

## Creating Tables

Date published: 2019-12-17

Date modified: 2022-09-28



# Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>Concept of tables in SSB.....</b>	<b>4</b>
<b>Creating Kafka tables.....</b>	<b>6</b>
Creating Kafka tables using Console wizard.....	6
Creating Kafka tables using Templates.....	8
Configuring Kafka tables.....	9
Schema tab.....	9
Event Time tab.....	10
Data Transformations tab.....	13
Properties tab.....	15
Deserialization tab.....	16
Assigning Kafka keys in streaming queries.....	19
Performance & Scalability.....	19
<b>Creating Flink tables using Templates.....</b>	<b>20</b>
<b>Creating Webhook tables.....</b>	<b>21</b>
<b>Managing time in SSB.....</b>	<b>23</b>

## Concept of tables in SSB

The core abstraction for Streaming SQL is a Table which represents both inputs and outputs of the queries. SQL Stream Builder (SSB) tables are an extension of the tables used in Flink SQL to allow a bit more flexibility to the users. When creating tables in SSB, you have the option to either add them manually, import them automatically or create them using Flink SQL depending on the connector you want to use.

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.

When you execute a query, the results go to the table you specify after the INSERT INTO statement 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 have created when you ran the query.

For example:

```
INSERT INTO air_traffic -- the name of the table sink
SELECT
lat,lon
FROM
airplanes -- the name of the table source
WHERE
icao <> 0;
```

### Table types in SSB

#### Kafka Tables

Apache Kafka Tables represent data contained in a single Kafka topic in JSON, AVRO or CSV format. It can be defined using the Streaming SQL Console wizard or you can create Kafka tables from the pre-defined templates.

#### Tables from Catalogs

SSB supports Kudu, Hive and Schema Registry as catalog providers. After registering them using the Streaming SQL Console, the tables are automatically imported to SSB, and can be used in the SQL window for computations.



**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.

#### Flink Tables

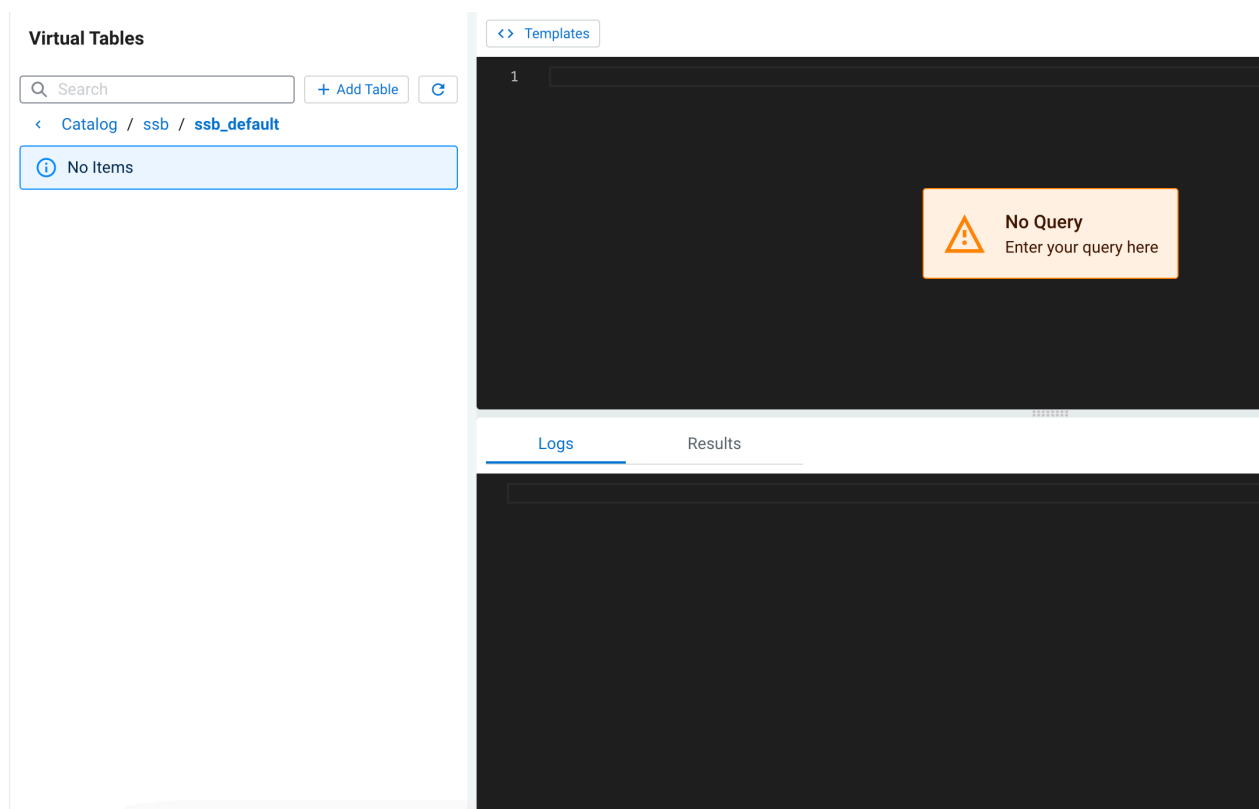
Flink SQL tables represent tables created by the standard CREATE TABLE syntax. This supports full flexibility in defining new or derived tables and views. You can either provide the syntax by directly adding it to the SQL window or use one of the predefined DDL templates.

