

Cloudera DataFlow for Data Hub 7.2.11

## Querying data with SQL Stream Builder

Date published: 2019-12-16

Date modified: 2021-09-09

The Cloudera logo is displayed in a bold, orange, sans-serif font. The word "CLOUDERA" is written in all caps, with a stylized 'E' that has a horizontal bar extending to the right.

<https://docs.cloudera.com/>

# Legal Notice

© Cloudera Inc. 2024. 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>Prepare your environment.....</b>	<b>4</b>
Assigning resource roles.....	4
Creating IDBroker mapping.....	5
Setting workload password.....	5
Creating Streaming Analytics cluster.....	5
Configuring Ranger policies for SSB.....	6
Retrieving keytab file.....	6
Uploading and unlocking your keytab in SSB.....	7
<b>Creating Kafka tables in SSB.....</b>	<b>8</b>
Adding Kafka as Data Provider.....	8
Creating Kafka tables.....	10
Configuring Kafka tables.....	12
Creating Input Transforms.....	16
<b>Creating tables with Flink DDL in SSB.....</b>	<b>18</b>
Adding catalogs as Data Provider.....	18
Adding Schema Registry as Catalog.....	18
Adding Kudu as Catalog.....	19
Adding Hive as Catalog.....	20
Adding Custom Catalogs.....	21
Adding custom DDL templates and connectors.....	21
Creating tables with Flink DDL.....	22
<b>Creating Webhook tables.....</b>	<b>24</b>
<b>Running SQL Stream jobs.....</b>	<b>26</b>
<b>Enriching streaming data with join.....</b>	<b>28</b>
Joining streaming and bounded tables.....	29
Example: joining Kafka and Kudu tables.....	30

## Prepare your environment

Before you start querying your data using SQL Stream Builder (SSB) in CDP Public Cloud, you need to register and prepare your environment so that a chosen user or group can use the clusters and services in the environment securely.

As a first step, an admin needs to register an AWS, Azure or GCP environment. In CDP Public Cloud, this environment is based on the virtual private network in your cloud provider account. This means that CDP can access and identify the resources in your cloud provider account which is used for CDP services, and shared between clusters in the environment. After registering your AWS, Azure or GCP environment, you can provision clusters, and set users or groups to access your environment and services.

For more information about registering an AWS, Azure or GCP environment, see the [Management Console documentation](#).

In CDP Public Cloud, the Streaming Analytics clusters use Apache Knox to authenticate users. Knox is automatically set up for your environment. To access the Streaming SQL Console, you need to upload your keytab to your cluster, and unlock the keytab in Streaming SQL Console. You also need to set Ranger policies for SSB in Kafka and YARN to grant access for job submission. You can further authorize the users using Team Management feature of SSB in the Streaming SQL Console

Make sure that you have completed the following prerequisites:

- You have a CDP Public Cloud environment.
- You have a CDP username (it can be your own CDP user or a CDP machine user) and a workload password set to access Data Hub clusters.

The predefined resource role of this user is at least EnvironmentUser. This resource role provides the ability to view Data Hub clusters and set the FreeIPA password for the environment.

- Your user is synchronized to the CDP Public Cloud environment.
- You have a Streaming Analytics cluster.
- You have uploaded your keytab file to the Streaming Analytics cluster.
- You have unlock your keytab in the Streaming SQL Console.
- Your CDP user has the correct permissions set up in Ranger allowing access to SQL Stream Builder.



**Important:** In case you cannot access the Streaming SQL Console after setting up your environment, see the [Release Notes - Known issues](#) section for further information.

## Assigning resource roles

As an administrator, you need to give permissions to users or groups to be able to access and perform tasks in your Data Hub environment.

### Procedure

1. Navigate to [Management Console > Environments](#) and select your environment.
2. Click [Actions > Manage Access](#).
3. Search for a user or group that needs access to the environment.
4. Select EnvironmentUser role from the list of Resource Roles.
5. Click [Update Roles](#).  
The Resource Role for the selected user or group will be updated.
6. Navigate to [Management Console > Environments](#), and select the environment where you want to create a cluster.
7. Click [Actions > Synchronize Users to FreeIPA](#).

8. Click Synchronize Users.



**Note:** There might be cases where the status of the environment is synchronized with warnings and has failed status. This does not indicate that the synchronization has failed.

## Creating IDBroker mapping

As an administrator, you must create IDBroker mapping for a user or group to access cloud storage. As a part of Knox, the IDBroker allows a user to exchange cluster authentication for temporary cloud credentials.

