Cloudera DataFlow for Data Hub 4.0.0

Using Cloudera SQL Stream Builder

Date published: 2019-12-16
Date modified: 2024-12-11



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. 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

Running a simple SQL job....6

Project structure and development....7
Creating a project....8
Navigating in a project....10
Managing member of a project....10
Source control of a project....11
Masking information before using source control....12
Setting the environment for a project....13
Importing a project....16

Registering Data Sources in Cloudera SQL Stream Builder....17
Adding Kafka Data Source....17
Adding Catalogs....20

Connectors....27
Using connectors with templates....27
Adding new connectors....28
Kafka connectors....29
CDC connectors....31
Webhook connector....32
Creating Webhook tables....32
JDBC connector....34
Filesystem connector....35
Datagen connector....35
Faker connector....35
Blackhole connector....35

Data formats....36
Adding data formats....36

Concept of tables in Cloudera SQL Stream Builder....37
Creating Kafka tables using wizard....38
Configuring Kafka tables....40
Creating Webhook tables....48
Creating Flink tables using Templates....50
Creating Iceberg tables....51

SQL jobs....53
Creating and naming SQL jobs....53
Running SQL Stream jobs....54
Configuring SQL job settings....57
Adjusting logging configuration in Advanced Settings....58

Configuring YARN queue for SQL jobs....62
Configuring state backend for Cloudera SQL Stream Builder....62
Managing session for SQL jobs....65
Executing SQL jobs in production mode....67

Functions....67
Creating Python User-defined Functions....67
Using System Functions....69

Introduction to Materialized Views....70
Creating Materialized Views....71
Configuring Retention Time for Materialized Views....75
Materialized View Pagination....77
Using Dynamic Materialized View Endpoints....78
Configuring Materialized View database information....80
Using Cloudera SQL Stream Builder with Cloudera Data Visualization....82

Using widgets for data visualization....83
Creating widgets....83
Choosing data sources....85
Managing data source jobs....87
Customizing visualization types....89
Configuring widget, fields and legend....89
Configuring graph style....94
Managing widgets on the Dashboard....115

Using SQL job notification....120

Using Cloudera SQL Stream Builder REST API....124

Monitoring SQL Stream jobs....127
Collecting diagnostic data....129
Using the OpenTelemetry Collector with Cloudera SQL Stream Builder [Technical Preview]....129

SSB metadata collection using Atlas....130

Flink SQL....130
Flink DDL....130
Managing time in Cloudera SQL Stream Builder....131
Flink DML....132
Flink Queries....132
Other supported statements....133
Data Types....133
Dynamic SQL Hints....134
SQL Examples....134

Enriching streaming data with join.....137
Joining streaming and bounded tables.....137
Example: joining Kafka and Kudu tables.....139
Updating SQL queries with PROCTIME function.....139

Running a simple SQL job

You can use this Getting Started use case to get familiar with the most simple form of running a SQL Stream job.

About this task

The Getting Started contains the basic steps of running a SQL Stream job. When executing the job, you do not need to select a sink as the results are displayed in the browser. Cloudera SQL Stream Builder provisions a job on your cluster to run the SQL queries. You can select the **Logs** tab to review the status of the SQL job. As data is returned, it shows up in the **Results** tab.

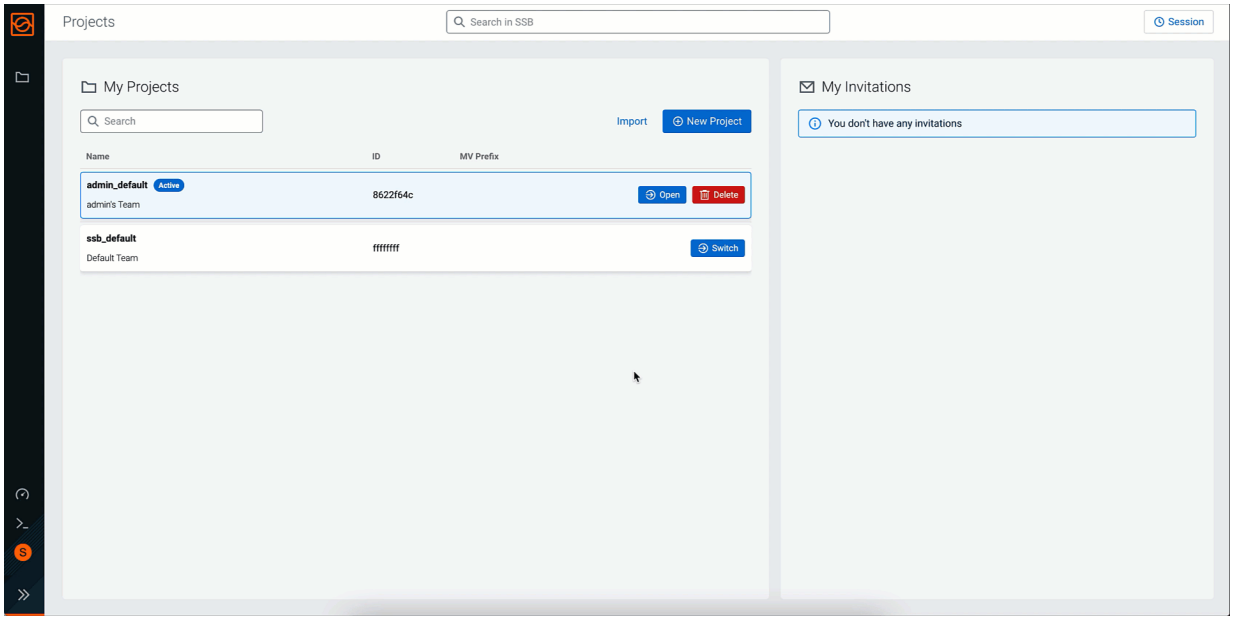
Procedure

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Jobs** from the **Explorer**.
4. Select New Job > Create .
   The SQL Editor for the created job opens in a tab.
5. Select *DATAGEN* from the Templates.
   The CREATE TABLE statement is imported to the SQL window.
6. Add a SELECT query to the SQL window after the datagen template.

```
SELECT * FROM <table_name>
```

   Note: When adding SQL statements to the SQL window, you do not need to add the semicolon (;) at the end of the statement as Cloudera SQL Stream Builder can run the command without the semicolons.

7. Click Execute.
   You can see the generated output on the Results tab.
8. Click on the Stop button to stop the job.



Project structure and development

Projects aim to provide a Software Development Lifecycle (SDLC) for streaming applications in Cloudera SQL Stream Builder: they allow developers to think about a task they want to solve using Cloudera SQL Stream Builder, and collect all related resources, such as job and table definitions or data sources in a central place.

Projects aim to facilitate collaboration between developers by sharing common resources among its members. Projects can be synchronized with a Git repository, allowing easy migration between different clusters. The environment concept in a project allows the templating of cluster-specific or sensitive properties.

A project is a collection of resources, static definitions of data sources, jobs with materialized views, virtual tables, user-defined functions (UDF), and materialized view API keys. These resources are called internal to a project and can be safely used by any job within the project.

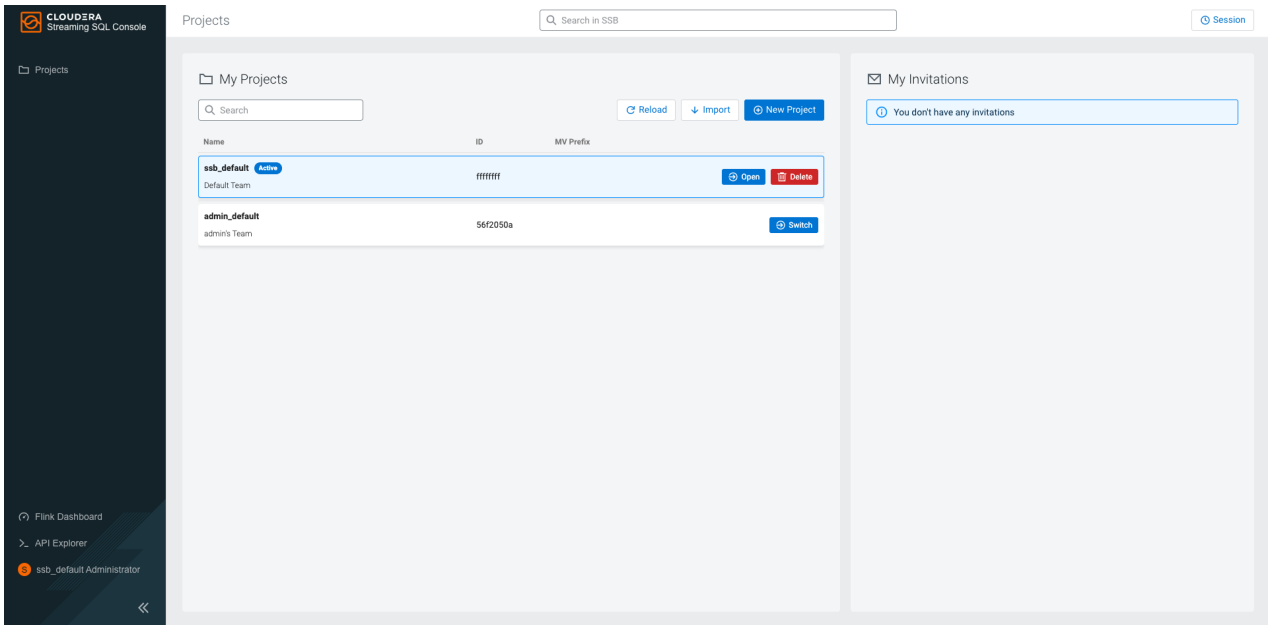
Jobs can also use external resources that the logged in user has access to. These resources are defined in other projects (for example, a UDF or a virtual table in another project), or can come from a Data Source (for example, a Kudu table from the Kudu catalog). External resources are outside the scope of the project, thus they can be changed or altered by external factors. It is recommended to use only internal resources for projects that are intended to be exported.

The following table summarizes the concepts and structural elements of projects in Cloudera SQL Stream Builder:

Project specific resources	
Jobs	Cloudera SQL Stream Builder job definitions including SQL, Materialized View configuration and endpoints, Job settings such as checkpointing and parallelism. The status of the job and its associated Flink job (if any) are not part of the definition.
Functions	User-defined functions (Python and Java).
Virtual Tables	Virtual tables of the project. These are stored in the *<PROJECT_NAME>* database of the ssb catalog. When exporting a project, only these tables (ssb.<project_name>.*) are exported.
Data Sources	Data Sources include Kafka providers and Catalogs in the project. While the definition of the Data Source is considered internal to the project, the tables or topics provided by them are external, as they are managed by an outside system.
Materialized Views	Materialized Views that have been defined for Jobs in the project.

API Keys	API Keys for accessing Materialized View endpoints.
Job Notifications	Job Notification actions (webhook and email notifications) as well as notification groups can be defined in a project. They may be assigned to Jobs to be triggered by failure. Job notifications are not synchronized when using the Git import/ export feature.
External resources	
Virtual Tables	Virtual tables from other projects and catalogs that the logged in user has access to. Other members of the active project might not have access to these.
Connectors	Connectors available in Cloudera SQL Stream Builder for connecting to external systems. These are shared among the whole SSB instance, and are accessible to all users in all projects.
Data Formats	Data Formats available in Cloudera SQL Stream Builder that can be used by connectors when connecting to external systems. These are shared among the whole Cloudera SQL Stream Builder instance, and are accessible to all users in all projects.

When you access the Streaming SQL Console, there are already existing projects on the **My Projects** page. The ssb_default and [***USERNAME***]_default projects are automatically generated projects. The ssb_default project and its resources are visible to every user, while the [***USERNAME***]_default project and its resources are only visible to that user. Every user who created an account in Cloudera SQL Stream Builder has its own project generated by default where members can be invited.



Creating a project

Before you can use Streaming SQL Console, you need to create a project where you can submit your SQL jobs and manage your project resources.

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 New Project on the **Projects** page of Streaming SQL Console.

   The **Create Project** window appears.
3. Provide a Name to the project.
4. Optional: Provide a Description to the project.
5. Optional: Provide a prefix that overrides the Materialized View table names.

   Providing a prefix to the Materialized View tables names that overrides the default one is an advanced option. The tables are prefix by default with the ID of the projects to avoid collision between projects. This configuration allows setting a custom prefix to the Materialized View tables created in the project.
6. Optional: Provide the **Source Settings** of the project.
   a) Add a remote Github repository URL to the **Clone URL** field.

      You can use the following example URLs with or without authentication:

      - To clone without authentication: https://github.com/cloudera/ssb-examples.git
      - To clone and push with HTTP basic authentication (username/password): https://github.com/cloudera/ssb-examples.git

        In case of using basic authentication, the personal access token needs to be provided as password. For more information about creating personal access tokens, see the Github documentation.
      - To clone and push with SSH authentication (SSH private key): git@github.com:cloudera/ssb-examples.git

        In case of using SSH authentication, you need to generate the private key using the following command:

        ```
        ssh-keygen -t ecdsa -b 256 -m pem -C "NAME@EXAMPLE.COM"
        ```

        As not every SSH private key is supported, you can use the following command to convert your key to the acceptable format:

        ```
        ssh-keygen -m pem -p -f [***PATH_TO_PRIVATE_KEY***]
        ```
      - To clone and push a local git repository on the filesystem of SSB host: file:///usr/src/ssb-examples/

   b) Provide the branch name or tag of the Github repository to the **Branch** field. This branch is checked out when pulling or pushing to the Github repository.

   Warning:

   When exporting a project, any sensitive information in CREATE TABLE statements, data sources, or SQL jobs will not be redacted. Ensure that usernames and passwords are hidden with the environment variables where required. For more information, see the Source control of a project documentation.
7. Enable or disable Allow deletions on import.

   Enabling the allow deletion upon import allows you to hard-reset your project to its versioned state when there are untracked resources in it. For example, the job is deleted from the project when importing the project from Github if the job does not exist in the repository. In other words, the upstream version overrides the version you have in Streaming SQL Console. If disabled, the project keeps the resources that were not pushed to the Github repository.

   The deletion affects the jobs, data sources, functions, Materialized Views and their API keys.
8. Enable or disable Authentication for the project.

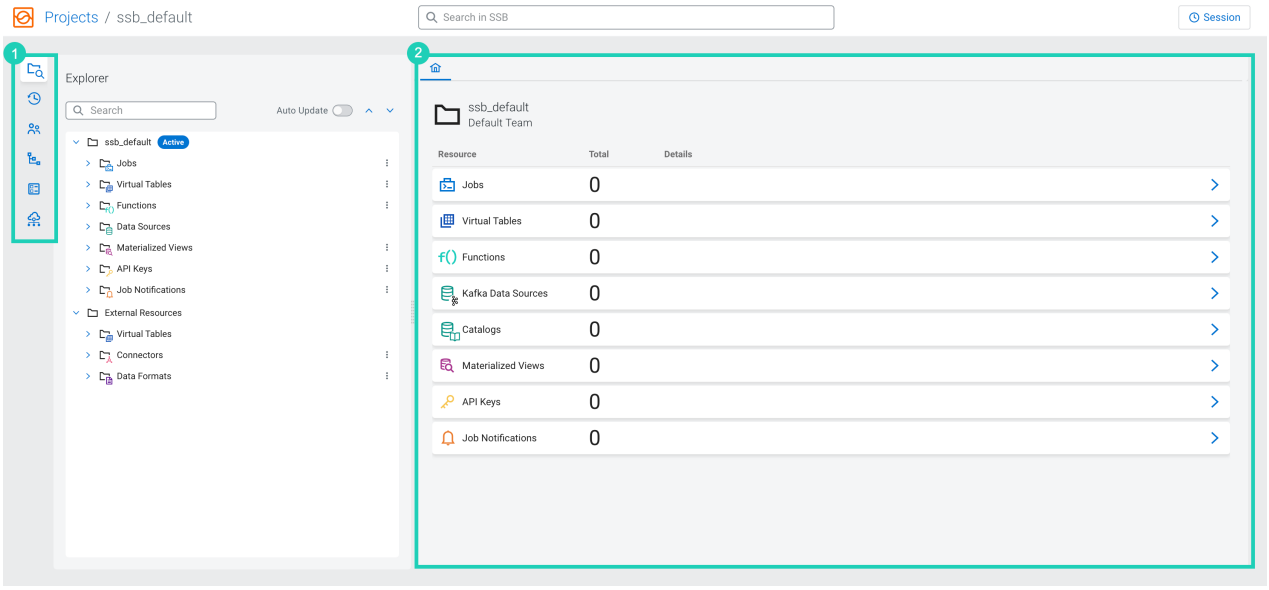
What to do next

After you have created the project, you have the following options to manage the project on the **Projects** page:

1. Click Open to use the created project and start creating SQL jobs.
2. Click Switch to open another project than the active one.
3. Click Delete to remove the project from Cloudera SQL Stream Builder.

Navigating in a project

When you open a project in Streaming SQL Console, you can navigate between the Project Manager and the Tab view of the Project Resources.



1. The **Project Manager** consists of the following views:
   - **Explorer** - tree view of the user defined and external resources of the project
   - **SQL History** - history of the queries that have been executed by the logged in user
   - **Members** - list of the users that are invited in the project
   - **Source Control** - configure the synchronization source (Git repository) for the project, import or export changes
   - **Environments** - manage environment configurations, define variables for templating
   - **Data Hub Service Discovery** - automatic discovery of Kafka, Kudu and Schema Registry services in your Cloudera on cloud environment
2. The **Home** tab of a project allows you to quickly access the user defined resources of a project. It also gives you a quick view about the total number of resources in each category with resource specific details. You can open the different resource tabs easily from the **Home** tab to create and manage items in a given resource category.

Managing member of a project

Created or imported projects can be shared with other users in Streaming SQL Console. You can invite members using their Streaming SQL Console username and set the access level to member or administrator.

About this task

When you create a project, you automatically become the owner of that project. You can invite Cloudera SQL Stream Builder users to your project and give them access to the jobs and resources. You can also set another user to be the admin of your project.

Table 1: Access levels in a project

Member	Creating and managing jobs and resources in a project, synchronizing the project, inviting other Members. Running jobs, stopping their own jobs.
Admin	In addition to Member privileges, Admins can stop jobs of other users, manage users of the project, invite other Admins or Members.

Owner	The priviliges of owners are identical to the Admins, but it cannot be revoked and Owners cannot be removed from the project.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click Members from the Project Manager.
4. Click Invite User.
   The **Invite Member** window appears.
5. Provide the Cloudera SQL Stream Builder Username of the member you want to invite.
6. Set the Access Level for the user.
7. Click Send Invitation.
   The sent invitation is visible on the **Projects** page of Streaming SQL Console under **My Invitations** or at the **Invitations** tab on the **Members** page.

   You can delete any members of a project that you are an owner or an admin of using  button next to their username.

Source control of a project

When you provide a Git repository and branch for a project, you can push and pull your project to Git. This enables you to track your versions of your project, create a backup of your project in Git and manage the development lifecycle of your project.

You can provide the Git repository URL and branch for a project when you create one or when you import a project on the **Projects** page of Streaming SQL Console.



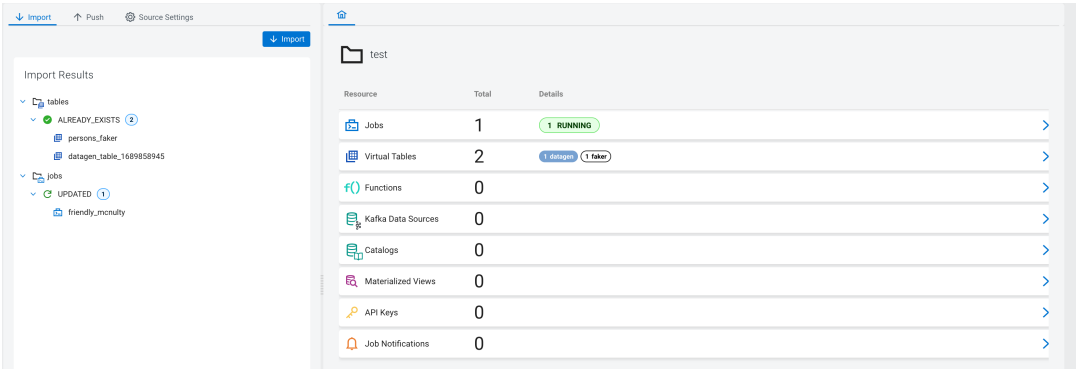
Note: The Source Control feature is in Technical Preview and not ready for production deployment. Cloudera encourages you to explore these features in non-production environments and provide feedback on your experiences through the *Cloudera Community Forums*.

After setting up your remote repository, you can navigate to Source Control in your project, and use the Import and Push buttons to synchronize your local state with the state in the repository.

Importing a version of project

When you import a project from Git to Cloudera SQL Stream Builder in Source Control, the configured Git repository is cloned to a temporary directory by Cloudera SQL Stream Builder, the specified branch is checked out, and the resource files found in the directory with the project's name are used to import resources to the Cloudera SQL Stream Builder project.

After pulling a version of a project, the results are listed based on the imported changes that include **UPDATED**, **DELETED**, **CREATED** or **ALREADY EXISTS**.



Warning: In case the imported changes affect a running job, the job must be restarted for the changes to take effect.

Pushing a version of project

When you push a project from to Git in Source Control, the configured repository is cloned to a temporary directory by SSB, and the project resources are written to files in a directory with the project's name into the temporary directory.

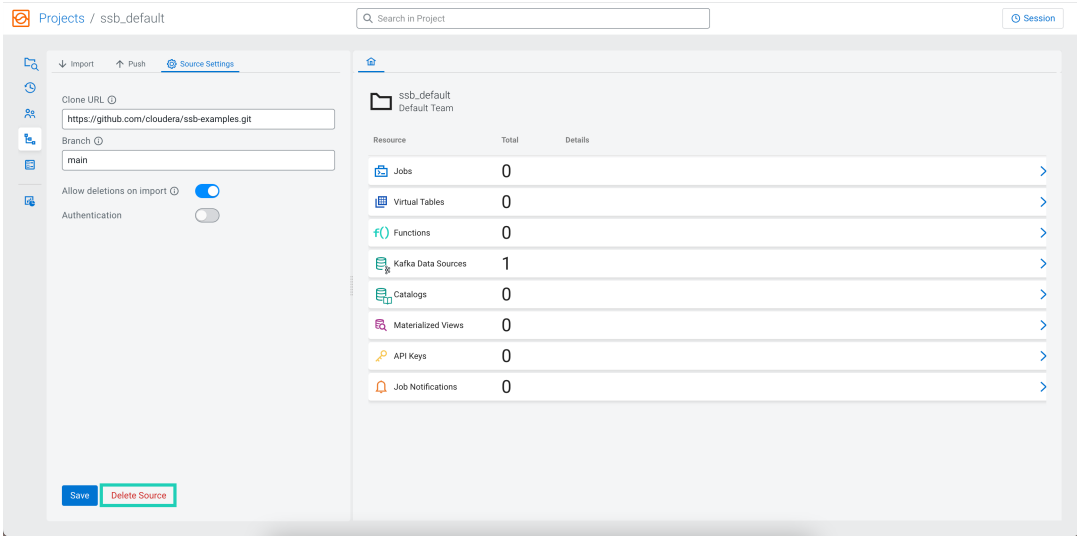
Changes are added and committed with the specified commit message, and pushed to the configured remote.

Changing the project settings

You can manage the source information of a project under the **Source Settings** tab. You can change the **Clone URL** and **Branch** information of the Git repository. You can enable **Allow deletions** for importing a project, and configure the **Authentication** method that is used when interacting with the repository.

Removing source settings

In case the source control settings need to be removed, you can delete the configurations for the repository using the Delete Source button.



Masking information before using source control

When exporting projects from Cloudera SQL Stream Builder, possible sensitive parameters in CREATE TABLE statements, data sources, or the SQL job will not be redacted. To conceal sensitive information, environment variables can be used.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.

   You are redirected to the **Explorer** view of the project.
3. Click Environments from the Project Manager.
4. Click New Environment.
5. Provide a Name for the environment.

   For example, you can name it Dev.
6. Add keys and values as needed.

   For example, you can add psql_password as key, and provide the password as value.
7. Click Activate to apply the specified information for the project.
8. When creating the table, reference the environment variable as shown in the following example:

```
CREATE TABLE  `postgres_cdc_table_1682585103` (
 `order_id` INT,
 `city` VARCHAR(2147483647),
 `street_address` VARCHAR(2147483647),
 `amount` INT,
 `order_time` TIMESTAMP(3),
 `order_status` VARCHAR(2147483647)
) WITH (
 'connector' = 'postgres-cdc',
 'PASSWORD' = '${SSB.ENV.PSQL_PASSWORD}'
 'username' = '...'
);
```

Setting the environment for a project

Environments are sets of user-defined variables that can be substituted into templates in your Cloudera SQL Stream Builder resources. Environments are like configuration files for your projects that adapt it to be used in a specific environment. They enable you to keep the common resources and logic of a project in a central repository and reuse it across different environments, different clusters.

About this task

Creating an environment file for a project means that users can create a template with variables that could be used to store environment-specific configuration.

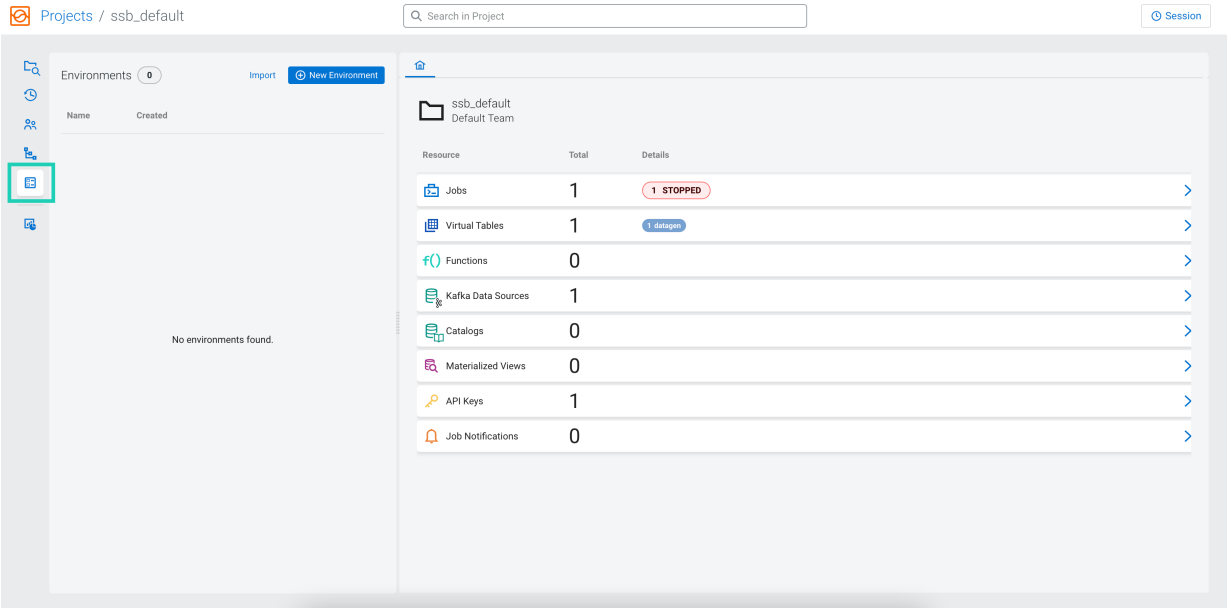
For example, you might have a development, staging and production environment, each containing different clusters, databases, service URLs and authentication methods. Projects and environments allow you to write the logic and create the resources once, and use template placeholders for values that need to be replaced with the environment specific parameters.

To each project, you can create multiple environments, but only one can be active at a time for a project. Environments can be exported to files, and can be imported again to be used for another project, or on another cluster.

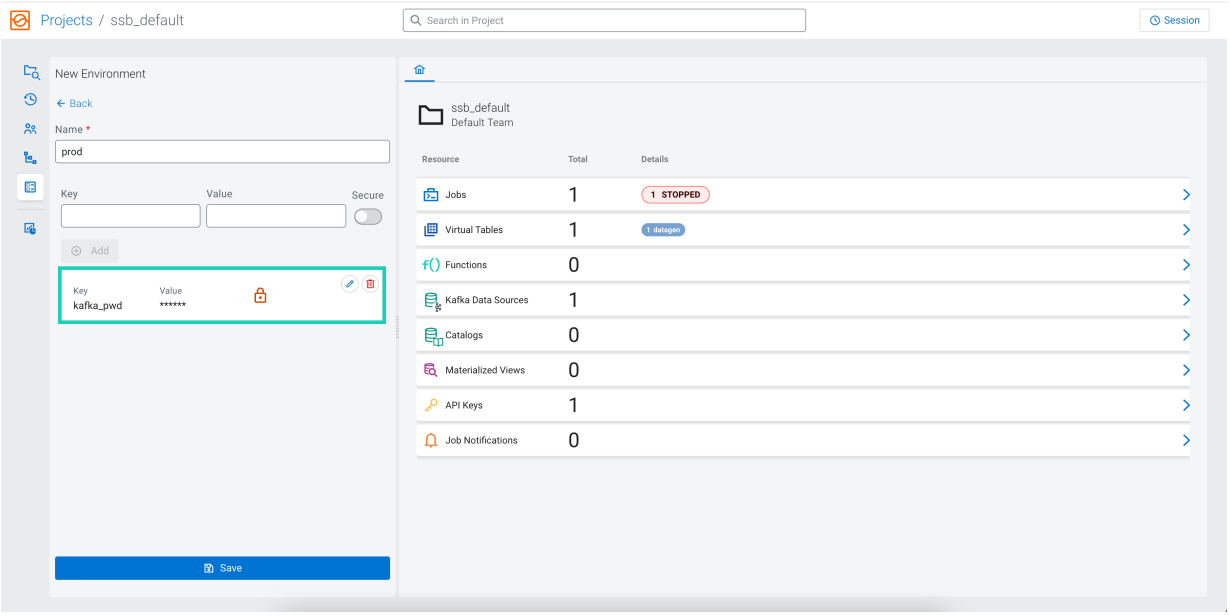
While environments are applied to a given project, they are not part of the project. They are not synchronized to Git when exporting or importing the project. This separation is what allows the storing of environment-specific values, or configurations that you do not want to expose to the Git repository.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click Environments from the Project Manager.



4. Click New Environment.
5. Provide a Name for the environment.
6. Add the Key and Value pairs as needed.
7. Enable or disable masking for the values using the Secure toggle.
   The Secure toggle allows you to hide sensitive information provided as values on the Streaming SQL Console as shown in the following example:



8. Click Add to save the key and value pair.
9. Click Create when you added the necessary environment variables.

   After creating an environment file, you have the following options to manage it:

   a. Click Activate to apply the specified information for the project. To deactivate an active environment file, you need to click the Deactivate button.

   b. Click ⋮ Export to export the environment file.

   c. Click Import and search for an exported environment file on your computer to import an environment file.

   d. Click ⋮ Delete to remove the environment file from the project.

What to do next

You can use a variable defined in your active environment in the following ways:

- As values in the properties section of a CREATE TABLE statement as shown in the following example:

```
CREATE TABLE (...) WITH ('key1'='${ssb.env.yourVariable}')
```

- In the properties of a Kafka data source, for example bootstrap servers, truststore location
- In the values of Catalog properties, for example Schema Registry URL, Kudu masters, Custom Catalog

To refer to an environment variable called *YOURVARIABLE*, you can use the following syntax: ${ssb.env.yourVariable}. You can use multiple substitutions or combine literals and variable substitutions in the same value string as shown in the following example:

```
'some-connector-url' = 'https://${ssb.env.connector-host}:${ssb.env.connector-port}'
```

Besides environments, it is also possible to define additional variables using the SET_VAR 'x' = 'y'; syntax in your queries. These do not become part of the environment and are not persisted, but live only in the current SQL

session. They can be used for substitutions like environment variables, but without the ssb.env prefix as shown in the following example:

```
SET_VAR 'yourVariable' = 'someValue';
CREATE TABLE (...) WITH ('key1'='${yourVariable}'))
```

You can reset a variable with the RESET_VAR 'x'; command.

Importing a project

You can use a Git repository to import a Cloudera SQL Stream Builder project that was previously created and exported using source control.

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 Import Project on the **Projects** page of Streaming SQL Console.

   The **Import Project** window appears.
3. Optional: Provide a prefix that overrides the Materialized View table names.

   Providing a prefix to the Materialized View tables names that overrides the default one is an advanced option. The tables are prefix by default with the ID of the projects to avoid collision between projects. This configuration allows setting a custom prefix to the Materialized View tables created in the project.
4. Provide the **Source Settings** of the project.
   a) Add a remote Github repository URL to the **Clone URL** field.

      You can use the following example URLs with or without authentication:

      - To clone without authentication: https://github.com/cloudera/ssb-examples.git
      - To clone and push with HTTP basic authentication (username/password): https://github.com/cloudera/ssb-examples.git

        In case of using basic authentication, the personal access token needs to be provided as password. For more information about creating personal access tokens, see the Github documentation.
      - To clone and push with SSH authentication (SSH private key): git@github.com:cloudera/ssb-examples.git

        In case of using SSH authentication, you need to generate the private key using the following command:

        ```
        ssh-keygen -t ecdsa -b 256 -m pem -C "NAME@EXAMPLE.COM"
        ```

        As not every SSH private key is supported, you can use the following command to convert your key to the acceptable format:

        ```
        ssh-keygen -m pem -p -f [***PATH_TO_PRIVATE_KEY***]
        ```
      - To clone and push a local git repository on the filesystem of SSB host: file:///usr/src/ssb-examples/

   b) Provide the branch name or tag of the Github repository to the **Branch** field. This branch is checked out when pulling or pushing to the Github repository.
5. Provide the name of the project to the **Project** field.
6. Enable or disable Allow deletions on import.

   Enabling the allow deletion upon import allows you to hard-reset your project to its versioned state when there are untracked resources in it. For example, the job is deleted from the project when importing the project from Github

if the job does not exist in the repository. In other words, the upstream version overrides the version you have in Streaming SQL Console. If disabled, the project keeps the resources that were not pushed to the Github repository.

The deletion affects the jobs, data sources, functions, Materialized Views and their API keys.

7. Enable or disable Authentication for the project.
8. Click Import.

What to do next

After you have created the project, you have the following options to manage the project on the **Projects** page:

1. Click Open to use the created project and start creating SQL jobs.
2. Click Switch to open another project than the active one.
3. Click Delete to remove the project from Cloudera SQL Stream Builder.

Registering Data Sources in Cloudera SQL Stream Builder

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

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

Adding Kafka Data Source

To create Kafka tables in Cloudera SQL Stream Builder you need to register Kafka as a Data Source, using the Streaming SQL Console.

About this task

A default Local Kafka is added to the Cloudera SQL Stream Builder data sources during installation, using a Kafka service within the same cluster as Cloudera SQL Stream Builder. This Local Kafka data source cannot be updated or delete, as it is used in the Streaming SQL Console for sampling results and cleaning up sample topics. To add your own, customizable Kafka data source instead, follow the steps in this task.

Before you begin

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

Procedure

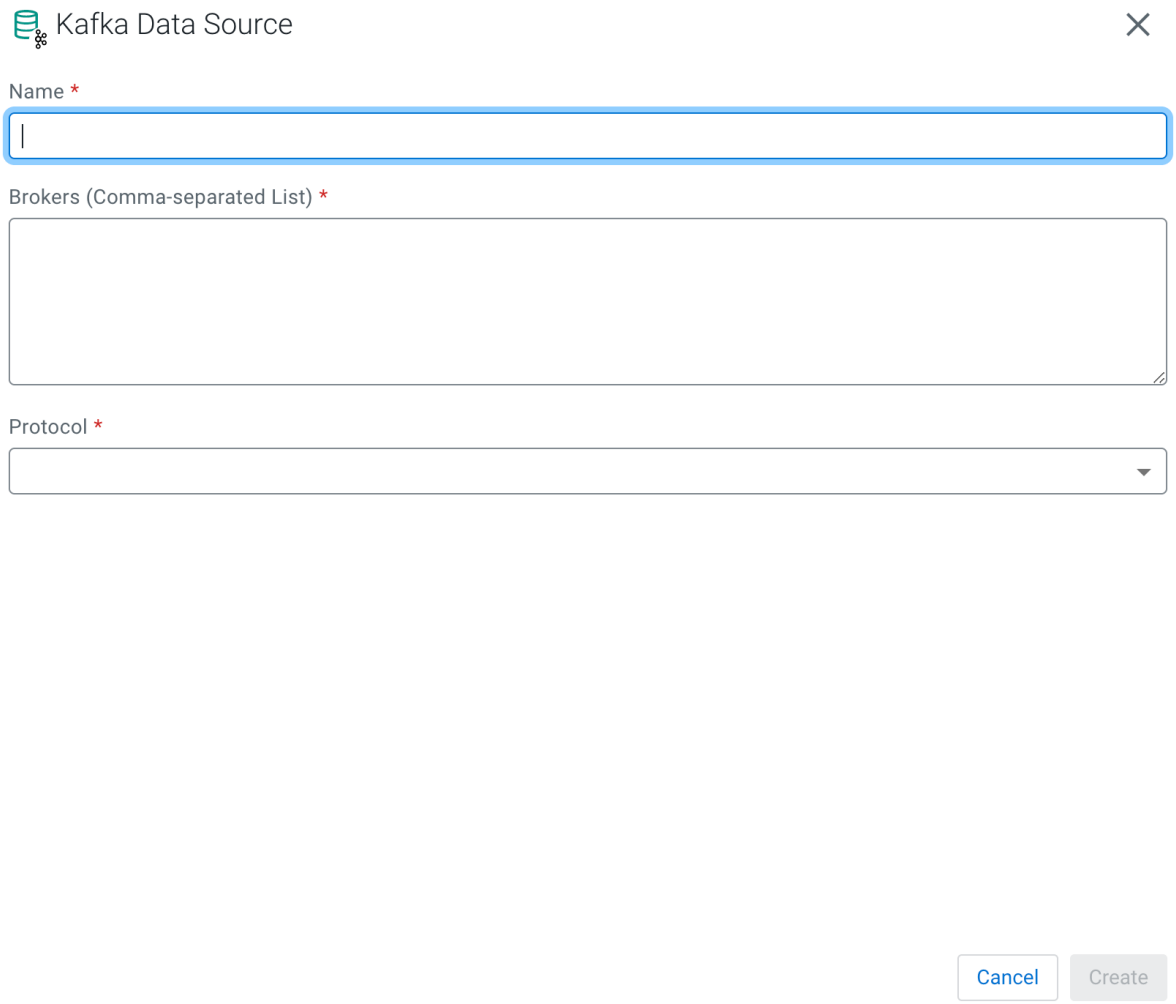
1. Navigate to the Streaming SQL Console.
   a) Go to your cluster in Cloudera Manager.
   b) Select SQL Stream Builder from the list of services.
   c) Click SQLStreamBuilder Console .

   The **Streaming SQL Console** opens in a new window.
2. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.

You are redirected to the **Explorer** view of the project.

3. Open Data Sources from the **Explorer** view.
4. Click next to **Kafka**.
5. Select New Kafka Source.

   The Kafka Source window appears.



6. Add a Name to your Kafka provider.
7. Add the broker host name(s) to Brokers.

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

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

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

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

Example:

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

8. Select the security 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:

   For PLAINTEXT

   a. Click Validate.
   b. Click Create after validation is successful.

   For SSL

   a. Choose your authentication method:

      By default the auto-discovery of Cloudera TrustStore is enabled. The auto-discovery of Cloudera TrustStore can be used for Kafka sources that are located in the same Cloudera Base on premises cluster as Cloudera SQL Stream Builder. This means that the default TrustStore path is used for authentication, which can be customized in Cloudera Manager using local.kafka.truststore.location and local.kafka.truststore.password parameters.

      Note: If you change the default value of local.kafka.truststore.location and local.kafka.truststore.password parameters, the same value will be applied to the Local Kafka data source as well.

      If you disable the auto-discovery, you need to provide the following configurations:

      - Kafka TrustStore path
      - Kafka TrustStore Password
      - Kafka TrustStore Type (select JKS, PKCS12, or BCFKS)

      You can also provide the Kafka KeyStore path, KeyStore Password, and KeyStore Type that belong to the Kafka source.

      You can specify Custom Properties directly in the Kafka Data Source form to further configure the connection.

   b. Click Validate.
   c. Click Create after validation is successful.

   For SASL SSL

   a. Choose your authentication method:

      By default the auto-discovery of Cloudera TrustStore is enabled. The auto-discovery of Cloudera TrustStore can be used for Kafka sources that are located in the same Cloudera Base on premises cluster as Cloudera SQL Stream Builder. This means that the default TrustStore path is used for authentication, which can be customized in Cloudera Manager using local.kafka.truststore.location and local.kafka.truststore.password parameters.

      Note: If you change the default value of local.kafka.truststore.location and local.kafka.truststore.password parameters, the same value will be applied to the Local Kafka data source as well.

      If you disable the auto-discovery, you need to provide the following configurations:

      - Kafka TrustStore path
      - Kafka TrustStore Password
      - Kafka TrustStore Type (select JKS, PKCS12, or BCFKS)

You can also provide the Kafka KeyStore path, KeyStore Password, and KeyStore Type that belong to the Kafka source.

You can specify Custom Properties directly in the Kafka Data Source form to further configure the connection.

b. Choose an SASL Mechanism.
c. Click Validate.
d. Click Create after validation is successful.

For SASL PLAINTEXT

a. Choose an SASL Mechanism.
b. Provide the Username for SASL.
c. Provide the Password for SASL.
d. Click Validate.
e. Click Create after validation is successful.

Results

You have registered Kafka as a data source 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.

What to do next

After registering the Kafka data source, you can edit, duplicate, and delete it from the Streaming SQL Console:

1. Open Data Sources from the **Explorer** view.
2. Click next to **Kafka**.
3. Select Manage.

   The **Kafka Sources** tab opens where the registered Kafka providers are listed. You have the following options to manage the Kafka sources:

   - Click on one of the existing Kafka providers to edit its configurations.
   - Click  to remove the Kafka provider.
   - Click  to duplicate the Kafka provider with its configurations.

Related Information

Configuring Ranger policies

Adding Catalogs

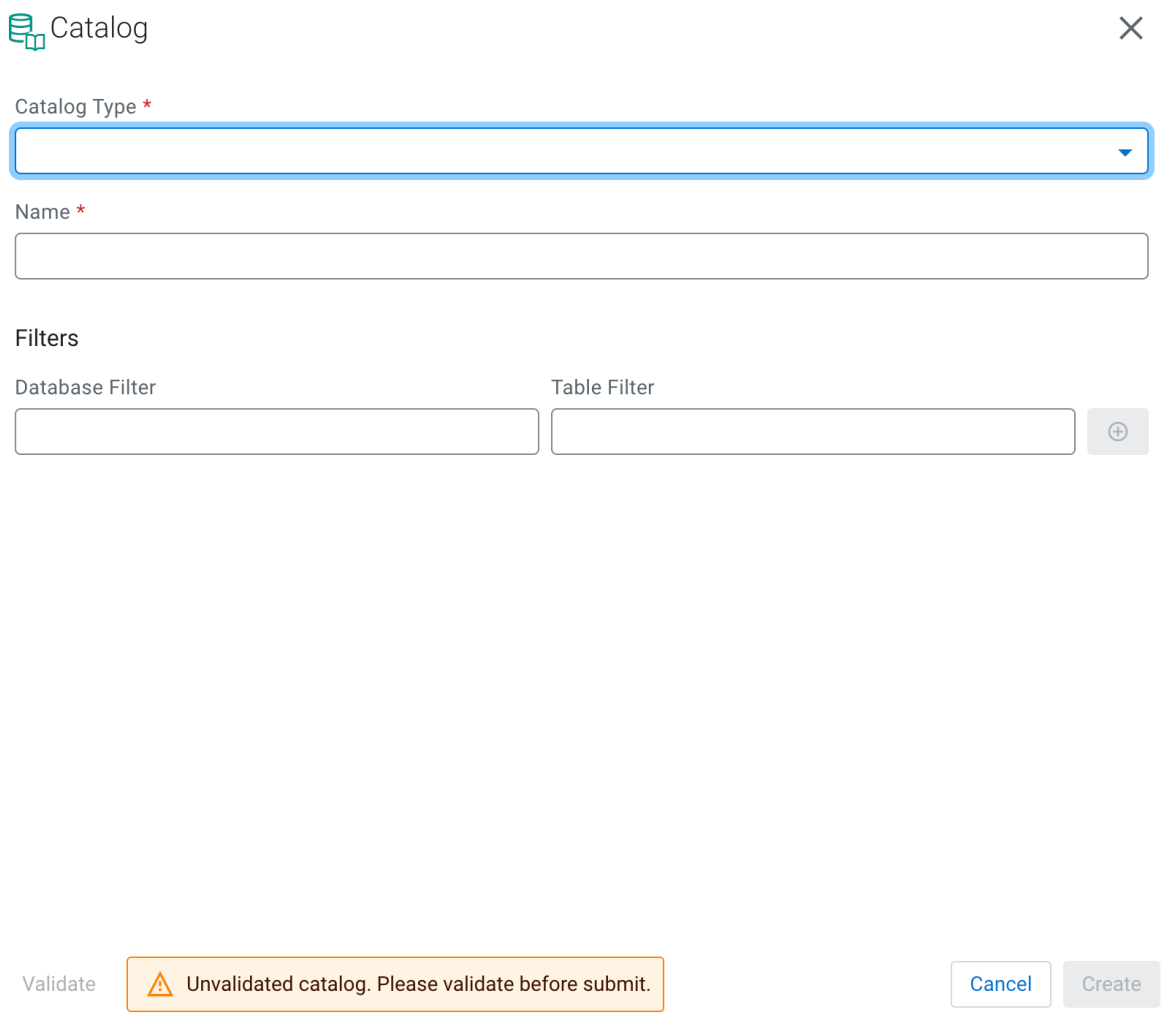
You need to add Cloudera Schema Registry, Kudu, Hive, Confluent Schema Registry or other services as a Catalog using the Streaming SQL Console in Cloudera SQL Stream Builder to use them with Flink DDL.

Before you begin

- Make sure that you have the required service on your cluster.
- Make sure that you have the right permissions set in Ranger for Cloudera SQL Stream Builder and the services.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Open Data Sources from the **Explorer** view.
4. Click next to **Catalogs**.
5. Select New Catalog.
   The Add Catalog window appears.

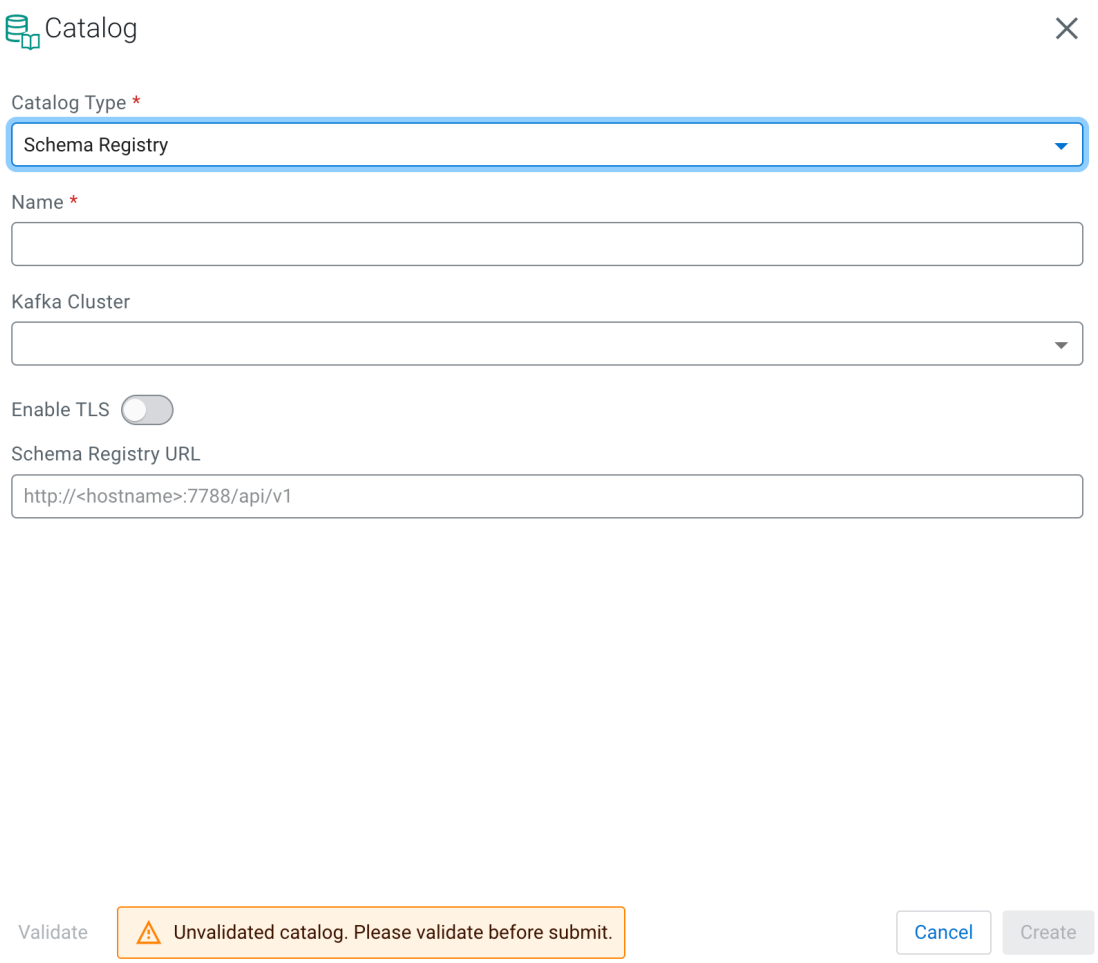


6. Select the Catalog Type from the following options:

For Schema Registry

a. Add a Name to your catalog.

b. Select Schema Registry from the **Catalog Type** drop-down.



c. Select the Kafka cluster you registered as Data Source.

d. Enable TLS, if needed for the communication.

1. If you enabled TLS, provide the Schema Registry Truststore location and password to the SR TrustStore and SR TrustStore Password field.

e. Add the Schema Registry URL.

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

   Example:

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

f. Optional: Specify a Schema key suffix and a Schema value suffix.

By default, the Schema Registry catalog looks for a topic in Kafka with the same name as the schemas stored in Cloudera Schema Registry. It uses the schema to deserialize the payload of the messages in that topic and ignores the message key. If you have different schemas to deserialize message key and payload

you can define suffixes to differentiate them.For example, if you configure the catalog's schema key suffix as -key and the schema value suffix as -value, and you have schemas stored in Cloudera Schema Registry named example-key and example-value. The Cloudera SQL Stream Builder catalog will automatically create a table called example and use those schemas to deserialize the key and the payload of the messages, respectively.

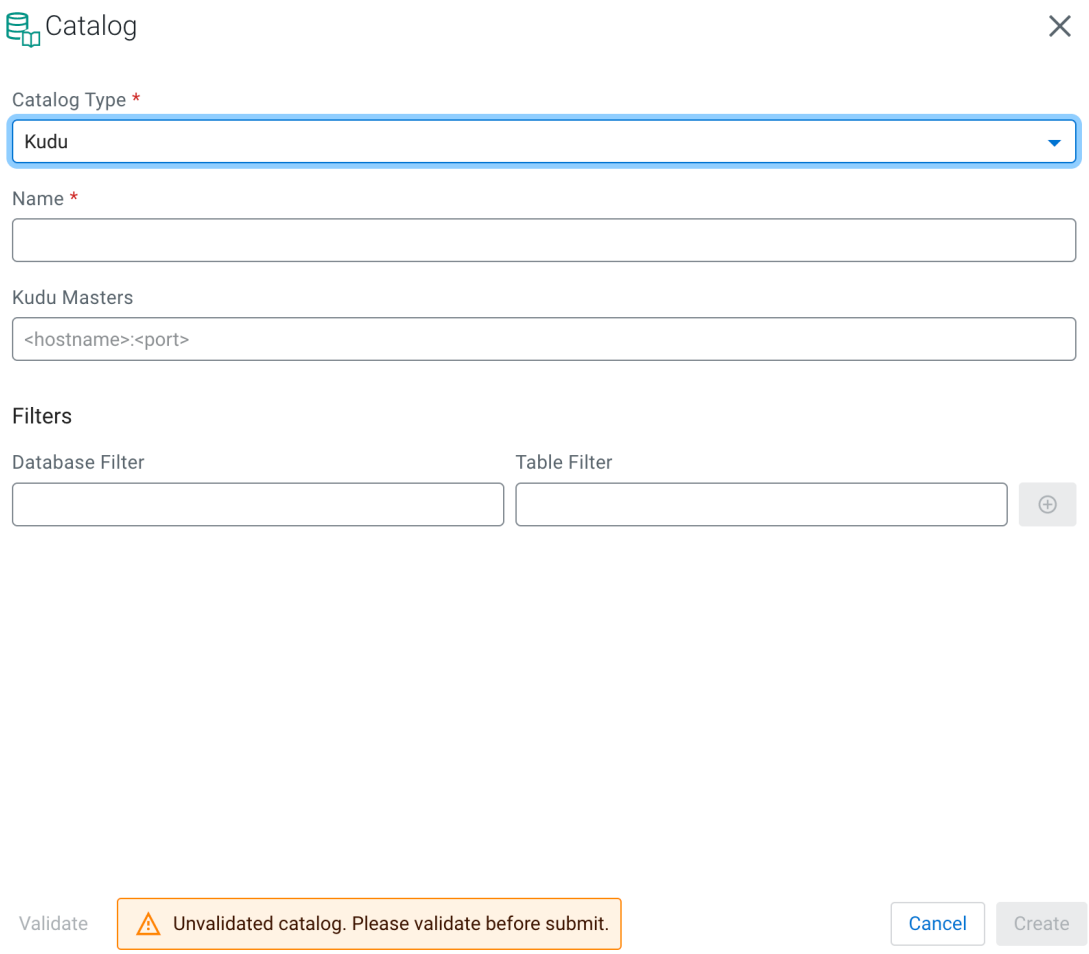
g. Optional: Add a Prefix for key fields in schema.

