark Seconds.
    b) Configure an Input Transform, add the code using the Transformations tab.
    c) Configure any Kafka properties required using the Properties tab.

    For more information about how to configure the Kafka table, see the Configuring Kafka tables section.
13. Select Save Changes.

Results

The Kafka Table is ready to be selected as a source or a sink for the SQL Stream job. To use Kafka as a source add it to the SQL query with the FROM statement. To use it as a sink, select the Kafka table when creating the SQL job from the Sink Table drop-down menu.

Configuring Kafka tables

The user defined Kafka table can be configured based on the schema, event time, input transformations and other Kafka specific properties.

Schema tab

Schema is defined for a given Kafka source when the source is created. The data contained in the Kafka topic can either be in JSON or AVRO format.

When specifying 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. The Detect Schema functionality is only available for JSON data.

If the schema of the Kafka table where the output data is queried is not known at the time of adding the table, you can select the Dynamic Schema option. This is useful when you want to insert data to the table, and there is no guarantee that the input schema matches with the output schema. If you select the Dynamic Schema option when adding a table, you can only use that table as a sink.

Note: If your schema contains a field named timestamp, this causes a schema validation error as timestamp is a reserved word used for Kafka internal timestamps.

Event Time tab

You can specify Watermark Definitions when adding a Kafka table. Watermarks use an event time attribute and have a watermark strategy, and can be used for various time-based operations. The Event Time tab provides the following properties to configure the event time field and watermark for the Kafka stream:

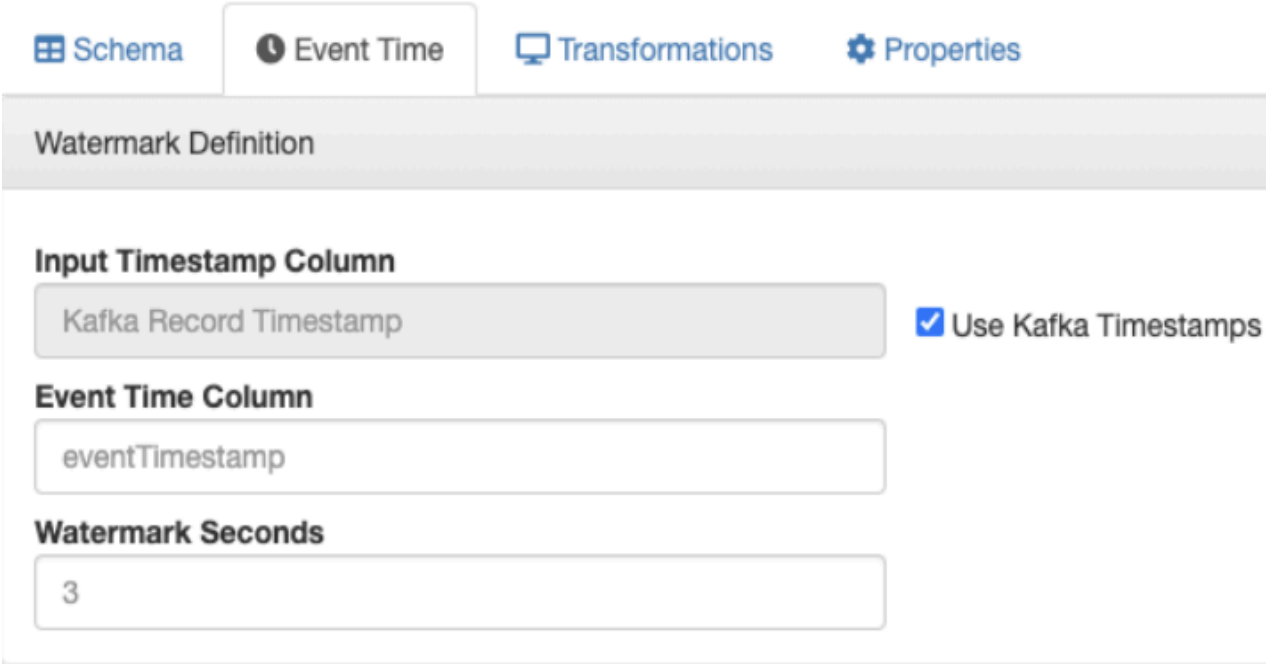
- Input Timestamp Column: name of the timestamp column in the Kafka topic from where the watermarks are mapped to the Event Time Column of the Kafka table
- Event Time Column: default or custom name of the resulting timestamp column where the watermarks are going to be mapped in the created Kafka table
- Watermark seconds: number of seconds used in the watermark strategy. The watermark is defined by the current event timestamp minus this value.