### Procedure

1. Navigate to Management Console > Environments and select your environment.
2. Click Actions > Manage Access .
3. Click on the IDBroker Mappings tab.
4. Click Edit to add a new user or group and assign roles to have writing access for the cloud storage.
5. Search for the user or group you need to map.
6. Go to the IAM Summary page where you can find information about your cloud storage account.
7. Copy the Role ARN.
8. Go back to the IDBroker Mapping interface on the Cloudera Management Console page.
9. Paste the Role ARN to your selected user or group.
10. Click Save and Sync.

## Setting workload password

As a user, you need to set a workload password for your EnvironmentUser account to be able to access the SQL Stream Builder nodes through SSH connection.

### Procedure

1. Navigate to Management Console > Environments and select your environment.
2. Click Actions > Manage Access .
3. Click Workload Password.
4. Give a chosen workload password for your user.
5. Confirm the given password by typing it again.
6. Click Set Workload Password.

## Creating Streaming Analytics cluster

As a user, you need to create the Streaming Analytics cluster in your environment to use SQL Stream Builder in Data Hub. You also need to add Streams Messaging, Real-time Data Mart, Data Engineering or Operational Database cluster based on your connector choice.

### Procedure

1. Navigate to Management Console > Environments , and select the environment where you want to create a cluster.
2. Click Create Data Hub.
3. Select Streaming Analytics Light Duty cluster from Cluster Definition.
4. Provide a chosen cluster name.

5. Click Provision Cluster.

## Configuring Ranger policies for SSB

You must add SQL Stream Builder (SSB) users to the Ranger policies that are needed for YARN, Kafka, Schema Registry, Hive and Kudu to provide access to topics, schemas and tables used by the components.

### Before you begin

- Add the required Ranger policies for Flink. For more information, see [Configuring Ranger policies for Flink](#) documentation.

### Procedure

1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Click on the Data Lake tab.
3. Select Ranger from the services.  
You are redirected to the Ranger user interface.
4. Select the Streaming Analytics cluster under the YARN service.
  - a) Add the SSB user to the all-queue policy.
5. Select the Streaming Analytics cluster under the Kafka service.
  - a) Add the SSB user to the following policies:
    - all-consumergroup
    - all-topic
6. Select the cluster under the needed service where you want to add the SSB user.

Based on the connector type you want to use with SSB, you need to add the SSB user to the Schema Registry, Kudu or Hive service. The following minimal permissions are required for the SSB user:

Service	Policy name
Schema Registry	<ul style="list-style-type: none"> <li>• all-schema-group, schema-metadata, schema-branch, schema-version</li> </ul>
Kudu	<ul style="list-style-type: none"> <li>• Database</li> <li>• Table</li> <li>• Column</li> </ul>
Hive	<ul style="list-style-type: none"> <li>• all-global</li> <li>• all-database, table, column</li> <li>• all-database, table</li> <li>• all-database</li> <li>• all-hiveservice</li> <li>• all-database, udf</li> <li>• all-url</li> </ul>

## Retrieving keytab file

As a user, you need to retrieve the keytab file of your profile and upload it to the Streaming SQL Console to be able to run SQL jobs.

## Procedure

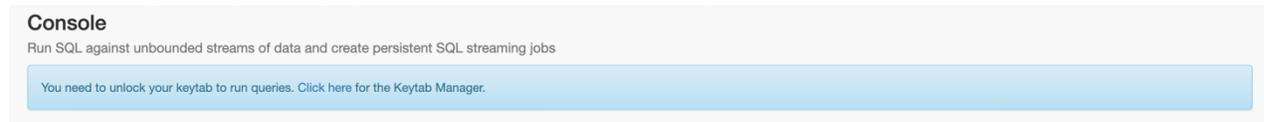
1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Click on your profile name.
3. Click Profile.
4. Click Actions > Get Keytab.
5. Choose the environment where your Data Hub cluster is running.
6. Click Download.
7. Save the keytab file in a chosen location.

## Uploading and unlocking your keytab in SSB

When accessing the Streaming SQL Console for the first time in Data Hub, you must upload and unlock the keytab file corresponding with your profile before you can use SQL Stream Builder (SSB).

### About this task

The following message is displayed on the Console in case the keytabs are still locked:



### Before you begin

- Retrieve the keytab file for your user profile. For more information, see the [Retrieving keytab file](#) section.



**Important:** In case you cannot access the Streaming SQL Console after setting up your environment, see the [Release Notes - Known issues](#) section for further information.