You can set a prefix for key field names to avoid table creation failure as the column names in the Flink schema are created based on the field names of the Avro schema. Table creation fails if the key and value schemas contain overlapping fields. The provided prefix will be assigned to the key fields of the Flink table as shown in the following example:

```
CREATE TABLE … (k_a INT, k_b INT, b INT, c INT0 with (..., 'key.fiel
ds'='k_a;k_b')
```

For Kudu

a. Select Kudu from the Catalog Type drop-down.



b. Add the host URL of Kudu Masters.

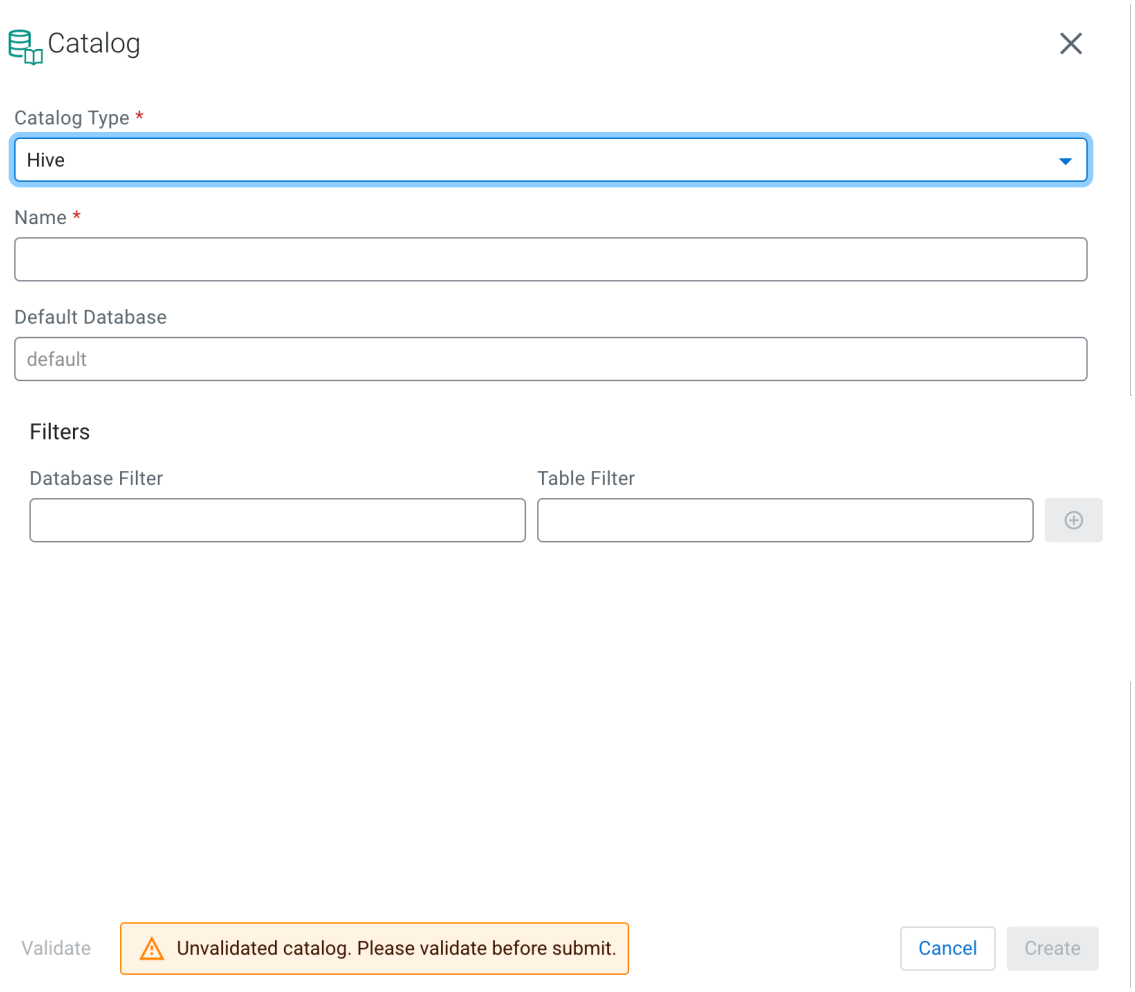
1. Go to your cluster in Cloudera Manager.
2. Select Kudu from the list of services.
3. Click on Instances.
4. Copy the Hostname of the Master Default Group.
5. Add the default port of Kudu after the hostname.

Example:

```
docs-test-1.vpc.cloudera.com:7051
```

For Hive

a. Select Hive from the Catalog Type drop-down.

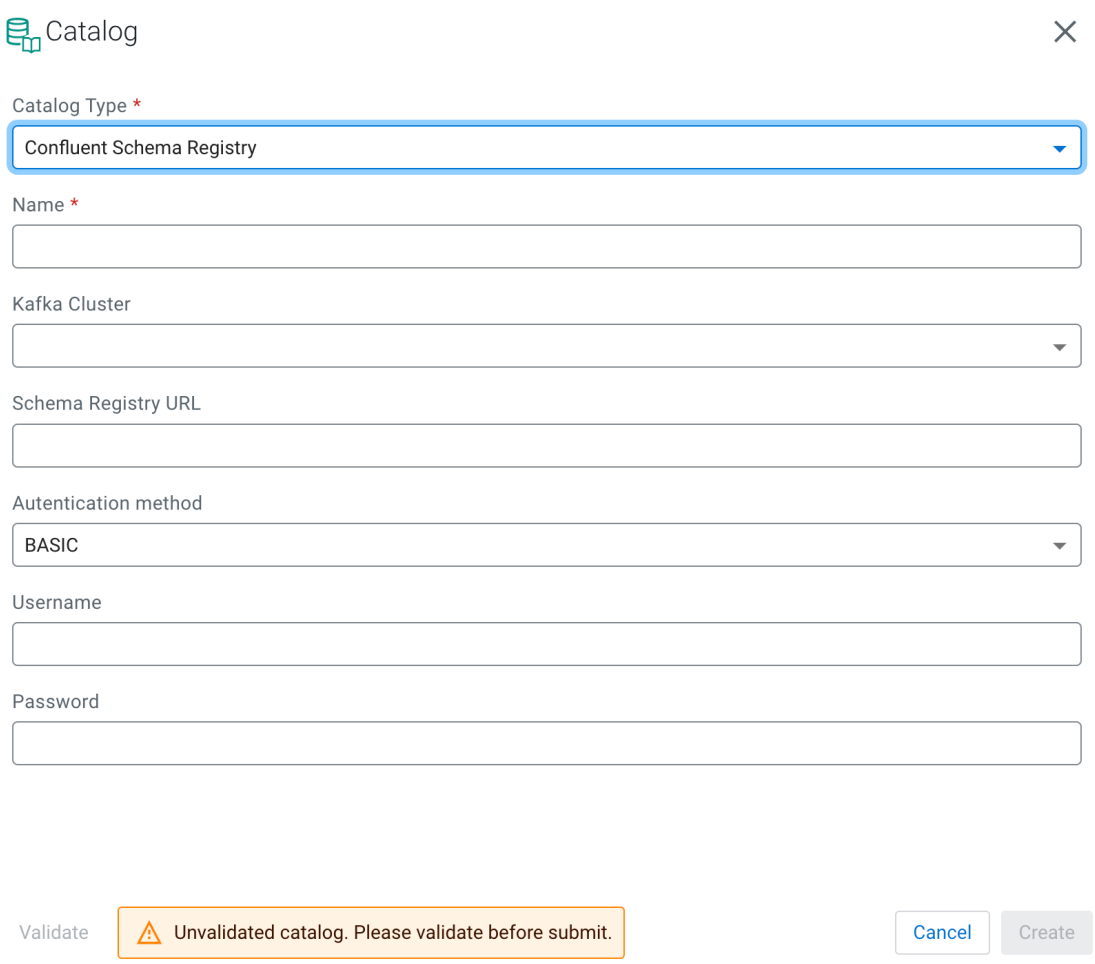


b. Provide a Name to the Hive catalog created in SSB.

c. Provide the name of the Default Database in Hive.

For Confluent Schema Registry

a. Select Confluent Schema Registry from the **Catalog Type** drop-down.



b. Add a Name to your catalog.
c. Select the Kafka cluster you registered as Data Source.
d. Add the Confluent Cloud Schema Registry URL provided by your Confluent connection configurations.
e. Choose an Authentication method from the following:

BASIC

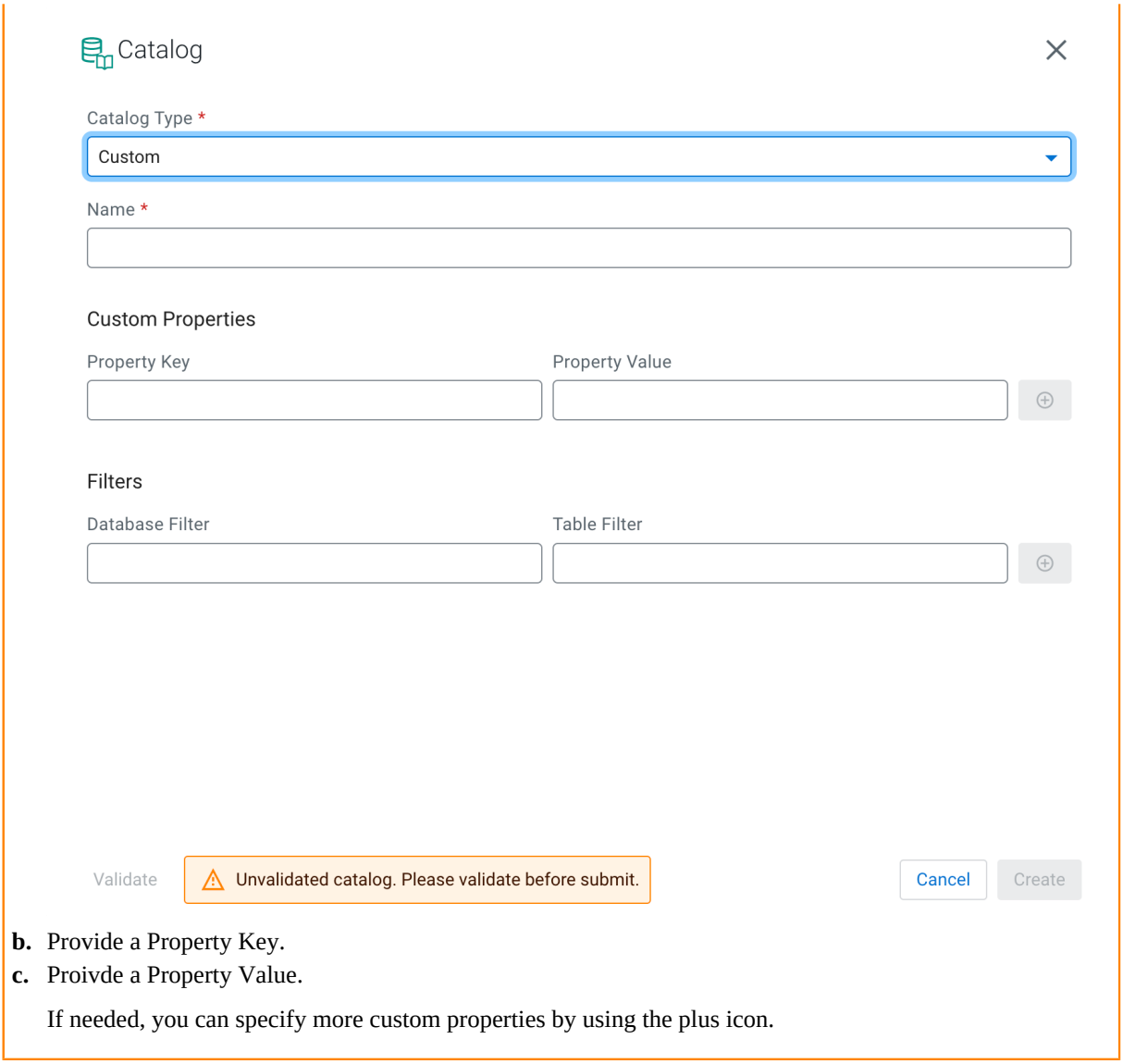
If the basic authentication method is selected, you must provide the username and password.

CUSTOM

Custom authentication methods include bearer authentication, SSL authentication and so on. If the custom authentication method is selected, you must provide the authentication property keys to the **Custom Properties** field as they are set for the Confluent Cloud Schema Registry. For example, if the Confluent Cloud Schema Registry is configured with bearer authentication, you need to provide the bearer.auth.credentials.source and bearer.auth.token. If SSL authentication is configured, the schema.registry.ssl.keystore.location, schema.registry.ssl.keystore.password and so on must be provided.

For Custom

a. Select Custom from the Catalog Type drop-down.



b. Provide a Property Key.
c. Proivde a Property Value.

If needed, you can specify more custom properties by using the plus icon.

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 Create.

Results

You are ready to use the added catalog in Cloudera SQL Stream Builder with Flink DDL. The already existing schemas in Schema Registry and Confluent Schema Registry, tables in Kudu and Hive are automatically imported to SSB.

What to do next

After registering a catalog, you can edit, duplicate and delete it from Streaming SQL Console:

1. Open Data Sources from the **Explorer** view.
2. Click next to **Catalogs**.
3. Select Manage.

The **Catalogs** tab opens where the registered catalogs are listed. You have the following options to manage the catalog sources:

- Click on one of the existing catalogs to edit its configurations.
- Click  to remove the catalog.
- Click  to duplicate the catalog with its configurations.

Related Information

Configuring Ranger policies

Connectors

Cloudera SQL Stream Builder supports different connector types and data formats for Flink SQL tables to ease development and access to all kinds of data sources.

The following table summarizes the supported connectors and how they can be used in Cloudera SQL Stream Builder:

Connector	Type	Description
Kafka	source/sink	Supported as exactly-once-sink
Hive	source/sink	Can be used as catalog
Kudu	source/sink	Can be used as catalog
Schema Registry	source/sink	Can be used as catalog
JDBC	source/sink	Can be used with Flink SQL. PostgreSQL, MySQL and Hive are supported.
Filesystems	source/sink	Filesystems such as HDFS, S3 and so on. Can be used with Flink SQL
Debezium CDC	source	Can be used with Flink SQL. PostgreSQL, MySQL, Oracle DB, Db2 and SQL Server are supported.
Webhook	sink	Can be used as HTTP POST/PUT with templates and headers
PostgreSQL	sink	Materialized View connection for reading views. Can be used with anything that reads PostgreSQL wire protocol
REST	sink	Materialized View connection for reading views. Can be used with anything that reads REST (such as notebooks, applications, and so on)
BlackHole	sink	Can be used with Flink SQL.

Using connectors with templates

Some of the connectors have default templates in Streaming SQL Console that allows you to create tables with them easily. These predefined templates not only contain the CREATE TABLE statement, but every mandatory and optional argument that is needed for the table.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Jobs** from the **Explorer**.
4. Select New Job > Create .
   The SQL Editor for the created job opens in a tab.
5. Click Templates.
6. Select one of the connector templates.
   The corresponding CREATE TABLE statement is imported to the **SQL Editor**.

What to do next

After importing the template to the SQL Editor, you need to provide the mandatory properties and you can also fill out the optional arguments if needed. When you click Execute, the table of the chosen connector is created and listed under **Virtual Tables**.

Adding new connectors

When adding new connectors to Cloudera SQL Stream Builder, you need to specify the property list, data types and must upload the connector JAR file to the Console.

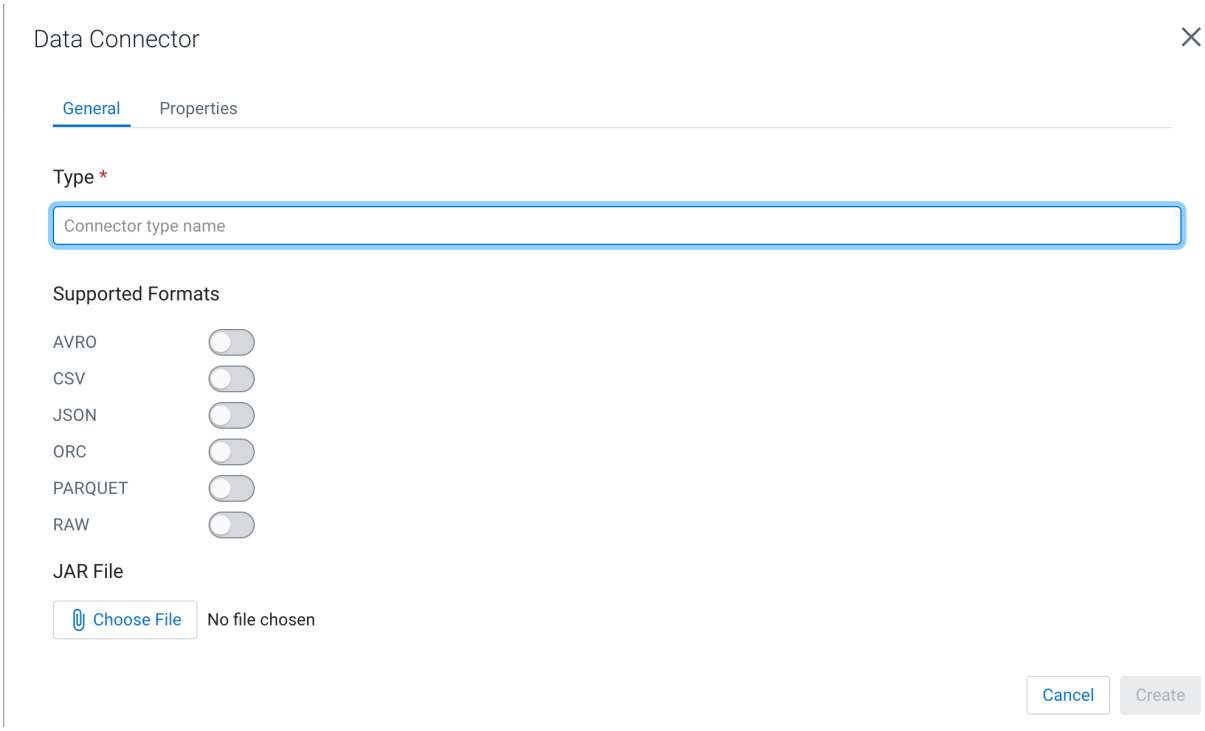
Before you begin

Adding connectors requires admin privileges.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Open **External Resources** from the **Explorer** view.

4. Click next to **Connectors**.
5. Click New Connector.



6. Provide a name for the connector as Type.
7. Select a data format to be supported for the connector from the Supported Formats.
8. Upload the connector JAR file.
9. Click Properties to add properties to the connector.
   a) Add a name to the property.
   b) Add a default value to the property.
   c) Add a description to the property.
   d) Click Required to make a property mandatory.
   e) Click Add to specify more properties.

   You can specify as many properties as needed for the connector.
10. Click Create.

    The newly added connector is listed under the **Connectors**.

Kafka connectors

When using the Kafka connector, you can choose between using an internal or external Kafka service. Based on the connector type you choose, there are mandatory fields where you must provide the correct information.

You can choose from the following Kafka connectors when creating a table in Streaming SQL Console:

Template: local-kafka

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

Type: source/sink

The following fields are mandatory to use the connector:

- scan.startup.mode: Startup mode for the Kafka consumer. group-offsets is the default value. You can choose from earliest-offset, latest-offset, timestamp and specific-offsets as startup mode.
- topic: The topic from which data is read as a source, or the topic to which data is written to. No default value is specified. You can also add a topic list in case of sources. In this case, you need to separate the topics by semicolon. You can only specify the topic-pattern or topic for the sources.
- format: The format used to deserialize and serialize the value part of Kafka messages. No default value is specified. You can use either the format or the value.format option.

Template: kafka

Using an external Kafka service as a connector. To connect to the external Kafka service, you need to speficy the Kafka brokers that are used in your deployment.

Type: source/sink

The following fields are mandatory to use the connector:

- properties.bootstrap.servers: Specifying a list of Kafka brokers that are separated by comma. No default value is specified.
- topic: The topic from which data is read as a source, or the topic to which data is written to. No default value is specified. You can also add a topic list in case of sources. In this case, you need to separate the topics by semicolon. You can only specify the topic-pattern or topic for the sources.
- format: The format used to deserialize and serialize the value part of Kafka messages. No default value is specified. You can use either the format or the value.format option.

Template: 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. For more information about the upsert Kafka connector, see the Apache Flink documentation.

Type: source/sink

- properties.bootstrap.servers: Specifying a list of Kafka brokers that are separated by comma. No default value is specified.
- topic: The topic from which data is read as a soruce, or the topic to which data is written to. No default value is specified. You can also add a topic list in case of sources. In this case, you need to separate the topics by semicolon. You can only specify the topic-pattern or topic for the sources.
- key.format: The format used to deserialize and serialize the key part of Kafka messages. No default value is specified. Compared to the regular Kafka connector, the key fields are specified by the PRIMARY KEY syntax.
- value.format: The format used to deserialize and serialize the value part of Kafka messages. No default value is specified. You can use either the format or the value.format option.

Configuring deserialization policy in DDL

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

1. Choose one of the Kafka template types from Templates.
2. Select any type of data format.

   The predefined CREATE TABLE statement is imported to the SQL Editor.
3. Fill out the Kafka template based on your requirements.
4. Search for the deserialization.failure.policy.
5. 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. In this case, you must provide the name of the created DLQ topic for the deserialization.failure.dlq.topic property.
6. Click Execute.

For more information about how to configure the error handling using the Kafka wizard, see the Deserialization tab section.

CDC connectors

You can use the Debezium Change Data Capture (CDC) connector to stream changes in real-time from MySQL, PostgreSQL, Oracle, Db2, SQL Server and feed data to Kafka, JDBC, the Webhook sink or Materialized Views using Cloudera SQL Stream Builder.

Concept of Change Data Capture

Change Data Capture (CDC) is a process to capture changes in a source system, and update the data within a downstream system or application with the changes.

The Debezium implementation offers CDC with database connectors from which real-time events are updated using Kafka and Kafka Connect. Debezium captures every row-level change in each database table of an event stream. Applications read these streams to see the change events in the same order as they occurred. The change events are routed to a Kafka topic from which Kafka Connect feeds the records to other systems and databases.

For more information about Debezium, see the official Debezium documentation.

CDC in Cloudera Streaming Analytics does not require Kafka or Kafka Connect as Debezium is implemented as a library within the Flink runtime. This means that the captured changes are propagated downstream to any connector that Flink supports. Cloudera Streaming Analytics allows queries to be issued at change data capture time, which means filtering, grouping, joining, and so on, can be performed on the change stream as it comes from the source database.

For more information about the Flink implementation of Debezium, see the official Apache Flink documentation.

From the supported set of Debezium connectors, MySQL, PostgreSQL, Oracle, Db2, and SQL Server are supported in Cloudera Streaming Analytics.

Note: You need to configure the databases, users and permissions for the supported connectors before you are able to use them in Cloudera SQL Stream Builder. For more information about setting up the databases, see the official Debezium documentation.

Webhook connector

The webhook connector can be used in SQL or through Python UDFs.

Example using the webhook connector in SQL

1. Create a webhook table. Refer to Creating Webhook tables for more information.
2. Write into the webhook table the query's INSERT statement. Example:

```
insert into [***WEBHOOK TABLE***]
select transactionId, ts, accountId, amount * 3
from [***SOURCE TABLE***]
where amount % 2 != 0
```

Example using the webhook connector with Python UDFs

1. Create your UDF using the Cloudera SQL Stream Builder UI. Example:

```
from pyflink.table.udf import udf
from pyflink.table import DataTypes
from pyflink.table import Row
@udf(result_type=DataTypes.ROW([
    DataTypes.FIELD("transactionId", DataTypes.STRING()),
    DataTypes.FIELD("ts", DataTypes.STRING()),
    DataTypes.FIELD("accountId", DataTypes.STRING()),
    DataTypes.FIELD("amount", DataTypes.STRING())
]))
def udf_function(transactionId, ts, accountId, amount):
    if amount % 2 != 0:
        processed_amount = amount * 3
        return Row(str(transactionId), str(ts), str(accountId), str(pro
cessed_amount))
```

2. Create a webhook table. Refer to Creating Webhook tables for more information.
3. Write into the webhook table using the UDF in the query's WHERE statement. Example:

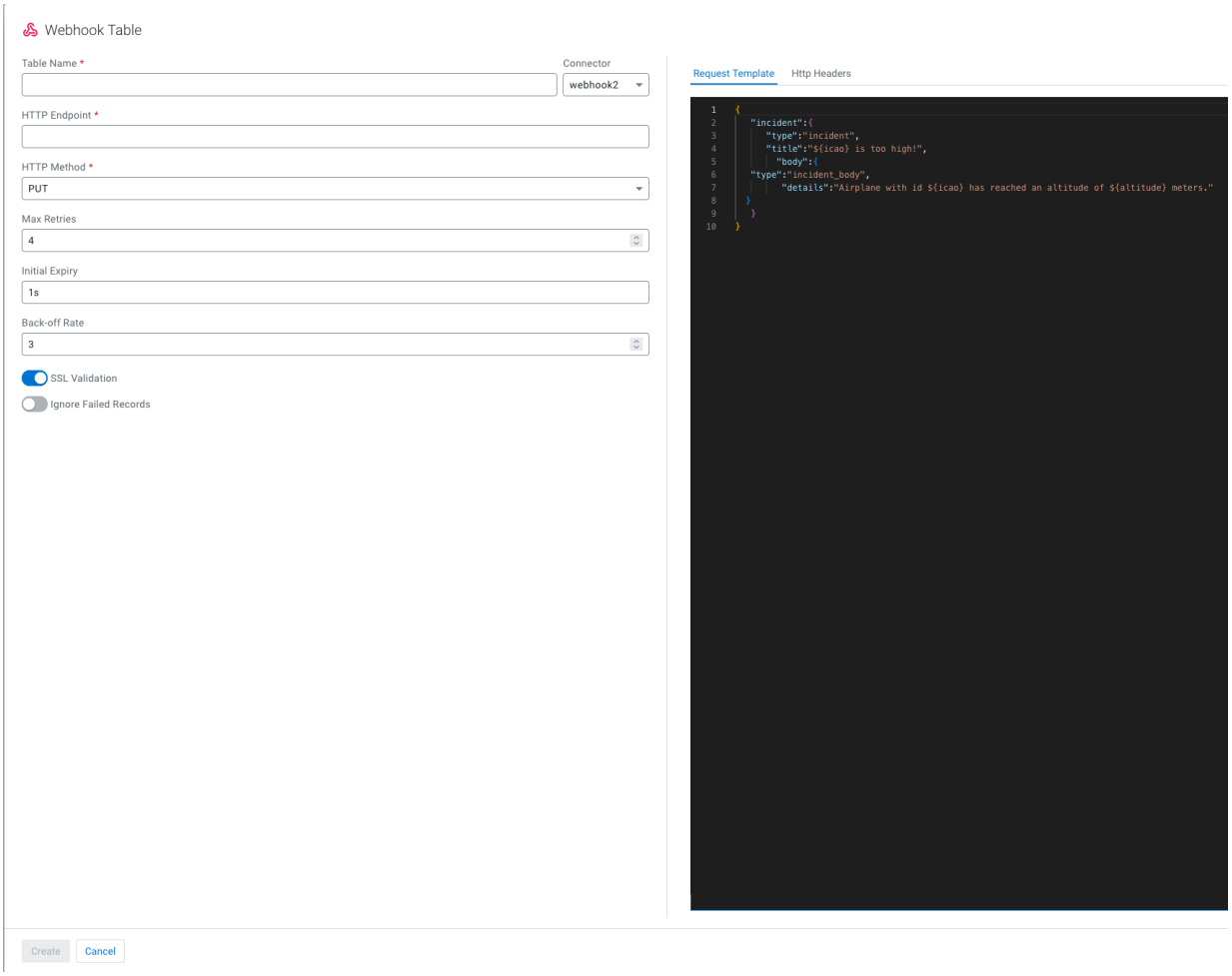
```
INSERT INTO [***TABLE NAME***]
SELECT transactionId, ts, accountId, amount
FROM (
    SELECT transaction(transactionId, ts, accountId, amount)
    FROM [***TABLE NAME***]"
) AS udf_results
WHERE transactionId IS NOT NULL
  AND ts IS NOT NULL
"  AND accountId IS NOT NULL "
"  AND amount IS NOT NULL",
destinationName, sourceName);
```

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Virtual Tables**.
4. Select New Webhook Table .
   The **Webhook Table** window appears.



Important: In the Connector menu, select webhook2. The v1 connector is deprecated and will be removed in an upcoming release of Cloudera Streaming Analytics.

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 Create.

Results

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

JDBC connector

When using the JDBC connector, you can choose between using a PostgreSQL, MySQL, Oracle, Hive, Db2 or SQL Server databases. Based on the connector type you choose, there are mandatory fields where you must provide the correct information.

Template: jdbc

Using either PostgreSQL, MySQL, Oracle, Hive, Db2 or SQL Server as databases. When you use the JDBC connector, you must specify the JDBC database which are going to be used for the connection.

Type: source/sink

The following fields are mandatory to use the connector:

- url: The URL of the JDBC database. No default value is specified.
- table-name: The name of the JDBC table in the database that you need to connect to. No default value is specified.

Filesystem connector

When using the Filesystem connector, you can choose between HDFS, S3 and so on storage systems. Based on the connector type you choose, there are mandatory fields where you must provide the correct information.

Template: filesystem

Using either HDFS, S3 or any type of storage system. When you use the Filesystem connector, you must specify the path to the file system which are going to be used for the connection.

Type: source/sink

The following fields are mandatory to use the connector:

- path: Path to the root directory of the table data. No default value is specified.
- format: The format used for the file system. No default value is specified.

Datagen connector

The Datagen connector can be used as a source to generate random data. The Datagen connector works out of the box, no mandatory field is required to use the connector. The Datagen connector is useful when you want to experiment and try out Cloudera SQL Stream Builder or to test your SQL queries using random data.

Template: datagen

You do not need to provide any type of information when using the Datagen connector and template.

Type: source

Faker connector

The Faker connector can be used as a source to generate random data. To use the Faker connector, you need to specify the usage The Datagen connector is useful when you want to experiment and try out Cloudera SQL Stream Builder or to test your SQL queries using random data.

Template: faker

You do not need to provide any type of information when using the Datagen connector and template.

Type: source

The following fields are mandatory to use the connector:

- fields.#.expression: The Java Faker expression to generate the values for a specific field. For more information about the list and use of the Faker expressions, see the flink-faker documentation.

Blackhole connector

The Blackhole connector can be used as a sink where you can write any type of data into. The Blackhole connector works out of the box, no mandatory field is required to use the connector. The Blackhole connector is useful when you want to experiment and try out Cloudera SQL Stream Builder or to test your SQL queries using random data.

Template: blackhole

You do not need to provide any type of information when using the Blackhole connector and template.

Type: sink

Data formats

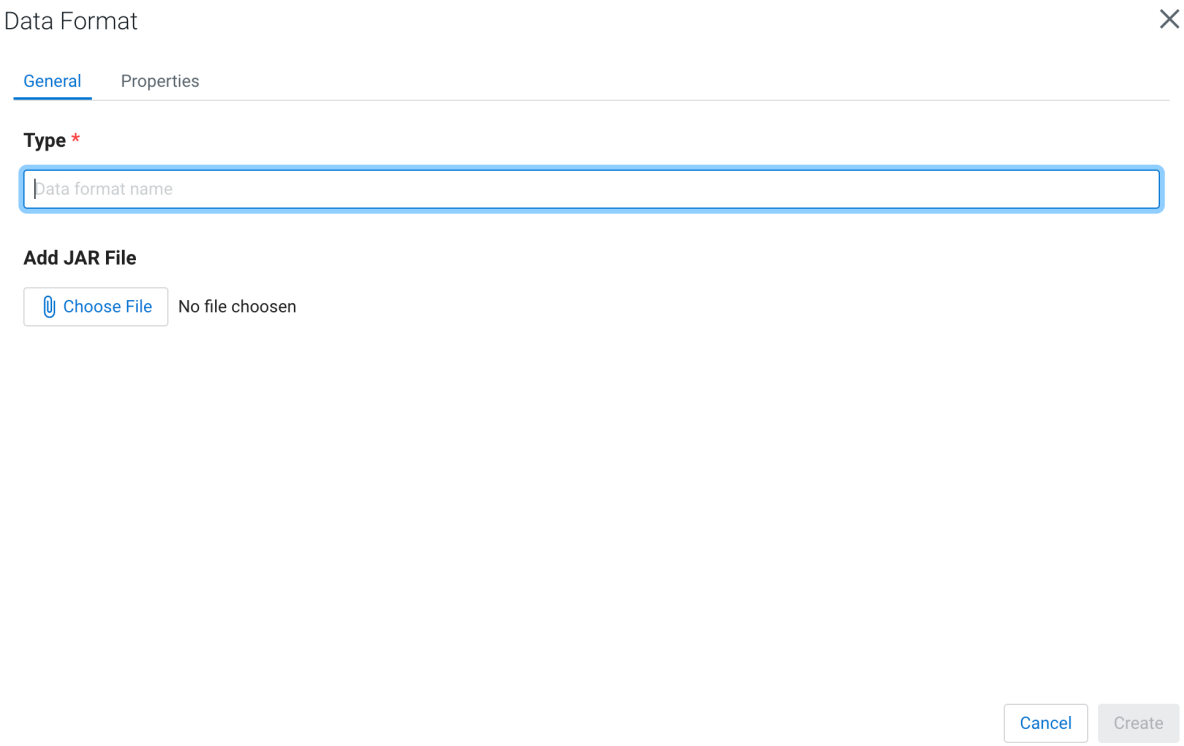
There is a set of table formats that can be used with the supported connectors of Cloudera SQL Stream Builder. The data format is a storage format that defines how to map binary data to the columns of a table. You can use the data formats based on which connector they are supported for.

Adding data formats

When adding new data formats to Cloudera SQL Stream Builder, you need to specify the property list and must upload the data format JAR file to the Console.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Open **External Resources** from the **Explorer** view.
4. Click next to **Data Formats**.
5. Click New Connector.



6. Provide a name for the data format as Type.
7. Upload the data format JAR file.
8. Click Properties to add properties to the data format.
   a) Add a name to the property.
   b) Add a default value to the property.
   c) Add a description to the property.
   d) Click Required to make a property mandatory.
   e) Click Add to specify more properties.

   You can specify as many properties as needed for the data format.
9. Click Create.

   The newly added data format is listed under the **Data Formats**.

Concept of tables in Cloudera SQL Stream Builder

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

Creating Kafka tables using wizard

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

About this task

You can query your streaming data using Kafka tables in Cloudera SQL Stream Builder. You have the option to use the Kafka service in your environment, or connect to an external Kafka service. When creating Kafka tables you can use the Add Kafka wizard, the predefined templates or you can directly add a custom CREATE TABLE statement with the required properties in the SQL window.

You can also create Kafka tables using one of the Kafka templates. For more information, see the Kafka connectors and Using connectors with templates sections.

Before you begin

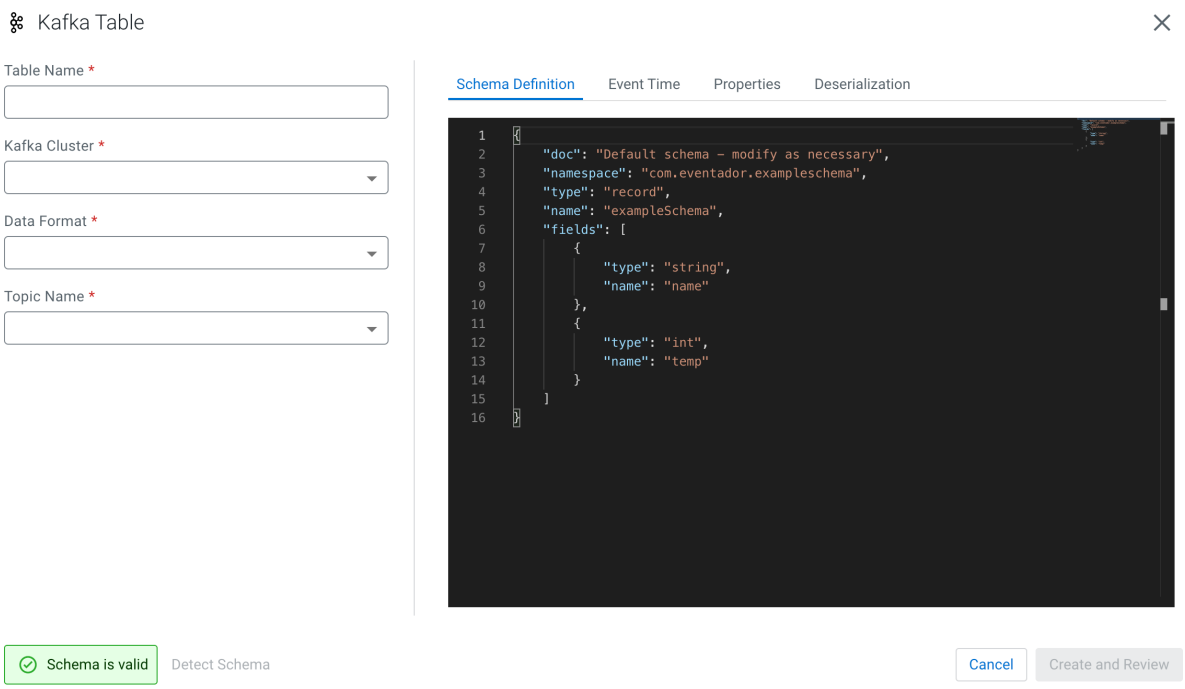
- Ensure that you have registered Kafka as a Data Source.
- Ensure that you have created topics in Kafka.

Important: When creating the topic for the Kafka sink, ensure that you do not use log compaction as it can cause the SQL job to fail.

- Ensure that there is generated data in the Kafka topic.
- Ensure 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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Virtual Tables**.
4. Click New Kafka Table .
   The **Kafka Table** window appears.



5. Provide a Table Name.

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

6. Select a registered Kafka provider as Kafka cluster.
7. Select the Data format.

- You can select JSON as data format.
- You can select AVRO as data format.

8. 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.

9. Add a customized schema to Schema Definition or click Detect Schema to read a sample of the JSON messages, and automatically infer the schema.

   Note: The Detect Schema functionality is only available for JSON data. If there are no messages in the topic, then no schema will be inferred. 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.

10. Customize your Kafka Table with the following options:
    a) Configure the Event Time if you do not want to use the default Kafka Timestamps.
    b) Configure any Kafka properties required on the Properties tab.
    c) 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.

11. 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.

Configuring Kafka tables

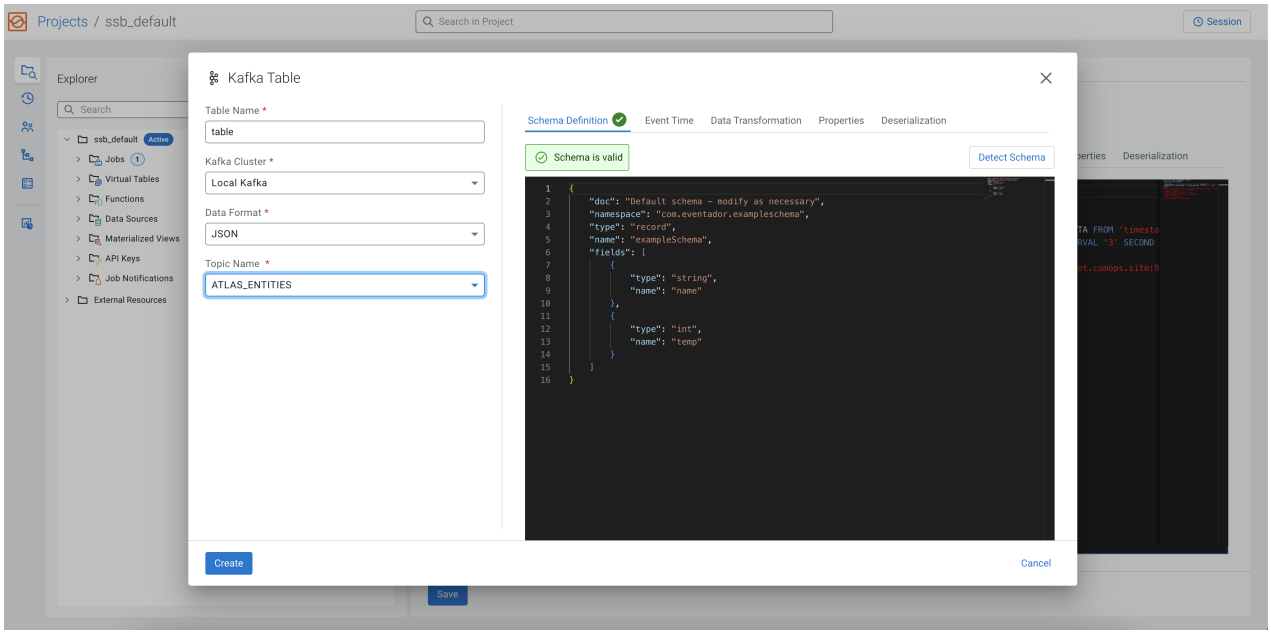
The user-defined Kafka table can be configured based on the schema, event time, and other Kafka-specific properties by using either the Kafka wizard or DDL.

Schema Definition 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.

Note: When using Schema Registry with Kafka, only the AVRO data format is supported.



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.

Tip: You can refresh the detected schema using Detect schema button. The new Avro schema is displayed after the new detection was successful.

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:

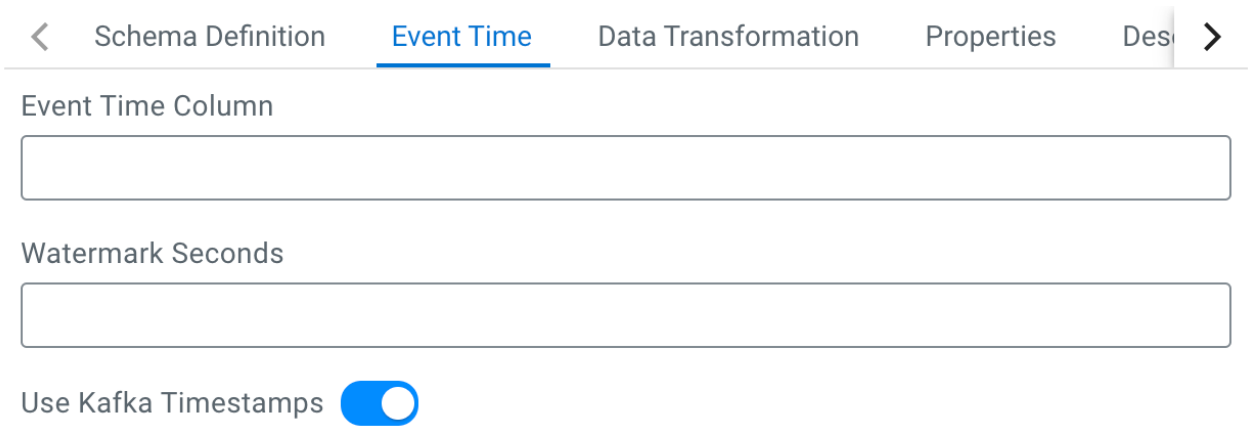
- Input Timestamp Column: name of the timestamp column in the Kafka topic from where the watermarks are mapped to the Event Time Column of the Kafka table
- Event Time Column: default or custom name of the resulting timestamp column where the watermarks are going to be mapped in the created Kafka table
- 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.



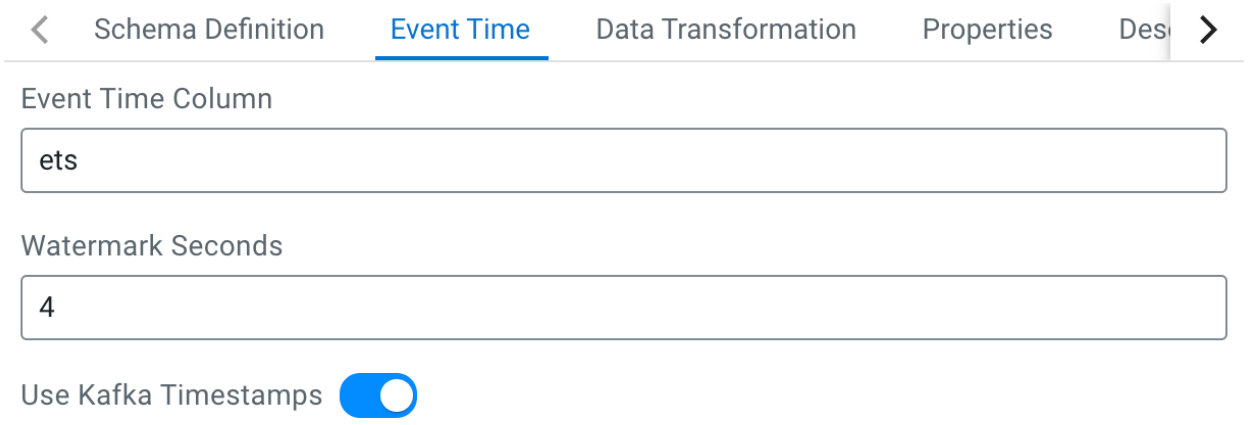
After saving your changes, you can view the created DDL syntax for the table next to the SQL Editor on the **Console** page.

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.



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

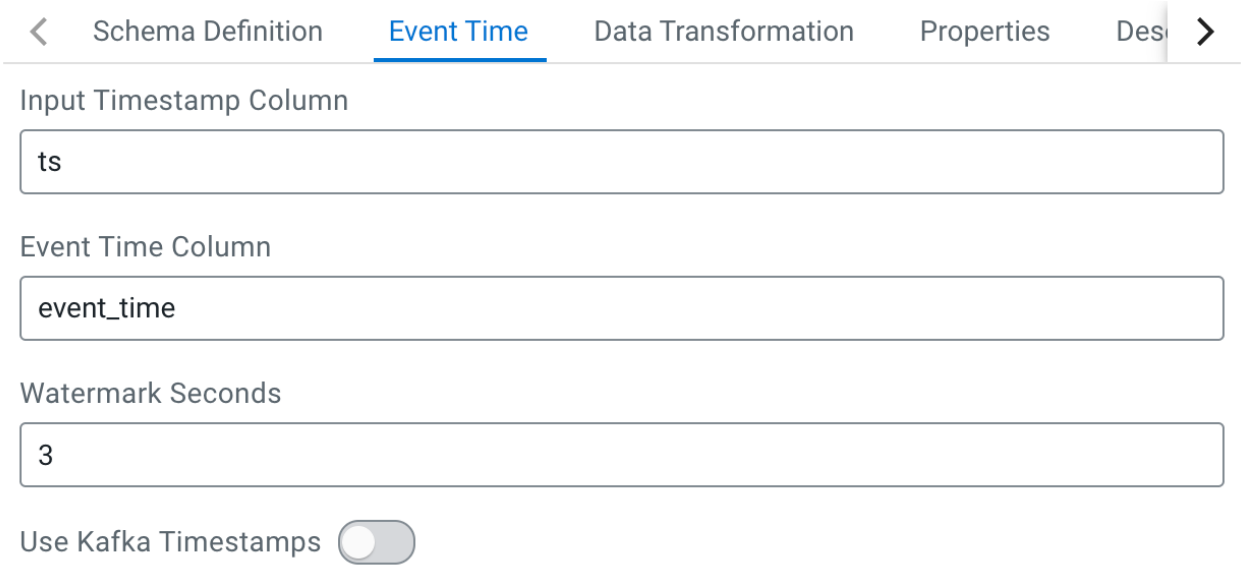
When you want to manually configure the watermark strategy of the Kafka table, you can provide the timestamp column name from the Kafka source, and add a custom column name for the resulting Kafka table. Make sure that

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 *'TS'*, and want to add a new timestamp column in your Kafka table as *'EVENT_TIME'*. You provide the original timestamp column name in the Input Timestamp Column as *'TS'*, and add the custom *'EVENT_TIME'* name to the Event Time Column.



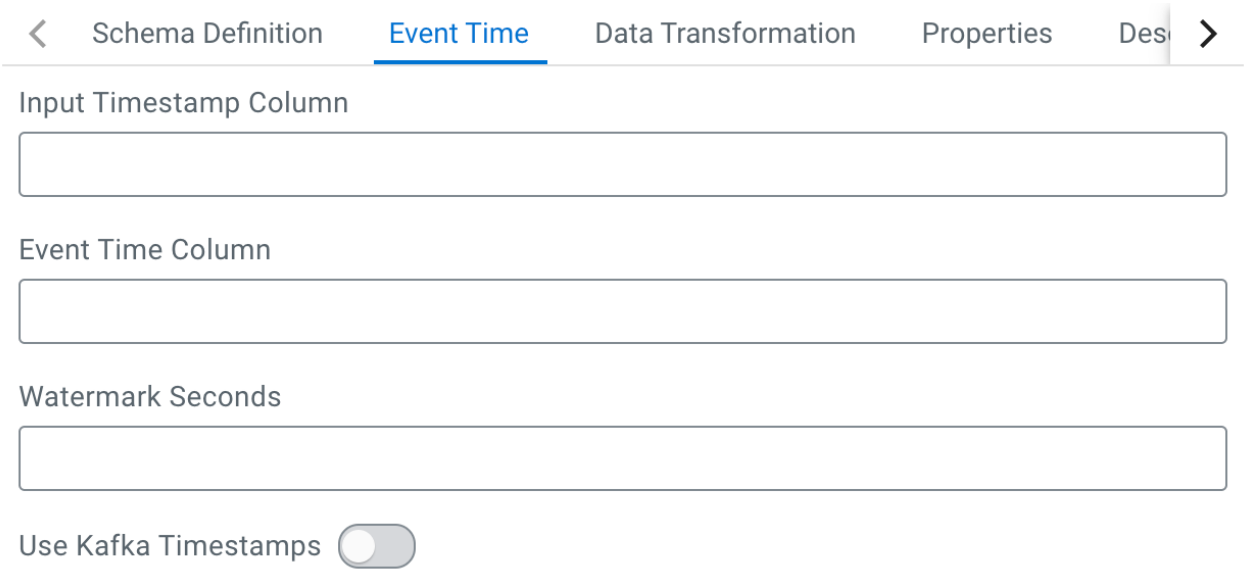
This results in that the watermarks from the *'TS'* column is going to be mapped to the *'EVENT_TIME'* column of the created Kafka table. As *'EVENT_TIME'* will become the timestamp column name in the Kafka table, you must use the custom name (in this example the *'EVENT_TIME'*) when querying the Kafka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



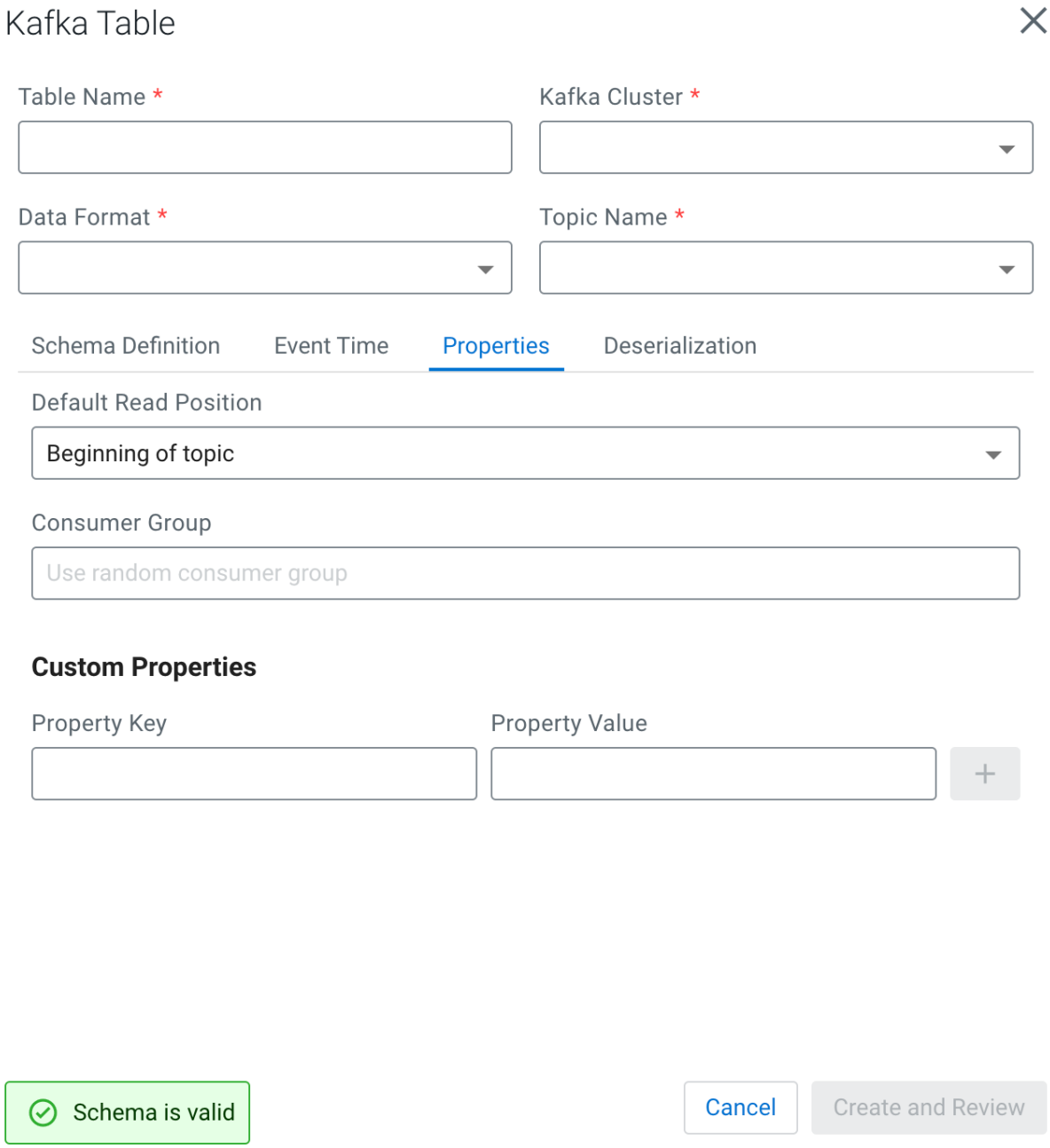
Note: Flink validates the input and output schemas of the data. You can only insert data into a Kafka topic, if the input and output data matches based on the schema defined for the topic. When customizing the timestamp column, make sure that the output data has the same schema.