You have the following options to configure the watermark strategy for the Kafka tables:

- Using the default Kafka Timestamps setting
- Using the default Kafka Timestamps setting, but providing custom name for the Event Time Column
- Not using the default Kafka Timestamps setting, and providing all of the Kafka timestamp information manually
- Not using watermark strategy for the Kafka table

Using the default Kafka Timestamp setting

By default, the Use Kafka Timestamps checkbox is selected when you create the Kafka table. In the Event Time Column, the new event time field is extracted from the Kafka message header with the *'eventTimestamp'* predefined column name.



After saving your changes, you can view the created DDL syntax for the Table on the right side under the DDL tab. You can review the generated watermark strategy for your table that was set on the Watermark Definition tab.

The following DDL example shows the default setting of the Event Time Column and Watermark Seconds where the corresponding fields were not modified.

```
'eventTimestamp' TIMESTAMPS(3) METADATA FROM 'timestamp',
WATERMARK FOR 'eventTimestamp' AS 'eventTimestamp' - INTERVAL '3' SECOND
```

Using the default Kafka Timestamp setting with custom Event Time Column name

When you want to modify the timestamp field of the DDL from the stream itself, you must provide a custom name of the Event Time Column. You can also add a custom value to the Watermark Seconds. The following example shows that *'ets'* is the custom name for the Event Time Column, and *'4'* is the custom value for the Watermark Seconds.

The Event Time Column can only be modified if the following requirements are met for the timestamp field of the Input Timestamp Column:

- The column type must be "long".
- The format must be in epoch (in milliseconds since January 1, 1970).

The DDL syntax should reflect the changes made for the watermark strategy as shown in the following example:

```
'ets' TIMESTAMP(3) METADATA FROM 'timestamp',
WATERMARK FOR 'ets' AS 'ets' - INTERVAL '4' SECOND
```

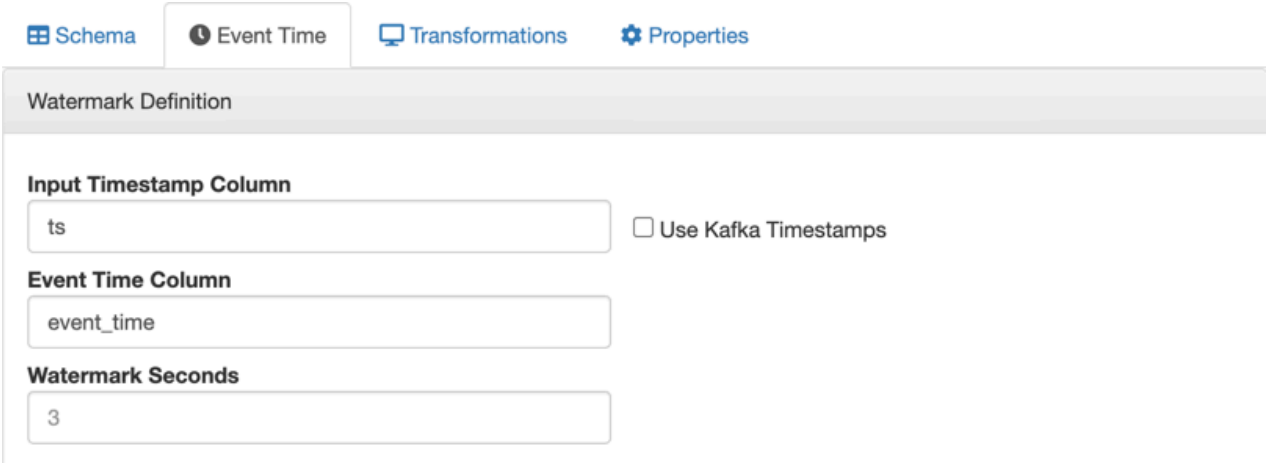
Manually providing the Kafka timestamp information

When you want to manually configure the watermark strategy of the Kafka table, you can provide the timestamp column name from the Kafka source, and add a custom column name for the resulting Kafka table. Make sure that you provide the correct column name for the Input Timestamp Column that exactly matches the column name in the Kafka source data.

To manually provide information for the watermark strategy, unselect the Use Kafka Timestamps checkbox, and provide the following information to the column name fields:

- Input Timestamp Column: name of the timestamp field in the Kafka source
- Event Time Column: predefined *'eventTimestamp'* name or custome column name of the timestamp field in the created Kafka table

As an example, you have a timestamp column in the source Kafka topic named as *'ts'*, and want to add a new timestamp column in your Kafka table as *'event_time'*. You provide the original timestamp column name in the Input Timestamp Column as *'ts'*, and add the custom *'event_time'* name to the Event Time Column.