#### Webhook Tables

Webhooks can only be used as tables to write results to. When you use the Webhook Tables the result of your SQL query is sent to a specified webhook.

## Table management on Console page


After creating your tables for the SQL jobs, you can review and manage them on the Console page next to the SQL Editor. The created tables are organized based on the teams a user is assigned to.



The + Add Table button can be used to create Apache Kafka and Webhook tables using an add table wizard. For any other supported tables of SQL Stream Builder (SSB), you can either manually add the CREATE TABLE statement to the **SQL Editor** or choose one of the predefined templates using the Template selector. After selecting the template of a connector, the CREATE TABLE statement is loaded to the **SQL Editor**.

The created tables are listed at the left panel, regardless if they are added using the wizard or created with Flink DDL. The tables appear based on which team is selected active for the user. The search bar can be used to find a specific table or shorten the list of tables by searching for a connector type.

The details of a table can be seen by clicking on the arrow next to the name of the table.


**datagen\_table\_1651486213**
datagen


TYPE: ssb  
TOPIC NAME:  
DATA FORMAT:




---

SCHEMA

Column	Type
col_str	STRING
col_int	INT
col_ts	TIMESTAMP(3) *ROWTIME*

When you hover over a table, expanded or not, the following buttons appear which can be used to manage your tables.


**datagen\_table\_1651486771**

- Show DDL - the DDL format of the table appears
  - You can use the copy button on the DDL window if you need to reuse the CREATE TABLE statement. You can also paste the DDL in the SQL Editor to edit the CREATE TABLE statement, and create a new table with changed attributes.
- Paste in Editor - the name of the table is added to the SQL editor
- Delete Table - removing table from SQL Stream Builder

## Creating Kafka tables

You have the option to create Kafka tables using the Console wizard, the built-in templates or directly adding a custom CREATE TABLE statement with the required properties in the SQL window.

### Creating Kafka tables using Console wizard

After registering a Kafka data provider, you can use the Add table wizard in Streaming SQL Console to create a Kafka table.

#### Before you begin

- Make sure that you have registered Kafka as a Data Provider.
- Make sure that you have created topics in Kafka.



**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. Navigate to the Streaming SQL Console.
  - a) Go to your cluster in Cloudera Manager.
  - b) Select SQL Stream Builder from the list of services.
  - c) Click SQLStreamBuilder Console .

The **Streaming SQL Console** opens in a new window.

2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Click Add table > Apache Kafka .

The Kafka Table window appears.

## Kafka Table



Table Name \*

Kafka Cluster \*

Data Format \*

Topic Name \*

Schema Definition

Event Time

Properties

Deserialization

```
1  {
2    "doc": "Default schema - modify as necessary",
3    "namespace": "com.eventador.exampleschema",
4    "type": "record",
5    "name": "exampleSchema",
6    "fields": [
7      {
8        "type": "string",
9        "name": "name"
10     },
11     {
12       "type": "int",
13       "name": "temp"
14     }
15   ]
16 }
```

✓ Schema is valid

Cancel

Create and Review

#### 4. Provide a Table Name.



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

#### 5. Select a registered Kafka provider as Kafka cluster.

#### 6. 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.

#### 7. 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.

#### 8. 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.

#### 9. Customize your Kafka Table with the following options:

- a) Configure the Event Time if you do not want to use the default Kafka Timestamps.