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

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.



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

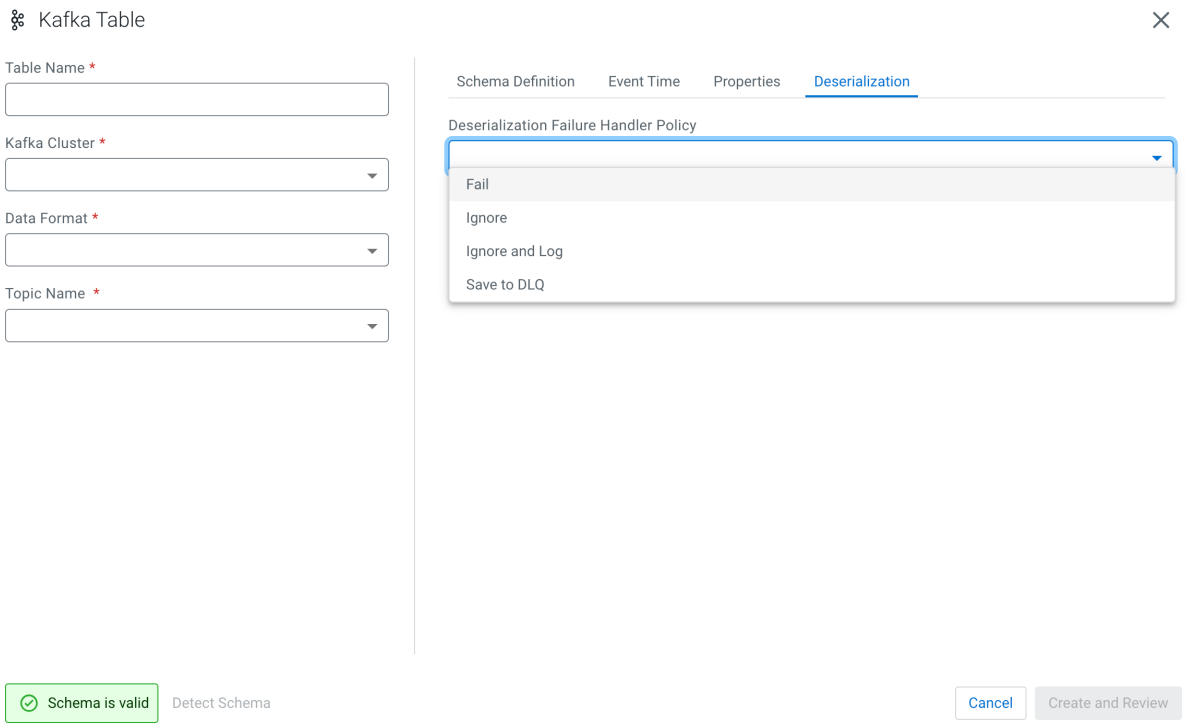
Using the Kafka wizard

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 Cloudera 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.

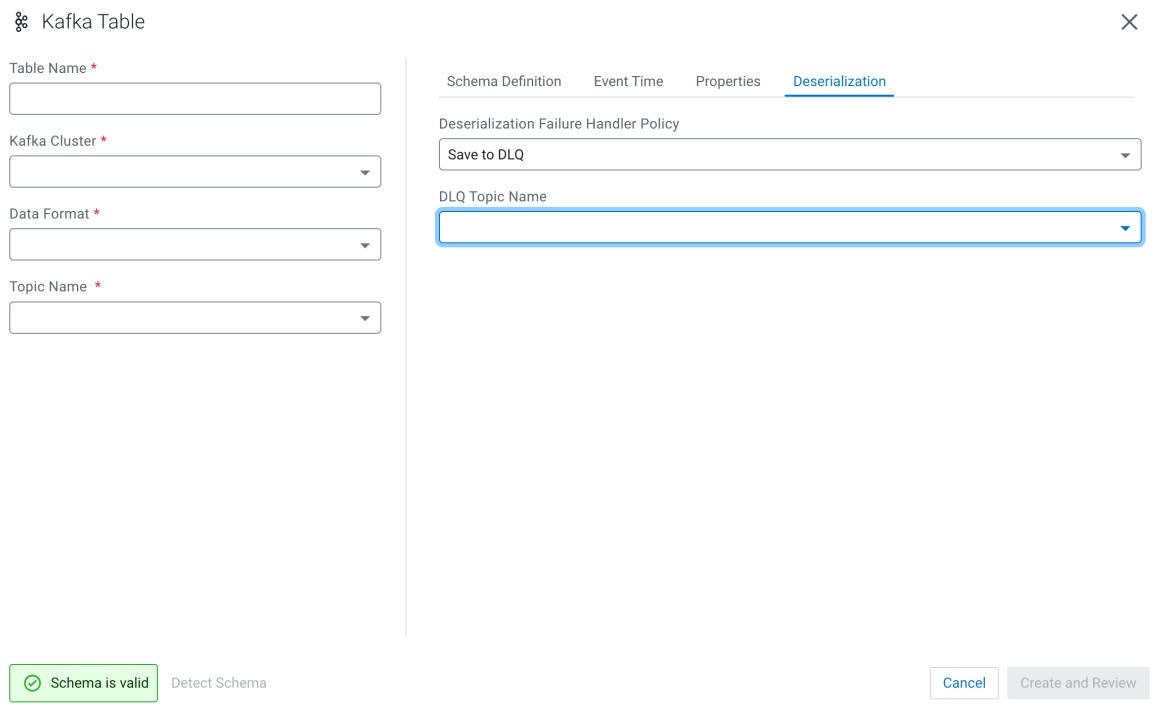
   You are redirected to the **Console** page.
3. Select Add tables > Apache Kafka .

   The **Add Kafka table** window appears
4. Select Deserialization tab.



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. In this case, you must select the dedicated DLQ topic from DLQ Topic Name.



- Click Create and Review.

For more information about how to configure the error handling using DDL, see the Kafka connectors section.

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 Cloudera 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 Cloudera SQL Stream Builder, but the proper configuration and design of the source Kafka topic is critical. Cloudera SQL Stream Builder can read a maximum of one thread per Kafka partition. You can achieve the highest performance configuration when setting the Cloudera SQL Stream Builder threads equal to or higher than the number of Kafka partitions.

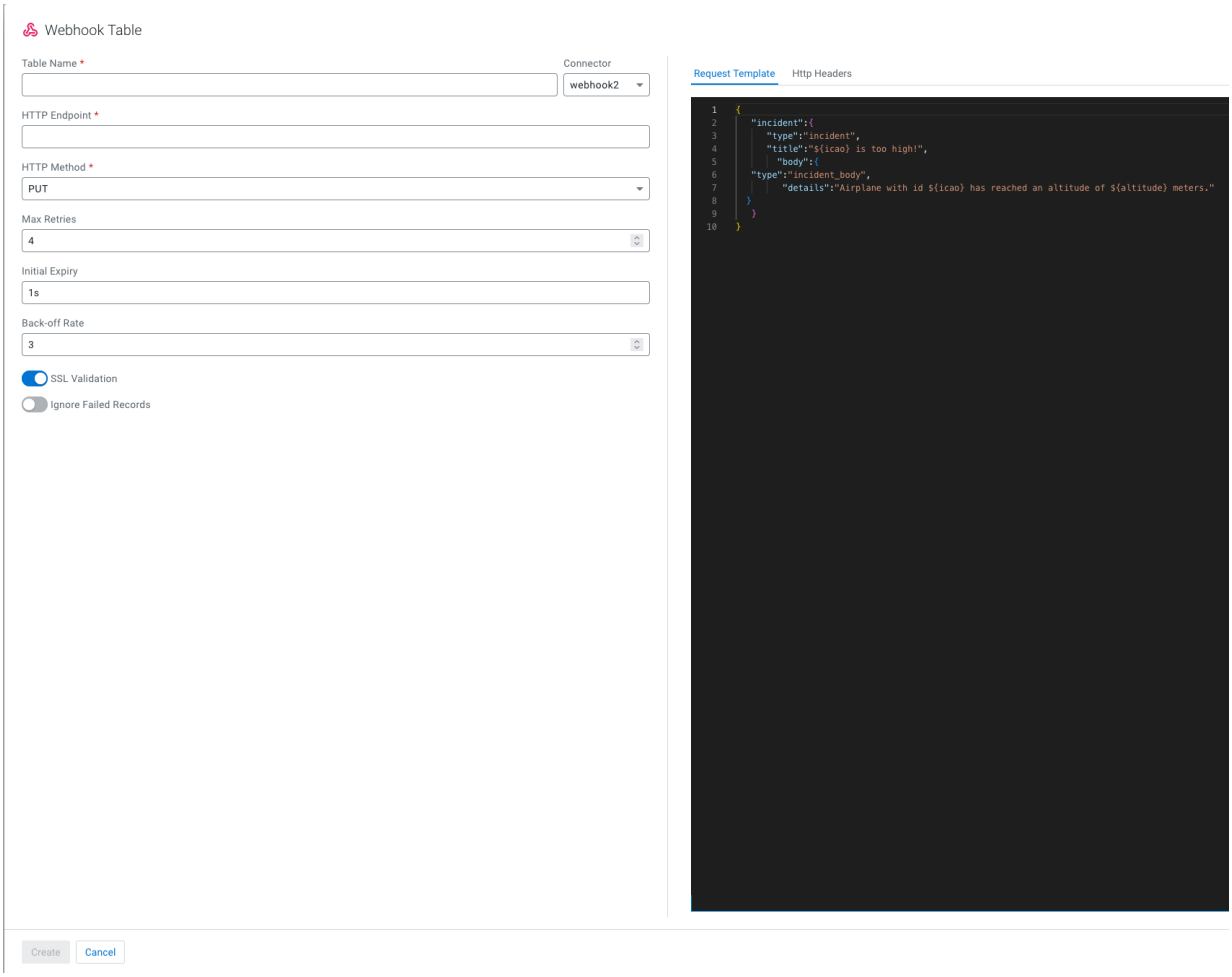
If the number of partitions is less than the number of Cloudera SQL Stream Builder threads, then Cloudera SQL Stream Builder has idle threads and messages show up in the logs indicating as such. For more information about Kafka partitioning, see the Kafka partitions documentation.

Creating 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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Virtual Tables**.
4. Select New Webhook Table .
   The **Webhook Table** window appears.



Important: In the Connector menu, select webhook2. The v1 connector is deprecated and will be removed in an upcoming release of Cloudera Streaming Analytics.

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 Create.

Results

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

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 Cloudera 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 Iceberg tables

Apache Iceberg is an open, high-performance table format for organizing datasets that can contain petabytes of data. Iceberg can be used to add tables to computing engines, such as Apache Hive and Apache Flink, from which the data can be queried using SQL.

As Iceberg is integrated as a connector to Flink, you can use the table format the same way for Cloudera SQL Stream Builder. Both the V1 and V2 version specifics are supported by the Flink connector.For more information, about the version changes, see the Apache Iceberg documentation.

Table 2: SQL feature support for the Iceberg and Cloudera SQL Stream Builder integration

Feature	SSB
Create catalog	Supported
Create database	Supported
Create table	Supported
Alter table	Supported **Note:** Only the altering table properties is supported, which does not inclide column and partition changes.
Drop table	Supported

Feature	SSB
Select	Supported
Insert into	Supported
Insert overwrite	Supported
Metadata tables	Supported
Rewrite files action	Supported
Upsert	Technical preview[1]
Equality delete	Technical preview[2]

Enabling checkpointing for streaming writes

When writing data to an Iceberg table in a streaming fashion using INSERT INTO or INSERT OVERWRITE, you must enable checkpointing in your Flink job. The Flink Iceberg sink commits the data only when checkpoints are completed. Without checkpointing, your data will not be written to the Iceberg table.

You can enable checkpointing in the Streaming SQL Console by setting the execution.checkpointing.interval property at the beginning of your script:

```
-- Set the checkpointing interval to 10 seconds
SET 'execution.checkpointing.interval' = '10s';
-- Insert data from a source table into your Iceberg table
INSERT INTO your_iceberg_table
SELECT * FROM your_source_table;
```

If you are using the programmatic Table API with Python, you can set the property on the execution environment:

```
env.get_config().set("execution.checkpointing.interval", "10 s")
```

Using Hive for Iceberg integration

When using the Hive service located on your cluster, you can add it as a catalog on Streaming SQL Console. Before creating the Iceberg table, ensure that you have added Hive as a catalog using the steps described in documentation.

After setting up Hive for Cloudera SQL Stream Builder, you can define Iceberg as a connector in the CREATE TABLE statement as the example shows below:

```
CREATE TABLE ‘ssb’.’ssb_default’.’iceberg_hive’ (
‘column_int’ INT,
‘column_str’ STRING,
) WITH (
‘connector’ = ‘iceberg’,
‘catalog-database’ = ‘test_db’,
‘catalog-type’ = ‘hive’,
‘catalog-name’ = ‘iceberg_hive_catalog’,
‘catalog-table’ = ‘iceberg_hive_table’,
‘ssb-hive-catalog’ = ‘ssb_hive_catalog’
‘engine.hive.enabled’ = ‘true’
)
```

The following properties are mandatory when using the Iceberg connector:

[1] This feature is in Technical Preview and is not ready for production deployments. Cloudera recommends trying this feature in test or development environments and encourages you to provide feedback on your experiences.

[2] This feature is in Technical Preview and is not ready for production deployments. Cloudera recommends trying this feature in test or development environments and encourages you to provide feedback on your experiences.

Property	Example	Description
catalog-database	test_db	The Iceberg database name in the backend catalog, uses the current Flink database name by default. It will be created automatically if it does not exist when writing records into the Flink table
catalog-type	hive	Type of the catalog
catalog-name	iceberg_hive_catalog	User-specified catalog name. It is required as the connector does not have any default value.
catalog-table	iceberg_hive_table	Name of the Iceberg table in the backend catalog.
ssb-hive-catalog	ssb_hive_catalog	The name of the Hive catalog that you have registered in SSB. The configuration can be used when creating Iceberg catalogs as well.

SQL jobs

Creating and naming SQL jobs

You need to create or select an already existing SQL job on the Streaming SQL Console to be able to submit SQL queries.

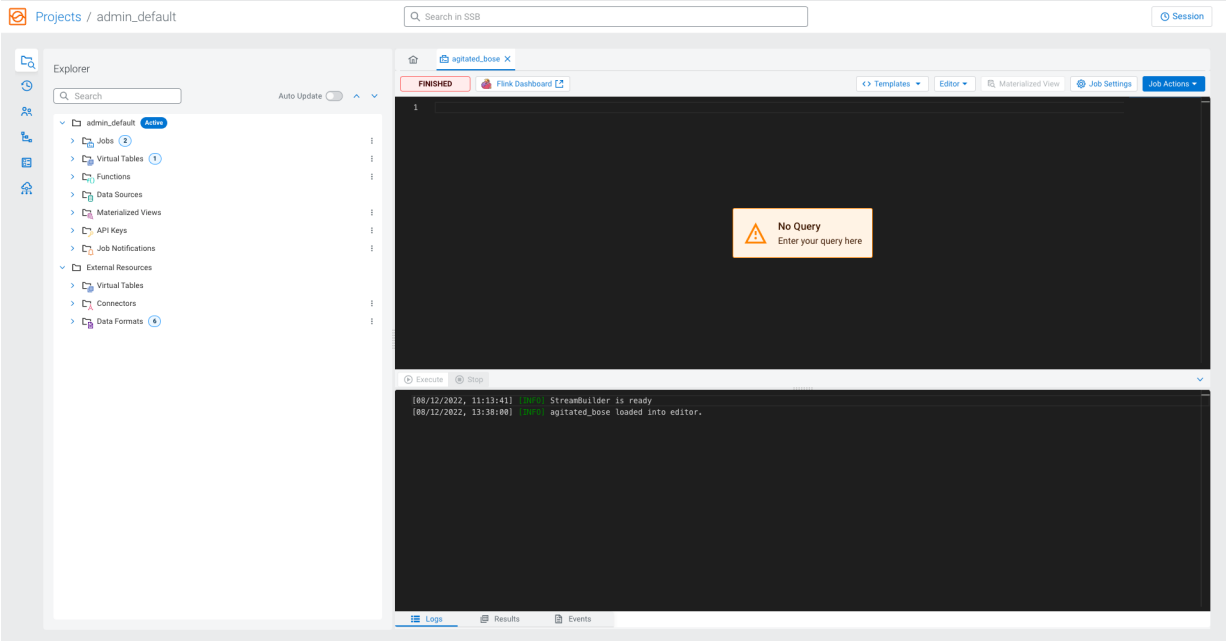
About this task

You can manage your SQL jobs and resources using the **Explorer** view or the **Home** tab of your project. Every job and resource related action is the same for both views of Streaming SQL Console.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3.
   Click New Job on the **Explorer** view.
4. Provide a Name to your SQL job.
   You have the following options to name your SQL job:
   - Use the predefined name in the Job Name field when opening the Streaming SQL Console
   - Use the Generate Job Name button to generate a new random name for your job
   - Manually provide the name of the job to the Job Name field
5. Click Create.

You are redirected to the SQL Editor where you can submit your SQL queries, create and manage Materialized Views and configure your SQL jobs.



What to do next

After a new job is created, you can save the job and its settings using the Job Actions Save button. When you make a change to the SQL query, job settings or create a Materialized View, you have the option to save the changes with the job. You can also revert your changes to the latest saved version of the job using the Revert Changes button. You can also delete the job from the SQL Editor using the Delete button.

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 Source if you use the Kafka service on your cluster.
- Make sure that you have added Kudu, Hive or Schema Registry as a catalog if you use them for your SQL job.

Procedure

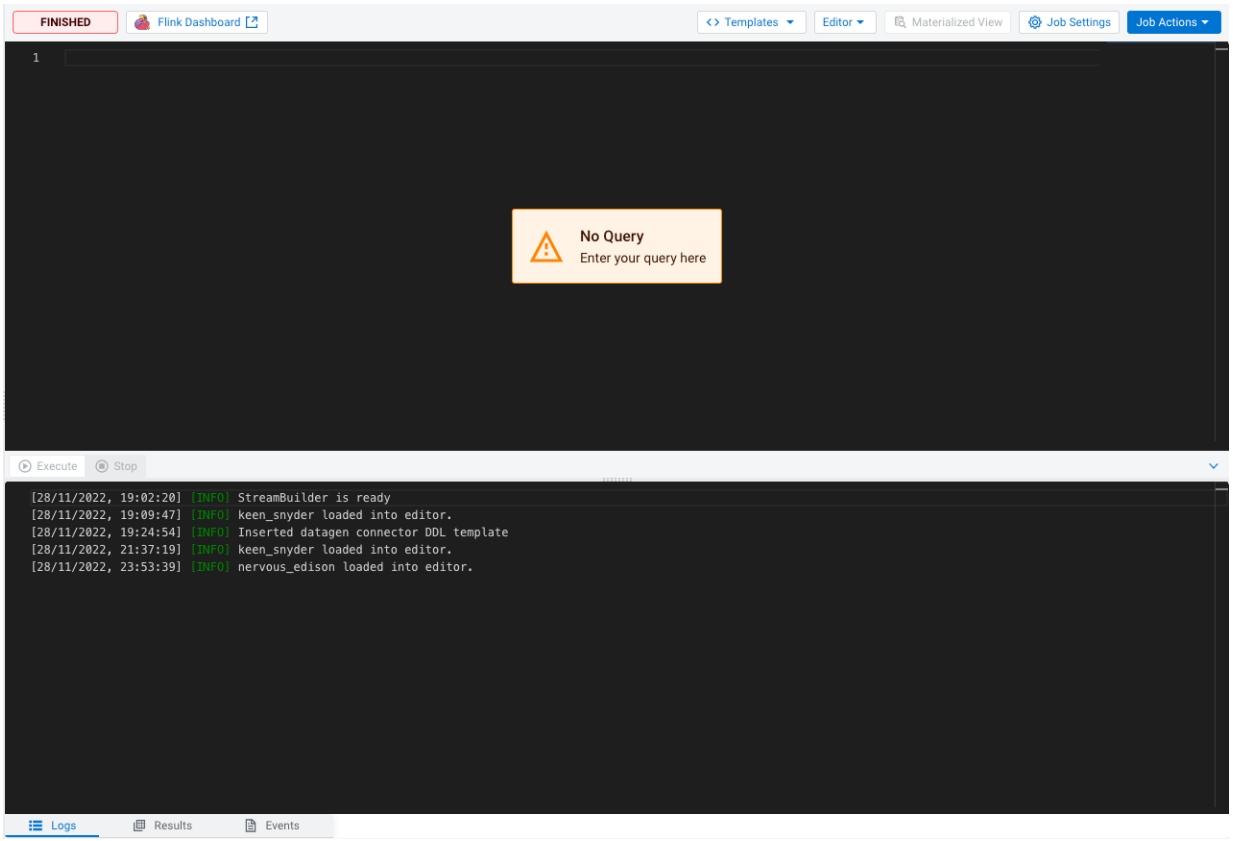
1. Navigate to the Streaming SQL Console.
   a) Go to your cluster in Cloudera Manager.
   b) Select SQL Stream Builder from the list of services.
   c) Click SQLStreamBuilder Console .

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

2. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.

   You are redirected to the **Explorer** view of the project.
3. Click next to **Jobs** from the **Explorer**.
4. Select New Job .

   The SQL Editor for the created job opens in a tab.
5. Provide a name for the SQL job.
   a) Optionally, you can click Generate Random Name to generate a name for the SQL job.
6. Click Create.
7. Create a table using one of the following options:
   - Using the Add Kafka or Webhook table wizard.
   - Using the Templates.
   - Add your custom CREATE TABLE statement to the SQL window.
8. Add a SQL query to the SQL Editor.

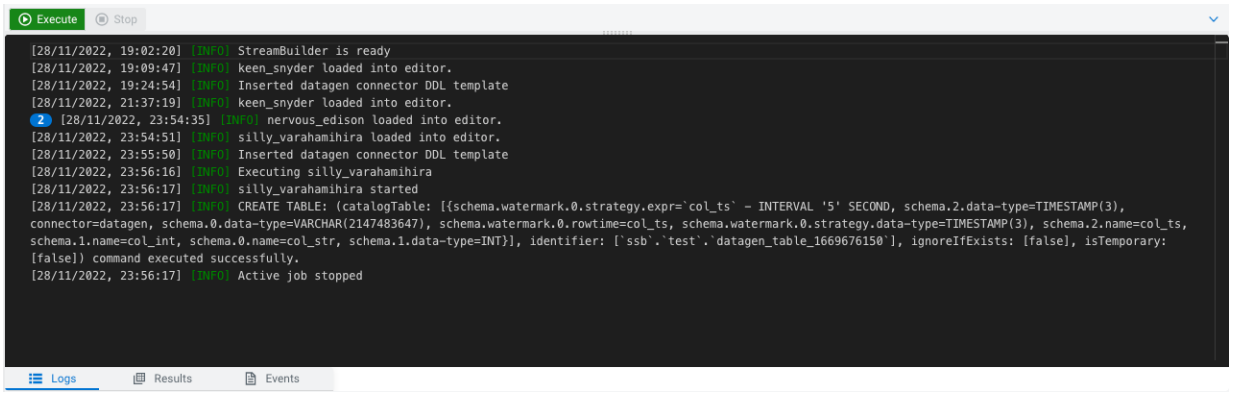


Note: You cannot start a job without adding a SQL statement in the SQL editor window. In case, there are no SQL statements provided and you click Execute , the following error message is displayed: You must provide a SQL query.

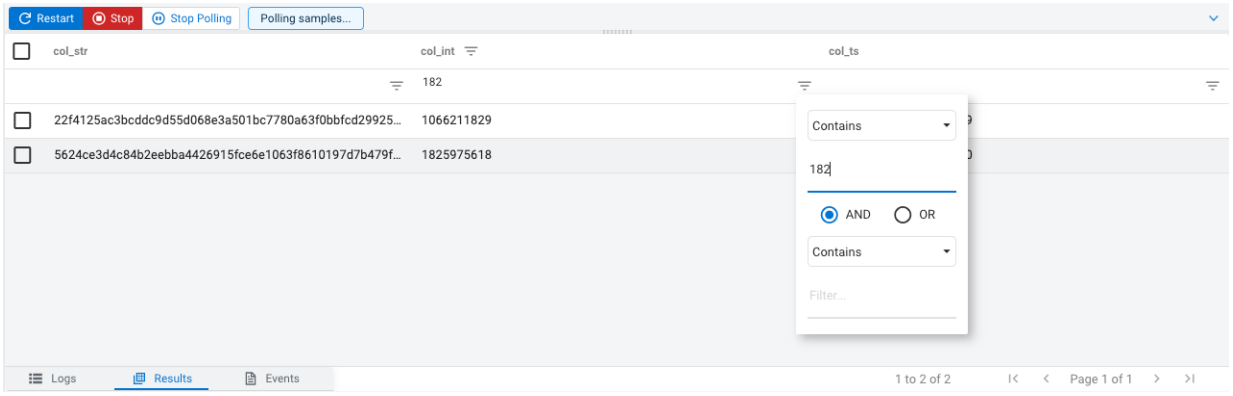
When starting a job, the number of slots consumed on the specified cluster is equal to the parallelism setting. The default is one slot. To change the parallelism setting and more job related configurations, click Job Settings.

9. Click Execute.

   The Logs window updates the status of SSB.



10. Click Results to check the sampled data.

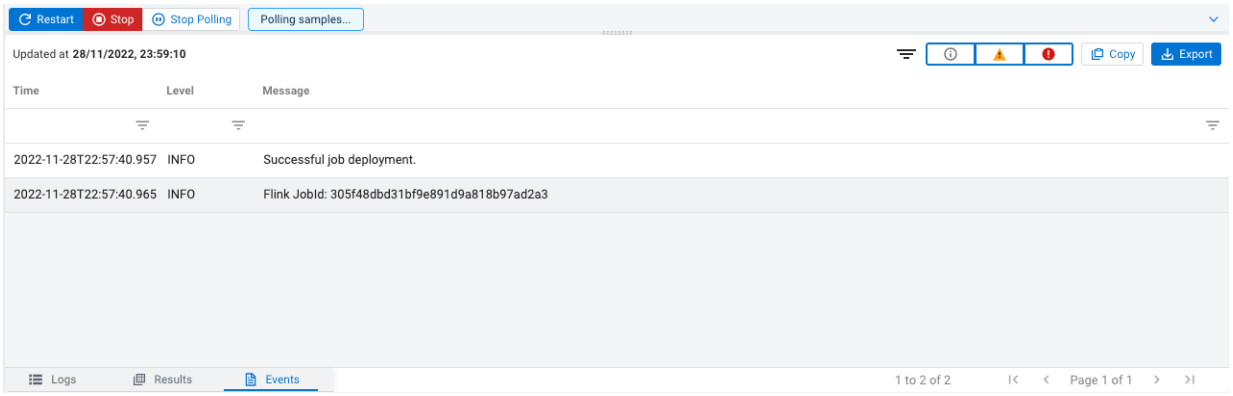


You can pause or restart the sampling of a SQL job with the Pause and Refresh buttons. The status of the polling can also be seen next to the Pause button. The Results tab also allows you to change the order of the samples, and to filter the samples using different types of conditions that you can also combine. For changing the order, click on the name of the column, the direction of the ordering is indicated next to the column name by arrows.



For filtering, you can either click the button next to the column name or under the row of the column name. The filtering value can be changed by typing the values directly to the filtering row, the conditions need to be changed using the filtering arrow.

11. Click Events to check the job related information.



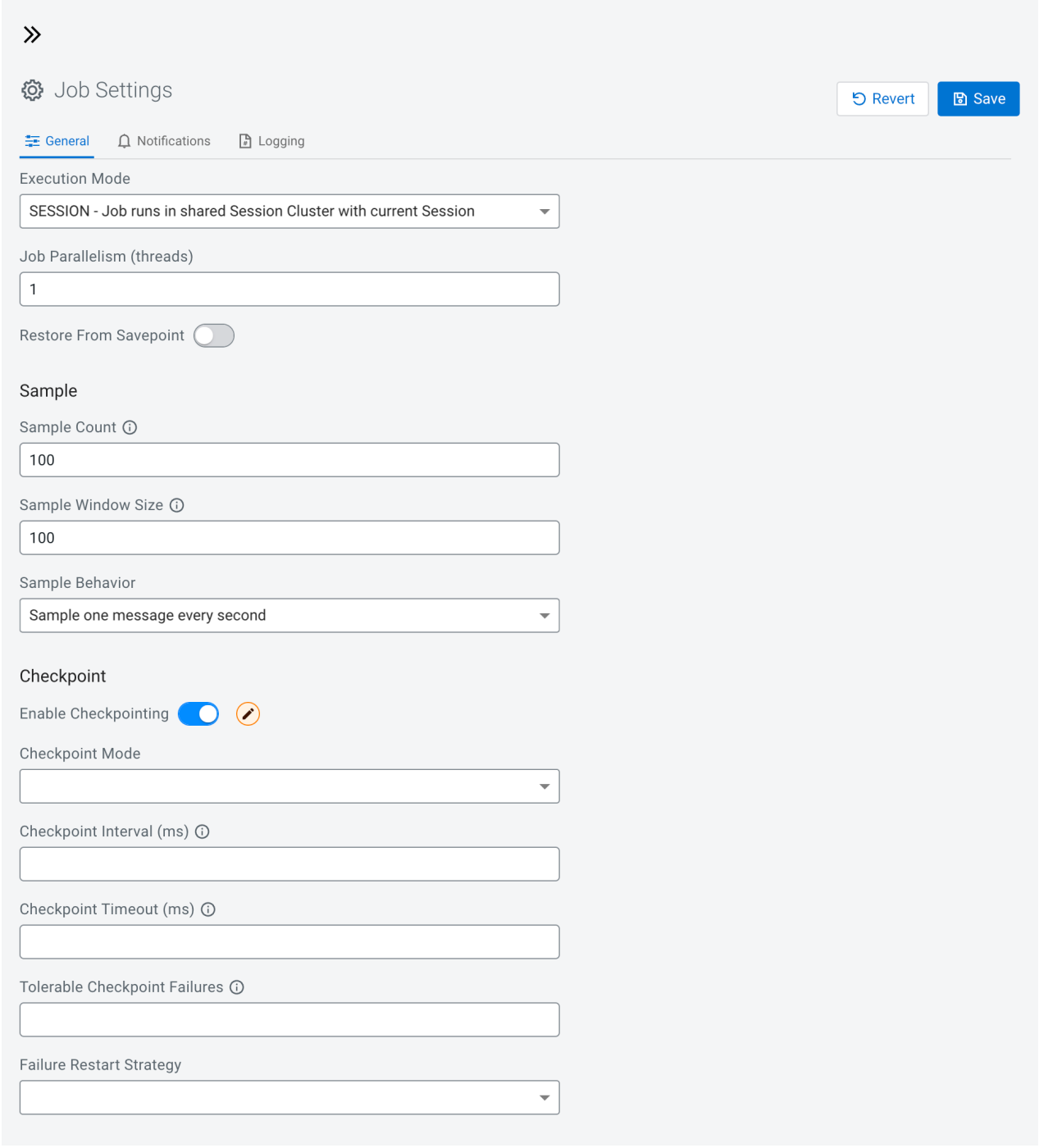
Results

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

Configuring SQL job settings

If you need to further customize your SQL Stream job, you can add more advanced features to configure the job restarting method and time, threads for parallelism, sample behavior, exactly once processing and restoring from savepoint.

Before running a SQL query, you can configure advanced features by clicking on the Job Settings button at the SQL Editor.



Execution Mode

You can select the execution mode for the SQL job that can be a session or a per-job mode. For more information, see the Executing SQL jobs in production mode documentation.

Job parallelism (threads)

The number of threads to start to process the job. Each thread consumes a slot on the cluster. When the Job Parallelism is set to 1, the job consumes the least resources. If the data source supports parallel reads, increasing the parallelism can raise the maximum throughput. For example, when using Kafka as a data source, setting the parallelism to the equal number as the partitions of the topic can be a starting point for performance tuning.

Restore From Savepoint

You can enable or disable restoring a SQL job from a Flink savepoint after stopping it. The savepoint is saved under hdfs:///user/flink/savepoints by default.

Sample Count

The number of sample entries to poll before sampling stops. To poll an unlimited number of sample entries, add 0 as the *SAMPLE COUNT* value.

Sample Window Size

The number of sample entries from active sampling displayed in the Results tab. To display an unlimited number of sample entries, add 0 as the *SAMPLE WINDOW SIZE* value.

Sample Behavior

You have the following options to choose the behavior of the sampled data under the Results tab:

- Sample all messages
- Sample one message every second
- Sample one message every five seconds

Enable Checkpointing

You can enable and disable checkpointing for a SQL job. By default the checkpointing is enabled.

Checkpoint Mode

Switching between checkpointing modes. You can choose between At Least Once or Exactly Once.

Checkpoint Interval

The time in milliseconds between checkpointing attempts.

Checkpoint Timeout

The maximum time in milliseconds until a checkpointing attempt is timed out.

Tolerable Checkpoint Failures

The number of checkpointing attempts until the job is aborted.

Failure Restart Strategy

Switching between restarting strategies when checkpointing is failed. You can choose between enabling and disabling auto recovery. By default auto recovery is enabled for checkpointing.

Adjusting logging configuration in Advanced Settings

You can customize the logging configurations for Cloudera SQL Stream Builder jobs on the Streaming SQL Console globally or at job level. Adjusting the log configuration enables you to control the log levels of all the underlying libraries: Flink, Hadoop, Kafka, Zookeeper, other common libraries, and connectors to get more or less information in your job's log.

About this task

The customization of the log configuration works differently whether it's applied globally or at job level based on the job deployment mode.

Note:

When executing a job, the order of applying configurations is as follows:

1. the job's runtime configuration,
2. the global log configuration,
3. and finally the default log configuration.

The first one found defined will be applied to the job.

Global logging configuration

Important: Only users with admin rights can access the global configuration. To grant admin rights, the user needs to be added to the ssb.admin property in Cloudera Manager.

1. Navigate to the Streaming SQL Console.
   a. Go to your cluster in Cloudera Manager.
      1. Select Cloudera SQL Stream Builder from the list of services.
      2. Click SQLStreamBuilder Console.
2. The Streaming SQL Console opens in a new window.
3. Click the Configuration tab.
4. Find the ssb.admins property.

   Tip: You can type ssb.admins into the filter bar to narrow the list of configuration properties.

5. Click the + icon on the right hand side of the last username.
6. Add the username to the new row.
7. Write a reason for change and click Save changes.
8. Restart the Cloudera SQL Stream Builder instance for the changes to be applied.

To set the global logging configuration:

1. Navigate to the Streaming SQL Console.
   a. Go to your cluster in Cloudera Manager.
      1. Select Cloudera SQL Stream Builder from the list of services.
      2. Click SQLStreamBuilder Console.
2. The **Streaming SQL Console** opens in a new window.
3. In the left hand navigation click Configuration.
4. If no global configuration is present, acknowledge the system notice by clicking Set global configuration.
5. In the editor, set the configuration.
6. Click Save.

Note: Before you add a global configuration, the editor contains the default logging configuration.

Session mode

The execution.target is set to *YARN-SESSION* mode, this is the default execution mode.

Note: Settings here will overwrite the global configuration.

The log configuration is set at the start time if the Flink YARN session is applied to every job execution. For example, the current log configuration is applied if and only if the Flink YARN session is not set on the Session tab of the Compose page.



Note: The Reset Session button only resets the SSB Session, not the underlying Flink YARN session. To do that, you have to kill the YARN application that is indicated under Flink Yarn Session on the Session tab.

Per-job mode

Per-job mode runs the job in an isolated YARN cluster. Enable it in **Job Settings** by selecting PROD - Job runs in isolated Cluster at Execution Mode.



Note: Settings here will overwrite the global configuration.

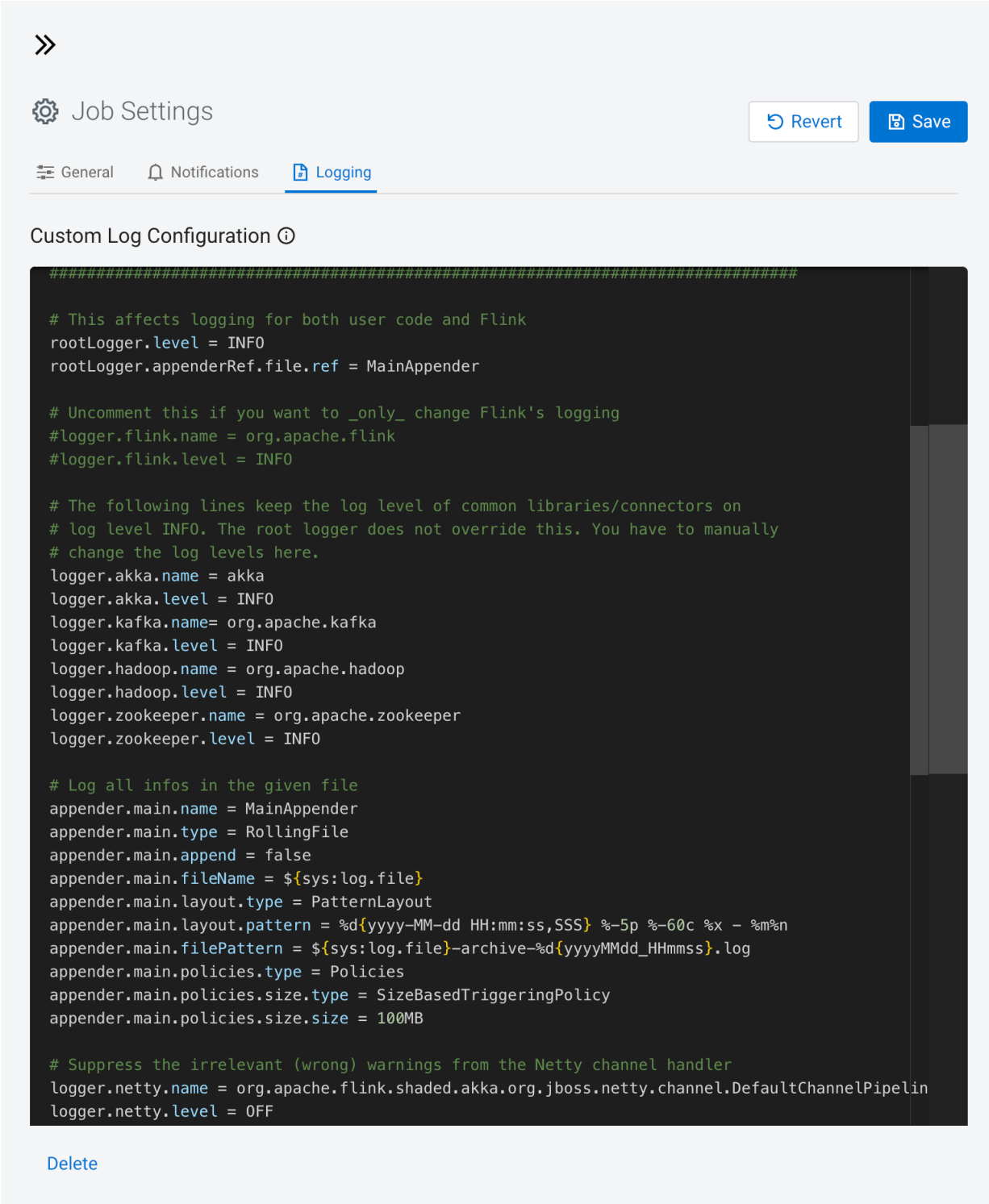
When you change the default execution mode to per-job, the currently applied log configuration is used for the job.

Procedure

1. Click New Job or select a previous job on the **Jobs** page.
   You are redirected to the **SQL Editor** of the job.
2. Click Job Settings.
3. Click Logging.
4. Click Set Log Configuration.
5. Modify the settings based on your requirements.



Important: Do not change the appender.main.fileName property from the default ${sys:log.file} or point logs to a fixed path. If you do, YARN might not aggregate those log files, and TaskManager and JobManager logs can be unavailable in the Flink Dashboard and YARN interfaces. You can still change logger levels and other Log4j settings.



6. Click Save. (You can restore the default logging configuration through clicking Delete and Set Log Configuration again.)
7. Close the **Job Settings** window.
8. Add and execute a SQL statement.
9. Click **SQL Jobs**.
10. Search for the job you have executed previously.
11. Click **Flink Dashboard**.

The **Flink Dashboard** opens in a new window.

12. Click Task Managers > Logs .

    The log information appears in the log window based on your custom configurations.

Configuring YARN queue for SQL jobs

You can configure the YARN application queue with a custom value for a SQL job using the Streaming SQL Console.

With YARN queues, you can deploy applications to a specific subset of nodes with separate or limited resources, according to configuration. This enables you to have a separate execution environment with limited resources to experiment with new applications without impacting any production operation.

You can customize the YARN application queue only for SQL jobs that run in production mode. In production mode, the job runs in an isolated cluster. The default YARN queue configured in Cloudera Manager is not changed; only the job that you run in production mode uses the queue you specify in the SQL editor.

The default value of the YARN application queue can be configured in Cloudera Manager. After accessing the SQL Stream Builder service on your cluster, open the service configurations and search for YARN application queue to change the default value. When you run a job in production mode, the queue you set with a SET statement in the SQL editor overrides that default for that job only.

Configure the YARN queue for a production SQL job

To set the YARN application queue for a SQL job, use the following steps:

1. In the **SQL Editor**, click Job Settings.
2. Under Execution Mode, select PROD - Job runs in isolated Cluster.
3. In the **SQL Editor**, remove any set 'execution.target' = 'yarn-per-job'; line from the script. Do not add that statement when production mode is set in Job Settings.
4. In the same script, add a SET statement for the queue and your SQL statement, then click Execute. For example:

```
set 'yarn.application.queue' = 'my-queue';
select * from datagen_table_1648801198
```



Important: When PROD is selected in Job Settings, do not use a SET statement for execution.target in the SQL editor. Combining both can cause the job to fail to deploy, including Kerberos login errors on YARN.

For more information about production mode, see the Executing SQL jobs in production mode documentation.

Configuring state backend for Cloudera SQL Stream Builder

The same state backend configuration is applied to the SQL jobs in Cloudera SQL Stream Builder as configured for the Flink jobs. While Hashmap stores data as object on Java heap, RocksDB can be used to store larger state that does not fit easily in memory. The RocksDB state backend uses a combination of fast in-memory cache and optimized disk based lookups to manage state.

The following state backend options are available for your Flink application:

RocksDB

By default, RocksDB is configured as the state backend for Flink. The RocksDB state backend holds in-flight data in a RocksDB database that is stored in the TaskManager local data directories and performs asynchronous snapshots. The data is stored as serialized byte arrays. The arrays are defined by the type serializer. This results in key comparisons being byte-wise instead of the hash Code() and equals() Java methods in the Hashmap state backend. The RocksDB state backend is recommended for large state and key/value states, long windows and high-availability setups. The amount of state is limited by the amount of disk space available. This also means that the maximum

throughput that can be achieved will be lower as all reads/writes through the RocksDB backend have to go through deserialization and serialization.

Hashmap

The Hashmap state backend holds data internally as objects on the Java heap. The key/value state and window operators hold hash tables that store the values, triggers and so on. The Hashmap state backend is recommended for moderate state and key/value states, long windows and high-availability setups.

You can choose between Hashmap and RocksDB as the state backend based on your performance and scalability requirements. State can be accessed and updated faster with Hashmap on the Java heap, but the size is limited by the available memory in the cluster. Even though RocksDB can scale based on available disk space and is the only state backend to support incremental snapshots, accessing and updating the state requires more resources that can affect the performance compared to storing the state in-memory. For more information about the State backends in Flink, see the Apache Flink documentation.

You can configure the state backend for your streaming application using Cloudera Manager or you can also specify the state backend in the SQL editor in Streaming SQL Console.

For Cloudera Manager

1. Open your cluster in Cloudera Manager.
2. Select Flink from the list of services.
3. Click Configuration.
4. Search for state.backend and select HASHMAP or ROCKSDB based on your requirements. ROCKSDB is set by default.



5. Click Save changes.

When configuring the state backend in Cloudera Manager, the configuration serves as a default global setting and is applied for every Cloudera SQL Stream Builder job. You can override the default set in Cloudera Manager by specifying the state backend directly when submitting your Cloudera SQL Stream Builder job.

For SQL Editor

Enable per-job mode in **Job Settings** by selecting PROD - Job runs in isolated Cluster at Execution Mode. The per-job state backend can be configured by setting the state.backend.type to hashmap or rocksdb on the Streaming SQL Console when executing your SQL job in a per-job mode as shown in the following example:

```
SET 'state.backend.type' = 'rocksdb';
```

```
SET 'state.backend.type' = hashmap;
```

Note: In case you use Hashmap as state backend, it is recommended to switch to RocksDB when a job fails with the following exception:

```
Caused by: java.lang.OutOfMemoryError: GC overhead limit exceeded
```

Configuring RocksDB

When choosing RocksDB as state backend, you can also adjust how much memory RocksDB should use as a cache to increase lookup performance by setting the memory managed fraction of the TaskManagers.

The default fraction value is 0.4, but with larger cache requirements you need to increase this value together with the total memory size. For more information, see the Set up TaskManager Memory and the Memory Tuning Guide documentations:

For Cloudera Manager

1. Open your cluster in Cloudera Manager.
2. Select Flink from the list of services.
3. Click Configuration.
4. Search for taskmanager.memory.managed.fraction and change the default value as needed.



5. Click Save changes.

For SQL Editor

Enable per-job mode in **Job Settings** by selecting PROD - Job runs in isolated Cluster at Execution Mode. You can configure the taskmanager.memory.managed.fraction on Streaming SQL Console with executing a SET statement when submitting your SQL job in a per-job mode as shown in the following example:

```
SET 'taskmanager.memory.managed.fraction' = '0.4';
```

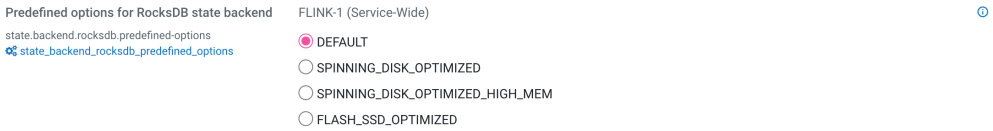
RocksDB also has predefined options that can be changed based on your system configurations:

- DEFAULT: option set by default and applicable for all settings.
- SPINNING_DISK_OPTIMIZED: option for regular spinning disks configured for better performance. The option is configured with the following values:
  - CompactionStyle=LEVEL
  - LevelCompactionDynamicLevelBytes=true
  - MaxBackgroundJobs=4
  - MaxOpenFiles=-1
- SPINNING_DISK_OPTIMIZED_HIGH_MEM: option for regular spinning hard disks configured for better performance, but with higher memory consumption. The option is configured with the following values:
  - BlockCacheSize=256 MBytes
  - BlockSize=128 KBytes
  - FilterPolicy=BloomFilter
  - LevelCompactionDynamicLevelBytes=true
  - MaxBackgroundJobs=4
  - MaxBytesForLevelBase=1 GByte
  - MaxOpenFiles=-1
  - MaxWriteBufferNumber=4
  - MinWriteBufferNumberToMerge=3
  - TargetFileSizeBase=256 MBytes
  - WriteBufferSize=64 MBytes
- FLASH_SSD_OPTIMIZED: option for flash SSDs configured for better performance. The following additional options are set:
  - MaxBackgroundJobs=4
  - MaxOpenFiles=-1

You can set the Rocksdb predefined options in the following ways:

For Cloudera Manager

1. Open your cluster in Cloudera Manager.
2. Select Flink from the list of services.
3. Click Configuration.
4. Search for state.backend.rocksdb.predefined-options and change the default value as needed.



5. Click Save changes.

For SQL Editor

Enable per-job mode in **Job Settings** by selecting PROD - Job runs in isolated Cluster at Execution Mode. You can configure the state.backend.rocksdb.predefined-options on Streaming SQL Console with executing a SET statement when submitting your SQL job in a per-job mode as shown in the following example:

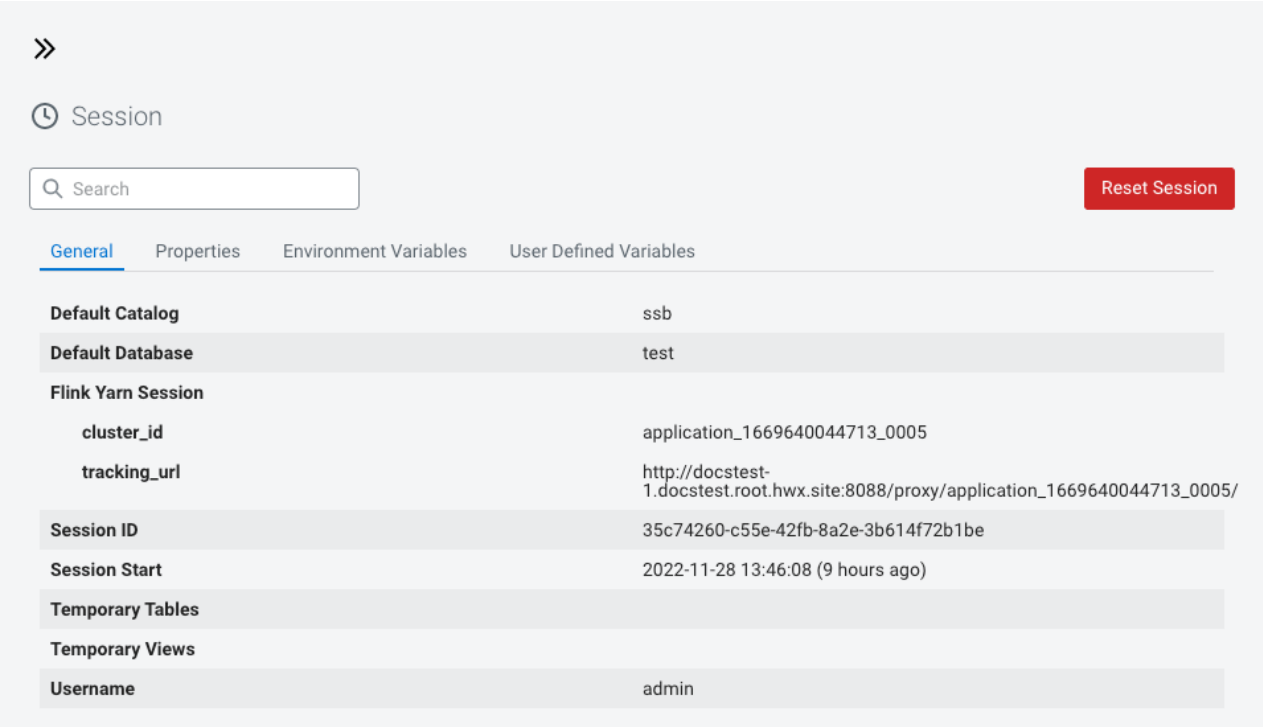
```
SET 'state.backend.rocksdb.predefined-options' = 'SPINNING_DISK_
OPTIMIZED';
SET 'state.backend.rocksdb.predefined-options' = 'SPINNING_DISK_
OPTIMIZED_HIGH_MEM';
SET 'state.backend.rocksdb.predefined-options' = 'FLASH_SSD_OPT
IMIZED';
```

Managing session for SQL jobs

By default, the SQL Stream jobs are running in a session cluster. This means that multiple Flink jobs run in the same YARN session sharing the cluster, allocated resources, the Job Manager and Task Managers. The session starts when you open the Streaming SQL Console. You can reset the session, and set the properties of the session using the Streaming SQL Console.