This results in that the watermarks from the *'ts'* column is going to be mapped to the *'event_time'* column of the created Kafka table. As *'event_time'* will become the timestamp column name in the Kafka table, you must use the custom name (in this example the *'event_time'*) when querying the Kafka stream. This configuration of the timestamp columns are optional.

The Event Time Column can only be modified if the following requirements are met for the timestamp field of the Input Timestamp Column:

- The column type must be "long".
- The format must be in epoch (in milliseconds since January 1, 1970).

Not using watermark strategy for Kafka table

In case you do not need any watermark strategies, unselect the Use Kafka Timestamps checkbox, and leave the column and seconds field empty.

Note: Flink validates the input and output schemas of the data. You can only insert data into a Kafka topic, if the input and output data matches based on the schema defined for the topic. When customizing the timestamp column, make sure that the output data has the same schema.

Note: When configuring the timestamp for the Kafka tables, you must consider the timezone setting of your environment as it can effect the results of your query. For more information, see the Known Issues in the Release Notes.

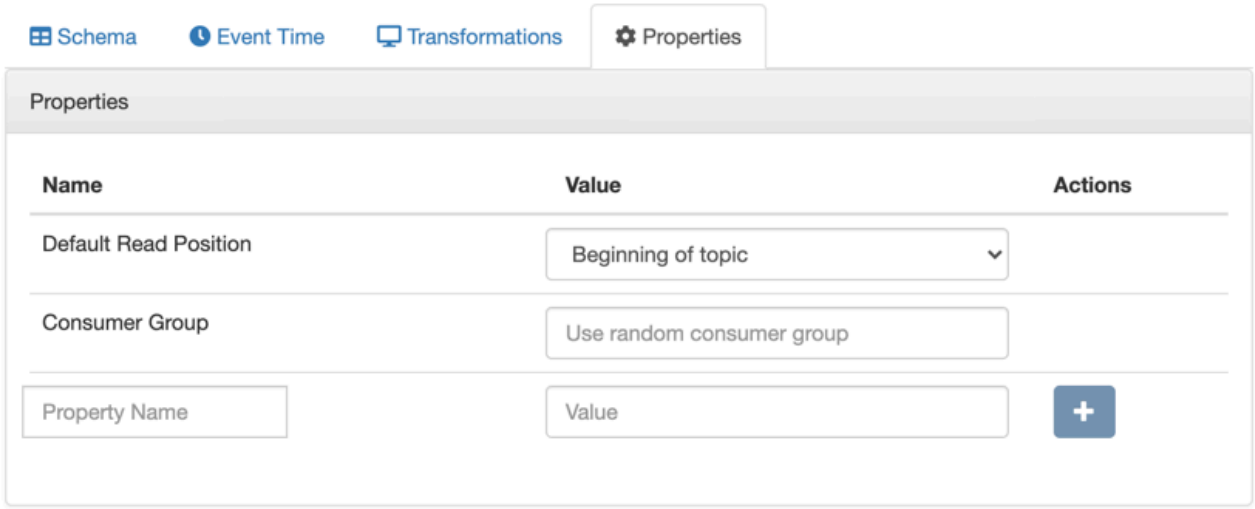
Input Transform tab

You can apply input transformations on your data when adding a Kafka table as a source to your queries. Input transformations can be used to clean or arrange the incoming data from the source using javascript functions.

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

Properties tab

You can specify certain properties to define your Kafka source in detail. You can also 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.



Assigning Kafka keys in streaming queries

Based on the Sticky Partitioning strategy of Kafka, when null keyed events are sent to a topic, they are randomly distributed in smaller batches within the partitions. As the results of the SQL Stream queries by default do not include a key, when written to a Kafka table, the Sticky Partitioning strategy is used. In many cases, it is useful to have more fine-grained control over how events are distributed within the partitions. You can achieve this in SSB by configuring a custom key based on your specific workload.

If using the Dynamic Kafka sink (created by selecting “Apache Kafka” in the “Add table” drop-down, as shown below), keys can be specified in the output of the query by including the special “_eventKey” column name or alias.

For example:

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

To configure keys in DDL-defined tables (those that are configured via “Flink DDL” in the “Add table” drop-down), refer to the official Flink Kafka SQL Connector documentation for more information (specifically the key.format and key.fields options).

Related Information

Creating Input Transforms

Creating Flink DDL tables

You can use Flink DDL to create tables or views by adding the CREATE statement to the Flink DDL window or filling out the Available templates.