1. Disable the Use Kafka Timestamps feature.
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 on the Data Transformations.



**Note:** The Input Transformation is only supported for JSON data formats.

- c) Configure any Kafka properties required on the Properties tab.
- d) Select a policy for deserialization errors on the Deserialization tab.

For more information about how to configure the Kafka table, see the *Configuring Kafka tables* section.

#### 10. Click Create and Review.

### Results

The Kafka Table is ready to be used for the SQL job either at the FROM or at the INSERT INTO statements.

## Creating Kafka tables using Templates

The built-in templates allow you to simply and easily create tables by filling out the imported CREATE TABLE statement in the SQL window with detailed description of the properties.

You can create tables directly from the SQL window on the Console page by using the pre-defined connector templates.

When using the predefined templates, you have the following options for the Kafka table:

### CDP Kafka



Automatically using the Kafka service that is registered in the Data Providers, and runs on the same cluster as the SQL Stream Builder service. You can choose between JSON, Avro and CSV data types.

### Kafka

When connecting to a Kafka service that is not hosted in your cluster. You can choose between JSON, Avro, CSV and raw data types.

### Upsert Kafka

Connecting to a Kafka service in the upsert mode. This means that when using it as a source, the connector produces a changelog stream, where each data record represents an update or delete event. The value in the data records is interpreted as an update of the last value for the same key. When using the table as a sink, the connector can consume a changelog stream, and write insert/update\_after data as normal Kafka message valuea. Null values are represented as delete.

You can access and import the Templates from Streaming SQL Console:

1. Navigate to the **Streaming SQL Console**.

- a. Go to your cluster in Cloudera Manager.
- b. Click on SQL Stream Builder from the list of Services.
- c. Click on the SQLStreamBuilder Console.

The **Streaming SQL Console** opens in a new window.

2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Click Templates at the SQL Editor.

4. Select the template you want to use.

The template is imported to the SQL window.

5. Customize the fields of the template.

6. Click Execute.

The table is created based on the selected template, and appears next to the SQL Editor.

## Configuring Kafka tables

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

### Related Information

[Dynamic SQL Hints](#)

### Schema tab

When using the Add Kafka table wizard on the Streaming SQL Console, you can configure the schema under the 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

When using the Add Kafka table wizard on the Streaming SQL Console, you can configure the event time under the 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 feature is enabled 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.

<
Schema Definition
Event Time
Data Transformation
Properties
Des
>

Event Time Column

Watermark Seconds

Use Kafka Timestamps ☒

After saving your changes, you can view the created DDL syntax for the table next to the SQL Editor on the **Console** page.

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.

< Schema Definition Event Time Data Transformation Properties Des >

Event Time Column

Watermark Seconds

Use Kafka Timestamps ☒

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' - 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, disable the Use Kafka Timestamps feature using the toggle, 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 custom 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.

< Schema Definition Event Time Data Transformation Properties Des >

Input Timestamp Column

ts

Event Time Column

event\_time

Watermark Seconds

3

Use Kafka Timestamps ☐

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, disable the Use Kafka Timestamps feature using the toggle, and leave the column and seconds field empty.

< Schema Definition Event Time Data Transformation Properties Des >

Input Timestamp Column

Event Time Column

Watermark Seconds

Use Kafka Timestamps ☐



**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.

## Data Transformations tab

When using the Add Kafka table wizard on the Streaming SQL Console, you can apply input transformation under the Transformations tab. Input transformations can be used to clean or arrange the incoming data from the source using javascript functions.

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.

Input Transforms are a powerful way to clean, modify, and arrange data that is poorly organized, has changing format, and has data that is not needed or otherwise hard to use. With the Input Transform feature of SQL Stream Builder, you can create a javascript function to transform the data after it has been consumed from a Kafka topic, and before you run SQL queries on the data.

You can use Input Transforms in the following situations:

- The source is not in your control, for example, data feed from a third-party provider
- The format is hard to change, for example, a legacy feed, other teams of feeds within your organization
- The messages are inconsistent
- The data from the sources do not have uniform keys, or without keys (like nested arrays), but are still in a valid JSON format
- The schema you want does not match the incoming topic



**Note:** When using Input Transforms the schema you define for the Kafka table is applied on the output of the transformed data.