Resetting a session

When you reset your session, every configuration, temporary table and view, default database and catalog will be lost.



Configuring properties for a session

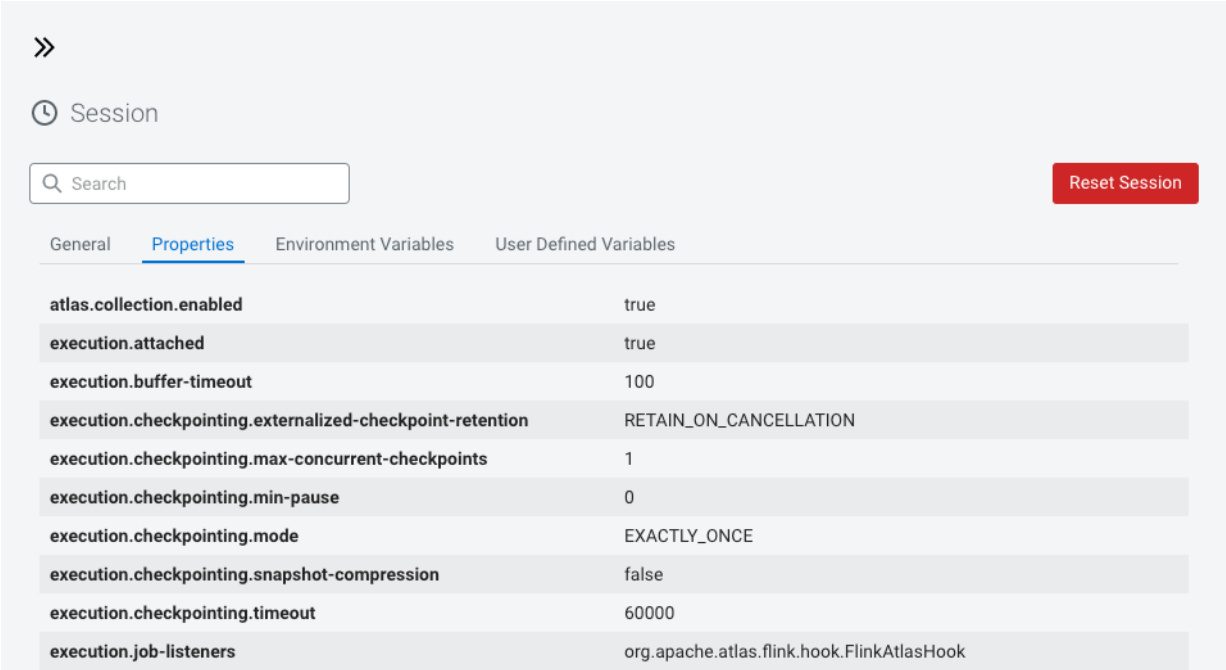
You can configure the session properties using the SET statement in the SQL window.

1. Use the SET statement in the **SQL Editor** to configure a session property.

   Example:

   ```
   SET state.backend=rocksdb
   ```

   You can review all of the configurable parameters on the Properties tab of the **Session** window.



2. Click Execute.

Executing SQL jobs in production mode

As by default the SQL jobs are running in a session cluster, there is a risk in case of a cluster failure that every job is affected within that cluster. However, you can set a per-job production mode in SQL Stream Builder to create a dedicated environment for your production jobs.

Production mode means that the deployed SQL job (Flink job) runs in a separate SQL session and on a dedicated YARN cluster configured specifically for that particular job. Set production (per-job) execution mode in the **Job Settings** window; do not use a SET statement for execution.target in the SQL Editor.

Setting production mode in Job Settings

To set the production mode for your SQL jobs in the **Jobs Setting** page, you need to select the execution mode with the following steps:

1. Click Job Settings at the **SQL Editor**.
2. Select PROD - Job runs in isolated Cluster at Execution Mode.
3. Add a SQL statement to the SQL Editor you want to execute.
4. Click Execute.

Setting the runtime mode in Job Settings

Select the desired runtime mode from the Runtime Mode dropwdown menu.

- STREAMING: the default streaming execution mode.
- BATCH: batch-stlye execution for bounded application.
- AUTOMATIC: the system decides, based on the boundedness of sources.

Functions

Creating Python User-defined Functions

With Cloudera SQL Stream Builder, you can create powerful custom functions in Python to enhance the functionality of SQL.

About this task

User functions can be simple translation functions like Celsius to Fahrenheit, more complex business logic, or even looking up data from external sources. User functions are written in Python. When you write them, you can create a library of useful functions.

Before you begin

The following Python versions are supported in Cloudera SQL Stream Builder

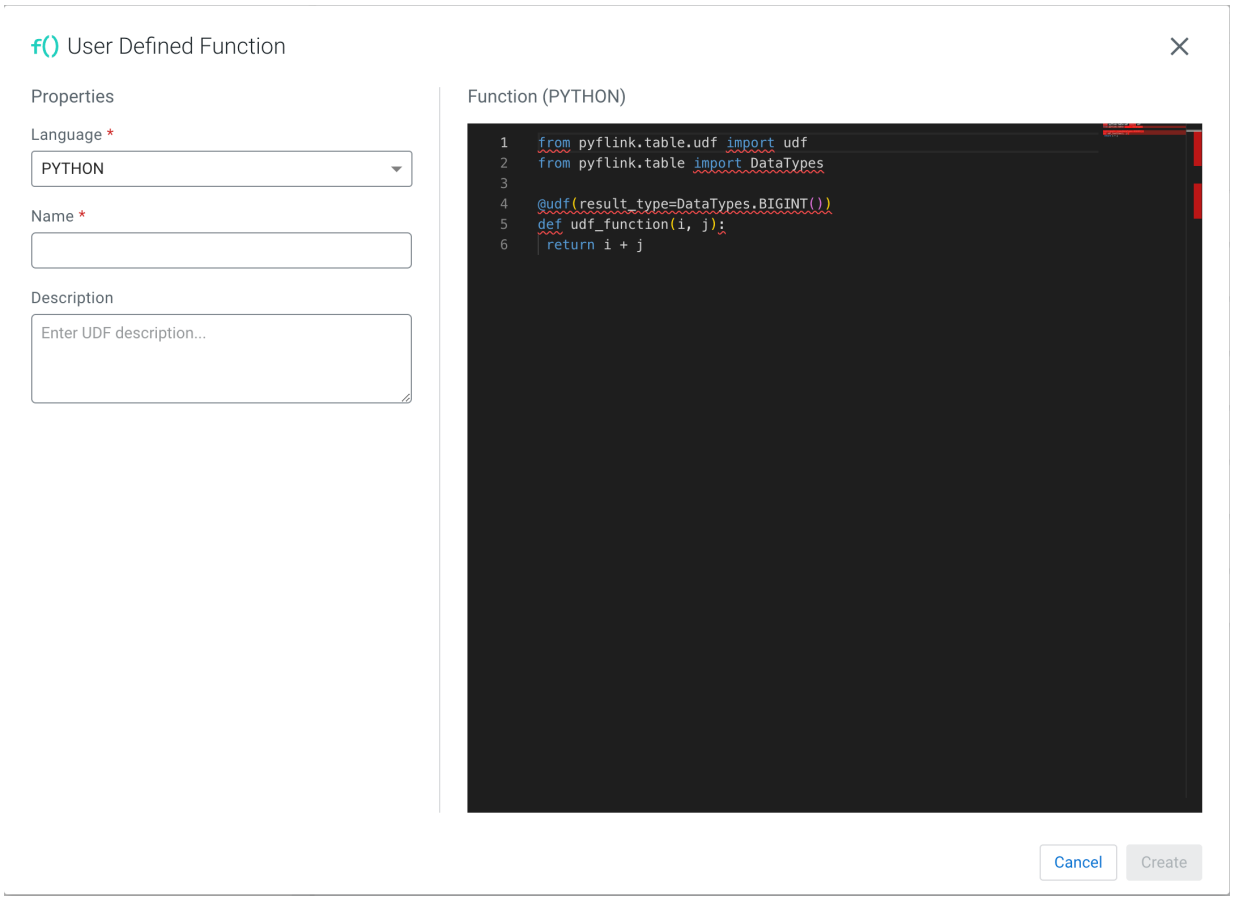
:

- 3.8
- 3.9
- 3.10
- 3.11

To use Python UDFs:

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3. Click next to **Functions**.
4. Click New Function.



5. From the dropdown menu select PYTHON.
6. Add a Name to the UDF.
   For example, name the UDF to ADD_FUNCTION.
7. Add a Description to the UDF. (Optional.)
8. Paste the Python code to the editor.

For example, a simple addition:

```
from pyflink.table.udf import udf
from pyflink.table import DataTypes

@udf(result_type=DataTypes.BIGINT())
def udf_function(i, j):
 return i + j
```



Important: The name of the function always has to be udf_function.

9. Click Create.
10. Once created, you can use the new User Defined Function in your SQL statement or as a computed column when creating a table.
    For example:

```
-- as a SQL statement
SELECT ADD_FUNCTION(myTable.n1, myTable.n2)

-- as a computed column
CREATE TABLE myTable (
  `number_a` BIGINT,
  `number_b` BIGINT
  `sum` AS ADD_FUNCTION(number_a, number_b)  -- evaluate expression and s
upply the result to queries
) WITH (
  'connector' = 'kafka'
  ...
);
```

Results

When a Flink job starts, Cloudera SQL Stream Builder will upload the Python file to the Artifact Storage, accessible for Flink to execute the UDF when called. The ssb.python.udf.reaper.period.seconds and ssb.python.udf.ttl configuration properties (set in Cloudera Manager) control the behavior of SSB to remove Python files associated with terminated jobs.

For more information on using Python UDFs, refer to the Apache Flink documentation.

Using System Functions

The same set of system functions can be used for Cloudera SQL Stream Builder as for Apache Flink.

As SSB runs on Flink, the following built-in system functions can also be used for data transformations in your SQL Jobs.

- Comparison Functions
- Logical Functions
- Arithmetic Functions
- String Functions
- Temporal Functions
- Conditional Functions
- Type Conversion Functions
- Collection Functions
- JSON Functions
- Value Construction Functions

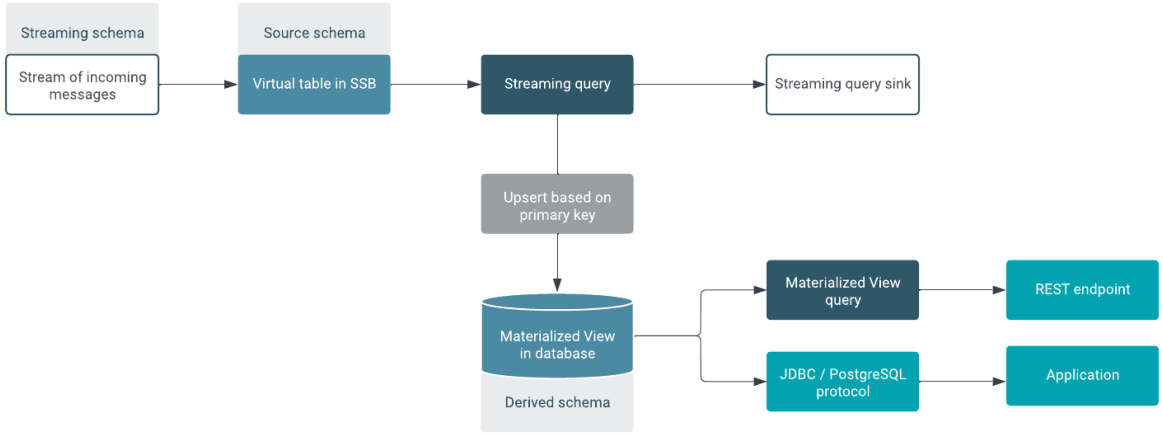
- Value Access Functions
- Grouping Functions
- Hash Functions
- Aggregate Functions
- Column Functions

For more information about the list of supported Functions, see the Apache Flink documentation.

Introduction to Materialized Views

Cloudera SQL Stream Builder has the capability to materialize results from a streaming SQL query to a persistent view of the data that can be read using REST API/endpoints. Business Intelligence tools and applications can use the Materialized View REST endpoint to query streams of data without deploying database systems. In addition to REST, Materialized Views can also be queried using JDBC/ PostgreSQL wire protocol.

Materialized Views are in synchronization with the mutating stream - they are updated by a primary key as data flows through the system. The data is updated by a given key, and it represents the latest view of the data by key. The following illustration shows the process of a Materialized View query in streaming SQL.



The following schemas are defined in the illustration:

- Streaming schema: refers to what Cloudera SQL Stream Builder knows from the source provider, for example Kafka
- Source schema: refers to the schema of the table defined by the catalog in Cloudera SQL Stream Builder
- Derived schema: refers to the schema of the Materialized View that is created in the database as a result of the streaming query against the catalog table

For example: vehicleID Z latest latitude and longitude is X and Y. As the vehicle moves, the latitude and longitude for the vehicleID are updated . The primary key is defined at creation time and is immutable.

Materialized Views can be created as mutating snapshots of the queried data result that is updated by a given key. The data is always the latest representation of itself by key (analogous to a primary key in most RDBMS systems).

You can query the Materialized Views using a GET request over REST, which returns a JSON response as "Content-Type: application/json". The queries are not defined at query time. Rather, they are curated, saved, and granted access through the Cloudera platform. You can configure a REST endpoint to query the Materialized View. Multiple query conditions can be created to allow various ways to query the same data. This is sometimes referred to as a 'pull query'.

Null Keys

In situations where information about a specific item is updated incrementally, for example, data about something is received in multiple messages, there may be keys missing from the incoming messages (for example, the streaming schema). In this case Cloudera SQL Stream Builder continues to consume these messages, but marks the missing key as NULL at the sink, therefore Cloudera SQL Stream Builder will also update the Materialized View with an upsert including the NULL values. To change this behavior, the “ignore null” option can be used, which tells Cloudera SQL Stream Builder to update the relevant record only with the non-null values of the incoming message. Similarly, when a key is removed from the source schema, but not from the streaming schema, Cloudera SQL Stream Builder ignores the key on the stream of incoming messages.

Creating Materialized Views

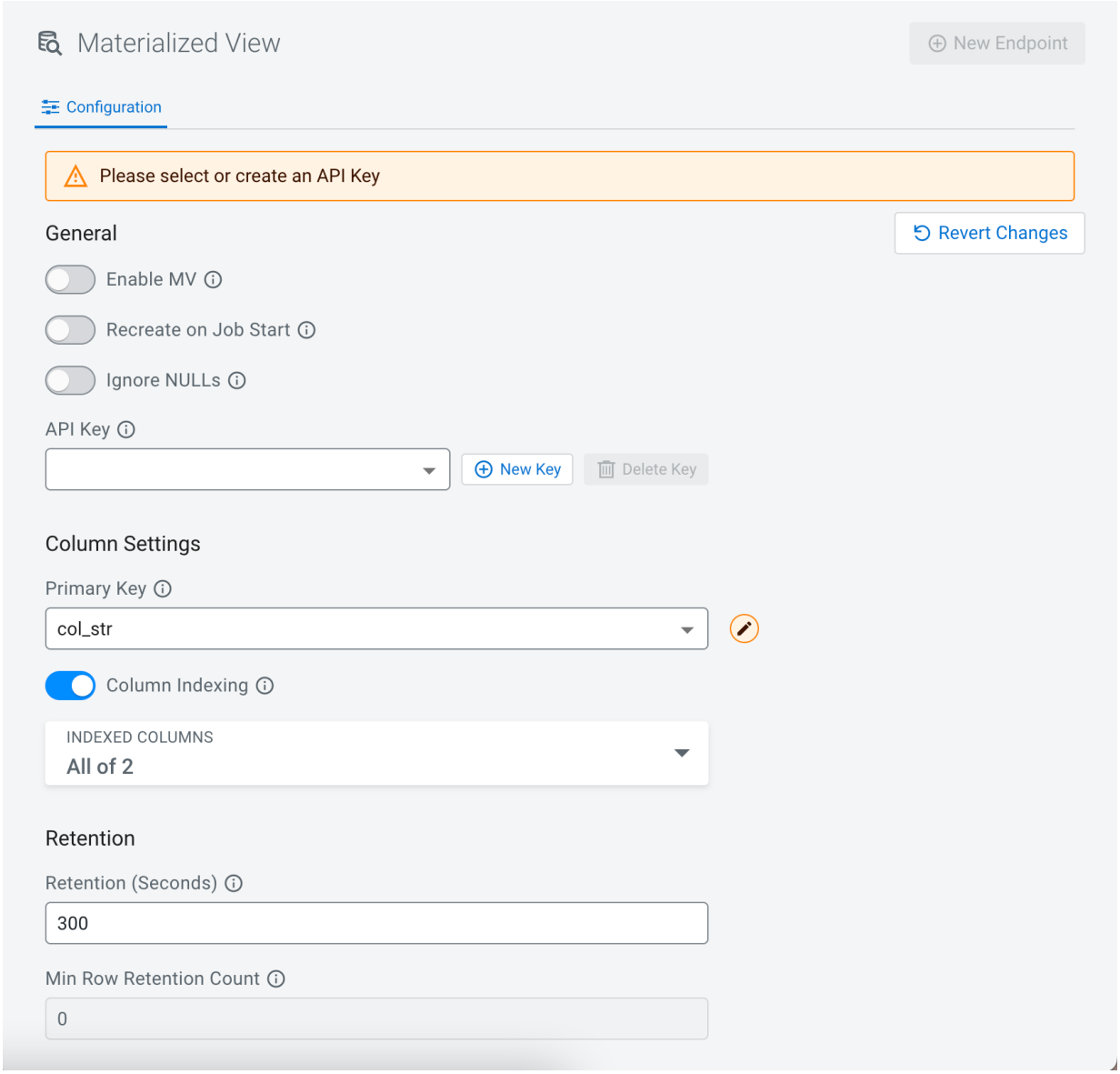
After executing a SQL Stream job, you can set up the Materialized Views to have a snapshot of your queried data. You can use the URL Pattern from the Materialized View to visualize the generated data.

Before you begin

- Make sure that PostgreSQL is installed and configured to Cloudera SQL Stream Builder to create Materialized Views.

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. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.
   You are redirected to the **Explorer** view of the project.
3.
   Click next to **Jobs** from the **Explorer**.
4. Select New Job .
   The SQL Editor for the created job opens in a tab.
5. Add a SELECT statement to the SQL Editor.
6. Click Materialized View.
   The configuration window for the Materialized View opens.

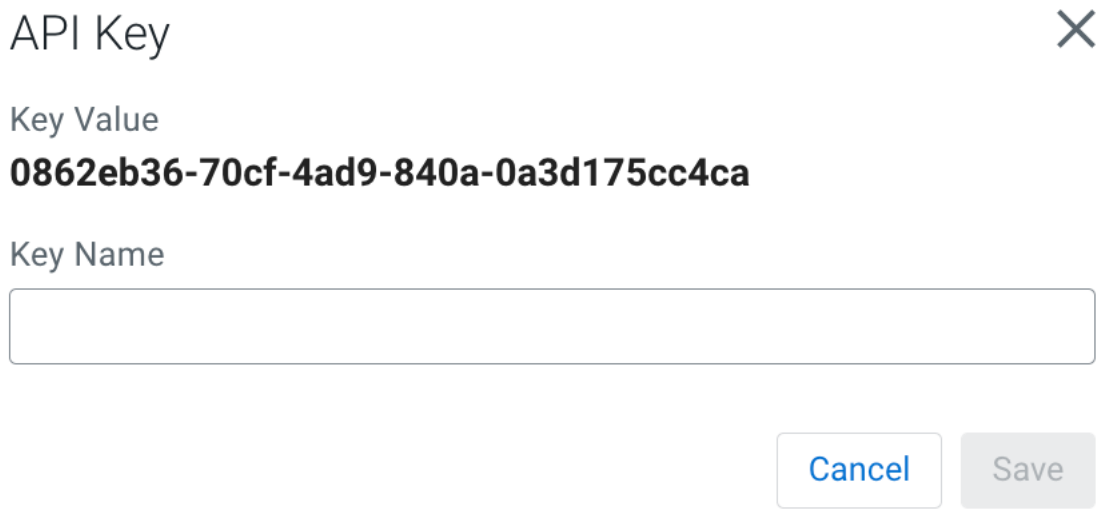


7. Switch Enable MV toggle.
8. Enable or disable Recreate on Job Start.

   If enabled, the Materialized View is deleted when a job is started or restarted.
9. Enable or disable Ignore NULLS.

   If enabled, NULL values will NOT update values that are non null - they are ignored.
10. Select an API Key.

    In case there are no API Keys, click Add API key, or click Materialized Views from the main menu. The add API key window appears. Provide a name for the API key, and click Save Changes.

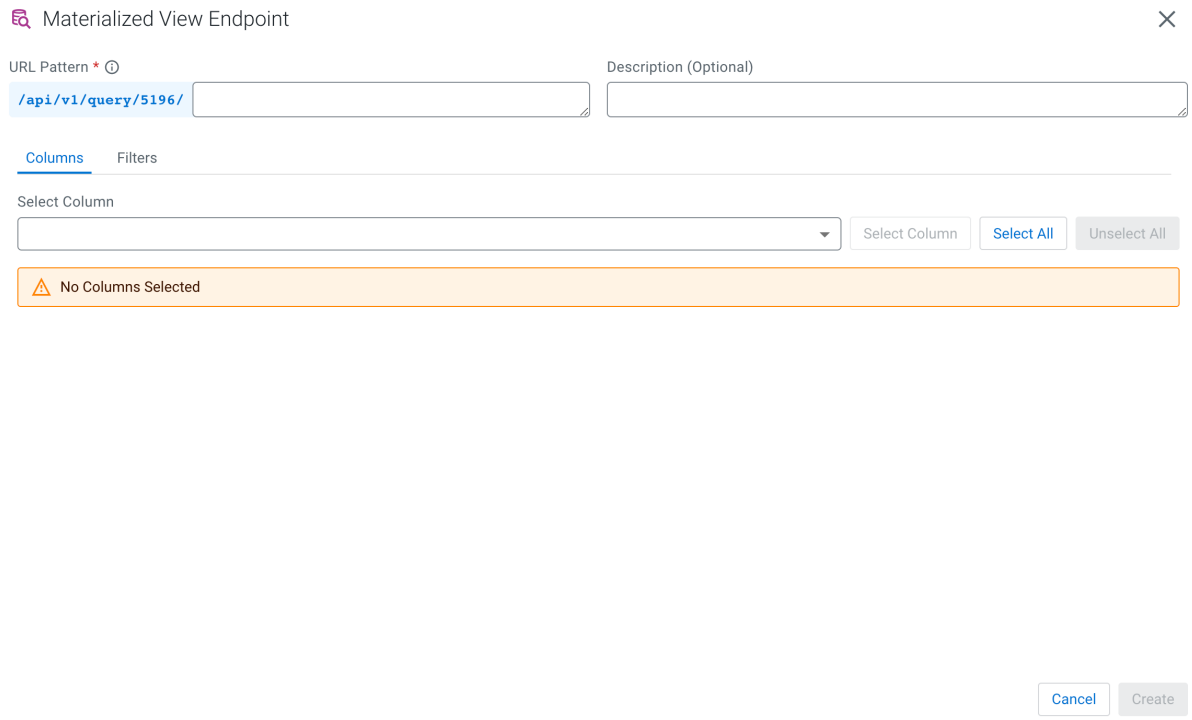


To check your created API keys:

a. Click Manage at the **API Keys** from the Project Explorer, or click on the created API key under the **API Keys**.



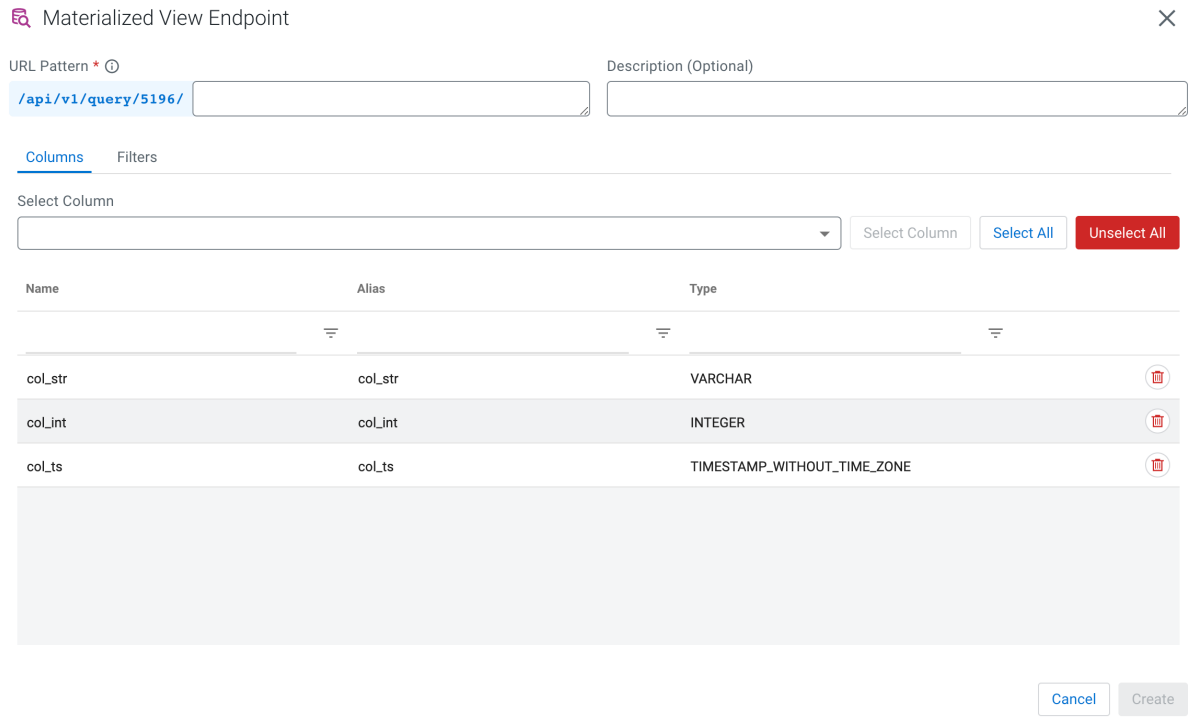
11. Configure the **Column Settings** for the Materialized View.
    a) Select a Primary Key.
       If this list is empty, then no SQL is specified in the **SQL Editor** or the SQL query is invalid. Select a key as a primary key for the Materialized View. All data will be updated by this key.
    b) Enable Column Indexing if needed, and select the column by which you need the results to be indexed.
12. Configure the retention time for the Materialized View.
    For more information about how you can set the retention time, see the Configuring Retention Time for Materialized Viewsection.
13. Click New Endpoint to create the Materialized View query.
    The **Materialized View Endpoint** window appears.



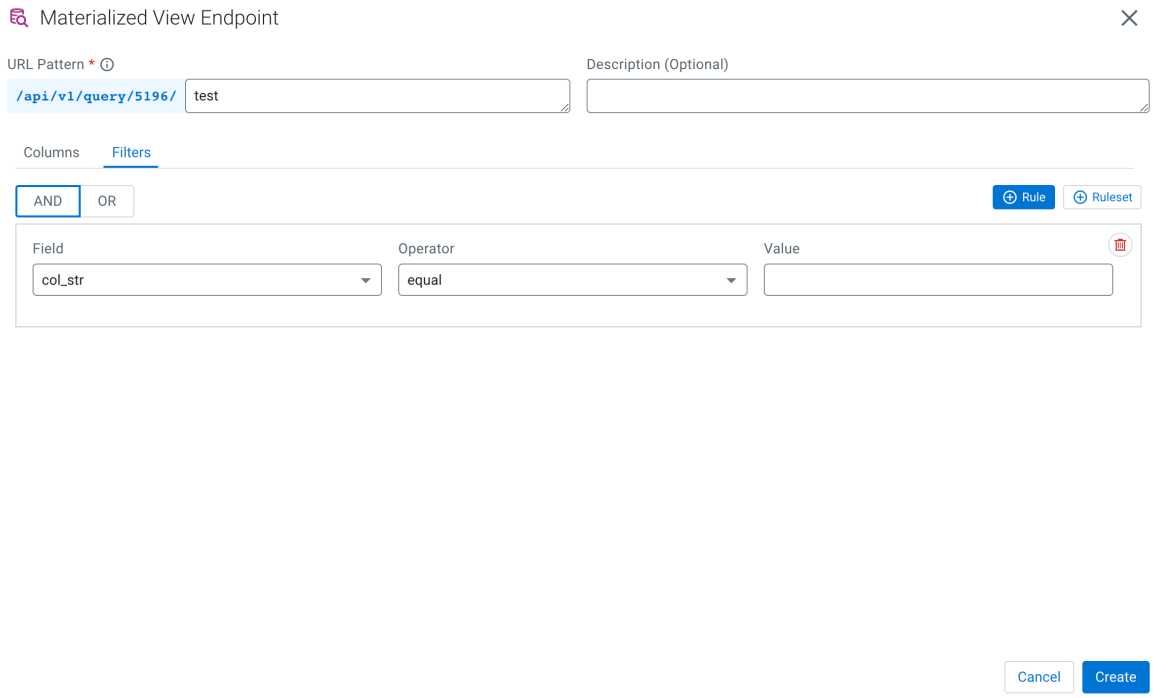
14. Provide a name in the **URL Pattern** field.

15. Provide a description of the Materialized View Query, if needed.

16. Customize the Materialized View in the **Endpoint Editor**.



a) Select the columns of the SQL job you want to use in the Materialized View Query.
   You can select every column in the table by clicking on the Select All button.
b) Click Filters tab to apply computations and further enrichment of your data.

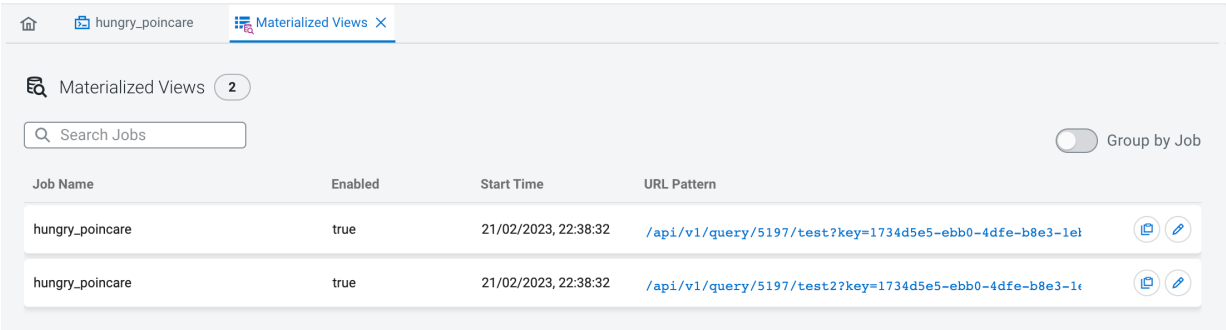


c) Provide details of the filter by selecting the Field, Operator and Value.

You can add more rules or set a ruleset for the query.

17. Click Create.

You can access the queried results by clicking Manage at the **Materialized Views** from the Project Explorer, and by opening the URL pattern.



Results

You can click on the created REST endpoint to review the data, or copy it and visualize the queried data in a Business Intelligence tool, notebook, code and so on. You can also review the list of Materialized View and API keys on the Materialized View page.

Configuring Retention Time for Materialized Views

When creating Materialized Views, you can configure how the system should retain the data rows in the Materialized Views. You can either choose between retaining the data by time or the row count.

The Materialized Views configuration allows you to set one of the following configuration parameters for data retention:

- Retention Time
- Min Row Retention Count

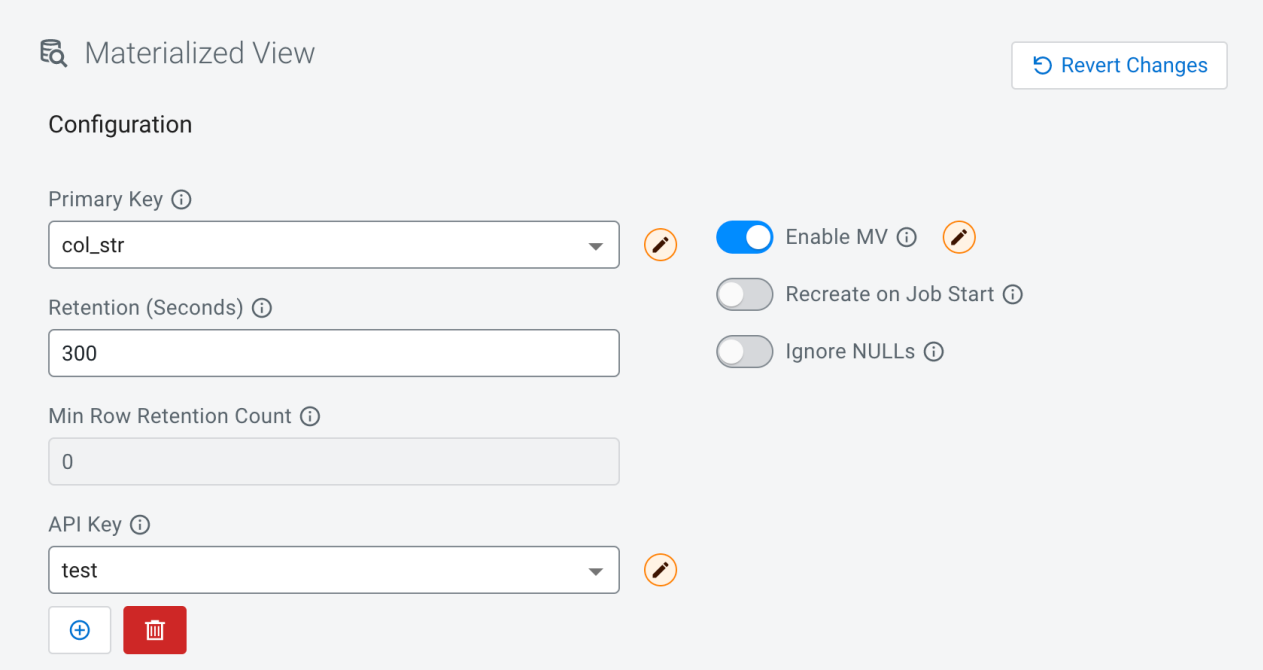
You can specify the Retention Time and Min Row Retention Count when creating a Materialized View for a SQL job on the Compose page of the Streaming SQL Console.

Retention Time

Retention Time is specified in seconds, and it tells the system to retain data rows as old as the specified retention time. The rows that are outside of the retention time are removed from the Materialized View.

Note: You can only add a Retention Time value, if the Min Row Retention Count field is empty or set to 0.

The following example shows how to set a Retention Time of 300 seconds. This means that only those rows are included in the Materialized View that are within the 300 seconds of the job execution. The older rows are removed from the Materialized Views.

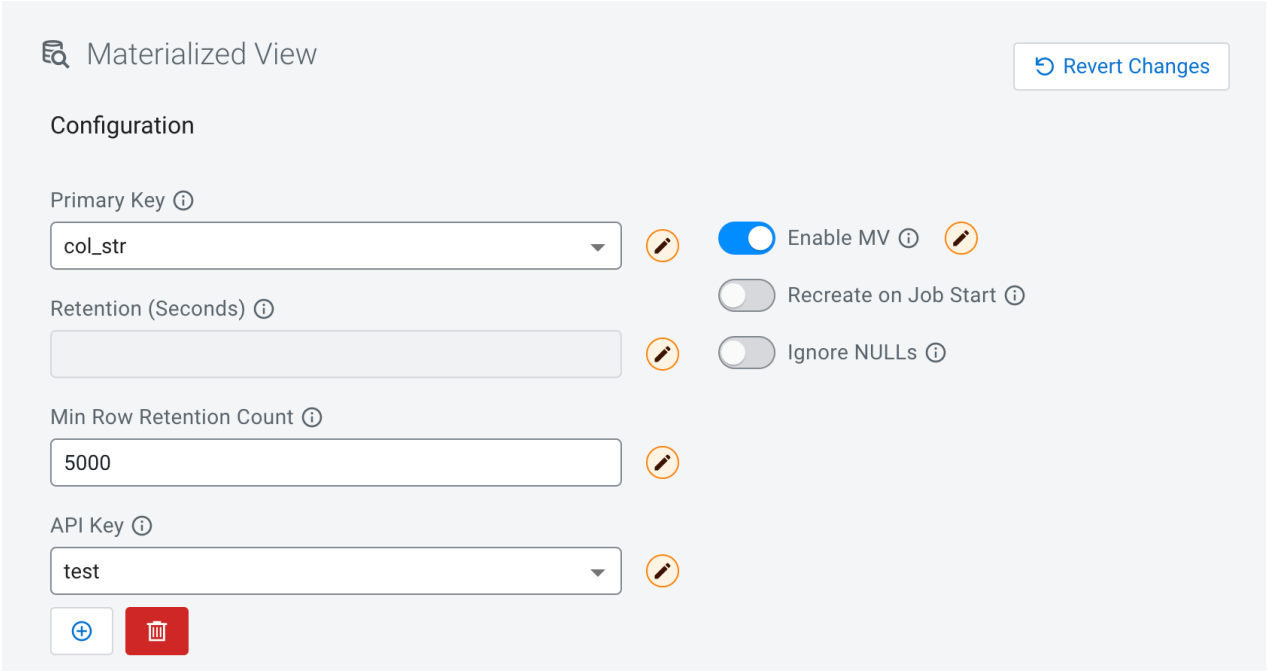


Minimum Row Retention Count

The Min Row Retention Count parameter indicated to the system to maintain a specific number of data rows in the Materialized View.

Note: You can only add a Min Row Retention Count value, if the Retention Time field is empty or set to 0.

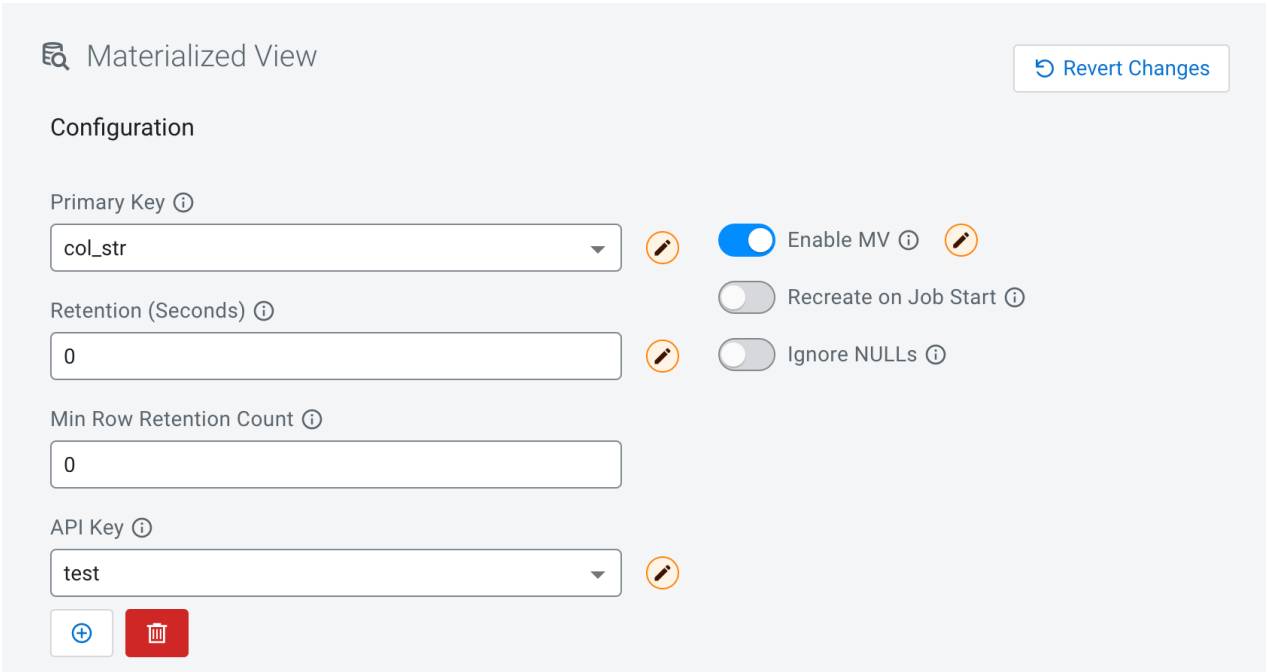
The following example shows how to configure the system to retain the last 5000 data rows. This means that only the first 5000 data rows are included in the Materialized View, and the data rows from 5001 are removed from the Materialized View.



Retaining all data without limit

In order to retain all data rows, regardless of time, or number of rows in the Materialized View, both settings can be reset to zero (0). This indicates to the system that all data must be preserved without a time limit.

The following example shows how to configure the Retention parameters to keep all of the data regardless of time:



Materialized View Pagination

You can set a limit and order the results of a Materialized View query by adding values to the limit and offset configurations.

After creating a Materialized View, you can order and limit the results after opening the Materialized View. The default limit of the results is 100 entries. You can set the limit and offset using the REST URL. You need to modify the URL address in your browser by setting the value and the offset value as shown in the following example:

```
localhost:18131/api/v1/query/5275/summary_query?key=6ef33a7a-c82a-4a72-946b-5fe0b33e033c&limit=20&offset=3
```

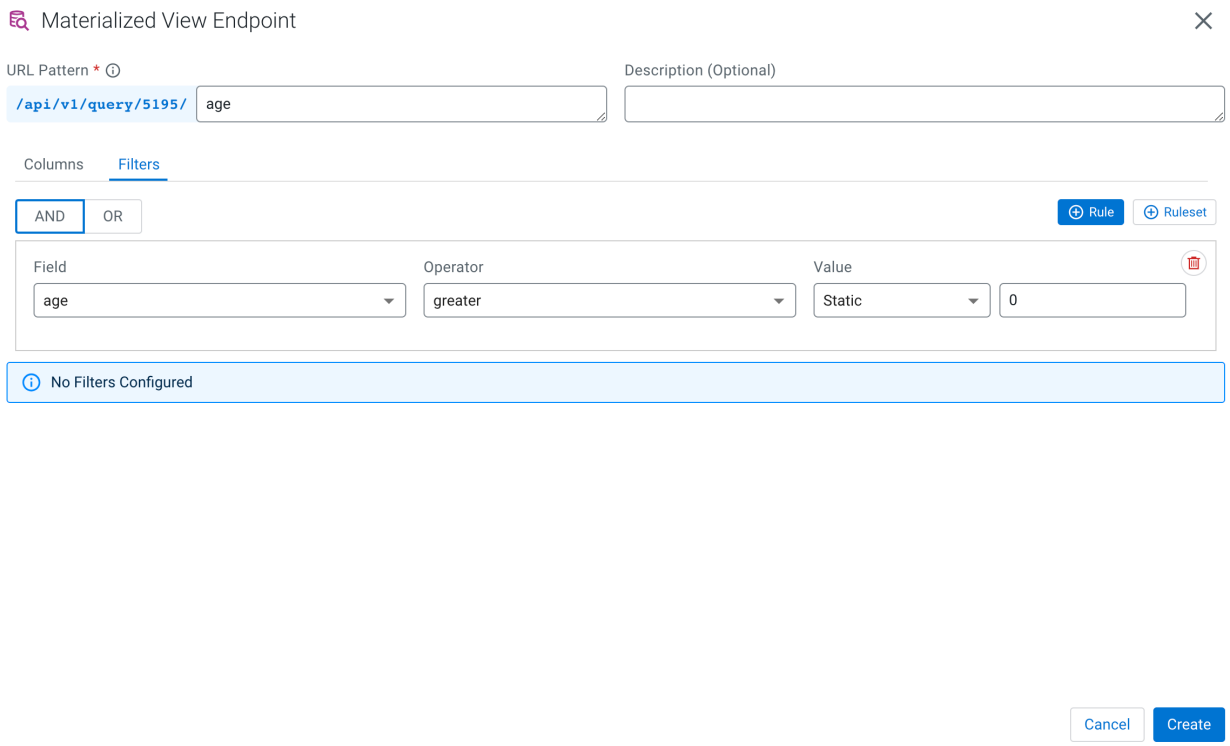
When you edit the Materialized View, the set limit and offset change based on the new filters set for the Materialized View query. If you set an invalid number for the limit and offset, an error message is displayed.

Using Dynamic Materialized View Endpoints

You can use static or dynamic REST endpoints when creating Materialized Views in Cloudera SQL Stream Builder. After setting filters for the Materialized View query, you can further filter down the results by using variables in the endpoint URL.

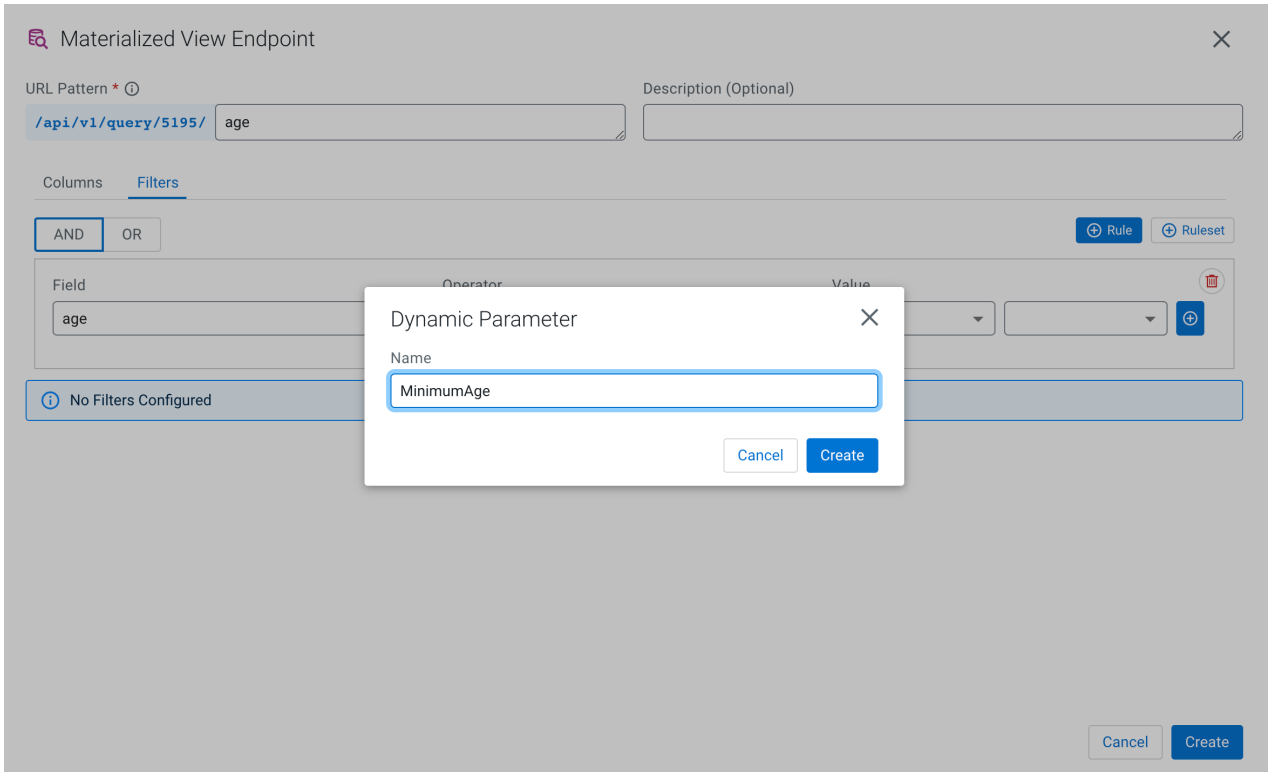
Difference between static and dynamic endpoints

When using a static endpoint, the endpoint URL is a constant string. The logic of filtering is defined statically at the endpoint configuration and the results of the endpoint call will always reflect this. A static endpoint is one that only uses filters with static values. For example, in the following static endpoint the Filter is set to age greater than 0, and the endpoint URL is always /api/v1/query/5200/*AGE*:

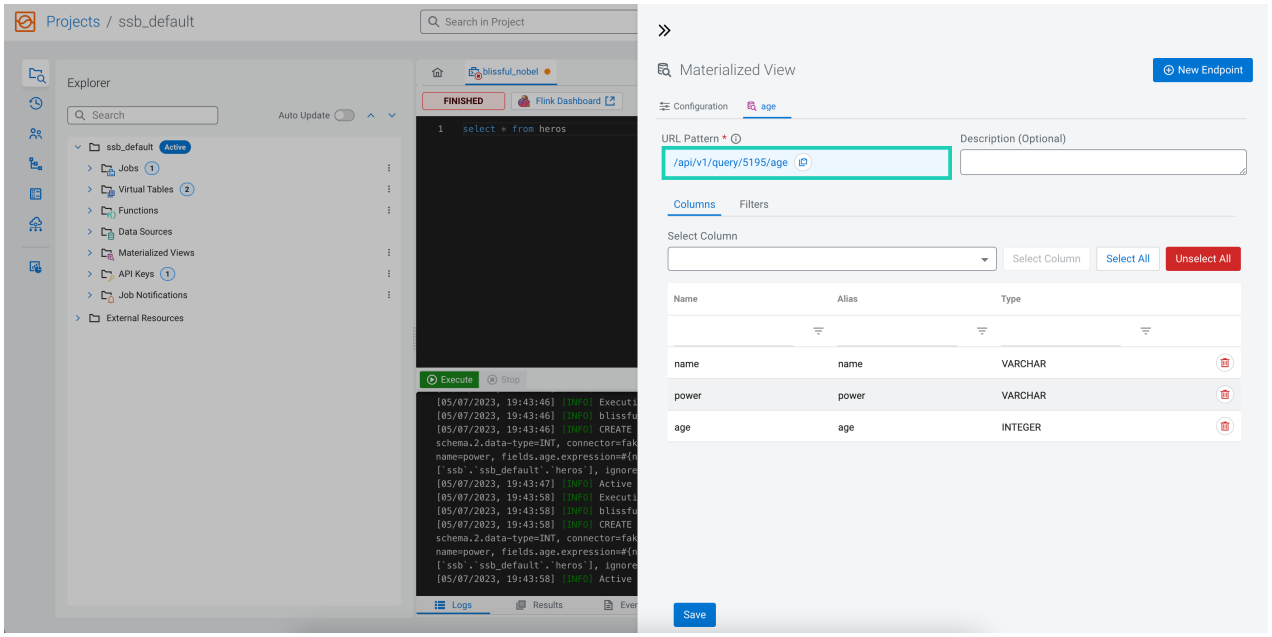


Dynamic materialized view endpoints use filters that reference parameters instead of static values. The value of these parameters are specified when the endpoints are used.

To create a dynamic filter, after selecting the Value type as Dynamic, click on the  button to add a new parameter, or select the parameter from the drop-down list, if it had already been created previously. When creating a new parameter, you will be prompted to provide the name of the Dynamic Parameter.



After specifying the dynamic parameter, click Create. Once the job is running, you can test the Materialized View endpoint by clicking on the endpoint URL in the endpoint page:



For dynamic endpoints, you will be prompted for the values of the dynamic parameters when opening the REST endpoint of the Materialized View.