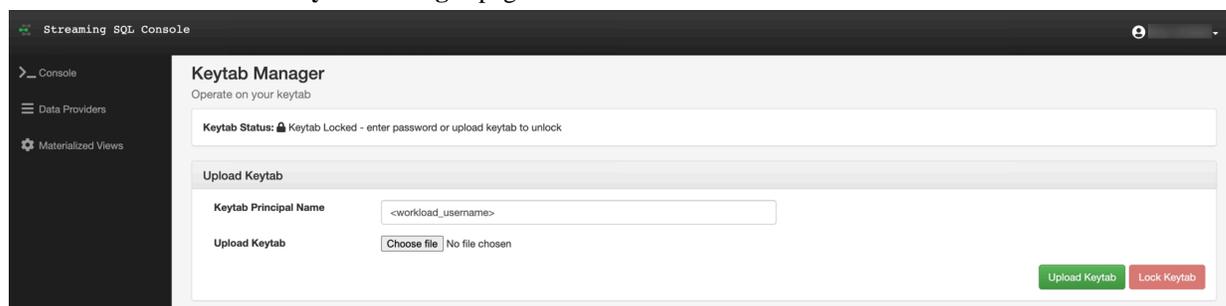
## Procedure

1. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.

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

2. Click your username at the right top corner of the Streaming SQL Console.
3. Click Manage keytab.

You are redirected to the **Keytab Manager** page.



4. Click Choose file to upload your keytab file.

5. Search and select your keytab file.
6. Click Upload Keytab.
7. Click Unlock Keytab.

In case there is an error when unlocking your keytab, you can get more information about the issue with the following steps:

- a. Manually upload your keytab to the Streaming Analytics cluster:

```
scp <location>/<your_keytab_file> <workload_username>@<manager_node_FQDN>:.  
Password:<your_workload_password>
```

- b. Access the manager node of your Streaming Analytics cluster:

```
ssh <workload_username>@<manager_node_FQDN>  
Password: <workload_password>
```

- c. Use kinit command to authenticate your user:

```
kinit -kt <keytab_filename>.keytab <workload_username>
```

- d. Use the flink-yarn-session command to see if the authentication works properly:

```
flink-yarn-session -d \  
-D security.kerberos.login.keytab=<keytab_filename>.keytab \  
-D security.kerberos.login.principal=<workload_username>
```

In case the command fails, you can review the log file for further information about the issue.

## Creating Kafka tables in SSB

You can query your streaming data using Kafka tables in SQL Stream Builder (SSB). When using Kafka tables, you need to create a Streams Messaging cluster within your environment, and navigate through the Streaming SQL Console to output the queried results into a Kafka topic.

### Before you begin

- You have prepared your environment based on the checklist in the *Prepare your environment* section.
- You have set Ranger policies for Flink and SSB.
- You have created a Streams Messaging cluster in your environment.
- You have created a Kafka topic from which you can read messages from in SSB.

## Adding Kafka as Data Provider

You need to register Kafka as a Data Provider using the Streaming SQL Console to create Kafka tables 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. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.

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

2. Click Data Providers from the main menu.

3. Click Register Kafka Provider.

The Add Kafka Provider window appears.

## Add Kafka Provider

### Name

### Brokers (comma-seperated list)

### Connection protocol

4. Add a Name to your Kafka provider.
5. 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
```

## 6. 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.

## 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 registered Kafka as a Data Provider.
- 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. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.The **Streaming SQL Console** opens in a new window.
2. Select Console from the left- side menu.
3. Go to the Tables tab.

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

## Kafka Source

Source is valid

### Virtual table name

Please provide a virtual table name.

### Kafka Cluster

### Topic Name

### Data Format

Schema
 Transformations
 Properties

Schema Definition