The Flink DDL templates are predefined examples of CREATE TABLE statements which you can fill out with your job specific values.

You can choose from the following templates:

- create-table-like
- datagen
- jdbc
- kafka-json
- upsert-kafka

For full reference on the Flink SQL DDL functionality, see the official Apache Flink documentation.

Adding custom DDL templates and connectors

You can add custom templates for the existing catalogs, and also add new custom templates and connector to SQL Stream Builder to further customize the SQL Stream job for your production environment.

About this task

You can customize SSB based on your requirements with the following available scenarios for Flink DDL:

- Add custom DDL templates for default connectors
- Add custom DDL templates with custom connectors
- Add custom connectors to use in SSB

You can specify the new template by uploading the template file to the User Defined DDL Templates directory. If you need to add a custom connector that can be used with the template, you need to upload the corresponding connector jar file to the Flink SQL Connector Jar directory. Both directories can be configured through the SQL Stream Builder service in Cloudera Manager.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click SQL Stream Builder from the list of services.
3. Select Configuration tab.
4. Search for User Defined DDL Templates Directory and Flink SQL Connector Jar Directory configurations.
   You need to upload the custom template and connector to these directories.
5. Check the predefined value for the configurations.
6. Connect to the host with ssh where the Streaming SQL Console is running.
   For example:

   ```
   ssh root@docstest-1.vpc.cloudera.com
   ```

7. Upload the custom template and the connector jar to the corresponding directories using the command line tool.
   For example:

   ```
   scp path/flink-faker-template root@docstest-1.vpc.cloudera.com:.
   ```

8. Review that the template and connector file is in the directory.
   For example:

   ```
   ll /usr/share/flink-ddl-templates/
   ```

   If you see the template and connector file in the directories, you must restart the Streaming SQL Console service.
9. Click on Instances tab.
10. Check the Streaming SQL Console role type checkbox.
11. Click on Actions > Restart.
    After the Streaming SQL Console service restarted, the new template and connector should be available for Flink DDL.
12. Click on SQLStreamBuilder Console.
13. Click on Tables from the main tabs.
14. Select Add table > Flink DDL.
    The Flink DDL window appears.