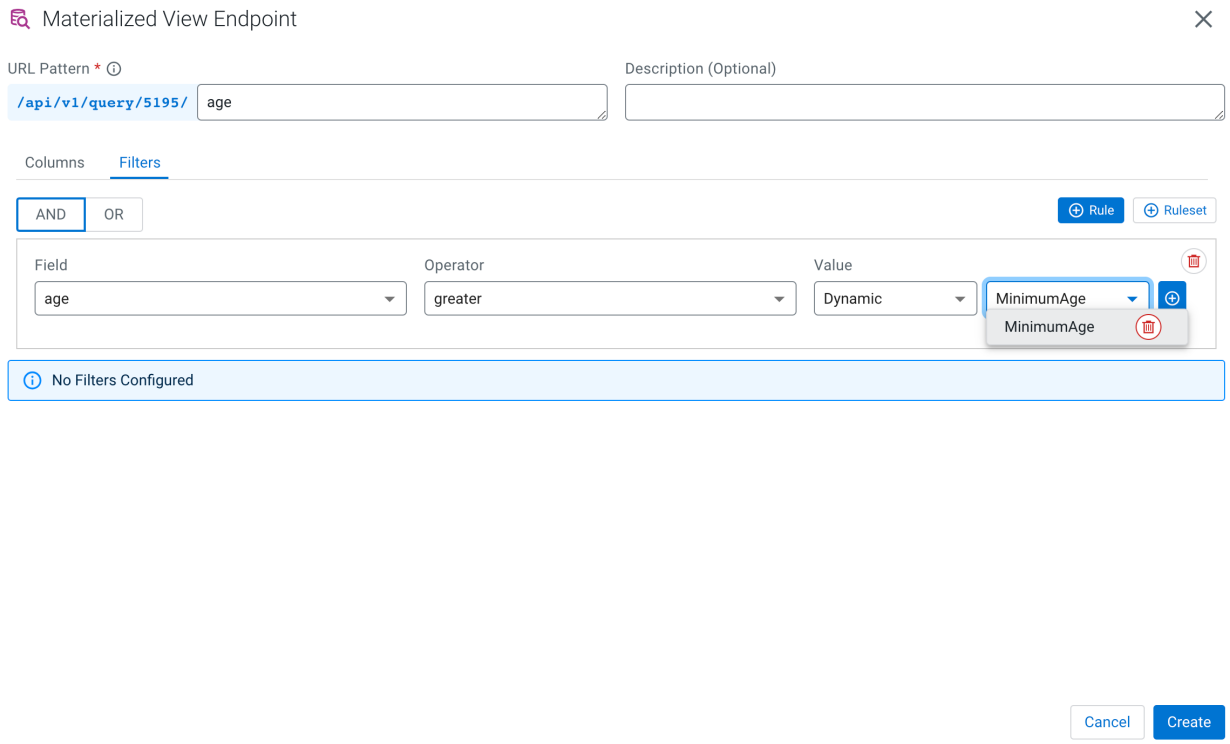


The dynamic parameters are defined as URL query parameters as shown below:

```
<ENDPOINT_URL>?key=<API_KEY>&limit=<MAX_RECORDS>&param1=<VALUE1>&par
am2=<VALUE2>...
```

The advantage of a dynamic endpoint is that by providing dynamic parameters and a corresponding value, you are able to filter the queried results at runtime, returning only the values you exactly need from that dataset.

You can use multiple dynamic parameters in the endpoint and you can also use the same dynamic parameter at the different filters to create more specific filtering methods for the results of the Materialized View query. When removing a filter from a Materialized View, the Dynamic parameter is not affected, which means that if a filter is deleted from the Materialized View query, the value for Dynamic Parameter still needs to be provided when opening the REST endpoint. The Dynamic Parameter needs to be deleted separately from the filter by clicking on the Name of the Dynamic Parameter, and selecting the delete option from the drop-down menu as shown in the following example:



Configuring Materialized View database information

In Cloudera Base on premises, PostgreSQL is configured during the installation of Cloudera SQL Stream Builder as a service. You can review the database configuration for Cloudera SQL Stream Builder using Cloudera Manager.

About this task

Cloudera SQL Stream Builder uses PostgreSQL database to store the queried data of the Materialized View feature. In Cloudera Base on premises, you can only use PostgreSQL as a database for Cloudera SQL Stream Builder. In case you need to connect to the PostgreSQL database of Cloudera SQL Stream Builder, you can review the necessary information of the database in Cloudera Manager. You can also configure the user and the password of the database using CLI, and then updating the existing information in Cloudera Manager.

Where to find the PostgreSQL database information?

1. Go to your cluster in Cloudera Manager.
2. Select Cloudera SQL Stream Builder from the list of services
3. Click Configuration.
4. Filter down the configuration parameters to Materialized View Engine.

   The following information is listed about the PostgreSQL database:

   - Database Name
   - Database Host
   - Database Port
   - Database User
   - Database Password

When you want to change the default configuration of PostgreSQL database, first you need to change the information in PostgreSQL and then update the configuration in Cloudera Manager for Cloudera SQL Stream Builder.

Procedure

1. Connect to the database host using ssh.

   ```
   ssh [***USERNAME***]@[***DATABASE HOST***]
   Password: [***PASSWORD***]
   ```

2. Connect to PostgreSQL.

   ```
   sudo -U postgres psql
   ```

   Note: As an EnvironmentUser, you will be prompted to provide the password related to the PostgreSQL client. In this case, please reach out to your administrator to provide the correct password for the PostgrSQL client connection.

3. Change the username or the password for the database.

   ```
   ALTER USER eventador_admin WITH PASSWORD [***NEW PASSWORD***];
   ALTER USER eventador_admin RENAME TO [***NEW USERNAME***];
   ```

   Note: When changing the database username, the existing password for the username is deleted. You must change the password as well when renaming the user of the database.

   After changing the password or username, you need to update the database information in Cloudera Manager.

4. Navigate to back to your cluster in Cloudera Manager.
5. Click Cloudera SQL Stream Builder Configuration .
6. Filter down the configuration parameters to Materialized View Engine.

   The list of Materialized View Engine paramaters are displayed.

7. Update the Database user or the Database password field based on the new parameters you provided in PostgreSQL.

Using Cloudera SQL Stream Builder with Cloudera Data Visualization

You can create Business Intelligence reports from the Materialized Views created in Cloudera SQL Stream Builder using Cloudera Data Visualization. To integrate Cloudera SQL Stream Builder with Cloudera Data Visualization, you need to provide the PostgreSQL database information of Cloudera SQL Stream Builder in Cloudera Data Visualization.

About this task

After creating Materialized Views of your SQL Stream job, you can create visualized reports from your queried result with Cloudera Data Visualization.

Cloudera Data Visualization enables you to explore data and communicate insights by using visual objects. You can connect to your data in Cloudera, create state-of-the-art visualizations on top of your datasets, build informative dashboards and applications, and publish them anywhere across the data lifecycle.

When connecting Cloudera SQL Stream Builder with Cloudera Data Visualization, you need to provide the PostgreSQL database information that stores the Materialized View data in the Cloudera Data Visualization web interface. For the connection, you can find the PostgreSQL database information in Cloudera Manager.

Before you begin

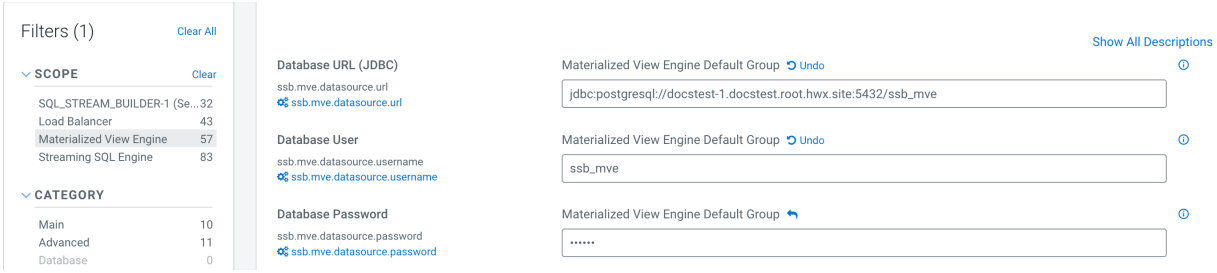
You have created a Materialized View query on your running SQL job.

Procedure

1. Go to your cluster in Cloudera Manager.
2. Click Cloudera SQL Stream Builder from the list of services.
3. Click Configuration.
4. Filter down the configuration parameters to Materialized View Engine.

   The list of Materialized View Engine paramaters are displayed.
5. Save the ssb.mve.datasource related information.

   Regarding the PostgreSQL information you need for the connection, the Database URL (JDBC) configuration parameter contains the PostgreSQL hostname, the PostgreSQL port and the PostgreSQL database name.



   When creating a data connection in Cloudera Data Visualization, you need to provide the connection information and credential as shown in the following example:

   - Database Name: ssb_mve
   - Database Host: docstest-1.docstest.root.hwx.site
   - Database Port: 5432
   - Database User: ssb_mve
   - Database Password: *[***POSTGRESQL DATABASE PASSWORD***]*

   The Database User and Database Password for the Materialized View Engine is configured based on what username and password was provided when installing the Cloudera SQL Stream Builder service on your cluster.

What to do next

After collecting the necessary information to the connection, you need to access the Cloudera Data Visualization web interface and create a connection to the PostgreSQL database of Cloudera SQL Stream Builder using the Cloudera SQL Stream Builder connector. For more information, see the Cloudera Data Visualization documentation.

Important: The Cloudera SQL Stream Builder connector is provided as a technical preview at this time in Cloudera Data Visualization. The tool is still under development and not recommended for a production environment.

Using widgets for data visualization

The built-in data visualization tool in Cloudera SQL Stream Builder enables you to present the sampling data and the results of the Materialized View query using widgets on Streaming SQL Console. As the widgets are integrated into Cloudera SQL Stream Builder, the visualization tool works out of the box without any dependencies, which offers easy access to the underlying, running jobs as data sources.

You can use widgets on Streaming SQL Console to visualize data from the samples of a running job or from the results of a Materialized View query. Various visualization types and configuration options are available to visualize data. Widgets automatically refresh to update the visualization based on the data sampled directly from the job output or from Materialized Views.

Creating widgets

Widgets can be created on Streaming SQL Console when a SQL query is submitted and the job is successfully running.

Before you begin

- Ensure that you have a running job with in progress sampling or you have a running job with a Materialized View endpoint.

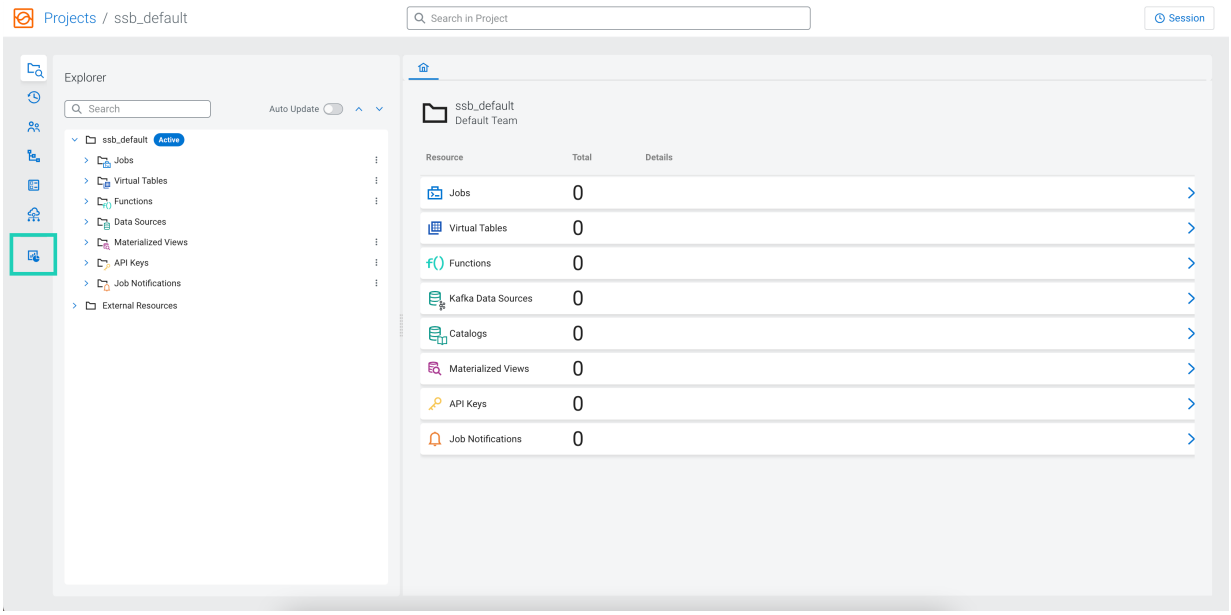
  For more information, see the Running SQL Stream jobs and Creating Materialized Views documentation.

Procedure

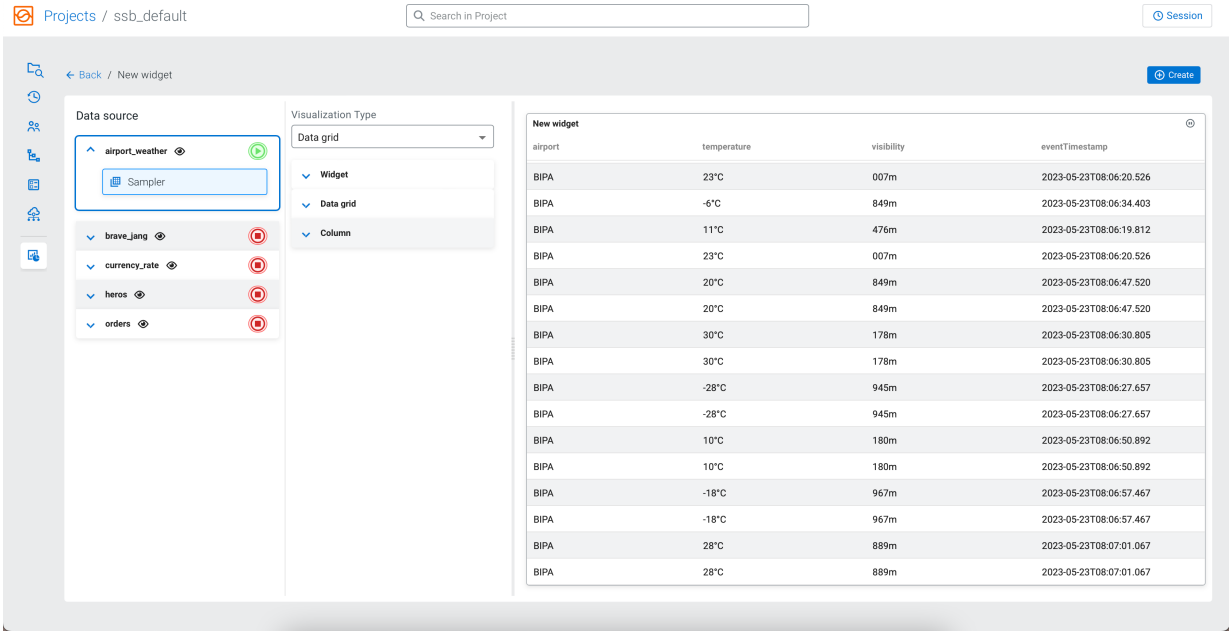
1. Navigate to the Streaming SQL Console.
   a) Go to your cluster in Cloudera Manager.
   b) Select SQL Stream Builder from the list of services.
   c) Click SQLStreamBuilder Console .

   The **Streaming SQL Console** opens in a new window.
2. Open a project from the **Projects** page of Streaming SQL Console.
   a) Select an already existing project from the list by clicking the Open button or Switch button.
   b) Create a new project by clicking the New Project button.
   c) Import a project by clicking the Import button.

   You are redirected to the **Explorer** view of the project.
3. Click Dashboard from the Project Manager.



4. Click New Widgets.
   The **Widget editor** page appears.



The Widget editor page consist of the following windows:

- **Data source**, where you can select the job and its corresponding data sources for the widget.
- **Visualization Type** where you can provide information and customize the visualization type.
- **Preview of the created widget**, where you can check the configurations and customization of the widget that will be added to the **Dashboard**.

5. Select a Data Source.

   The running jobs will be automatically displayed and categorized based on the Sampler or the Materialized View endpoint name of the corresponding job.For more information, see the Choosing data sources section.

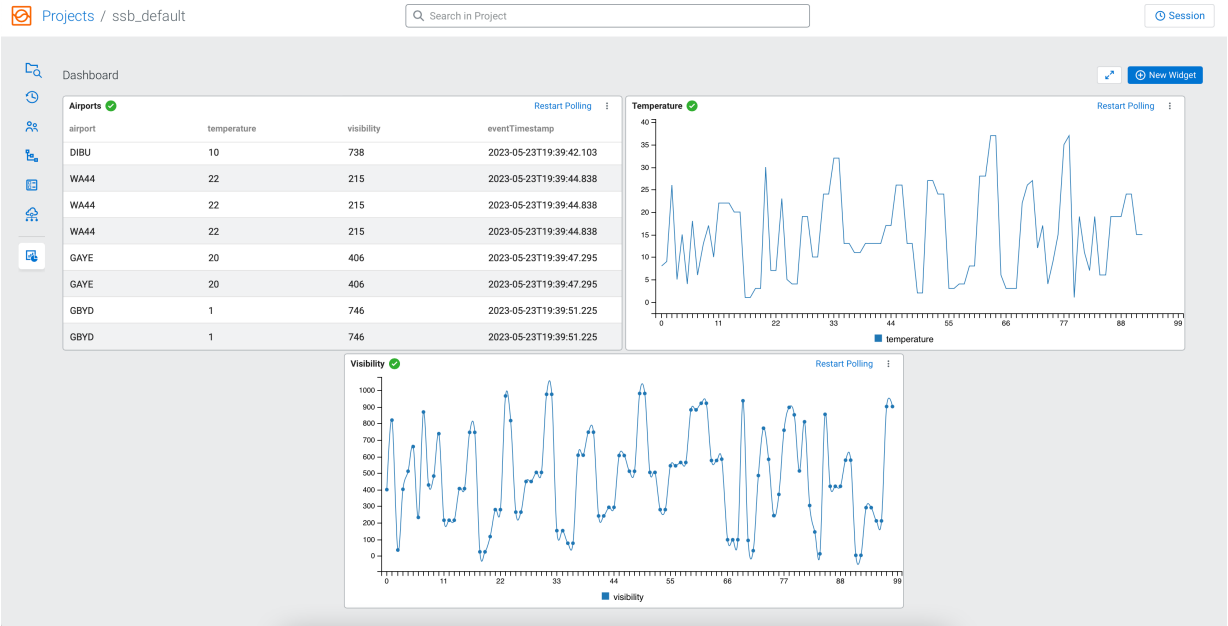
6. Select a Visualization type.

   A preview is available from the selected data source and visualization type.

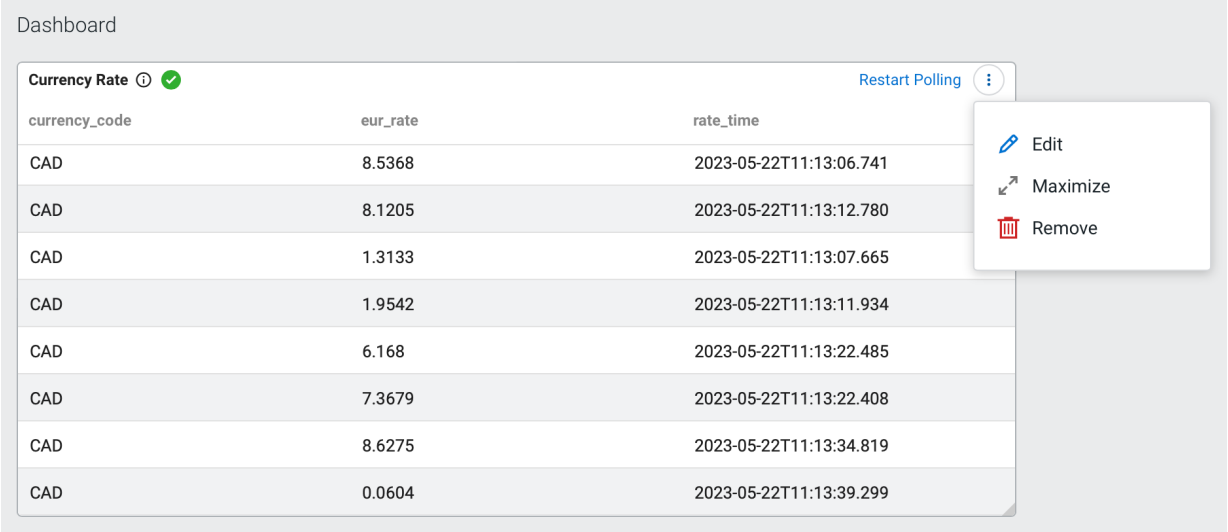
7. Customize the Visualization based on the available options.

For more information, see the Customizing visualization types section.

8. Click Create.
The created widget is added to the **Dashboard** page.



You can manage the created widget on the Dashboard by clicking , and selecting Edit to go back to the widget editor page and make changes to the visualization, or Remove to delete the widget.



What to do next

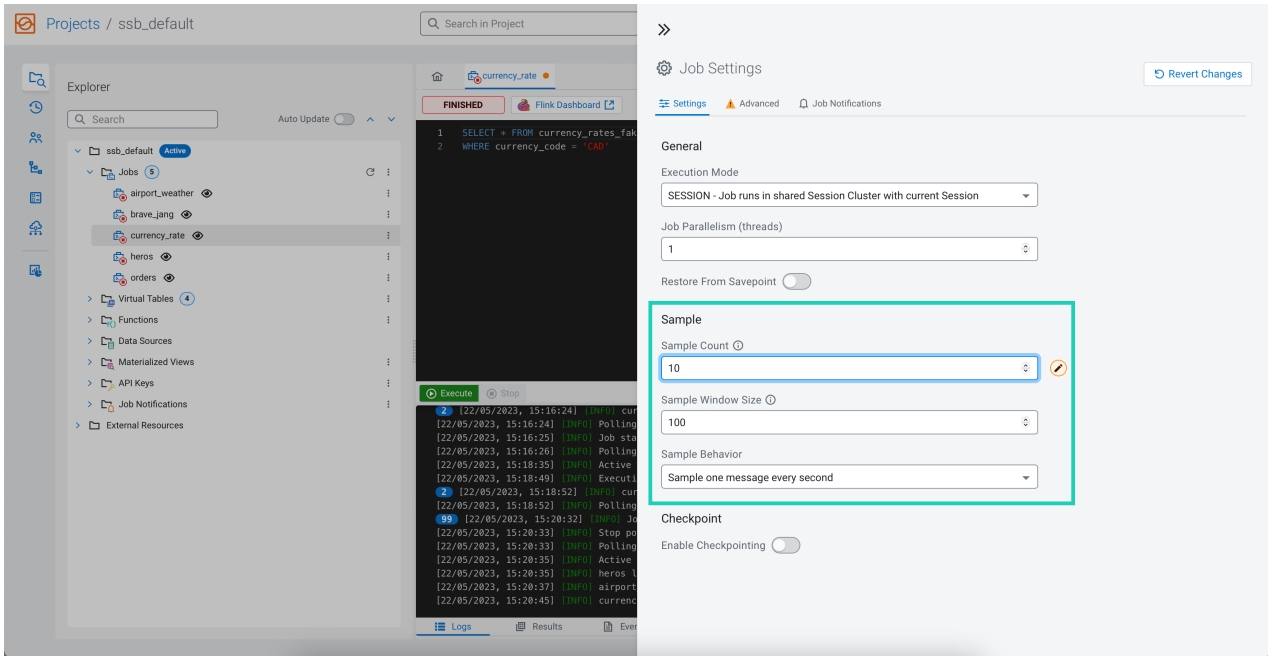
You can create more widgets to the Dashboard page. You can resize and reorganize the widgets based on your requirements. For more information, see the Managing widgets on the Dashboard section.

Choosing data sources

The data sources are available for visualization as soon as a job sampling is in progress or a Materialized View endpoint is created. In both cases, the data is shown in the widgets when the corresponding SQL jobs are in running state.

Samples as data source

When using sampling as a data source, you must have a running job on Streaming SQL Console, which produces sampling data as the job runs. You can change the default sampling behavior at the **Job Settings** that affects the frequency, count and the sample window size of the sampled messages.



Materialized View as data source

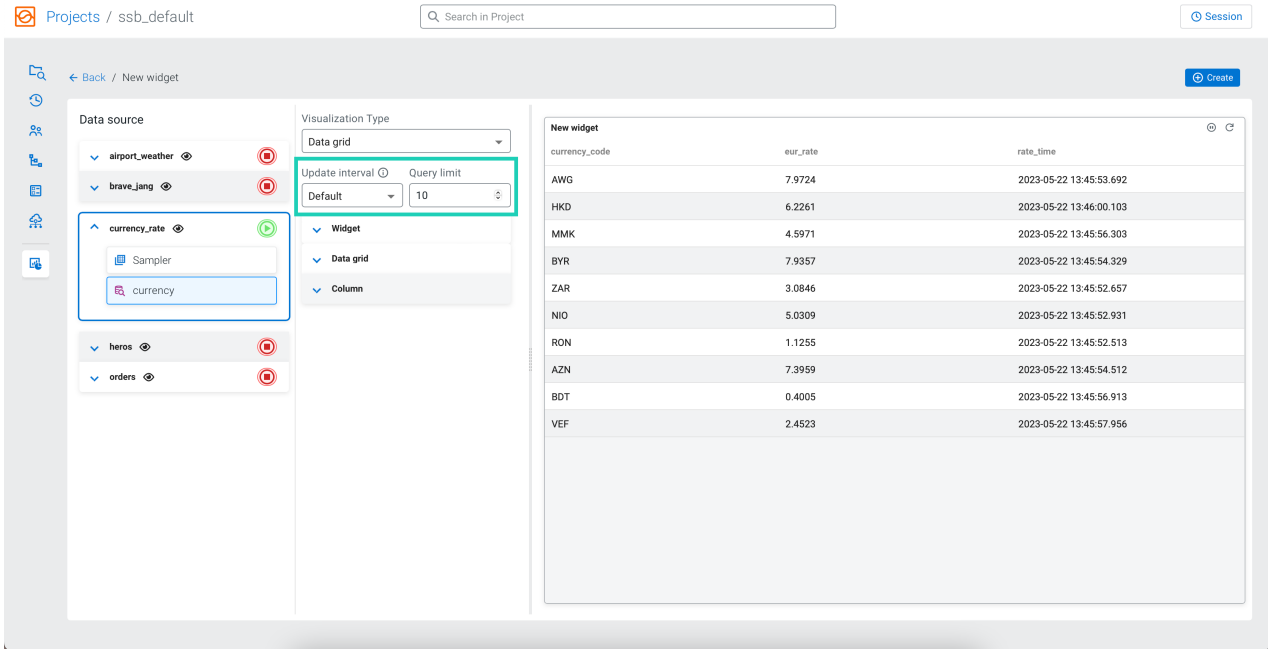
When using Materialized View as a data source, you must have a running job with a Materialized View query. The Materialized View endpoint is called in a period of time according to the update interval and query limit configured for the widget.

Update interval

The update interval is the time the widgets waits to call from the endpoint.

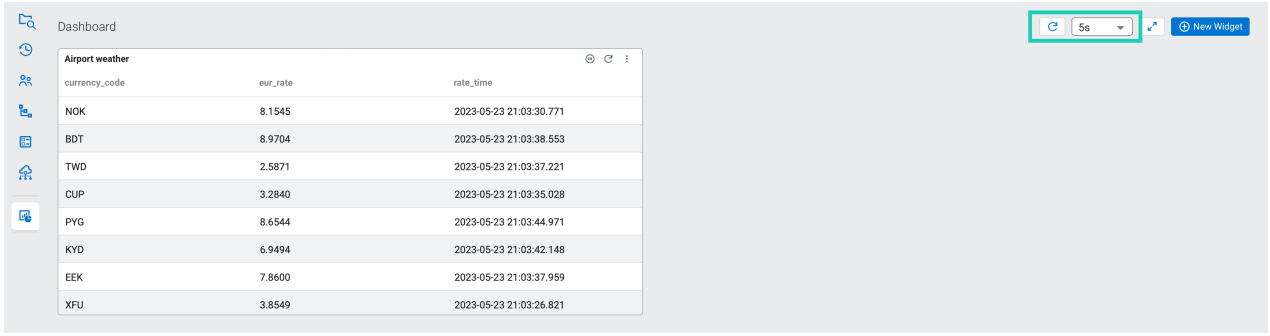
Query limit

The query limit sets the maximum amount of records to be returned for each call



Note: The data retention configuration that is provided when creating the Materialized View affects the results of this data source. For example, when setting the retention for row count, the specified number of values will be shown on the widget.

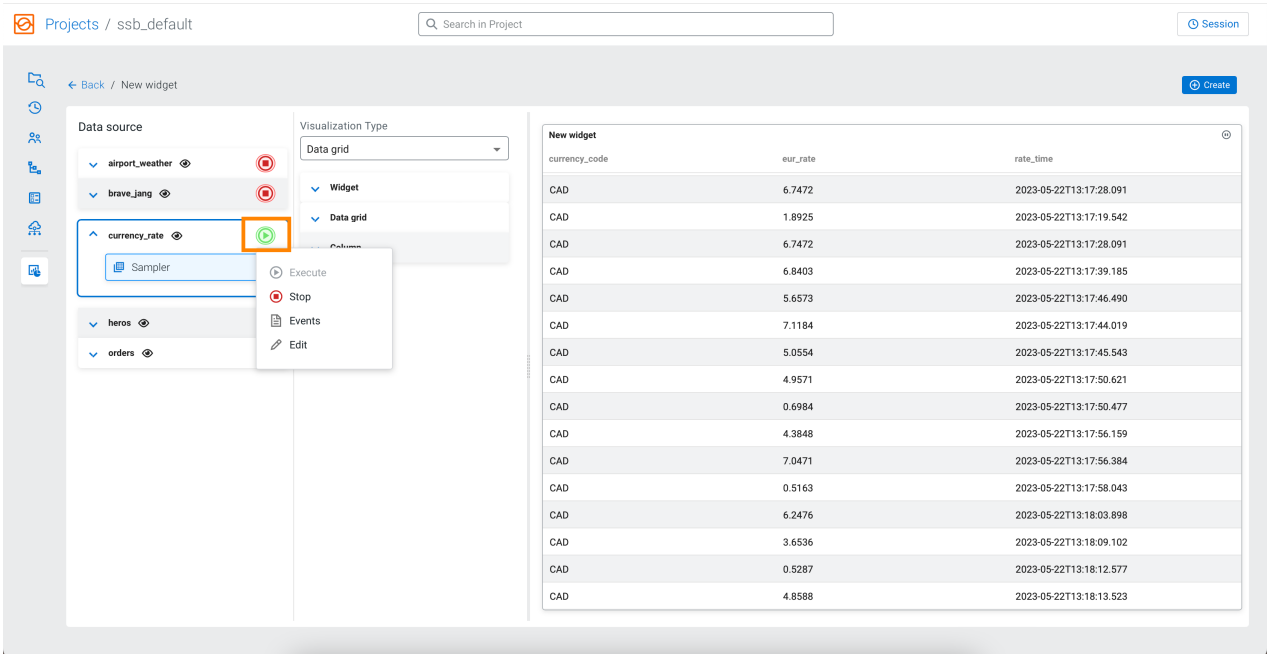
If the update interval is set to Default, the update interval can be specified on the **Dashboard** after the widget is created. The time selected on the **Dashboard** will be applied to every widget that has **Default** configured for update interval.



Managing data source jobs

The running jobs corresponding to the widgets can be manage from the Widget editor page.

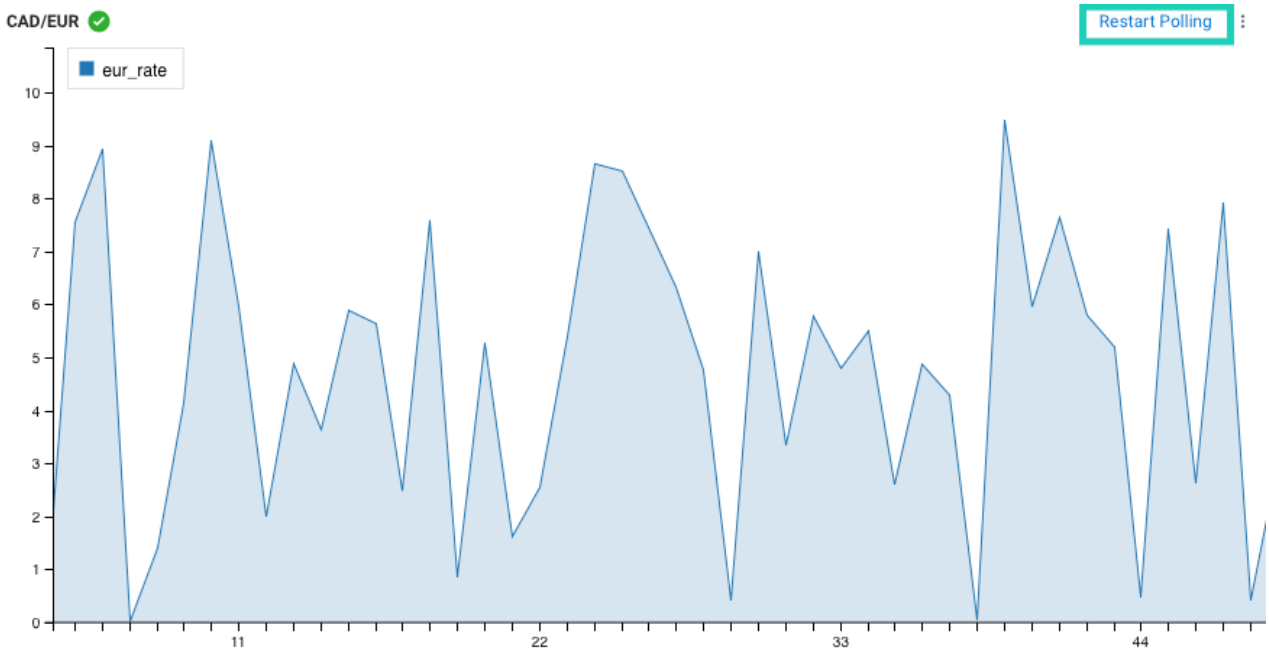
You can manage the jobs corresponding to the data sources from the widget editor page by clicking on the job status icon at the data source as shown in the following example:



You can Execute a stopped job when you want to use it for previewing the data on a visualization, and you can also Stop the running job if you no longer need it. When using sampling as a data source, you cannot select the sampler if the job is stopped. If a Materialized View endpoint is used as a data source, you can view the data on the visualization as long as the data retention time is configured in the Materialized View settings.

The Events option can be used to check the information, warning and error messages that correspond to the selected job.

By clicking on Edit, you are redirected to the SQL editor to where you can configure the job or change the SQL query of the job. In case you have created a widget and restart the corresponding job with changes, you need to restart the polling for the widget on the **Dashboard** as well.



Customizing visualization types

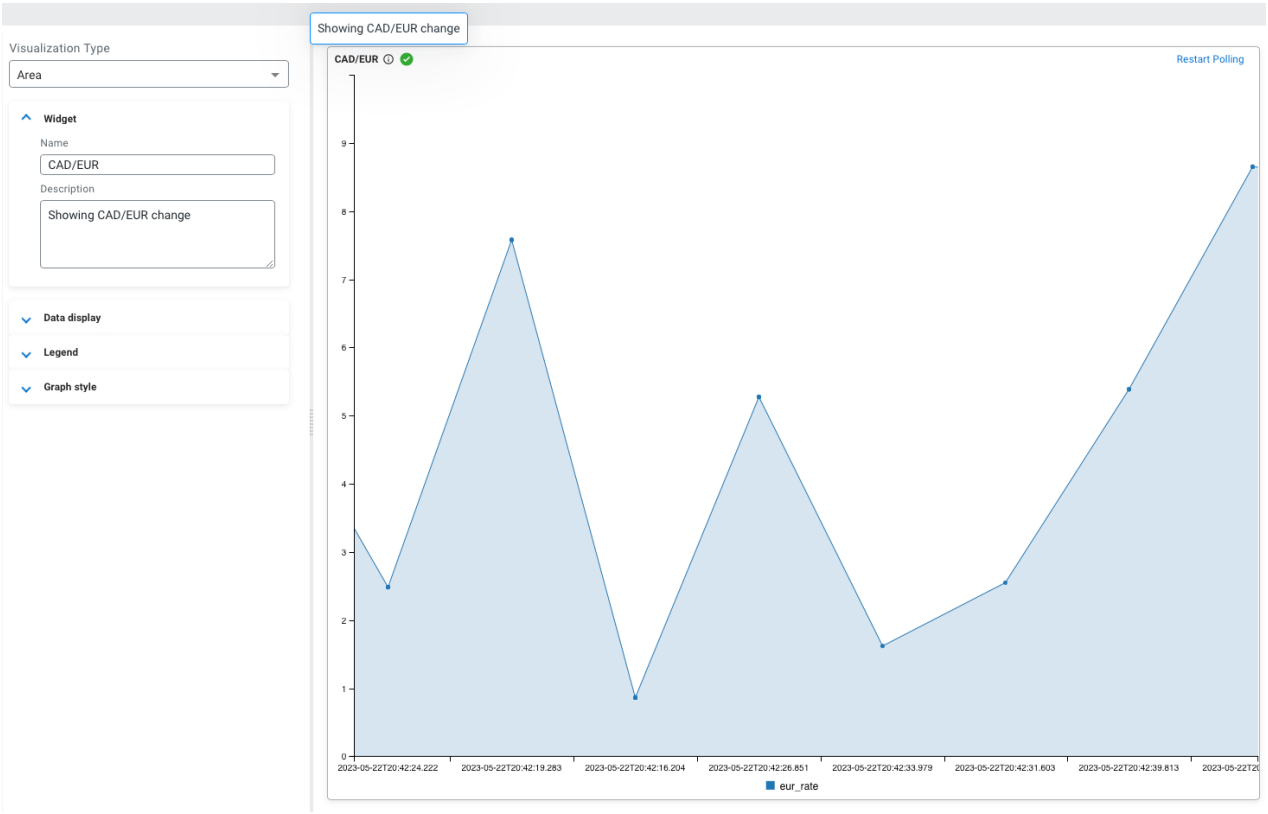
Widgets allow you to choose from a variety of visualization types and customize them based on your requirements. Each visualization type has general configurations and its own graph style configuration that can be set when creating the widget.

Configuring widget, fields and legend

The widget, data field, label field and legend are general configurations that can be set for all of the visualization types.

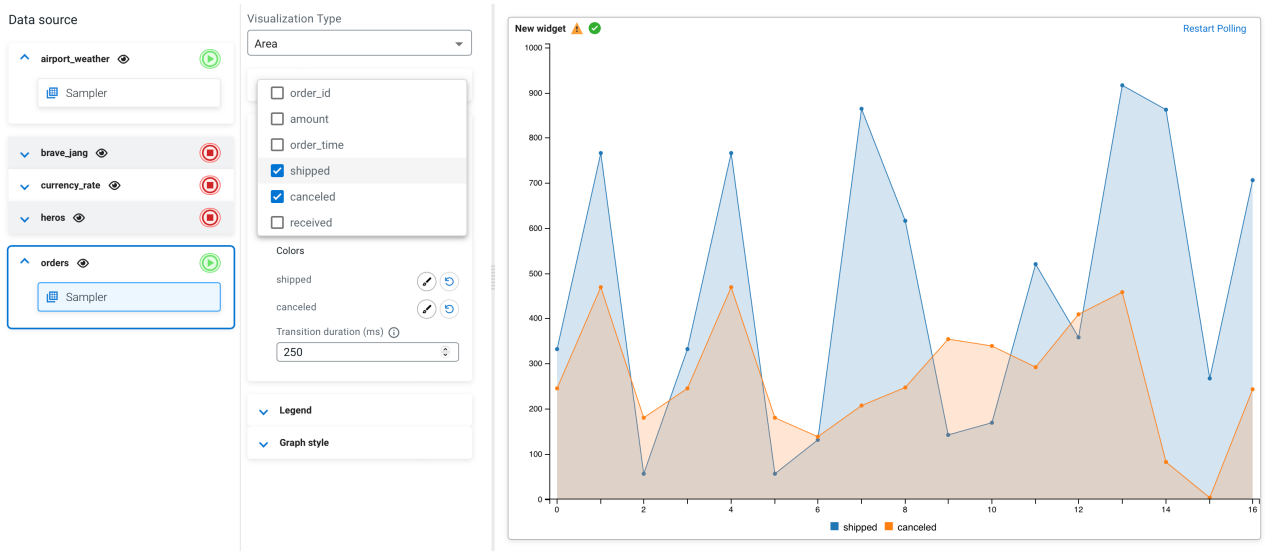
Name and Description

You can provide a custom name and short description about the widget. The name of the widget appears at the right top corner of the widget. The description can be viewed if it is provided by hovering over ⓘ next to the widget name.



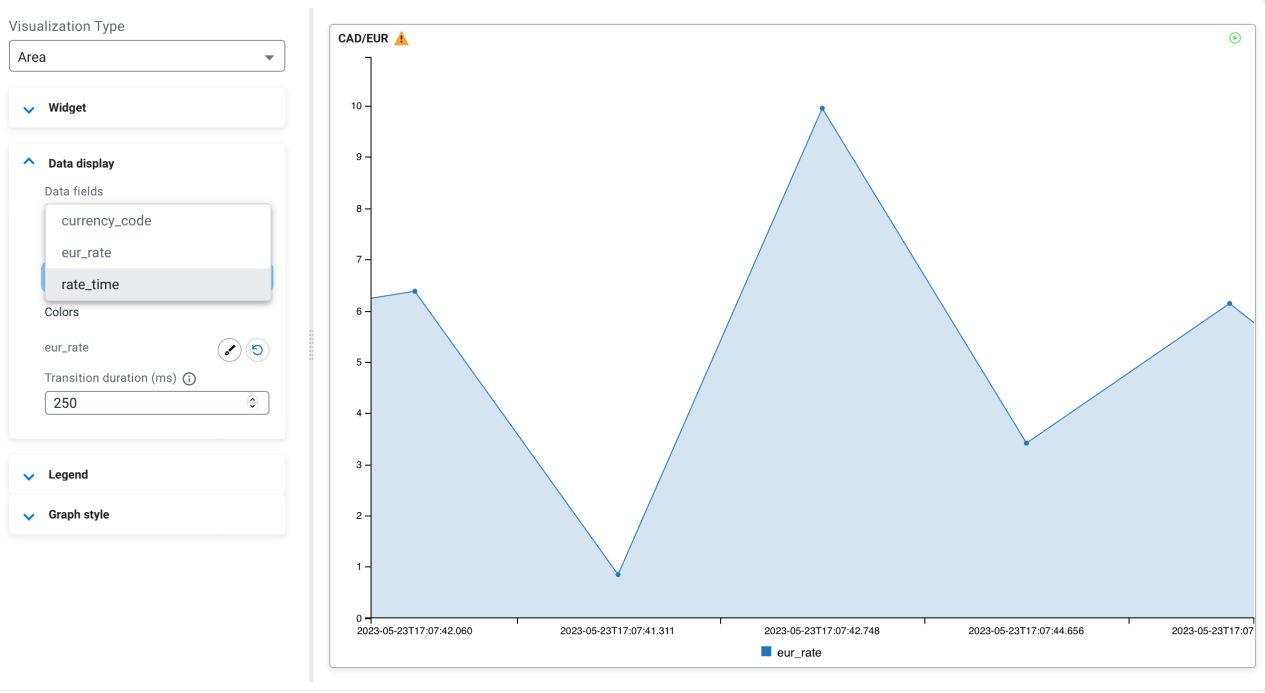
Data fields

You need to select the data fields to show on the axis of the visualization. The fields correspond to the columns of the table. You can select more than one data field to be displayed on the widget. The values of the data field are shown on the Y axis.



Label field

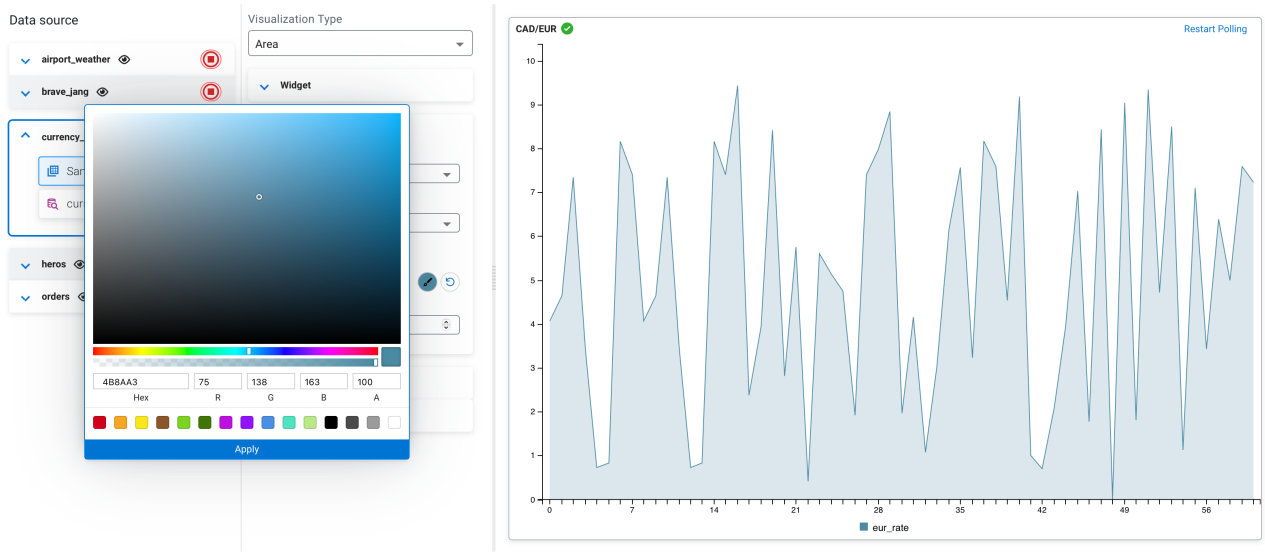
You have the option to select a field to show as labels. The fields correspond to the columns of the table. If the Label field is left empty, the number of records will be displayed as labels on the X axis.



Color picker



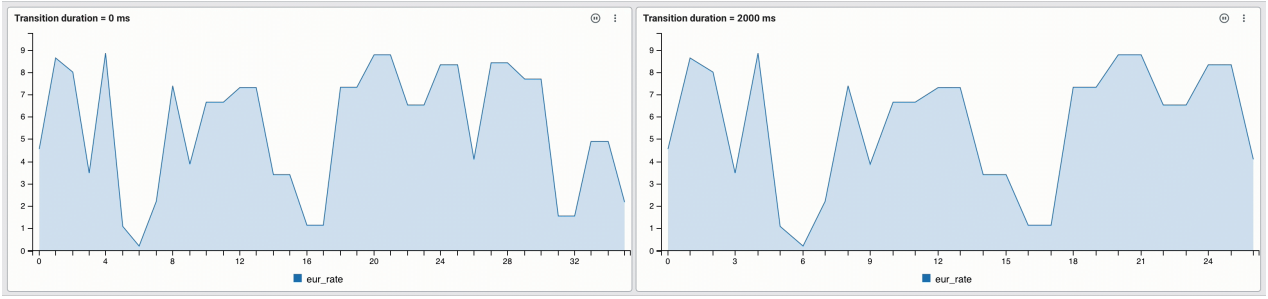
You can change the default visualization color of the selected data fields. You can open the color picker using button, and pick a color over the palette, or provide the Hex, RGBA value to a specific color. You can also choose from one of the preset colors. After clicking on Apply, the color of the selected data field is updated.



You can reset the color of the visualization using the button.

Transition duration (ms)

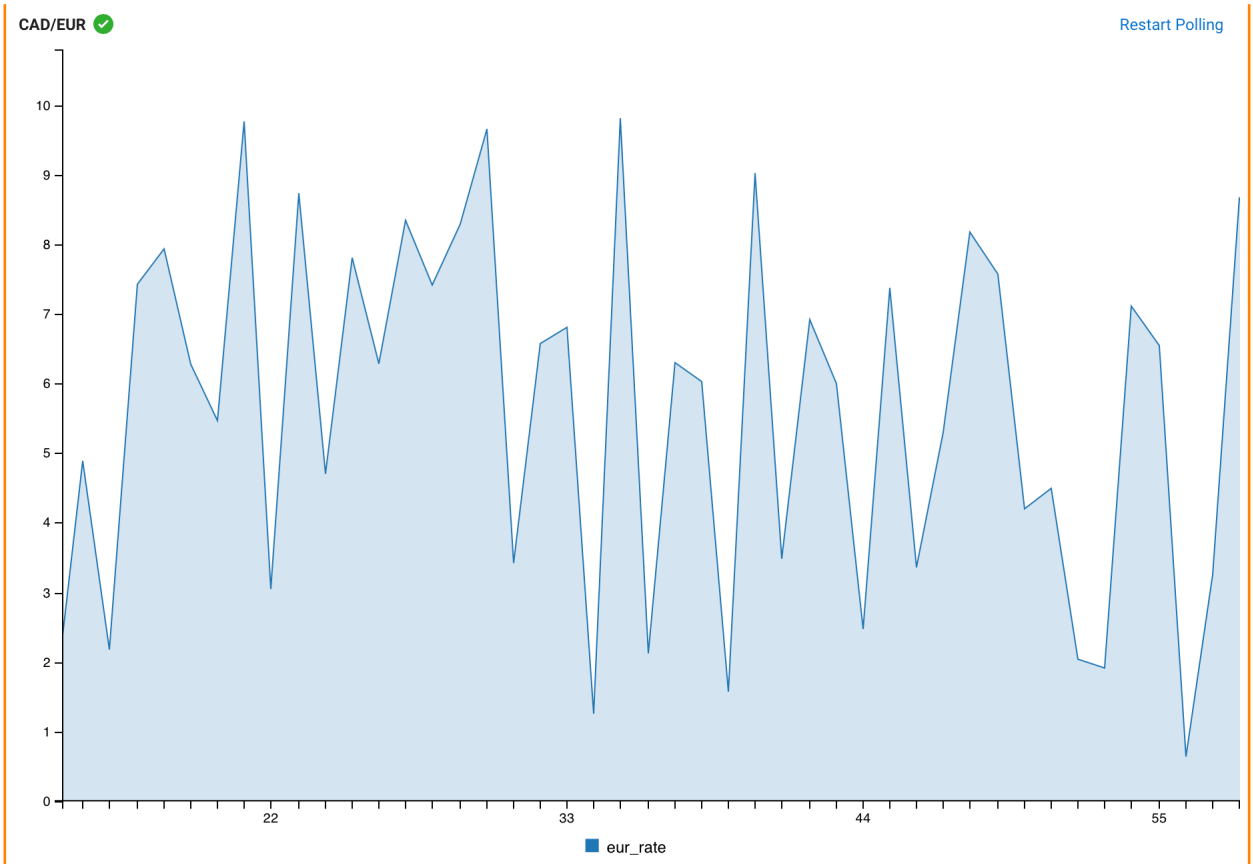
You can set the time for how slow or fast the data should change on the widget. The default is 250 ms. If set to 0 or left empty, the transition is skipped. The higher the time is set, the slower the visualization changes on the widget.