```

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 }

```

Ln: 1 Col: 1




- Provide a Name for the Table.



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

- Select a registered Kafka provider as Kafka cluster.
- 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.** 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.

**9.** 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.

**10.** 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.

**11.** 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.

**Input Timestamp Column**

Kafka Record Timestamp  Use Kafka Timestamps

**Event Time Column**

ets

**Watermark Seconds**

4

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, 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 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 Event Time Transformations Properties

**Watermark Definition**

**Input Timestamp Column**

ts  Use Kafka Timestamps

**Event Time Column**

event\_time

**Watermark Seconds**

3

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.

The screenshot shows a configuration panel for a Kafka table. It contains three input fields and one checkbox. The first field is labeled "Input Timestamp Column" and contains the text "tsColumn". To its right is a checkbox labeled "Use Kafka Timestamps" which is currently unchecked. The second field is labeled "Event Time Column" and contains the text "eventTimestamp". The third field is labeled "Watermark Seconds" and contains the number "3".



**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.

Name	Value	Actions
Default Read Position	Beginning of topic	
Consumer Group	Use random consumer group	
Property Name	Value	+

### 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 Input Transforms

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.

### About this task

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

## Procedure

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

- Navigate to Management Console > Environments, and select the environment where you have created your cluster.
- Select the Streaming Analytics cluster from the list of Data Hub clusters.
- Select Streaming SQL Console from the list of services.

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

### 2. Select Console from the left-hand menu.

### 3. Select Tables.

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

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

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

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

The Kafka table wizard appears.

### 4. Click Transformations.

You have the following options to insert your Input Transform:

- 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.

- 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 Save changes.

## Creating tables with Flink DDL in SSB

You can query your streaming data using Flink DDL in SQL Stream Builder (SSB). When using Flink DDL, you can use Kudu, Hive, Schema Registry and custom catalogs to create tables and views. There are also predefined templates that you only need to fill out with the connection information. When using Kudu, Hive and Schema Registry catalog, you need to create a Streams Messaging, Real-time Data Mart or Data Engineering cluster within your environment, and navigate through the Streaming SQL Console to output the queried results into a preferable sink.

### Before you begin

- You have prepared your environment based on the checklist in the *Prepare your environment* section.
- You have set Ranger policies for Flink and SSB.
- You have created a Streams Messaging, Real-time Data Mart or Data Engineering cluster in your environment.
- You have tables in Kudu, Hive and added a schema to Schema Registry.

## Adding catalogs as Data Provider

When using Flink DDL in SQL Stream Builder (SSB), you need to add Schema Registry, Kudu or Hive as a Catalog in the Streaming SQL Console. You can also add File systems and JDBC as Custom Catalogs from the supported connectors.

## Adding Schema Registry as Catalog

You need to add Schema Registry as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to use Schema Registry with Flink DDL.

### Before you begin

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

### Procedure

1. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.The **Streaming SQL Console** opens in a new window.
2. Click Data Providers from the main menu.
3. Click Register Catalog.

The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Schema Registry from the Catalog Type drop-down.
6. Select the Kafka cluster you registered as Data Provider.

7. 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
```

8. Select if you want to enable TLS for Schema Registry.
  - a) If yes, provide the Schema Registry Truststore location and password.
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

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

You need to add Kudu as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to create Kudu tables with Flink DDL.

### Before you begin

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

### Procedure

1. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.The **Streaming SQL Console** opens in a new window.
2. Click Data Providers from the main menu.
3. Click Register Catalog.

The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Kudu from the Catalog Type drop-down.

6. 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
```

7. Click on Add Filter.
  - a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
8. Click on Validate.
9. 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

You need to add Hive as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to create Hive tables with Flink DDL.

### Before you begin

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

### Procedure

1. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.The **Streaming SQL Console** opens in a new window.
2. Click Data Providers from the main menu.
3. Click Register Catalog.

The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Hive from the Catalog Type drop-down.
6. Click on Add Filter.
  - a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
7. Click on Validate.
8. 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. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.

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

2. Click Data Providers from the main menu.
3. Click Register Catalog.

The Add Catalog window appears.

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

### Results

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

## 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. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Cloudera Manager from the services.  
You are redirected to the **Cloudera Manager** user interface.
  - d) Click Configuration.
2. Select Local Directories and Files.
3. Select sql\_stream\_builder from Scope to filter down the configuration properties.  
The Flink SQL Connector Jar Directory and User Defined DDL Templates Directory information is displayed.  
You need to upload the custom template and connector to these directories.
4. Check the predefined value for the configurations.
5. Connect to the host with ssh where the Streaming SQL Console is running.

For example:

```
ssh <your_workload_username>@<master_node_FQDN>  
Password:<your_workload_password>
```

6. Upload the custom template and the connector jar to the corresponding directories using the command line tool.

For example:

```
scp path/flink-faker-template <your_workload_username>@<master_node_FQDN>:  
>:.
```

7. 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.

8. Select Clusters > sql\_stream\_builder .
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.

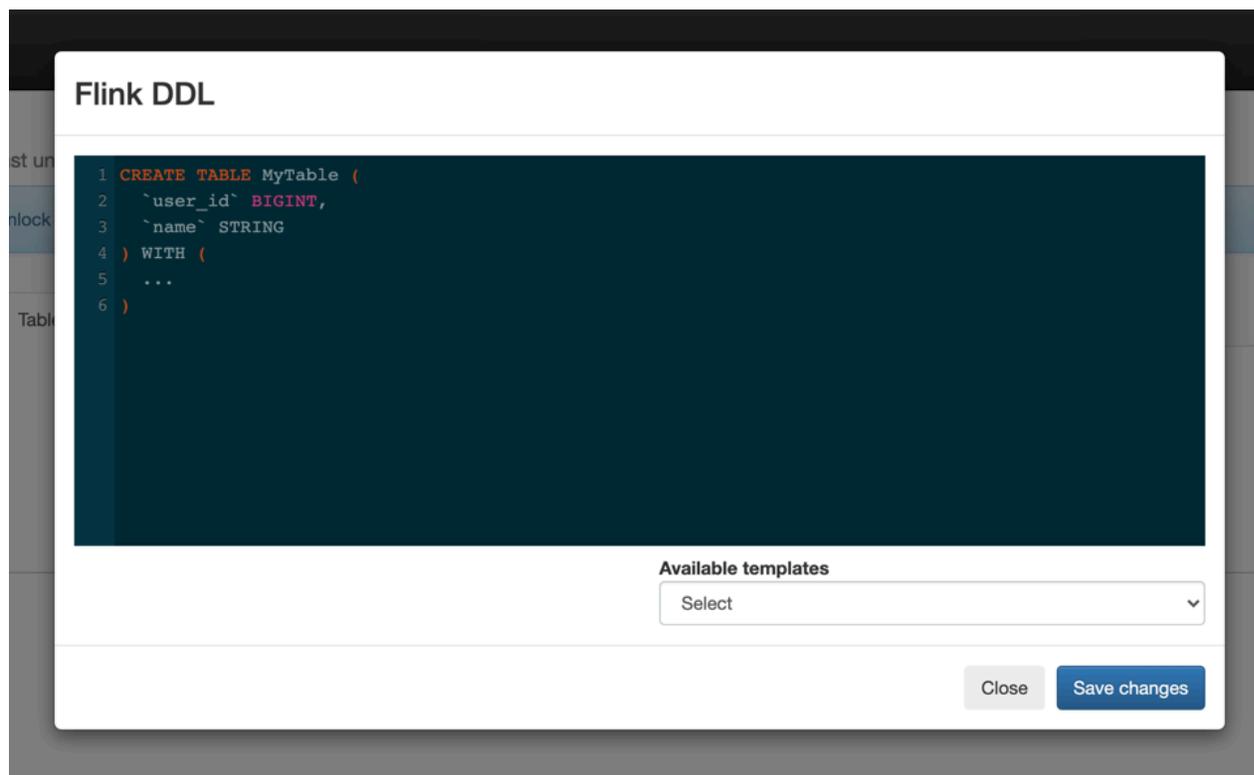
12. Click on SQLStreamBuilder Console.
13. Click on Tables from the main tabs.
14. Select Add table > Flink DDL.  
The Flink DDL window appears.
15. Click on Available templates drop-down menu.

The custom template is listed after the predefined templates.

## Creating tables with Flink DDL

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.

When using Flink DDL, you have the option to either define your CREATE TABLE statement using SQL in the Flink DDL window, or use the available Flink DDL 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:

#### For Create table like

```

CREATE TABLE like_table_sample (
  -- new_time_col AS to_timestamp(time_col),
  -- WATERMARK FOR new_time_col AS new_time_col - INTERVAL '5' SECOND
) WITH (
  -- 'scan.startup.mode' = 'latest-offset'
)
LIKE kafka_table (
  INCLUDING ALL
  OVERWRITING OPTIONS
  EXCLUDING GENERATED
)

```

#### For Datagen

```

CREATE TABLE datagen_sample (
  column_int INT,
  column_str STRING,
  column_ts TIMESTAMP(3),
  WATERMARK FOR column_ts AS column_ts - INTERVAL '3' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.column_int.min' = '100000',
  'fields.column_int.max' = '999999',
  'fields.column_str.length' = '40'
)

```

### For JDBC

```
CREATE TABLE jdbc_sample (  
  column_int INT,  
  column_str STRING,  
  column_ts   TIMESTAMP(3)  
) WITH (  
  'connector' = 'jdbc',  
  'url' = 'jdbc:postgresql://<db_host>:5432/<db_name>',  
  'table-name' = '...',  
  'username' = '...',  
  'password' = '...' )
```

### For Kafka JSON

```
CREATE TABLE kafka_sample (  
  column_int INT,  
  column_str STRING,  
  column_ts   BIGINT,  
  event_time AS CAST(from_unixtime(floor(column_ts/1000)) AS TIMESTAMP(3  
) ),  
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND  
) WITH (  
  'connector' = 'kafka',  
  'topic' = '...',  
  'properties.bootstrap.servers' = '...',  
  --'properties.group.id' = '...',  
  'scan.startup.mode' = 'earliest-offset',  
  'format' = 'json'  
)
```

### For Upsert Kafka