•



You can use the Input Transforms on Kafka tables that have the following characteristics:

- Allows one transformation per source.
- Takes record as a JSON-formatted string input variable. The input is always named record.
- Emits the output of the last line to the calling JVM. It could be any variable name. In the following example, out and emit is used as a JSON-formatted string.

A basic input transformation looks like this:

```

var out = JSON.parse(record.value);    // record is input, parse JSON f
ormatted string to object

// add more transformatio
ns if needed
JSON.stringify(out);                  // emit JSON formatted
string of object
  
```

## 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.

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 definition

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"
    }
  ]
}

```

### Creating Data Transformations

Input Transforms are a powerful way to clean, modify, and arrange data that is poorly organized, has changing format, has data that is not needed or otherwise hard to use. With the Input Transform feature of SQL Stream Builder, you can create a javascript function to transform the data after it has been consumed from a Kafka topic, and before you run SQL queries on the data.

### Procedure

1. Navigate to the Streaming SQL Console.

- a) Go to your cluster in Cloudera Manager.
- b) Select SQL Stream Builder from the list of services.
- c) Click SQLStreamBuilder Console .

The **Streaming SQL Console** opens in a new window.

2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Open the Kafka table configurations.

You can add the Input Transform to the Kafka table when you create the Kafka table:

- a) Choose Apache Kafka from the Add table drop-down.

You can add the Input Transform to an already existing Kafka table:

- a) Select the edit button for the Kafka table you want to add a transformation.

The Kafka table wizard appears.

4. Click Data Transformation.

You have the following options to insert your Input Transform:

- a) Add your javascript transformation code to the Data Transformation box.

Make sure the output of your transform matches the Schema definition detected or defined for the Kafka table.

- b) Click Install default template and schema.

The Install Default template and schema option fills out the Data Transformation box with a template that you can use to create the Input Transform, and matches the schema with the format.

5. Click Review and Create.

### Properties tab

When using the Add Kafka table wizard on the Streaming SQL Console, you can configure the properties under the 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.

## Kafka Table ✕

Table Name \*

Kafka Cluster \*

Data Format \*

Topic Name \*

Schema Definition

Event Time

Properties

Deserialization

Default Read Position

Consumer Group

### Custom Properties

Property Key

Property Value

+

✓ Schema is valid

Cancel

Create and Review

### Deserialization tab

When creating a Kafka table, you can configure how to handle errors due to schema mismatch using DDL or the Kafka wizard.

You can configure every supported type of Kafka connectors (local-kafka, kafka or upsert) how to handle if a message fails to deserialize which can result in job submission error. You can choose from the following configurations:

#### Fail

In this case an exception is thrown, and the job submission fails



**Ignore**

In this case the error message is ignored without any log, and the job submission is successful

**Ignore and Log**

In this case the error message is ignored, and the job submission is successful

**Save to DLQ**

In this case the error message is ignored, but you can store it in a dead-letter queue (DLQ) Kafka topic

**Using the Kafka wizard**

When you create the Kafka table using the wizard on the Streaming SQL Console, you can configure the error handling with the following steps:

1. Navigate to the Streaming SQL Console.
  - a. Go to your cluster in Cloudera Manager.
  - b. Select SQL Stream Builder from the list of services.
  - c. Click `SQLStreamBuilder Console`.
2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Select `Add tables > Apache Kafka`.

The **Add Kafka table** window appears

**4.** Select Deserialization tab.

### Kafka Table ✕

Table Name \*

Kafka Cluster \*

Data Format \*

Topic Name \*

Schema Definition

Event Time

Properties

Deserialization

Deserialization Failure Handler Policy

✔ Schema is valid

Cancel

Create and Review

**5.** Choose from the following policy options under **Deserialization Policy**:

- Fail
- Ignore
- Ignore and Log
- Save to DLQ

If you choose the Save to DLQ option, you need to create a dedicated Kafka topic where you store the error message. After selecting this option, you need to further select the created DLQ topic.

- Click Create and Review.

**Using DDL**

When you create the Kafka table using DDL on the Streaming SQL Console, you can configure the error handling with the following optional arguments:

1. Navigate to the Streaming SQL Console.
  - a. Go to your cluster in Cloudera Manager.
  - b. Select SQL Stream Builder from the list of services.
  - c. Click Web UI > SQLStreamBuilder Console .
2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Choose one of the Kafka template types from Templates.
4. Select any type of data format.

The predefined CREATE TABLE statement is imported to the SQL Editor.