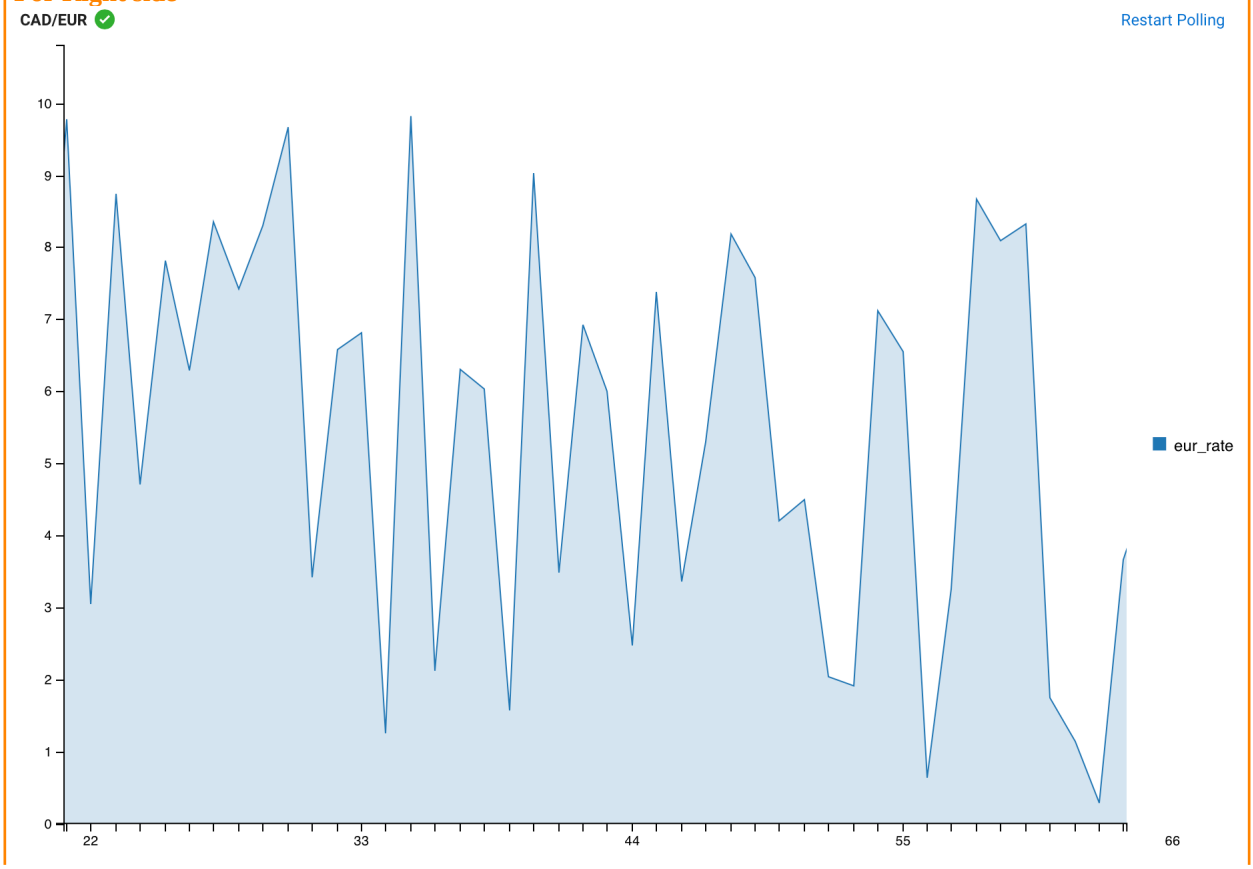
Visibility and position of Legend

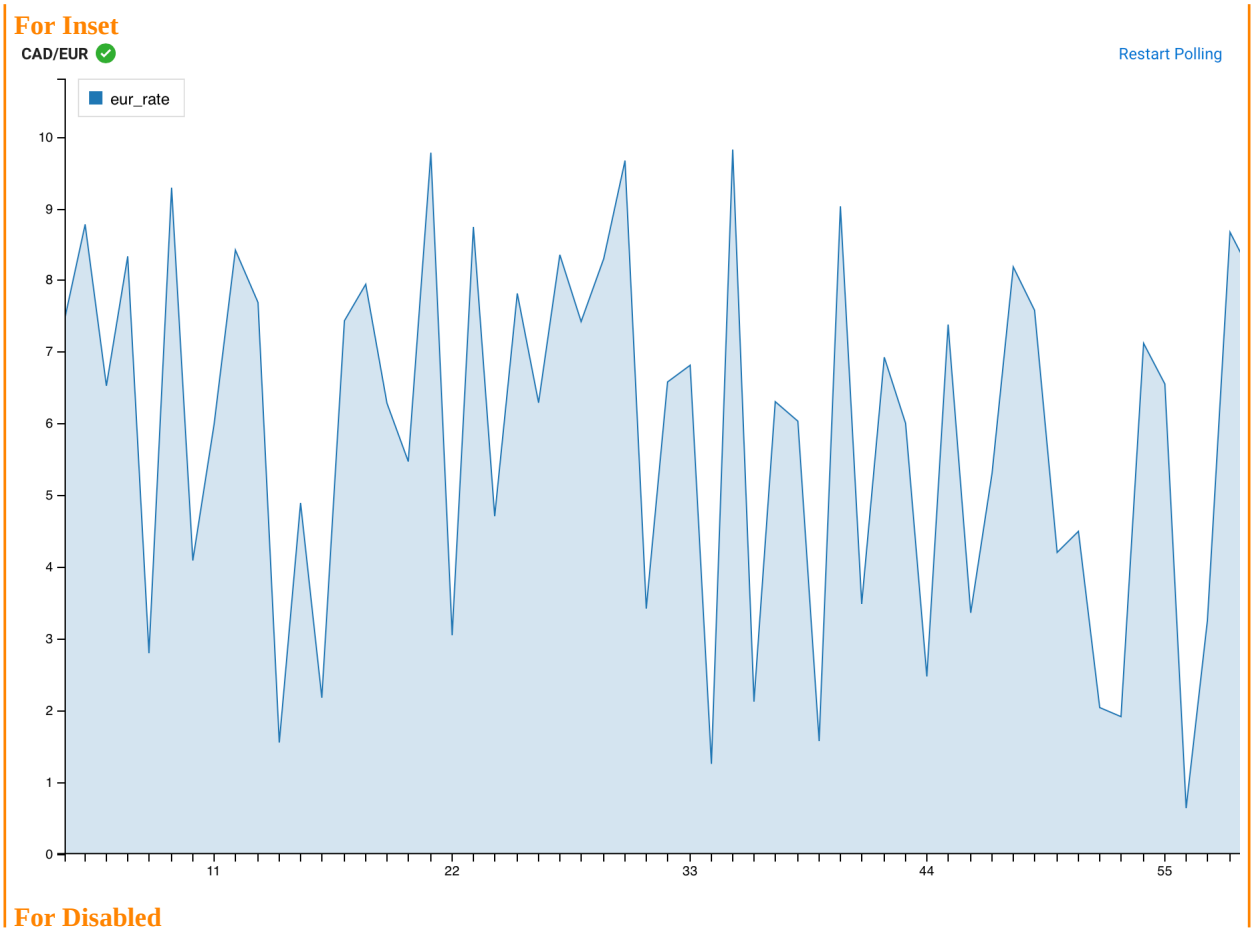
You can enable or disable the legend on the visualization, and also choose to position the legend to the bottom or right side of the widget, or place it in the visualization.

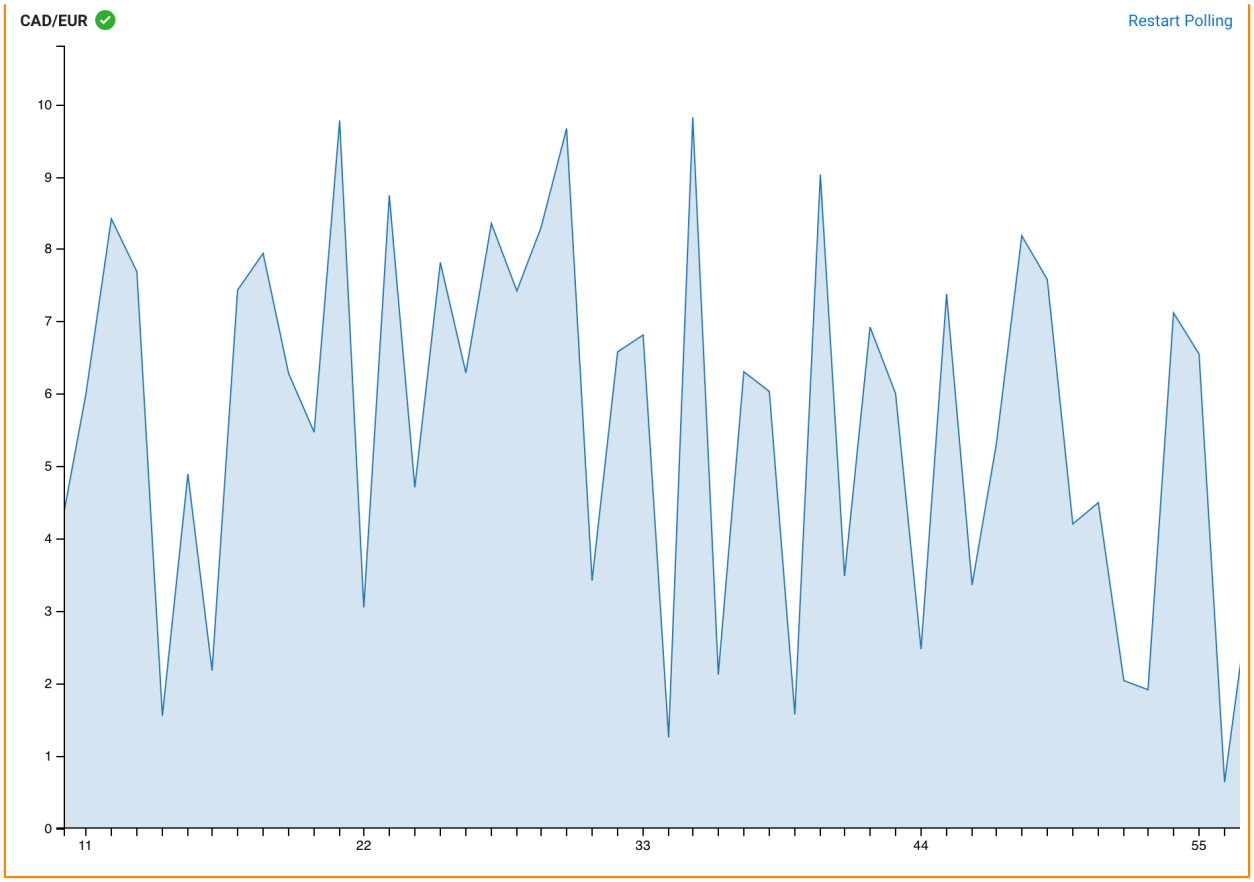
For Bottom



For Right side




For Inset
CAD/EUR
Restart Polling
eur_rate
10
9
8
7
6
5
4
3
2
1
0
11
22
33
44
55
For Disabled

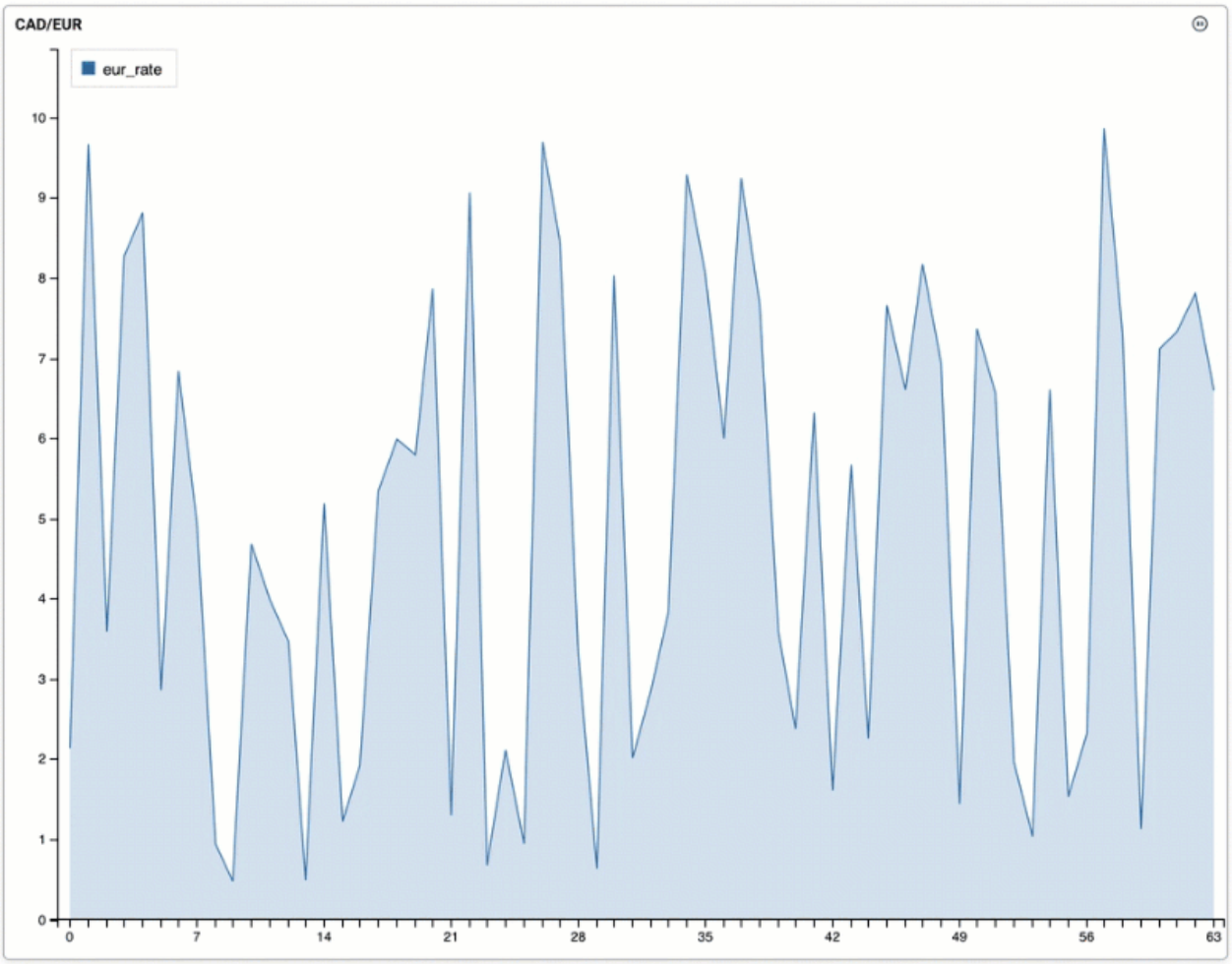


Configuring graph style

You can customize the styles of the widgets based on how you want to visualize the data.

Visualization type: Area

You can use the Area visualization type when you want to show how the values changed over a period of time.



The following table and query was used for the Area widget:

```
CREATE TABLE currency_rates (
  `currency_code` VARCHAR(2147483647),
  `eur_rate` DECIMAL(6, 4),
  `rate_time` TIMESTAMP(3)
) WITH (
  'fields.currency_code.expression' = '#{Currency.code}',
  'fields.rate_time.expression' = '#{date.past ''15'',''SECONDS''}',
  'connector' = 'faker',
  'rows-per-second' = '100',
  'fields.eur_rate.expression' = '#{Number.randomDouble ''4'',''0'',''10''}'
);

SELECT * FROM currency_rates
WHERE currency_code = 'CAD'
```

The following configurations have been set for the Area widget:

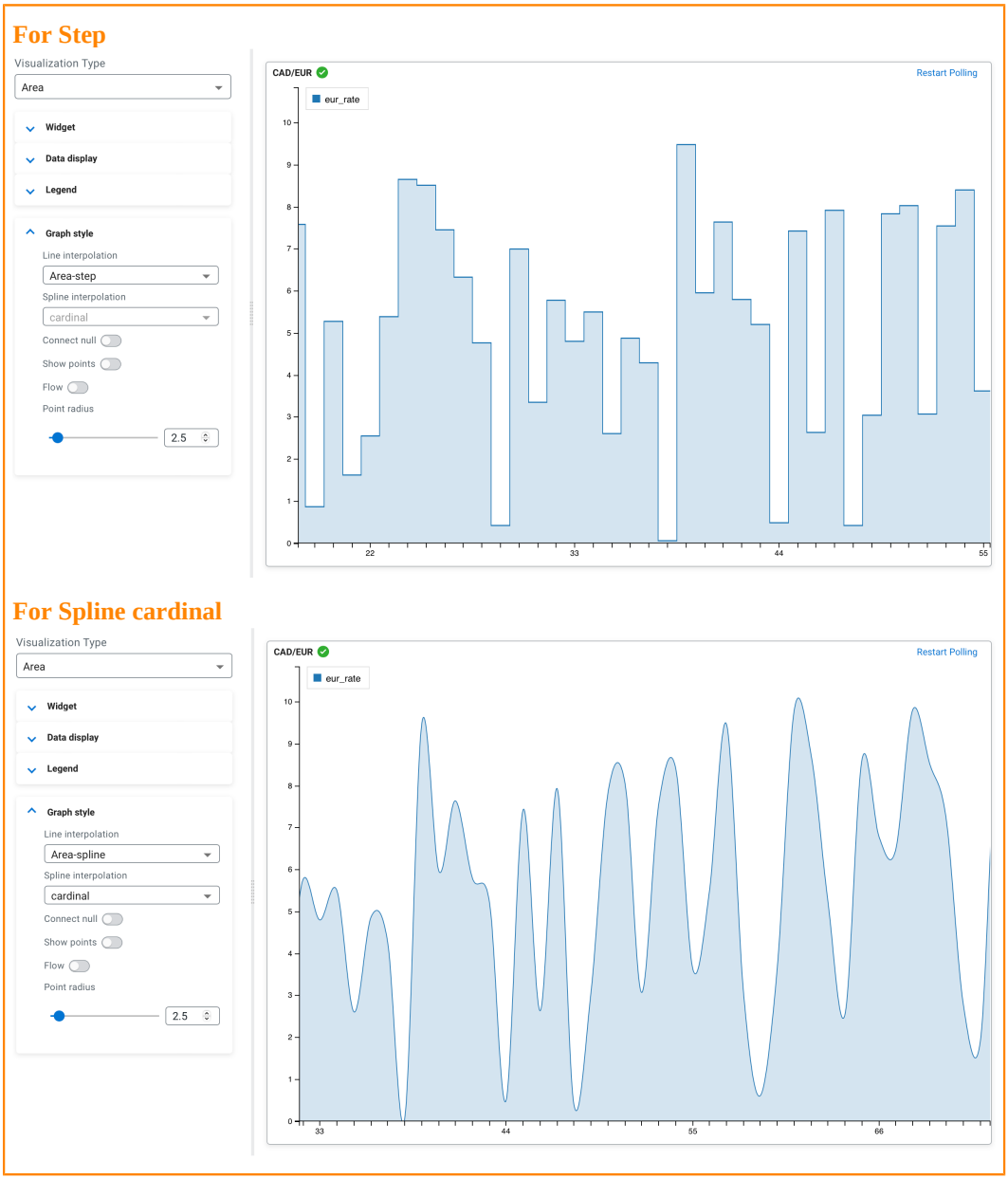
Name	CAD/EUR
Data fields	eur_rate
Legend poition	Inset
Line interpolation	Area
Connect null	enabled

Show points	disabled

You can further customize the graph style of the Area widget with the following Graph style configurations:

Line interpolation

Determines the line interpolation of the visualization. You have the option to select Area-spline and Area-step for line interpolation.

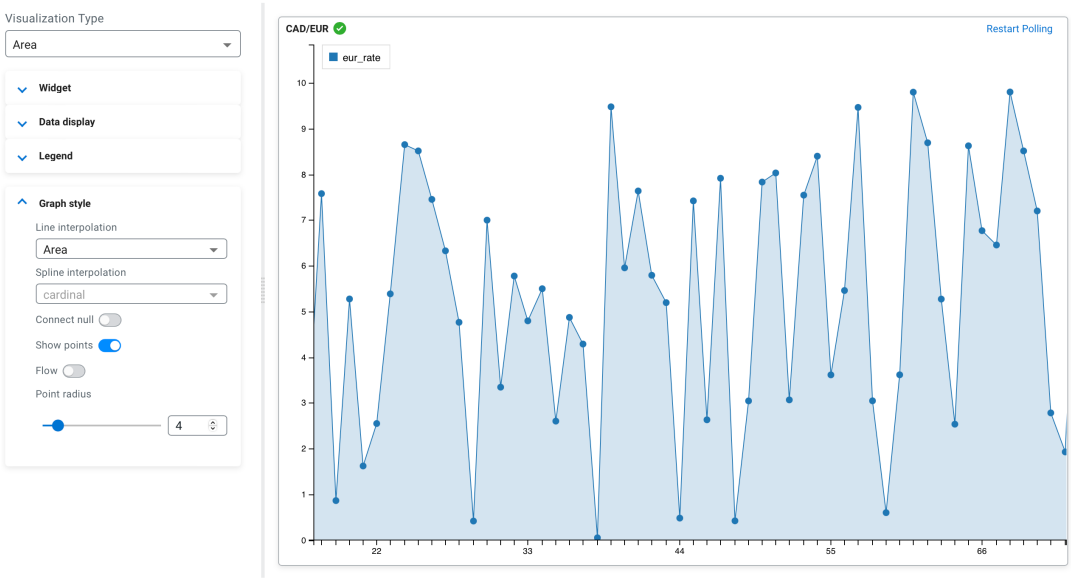


When selecting Area-spline as interpolation, you can also specify the Spline interpolation which includes the following options:

- Linear, linear-closed
- Basis, basis-open, basis-closed
- Cardinal, cardinal-open, cardinal-closed
- Monotone
- Step, step-before, step-after

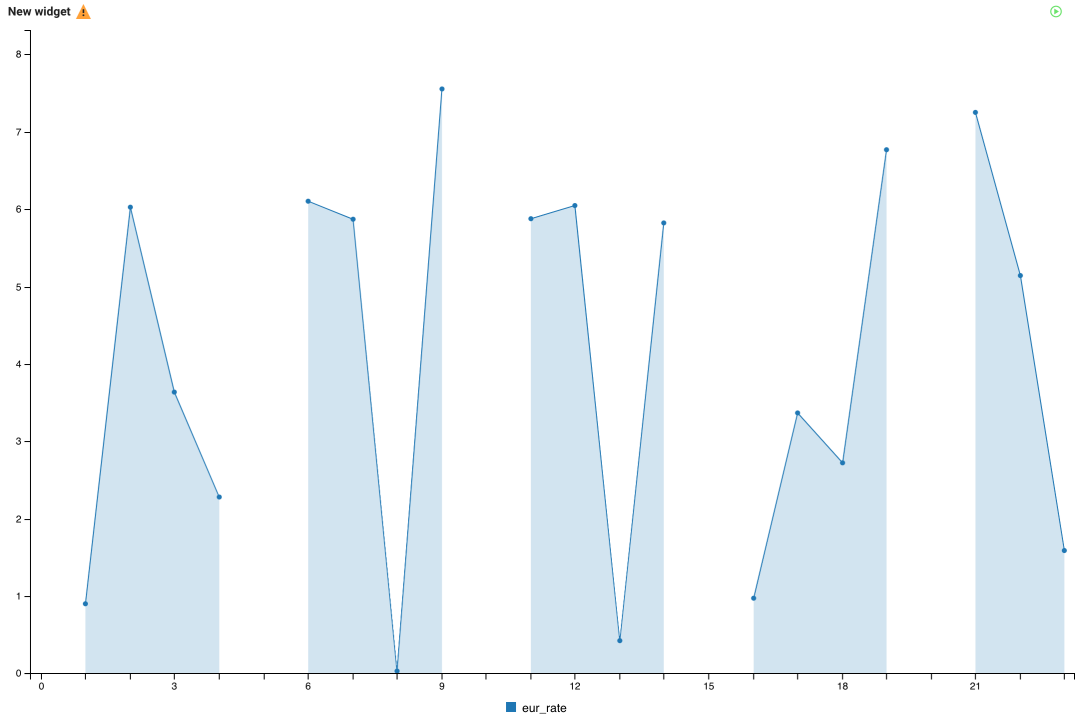
Show points and Point radius

You can enable whether to show each data point in line using the Show points toggle. The size of the points can be changed using the Point radius configuration, which is set to 2.5 by default. The following example shows the enabled Show points with 4.0 Point radius:



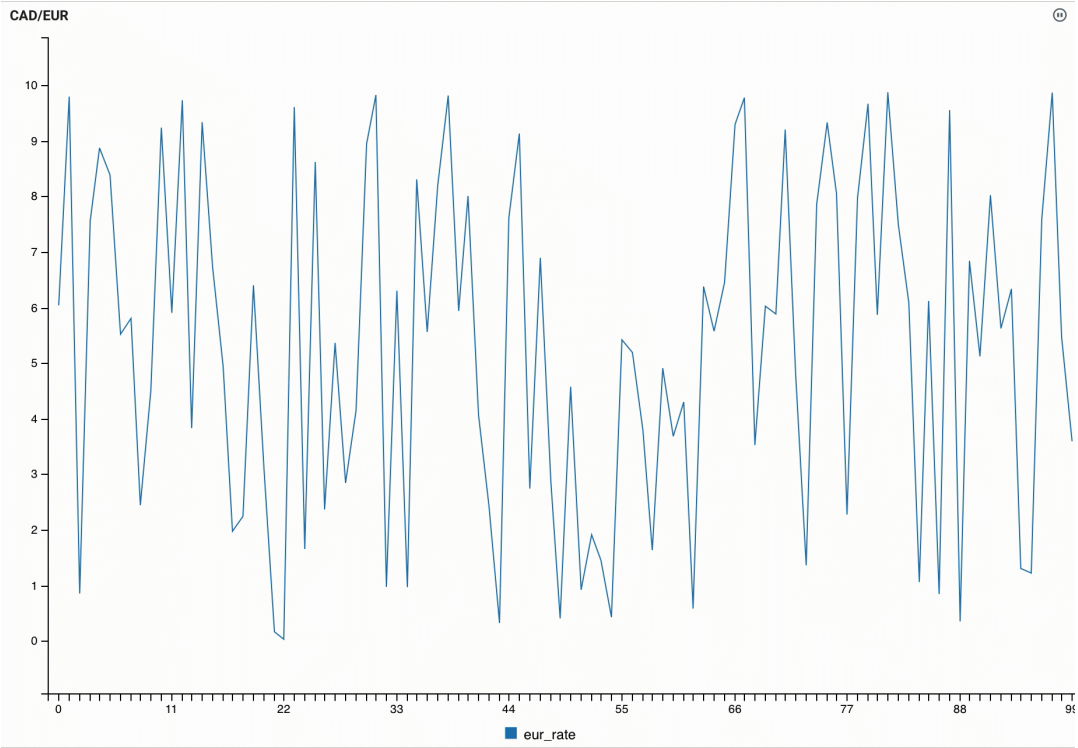
Connect null

Using the connect null configuration, you can set if the null data points are connected or not. If enabled, the region of the null data is connected without any data point. When disabled, the data points are not connected as shown in the following example:



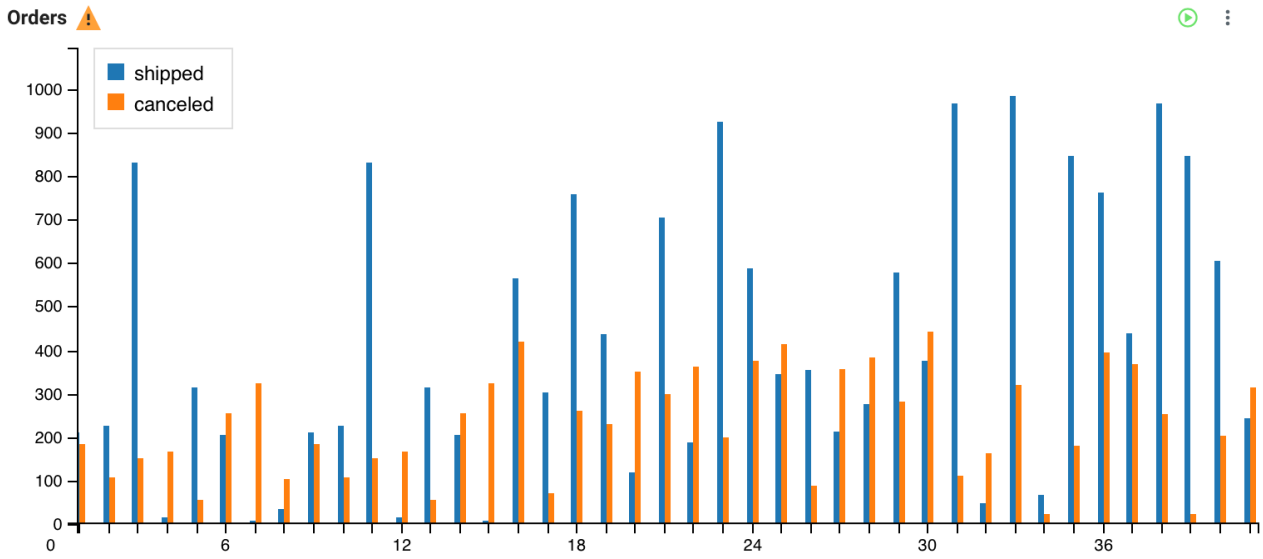
Flow

You can set the movement of the data by enabling or disabling the Flow function, which means the new data points flow to the chart from the right side compared to arriving periodically as processed on the widget.



Visualization type: Bar

You can use the Bar visualization type when you want to show how the data is distributed or want to compare the data between different groups.



The following table and query was used for the Bar widget:

```
CREATE TABLE orders (
  `order_id` INT,
  `amount` INT,
  `order_time` TIMESTAMP(3),
  `shipped` VARCHAR(2147483647),
  `canceled` VARCHAR(2147483647),
  `received` VARCHAR(2147483647),
  WATERMARK FOR `order_time` AS `order_time` - INTERVAL '15' SECOND
) WITH (
```

```
  'fields.order_time.expression' = '#{date.past ''15'',''SECONDS''}',
  'fields.amount.expression' = '#{number.numberBetween ''0'',''100''}',
  'fields.order_id.expression' = '#{number.numberBetween ''0'',''99999999''
}',
  'connector' = 'faker',
  'fields.shipped.expression' = '#{number.numberBetween ''0'',''1000''}',
  'fields.received.expression' = '#{number.numberBetween ''0'',''800''}',
  'fields.canceled.expression' = '#{number.numberBetween ''0'',''500''}',
  'rows-per-second' = '100'
);

SELECT * FROM orders
```

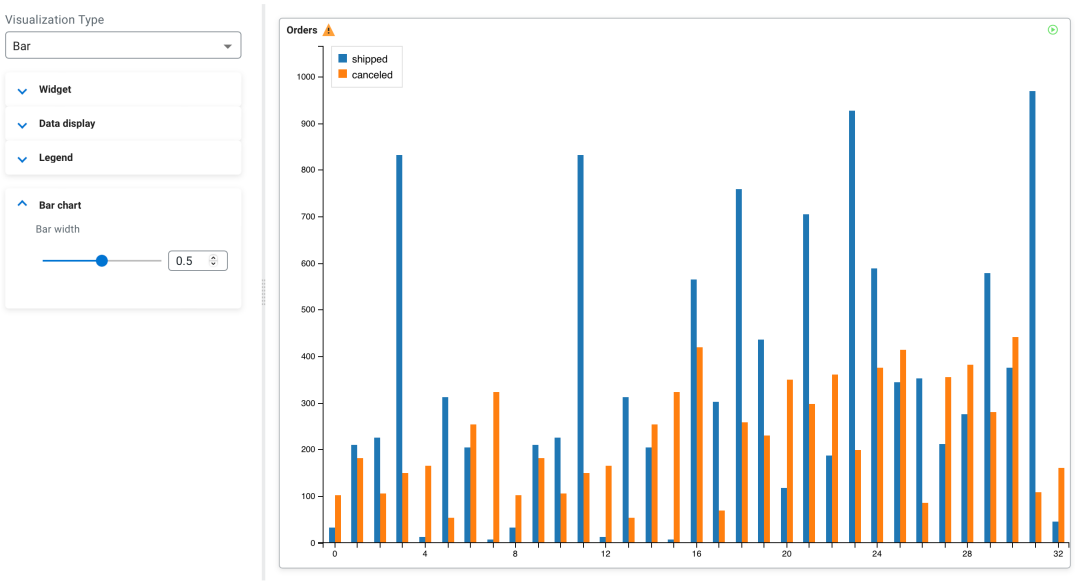
The following configurations have been set for the Bar widget:

Name	Orders
Data fields	shipped, canceled
Legend poition	Inset
Bar width	0.5

You can further customize the graph style of the Bar widget with the following Graph style configurations:

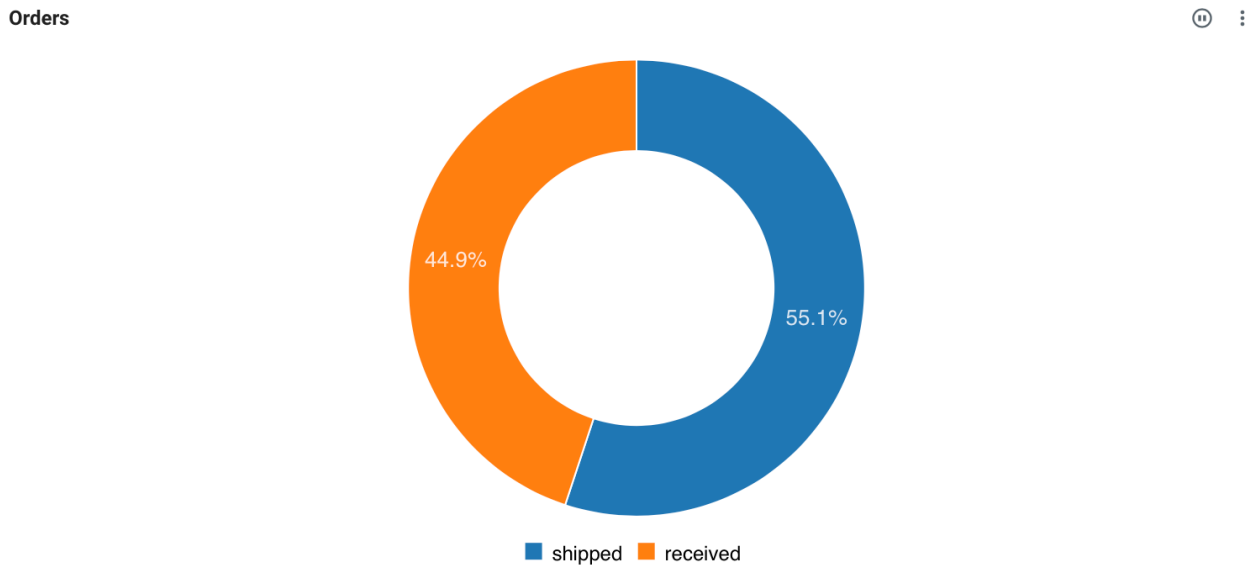
Bar width

You can specify the width of the bars on the widget by changing the default value.



Visualization type: Donut

You can use the Donut visualization type when you want to show the proportions of data within a category where the size of the pieces represent the proportion of each category.



The following table and query was used for the Donut widget:

```
CREATE TABLE orders (
  `order_id` INT,
  `amount` INT,
  `order_time` TIMESTAMP(3),
  `shipped` VARCHAR(2147483647),
  `canceled` VARCHAR(2147483647),
  `received` VARCHAR(2147483647),
  WATERMARK FOR `order_time` AS `order_time` - INTERVAL '15' SECOND
) WITH (
  'fields.order_time.expression' = '#{date.past ''15'',''SECONDS''}',
  'fields.amount.expression' = '#{number.numberBetween ''0'',''100''}',
  'fields.order_id.expression' = '#{number.numberBetween ''0'',''99999999''
}',
  'connector' = 'faker',
  'fields.shipped.expression' = '#{number.numberBetween ''0'',''1000''}',
  'fields.received.expression' = '#{number.numberBetween ''0'',''800''}',
  'fields.canceled.expression' = '#{number.numberBetween ''0'',''500''}',
  'rows-per-second' = '100'
);

SELECT * FROM orders
```

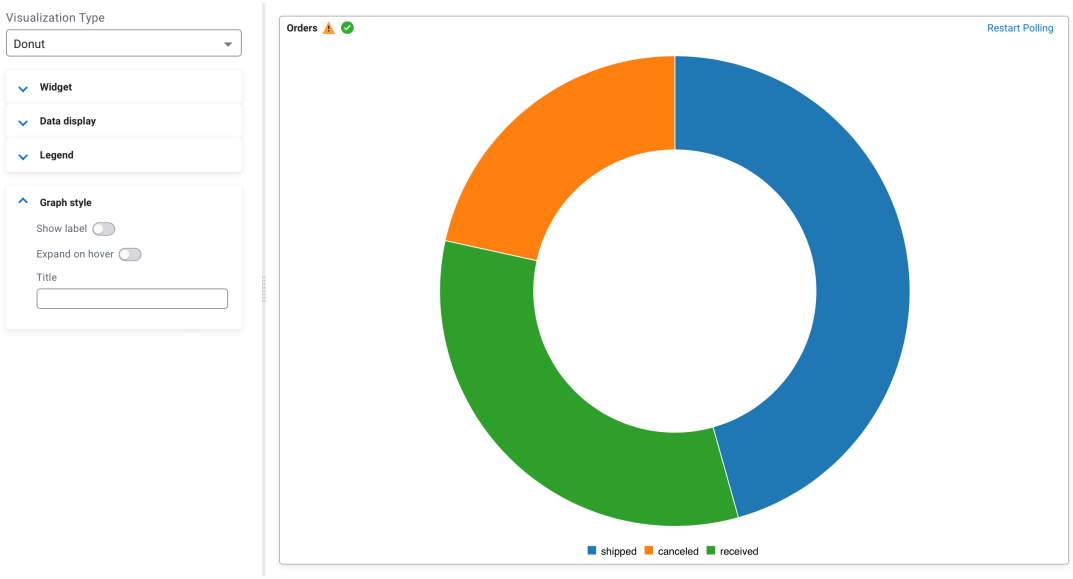
The following configurations have been set for the Donut widget:

Name	Orders
Data fields	shipped, received
Legend poition	Bottom
Show label	enabled

You can further customize the graph style of the Donut widget with the following Graph style configurations:

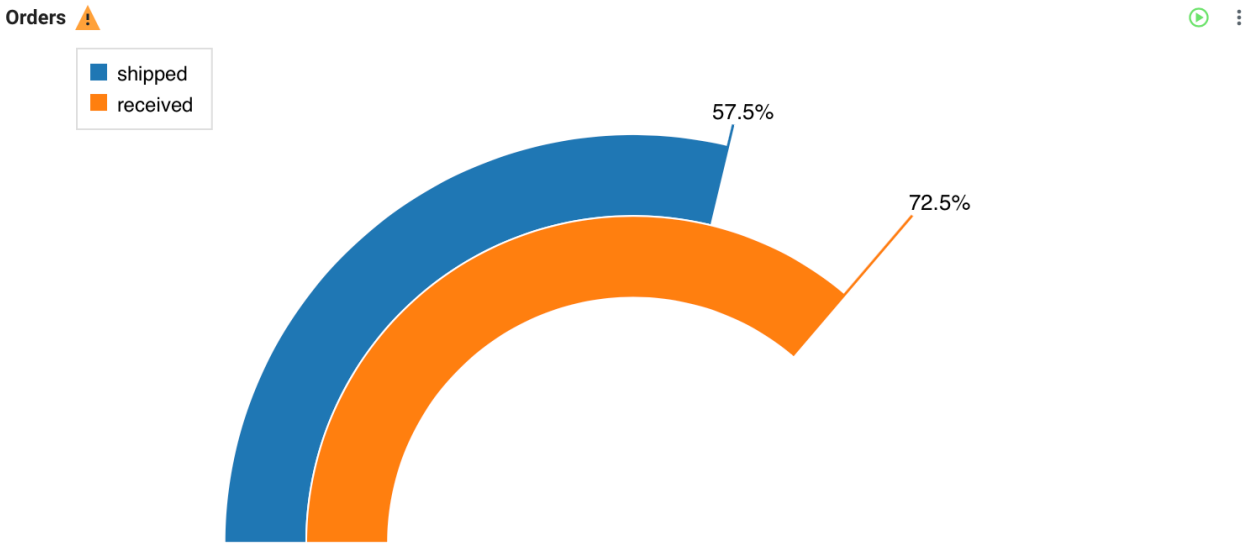
Show label

You can enable or disable the labels on the visualization as shown in the following illustration where labels are disabled:



Visualization type: Gauge

You can use the Gauge visualization type when you want to show progress or amount compared to the maximum value.



The following table and query was used for the Gauge widget:

```
CREATE TABLE orders (
  `order_id` INT,
  `amount` INT,
  `order_time` TIMESTAMP(3),
  `shipped` VARCHAR(2147483647),
  `canceled` VARCHAR(2147483647),
  `received` VARCHAR(2147483647),
  WATERMARK FOR `order_time` AS `order_time` - INTERVAL '15' SECOND
) WITH (
  'fields.order_time.expression' = '#{date.past ''15'',''SECONDS''}',
  'fields.amount.expression' = '#{number.numberBetween ''0'',''100''}',
  'fields.order_id.expression' = '#{number.numberBetween ''0'',''99999999''
}',
  'connector' = 'faker',
  'fields.shipped.expression' = '#{number.numberBetween ''0'',''1000''}',
```

```
  'fields.received.expression' = '#{number.numberBetween ''0'',''800''}',
  'fields.canceled.expression' = '#{number.numberBetween ''0'',''500''}',
  'rows-per-second' = '100'
);

SELECT * FROM orders
```

The following configurations have been set for the Gauge widget:

Name	Orders
Data fields	shipped, received
Legend poition	Inset
Reducer	Last
Show label	enabled
Min	100
Max	300

You can further customize the graph style of the Gauge widget with the following Graph style configurations:

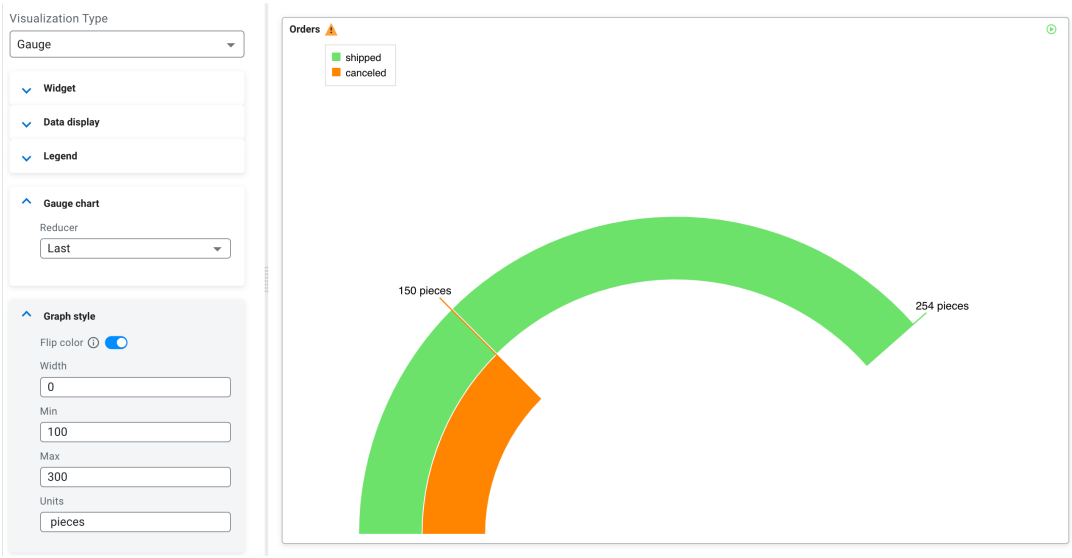
Reducer

A reducer function that will be used to reduce many fields to a single value. You have the following options to choose from:

- Last
- First
- Min
- Max
- Total
- All

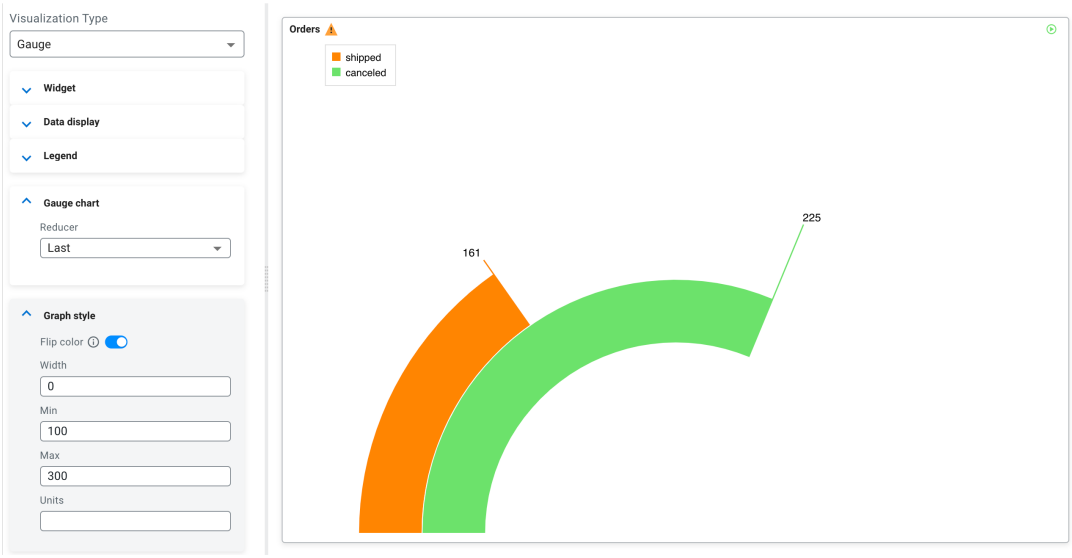
Min, Max

You can specify the minimum and maximum value of the gauge chart.



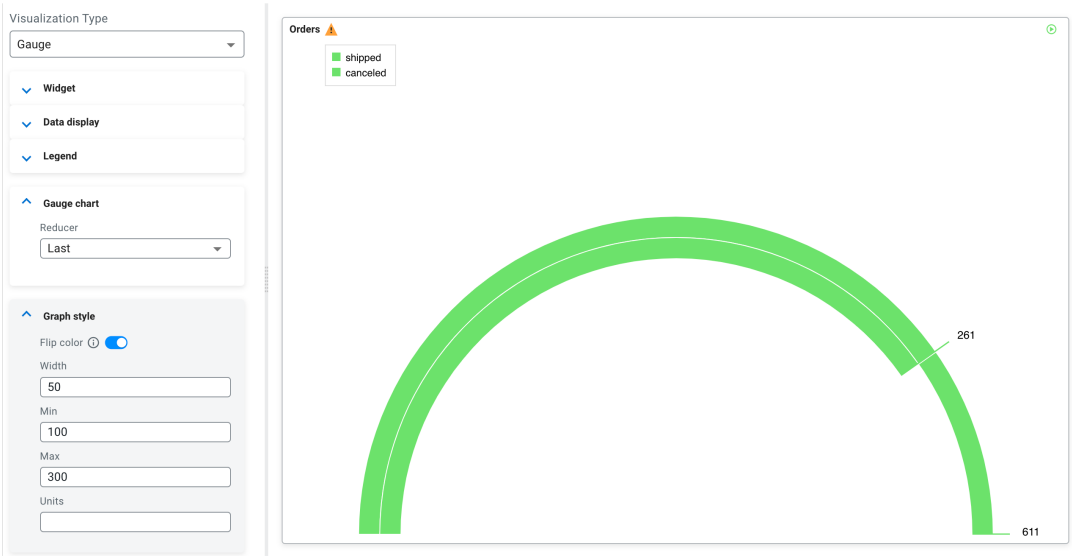
Flip color

You can enable color switching based on the values set for minimum and maximum. When the values are larger than 75% of Max then the color of the widget changes to green. When the values are smaller than 25% of Min then the color of the widget changes to red.



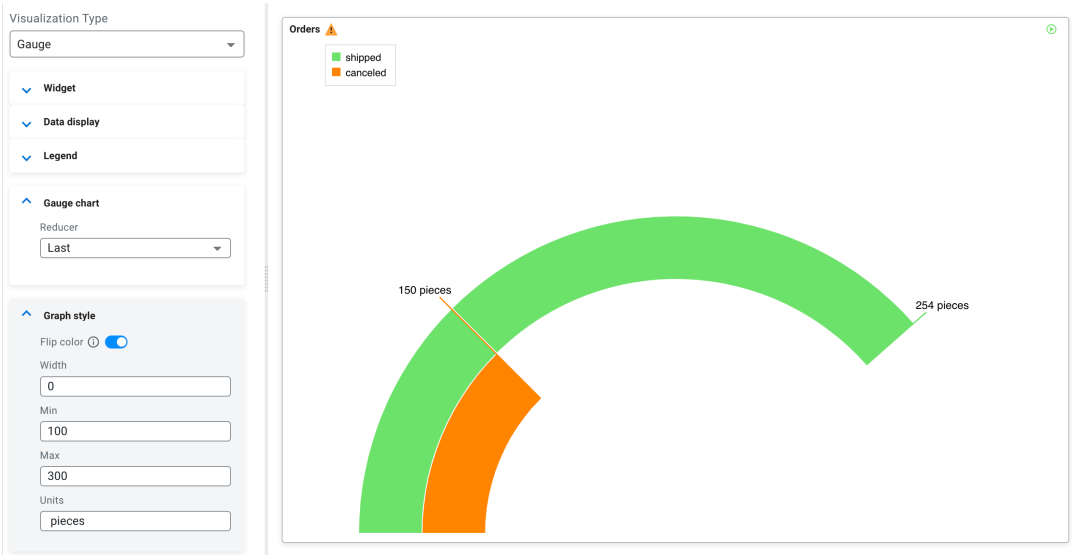
Width

You can increase or decrease the width of the gauges on the widget.



Units

You can specify a unit for the displayed values on the widget.



Visualization type: Data grid

You can use the Data grid visualization type when you want to show data in a table format.

Airport weather Restart Polling

airport	temperature	visibility	eventTimestamp
FAPA	-7°C	191m	2023-05-22T22:41:36.088
FAPA	-7°C	191m	2023-05-22T22:41:36.088
SAAG	-23°C	876m	2023-05-22T22:41:49.908
SAAG	-23°C	876m	2023-05-22T22:41:49.908
DIBU	29°C	766m	2023-05-22T22:41:44.142
DIBU	29°C	766m	2023-05-22T22:41:44.142
AGAT	-6°C	783m	2023-05-22T22:41:48.407
AGAT	-6°C	783m	2023-05-22T22:41:48.407
USRR	-3°C	131m	2023-05-22T22:41:38.508
USRR	-3°C	131m	2023-05-22T22:41:38.508

91 to 100 of 100 Page 10 of 10

The following table and query was used for the Data grid widget:

```
CREATE TABLE airport_weather (
  `airport` VARCHAR(2147483647) NOT NULL,
  `temperature` VARCHAR(2147483647) NOT NULL,
  `visibility` VARCHAR(2147483647) NOT NULL,
  `eventTimestamp` TIMESTAMP(3),
  WATERMARK FOR `eventTimestamp` AS `eventTimestamp` - INTERVAL '30' SECOND,
  CONSTRAINT `PK_-991666966` PRIMARY KEY (`airport`) NOT ENFORCED
) WITH (
  'fields.eventTimestamp.expression' = '#{date.past ''30'',''SECONDS''}',
  'connector' = 'faker',
  'fields.temperature.expression' = '#{Weather.temperatureCelsius}',
  'fields.airport.expression' = '#{Aviation.airport}',
```

```
  'rows-per-second' = '100',
  'fields.visibility.expression' = '#{numerify ''###m''}'
);

SELECT * FROM airport_weather
```

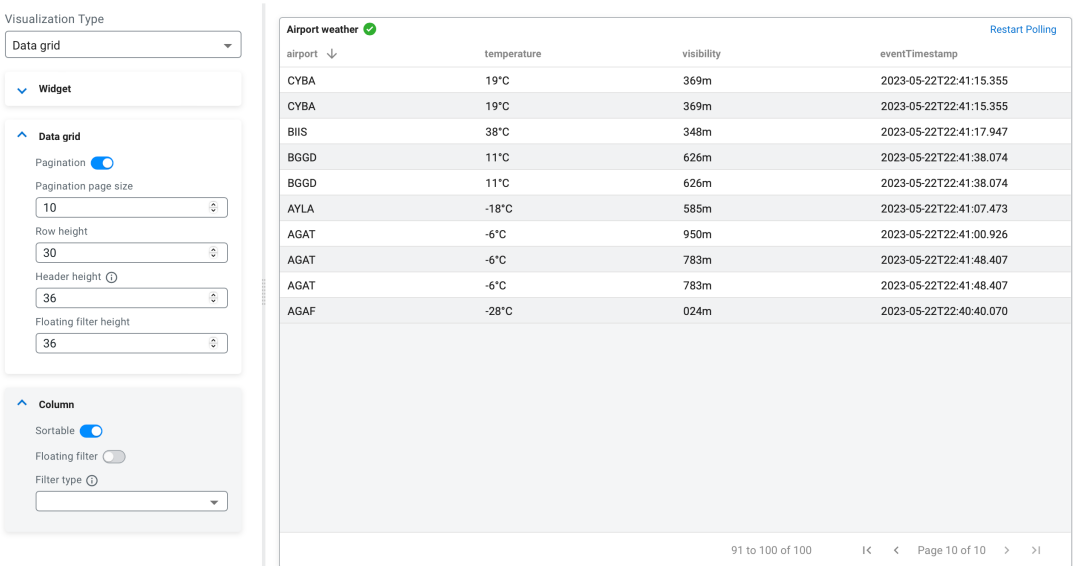
The following configurations have been set for the Data grid widget:

Name	Airport weather
Pagination	enabled
Page size	10
Row height	30
Header height	36
Sortable	disabled

You can further customize the graph style of the Data grid and Column configurations:

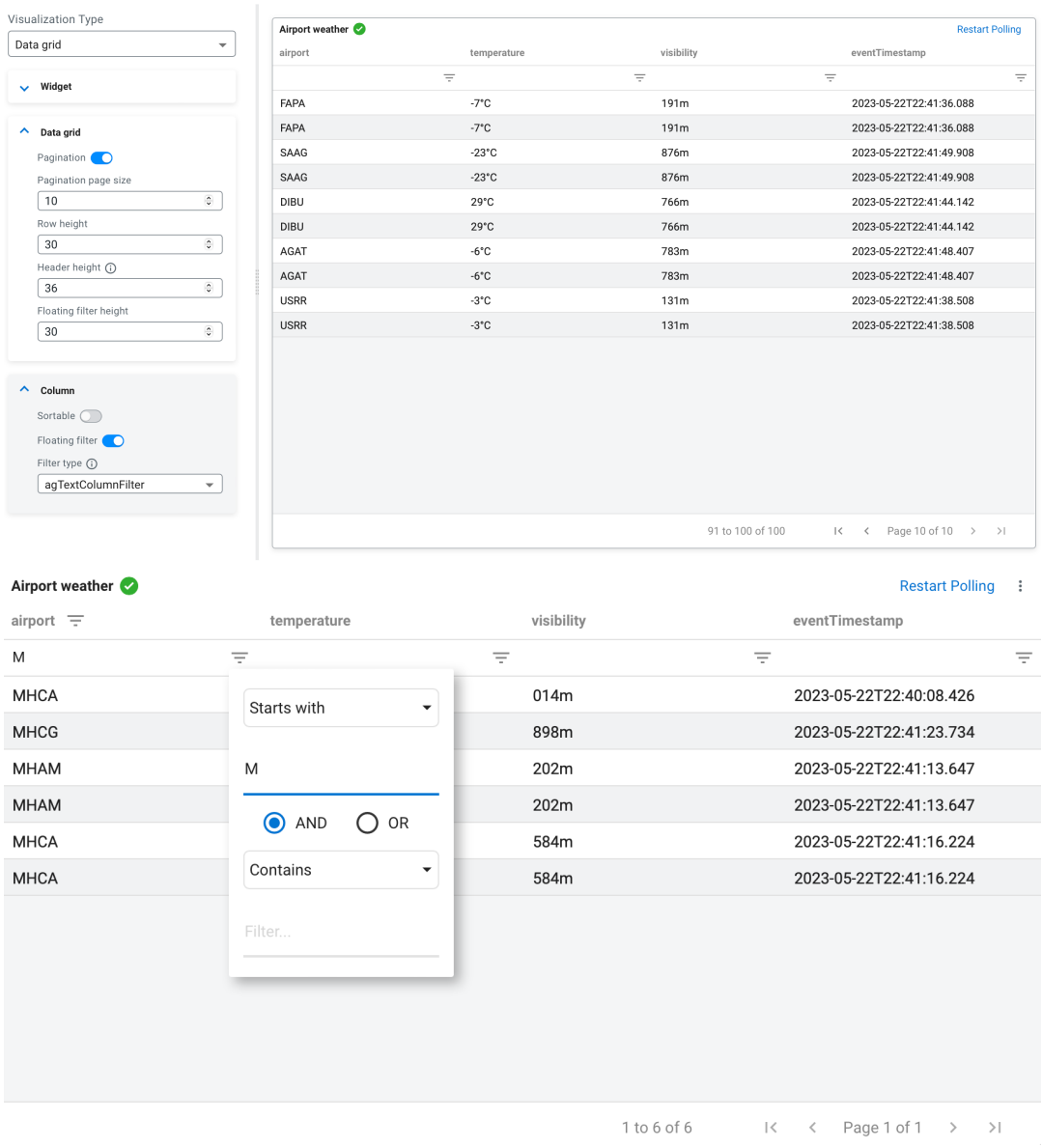
Pagination and pagination page size

You can enable and disable the pagination of the data on the visualization where the page size provides the number of rows displayed.