```
CREATE TABLE upsert_kafka (  
  column_int INT,  
  column_str STRING,  
  column_ts   TIMESTAMP(3),  
  PRIMARY KEY (column_int) NOT ENFORCED  
) WITH (  
  'connector' = 'upsert-kafka',  
  'topic' = '...',  
  'properties.bootstrap.servers' = '...',  
  'key.format' = 'avro',  
  'value.format' = 'avro'  
)
```

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) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.

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

2. Select Console from the main menu.

3. Select the Tables tab.

4. Select Add table > Webhook .

The Webhook Table window appears.

### Webhook Table

<b>Table name</b>	<b>Http EndPoint</b>
<input type="text" value="Name of table"/>	<input type="text" value="EndPoint on Service"/>
<b>Description</b> <input type="text"/>	
<b>HttpMethod</b> <input type="text" value="POST"/>	
<b>Disable SSL Validation</b> <input type="text" value="no"/>	
<b>Enable Request Template</b> <input type="text" value="no"/>	
Warning: Request template is defined but not enabled.	

Code Http Headers Request Template

**Code**

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

Close Save changes

5. Provide a name to the Table.

6. 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.

7. Add a Description about the webhook sink.
8. Select POST or PUT in the HTTP Method select box.
9. Choose to Disable SSL Validation, if needed.
10. 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."
    }
  }
}
```

11. 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.

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

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

13. 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.

## Running SQL Stream jobs

Every time you run an SQL statement in the SQL Stream console, it becomes a job and runs on the deployment as a Flink job. You can manage the running jobs using the Jobs tab on the UI.

### About this task

There are two logical phases to run a job:

1. Parse: The SQL is parsed and checked for validity and then compared against the virtual table schema(s) for correct typing and key/columns.
2. Execution: If the parse phase is successful, a job is dynamically created, and runs on an open slot on your cluster. The job is a valid Flink job.

### Before you begin

- Make sure that you have registered a Data Provider.
- Make sure that you have created a Table that can be used as a source in the SQL query.

### Procedure

1. Navigate to the Streaming SQL Console.
  - a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
  - b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
  - c) Select Streaming SQL Console from the list of services.

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

2. Provide a name for the SQL job.
  - a) Optionally, you can click Random Name to generate a name for the SQL job.
3. Select a Sink Table.
  - a) Optionally, you can leave the sink as None.



**Note:** The Sink Table is an optional argument to a job that specifies the destination for the continuous results. If you select None, the results are not sent to a sink, but sampled on the screen or to a materialized view.

4. Add a SQL query to the SQL window.

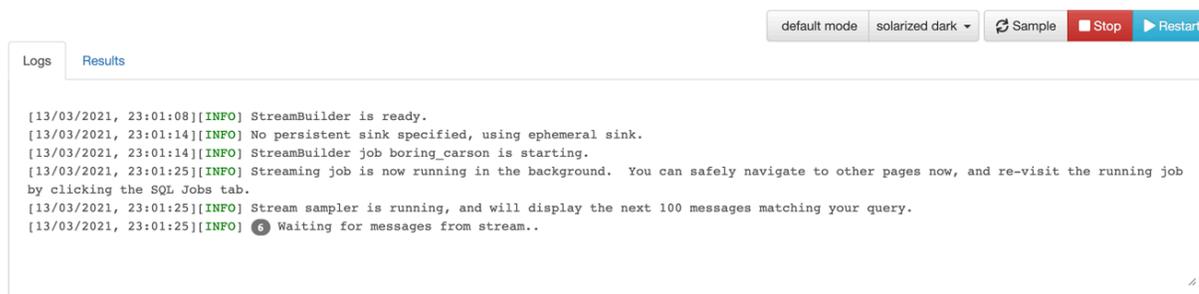


**Note:** You cannot start a job without adding a SQL statement in the SQL editor window. In case, there are no SQL statements provided and you click Execute , the following error message is displayed: You must provide a SQL query.

When starting a job, the number of slots consumed on the specified cluster is equal to the parallelism setting. The default is one slot. To change the parallelism setting, click Advanced settings.

## 5. Click Execute.

The Logs window updates the status of SSB.



```

[13/03/2021, 23:01:08][INFO] StreamBuilder is ready.
[13/03/2021, 23:01:14][INFO] No persistent sink specified, using ephemeral sink.
[13/03/2021, 23:01:14][INFO] StreamBuilder job boring_carson is starting.
[13/03/2021, 23:01:25][INFO] Streaming job is now running in the background. You can safely navigate to other pages now, and re-visit the running job by clicking the SQL Jobs tab.
[13/03/2021, 23:01:25][INFO] Stream sampler is running, and will display the next 100 messages matching your query.
[13/03/2021, 23:01:25][INFO] ⌚ Waiting for messages from stream..
  