5. Fill out the Kafka template based on your requirements.
6. Search for the deserialization.failure.policy.
7. Provide the value for the error handling from the following options:
  - a. 'error'
  - b. 'ignore'
  - c. 'ignore\_and\_log'
  - d. 'dlq'

If you choose the dlq option, you need to create a dedicated Kafka topic where you store the error message.

After selecting this option, you need to further provide the name of the created DLQ topic.

8. Click Execute.

## 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.

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 using the Templates), refer to the official [Flink Kafka SQL Connector](#) documentation for more information (specifically the key.format and key.fields options).

## Performance & Scalability

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.

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.

## Creating Flink tables using Templates

You can use the predefined templates to create tables by choosing one of the connector templates on the Console page of Streaming SQL Console. The Flink SQL templates are predefined examples of CREATE TABLE statements which you can fill out with your job specific values.

You can create tables using the predefined templates in SQL Stream Builder. The predefined templates consist of the CREATE TABLE statement, and every connection property that is needed for the given connector. You only need to fill out the templates and execute them in the Streaming SQL Console to create the table. When filling out the templates, you can add configuration based on what is specified in the given connector template. You can also customize the table name, column names and timestamp information for a template.

You can choose from the following templates based on the connector type:

### Blackhole

The BlackHole connector can be used to write all input records into. It is designed for high performance testing and UDF to output, not a substantive sink.

### Datagen

The Data generator connector can be used to sample randomly generated data in SSB. You can use this connector to try out and test SQL queries as records are generated until the job is running.

### Faker

The Faker connector can be used to generate fake data based on the Java faker expression. You can use this connector to try out and test SQL queries as records are generated until the job is running.

### Filesystem

The filesystem connector can be used to access partitioned files in file systems, and enables reading and writing from a local or distributed file system such as local, HDFS, S3 and so on. You only need to specify the data format that is used in the file system. You can choose from the following formats:

- Avro
- JSON
- CSV
- ORC
- Parquet

### JDBC

The JDBC connector enables reading data from and writing data into any relational databases with a JDBC driver. You can use PostgreSQL, MySQL and Hive as databases for the connector.

### Debezium CDC

You can use the Debezium CDC connector to stream changes in real-time from MySQL, PostgreSQL, Oracle, Db2 and SQL Server into Kafka. Debezium provides a unified format schema for changelog and supports serializing messages using JSON and Avro.

You can access and import the Templates from Streaming SQL Console:

#### 1. Navigate to the **Streaming SQL Console**.

- a. Go to your cluster in Cloudera Manager.
- b. Click on SQL Stream Builder from the list of Services.
- c. Click on the SQLStreamBuilder Console.

The **Streaming SQL Console** opens in a new window.

#### 2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Click Templates at the SQL Editor.
4. Select the template you want to use.

The template is imported to the SQL window.

5. Customize the fields of the template.
6. Click Execute.

The table is created based on the selected template, and appears next to the SQL Editor.

For full reference on the Flink SQL DDL functionality, see the official [Apache Flink](#) documentation.

## 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. Navigate to the Streaming SQL Console.
  - a) Go to your cluster in Cloudera Manager.
  - b) Select SQL Stream Builder from the list of services.
  - c) Click SQLStreamBuilder Console .The **Streaming SQL Console** opens in a new window.
2. Click Create Job or select a previous job on the **Getting Started** page.

You are redirected to the **Console** page.

3. Select Add table > Webhook .

The Webhook Table window appears.

## Webhook Table ✕

Table Name \*

HTTP Endpoint \*

Description \*

HTTP Method

☐ SSL Validation

☐ Request Template

Code

Http Headers

Request Template

```
1 // Boolean function that takes entire row from query
2
3 // as Json Object
4
5 function onCondition(rowAsJson){
6
7     return false;
8
9 }
10
11 onCondition($p0)
```

Cancel

Create

4. Provide a name to the Table.

5. 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.

6. Add a Description about the webhook sink.
7. Select POST or PUT in the HTTP Method select box.
8. Choose to Disable SSL Validation, if needed.

**9. 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."
    }
  }
}
```

**10. 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.

**11. Add HTTP headers using the HTTP Headers tab, if needed.**

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

**12. Click Create.****Results**

The Webhook table is ready to be used after the INSERT INTO statement in your SQL query.

## 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.