Floating filter and floating filter height:

You can add floating filter to the data grid which enables text, number or date based filtering of the columns. For example, you can set the floating filter to text and filter the data to start with M:



Sortable

You can enable or disable sortable columns for the data grid. You can sort the data in the filtered column by clicking on the column name so the values are in increasing or decreasing order.

Airport weather

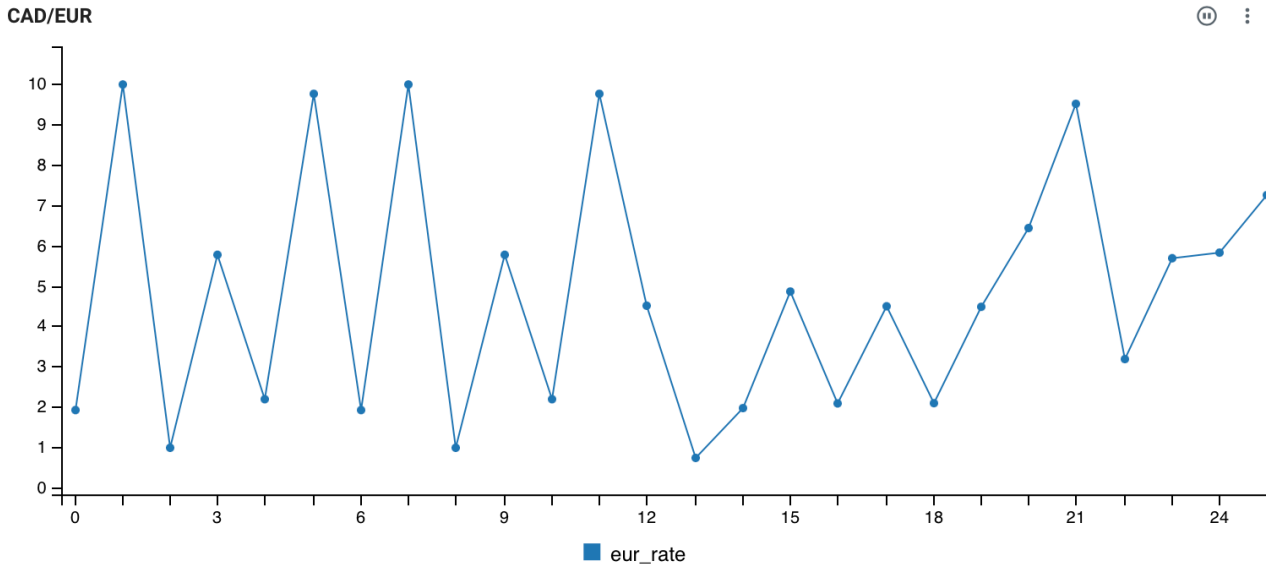
Restart Polling

airport	temperature	visibility	eventTimestamp
NFCI	36°C	865m	2023-05-22T22:41:15.471
SGAS	37°C	379m	2023-05-22T22:40:04.549
DAAD	37°C	219m	2023-05-22T22:40:58.494
BIIS	38°C	348m	2023-05-22T22:41:17.947
FAMW	5°C	374m	2023-05-22T22:41:26.716
FAMW	5°C	374m	2023-05-22T22:41:26.716
GANR	5°C	130m	2023-05-22T22:41:28.521
GANR	5°C	130m	2023-05-22T22:41:28.521
GCFV	7°C	071m	2023-05-22T22:40:31.594
USRR	9°C	951m	2023-05-22T22:39:57.255

91 to 100 of 100 Page 10 of 10

Visualization type: Line

You can use the Line visualization type when you want to show how the values changed over a period of time



The following table and query was used for the Line widget:

```
CREATE TABLE currency_rates (
  `currency_code` VARCHAR(2147483647),
  `eur_rate` DECIMAL(6, 4),
  `rate_time` TIMESTAMP(3)
) WITH (
  'fields.currency_code.expression' = '#{Currency.code}',
  'fields.rate_time.expression' = '#{date.past ''15'',''SECONDS''}',
  'connector' = 'faker',
  'rows-per-second' = '100',
  'fields.eur_rate.expression' = '#{Number.randomDouble ''4'',''0'',''10''}'
);

SELECT * FROM currency_rates
WHERE currency_code = 'CAD'
```

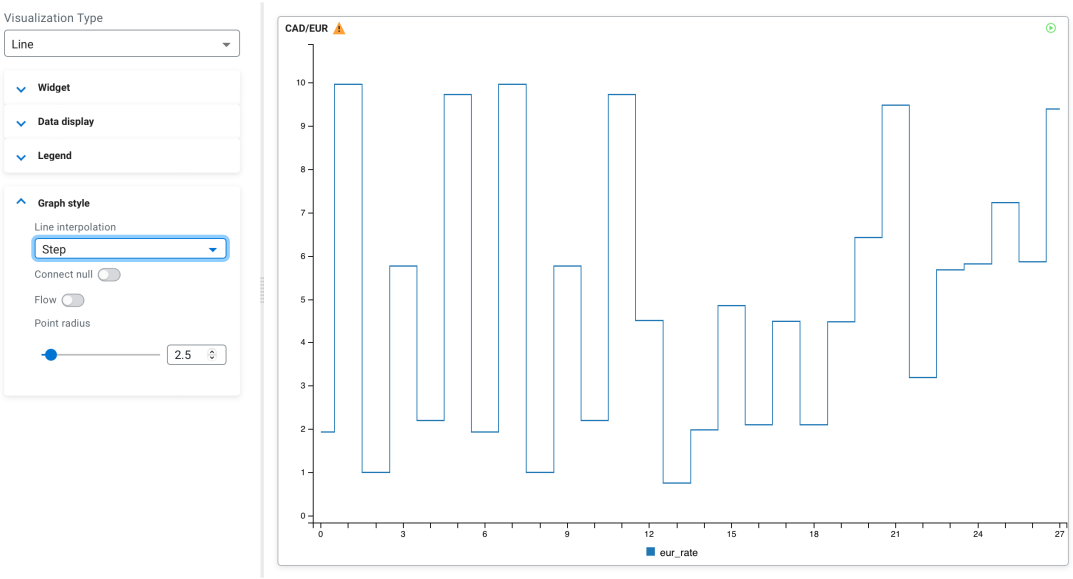
The following configurations have been set for the Line widget:

Name	CAD/EUR
Data fields	eur_rate
Legend poition	Bottom
Line interpolation	Line
Point radius	2.5

You can further customize the graph style of the Line widget with the following Graph style configurations:

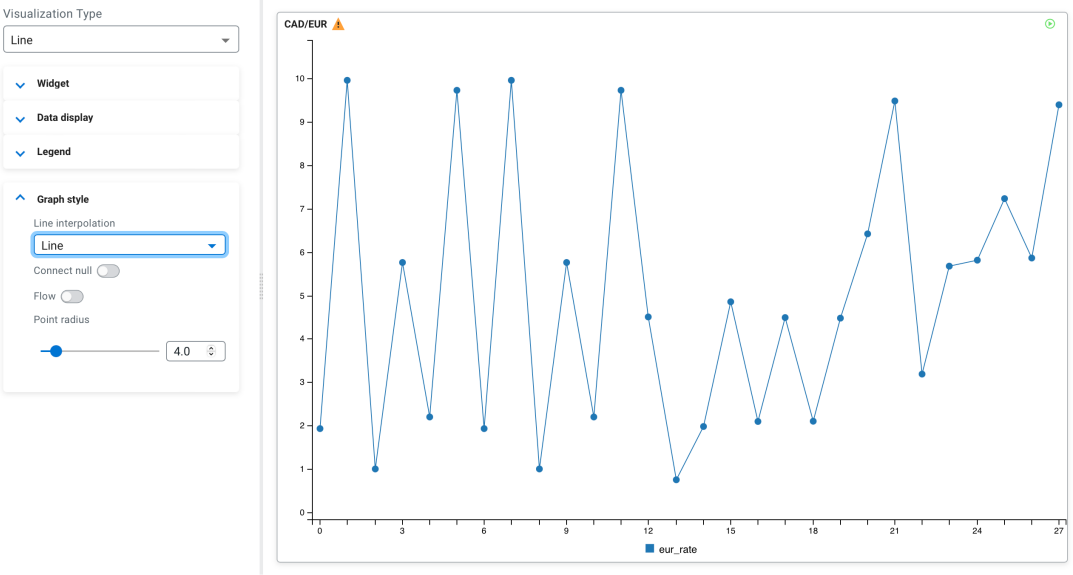
Line interpolation

Determines the line interpolation of the visualization. You have the option to select Step as well as the interpolation as shown in the following example:



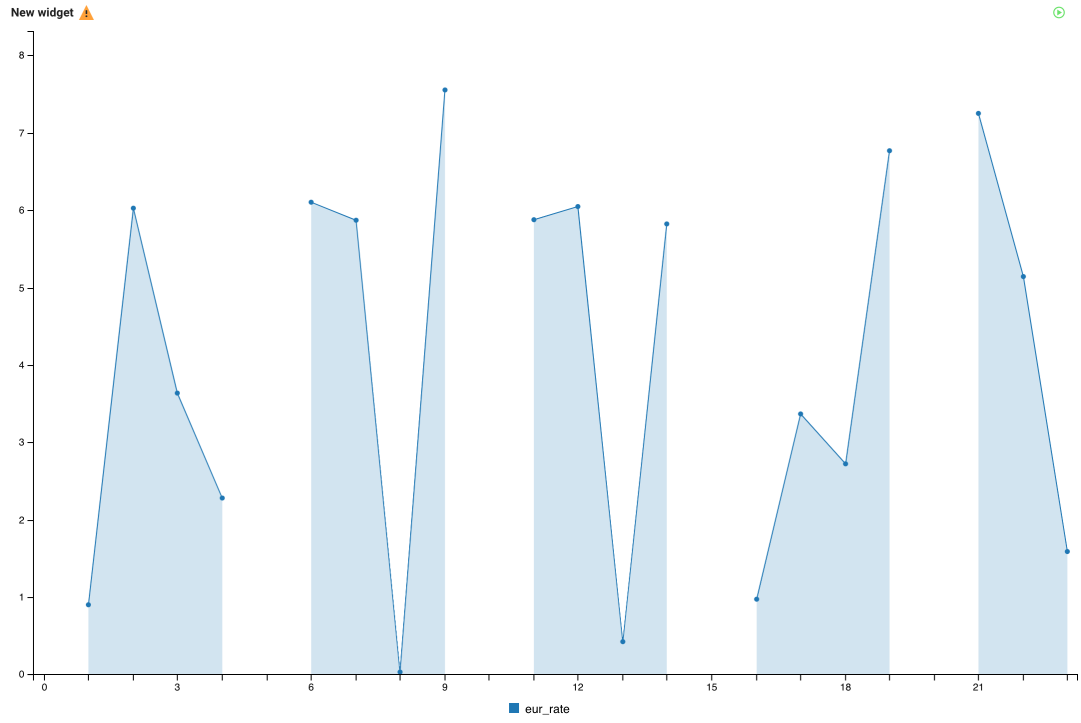
Point radius

You can enable whether to show each data point in line using the Show points toggle. The size of the points can be changed using the Point radius configuration, which is set to 2.5 by default. The following example shows the enabled Show points with 4.0 Point radius:



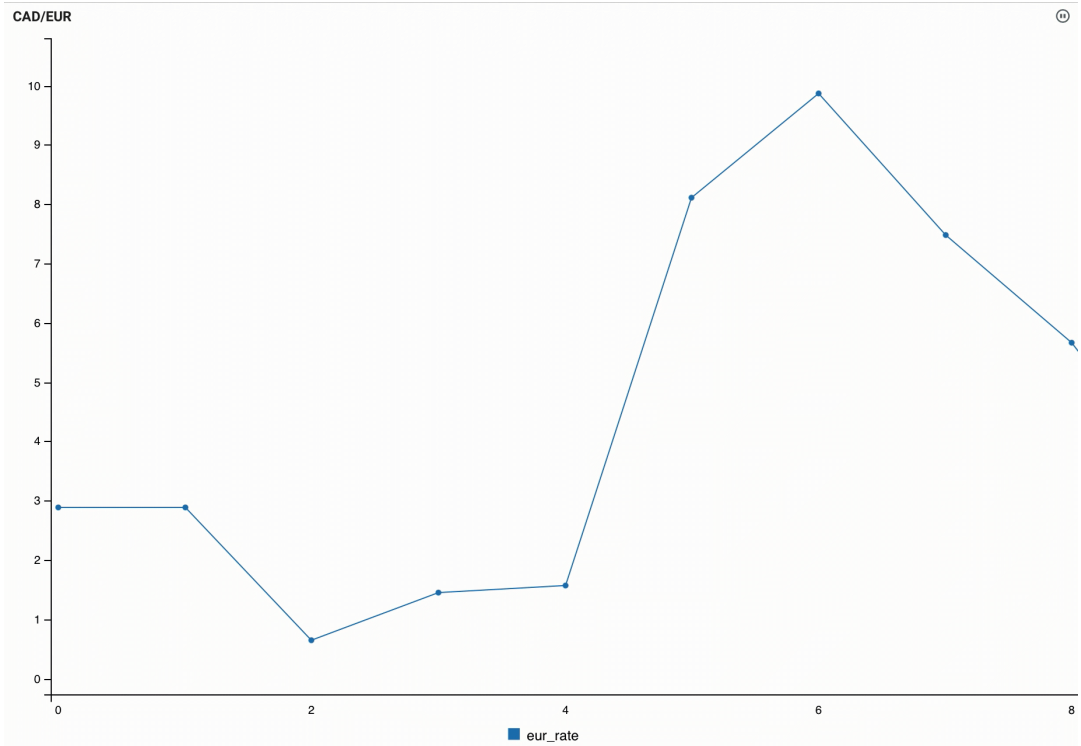
Connect null

Using the connect null configuration, you can set if the null data points are connected or not. If enabled, the region of the null data is connected without any data point. When disabled, the data points are not connected as shown in the following example:



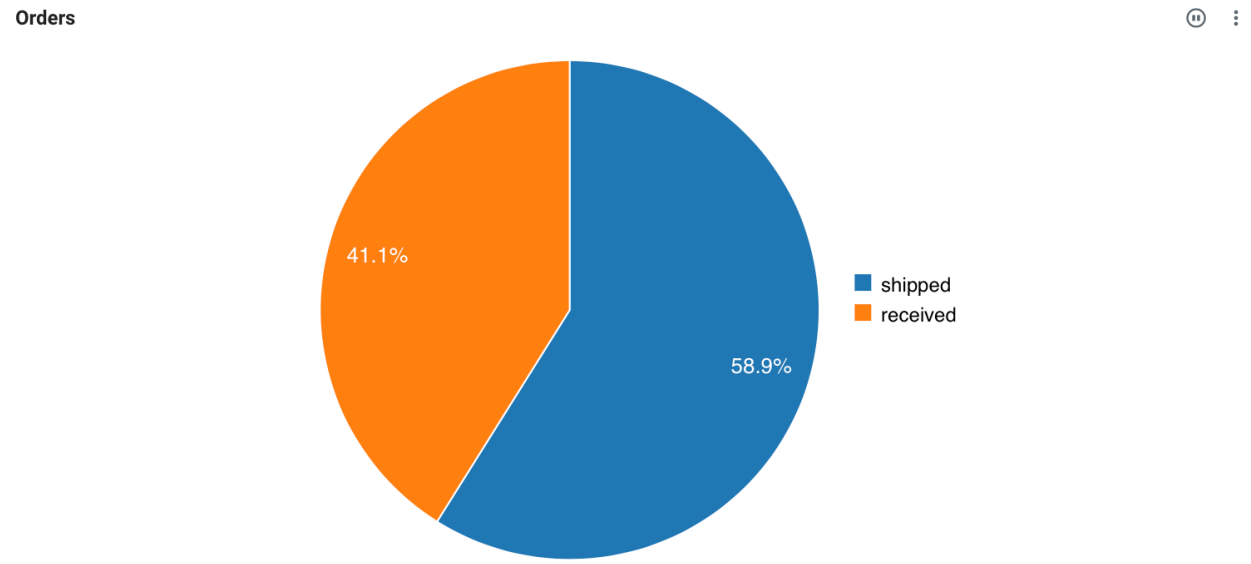
Flow

You can set the movement of the data by enabling or disabling the Flow function, which means the new data points flow to the chart from the right side compared to arriving periodically as processed on the widget.



Visualization type: Pie

You can use the Pie visualization type when you want to show data as a percentage compared to the whole dataset where each slice of the pie represents a category of data.



The following table and query was used for the Pie widget:

```
CREATE TABLE orders (
   `order_id` INT,
   `amount` INT,
   `order_time` TIMESTAMP(3),
   `shipped` VARCHAR(2147483647),
   `canceled` VARCHAR(2147483647),
   `received` VARCHAR(2147483647),
  WATERMARK FOR `order_time` AS `order_time` - INTERVAL '15' SECOND
) WITH (
   'fields.order_time.expression' = '#{date.past ''15'',''SECONDS''}',
   'fields.amount.expression' = '#{number.numberBetween ''0'',''100''}',
   'fields.order_id.expression' = '#{number.numberBetween ''0'',''99999999''
}',
   'connector' = 'faker',
   'fields.shipped.expression' = '#{number.numberBetween ''0'',''1000''}',
   'fields.received.expression' = '#{number.numberBetween ''0'',''800''}',
   'fields.canceled.expression' = '#{number.numberBetween ''0'',''500''}',
   'rows-per-second' = '100'
);

SELECT * FROM orders
```

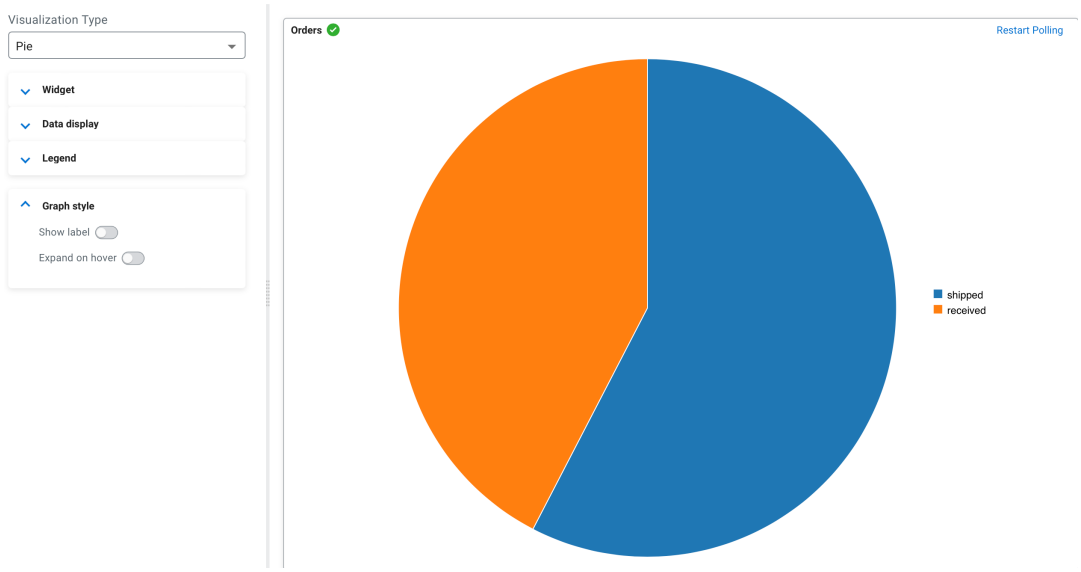
The following configurations have been set for the Pie widget:

Name	Orders
Data fields	shipped, received
Legend poition	Right
Show label	enabled

You can further customize the graph style of the Pie widget with the following Graph style configurations:

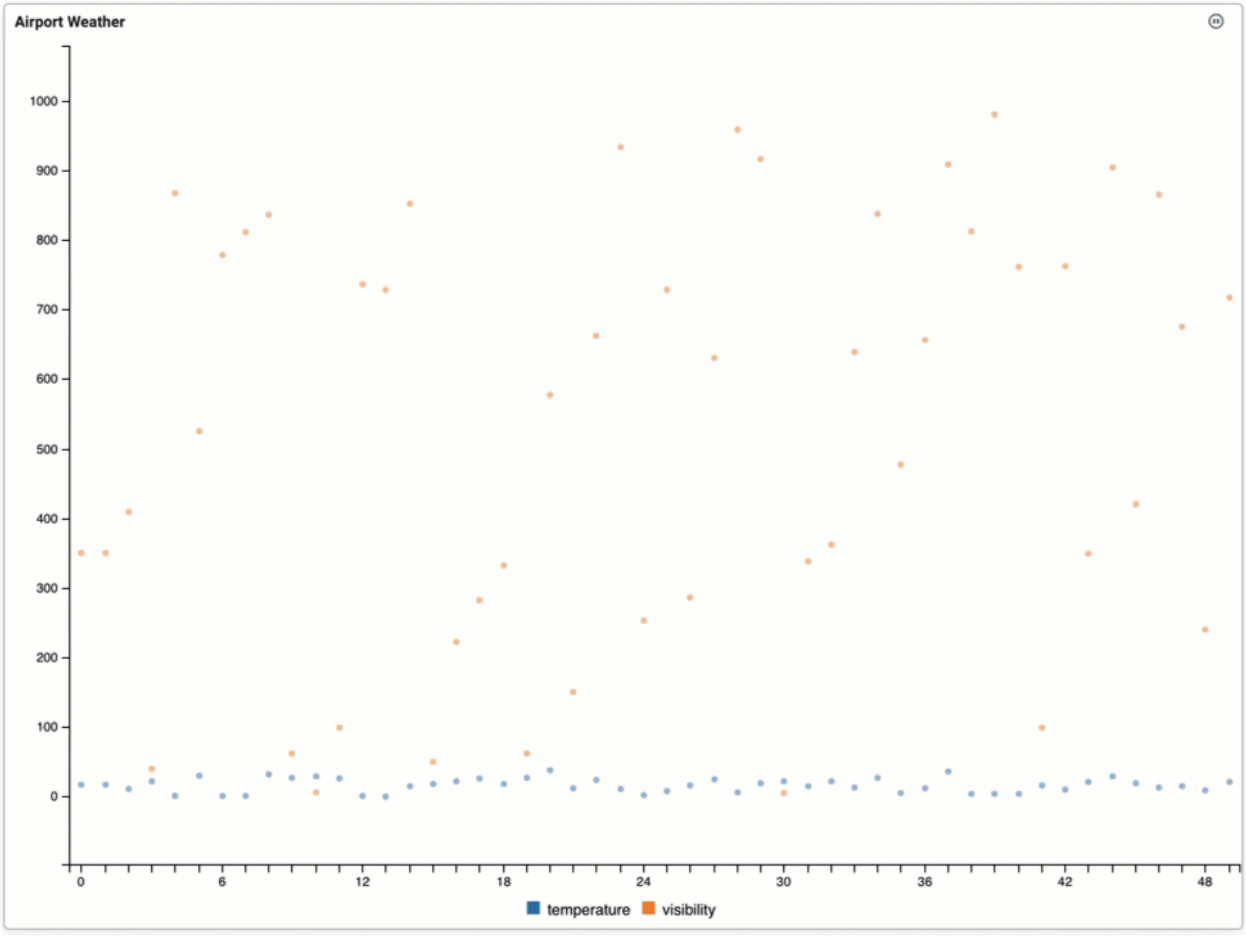
Show label

You can enable or disable the labels on the visualization as shown in the following illustration where labels are disabled:



Visualization type: Scatter

You can use the Scatter visualization type when you want to show the correlation between different datasets.



The following table and query was used for the Scatter widget:

```
CREATE TABLE airport_weather (
  `airport` VARCHAR(2147483647) NOT NULL,
  `temperature` VARCHAR(2147483647) NOT NULL,
```

```
  `visibility` VARCHAR(2147483647) NOT NULL,
  `eventTimestamp` TIMESTAMP(3),
  WATERMARK FOR `eventTimestamp` AS `eventTimestamp` - INTERVAL '30' SECOND,
  CONSTRAINT `PK_-991666966` PRIMARY KEY (`airport`) NOT ENFORCED
) WITH (
  'fields.eventTimestamp.expression' = '#{date.past ''30'',''SECONDS''}',
  'connector' = 'faker',
  'fields.temperature.expression' = '#{Weather.temperatureCelsius}',
  'fields.airport.expression' = '#{Aviation.airport}',
  'rows-per-second' = '100',
  'fields.visibility.expression' = '#{numerify ''###m''}'
);

SELECT * FROM airport_weather
```

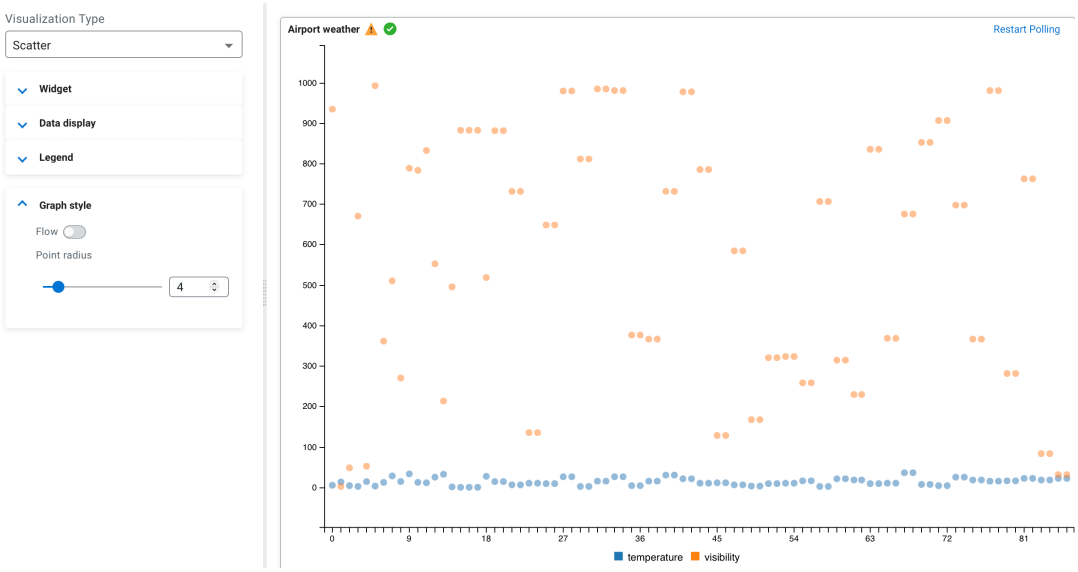
The following configurations have been set for the Scatter widget:

Name	Airport weather
Data fields	temperature, visibility
Legend poition	Bottom
Point radius	2.5

You can further customize the graph style of the Scatter widget with the following Graph style configurations:

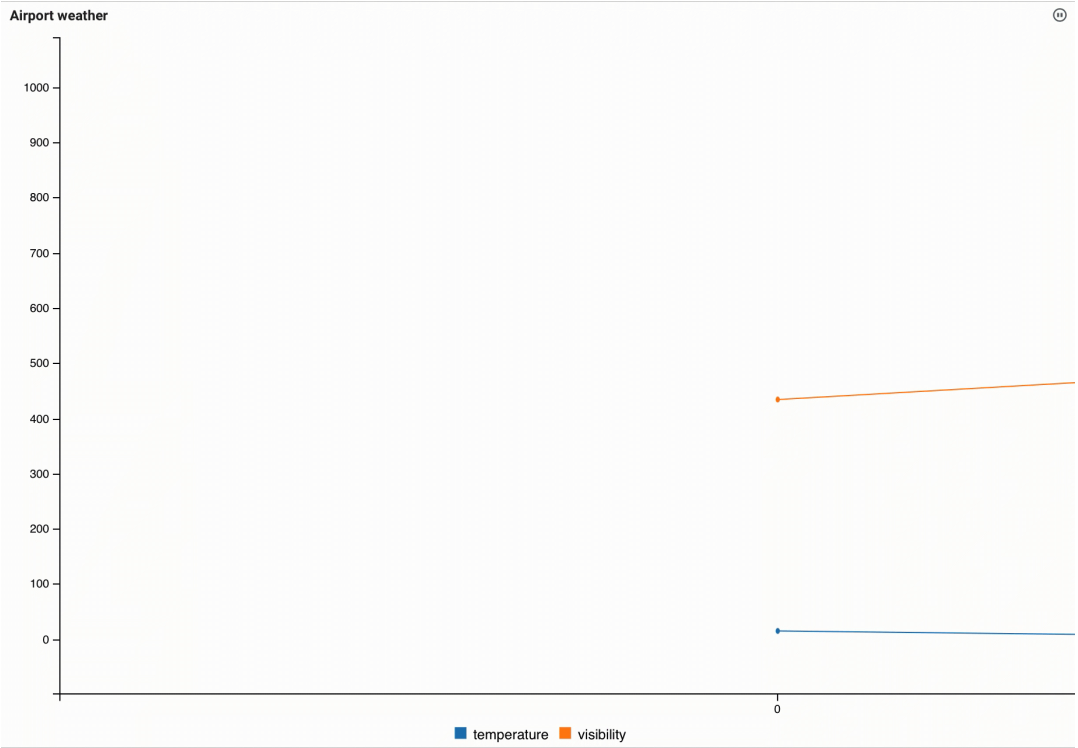
Point radius

You can enable whether to show each data point in line using the Show points toggle. The size of the points can be changed using the Point radius configuration, which is set to 2.5 by default. The following example shows the enabled Show points with 4.0 Point radius:



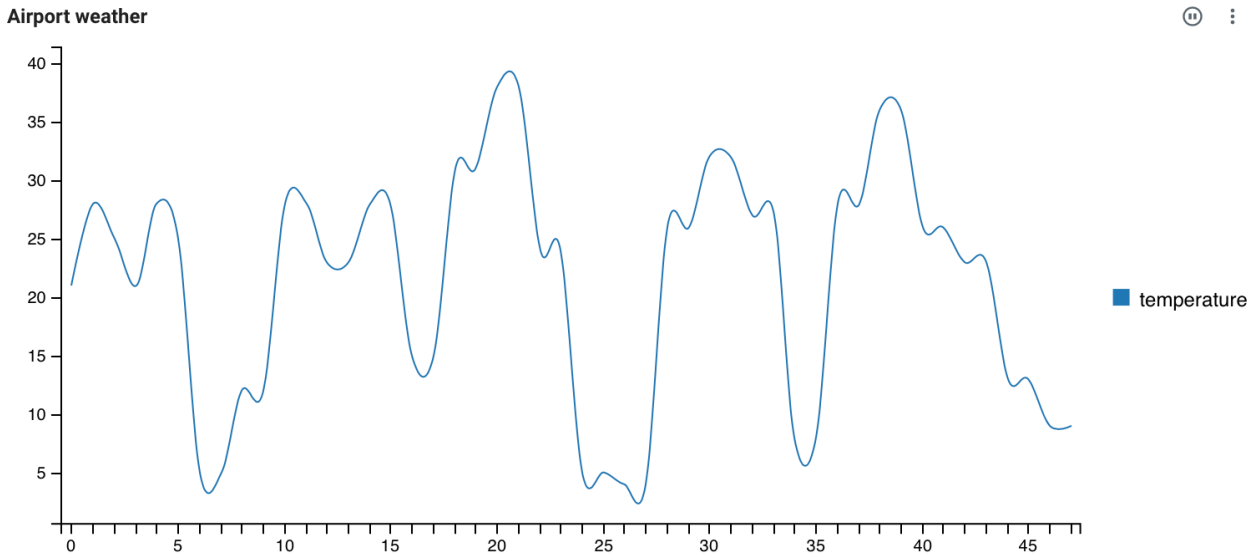
Flow

You can set the movement of the data by enabling or disabling the Flow function, which means the new data points flow to the chart from the right side compared to arriving periodically as processed on the widget.



Visualization type: Spline

You can use the Spline visualization type when you want to show how the data changed over a period of time where the trends of change are emphasized.



The following table and query was used for the Spline widget:

```
CREATE TABLE airport_weather (
  `airport` VARCHAR(2147483647) NOT NULL,
  `temperature` VARCHAR(2147483647) NOT NULL,
  `visibility` VARCHAR(2147483647) NOT NULL,
  `eventTimestamp` TIMESTAMP(3),
  WATERMARK FOR `eventTimestamp` AS `eventTimestamp` - INTERVAL '30' SECOND,
  CONSTRAINT `PK_-991666966` PRIMARY KEY (`airport`) NOT ENFORCED
) WITH (
  'fields.eventTimestamp.expression' = '#{date.past ''30'',''SECONDS''}',
```

```
  'connector' = 'faker',
  'fields.temperature.expression' = '#{Weather.temperatureCelsius}',
  'fields.airport.expression' = '#{Aviation.airport}',
  'rows-per-second' = '100',
  'fields.visibility.expression' = '#{numerify ''###m''}'
);

SELECT * FROM airport_weather
WHERE airport = 'BIPA'
```

The following configurations have been set for the Spline widget:

Name	Airport weather
Data fields	temperature
Legend poition	Right
Spline interpolation	cardinal
Point radius	0

You can further customize the graph style of the Spline widget with the following Graph style configurations:

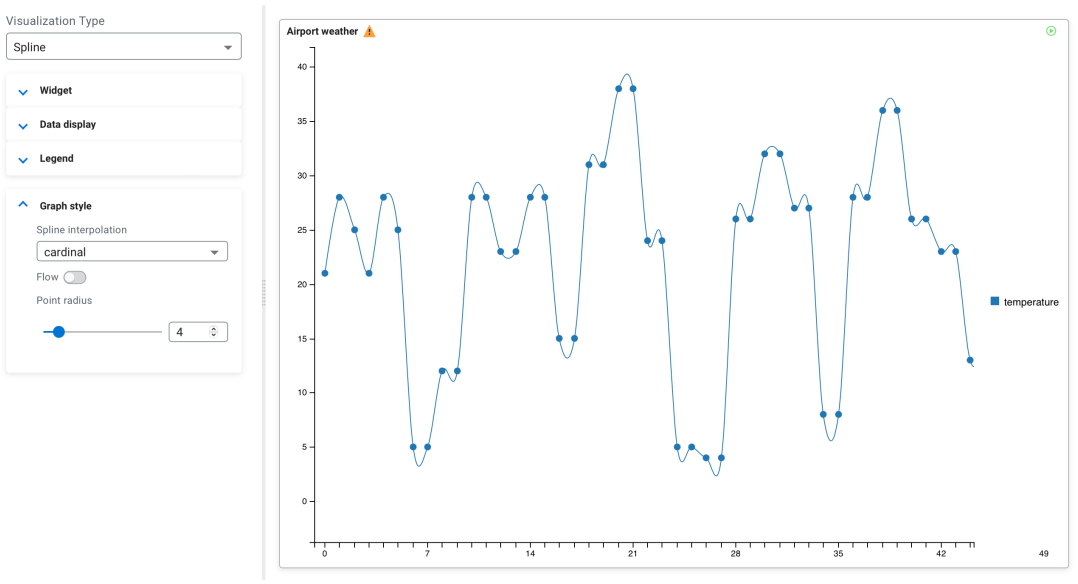
Line interpolation

You can specify the Spline interpolation which includes the following list of options:

- Linear, linear-closed
- Basis, basis-open, basis-closed
- Cardinal, cardinal-open, cardinal-closed
- Monotone
- Step, step-before, step-after

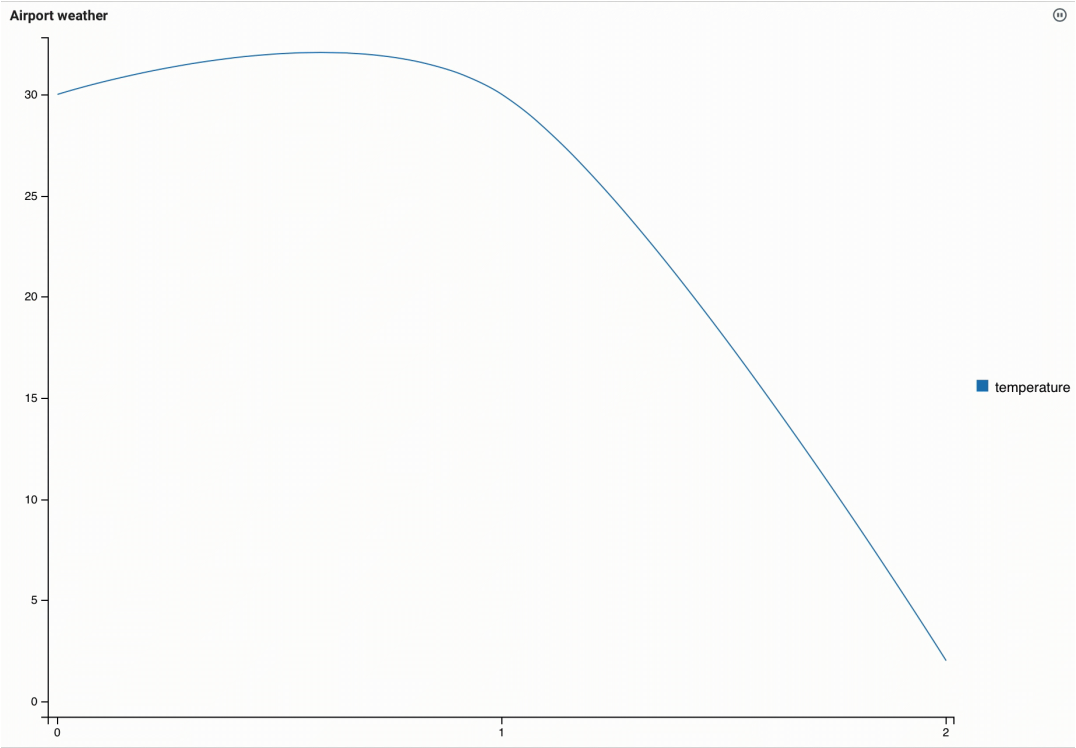
Point radius

You can enable whether to show each data point in line using the Show points toggle. The size of the points can be changed using the Point radius configuration, which is set to 2.5 by default. The following example shows the enabled Show points with 4.0 Point radius:



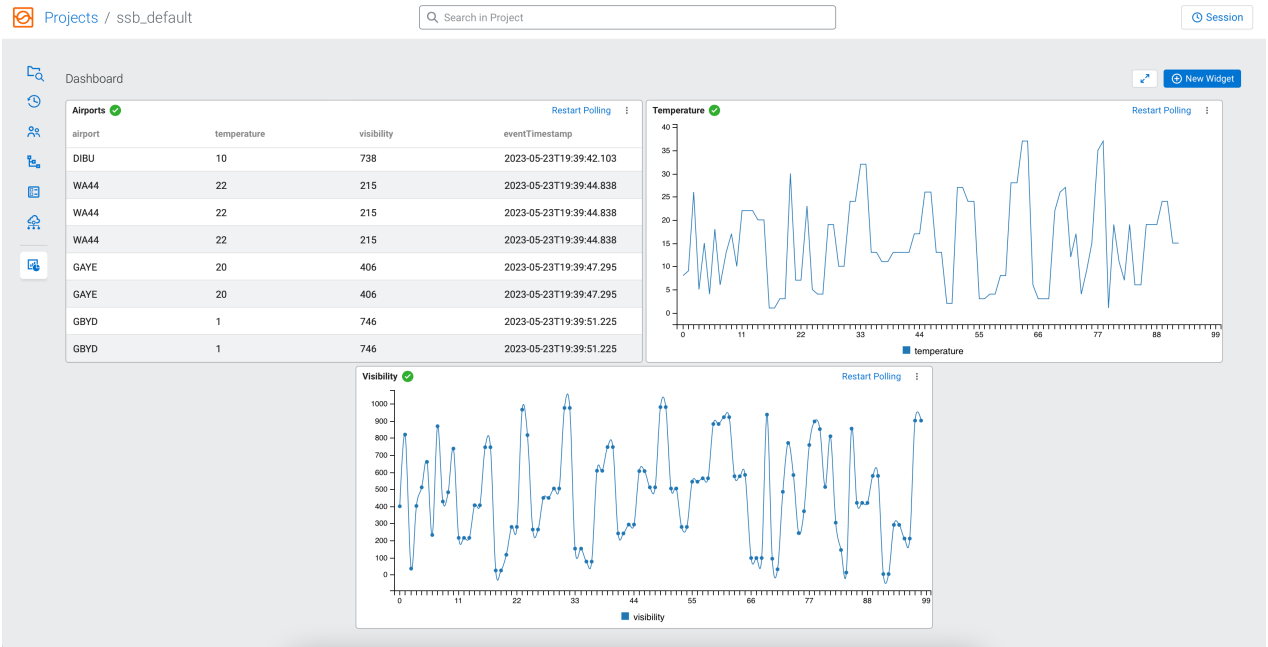
Flow

You can set the movement of the data by enabling or disabling the Flow function, which means the new data points flow to the chart from the right side compared to arriving periodically as processed on the widget.



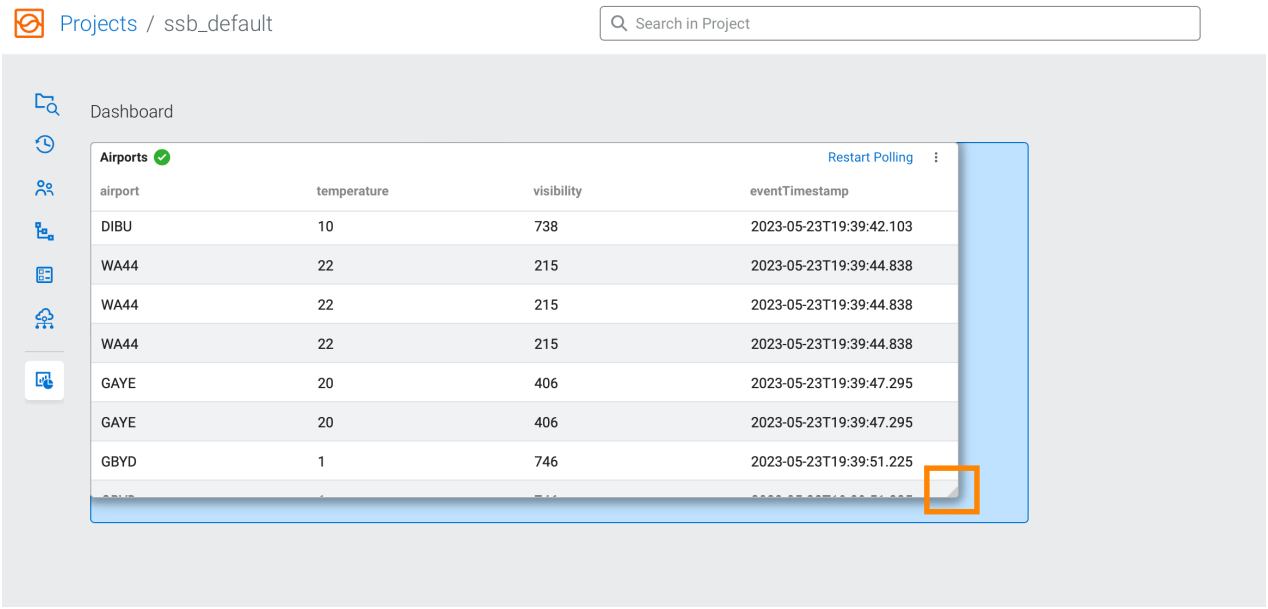
Managing widgets on the Dashboard

After a widget is created, you can customize the layout and control the frequency of the data update from the Dashboard page.

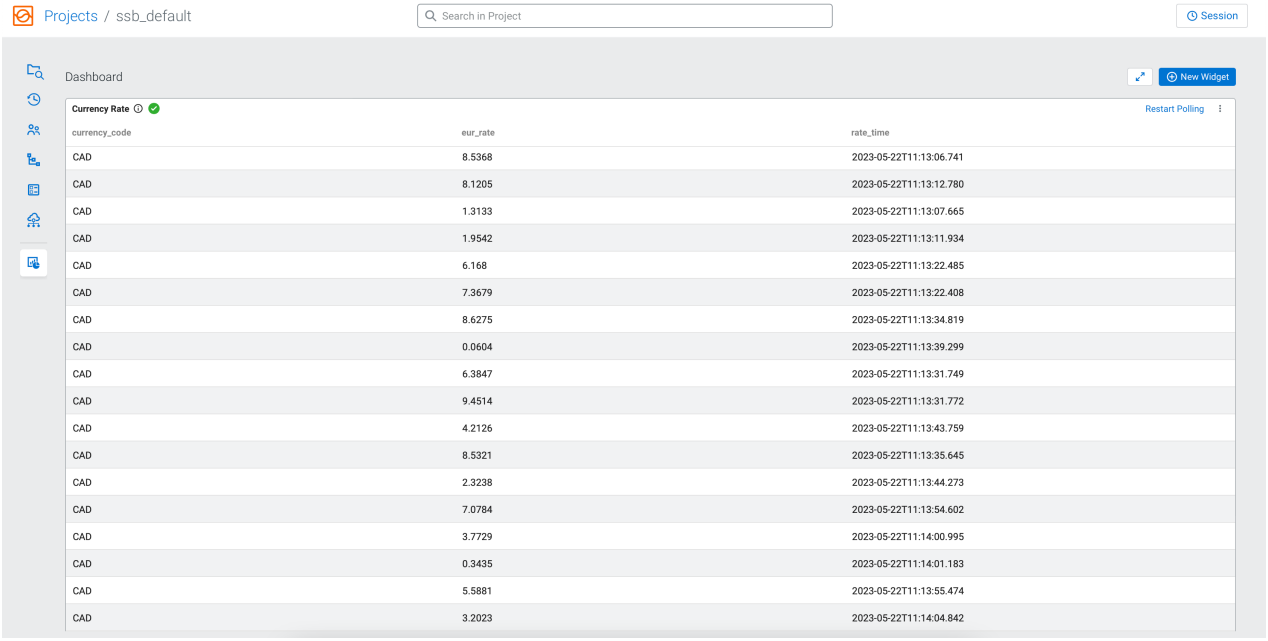


Layout

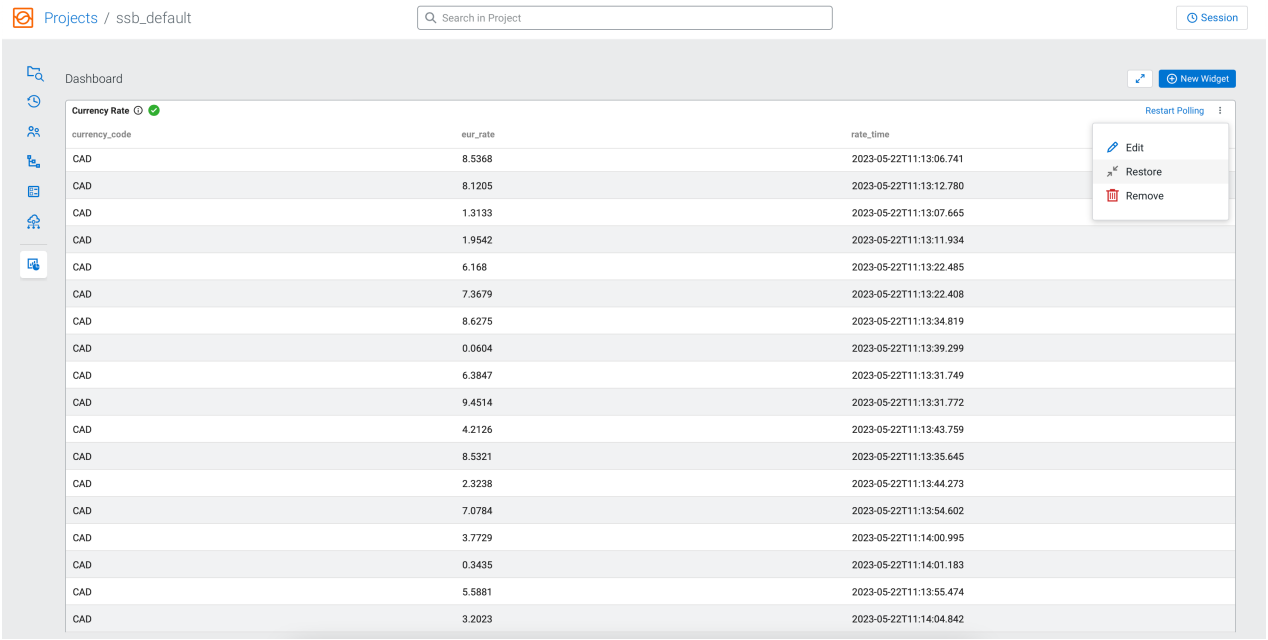
You can resize and reorganize the widgets on the **Dashboard** page based on your preferences. You can drag and drop the widgets to position them to different places on the **Dashboard**. To resize a widget, you need to click and drag to the preferred size the right bottom corner of the widget.



You can also maximize the widget window size to fill out the empty spaces of the Dashboard when only one widget is required for data visualization using the Maximize option.