```

## 6. Click Results to check the sampled data.

These results are only samples, not the entire result of the new stream being created from the output of the query. The entire result set is sent to the Sink Table and/or a Materialized View.



**Note:** While SSB is querying unbounded data streams, you can click Stop to stop the job execution.

### Results

A job is generated that runs the SQL continuously on the stream of data from the source Table, and pushes the results to a sink Table, to the Console under the Results tab or to a Materialized View.

### Related Information

[Registering Data Providers in SSB](#)

[Using Tables in SQL Stream jobs](#)

[SQL Job Lifecycle SQL Stream jobs](#)

[Monitoring SQL Stream jobs](#)

## Enriching streaming data with join

In SQL Stream builder, you can enrich your streaming data with values from a slowly changing dataset using join statements.

### Regular join

Join statements in SQL serves to combine columns and rows from two or more tables based on a shared column. When you join tables from a slowly changing source such as HDFS, Kudu, Hive and so on, you can simply use the regular JOIN syntax of SQL. The following example shows a regular INNER JOIN where the *Orders* table is joined with *Product* table based on the *productId*:

```

SELECT * FROM Orders
INNER JOIN Product
ON Orders.productId = Product.id
  
```

A regular join can only be used with bounded tables. In a streaming context data is produced continuously, and with a regular join both sides of the join would need to be buffered indefinitely to store all of the events that would match with the result of the SQL query. To get results from a given amount of time and to join streaming tables, a time boundary needs to be specified. This means the tables not only need to be joined by a key or column, but also on a time attribute.

## Interval join

When joining streaming tables, the time attribute can be defined in the SQL syntax using `BETWEEN` and an interval value:

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.order_id
AND o.order_time BETWEEN s.ship_time - INTERVAL '4' HOUR AND s.ship_time
```

In this case, *Orders* table is joined with the *Shipments* table and the results are going to be generated based on the *id* column as long as the order time and shipment time is within four hours of each other. The condition of this scenario is that the events in the streams happen almost at the same time with minimal delay, so the time boundary can be defined between an approximate interval.

When you want to join a streaming table with a slowly changing table, time attributes can differ as one of the tables stores data over a long period of time, while the streaming table receives new data continuously. To join these types of tables, a time needs to be defined that can serve as a reference point for both types of tables.

## Joining streaming and bounded tables

Beside regular join and interval join, in Flink SQL you are able to join a streaming table and a slowly changing dimension table for enrichment. In this case, you need to use a temporal join where the streaming table is joined with a versioned table based on a key, and the processing or event time.

A versioned table is a table that contains a time attribute, and reflects the records from a table at a specific point of time. When you use append-only or regularly updated sources, the values related to a key are updated over a long period of time. For example, a table can contain the currency rates since last month. At every change of the currency rate, a new value is added to the stream, therefore to the table. With creating a versioned table of the currency rate, you can specify which rate you need in an exact point of time: use the currency rates from 12:00.

For more information of Version Tables, see the official [Apache Flink documentation](#).

## Temporal join

After determining the version of the bounded table, you also need to define a time for the streaming table. In Flink, event time and processing time can be specified. When using event time, you need to create a temporal join.

Event time is the time that each individual event occurred on its producing device. To have an event time attribute in your SQL query, you need to define a timestamp column with a watermark definition column when creating the table:

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

When you want to use event time in a `JOIN`, you need to refer to the event time column as defined for the table in the `FOR SYSTEM_TIME AS OF` part of the SQL query:

```
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
```

### Lookup join

As a special case of temporal join, you can use the processing time as a time attribute. In Flink, processing time is the system time of the machine, also known as “wall-clock time”. When you use the processing time in a JOIN SQL syntax, Flink translates into a lookup join and uses the latest version of the bounded table. The following example shows the join syntax that needs to be used for enriching streaming data:

```
SELECT o.order_id, o.total, c.country, c.zip
FROM Orders AS o
JOIN Customers FOR SYSTEM_TIME AS OF PROCTIME()
ON o.customer_id = c.id
```

In the above example, *Customers* serves as the lookup table. The `FOR SYSTEM_TIME AS OF PROCTIME()` syntax indicates that you always want to look up in the latest version of the table. With including the processing time in the SQL syntax, you can query the latest version of a lookup table, and enrich your streaming data with the corresponding value.



**Note:** When using SQL Stream Builder, you can simply use the `PROCTIME()` function as the version of the lookup table when performing a lookup join. This means that you do not need to create and reference a processing time column in your probe stream anymore.

In SQL Stream Builder, the following connectors are supported as lookup tables:

- Kudu
- Hive
- JDBC

## Example: joining Kafka and Kudu tables

Using lookup join, you can join Kafka and Kudu tables to enrich the streaming data of Kafka with information from the Kudu tables. In this example, Orders of a Kafka streaming table are enriched with metadata information from a Kudu table.

As a prerequisite for the example, the following steps were completed in SQL Stream Builder:

- Registering Kafka as a Data Provider.
- Registering Kudu as a Catalog.
- Creating Orders Kafka table.
- Creating ItemMeta Kudu table.
- Generating data for Kafka and Kudu tables.

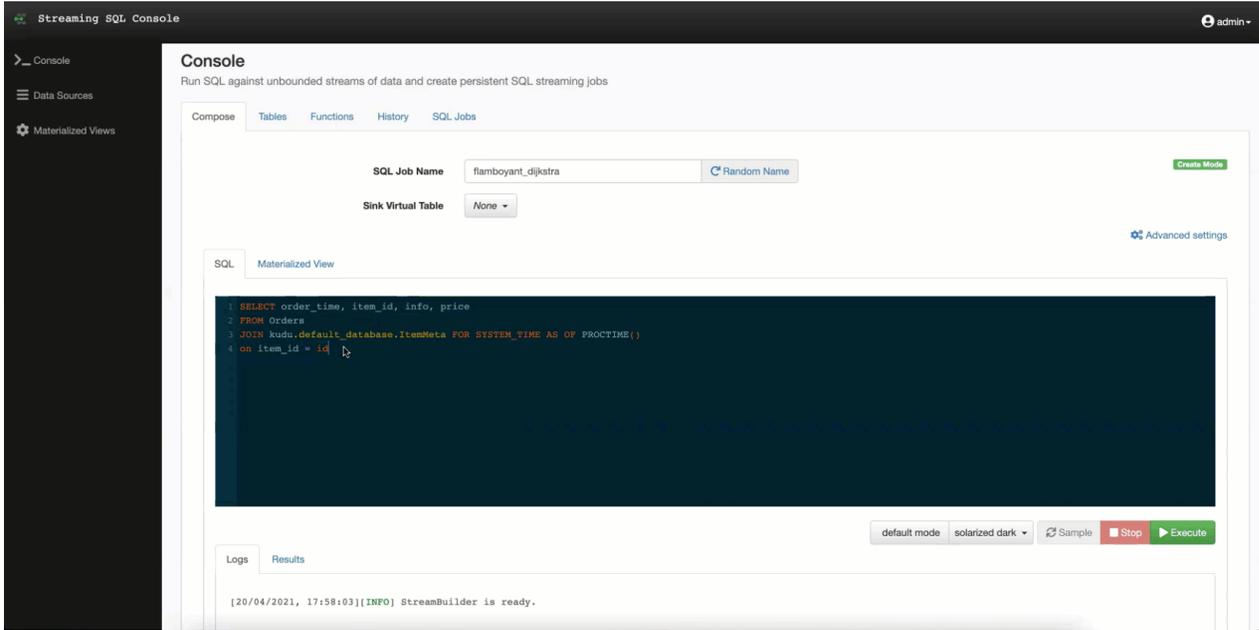
The tables in the example contain the following information:

Tables	Columns
Kafka - Orders	<ul style="list-style-type: none"> <li>• order_number</li> <li>• price</li> <li>• order_time</li> <li>• item_id</li> </ul>
Kudu - ItemMeta	<ul style="list-style-type: none"> <li>• id</li> <li>• info</li> </ul>

In the scope of this example, the *Orders* table will be joined with latest version of the *ItemMeta* table based on the item *id*, and the selected information is sampled under the **Results** tab of the Streaming SQL Console:

```
SELECT order_time, item_id, info, price
FROM Orders
JOIN kudu.default_database.ItemMeta FOR SYSTEM_TIME AS OF PROCTIME()
ON item_id = id
```

After running the SQL query, the results are continuously sampled to the Streaming SQL Console, the rows of *order\_time*, *item\_id* and *price* are enriched with *info* column from the *ItemMeta* Kudu table:



The screenshot displays the Cloudera Streaming SQL Console interface. The main area shows a SQL query in a dark-themed editor:

```
1 SELECT order_time, item_id, info, price
2 FROM Orders
3 JOIN kudu.default_database.ItemMeta FOR SYSTEM_TIME AS OF PROCTIME()
4 ON item_id = id
```

Below the query editor, there are controls for the job: "SQL Job Name" (flamboyant\_dijkstra), "Sink Virtual Table" (None), and "Advanced settings". The "Results" tab is active, showing a log entry: "[20/04/2021, 17:58:03][INFO] StreamBuilder is ready." The interface also includes a "Compose" sidebar on the left and a "Results" tab at the bottom.