15. Click on Available templates drop-down menu.
    The custom template is listed after the predefined templates.

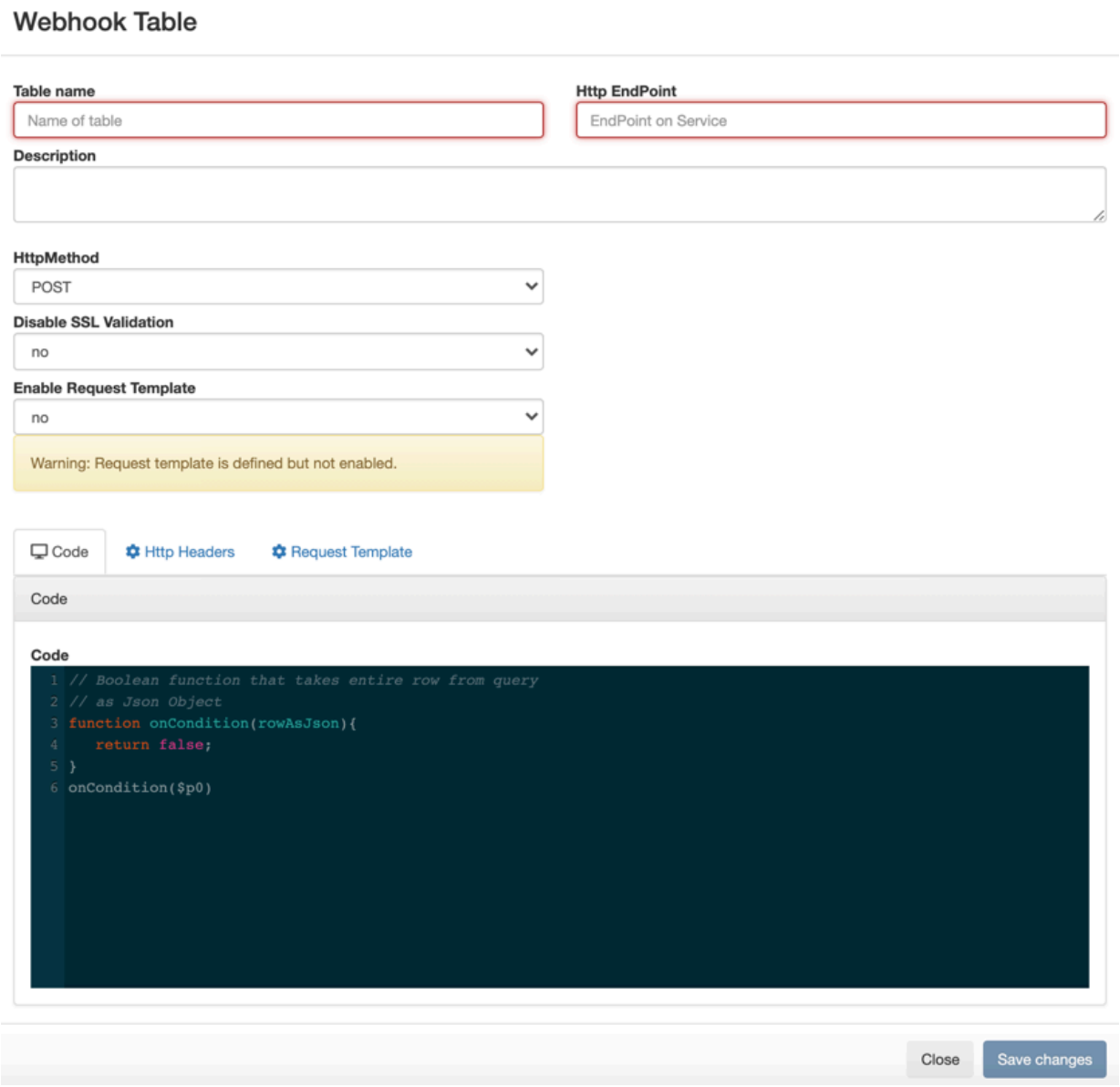
Creating Webhook tables

You can configure the webhook table to perform an HTTP action per message (default) or to create code 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 SQL Stream Builder from the list of Services.
3. Click SQLStreamBuilder Console.
   The Streaming SQL Console opens in a new window.
4. Select Console from the main menu.
5. Select the Tables tab.

6. Select Add table > Webhook .
   The Webhook Table window appears.



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

   Note: You can use hookbin for 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.
9. Add a Description about the webhook sink.
10. Select POST or PUT in the HTTP Method select box.
11. Choose to Disable SSL Validation, if needed.

12. 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 a valid JSON format, 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."
 }
    }
}
```

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

    For a webhook that is called for each message the following code is 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 the {"name":"value"} format.

14. Add HTTP headers using the HTTP Headers tab, if needed.

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

15. Click Save Changes.

Results

The Webhook table is ready to be used as sink in the SQL Stream jobs. You can add the created table on the Compose tab by selecting its name from the list of Sink Tables.

Managing time in SSB

Time attributes define how streams behave for time based operations such as window aggregations or joins. For Kafka tables you can use the Event Time tab to create source provided or user provided timestamp and watermarks. For other tables you can define time related attributes in the Flink DDL or directly in the SQL query. You can use timestamps that are already provided in the source or you can use custom timestamps.

Source-provided timestamps

Source-provided timestamps are inserted directly into the data stream by the source connector. This query uses the source-provided order_time field to perform a temporal join on multiple Kafka topics:

```
-- Table of orders
CREATE TABLE orders (
    order_id    STRING,
    price       DECIMAL(32,2),
    currency    STRING,
    order_time  TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time
```

```
) WITH (/* ... */);

-- Table of currency rates
CREATE TABLE currency_rates (
    currency STRING,
    conversion_rate DECIMAL(32, 2),
    update_time TIMESTAMP(3),
    WATERMARK FOR update_time AS update_time
) WITH (/* ... */);
-- Event time temporal join to enrich orders with currencies
SELECT
     order_id,
     price,
     currency,
     conversion_rate,
     order_time,
FROM orders
LEFT JOIN currency_rates FOR SYSTEM TIME AS OF orders.order_time
ON orders.currency = currency_rates.currency
```

User-provided timestamps

You can also specify timestamps contained in the data stream itself. For example, if your schema includes a field called "order_time", it is possible to construct a query such as:

```
-- Table of orders
-- Converts order_time_string field to timestamp
CREATE TABLE orders (
    order_id    STRING,
    price       DECIMAL(32,2),
    currency    STRING,
    order_time_string STRING,
    order_time as to_timestamp(order_time_string),
    WATERMARK FOR order_time AS order_time
) WITH (/* ... */);
```

When an invalid timestamp is found in the stream (for example, NaN), the timestamp of the message is going to be 0. This way the message is excluded from the current window.

When 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 the Input Transform feature of SSB.