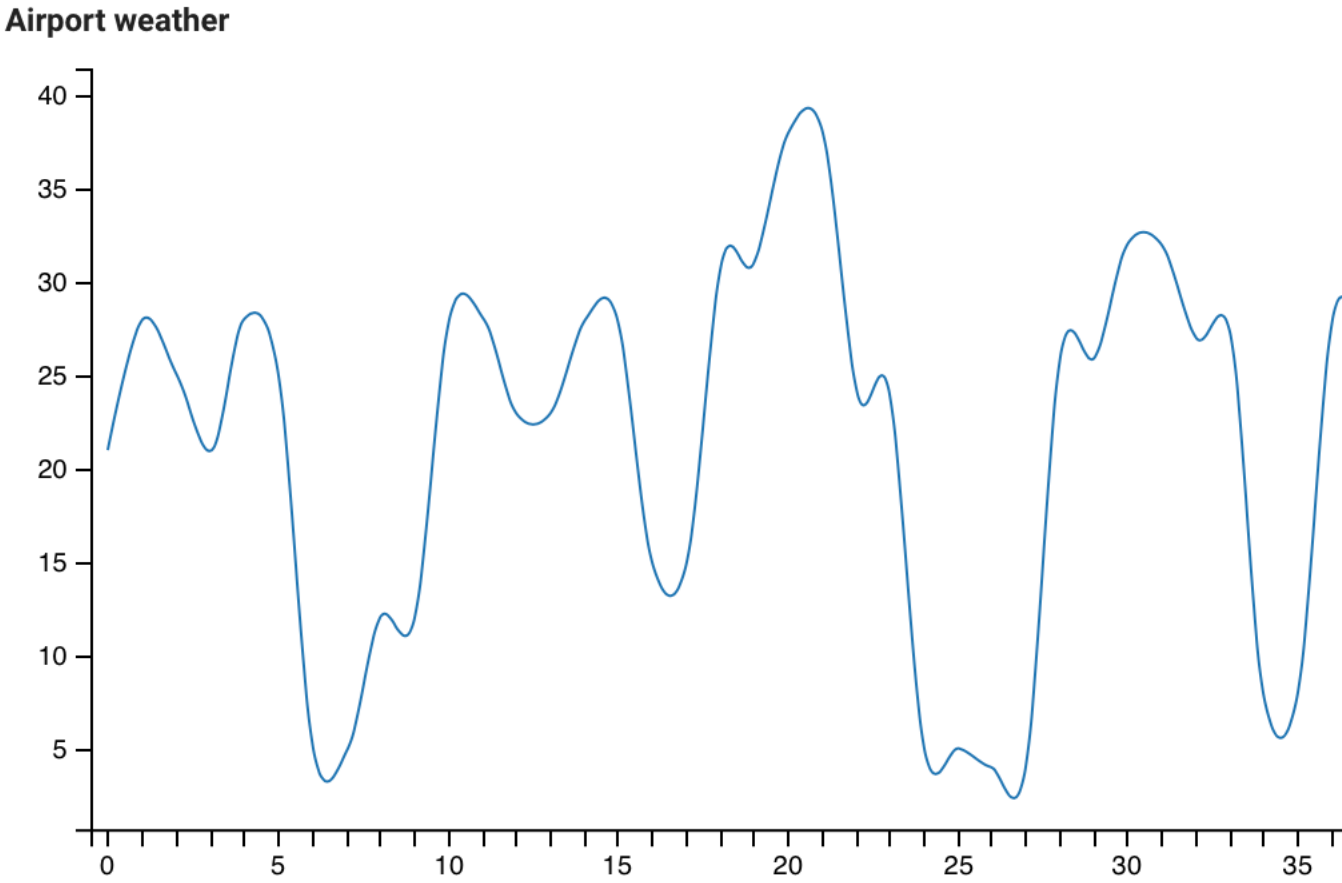
When the original size is required, the size can be reverted back by clicking Revert.



Not only the widgets, but you can also set the **Dashboard** page to a full screen mode using button to present the widgets without the Project Manager and Search bar on the screen.

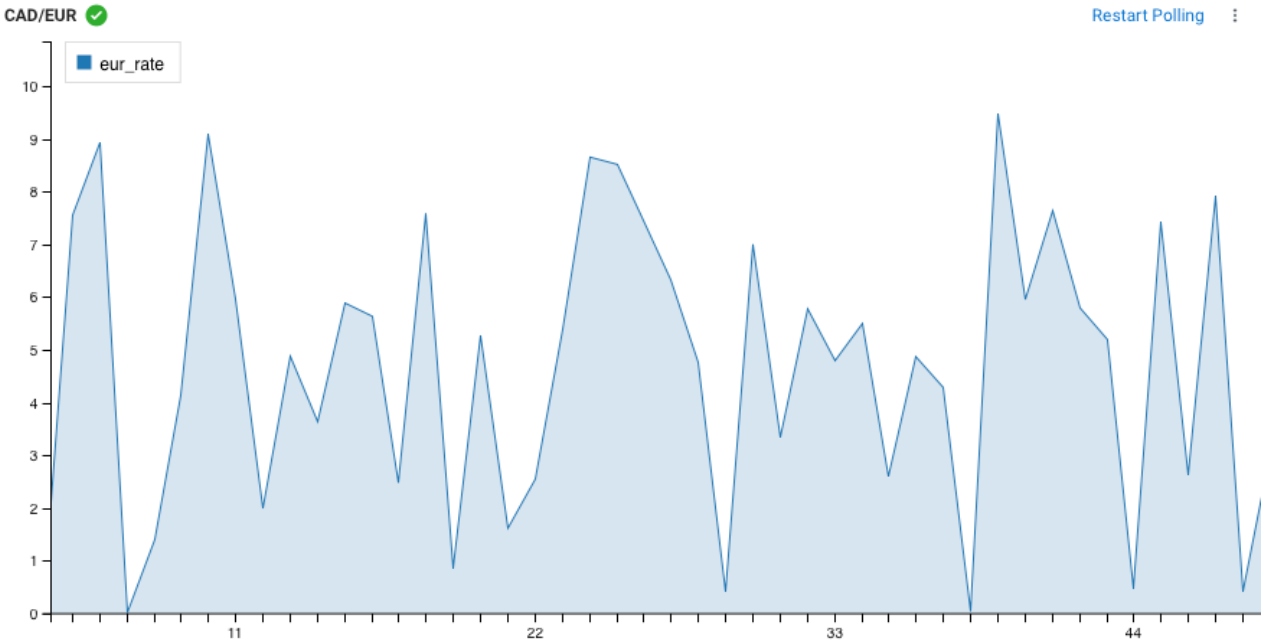
Data update

You can control the frequency of the data update for a widget by either refreshing it manually between the update cycles, or you can temporarily pause the automatic data update to have a snapshot of the streaming data in that



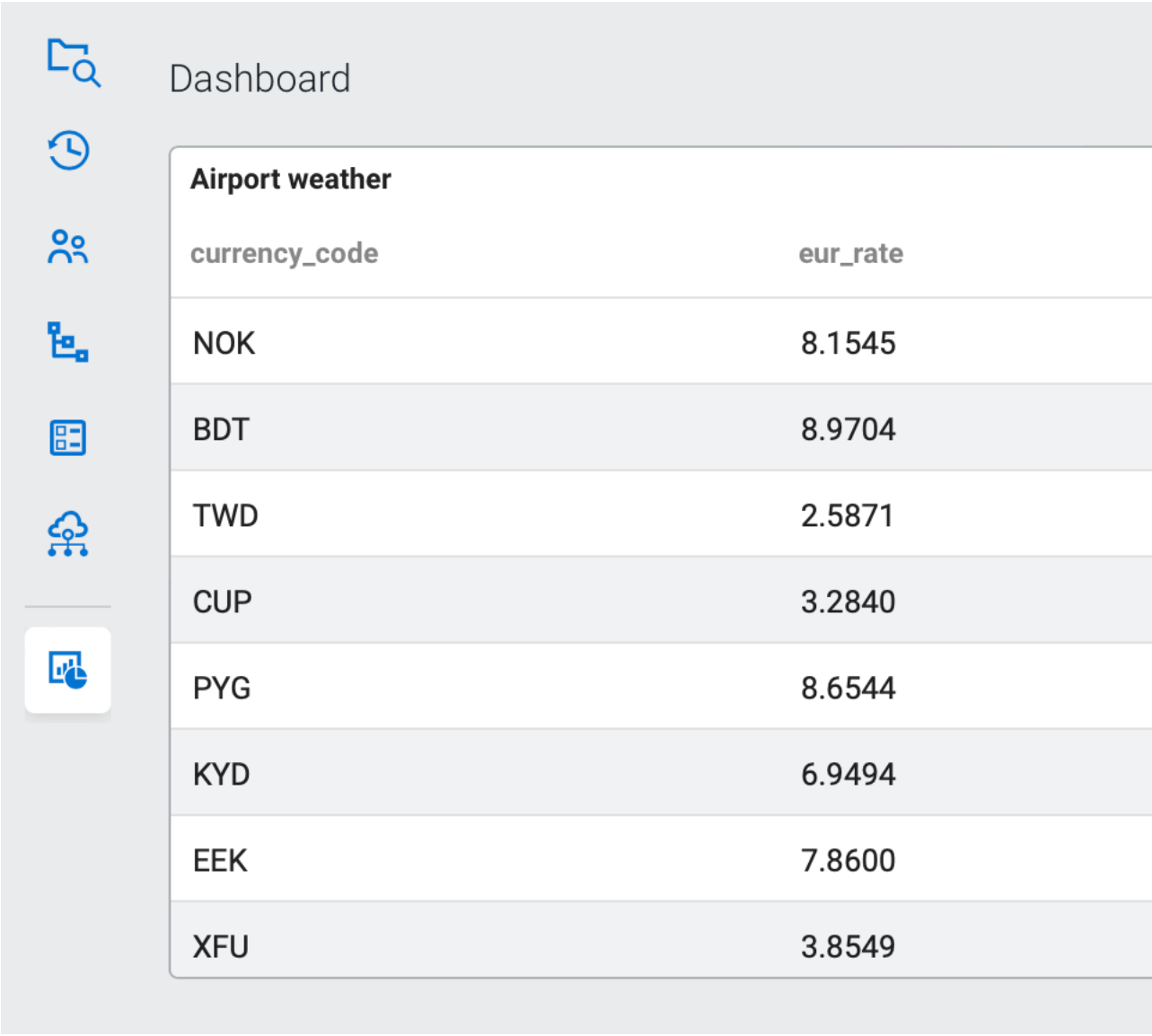
point.

When sampling is in progress, you can use the button to pause the auto update. When the auto update is paused, icon appears next to the name of the widget to indicate that the sampling is stopped. You can resume the update of data by clicking on button.When polling the samples are finished as the configured amount of messages are processed by the widget, icon appears next to the name of the widget. You can restart the polling by clicking Restart Polling.



When a Materialized View endpoint is used as a data source, you can manually update the data on the visualization by clicking button individually on the widget.

You can also update every Materialized View widget using the refresh button on the



Dashboard.

If the Update Interval of the widget is set to Default, the interval can be specified using the time drop-down and selecting an interval.

Storing configuration

Widgets and Dashboard settings are stored in the local storage of the browser, which enables you to always access the created widgets within a project.

Note: Widgets are not included when exporting a project.

Using SQL job notification

You can set up notifications for SQL jobs using Streaming SQL Console to alert individuals and teams about job failures. The notifications include email and webhook alerts, which you can also group together for easier organization.

You can add job notifications to SQL jobs to send alerts and information when the status of the job is changed. The information in the notification can include the job name, job ID, cluster ID, last exception, and Flink job ID.

You can create the following notification actions for SQL jobs:

Email

Sending notification email to the provided email address

Webhook

Sending notification to a webhook address using POST and PUT

You can also collect the Email and Webhook notifications into a **Group** to organize the notifications. These groups do not trigger any action, but serve as an organizational element.

After creating and adding the notification to a SQL job, anytime a change occurs for that job, a notification is sent to the specified address.

Enabling job notifications

Before you are able to use the job notifications, you need to enable it in Cloudera Manager and based on the notification type configure the email and webhook parameters.

1. Go to your cluster in Cloudera Manager.
2. Select Cloudera SQL Stream Builder from the list of services.
3. Click Configuration.
4. Search for *ENABLE JOB NOTIFICATIONS FUNCTIONALITY*.
5. Enable the job notifications by checking the checkbox.

The following parameters also need to be configured for the Cloudera SQL Stream Builder service in Cloudera Manager based on the type of the notification:

Configuration	Description
Job notifications monitoring interval	Sets the interval of the job monitoring in seconds.
Mail server host for job notifications	The host of the SMTP server for job failure notifications.
Mail server username for job notifications	The username to access the SMTP server for job failure notifications.
Mail server password for job notifications	The password to access the SMTP server for job failure notifications.
SMTP authentication for job notifications	Enable SMTP authentication for job notifications.
StartTLS for job notifications	Use the StartTLS command to establish a secure connection to the SMTP server for job notifications.
Job notifications sender mail address	Sender mail address for job notifications.
Mail server port for job notifications	The port of the SMTP server for job failure notifications.
Job notifications webhook sender parallelism	Number of threads used by the job notification task to call user-specified webhooks when notifying about a failed or missing job.



Note: In case the YARN cluster of a failed job no longer exists, Cloudera SQL Stream Builder attempts to fetch the information about the failed job from the Flink History Server. If information cannot be fetched by the first attempt, Cloudera SQL Stream Builder attempts to fetch the information again based on the interval configured for the History server fetch retry time parameter.

Creating job notifications

After enabling the job notification function, you can create email, webhook and group notifications with Streaming SQL Console.

1. Navigate to Streaming SQL Console.
   a. Go to your cluster in Cloudera Manager.
   b. Select Cloudera SQL Stream Builder from the list of services.

c. Click Streaming SQL Console.

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

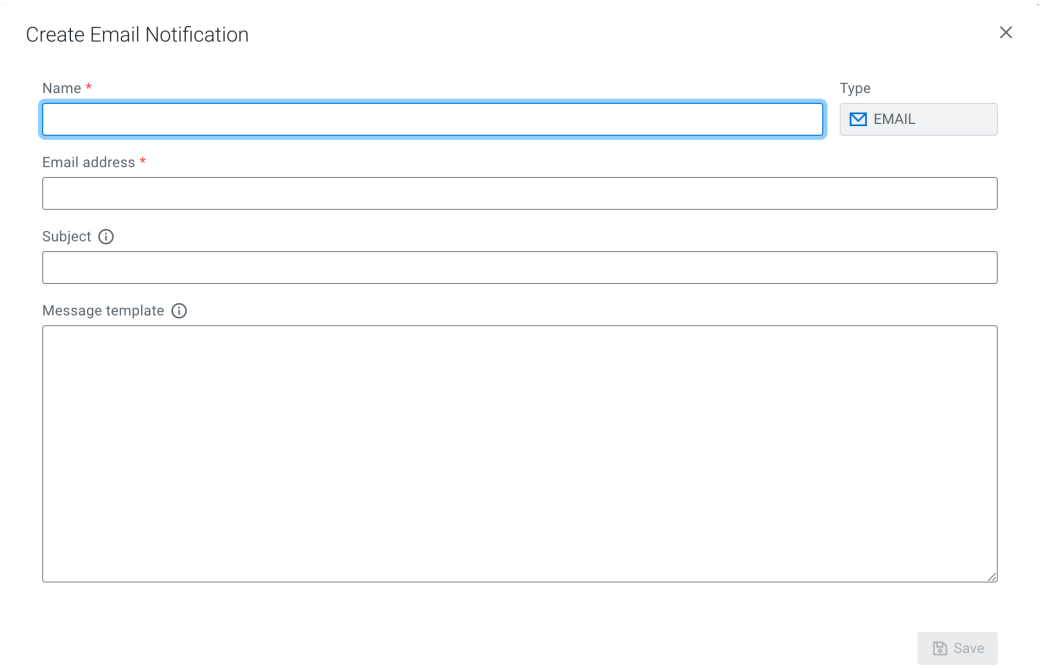
2. Open a project from the **Projects** page.
   a. Select an already existing project from the list by clicking the Open button or Switch button.
   b. Create a new project by clicking the New Project button.
   c. Import a project by clicking the Import button.

   You are redirected to the **Explorer** view of the project.
3. Click ⋮ next to **Job Notifications**.
4. Select from the following options based on which notification type you want to use:

For Email

a. Click New Email Notification .

The **Create Email Notification** window appears.



b. Provide a Name for the notification.
c. Add the Email address you want to send the notification.
d. Provide a Subject.
e. Provide a Message.

The message of the Webhook notification must be a valid JSON.

You can add placeholder items to the payload that will be converted into the specific job information. You can add the placeholders in ${placeholder} format, and the following placeholders are available:

- jobName
- jobStatus
- jobStatusDescription
- ssbJobId
- flinkJobId

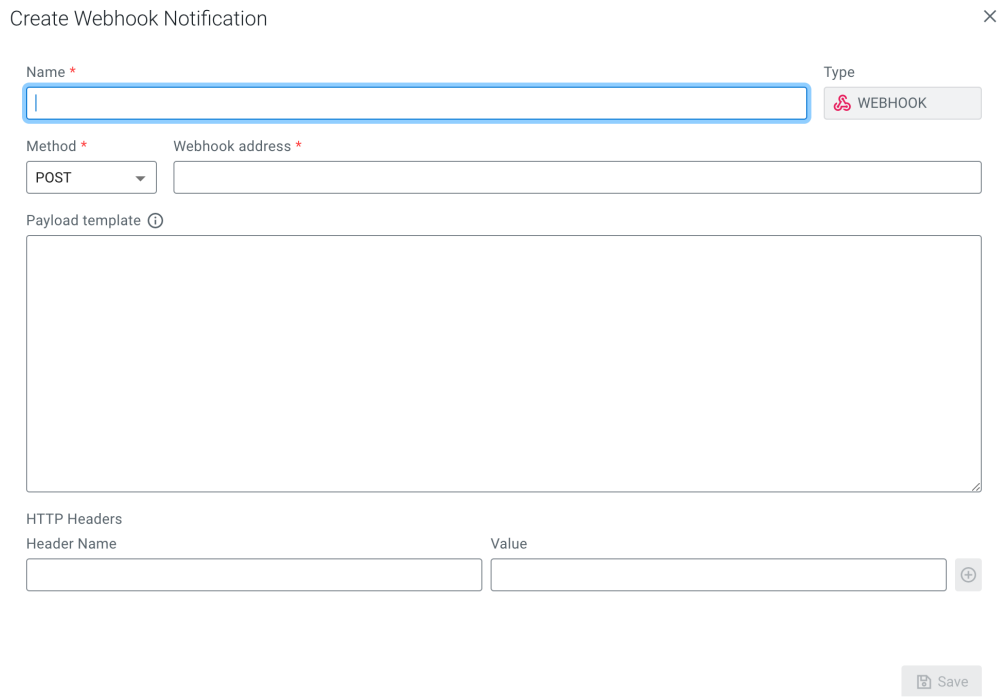
- clusterId
- lastException

f. Click Save.

For Webhook

a. Click New Webhook Notification .

The **Create Webhook Notification** window appears.



b. Provide a Name for the notification.

c. Select between POST or PUT method.

d. Provide a Webhook address.

e. Provide a Payload to the webhook.

Warning: The payload must be a valid JSON object.

You can add placeholder items to the payload that will be converted into the specific job information. You can add the placeholders in ${placeholder} format, and the following placeholders are available:

- jobName
- jobStatus
- jobStatusDescription
- ssbJobId
- flinkJobId
- clusterId
- lastException

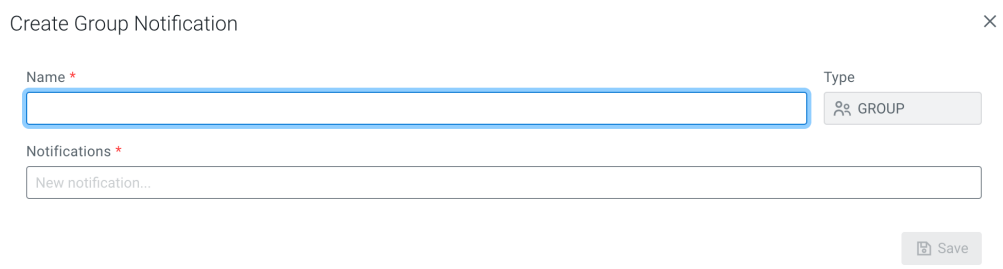
f. Add **HTTP Headers** if needed by specifying the Header Name and Value.

g. Click Save.

For Groups

a. Click New Group Notification .

The **Create Group Notification** window appears.



b. Select from the already created Email and Webhook notifications that you would like to add to this group.
c. Click Save.

Enabling notifications for a SQL job

After creating a job notification, you can enable it for a selected job from the **Job Settings** window.

1. Select Job Settings from the **SQL Editor**.
2. Click Job Notifications.
3. Search for the name that you have provided when creating the notification.
4. Click on the notification.

   The selected notification is added to the job and the notification is sent with the information provided in the message template.

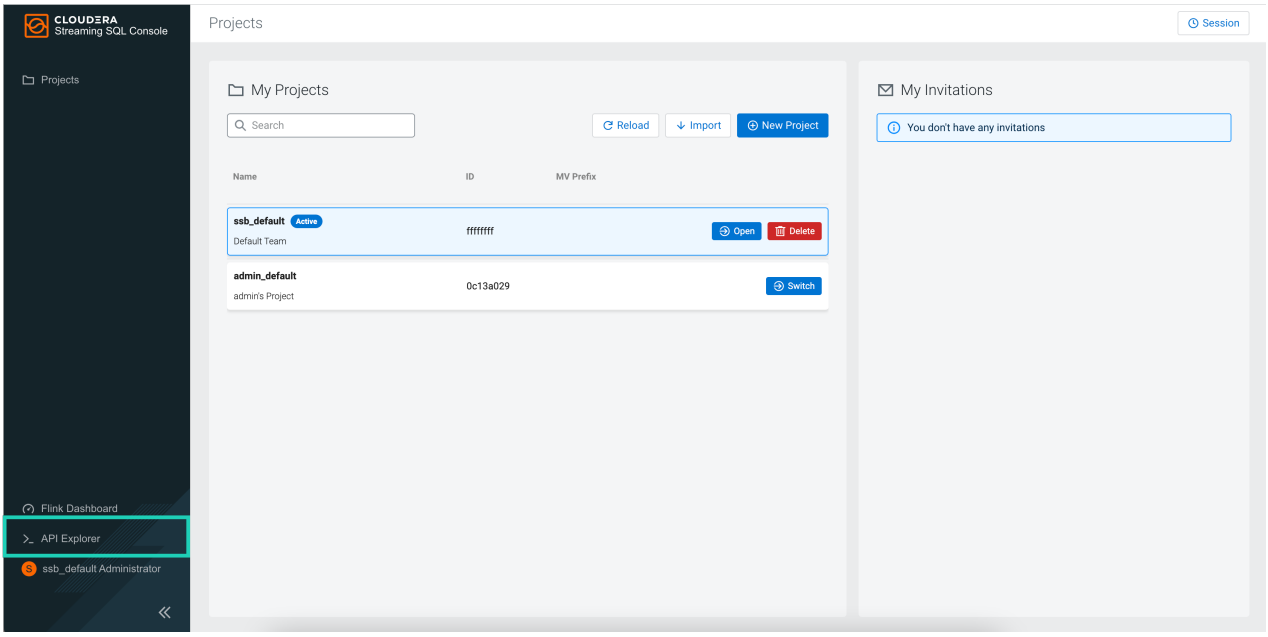
Using Cloudera SQL Stream Builder REST API

You can use the REST API to monitor, manage and configure the SQL Stream jobs with GET, POST and DELETE HTTP methods. You can use the Cloudera SQL Stream Builder REST API in the command line, import them to REST API Tools or by accessing the Swagger UI.

The following HTTP methods are available for SCloudera SQL Stream Builder:

- GET to query information about the specified endpoint
- POST to create resources for the specified endpoint
- PUT to update existing resources for the specified endpoint as a whole
- PATCH to partially update existing resources for the specified endpoint
- DELETE to remove objects from the specified endpoint

The REST API Reference document contains the available endpoints for Cloudera SQL Stream Builder. However, you can also reach the Cloudera SQL Stream Builder REST API reference document from on the main menu of Streaming SQL Console.



You can use the Cloudera SQL Stream Builder REST API with Command Line Interface (CLI), you can also import the REST API swagger.json file to a REST API Tool, for example Postman, and you also have the option to use the REST API with the Swagger User Interface (UI).

The Streaming SQL Engine API details the following operations for SQL Stream Builder and Flink:

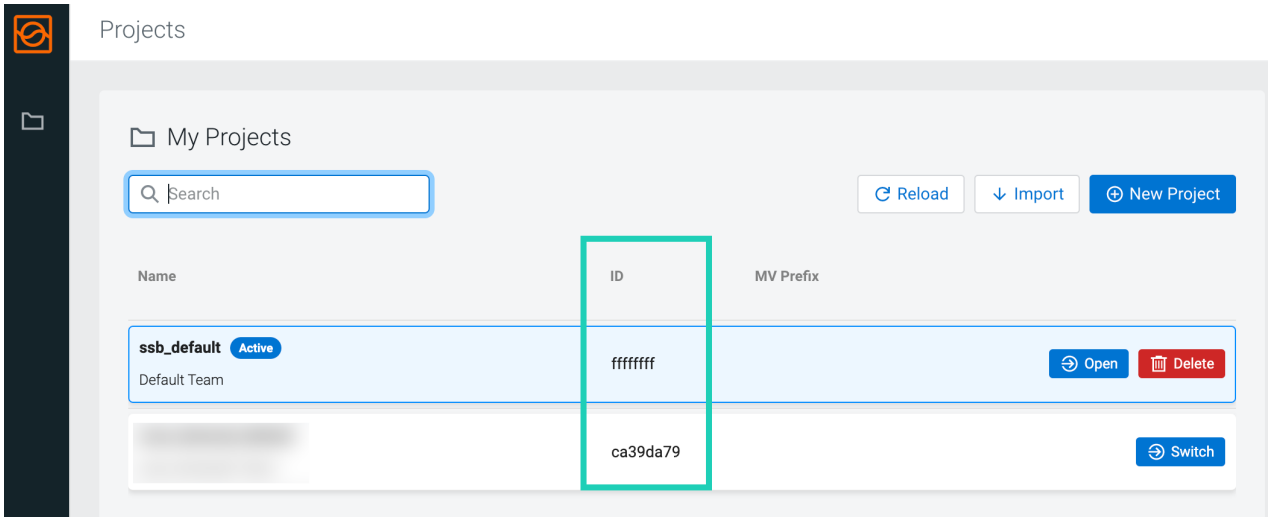
- Heartbeat
- Admin Operations
- User Operations
- User Keytab Operations
- Data Source Operations
- Table Operations
- Project Operations
- Project Environment Operations
- Project Sync Operations
- Project Invitation Operations
- Project Permission Operations
- SQL Operations
- SSB Session Operations
- Job Operations
- Sampling Operations
- Flink Job Operations
- Flink Session Cluster Operations
- Artifact Operations
- UDF Artifact Operations
- UDF Operations
- API Key Operations
- Connector Operations
- Data Format Operations
- Diagnostic Operations

The Cloudera SQL Stream Builder project-related endpoints can be used with specifying which project the request is submitted to by adding the project ID parameter to the endpoint. This means that even though a project is selected as the active project on Streaming SQL Console, you can submit any request to a different project to which you have

access to. The REST API user permissions are the same for a project as it is configured on the Members page in a project. The requests that are submitted using a specific project ID are submitted as a new session that does not interfere with the already existing user sessions in SSB.

Accessing the project ID

You can find the project ID on the homepage of Streaming SQL Console:



You can also use the GET /api/v2/projects call in CLI to list all projects that the user has access to.

Using REST API Tool

When using REST API Tools, you need to connect to the cluster that hosts the Cloudera SQL Stream Builder REST API. You can also use the following URL in a browser to download the swagger.json file when you need to manually upload the REST API endpoints for a Tool.

```
http(s)://[***CLUSTER_DOMAIN***]:[***STREAMING_SQL_ENGINE_SERVER_PORT***]/
swagger/api-docs/public
```

Using REST API with CLI

When using the Cloudera SQL Stream Builder REST API with CLI Tool, you need to create the POST or GET commands with curl, and also include the Streaming SQL Engine hostname, port and the required operation for the endpoint. The submitted commands return the information, or complete the process you have requested and display the status in the CLI.

The following examples show a secured and unsecured GET method to list the SQL jobs:

For Unsecured

```
$ curl -H 'Username: [***YOUR_USERNAME***]' \
       'http
s://[***CLUSTER_DOMAIN***]:[***STREAMING_SQL_ENGINE_SERVER_PORT***]/api/
v1/ssb/jobs'
```

For Secured with SPENGO

```
$ kinit [***YOUR_USERNAME***]
$ curl -ik --negotiate -u: \
       'https:
//[***CLUSTER_DOMAIN***]:[***STREAMING_SQL_ENGINE_SERVER_PORT***]/api/v1/
ssb/jobs'
```

For Secured with Knox

```
$ curl -ik --negotiate -u [***KNOX_USERNAME***[:[***KNOX_PASSWORD***] \
      'https
://[***CLUSTER_DOMAIN***]:[***STREAMING_SQL_ENGINE_SERVER_PORT***]/gateway
/cdp-proxy-api/ssb-sse-api/api/v1/ssb/jobs'
```

Note: When using the unsecured endpoints, you must provide the Username HTTP header for the command as an authentication method to indicate which user sends the request.

Using Swagger UI

You can access the Swagger UI to use the Cloudera SQL Stream Builder REST API by adding your hostname and the Streaming SQL Engine port to the Swagger UI URL.

```
http(s)://[***CLUSTER_DOMAIN***]:[***STREAMING_SQL_ENGINE_SERVER_PORT***]/
swagger-ui/index.html?configUrl=/swagger/api-docs/swagger-config
```

You can also download the swagger.json for the Cloudera SQL Stream Builder REST API using the Swagger UI.

Monitoring SQL Stream jobs

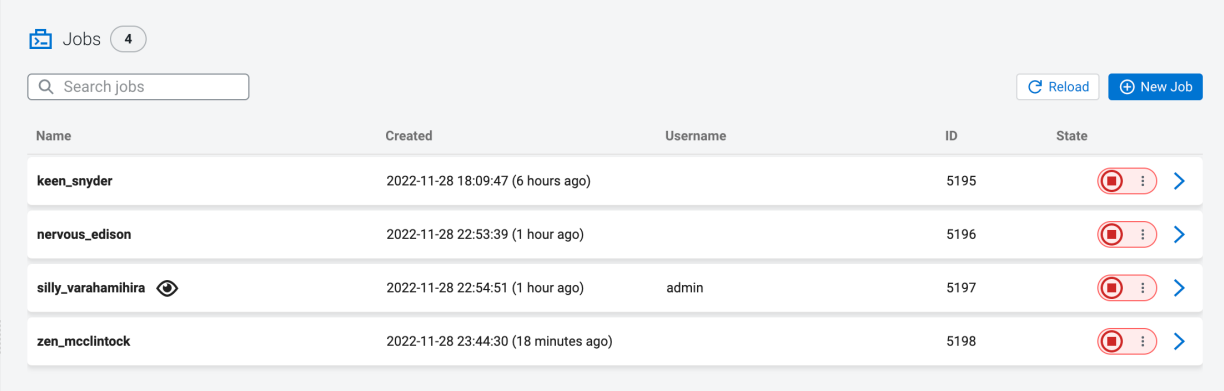
You can use the Streaming SQL Console to review the status, properties and log of your SQL Stream jobs executed in Cloudera SQL Stream Builder. Using the Flink Dashboard, you can also monitor the Flink job that is submitted when you execute a SQL query.

Using the Streaming SQL Console

When using the **Jobs** tab in Streaming SQL Console to monitor your SQL jobs, you can review the ID, the Name, the Start time, the State of the submitted jobs, and the User who submitted the SQL job.

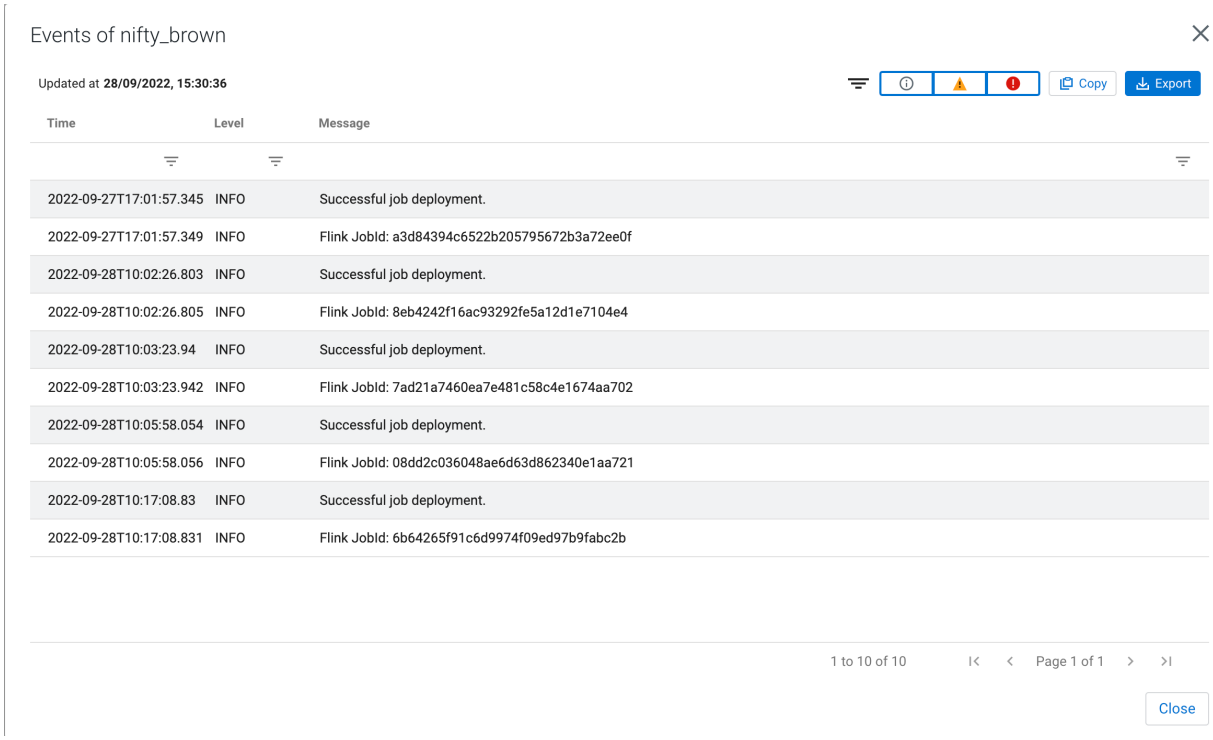
1. Click next to **Jobs**.
2. Select Manage.

   By default, all of the jobs are displayed on the **Jobs** tab.



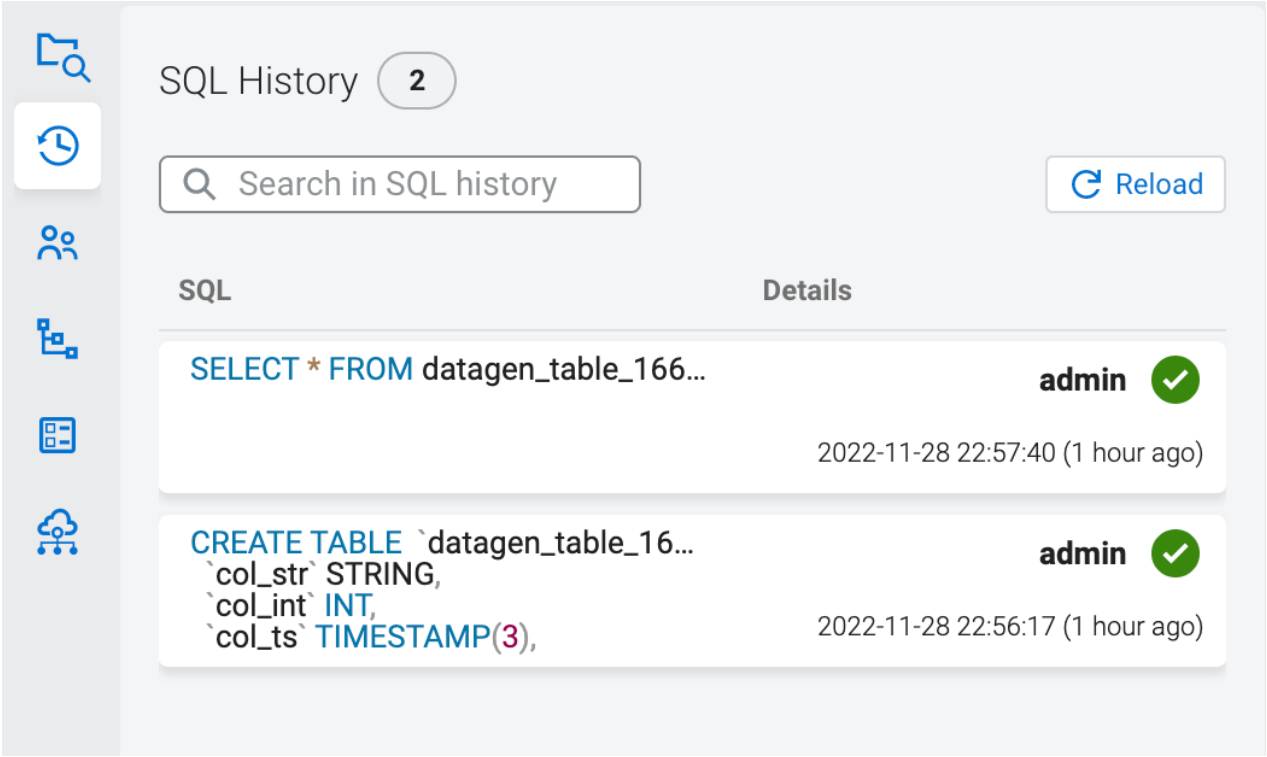
3. Click .
4. Select Events.

The **Events** window appears detailing the Flink job information that can be further filtered based on the log level settings.



History of SQL queries

You can review and reuse the SQL queries that were previosuly executed on the **SQL History** view of the Project Manager. When you click on one of the SQL queries, it is automatically imported to the SQL window for execution. You can filter the SQL queries by the time they were last run or by the user who run them.



Using the Flink Dashboard

You can also monitor your running SQL jobs using the Flink Dashboard. You can easily reach the Flink Dashboard from the main menu of Streaming SQL Console or from the SQL Editor.

Collecting diagnostic data

You can collect diagnostic data from Cloudera SQL Stream Builder and Flink using Cloudera Manager, and send it to Cloudera Support for easier troubleshooting.

When using Cloudera SQL Stream Builder and Flink, Cloudera Manager automatically collects the usage and diagnostic data from the services. The collected data is analyzed to further improve the software, and to help you in support cases when you encounter an error. You can configure the frequency of data collection, and you can also manually trigger sending the diagnostic data to Cloudera.

For more information about the diagnostic data collection and configuration, see the Cloudera Manager documentation.

The following telemetry data is collected when using Cloudera SQL Stream Builder and Flink:

- Number of users
- Number of jobs
- Number of registered data sources and connectors
- Number of Materialized Views
- Number of API keys
- Number of tables
- Number of catalogs
- Total tasks slots in use
- Log files from Streaming Engine and Materialized View Engine
- Log files from YARN to collect Flink Job and Task Manager processes

Using the OpenTelemetry Collector with Cloudera SQL Stream Builder [Technical Preview]

The OpenTelemetry metrics reporter makes it easier and more efficient to aggregate Cloudera SQL Stream Builder jobs' metrics to a central OpenTelemetry-compatible service using open standards.



Important: The OpenTelemetry Connector is in Technical Preview in Cloudera Streaming Analytics 1.17.1.



Tip: The Flink OpenTelemetry collector currently can only send data over gRPC, and the HTTP/Protobuf protocol is not supported. Because Prometheus doesn't support gRPC, as a workaround you can send data to OpenTelemetry-compatible applications that support gRPC, and/or OpenTelemetric collectors. This can be then used to receive data and forward it to other tools, such as Prometheus.

The following artifacts need to be downloaded and copied to the /opt/cloudera/parcels/FLINK/lib/flink/lib/ directory on all the cluster nodes to use this feature.

- opentelemetry-api-1.30.0.jar
- opentelemetry-context-1.30.0.jar
- opentelemetry-exporter-common-1.30.0.jar
- opentelemetry-exporter-otlp-1.30.0.jar
- opentelemetry-exporter-otlp-common-1.30.0.jar
- opentelemetry-exporter-sender-okhttp-1.30.0.jar
- opentelemetry-sdk-common-1.30.0.jar
- opentelemetry-sdk-metrics-1.30.0.jar

- opentelemetry-semconv-1.30.0-alpha.jar

Restart Cloudera SQL Stream Builder after the JAR files have been downloaded.

Add the following SET commands to the beginning of your job to start collecting metrics with OpenTelemetry:

```
set metrics.reporters=otel;
set metrics.reporter.otel.factory.class=org.apache.flink.metrics.otel.OpenTe
lemetryMetricReporterFactory;
set metrics.reporter.otel.exporter.endpoint=http://[*** FQDN OR IP ADDRESS
 ***]:[*** PORT ***];
set metrics.reporter.otel.service.name=[*** SERVICE NAME ***];
set metrics.reporter.otel.service.version=[*** SERVICE VERSION ***];
```

Note: The url prefix in the variable metrics.reporter.otel.exporter.endpoint can be http or https, depending on whether the collectors use TLS or not.

For more information on using the Flink OpenTelemetry Metrics Reporter, refer to the Apache Flink documentation.

SSB metadata collection using Atlas

As Cloudera SQL Stream Builder is built on Flink, you can use the Apache Atlas and Flink integration, and collect the metadata of your Flink jobs submitted from it. Using Atlas, you can find, organize and manage different assets of data about your Flink applications and how they relate to each other.

To use Atlas with Cloudera SQL Stream Builder, you need to create entity type definitions for Flink in Atlas, and enable the metadata collection in Cloudera Manager for Flink.

For more information, see the Flink metadata collection using Atlas .

Flink SQL

As Flink SQL is used in Cloudera SQL Stream Builder, you can execute the supported Flink DDL, DML and query statements directly from the SQL window in Streaming SQL Console.

Flink DDL

Flink SQL supports Data Definition Language (DDL) statements to create, modify and remove objects within a data structure.

The following table summarizes the supported DDL statements in SQL Stream Builder:

DDL	Description	Option
CREATE TABLE	Creating table for the SQL query. You can create tables based on the supported connectors.	• Adding WATERMARK and PRIMARY KEY information • Creating table with LIKE clause
CREATE VIEW	Creating custom views using columns from tables. There is no physical data behind a view.	• Adding queries, expressions and joins
CREATE FUNCTION	Creating User Defined Function (UDF) for a query.	• Using Python or Java language • Using Functions tab on Streaming SQL Console

DDL	Description	Option
DROP TABLE	Deleting a table, view or function.	• Using Streaming SQL Console functionality to delete table, view or function
DROP VIEW		
DROP FUNCTIONS		
ALTER TABLE	Modifying table, view or function properties.	• Using RENAME TO or SET for table • Using RENAME TO or AS for view and function
ALTER VIEW		
ALTER FUNCTION		

For more information about Flink SQL, see the Apache Flink documentation.

Managing time in Cloudera SQL Stream Builder

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, compute a new timestamp column in your CREATE TABLE statement by using a Flink SQL expression (for example, to_timestamp on a string field) or a Python User-Defined Function (UDF).

Flink DML

Flink SQL supports Data Manipulation Language (DML) statements to manipulate the data itself with adding, deleting or modifying.

The following table summarizes the supported DML statements in SQL Stream Builder:

DML	Description	Option
INSERT INTO	Inserting query results into a specified table.	• Inserting columns or values into a table • Adding OVERWRITE to overwrite the existing data in the table

For more information about Flink SQL, see the Apache Flink documentation.

Flink Queries

Flink SQL supports querying data with SELECT statements and using different types of operations.

You can query data from a table using the SELECT statement.

Query	Description	Option
SELECT	Querying data from a table using different operations.	• Selecting all data in a table • Selecting data using different types of operations

The following table summarizes the supported operations for SELECT statement in SQL Stream Builder:

Operation	Description	Option
WHERE	Querying data based on adding filters.	• Using boolean expression
JOIN	Joining data from tables based on an equivalent column.	• Using temporal joins based on event time or processing time • Using lookup join
GROUP BY	Grouping results using built-in or user defined functions.	• Using with streaming table provides updated results
ORDER BY	Ordering results to be sorted based on a specified expression.	• Using with streaming table the primary sorting key needs to be time

For more information about Flink SQL, see the Apache Flink documentation.

Other supported statements

Beside the supported DDL, DML and SELECT, the supported statements also include DESCRIBE, SHOW and SET that you can use in SQL Stream Builder.

The following table summarizes the supported statements for in SQL Stream Builder:

Operation	Description
DESCRIBE TABLES	Describing the schema of a table
SHOW TABLES	Showing a list of existing tables, views, functions, databases or catalogs
SHOW VIEWS	
SHOW FUNCTIONS	
SHOW DATABASES	
SHOW CATALOGS	
SET	Setting properties of session

For more information about Flink SQL, see the Apache Flink documentation.

Data Types

The logical type of a value to declare input and output types of operations in a table ecosystem is described by data types. Flink support a set of pre-defined data types that can be also used in Cloudera SQL Stream Builder.

The following list summarizes the pre-defined data types in Flink and SQL Stream Builder:

- CHAR
- VARCHAR
- STRING
- BOOLEAN
- BINARY
- VARBINARY
- BYTES
- DECIMAL - Supports fixed precision and scale.
- TINYINT
- SMALLINT
- INTEGER
- BIGINT
- FLOAT
- DOUBLE
- DATE
- TIME - Only supports a precision of 0.
- TIMESTAMP
- TIMPESTAMP_LTZ
- INTERVAL - Only supports interval of MONTH and SECOND(3).
- ARRAY
- MULTISET
- MAP
- ROW
- RAW
- Structured types - Only exposed in user-defined functions.

For more information about Data Types in Flink SQL, see the Apache Flink documentation.

Dynamic SQL Hints

SQL hints are supported for SQL Stream Builder (SSB) that allows you to use the dynamic table options. With the dynamic table options, you can alter any option of a table on a query level.

The dynamic table options of Flink SQL allows you to specify and override options for a table in a SQL query. The hint is formatted as a SQL comment containing an OPTIONS() clause, for example:

```
/*+ OPTIONS('key1'='value1', 'key2'='value2') */
```

You need to place the clause directly afte the table name where you need to change the options. You can use separate clauses for separate tables within a SQL query.

For example, you can use the following SQL hint to dynamically configure a Kafka consumer group at run time:

```
SELECT t1.column_a, t2.column_b
FROM tableName_A /*+ OPTIONS('properties.group.id'='my_consumer_group') */ A
S t1
JOIN tableName_B /*+ OPTIONS('properties.group.id'='my_other_consumer_group
') */ AS t2
ON t1.column_a = t2.column_b
```

Any option of a table that is accessible in the DDL of a connector can be overridden using the SQL hints. The options of a connector can be viewed on the **Connector** page of Streaming SQL Console.

For more information about the SQL Hints, see the official Apache Flink documentation.

SQL Examples

You can use the SQL examples for frequently used functions, syntax and techniques in Cloudera SQL Stream Builder. Cloudera SQL Stream Builder uses Calcite Compatible SQL, but to include the functionality of Flink you need to customize certain SQL commands.

Metadata commands

```
-- show all tables
SHOW tables;
-- describe or show schema for table
DESCRIBE payments;
DESC payments;
```

Timestamps, intervals and time

```
-- eventTimestamp is the Kafka timestamp
-- as unix timestamp. Magically added to every schema.
SELECT max(eventTimestamp) FROM solar_inputs;

-- make it human readable
SELECT CAST(max(eventTimestamp) AS varchar) as TS FROM solar_inputs;

-- date math with interval
SELECT * FROM payments
WHERE eventTimestamp > CURRENT_TIMESTAMP-interval '10' second;
```

Aggregation

```
-- hourly payment volume

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

```
-- detect multiple auths in a short window and
-- send to lock account topic/microservice

SELECT card,
MAX(amount) as theamount,
TUMBLE_END(eventTimestamp, interval '5' minute) as ts
FROM payments
WHERE lat IS NOT NULL
AND lon IS NOT NULL
GROUP BY card, TUMBLE(eventTimestamp, interval '5' minute)
HAVING COUNT(*) > 4 -- >4==fraud
```

Working with arrays

```
-- unnest each array element as separate row
SELECT b.*, u.*
 FROM bgp_avro b,
 UNNEST(b.path) AS u(pathitem)
```



Note: Arrays start at 1 not 0.

Union ALL

```
-- union two different tables
SELECT * FROM clickstream
WHERE useragent = 'Chrome/62.0.3202.84 Mobile Safari/537.36'
UNION ALL
SELECT * FROM clickstream
WHERE useragent = 'Version/4.0 Chrome/58.0.3029.83 Mobile Safari/537.36'
```

Math

```
-- simple math
SELECT 42+1 FROM mylogs;
-- inline
SELECT (amount+10)*upcharge AS total_amount
FROM payments
WHERE account_type = 'merchant'
```

```
-- convert C to F
SELECT (temp-32)/1.8 AS temp_fahrenheit
FROM reactor_core_sensors;
```

```
-- daily miles accumulator, 100:1
-- send to persistent storage microservice
-- for upsert of miles tally
SELECT card,
```

```
SUM(amount)/100 AS miles,
TUMBLE_END(eventTimestamp, interval '1' day)
FROM payments
GROUP BY card, TUMBLE(eventTimestamp, interval '1' day);
```

Joins

```
-- join multiple streams
SELECT o.name,
       sum(d.clicks),
       hop_end(r.eventTimestamp, interval '20' second, interval '40' second)
 FROM click_stream o join orgs r on o.org_id = r.org_id
                join models d on d.org_id = r.org_id
 GROUP BY o.name,
        hop(r.eventTimestamp, interval '20' second, interval '40' second)
```

```
-- join with temporal table where LatestRates is a temporal table
SELECT
  o.amount, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency
```

Hyperjoins

Joins are considered "hyperjoins" because SQL Stream Builder has the ability to join multiple tables in a single query, and because the Kafka table is created from a data provider, these joins can span multiple clusters/connect strings, but also multiple types of sources (join Kafka and a database for instance).

```
SELECT us_west.user_score+ap_south.user_score
FROM kafka_in_zone_us_west us_west
FULL OUTER JOIN kafka_in_zone_ap_south ap_south
ON us_west.user_id = ap_south.user_id;
```

Misc SQL tricks

```
-- concatenation
SELECT 'testme_'||name FROM logs;
```

```
-- select the datatype of the field
SELECT eventTimestamp, TYPEOF(eventTimestamp) as mytype FROM airplanes;
```

Escaping and quoting

Typical escaping and quoting is supported.

- Nested columns

  ```
  SELECT foo.`bar` FROM table; -- must quote nested column
  ```

- Literals

  ```
  SELECT "some string literal" FROM mytable; -- a literal
  ```

Built In Functions

```
-- convert EPOCH time to timestamp
select EPOCH_TO_TIMESTAMP(1593718981) from ev_sample_fraud;

-- convert EPOCH milliseconds to timestamp
select EPOCHMILLIS_TO_TIMESTAMP(1593718838150) from ev_sample_fraud;
```

Enriching streaming data with join

In Cloudera 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 serve 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 at an exact point of time: use the currency rates from 12:00.

For more information about 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),
    proc_time AS PROCTIME(),
    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 o.proc_time AS k
    ON o.customer_id = c.id
```

In the above example, *CUSTOMERS* serves as the lookup table. The FOR SYSTEM_TIME AS OF o.proc_time syntax indicates that you always want to look up in the latest version of the table. By 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.

In Cloudera 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 the 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 Source.
- 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	• order_number • price • order_time • item_id
Kudu - ItemMeta	• id • info

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 o.proc_time AS k
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.

Updating SQL queries with PROCTIME function

After upgrading to Cloudera Streaming Analytics 1.12.0, you need to migrate your SQL queries using PROCTIME() function in lookup joins, because the PROCTIME() function is no longer avaialable in CSA.

As the PROCTIME() shortcut function was removed from Cloudera Streaming Analytics, the SQL syntax changes for lookup joins. For example, let's say you have a streaming_table for continuously incoming data (for example, a Kafka source table), and a kudu_table to enrich that streaming data, and you want to join them on an id column. With PROC TIME(), you would use the following syntax:

```
SELECT * FROM `streaming_table` AS s
  JOIN `kudu_table` FOR SYSTEM_TIME AS OF PROCTIME() AS k
    ON s.id = k.id;
```

But as PROCTIME() is no longer available, the syntax should look like the following example:

```
SELECT * FROM `streaming_table` AS s
  JOIN `kudu_table` FOR SYSTEM_TIME AS OF s.proc_time AS k
    ON s.id = k.id;
```

The difference is that the left side table is required to have a calculated column that represents the processing time attribute. The name of the column can be anything, but it is required to use that column for the FOR SYSTEM_TIME AS OF ... expression instead of the raw PROCTIME() function.

You can update the existing SQL queries that use PROCTIME() to avoid SQL jobs to fail using one of the following methods:

- Adding the computed PROCTIME() column to the existing table:

```
ALTER TABLE `streaming_table` ADD `proc_time` AS PROCTIME();
```

- Creating a new table through copying the original DDL:

```
CREATE TABLE `streaming_table_with_proc_time` (
  -- Add proc_time column.
`proc_time` AS PROCTIME()
) WITH (
  -- No changes here.
) LIKE `streaming_table`;
```

  You can also create the new table using the LIKE clause:

```
CREATE TABLE `streaming_table_with_proc_time` (
  ... -- The existing table schema.
`proc_time` AS PROCTIME()
) WITH (
  ... -- The existing table properties.
) LIKE `streaming_table`;
```